Intro to Web Exploitation

OreSec

oresec.mines.edu/attend
What is Web Exploitation?

- Field of InfoSec concerned with hacking websites and web applications.
- List of pwned websites: [https://haveibeenpwned.com/PwnedWebsites](https://haveibeenpwned.com/PwnedWebsites)
  - Scrolled for a solid 10 seconds at full speed
- Cross-site scripting (XSS), SQL Injection, Reflected XSS, CSRF, Template Injection, Logic Flaws, etc.
Let’s get started!

- We’ll be working through a couple of these problems, and relating them to real-world hacks that have happened.

- This is what’s a called a Capture the Flag (CTF) in the security world. They are competitions that revolve around finding a “flag” by employing our hacking knowledge.

- Here’s what you’re looking for:

  competition_name{1337_5p34k_h4x0r3r5}
Hack #1: Equifax - 143,000,000 US Consumers

- Last year Equifax was hacked and lost 143 million records on US citizens.
- Information included: names, Social Security numbers, birthdates, addresses, drivers license numbers, and some credit card numbers.

Just how easy was it for the hackers to gain their initial foothold?

- Inspect Me - 75 points
Inspect Me - HTML, CSS, JS in Websites

Let’s hack Equifax!

- Nice hint from the name **Inspect** me.
- Right click on the website, and inspect element.
Inspect Me - Done!

- Equifax’s website contained login information in the HTML, and the logins were found just like this flag.
- Only 2 of 3 parts needed for the flag, the other one was empty (?)
- What did we learn? Don’t put confidential information in your website’s source code. *Never Trust the Client*
Hack #2: US Election Commission

- On December 1st 2016, Recorded Future (a cybersecurity company) discovered a Russian Hacker under the name of Rasputin selling credentials to US Election Commission databases.
- “The breach appeared to include more than 100 access credentials, including some with the highest administrative privileges.”
- Rasputin claims to have used a SQL Injection vulnerability in order to access the credentials. Most likely dumping insecure user and password hashes using some sort of SQL Injection.
SQL Injection

- Structured Query Language or SQL is the defacto way of communicating with databases using code. Very powerful, and when done poorly, very dangerous.
- Many different kinds of queries
  - `SELECT some_row FROM my_table WHERE username="fisher" AND password="5qL1";`
  - `UPDATE my_table SET username="fisher" AND password="5qL1" WHERE username="bob";`
  - `DELETE * FROM my_table WHERE username="hacker" AND password="93Th4XoR3D";`
- What does a SQL query look like in code (php for this example)?

```php
$username = $_POST["username"];  
$password = $_POST["password"];  
$query = "SELECT username, password  
            FROM users  
            WHERE (username='\${username}')  
            AND (password='\${password}');";```
SQL Injection Continued

- Why is this query vulnerable?

```
$query = "SELECT * FROM users
    WHERE (username='{$username}')
    AND (password='{$password}');"
```

- Let’s say the user inputs, as a username: `oresec') OR 1=1; --` then the query will become:

```
$query = "SELECT * FROM users
    WHERE (username='oresec') OR 1=1; --'
    AND (password='{$password}');"
```

- Which is equivalent to:

```
$query = "SELECT * FROM users
    WHERE (username='oresec') OR 1=1;
```

SQL Injection - Harder

- UNION based SQL Injection
SQL Injection in the Wild

- SQL Injection is not so easy anymore. It is “very” easy to *Never Trust the Client* and sanitize your inputs. Though there are still websites out there that are still vulnerable.

- Nowadays, there is Blind SQLi, this is SQL Injection that doesn’t give you an easy *Yes* or *No* if the injection was successful.
  - A lot of Blind SQLi is based on the timing between a successful and unsuccessful request.
Hack #3 - Samy Worm

- Fastest spreading worm ever. In ~24 hours it had infected over 1 million accounts on MySpace.
- Self-propagating javascript-based virus.
- The code would put “but most of all, Samy is my hero.” underneath the “my heros” category.
- Benign and fun, but changed website security forever.
- One of the first publicized examples of Cross-Site Scripting (XSS).
XSS - Cross-site Scripting

- Inject javascript into a web page with the intention of it being executed.
  - `<img src=ERROR onerror='alert(document.cookie)'>`

- Two types:
  - Reflected XSS:
    - Payload is delivered and executed within the same request and response.
    - Example:
      - A link has a query with javascript code. When that link is clicked, the code executes.
  - Stored XSS:
    - Payload is delivered and then stored on a server. Every request to some other resource then leads to executing this payload.
    - Example:
      - Upload a file to a web server and when users view that file, code is executed.
XSS in the Wild

- Extremely common in the wild.
- With the increase in web programmers, and the general lack of security training, XSS is everywhere.
- Can be incredibly dangerous, as the user only needs to click on a link. Something that almost everyone blindly does.
- Companies generally pay well for a Stored XSS vulnerability, and a fair amount for Reflected XSS.
Further Exploits:

- **Template Injection:**
  - Like XSS but you inject HTML templating code
    - `{{ print("This is template injection") }}`

- **Subdomain Enumeration:**
  - Not necessarily an exploit, but you’d be surprised with what companies put on a subdomain with no authentication. *cough* NSA *cough*

- **SAML Logic Errors:**
  - Security Assertion Markup Language (SAML) is used to control the login authentication flow for users on websites. There’s an entire book written on how to exploit poor configurations.

- **Path Traversal:**
  - When getting a file from a web server, one can sometimes put “..//../../” in the path to find and access other files.
Further Exploits Continued:

- **Cross-Site Request Forgery (CSRF):**
  - Malicious requests that use the victims authentication to change some sort of state that benefits the attack. For example, sending an email with the victims account number without the victim explicitly doing it.

- **Application Logic Bugs:**
  - Bugs in the implementation logic for an application. For example, discovering that the application made their “.git” file public, allowing one to recreate the file.

- **Errors:**
  - Sometimes when applications error out, they may reveal information that is either an exploit in itself, or contains the necessary data to get another exploit. Applications should never fail in a low level way and if they do, keep probing.
More Reading

- XSS:  
  - [https://excess-xss.com/](https://excess-xss.com/)

- CSRF:  

- Template Injection:  
  - [https://portswigger.net/blog/server-side-template-injection](https://portswigger.net/blog/server-side-template-injection)

- Subdomain Enumeration:  
  - [https://goo.gl/9i8HWh](https://goo.gl/9i8HWh)

- SAML Exploits:  
  - [https://research.aurainfosec.io/bypassing-saml20-SSO/](https://research.aurainfosec.io/bypassing-saml20-SSO/)

Do CTFs! Funnest way to learn and get practical experience!
Questions?

- We will now start working on some picoCTF web exploitation problems.
- Keep asking questions! I’ll do my best to answer any that come up, but I am pretty good at knowing what to google as well.
- There are writeups online for the picoCTF problems, use those if you are really stuck.

Google: Problem_Name writeups picoCTF Year