Web Application Penetration Testing Cheat Sheet ∞

CHEAT-SHEETS 18 Mar 2018  dow-j

This cheatsheet is intended to run down the typical steps performed when conducting a web application penetration test. I will break these steps down into sub-tasks and describe the tools I recommend using at each level.

Many of the ideas presented in this sheet come from the **fantastic** teachings of Tim "lanmaster53" Tomes, who has kindly allowed me to share them with you here. If you or anyone you know is interested in web application penetration testing Training I **highly** recommend that you or your company consider Tim.

Please bear in mind that these steps are **iterative** so in a typical engagement you can expect to do them multiple times. This is

All Blog Posts
Software Development
Information Security
Walkthroughs
Cheat Sheets

SOFTWARE DEVELOPMENT

Setting Up a New Mac for Software Development
More »

INFORMATION SECURITY

OWASP Top 10 2017 – Application Security Risks
More »

WALKTHROUGHS

Mr Robot 1

particularly true if you manage to traverse different levels of access in an application (e.g. elevate from a regular user to an admin).

Finally, throughout this sheet I will heavily discuss tools included in PortSwigger's Burp Suite Professional which is a paid product intended for professional use. I apologize if this dissuades you, but at the price they offer the tool for I consider it a bargain.

Table of Contents

Walkthough
OverTheWire
Bandit Wargame
Walkthrough
More »

CHEAT SHEETS

Web Application Penetration Testing Cheat Sheet
More »

Information Gathering

> *In an engagement the goal of information gathering is to gain an understanding of the application from an **outsiders perspective.***

Target Validation

TOOL	DESCRIPTION

TOOL	DESCRIPTION
Whois	A protocol for searching internet registration databases based on RFC 3912 for domain names, IPs, autonomous systems, etc.
Dig	dig (domain information groper) is a network administration command-line tool for querying Domain Name System (DNS) servers.
DNSRecon	Tool for automating many DNS enumeration techniques maintained by darkoperator

Domain Registration

Use the following steps to validate ownership of a target:

1. Whois the target domains/hosts.

```
whois example.com
```

2. Resolve the IP addresses for the target domains/hosts.

```
dig +short example.com
```

3. Whois the IP addresses.

```
whois 104.27.178.12
```

4. Analyze the results.

Results could be mixed depending on whether or not the target is using whois privacy protection.

> **!! Never attack a target that you are not positive you have permission to be testing**
>
> As a penetration tester it is your responsibility to ensure that you have permission from the owner of a target before you start testing it. This is why target validation should always be your first step when beginning an engagement.

DNS Interrogation

I like to start off with https://dnsdumpster.com/ which is a great online tool for getting a quick overview of a target domain's internet footprint via DNS.

- Forward Lookups

```
dig +nocmd example.com A +noall +answe
dig +nocmd example.com NS +noall +answ
dig +nocmd example.com MX +noall +answ
dig +nocmd example.com TXT +noall +ans
dig +nocmd example.com SOA +noall +ans
...
dig +nocmd example.com ANY +noall +ans
```

- Reverse Lookups

```
dig -x 104.27.179.12
dig -x 104.27.178.12
```

Testing for Zone Transfers

Zone transfers are a DNS transaction used to replicate records between DNS servers. It's rare to find a DNS server these days that allow a zone transfer, but it's something you should check. If you succeed in performing a zone transfer, you stand to gain access to all of the records of a domain.

★ **DNS Zone Transfer Attacks are EASY to prevent!**
At the bare-minimum an administrator can whitelist zone transfers to a select group of IP addresses.

- Example using

```
dig -t NS zonetransfer.me +short
dig -t AXFR zonetransfer.me @nsztm1.di
dig -t AXFR zonetransfer.me @nsztm2.di
```

- DNSRecon is useful for automating many of the DNS enumeration tasks above, and can often find extra information that may have been missed.

```
dnsrecon -d example.com
```

OSINT Harvesting

TOOL	DESCRIPTION
Recon-NG	Open source reconnaissance framework created by Tim 'Lanmaster53' Tomes, maintained by a community of developers on http://recon-ng.com/
Maltego	Maltego is an interactive data mining tool that renders directed graphs for link analysis.
theharvester	theHarvester is a tool for gathering e-mail accounts, subdomain names, virtual hosts,

TOOL	DESCRIPTION
	open ports/ banners, and employee names from different public sources

I thought about including a detailed section on OSINT in this cheat sheet, but at this time I've decided not to since I believe it deserves its own cheat sheet (perhaps later down the line).

Instead I'd like to point you towards some excellent resources on OSINT that I think all pentesters should familiarize themselves with.

- Michael Bazzell
  - https://inteltechniques.com
  - Open Source Intelligence Techniques
- Google Dorking
  - https://www.exploit-db.com/google-hacking-database/

Mapping

> *In an engagement the goal of mapping is to gain an understanding of the application from a typical users perspective.*

Identify Technologies

TOOL	DESCRIPTION
NMap	TCP/IP host and port scanning tool with fantastic service and OS fingerprinting capabilities.

Port Scanning, Service Fingerprinting, and OS Detection

- Port scans top 1000 TCP ports.

```
nmap 192.168.100.2
```

- Ping sweep 8 localhost addresses (actually does an ARP, ICMP, then TCP 80)

```
nmap -sP 192.168.100.0-7
```

- Port scans TCP ports 80 & 443

```
nmap -p 80,443 192.168.100.2
```

- Port scans top 1000 TCP ports, fingerprints OS and services, then runs NSE scripts

```
sudo nmap -A 192.168.100.2
```

- Port scans all 65535 TCP ports, fingerprints them, and runs NSE scripts

```
sudo nmap -A -p- 192.168.100.2
```

- Port scans top 1000 UDP ports

```
sudo nmap -sU 192.168.100.2
```

- Port scans all 65535 UDP ports.

```
sudo nmap -sU -p- 192.168.100.2
```

- Port scans all 65535 UDP ports, fingerprints them, and runs some NSE scripts.

```
sudo nmap -sU -p- -A 192.168.100.2
```

Port scanning the web server typically marks the transition from information gathering to mapping.

Be sure to make note of exposed ports, service versions, and OS/s!

Browser and Interception Proxy Setup

Firefox

TOOL	DESCRIPTION
Firefox	Modern, cross-platform, web browser that boasts a large

TOOL	DESCRIPTION
	collection of useful extensions.

`Firefox` is typically the default choice in browser when it comes to web application penetration testing. This is namely because of the amount of useful extensions available, and the fact that it doesn't affect your global proxy settings.

Firefox Extensions

TOOL	DESCRIPTION
User Agent Switcher	Firefox addon that allows for quickly changing the browser's user agent string
Wappalyzer	Browser extension that detects various software components and technologies used on websites.
FoxyProxy	Web browser proxy switcher

These are extensions that I generally use in one way or another in every engagement and I recommend installing them prior to mapping.

Configure Firefox for Burp

Before you can begin mapping the application you must first configure your browser to proxy traffic through `Burp`

- In `Firefox`
  - Configure the `FoxyProxy` extension with a proxy for Burp Suite
    - IP: `127.0.0.1`
    - Port: `8080`
  - Configure `Firefox` to trust Burp's dynamic SSL certificates
    - Open http://burp/
    - Save the certificate
    - Import the certificate into `Firefox`

Burp Configuration

TOOL	DESCRIPTION
Burp Suite Pro	Integrated platform for performing security testing of web applications.

You should customize Burp so that it suites your preferences. But, at the very least I

recommend setting the `Scan Speed` to `thorough` since you will typically be using the scanner sparingly and this way it will make more requests and check for more vulnerabilities when it is being used.

Burp Extensions

TOOL	DESCRIPTION
Burp Extender	API for expanding the functionality of Burp suite, extensions available within the BApp store.
Retire.js (BApp)	Burp suite extension used to detect vulnerabilities in outdated JavaScript components.
Wsdler (BApp)	Burp suite extension that parses WSDL files to create sample requests for all available methods
Python Scripter (BApp)	Executes a custom python script on each HTTP request and response by processed by Burp.

These are the `Burp` extensions that I typically use in most engagements. Much like the

`Firefox` extensions I like to have them installed prior to mapping the application.

These are installed using the `Burp Extender` module within Burp Suite Pro.

Manual Mapping

Manual enumeration of the web application is perhaps the most important part of the mapping process. It's imperative that you visit every page, follow every link, and perform every iteration with the application that you possibly can in order to establish a request/response baseline in Burp's sitemap.

★ **Manual mapping is critical for Single Page Apps**
Automatic spidering often fails when it comes to mapping single page applications due to the fact that most HTTP requests are sent asynchronously via JavaScript (AJAX).

Automated Mapping

Automated Mapping is done using `Burp Spider` and can be useful in finding additional pages that you normally wouldn't find during manual enumeration. Typically

Burp Spider will find more pages in legacy applications than in more-modern SPAs.

> !! **Automated Spidering is dangerous**
> Depending on the situation I will map anywhere from 80% to 95% of the target application manually and only use the spider in very specific situations. This is because the spider can potentially be destructive in certain situations.

Post-Mapping Analysis

At this point you should have the first iteration of the sitemap of the application you're testing in `Burp` and you should make note of what you've identified thus far.

Specifics you should make note of:

- Web server
- Application Architecture (Tech Stack)
- Programming Language/s
- Frameworks
- Design Patterns

This is also a good time to make note of any areas of the application where you are

expected to complete a series of actions in a certain order. Oftentimes these processes can be manipulated and executed out of order and can possibly lead to significant findings (e.g. e-commerce checkout experience, password reset form, etc.).

Discovery

> *In an engagement the goal of discovery is to gain an understanding of the application from an **attackers perspective.***

Transitioning from Mapping to Discovery

It's generally time to transition from mapping to discovery after you've established your sitemap, and conducted some initial functional analysis of the target. At this point you're going to be looking to identify as many vulnerabilities in the target application as possible. This includes, not only, The OWASP Top 10 but also flaws in application business logic. Bear in mind that you will encounter a myriad of vulnerabilities that do not fit nicely into one specific category so you should

always be vigilant.

Content Discovery

Vulnerability Scanning

NAME	DESCRIPTION
Nikto	Web server vulnerability scanner with fantastic fingerprinting capabilities.

`Nikto` is among the best when it comes to vulnerability scanners and does an excellent job at identifying vulnerabilities in the target application. It also has a `-Format` switch to easily export the scan results into a format that is easy to read and refer to later on.

Scan the target & output results into HTML format for easier readability.

```
nikto -h http://example.com -output ~/nikt
```

Vulnerability scanning with Nikto will typically mark the transition from mapping to discovery.

Once you have the results of your scan it's important to take the time to review them and visit any page/s that stand out

Forced Browsing

NAME	DESCRIPTION
Burp Engagement Tools	A set of special purpose tools built into Burp Suite Pro that apply to specific resources
Engagement Tool: Discover Content	Forced borwsing tool built into Burp Suite Pro
Burp Intruder	Burp suite tool used to automate customized attacks against web applications (e.g. brute forcing, injection, etc.)
FuzzDB	Open source database of malicious inputs, predictable resource names, greppable strings for server response messages, and other resources like web shells

Forced browsing is a discovery technique for identifying resources that are not referenced by the target application, but are reachable

nonetheless. `Discover Content` is a `Burp` tool that exists specifically for this purpose. Additionally, `Burp Intruder` can also be used for forced browsing by conducting a dictionary attack against the target (usually by injecting into url parameters or file paths). `FuzzDB` contains some excellent wordlists for this purpose, specifically take a look at the discovery section for wordlists for the purpose of forced browsing.

Testing for Alternative Content

NAME	DESCRIPTION
User Agent Switcher	Firefox addon that allows for quickly changing the browser's user agent string
Burp Intruder	Burp suite tool used to automate customized attacks against web applications (e.g. brute forcing, injection, etc.)
FuzzDB	Open source database of malicious inputs, predictable resource names, greppable strings for server response messages, and other resources like web shells

One thing I really like to do early on during the discovery process is to try different user agents on certain pages using the `User Agent Switcher` extension for `Firefox`. This extension does exactly what it sounds like and changes the `User-Agent` request header to the one you select in the extension. This is because some applications are written to respond differently to different `User-Agent` and `Referer` headers.

I'll often use `Burp Intruder` to fuzz the `User-Agent` and `Referer` headers for some of these pages as well. Typically using a wordlist from the `FuzzDB` project as well.

Automated Discovery

NAME	DESCRIPTION
Burp Scanner	Burp Suite tool used for automatically finding security vulnerabilities in web applications.

While you've been mapping the application and conducting the initial parts of discovery `Burp Passive Scanner` has been running in the background analyzing the target for

vulnerabilities. These scan results should be reviewed prior to kicking off a scan with `Burp Active Scanner` with any particularly stand-out pages made note of so that you can investigate them further at a later time.

Due to the **very** long amount of time that it takes for `Burp Active Scanner` to complete I usually prefer to run it in bursts, reviewing scan results in between and making note of the results.

> **!! Automated Scanning is dangerous**
> Using Burp Scanner may result in unexpected effects in some applications. Until you are fully familiar with its functionality and settings, you should only use Burp Scanner against non-production systems.

Configuration

Default Configurations

Testing for default configurations is a logical follow-up step after enumerating the different technologies in-use by the target during mapping. A lot of frameworks ship with vulnerable pre-configured applications in order to introduce developers to their

offerings. However, a lot of these "sample applications" are vulnerable out of the box! Better still if they exist on the same web server as the target (usually due to developer negligence, and forgotten about) they can oftentimes expose it to attack.

Testing for Misconfigurations

Ideally you should be keeping an eye out for misconfigurations in the application at all stages of testing. But, some big things you want to look out for at this point are verbose error messages since they will oftentimes give away useful information about **database structure** and the **web server file system**.

> ★ **Verbose Error Messages Are Almost Always a Finding**
> The knowledge that these error messages give to attackers can often help in exploiting successful injection or LFI (local file include) attacks.

Another thing to look out for is any sensitive form fields that do not explicitly disable autofill. A big one that I see quite a bit is password fields with a "show/hide" button. By default browsers do not save the values

entered for `input` tags with a `type="password"` attribute. However, when the “show” button is pressed it changes the `input` to `type="text"` which the browser can put in it’s autofill cache. This is a risk if the application can ever be used in a shared computing environment.

Authentication

During the discovery process you should scrutinize any login forms that you come across. If these forms are not properly secured (e.g. 2-factor, captcha, retry attempt lockout, etc.) an attacker can often gain unauthorized access to user account/s. Depending on how these forms are written or what framework / CMS is being used the target application may reveal the existence of a user account even if a login attempt fails.

If any of these are true in your test it should almost always be noted and reported as a finding. Further, if any of the login forms are not using encryption (or an outdated/broken version of SSL/TLS) that should be noted and reported as well.

Fuzzing Logins

NAME	DESCRIPTION
CeWL	Wordlist generator that creates wordlists by spidering target web sites
Burp Intruder	Burp suite tool used to automate customized attacks against web applications (e.g. brute forcing, injection, etc.)

After you've scrutinized the applications login form/s it's usually a good time to do some fuzzing. `CeWL` is a really great tool for building custom wordlists for just this purpose. You should review the help documentation first by using the `-h` switch.

For reference the syntax you will use will always be something like

```
cewl [options] www.example.com
```

Once you've crafted your custom wordlist it's time to break out `Burp Intruder` once again and do the actual fuzzing. Generally I do this using two-payload sets (one being a wordlist of usernames and the other my `CeWL` generated list for the passwords). The attack

type you should typically use in intruder for this type of fuzzing is `Cluster Bomb`.

Session Management

Session-token / Cookie analysis is not a particularly glamorous part of any engagement but it's an important one nonetheless. Typically you want to try and get an understanding of what the target application is using for session tracking and then test the session-tokens themselves using a tool like `Burp Sequencer` in order to determine how random/predictable they are. Further, some applications (generally legacy apps) will store session contents on the client-side. Occasionally this data will contain encoded, serialized objects that could potentially hold sensitive information.

This is also the time to check if the `Set-Cookie` header of the http responses from the web server contains `Secure` and `HttpOnly` flags. If not this should be noted and reported as a finding as there is never really a reason not to include these flags.

Ask Google

It only takes a minute to google search the particular session token that you have from the target. This could potentially lead to being able to predict session tokens, which opens the door to session hijacking attacks.

Testing Session Tokens With Burp

NAME	DESCRIPTION
Burp Sequencer	Burp Suite tool for analyzing randomness in a collection of data.

`Burp Sequencer` is an excellent tool that allows us to test session-tokens for randomness and predictability. While you are testing session-management you should use this tool to analyze the session token by **clearing your cookies** and then authenticating into the target application. Send the HTTP response containing the `Set-Cookie` header to `Burp Sequencer` and then have the sequencer analyze the token by starting a live capture. Typically it's okay stop the live capture after ~10,000 requests as that is generally sufficient to determine randomness and predictability in the session-

tokens.

If session tokens are not sufficiently random it opens the door to session hijacking attacks, and should be noted.

Authorization

Authorization issues like missing function level access control, and insecure direct object reference are among the most prevalent findings that I come across a majority of the time. This is because a lot of developers do not take into account the idea that a low-privilege user, or even an unauthenticated user, would try to send requests to higher-privilege endpoint (broken access control)

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

or attempt to access data belonging to a different user (insecure direct objet reference)

```
http://example.com/app/accountInfo?acct=no
```

Testing Access Control

NAME	DESCRIPTION

NAME	DESCRIPTION
Compare Site Maps	Burp Tool for testing authorization by comparing site maps

This is a technique that I like to use after I've acquired two different user accounts intended to have different access levels within the target application (usually a standard account and an admin account, but this also works for testing unauthenticated users as well).

I will map the application using the higher-privilege account, logout of the application, login as the lower-privilege user, then use Burp's `Compare Site Maps` tool in order to see what resources I can access as the lower-privilege user that should be limited to only the higher-privilege user.

★ **Make sure your target is added to the default session-handing rule's scope!**

Data Validation Testing

NAME	DESCRIPTION

NAME	DESCRIPTION
Burp Repeater	Simple tool for manually manipulating and reissuing individual HTTP requests

Injection vulnerabilities exist in web applications because they accept arbitrary user input, and do not properly validate it on the server-side. As a tester you should make note of any areas of the application that accept arbitrary user input and try to inject into them.

All applications are different so there is no one single blanket checklist you should follow when testing for injection flaws. However, I will try to break them down into sub-sections along with example injection payloads. Further, `Burp Repeater` is typically what I use the most when testing for injection flaws. It is a relatively-simple tool that lets you repeat requests to the same endpoint while giving you the opportunity to change the payload each time.

Something **very** important to keep in mind: The purpose of discovery is to simply identify the flaws, whereas with exploitation we want

to see how far they go. Obviously, the existence of any of these injection flaws is worthy of reporting, however try to save in-depth testing of each one after the discovery phase.

Please refer to the OWASP links in each sub-section for more information.

SQL Injection

If any inputs are used in queries to a database they could possibly be vulnerable to SQL injection. Combined with misconfiguration issues such as **verbose error messages** this can lead to significant amounts of data being compromised by an attacker.

I recommend reading this wiki by Netspi when testing for SQL injection as it's a great resource. At it's simplest though, if you get a database error back from any of these payloads odds are that the target is vulnerable to SQL injection.

`Sqlmap` is a tool that allows for testing for sql injection in an automated way, and I will dig into its use further in the **exploitation** section.

OWASP - Testing for SQL Injection

Examples

```
' OR 1=1 -- 1
' OR '1'='1
' or 1=1 LIMIT 1;--
admin';--

http://www.example.com/product.php?id=10 A
```

Cross-site Scripting (XSS)

Cross-site Scripting (XSS) attacks occur when an attacker uses a web application to send malicious code, generally in the form of JavaScript, to a different end user.

There are 3 different types of XSS that you should keep an eye out for:

1. **Stored** - A Stored XSS vulnerability exists when data provided to a web application by a user is first stored persistently on the server.
2. **Reflected** - A Reflected XSS vulnerability exists when data provided by a web client is used immediately by server-side scripts to generate a page of results for that user.
3. **DOM-Based** - A DOM-based XSS vulnerability exists within a page's client-

side script itself.

OWASP - Testing for Cross-site Scripting

Examples

```
<IMG SRC=javascript:alert('XSS')>
"><script>alert('XSS')</script><"
" onmouseover="alert('XSS')

http://server/cgi-bin/testcgi.exe?<SCRIPT>
%3cscript src=http://www.example.com/malic
```

XML Injection

XML Injection testing is when a tester tries to inject an XML doc to the application. If the XML parser fails to contextually validate data, then the test will yield a positive result.

OWASP - Testing for XML Injection

Examples

```
Username = foo<
Username = foo<!--
```

XML External Entities (XXE)

If the definition of an entity is a URI, the entity is called an external entity. Unless configured

to do otherwise, external entities force the XML parser to access the resource specified by the URI, e.g., a file on the local machine or on a remote systems.

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

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

<?xml version="1.0" encoding="ISO-8859-1"
<!DOCTYPE foo [
  <!ELEMENT foo ANY >
  <!ENTITY xxe SYSTEM "file:///c:/boot.in

<?xml version="1.0" encoding="ISO-8859-1"
<!DOCTYPE foo [
  <!ELEMENT foo ANY >
  <!ENTITY xxe SYSTEM "http://www.attacke
```

Template Injection

Template injection allows an attacker to include template code into an existant (or not) template.

Portswigger - Server Side Template Injecton

Examples

```
<%= 7 * 7 %>
{{ 7 * 7 }}
```

OS Command Injection

OS Command Injection is a technique where a user injects OS commands into web application interface with the intention of those commands executing on the web server.

OWASP - Testing for Command Injection

Examples

```
http://sensitive/cgi-bin/userData.pl?doc=/
http://sensitive/something.php?dir=%3Bcat%


Doc=Doc1.pdf+|+Dir c:\
```

Open Redirection

Open redirection is an input validation flaw that exists when an application accepts an user controlled input which specifies a link that leads to an external URL that could be malicious.

OWASP - Testing for Client Side URL Redirect

Examples

```
http://www.target.site?#redirect=www.fake-
http://www.target.site??url=http://www.fak
```

Local File Inclusion (LFI)

Local File Inclusion (also known as LFI) is the process of including files, that are already locally present on the server, through the exploiting of vulnerable inclusion procedures implemented in the application.

OWASP - Testing for Local File Inclusion

Examples

```
http://vulnerable_host/preview.php?file=..
http://vulnerable_host/preview.php?file=..
```

Remote File Inclusion (RFI)

Remote File Inclusion (also known as RFI) is the process of including remote files through the exploiting of vulnerable inclusion procedures implemented in the application.

OWASP - Testing for Remote File Inclusion

Examples

```
http://vulnerable_host/vuln_page.php?file=
```

Business Logic

Discovering flaws in business logic requires you, as the attacker, to have a decent fundemental understanding of the target application. Once you know the way the application is intended to be used you can start to reason how it could be exploited. When testing for business logic flaws refer back to areas of the application where you as the user are expected to complete a series of actions in a certain order (e.g. password reset form, order form, etc.) and try executing those actions out of order.

Further, you should check if a user is able to enter unrealistic values into certain inputs fields within the application (e.g. a fitness app where a user is expected to enter the amount of miles run)?

This is also a good time to test for unrestricted file upload.

Encryption Flaws

NAME	DESCRIPTION
SSLyze	TLS/SSL Implementation Analyzer

When determining the quality of an applications SSL/TLS implementation I always recommend starting out with the SSL Labs' Server Test, and `SSLyze` if that can't be done.

SSLyze Example

```
sslyze --regular www.example.com
```

Generally, this boils down to:

1. Whether or not the application is using some form of encryption.
2. Is that encryption protocol insecure or vulnerable (TLS 1.2, SSL2/SSl3)?

You should also use this time to investigate if the application is using weak encryption algorithms (e.g. MD5, RC4, etc.) and supports forward secrecy.

Denial-of-Service

In a nutshell Denial-of-Service (DoS) is an

attack wherein the prepetrator aims to make the target application unavilable to its users in one way or another. Denial-of-Service attack vectors can range from user file uploads (if they are not logically restricted by size) to a user account lockout mechanism in place to prevent brute force login attempts.

If there are any pages that take a long time to load or ajax requests that take a long time to come back, these could be indicators of poorly written SQL queries that could be leveraged to perform a DoS attack.

Flash Applications

NAME	DESCRIPTION
Firefox Developer Tools	Developer tools built into Firefox for examining, editing, and debugging client-side code
JPEXS (FFDec)	Open source SWF decompiler and editor

If the application that you're testing makes use of flash or other compiled client-side technologies (e.g. silverlight) it's worth it to download them to your filesystem and use a

tool like `JPEXS FFDec` to decompile them and examine the source code. If you're able to successfully reverse engineer the application you may uncover some flaws in the underlying code.

Testing Web Services

Web services are technologies used for machine to machine communication, but they should be tested using the same methodology that you've been employing prior to this (mapping -> discovery -> exploitation), using `Burp` to intercept requests and analyze responses from the API endpoint/s.

Testing REST Web Services

Ideally the first thing you should do prior to testing a REST web service is read the documentation if it is available. Typically, this will be the case when conducting a white-box or grey-box penetration testing and is generally perfered since it will allow for a more comprehensive test.

In a black-box test we can use `Burp` to intercept the request/response pairs and look

at any information returned in a `JSON` format to try and gain and understanding of the API, but this is very tedious and not recommended if possible.

Regardless, since REST uses the `http` protocol vs something like SOAP services we can test the API endpoints for the same vulnerabilities that we have previously in the **discovery** phase such as SQL injection, and XSS.

Please refer to these articles for additional resources when testing REST APIs.

- https://support.portswigger.net/customer/portal/articles/1965674-using-burp-to-test-for-cross-site-request-forgery-csrf-
- http://blog.isecurion.com/2017/10/10/penetration-testing-restful-web-services/
- https://www.owasp.org/index.php/REST_Assessment_Cheat_Sheet

Testing SOAP Web Services

NAME	DESCRIPTION
WSDLER	Parses WSDL files to

NAME	DESCRIPTION
(BApp)	create sample requests for all available methods

Although these days I certainly see more REST than I do SOAP when testing application web services it is still there and still something you should keep an eye out for.

Something very nice about SOAP-based web services is that they are self documenting via WSDL (web service definition language) files. You can use a tool such as `Wsdler (BApp)` to parse these files and send sample requests via `Burp Repeater` to the endpoint.

Same as REST, these services should be tested for the same vulnerabilities as the other areas of the application (e.g. sql injection, xss, etc.)

- Check out any service-related paths found during mapping/discovery
  - e.g. http://exampleapplication.com/service
- View the WSDL file for the service to load it into Burp
- Go to the Burp Proxy history tab and add

the WSDL file to the Wsdler extension via the “Parse WSDL” context menu option.

- Send the sample request to Repeater and determine how the service works

Please refer to these articles for additional resources when testing SOAP APIs.

https://blog.netspi.com/hacking-web-services-with-burp/

Exploitation

> ***In an engagement the goal of exploitation is to leverage the vulnerabilities found during discovery and measure how deep they go and the true risk that they pose.***

In a nutshell what you want to do at this point is refer to the notes that you’ve taken during **information gathering**, **mapping**, and **discovery** and dig as deep as possible into the vulnerabilities that you’ve discovered. Occasionally during the exploitation process you may traverse different (higher) levels of privilege within the target application. If this happens you want to go back and iterate over the methodology again starting at **mapping**.

What follows are some example **exploitation** scenarios, but this step is unique to every engagement.

★ **Remember: you are taking your discovery findings to the next level!**
which requires a lot of independent research that I cannot possibly fit into a single cheat sheet.

Exploitation Scenarios

Exploiting Cross-site Scripting

Browser Hyjacking

NAME	DESCRIPTION
BeEF	A web-based interface for command and control of XSS-ed zombie browsers

If you've determined that the target application is indeed vulnerable to XSS during **discovery** a good step to take during exploitation is to see if you can use a tool like `BeEF` to "hook" zombie browsers via XSS.

Here is a good article on doing just that

Generally this is something you would use your own browser for in an isolated environment as a proof of concept for just how dangerous XSS is when you eventually present your findings to the client.

Exploiting SQL injection

Data Extraction

NAME	DESCRIPTION
SQLMap	An automated SQL injection tool that both detects and exploits SQL injection flaws on many popular RDBMSs

If the target application is vulnerable to SQL injection `SQLMap` will usually be the go-to tool in order to extract data during exploitation.

There is a great tutorial on the SQLMap site regarding the specific switches and how they work. that I recommend you refer to when first starting out with the tool.

Offline Password Cracking

NAME	DESCRIPTION
Hashcat	World's fastest and most

NAME	DESCRIPTION
	advanced password recovery utility

This one is a bit of a strech but something you should try if you get a dump of user account passwords from the target application.

If the passwords are stored using a 1-way hashing algorithm you can likely use `hashcat` along with a good wordlist such as `rockyou.txt` to crack them as detailed in this article.

Needless to say something like this would be among the biggest findings you could bring to a client at the end of a penetration test.

Authentication Bypass

Something worth trying at this point (if you haven't already) is to attempt to elevate your privilege within the application via SQL injection if you can. There are a good number of articles online about this, but here are a few sample payloads you can try entering on a vulnerable login form (leave password field blank of course)

```
admin' --
admin' #
admin'/*
admin' or '1'='1
admin' or '1'='1'--
admin' or '1'='1'#
admin' or '1'='1'/*
admin'or 1=1 or ''='
admin' or 1=1
```

Cross-site Request Forgery (CSRF)

NAME	DESCRIPTION
Burp: Generate CSRF PoC	Burp tool used to generate a CSRF proof of concept against the target application

If it comes to light that your target is vulnerable to CSRF (usually one of the first things `Burp Scanner` will pick up) you can use the Burp tool `Generate CSRF PoC` to validate if the target application is indeed vulnerable.

Here is a tutoral from portswigger on how to use the tool.

However, it's generally just a matter of:

1. Intercepting an HTTP request (ideally one where your testing account is changing some account information)
2. Right-clicking that request in Burp
3. Generating the CSRF PoC (and modifying some details of the request body)
4. Saving the CSRF PoC to an `html` file
5. Opening the new `html` file and clicking the `submit request` button
6. Validating that the attacker-intended change happened.

Closing

Thanks for reading! The goal of this sheet is to be a resource to anyone conducting a web application penetration test. The intent being to highlight **methodology** over all else.

With that in mind this is a living document and will be updated regularly over time as I receive feedback on it.

Enjoy the content? Share

it!

Twitter Google+

LinkedIn Reddit

Hacker News