Introduction to WebHacking

Get hands-on, learn about and exploit some of the most popular web application vulnerabilities seen in the industry today.

- Walking an Application
- Content Discovery
- Subdomain Enumeration
- Authentication Bypass
- IDOR
- File Inclusion
- Intro to SSRF
- Intro to Cross-site Scripting
- Race Conditions
- Command Injection
- SQL Injection

Walking an Application

Introduction

In this room you will learn how to manually review a web application for security issues using only the in-built tools in your browser. More often than not, automated security tools and scripts will miss many potential vulnerabilities and useful information.

Browser Tools Overview

Here is a short breakdown of the in-built browser tools you will use throughout this room:

- **View Source** - Use your browser to view the human-readable source code of a website.
- **Inspector** - Learn how to inspect page elements and make changes to view usually blocked content.
- **Debugger** - Inspect and control the flow of a page's JavaScript
- **Network** - See all the network requests a page makes.

The Penetration Testing Approach

As a penetration tester, your role when reviewing a website or web application is to discover features that could potentially be vulnerable and attempt to exploit them to assess whether or not they are. These features are usually parts of the website that require some interactivity with the user.

Finding interactive portions of the website can be as easy as spotting a login form to manually reviewing the website's JavaScript. An excellent place to start is just with your browser exploring the website and noting down the individual pages/areas/features with a summary for each one.

Example Site Review

An example site review for the Acme IT Support website would look something like this:

Feature	URL	Summary
Home Page	`/`	This page contains a summary of what Acme IT Support does with a company photo of their staff.
Latest News	`/news`	This page contains a list of recently published news articles by the company, and each news article has a link with an id number, i.e. `/news/article?id=1`
News Article	`/news/article?id=1`	Displays the individual news article. Some articles seem to be blocked and reserved for premium customers only.
Contact Page	`/contact`	This page contains a form for customers to contact the company. It contains name, email and message input fields and a send button.
Customers	`/customers`	This link redirects to `/customers/login`.
Customer Login	`/customers/login`	This page contains a login form with username and password fields.
Customer Signup	`/customers/signup`	This page contains a user-signup form that consists of a username, email, password and password confirmation input fields.
Customer Reset Password	`/customers/reset`	Password reset form with an email address input field.
Customer Dashboard	`/customers`	This page contains a list of the user's tickets submitted to the IT support company and a "Create Ticket" button.
Create Ticket	`/customers/ticket/new`	This page contains a form with a textbox for entering the IT issue and a file upload option to create an IT support ticket.
Customer Account	`/customers/account`	This page allows the user to edit their username, email and password.
Customer Logout	`/customers/logout`	This link logs the user out of the customer area.

Viewing Page Source

The page source is the human-readable code returned to our browser/client from the web server each time we make a request.

The returned code is made up of HTML (HyperText Markup Language), CSS (Cascading Style Sheets) and JavaScript, and it's what tells our browser what content to display, how to show it and adds an element of interactivity with JavaScript.

For our purposes, viewing the page source can help us discover more information about the web application.

How to View Page Source

There are three main ways to view page source:

1. **Right-click method**: While viewing a website, you can right-click on the page, and you'll see an option on the menu that says "View Page Source".
2. **URL prefix method**: Most browsers support putting `view-source:` in front of the URL, for example: `view-source:https://www.google.com/`
3. **Browser menu**: In your browser menu, you'll find an option to view the page source. This option can sometimes be in submenus such as developer tools or more tools.

What to Look For in Page Source

Comments

At the top of the page, you'll notice some code starting with `<!--` and ending with `-->` — these are comments. Comments are messages left by the website developer, usually to explain something in the code to other programmers or even notes/reminders for themselves. These comments don't get displayed on the actual webpage.

Links

Links to different pages in HTML are written in anchor tags (these are HTML elements that start with `<a`), and the link that you'll be directed to is stored in the `href` attribute.

If you view further down the page source, there may be hidden links to private areas used by the business for storing company/staff/customer information.

External Files and Directory Listings

External files such as CSS, JavaScript and Images can be included using the HTML code. Sometimes these files are all stored in the same directory. If you view this directory in your web browser, there may be a configuration error.

What should be displayed is either a blank page or a 403 Forbidden page with an error stating you don't have access to the directory. Instead, if the directory listing feature has been enabled, it will list every file in the directory. Sometimes this isn't an issue, and all the files in the directory are safe to be viewed by the public, but in some instances, backup files, source code or other confidential information could be stored here.

Framework Detection

Many websites these days aren't made from scratch and use what's called a framework. A framework is a collection of premade code that easily allows a developer to include common features that a website would require, such as blogs, user management, form processing, and much more, saving the developers hours or days of development.

Viewing the page source can often give us clues into whether a framework is in use and, if so, which framework and even what version. Knowing the framework and version can be a powerful find as there may be public vulnerabilities in the framework, and the website might not be using the most up to date version.

Developer Tools

Every modern browser includes developer tools — a toolkit used to aid web developers in debugging web applications and gives you a peek under the hood of a website to see what is going on. As a pentester, we can leverage these tools to provide us with a much better understanding of the web application.

We're specifically focusing on three features of the developer toolkit: **Inspector**, **Debugger**, and **Network**.

Opening Developer Tools

The way to access developer tools is different for every browser. If you're not sure how to access it, consult your browser's documentation for specific instructions.

Inspector

The page source doesn't always represent what's shown on a webpage; this is because CSS, JavaScript and user interaction can change the content and style of the page, which means we need a way to view what's been displayed in the browser window at this exact time. Element inspector assists us with this by providing us with a live representation of what is currently on the website.

As well as viewing this live view, we can also edit and interact with the page elements, which is helpful for web developers to debug issues.

Bypassing Paywalls

Floating boxes blocking page contents are often referred to as paywalls as they put up a metaphorical wall in front of the content you wish to see until you pay.

How to bypass using Inspector:

1. Right-click on the premium notice (paywall)
2. Select the "Inspect" option from the menu
3. This opens the developer tools with the elements/HTML that make up the website
4. Locate the `DIV` element with the class `premium-customer-blocker`
5. Find the CSS style `display: block`
6. Click on the word `block` and type `none`
7. The box will disappear, revealing the content underneath

Remember: This is only edited on your browser window, and when you press refresh, everything will be back to normal.

Debugger

This panel in the developer tools is intended for debugging JavaScript, and is an excellent feature for web developers wanting to work out why something might not be working. As penetration testers, it gives us the option of digging deep into the JavaScript code.

> **Note**: In Firefox and Safari, this feature is called "Debugger", but in Google Chrome, it's called "Sources".

Understanding Minified and Obfuscated Code

Many times when viewing JavaScript files, you'll notice that everything is on one line. This is because it has been **minimised**, which means all formatting (tabs, spacing and newlines) have been removed to make the file smaller. Files may also be **obfuscated**, which makes them purposely difficult to read, so they can't be copied as easily by other developers.

Using the Pretty Print Feature

We can return some of the formatting by using the "Pretty Print" option, which looks like two braces `{ }` to make it a little more readable.

Using Breakpoints

Breakpoints are points in the code that we can force the browser to stop processing the JavaScript and pause the current execution.

How to set a breakpoint:

1. Navigate to the JavaScript file you want to debug
2. Click the line number where you want to pause execution
3. The line will turn blue, indicating a breakpoint is set
4. Refresh the page
5. The code execution will pause at that line, allowing you to inspect the current state

Network Tab

The network tab on the developer tools can be used to keep track of every external request a webpage makes. If you click on the Network tab and then refresh the page, you'll see all the files the page is requesting.

Monitoring AJAX Requests

With the network tab open, when you submit forms, you'll notice events in the network tab. Forms are often submitted in the background using a method called **AJAX**. AJAX is a method for sending and receiving network data in a web application background without interfering by changing the current web page.

To monitor form submissions:

1. Open the Network tab
2. Fill in a form on the website
3. Press the submit button
4. Examine the new entry in the network tab
5. View the page the data was sent to and the response received

This can reveal:

- API endpoints
- Data being transmitted
- Server responses
- Hidden parameters or flags

Summary

Manual web application security testing using browser tools is an essential skill for penetration testers. By leveraging View Source, Inspector, Debugger, and Network tools, you can:

- Discover hidden information in comments and source code
- Identify framework versions and potential vulnerabilities
- Bypass client-side restrictions
- Debug and analyze JavaScript code
- Monitor network traffic and AJAX requests
- Find exposed directories and sensitive files

These techniques complement automated tools and often reveal vulnerabilities that automated scanners miss.

Content Discovery

What is Content Discovery?

In the context of web application security, content can be many things: a file, video, picture, backup, or website feature. When we talk about content discovery, we're not talking about the obvious things we can see on a website; it's the things that aren't immediately presented to us and that weren't always intended for public access.

This content could be, for example:

- Pages or portals intended for staff usage
- Older versions of the website
- Backup files
- Configuration files
- Administration panels

There are **three main ways** of discovering content on a website:

1. **Manually**
2. **Automated**
3. **OSINT** (Open-Source Intelligence)

Manual Discovery Methods

There are multiple places we can manually check on a website to start discovering more content.

Robots.txt

The `robots.txt` file is a document that tells search engines which pages they are and aren't allowed to show on their search engine results or ban specific search engines from crawling the website altogether. It can be common practice to restrict certain website areas so they aren't

displayed in search engine results. These pages may be areas such as administration portals or files meant for the website's customers. This file gives us a great list of locations on the website that the owners don't want us to discover as penetration testers.

Practical Exercise:

```
http://MACHINE_IP/robots.txt
```

Favicon

The favicon is a small icon displayed in the browser's address bar or tab used for branding a website. Sometimes when frameworks are used to build a website, a favicon that is part of the installation gets leftover, and if the website developer doesn't replace this with a custom one, this can give us a clue on what framework is in use.

OWASP hosts a database of common framework icons that you can use to check against the target's favicon:

- https://wiki.owasp.org/index.php/OWASP_favicon_database

Practical Exercise:

Visit: https://static-labs.tryhackme.cloud/sites/favicon/

To download and hash the favicon:

Using curl (Linux/Mac):

```
curl https://static-labs.tryhackme.cloud/sites/favicon/images/favicon.ico | md5sum
```

Using PowerShell (Windows):

```
PS C:\> curl https://static-labs.tryhackme.cloud/sites/favicon/images/favicon.ico -
UseBasicParsing -o favicon.ico
PS C:\> Get-FileHash .\favicon.ico -Algorithm MD5
```

> **Note:** If your hash ends with 427e then your curl failed, and you may need to try it again.

Sitemap.xml

Unlike the `robots.txt` file, which restricts what search engine crawlers can look at, the `sitemap.xml` file gives a list of every file the website owner wishes to be listed on a search engine. These can sometimes contain areas of the website that are a bit more difficult to navigate to or even list some old webpages that the current site no longer uses but are still working behind the scenes.

Example:

```
http://10.67.149.33/sitemap.xml
```

HTTP Headers

When we make requests to the web server, the server returns various HTTP headers. These headers can sometimes contain useful information such as the webserver software and possibly the programming/scripting language in use.

Example using curl:

```
curl http://10.67.149.33 -v
```

Sample output:

```
*   Trying 10.67.149.33:80...
* TCP_NODELAY set
* Connected to 10.67.149.33 (10.67.149.33) port 80 (#0)
> GET / HTTP/1.1
> Host: 10.67.149.33
> User-Agent: curl/7.68.0
> Accept: */*
>
* Mark bundle as not supporting multiuse
< HTTP/1.1 200 OK
```

```
< Server: nginx/1.18.0 (Ubuntu)
< X-Powered-By: PHP/7.4.3
< Date: Mon, 19 Jul 2021 14:39:09 GMT
< Content-Type: text/html; charset=UTF-8
< Transfer-Encoding: chunked
< Connection: keep-alive
```

In this example, we can see the webserver is NGINX version 1.18.0 and runs PHP version 7.4.3. Using this information, we could find vulnerable versions of software being used.

Framework Stack

Once you've established the framework of a website, either from the favicon example above or by looking for clues in the page source such as comments, copyright notices or credits, you can then locate the framework's website. From there, we can learn more about the software and other information, possibly leading to more content we can discover.

Looking at the page source, you might find comments linking to framework documentation, which can reveal paths to administration portals or other sensitive areas.

OSINT (Open-Source Intelligence) Methods

There are external resources available that can help in discovering information about your target website. These resources are freely available tools that collect information.

Google Hacking / Dorking

Google hacking/Dorking utilizes Google's advanced search engine features, which allow you to pick out custom content.

Common Google Dork Filters:

Filter	Example	Description

site	site:tryhackme.com	Returns results only from the specified website address
inurl	inurl:admin	Returns results that have the specified word in the URL
filetype	filetype:pdf	Returns results which are a particular file extension
intitle	intitle:admin	Returns results that contain the specified word in the title

Example usage:

```
site:tryhackme.com admin
```

This would only return results from the tryhackme.com website which contain the word "admin" in its content.

More information: https://en.wikipedia.org/wiki/Google_hacking

Wappalyzer

Wappalyzer (https://www.wappalyzer.com/) is an online tool and browser extension that helps identify what technologies a website uses, such as:

- Frameworks
- Content Management Systems (CMS)
- Payment processors
- Version numbers

Wayback Machine

The Wayback Machine (https://archive.org/web/) is a historical archive of websites that dates back to the late 90s. You can search a domain name, and it will show you all the times the service scraped the web page and saved the contents. This service can help uncover old pages that may still be active on the current website.

GitHub

Git is a **version control system** that tracks changes to files in a project. GitHub is a hosted version of Git on the internet. Repositories can either be set to public or private and have various

access controls.

You can use GitHub's search feature to look for:

- Company names
- Website names
- Source code
- Passwords or credentials
- Configuration files

Once discovered, you may have access to content that you hadn't yet found.

S3 Buckets

S3 Buckets are a storage service provided by Amazon AWS, allowing people to save files and even static website content in the cloud accessible over HTTP and HTTPS. Sometimes access permissions are incorrectly set and inadvertently allow access to files that shouldn't be available to the public.

S3 Bucket Format:

```
http(s)://{name}.s3.amazonaws.com
```

Common naming patterns:

- {name}-assets
- {name}-www
- {name}-public
- {name}-private

S3 buckets can be discovered by:

- Finding URLs in the website's page source
- Searching GitHub repositories
- Automating the process with common naming patterns

Automated Discovery

What is Automated Discovery?

Automated discovery is the process of using tools to discover content rather than doing it manually. This process is automated as it usually contains hundreds, thousands or even millions of requests to a web server. These requests check whether a file or directory exists on a website, giving us access to resources we didn't previously know existed.

What are Wordlists?

Wordlists are text files that contain a long list of commonly used words. For content discovery, we require lists containing the most commonly used directory and file names.

Recommended wordlist resource:

- SecLists by Daniel Miessler: https://github.com/danielmiessler/SecLists

Automation Tools

Three popular content discovery tools:

1. ffuf

```
ffuf -w /usr/share/wordlists/SecLists/Discovery/Web-Content/common.txt -u
http://10.67.149.33/FUZZ
```

2. dirb

```
dirb http://10.67.149.33/ /usr/share/wordlists/SecLists/Discovery/Web-Content/common.txt
```

3. Gobuster

```
gobuster dir --url http://10.67.149.33/ -w /usr/share/wordlists/SecLists/Discovery/Web-
Content/common.txt
```

Summary

Content discovery combines multiple techniques:

- **Manual methods** help understand the structure and technology stack
- **OSINT** provides external information and historical data
- **Automated tools** efficiently scan for hidden directories and files

Each method has its strengths, and using them in combination provides the most comprehensive results for web application security testing.

Subdomain Enumeration

Overview

Room URL: https://tryhackme.com/room/subdomainenumeration

Difficulty: Easy

Category: Reconnaissance/Subdomain Enumeration

Date Completed: 01/21/2026

Objective

Learn and practice three different subdomain enumeration methods to expand the attack surface and discover potential points of vulnerability: Certificate Transparency logs, DNS Brute Force, OSINT tools, and Virtual Host enumeration.

Table of Contents

Introduction

What is Subdomain Enumeration?

Subdomain enumeration is the process of finding valid subdomains for a domain. The purpose is to expand our attack surface to discover more potential points of vulnerability that may not be immediately visible from the main domain.

Why Subdomain Enumeration Matters

Organizations often have multiple subdomains for different purposes:

- Development and staging environments
- Administrative portals
- API endpoints
- Regional or departmental sites
- Legacy applications

These subdomains may have different security configurations, and some may be less secured than the main domain, making them valuable targets for security testing.

Method 1: Certificate Transparency (CT) Logs

What are CT Logs?

When an SSL/TLS (Secure Sockets Layer/Transport Layer Security) certificate is created for a domain by a CA (Certificate Authority), CAs participate in what's called "Certificate Transparency logs". These are publicly accessible logs of every SSL/TLS certificate created for a domain name.

Purpose:

The purpose of Certificate Transparency logs is to stop malicious and accidentally made certificates from being used. We can use this service to discover subdomains belonging to a domain.

Tool:

Sites like https://crt.sh offer a searchable database of certificates that shows current and historical results.

Method 2: DNS Brute Force Enumeration

What is it?

Brute force DNS enumeration is the method of trying tens, hundreds, thousands, or even millions of different possible subdomains from a pre-defined list of commonly used subdomains.

Why Automate?

Because this method requires many requests, we automate it with tools to make the process quicker.

Common Tools:

- `dnsrecon` - DNS reconnaissance tool
- `dnsenum` - DNS enumeration tool
- `fierce` - DNS scanner

How it Works:

The tool takes a wordlist of common subdomain names (e.g., www, mail, ftp, admin, api, dev) and tests each one to see if it resolves to an IP address.

Method 3: OSINT - Automated Discovery

What is OSINT?

Open-Source Intelligence (OSINT) refers to using publicly available information sources to gather intelligence about a target.

Tools:

Tools like Sublist3r automate the OSINT subdomain discovery process by:

- Searching multiple search engines (Google, Bing, Yahoo)
- Querying DNS databases
- Using certificate transparency logs
- Checking web archives
- Searching threat intelligence platforms

This approach speeds up discovery by aggregating results from multiple sources automatically.

Method 4: Virtual Host Enumeration

The Concept:

Some subdomains aren't always hosted in publicly accessible DNS results. These might include:

- Development versions of web applications
- Administration portals
- Internal testing environments

Where These Records Live:

- Private DNS servers
- Local `/etc/hosts` file (Linux/Mac)
- `c:\windows\system32\drivers\etc\hosts` file (Windows)

How Web Servers Handle Multiple Sites:

Web servers can host multiple websites from one server. When a website is requested from a client, the server knows which website the client wants from the **Host header**.

The Attack Method:

We can utilize this Host header by making changes to it and monitoring the response to see if we've discovered a new website. Like with DNS brute force, we automate this process using a wordlist of commonly used subdomains.

Tool: ffuf (Fuzz Faster U Fool)

`ffuf` is a fast web fuzzer that can be used for virtual host discovery by fuzzing the Host header.

Basic Syntax:

```
ffuf -w [wordlist] -H "Host: FUZZ.[domain]" -u http://[IP]
```

Key Flags:

- `-w`: Specifies the wordlist to use
- `-H`: Adds/edits a header (in this case, the Host header)
- `FUZZ`: Keyword that will be replaced with each word from the wordlist
- `-u`: Target URL
- `-fs`: Filter by size - tells ffuf to ignore results of a specified size

Why Filtering is Important:

The command will always produce a result (even for non-existent subdomains) because the web server responds. We need to filter out false positives by excluding the most common response size.

Walk Through

Task 1: Certificate Transparency Logs

1. Navigate to https://crt.sh
2. Search for `tryhackme.com`
3. Look through the results for entries logged on `2020-12-26`
4. Identify the subdomain

Answer: `store.tryhackme.com`

Task 2: DNS Brute Force with dnsrecon

1. Click the "View Site" button on the TryHackMe page
2. Press the "Run DNSrecon Request" button to start the simulation
3. Observe the output from the dnsrecon tool
4. Identify the first subdomain found

Answer: `api.acmeitsupport.thm`

Task 3: OSINT with Sublist3r

1. Click the "View Site" button on the TryHackMe page
2. Run the sublist3r simulation
3. Review the discovered subdomains
4. Identify the first subdomain found

Answer: `web55.acmeitsupport.thm`

Task 4: Virtual Host Enumeration with ffuf

Step 1: Initial Scan (Unfiltered)

```
ffuf -w /usr/share/wordlists/SecLists/Discovery/DNS/namelist.txt -H "Host:
FUZZ.acmeitsupport.thm" -u http://MACHINE_IP
```

- This command will return many results
- Note the most common response size value

Step 2: Filtered Scan

```
ffuf -w /usr/share/wordlists/SecLists/Discovery/DNS/namelist.txt -H "Host:
FUZZ.acmeitsupport.thm" -u http://MACHINE_IP -fs {size}
```

- Replace `{size}` with the most common size from the previous results
- This filters out false positives
- The filtered results reveal two new subdomains

Answers:

First subdomain discovered: `delta`

Second subdomain discovered: `yellow`

Lessons Learned

- Certificate Transparency logs are publicly accessible and provide historical SSL/TLS certificate data useful for subdomain discovery
- DNS brute forcing automates the process of testing thousands of potential subdomains from wordlists
- OSINT tools like Sublist3r aggregate data from multiple sources simultaneously for comprehensive results
- Virtual host enumeration can discover subdomains not listed in public DNS records
- Filtering false positives (using `-fs` in ffuf) is essential when fuzzing to identify legitimate results
- Multiple enumeration techniques should be combined for comprehensive subdomain discovery
- Hidden subdomains (dev environments, admin panels) may only be accessible through virtual host enumeration
- Pre-defined wordlists (like SecLists) are crucial for effective brute force enumeration

---

Resources

TryHackMe

crt.sh - Certificate Transparency Search

SecLists - Security Testing Wordlists

ffuf - Fast Web Fuzzer

dnsrecon - DNS Enumeration Tool

Sublist3r - Subdomain Enumeration Tool

OWASP - Subdomain Enumeration

Authentication Bypass

Overview

Room URL: https://tryhackme.com/room/authenticationbypass

Difficulty: Easy

Category: Web Application Security/Authentication

Objective

Learn how to exploit common authentication vulnerabilities including username enumeration, brute force attacks, and logic flaws in cookie-based authentication systems.

Table of Contents

Introduction

What are Authentication Vulnerabilities?

Authentication vulnerabilities are weaknesses in the way applications verify user identity. These flaws can allow attackers to:

- Enumerate valid usernames
- Brute force passwords
- Bypass authentication mechanisms
- Gain unauthorized access to accounts
- Escalate privileges

Why This Matters

Authentication is the first line of defense for web applications. Weak authentication mechanisms can lead to:

- Account takeovers
- Unauthorized access to sensitive data
- Privilege escalation
- Complete system compromise

Technique 1: Username Enumeration

What is it?
Username enumeration is the process of identifying valid usernames in a system by analyzing application responses. This is often the first step in an authentication attack.

Common Indicators:

- Different error messages for valid vs. invalid usernames
- Different response times
- Different HTTP status codes
- Different page content or redirects

Why This is Dangerous:
Once attackers have a list of valid usernames, they can:

- Focus brute force attacks on known accounts
- Attempt credential stuffing with leaked passwords
- Target specific users for social engineering

Tool: ffuf for Username Enumeration

ffuf can be used to test multiple potential usernames against a registration or login form and identify which ones already exist in the system.

Basic Syntax for POST Requests:

```
ffuf -w [wordlist] -X POST -d "param1=FUZZ&param2=value" -H "Content-Type: application/x-www-form-urlencoded" -u [URL] -mr "[match-text]"
```

Key Flags:

- `-w`: Wordlist containing potential usernames
- `-X`: HTTP method (POST in this case)
- `-d`: Data to send in the POST request
- `FUZZ`: Keyword that gets replaced with wordlist values
- `-H`: Additional headers (Content-Type for form data)
- `-u`: Target URL
- `-mr`: Match text - only show responses containing this text

Technique 2: Brute Force Attacks

What is it?

A brute force attack is an automated process that tries a list of commonly used passwords against either a single username or a list of usernames until a valid combination is found.

How it Works:

The attacker uses wordlists containing:

- Common passwords (password123, qwerty, etc.)
- Leaked password databases
- Dictionary words
- Password patterns

Multiple Wordlists in ffuf:

When using multiple wordlists (usernames + passwords), you need to specify custom keywords instead of just FUZZ.

Syntax:

```
ffuf -w [wordlist1]:W1,[wordlist2]:W2 -X POST -d "username=W1&password=W2" -H "Content-Type: application/x-www-form-urlencoded" -u [URL] -fc [filter-code]
```

Key Flags:

- `-w [file]:W1,[file]:W2`: Multiple wordlists with custom keywords
- `W1`, `W2`: Custom keywords for each wordlist
- `-fc`: Filter by HTTP status code (exclude specific codes)

Why `-fc 200` is Used:

A successful login typically redirects (302) or shows a different status code, while failed attempts return 200. By filtering out 200 responses, we only see successful authentications.

Technique 3: Logic Flaws - Cookie Manipulation

What are Cookies?

Cookies are small pieces of data stored by the browser and sent with every request to a website. They're commonly used for:

- Session management
- User preferences
- Authentication state

Why Cookie Manipulation Works:

Poorly implemented authentication systems may store sensitive information in cookies without proper validation, allowing attackers to modify their authentication state or privileges.

Method 1: Plain Text Cookies

What is it?

Some applications store authentication data in cookies as plain, readable text.

Example:

```
Set-Cookie: logged_in=true; Max-Age=3600; Path=/
Set-Cookie: admin=false; Max-Age=3600; Path=/
```

The Vulnerability:

If the server trusts cookie values without server-side validation, an attacker can simply modify these values.

Testing with curl:

```
curl -H "Cookie: logged_in=true; admin=true" http://target.com/page
```

Method 2: Hashed Cookies

What is Hashing?

A hash is a one-way cryptographic function that converts data into a fixed-length string of characters. The same input always produces the same output, but the process cannot be reversed.

Common Hash Types:

Original	Hash Method	Output
1	MD5	c4ca4238a0b923820dcc509a6f75849b
1	SHA-1	356a192b7913b04c54574d18c28d46e6395428ab
1	SHA-256	6b86b273ff34fce19d6b804eff5a3f5747ada4eaa22f1d49c01e52ddb7875b4b
1	SHA-512	4dff4ea340f0a823f15d3f4f01ab62eae0e5da579ccb851f8db9dfe84c58b2b37b89903a740e1ee172da793a6e79d560e5f7f9bd058a12a280433ed6fa46510a

The Vulnerability:

While hashes can't be reversed, they can be looked up in databases of pre-computed hashes.

Tools:

- https://crackstation.net/ - Database of billions of hashes
- HashCat - Offline hash cracking tool
- John the Ripper - Password cracking tool

Method 3: Encoded Cookies

What is Encoding?

Encoding converts data into a different format using a scheme that can be easily reversed. Unlike hashing, encoding is designed to be reversible.

Purpose of Encoding:

- Convert binary data to human-readable text
- Safely transmit data over mediums that only support ASCII characters
- NOT for security (it's easily reversible)

Common Encoding Types:

- **Base32**: Uses A-Z and 2-7
- **Base64**: Uses a-z, A-Z, 0-9, +, /, and = for padding

Example:

```
Cookie: session=eyJpZCI6MSwiYWRtaW4iOmZhbHNlfQ==
```

Decoded:

```
{"id":1,"admin":false}
```

The Attack:

1. Decode the base64 string
2. Modify the JSON (change admin to true)
3. Encode back to base64
4. Send modified cookie

Tools:

- Command line: `echo [string] | base64` or `echo [string] | base64 -d`
- Online: https://www.base64decode.org/
- CyberChef: https://gchq.github.io/CyberChef/

Walk Through

Task 1: Username Enumeration

Objective: Find valid usernames on the Acme IT Support signup page

1. Navigate to the signup page: `http://MACHINE_IP/customers/signup`
2. Test manually by trying username "admin" - observe the error message: "An account with this username already exists"
3. Use ffuf to automate username enumeration:

```
ffuf -w /usr/share/wordlists/SecLists/Usernames/Names/names.txt -X POST -d "username=FUZZ&email=x&password=x&cpassword=x" -H "Content-Type: application/x-www-form-urlencoded" -u http://MACHINE_IP/customers/signup -mr "username already exists"
```

4. Create a file called `valid_usernames.txt` with the discovered usernames
5. Review results to answer questions

Answers:

Username starting with si***: `simon`

Username starting with st***: `steve`

Username starting with ro****: `robert`

Task 2: Brute Force Attack

Objective: Use valid usernames to brute force passwords on the login page

Note: Ensure your `valid_usernames.txt` file contains only clean username data (one username per line, no extra output).

1. Navigate to login page: `http://MACHINE_IP/customers/login`
2. Use ffuf with multiple wordlists (usernames + passwords):

```
ffuf -w valid_usernames.txt:W1,/usr/share/wordlists/SecLists/Passwords/Common-Credentials/10-million-password-list-top-100.txt:W2 -X POST -d "username=W1&password=W2" -H "Content-Type: application/x-www-form-urlencoded" -u http://MACHINE_IP/customers/login -fc 200
```

Command Breakdown:

- `valid_usernames.txt:W1` - First wordlist (usernames) referenced as W1
- `10-million-password-list-top-100.txt:W2` - Second wordlist (passwords) referenced as W2
- `-d "username=W1&password=W2"` - POST data using both wordlists
- `-fc 200` - Filter out HTTP 200 responses (failed logins)

3. Wait for ffuf to find valid username/password combination

Answer: `steve/thunder`

Task 3: Cookie Manipulation - Plain Text

Objective: Manipulate plain text cookie values to gain admin access

Step 1: Test Initial Request

```
curl http://MACHINE_IP/cookie-test
```

Response: "Not Logged In"

Step 2: Set logged_in Cookie

```
curl -H "Cookie: logged_in=true; admin=false" http://MACHINE_IP/cookie-test
```

Response: "Logged In As A User"

Step 3: Set Both Cookies to True

```
curl -H "Cookie: logged_in=true; admin=true" http://MACHINE_IP/cookie-test
```

Response: "Logged In As An Admin" + Flag

Answer: `THM{REDACTED}`

Task 4: Cookie Manipulation - Hashing

Objective: Crack an MD5 hash found in a cookie value

1. Take the hash: `3b2a1053e3270077456a79192070aa78`
2. Navigate to https://crackstation.net/
3. Paste the hash and submit
4. Retrieve the cracked value

Answer: `463729`

Task 5: Cookie Manipulation - Encoding/Decoding

Part 1: Decode Base64 Cookie

Objective: Decode the base64 string: `VEhNe0JBU0U2NF9FTkNPRElOR30=`

Method 1: Command Line

```
echo "VEhNe0JBU0U2NF9FTkNPRElOR30=" | base64 -d
```

Method 2: Online Tool

- Visit https://www.base64decode.org/
- Paste the string and decode

Answer: `THM{REDACTED}`

Part 2: Encode to Base64

Objective: Encode `{"id":1,"admin":true}` to base64

Method 1: Command Line

```
echo -n '{"id":1,"admin":true}' | base64
```

Method 2: Online Tool

- Visit https://www.base64encode.org/
- Paste the JSON and encode

Answer: `eyJpZCI6MSwiYWRtaW4iOnRydWV9`

Lessons Learned

- Username enumeration through error messages is a critical first step in authentication attacks

- Verbose error messages that differentiate between invalid usernames and incorrect passwords are a security vulnerability
- ffuf is a versatile tool that can perform both username enumeration and brute force attacks with proper wordlists
- Multiple wordlists can be used simultaneously in ffuf using custom keywords (W1, W2, etc.)
- Filtering responses by status code (`-fc`) or matching regex (`-mr`) is essential for identifying successful attacks
- Plain text cookies storing authentication state are easily manipulated and should never be trusted without server-side validation
- Cookie values should always be validated server-side, not just checked client-side
- Hashing provides one-way transformation but common values can be looked up in rainbow tables and hash databases
- MD5 and SHA-1 are considered cryptographically broken and should not be used for security purposes
- Encoding (like Base64) is NOT encryption - it's easily reversible and provides no security
- Base64 encoded cookies can be decoded, modified, and re-encoded to change application behavior
- Never store sensitive authorization data (like admin status) in cookies without cryptographic signatures
- Proper authentication systems should use signed session tokens or JWTs to prevent tampering

---

Resources

TryHackMe

ffuf - Fast Web Fuzzer

SecLists - Wordlist Collection

CrackStation - Hash Lookup

Base64 Decode

Base64 Encode

CyberChef - Data Analysis Tool

OWASP Authentication Cheat Sheet

HTTP In Detail - TryHackMe Room

IDOR

Overview

Room URL: https://tryhackme.com/room/[room-name]

Difficulty: Easy

Category: Web Application Security/Access Control

Date Completed: [Date]

Objective

Learn what IDOR (Insecure Direct Object Reference) vulnerabilities are, how to identify them through various obfuscation methods, and how to exploit them to access unauthorized data.

Table of Contents

Introduction

What is IDOR?

IDOR stands for **Insecure Direct Object Reference** and is a type of access control vulnerability.

This vulnerability occurs when a web server:

1. Receives user-supplied input to retrieve objects (files, data, documents)
2. Places too much trust in that input data

3. Fails to validate on the server-side whether the requested object belongs to the user requesting it

Why IDOR Matters

IDOR vulnerabilities can lead to:

- Unauthorized access to other users' data
- Privacy breaches and data leaks
- Horizontal privilege escalation (accessing data of users at the same privilege level)
- Vertical privilege escalation (accessing admin or higher-privilege data)
- Financial fraud and identity theft

Real-World Example

Scenario:

You sign up for an online service and navigate to your profile:

```
http://online-service.thm/profile?user_id=1305
```

You can see your information. Out of curiosity, you change the `user_id` parameter to `1000`:

```
http://online-service.thm/profile?user_id=1000
```

Result: You can now see another user's information!

The Problem: The website doesn't verify that the requested user information belongs to the authenticated user making the request.

Detection Method 1: Encoded IDs

What is Encoding?

When passing data between pages (via POST data, query strings, or cookies), web developers often encode raw data to ensure the receiving server can understand the contents.

How Encoding Works:

Encoding converts binary data into an ASCII string using characters like:

- a-z, A-Z, 0-9
- `=` for padding

Most Common Encoding: Base64

Base64 encoding is easily recognizable and can be identified by:

- Length of the string
- Character set used (alphanumeric + `/` and `+`)
- Padding with `=` characters

Testing for IDOR with Encoded IDs:

1. Identify a base64 encoded ID in the URL, cookie, or POST data
2. Decode the value using https://www.base64decode.org/
3. Modify the decoded value (change user ID, etc.)
4. Re-encode using https://www.base64encode.org/
5. Submit the modified request
6. Check if you can access unauthorized data

Example:

```
Original: user_id=MTIzNDU=
Decoded: 12345
Modified: 12346
Re-encoded: MTIzNDY=
```

Detection Method 2: Hashed IDs

What are Hashed IDs?

Hashed IDs are more complex than encoded IDs, but they may follow predictable patterns. The hash is often the hashed version of an integer value.

Common Pattern:

Sequential integer IDs hashed with MD5, SHA-1, or other algorithms.

Example:

```
ID: 123
MD5 Hash: 202cb962ac59075b964b07152d234b70
```

Testing for IDOR with Hashed IDs:

1. Identify what appears to be a hashed ID
2. Try to determine the hashing algorithm (length can provide clues)
3. Use hash lookup services like https://crackstation.net/
4. If you find the original value, try adjacent values (ID+1, ID-1)
5. Hash those values with the same algorithm
6. Submit requests with the new hashed IDs

Hash Length Indicators:

- 32 characters: Likely MD5
- 40 characters: Likely SHA-1
- 64 characters: Likely SHA-256

Tools:

- CrackStation - Database of billions of hashes
- HashCat - Offline hash cracking
- Online MD5/SHA generators

Detection Method 3: Unpredictable IDs

When to Use This Method:
When IDs cannot be detected or predicted using encoding or hashing methods (e.g., UUIDs, random strings).

The Technique:
Create two separate user accounts and swap their ID numbers between them.

Test Process:

1. Create Account A and note its ID
2. Create Account B and note its ID

3. While logged in as Account A, try to access Account B's resources using Account B's ID
4. If you can view Account B's content while authenticated as Account A, you've found an IDOR vulnerability

Why This Works:

Even with unpredictable IDs, the application may still fail to verify that the authenticated user has permission to access the requested resource.

Variations:

- Test while logged in with different accounts
- Test while not logged in at all
- Test with session tokens from different users

Where IDORs Are Located

IDOR vulnerabilities aren't always obvious. They can be hidden in various places:

1. URL Parameters

```
/profile?user_id=123
/document?doc_id=456
```

2. AJAX Requests

Content loaded asynchronously via JavaScript that isn't visible in the address bar.

3. API Endpoints

RESTful APIs often use IDs in the path or parameters:

```
/api/v1/customer?id=123
/api/users/456/details
```

4. JavaScript Files

Referenced endpoints in client-side JavaScript code.

5. POST Data

IDs sent in form submissions or API requests.

6. Cookies

Session or reference IDs stored in cookies.

Parameter Mining

What is it?

Discovering unreferenced parameters that may have been useful during development but were pushed to production.

Example:

```
Normal endpoint: /user/details (displays your info via session)
Discovered parameter: /user/details?user_id=123 (displays any user's info)
```

How to Find Hidden Parameters:

- Use tools like Burp Suite's Param Miner
- Try common parameter names (id, user_id, uid, account_id, etc.)
- Review JavaScript files for API calls
- Check API documentation if available
- Analyze network traffic in browser developer tools

Walk Through

Task 1: Understanding IDOR

Question: What does IDOR stand for?

Answer: `Insecure Direct Object Reference`

Task 2: Basic IDOR Exploitation

Objective: Find and exploit an IDOR vulnerability on the example website

1. Click the "View Site" button
2. Explore the website and look for user-specific endpoints
3. Identify URLs or API calls with ID parameters

4. Modify the ID parameter to access other users' data
5. Capture the flag

Answer: `THM{REDACTED}`

Task 3: Encoded IDs

Question: What is a common type of encoding used by websites?

Answer: `base64`

How to Identify Base64:

- Look for strings ending in `=` or `==`
- Alphanumeric characters with `/` and `+`
- Can be decoded at https://www.base64decode.org/

Task 4: Hashed IDs

Question: What is a common algorithm used for hashing IDs?

Answer: `md5`

MD5 Hash Characteristics:

- Always 32 characters long
- Hexadecimal (0-9, a-f)
- Can be looked up at https://crackstation.net/

Task 5: Unpredictable IDs

Question: What is the minimum number of accounts you need to create to check for IDORs between accounts?

Answer: `2`

Testing Process:

1. Create first account (note the ID)
2. Create second account (note the ID)

3. While logged in as one account, try accessing the other's resources

---

Task 6: Locating IDORs

Objective: Understand that IDORs can be in various locations including API endpoints and hidden parameters

Read and understand where IDORs might be located beyond just the URL bar.

Task 7: Practical IDOR Exploitation

Setup:

1. Press "Start Machine" button
2. Navigate to the lab URL: https://LAB_WEB_URL.p.thmlabs.com
3. Click on "Customers" section
4. Create an account
5. Log in with your new account

Step 1: Navigate to Your Account

1. Click on "Your Account" tab
2. Notice your username and email are pre-filled

Step 2: Investigate the API Call

1. Open browser Developer Tools (F12)
2. Go to the "Network" tab
3. Refresh the page
4. Observe the API call to: `/api/v1/customer?id={user_id}`

Step 3: Analyze the Response The endpoint returns JSON format containing:

- User ID
- Username
- Email address

Step 4: Test for IDOR

1. Note the `id` parameter in the query string
2. Modify the `id` value to test other user IDs
3. Try `id=1` to see user 1's information
4. Try `id=3` to see user 3's information

Answers:

What is the username for user id 1? `adam84`

What is the email address for user id 3? `j@fakemail.thm`

Lessons Learned

- IDOR is an access control vulnerability where user-supplied input is trusted without server-side validation
- The fundamental problem is lack of authorization checks on the server side
- IDORs can occur in URL parameters, API endpoints, AJAX requests, and hidden parameters
- Base64 encoding is NOT security - it's easily reversible and commonly used in IDOR vulnerabilities
- Hashed IDs may seem secure but can follow predictable patterns or be cracked using rainbow tables
- Even unpredictable IDs (UUIDs) can be vulnerable if the server doesn't verify authorization
- Always check API calls in browser Developer Tools (Network tab) for potential IDOR endpoints
- Testing with multiple accounts is essential for identifying IDOR vulnerabilities
- Parameter mining can reveal hidden parameters that weren't properly secured
- JSON responses from APIs often expose more data than intended
- Proper IDOR prevention requires server-side authorization checks for every resource access
- Never rely solely on obscurity (encoding, hashing) for access control
- Sequential IDs make IDOR testing easier but even random IDs aren't protection without proper authorization
- Browser developer tools are essential for discovering API endpoints and hidden requests

---

Resources

TryHackMe

Base64 Decode

Base64 Encode

CrackStation - Hash Lookup

OWASP - Insecure Direct Object References

PortSwigger - Access Control Vulnerabilities

Burp Suite - Param Miner Extension

OWASP API Security Top 10

File Inclusion

Intro to SSRF

Intro to Cross-site Scripting

Race Conditions

Command Injection

SQL Injection