

Information Security Project

Evaluation of security of an API

Case study: OSCAR EMR Project

Prepared by:

Qasim Ali,

Sai Venkata Anirudh Sikhivahan Varanasi,

Sandeep Kaur,

Ramandeep Sharma,

Khushpreet Singh Brar,

Prepared for:

Brian Campbell

Ig Kolenko

Conestoga College Institute of Technology and Advanced Learning

Submitted in partial fulfillment of the requirements for “Information Security Project”
(INFO8630) course in the Computer Application Security program

Contents

| | |
|-----------------------------------------------|----|
| Abstract | 4 |
| Problem Statement..... | 5 |
| Introduction..... | 5 |
| Problem Statement..... | 5 |
| Relevance/Significance Resources | 5 |
| Resources..... | 6 |
| Literature Review | 7 |
| API Testing General..... | 7 |
| API Security Testing Principles..... | 7 |
| API Testing Tools General | 8 |
| API Secure Testing Tools | 9 |
| OWASP's ZAP (Zed Attack Proxy):..... | 9 |
| Metasploitable | 9 |
| Burp Suite | 9 |
| Nessus..... | 10 |
| SSL Scan | 11 |
| Hydra | 12 |
| Nmap..... | 13 |
| Summary | 14 |
| Methodology..... | 15 |
| Tools Used for API Testing | 17 |
| HYDRA..... | 17 |
| Nmap..... | 18 |
| SSL Scan | 18 |
| Nessus..... | 19 |
| Burp Suite | 19 |
| SOAP UI PRO | 20 |
| Summary | 21 |
| Results..... | 22 |
| Vulnerabilities Identified | 22 |
| Broken Object Level Authorization (BOLA)..... | 23 |
| Broken User Authentication | 23 |
| Sensitive Data Exposure | 24 |
| Lack of Resource & Rate Limiting | 24 |
| Broken Function Level Authorization | 25 |
| Mass Assignment..... | 26 |
| SQL Injection..... | 27 |
| Improper Assets Management..... | 27 |
| Insufficient Logging and Monitoring..... | 27 |
| XPath Injection | 28 |
| Malformed XML..... | 28 |
| Summary of Test results | 29 |
| Zap Proxy | 29 |
| Burp Suite | 29 |

| | |
|-------------------------------------------------|----|
| Nessus..... | 30 |
| Hydra | 31 |
| Nmap..... | 31 |
| Summary | 32 |
| Conclusion | 33 |
| Vulnerabilities..... | 33 |
| Broken Object Level Authorization (BOLA)..... | 33 |
| Broken User Authentication | 35 |
| Excessive Data Exposure..... | 36 |
| Lack of Resource & Rate Limiting | 36 |
| Broken Function Level Authorization | 37 |
| Mass Assignment..... | 38 |
| Security Misconfiguration | 38 |
| Injection..... | 39 |
| Insufficient Logging and Monitoring..... | 40 |
| Malformed XML..... | 41 |
| Summary | 41 |
| References | 43 |
| Appendix A..... | 45 |
| Install OSCAR on VM for SOAP API testing..... | 45 |
| 1 – Installing the Infrastructure Packages..... | 45 |
| 2 – SSL Configuration..... | 45 |
| 3 – Manual Configuration..... | 45 |
| 4 – Java 8 or 11 | 45 |
| 5 – MariaDB 10.3 | 45 |
| 6 – Java 8 or 11 | 46 |
| 7 – Installing OSCAR..... | 46 |
| Oscar Screenshots..... | 46 |
| APPENDIX B..... | 48 |
| Security Issues Identified in OSCAR API | 48 |
| Burp Suite | 48 |
| HYDRA..... | 52 |
| Nmap..... | 54 |
| ZAP Scanning Report | 55 |

Abstract

This project report, conducted by Qasim Ali, Sai Venkata Anirudh Sikhivahan Varanasi, Sandeep Kaur, Ramandeep Sharma, and Khush Preet Singh Brar, presents an evaluation of the security of an API, with a case study on the OSCAR EMR project. The report provides an overview of the OSCAR EMR project and its significance in the healthcare industry, followed by a comprehensive security evaluation of the API.

The team utilized various security testing techniques and tools, such as manual testing, automated scanning, and code analysis, to identify vulnerabilities in the API. The report includes an in-depth analysis of the identified vulnerabilities and their potential impact on the security of the OSCAR EMR system. These vulnerabilities include Insecure Direct Object References, Predictable Object References, Insufficient Authorization Checks, Privilege Escalation, and Cross-Site Request Forgery (CSRF). The team also demonstrated proof-of-concept demonstrations of the vulnerabilities to emphasize their severity and potential impact further.

Based on their findings, the team provided recommendations to address the identified vulnerabilities and improve the overall security of the OSCAR EMR API. These recommendations include implementing proper access controls, using parameterized queries to prevent SQL injection attacks, and implementing proper error handling and input validation to prevent XSS attacks.

The report concludes that the findings can serve as a valuable reference for developers and organizations seeking to improve the security of their APIs. This project report provides a valuable contribution to API security. Its detailed analysis of the OSCAR EMR API's security serves as a valuable reference for developers and organizations seeking to improve the security of their APIs.

Problem Statement

Introduction

The scope of centralized operations has grown leaps and bounds in terms of its magnitude. However, with evolution over time, the user community has made a few tweaks to the systems architecture such that the operations are viable both as a monetary source and as an ease of use for the vendors. This tweak looks beneficial from the outer perspective. However, as we dig deeper into the operations, we find something that intrigues us, i.e., the functions of API.

API, in simpler terms, can be interpreted as the medium that initiates two different applications to communicate with each other.

Since there is a vast scope in this area, in this project, we will do a case study analyzing the current state of SOAP (Simple Object Access Protocol) API (API, 2023) in the OSCAR EMR (Electronic Medical Record) project and will make suggestions to improve its security. The project will be conducted in a manner that will allow us to demonstrate how to test an API from the cybersecurity perspective.

OSCAR (Oscar, 2022) is an Electronic Medical Record (EMR) System. It has been designed by doctors and is used by various clinics in different provinces of Canada. It has features that both doctors and frontline healthcare staff utilize. SOAP (Simple Object Access Protocol) (Soap, 2022) has been widely used by different APIs for many years. In the Context of OSCAR, many 3rd party apps use SOAP API.

The project team will conduct a thorough security analysis of the current SOAP-API and suggest improvements in security.

Problem Statement

A case study to evaluate the security of the SOAP API of the OSCAR EMR project. The case study includes best practices for assessing the SOAP API from the cyber security perspective and how to remediate the deficiencies found.

Relevance/Significance Resources

The conscious effort being put into deployment by the team to work on the SOAP-API of OSCAR is to initiate a few analysis observations to realize the API's working methodology with security contexts in the foreground. As per the team's preliminary observations, we have collectively zeroed in on the fact that the users or people eligible to operate OSCAR are being provided with more data than they are deemed to get access to.

So, as a base footing into this project, the team is willing to work its way on the grounds of our initial observations. Upon further progress, we believe there would be scope for security-related improvements, and even a little contribution from our side would be a betterment not just in security prospects but to society as well.

Resources

API Security testing is dependent on automation and tool. For the testing of API, we have identified the following tools for API testing from a security perspective:

- **Nessus:** used for Networks and applications.
- **Nmap:** It is also used for networking.
- **Burp Suite:** It is a web proxy and used for manual testing of web applications and APIs.
- **Hydra:** It is used for brute force testing.
- **SSL Scan:** it is used for ssl configuration testing.
- **Metasploit:** it is used for exploitation and scanning.

We will use these tools to test the OSCAR EMR SOAP API from a security perspective.

Literature Review

According to Frye (n.d.), an API or "Application Programming Interface" acts as a software intermediary that facilitates communication between two internet-based applications. APIs define how software components should interact, enabling different systems and applications to communicate with each other more easily.

In the previous unit, we identified resources, and in this unit, we will delve deeper into the available tools for API testing, both generally and from a security perspective. It is crucial to follow general testing principles for API testing as they not only ensure functional accuracy but also contribute to making the API more secure. Although this document aims to explore more from a security perspective, we will provide a brief overview of general testing principles.

API Testing General

API enables data exchange and allows programmers to incorporate the data or functionality a service or application provides into their applications. To perform comprehensive API testing, various methods can be used.

Unit testing involves developers writing tests for specific sections or parts of the code and utilizing the API to test individual endpoints before integration with other systems (Pp_Pankaj, 2023).

Functional testing ensures the API functions as intended and meets the functional requirements. It involves testing each endpoint and validating the response (Pp_Pankaj, 2019). Load testing evaluates the stability and performance of an API under heavy loads by simultaneously sending numerous requests to the API while monitoring response time and error rate.

Security testing aims to identify API flaws that attackers could exploit and includes testing for common security vulnerabilities, such as cross-site scripting and SQL injection, as well as weaknesses in authentication (Pp_Pankaj, 2023).

Integration testing assesses whether the API is compatible with other systems by testing it alongside other components like databases and front-end applications (Sanjay, 2023).

API Security Testing Principles

From a security standpoint, a comprehensive API testing strategy is as follows (Chawla, 2021):

Input Validation:

- Test the API to ensure it properly validates and sanitizes the data entered to prevent SQL Injection and Cross-Site Scripting attacks.
- Look for buffer overflows and other injection attacks.
- Look for attacks like format string and code injection.

Authenticity and Authority:

- Run a test to ensure that the API's authentication method is safe and gives users the proper credentials to sign in.
- Look for dictionaries, known passwords, and brute force attacks.
- Ensure that users can only access resources granted to them by testing the authorization procedure.
- Check to see if the principle of least privilege is being followed, which states that users should only have access to the information they require to do their job.

Controlling Sessions:

- Verify that the API appropriately manages user sessions and prevents attacks like a hijacking.
- To test for session fixation attacks, ensure that sessions are correctly invalidated when a user logs out or after a predetermined amount of inactivity.
- Confirm that the API uses secure session tokens and is safe from attacks like cross-site request forgery (CSRF).

Encryption:

- Verify the encryption of sensitive data, such as passwords and credit card numbers, by examining the API.
- Ensure the encryption algorithm is secure and the keys used for encryption are appropriately managed.

Solving Problems:

- Conduct a test to ensure that the API's error-handling mechanism prevents hackers from accessing sensitive data such as stack traces and database error messages.
- Ensure that the API returns error messages that are consistent and meaningful and do not reveal any sensitive information.

Management of Access:

- Conduct tests on the API's access control mechanism to ensure that it functions correctly and that users cannot access restricted resources or sensitive data.
- Verify that the API appropriately handles access control for all resources and methods, including those used for administrative tasks.
- Keeping the rate low:
- Test the API's rate-limiting mechanism to ensure that it properly limits the number of requests from a single source to prevent denial of service (DoS) attacks.

API Testing Tools General

Utilizing these tools can increase the accessibility of implementing and testing APIs (TestingEmpire (31)).

Postman, a popular tool for testing and documenting APIs, enables testing individual API endpoints, managing environments, and saving test collections.

SoapUI, an open-source tool, is capable of functional, load, and security testing of SOAP and REST APIs.

Curl, a command-line tool, permits quick testing of API endpoints and analysis of the response by sending HTTP requests.

Jmeter, an open-source load-testing tool, can send multiple requests simultaneously to test APIs and web applications.

Fiddler, a web traffic debugging proxy, can capture and scrutinize HTTP requests and responses, including API-related ones.

API Secure Testing Tools

Tools like these can be used to test API security:

OWASP's ZAP (Zed Attack Proxy):

OWASP ZAP is an open-source tool that can test the security of web applications and APIs (Tesauro and kingthorin (n.d.)). It can identify and address vulnerabilities like CSRF, SQL injection, and XSS. The tool has an easy-to-use interface and supports automated testing.

Metasploitable

Metasploitable is a vulnerable virtual machine (VM) used to test and show common vulnerabilities in network services and web applications. It was made a target for ethical hacking and penetration testing exercises. The Apache web server, FTP server, database servers, and several other vulnerable services and applications are included in Metasploitable, based on the Linux operating system. By attempting to exploit the VM's vulnerabilities, Metasploitable aims to enable security professionals and students to practice and enhance their penetration testing abilities. Metasploitable is a free, open-source project that can be downloaded. However, it should be used with caution because it is illegal and unethical to attempt to exploit vulnerabilities in real-world systems. Metasploitable should only be used in a controlled, isolated setting for educational and testing purposes (Metasploit, n.d.).

should include functional, performance, and security testing. Also, security testing should be done regularly to keep APIs safe and protect them from new and changing security threats.

Burp Suite

Burp Suite is a tool that can test all aspects of web security, including authentication and session management testing, and API security. You can manually test, check for vulnerabilities, and intercept API requests and responses with this tool. Burp Suite has a professional version with additional features like an automated vulnerability scanner (Gianchandani, 2012).

It is intended to help information security experts evaluate the security of online applications. Various tasks relating to web application security testing can be carried out using the suite's capabilities, including altering online traffic, finding, and exploiting vulnerabilities, and automating security tests.

Burp Suite, a widely used tool by information security professionals, provides various critical features for web application security testing.

One of these features is the **Web Proxy**, which monitors, examines, and modifies web traffic between a client and a server. Another feature is the **Spider**, which automatically crawls a web application to identify its functionality and potential attack surfaces. Additionally, the **Scanner** can automatically detect vulnerabilities such as cross-site scripting (XSS), SQL injection, and cross-site request forgery (CSRF) in web applications.

Intruder is also available to automate attacks to identify security flaws in web applications. Meanwhile, the Repeater tool allows for manual testing and optimization of requests sent to a web application. Lastly, the **Sequencer** tool helps verify the randomness of session tokens generated by a web application to prevent session hijacking.

Overall, the comprehensive and highly adaptable Burp Suite platform is widely used by information security experts to evaluate the security of web applications.

Nessus

Nessus, an application for remote security scanning, conducts over 1200 checks on a specific machine to identify potential security threats that could allow criminal hackers to access any computer connected to the network. If any flaws are detected, the application alerts the user immediately. (Nessus, n.d.)

This tool is especially beneficial for administrators responsible for managing multiple computers connected to the internet. It helps them to ensure that their domains remain free of the common vulnerabilities that hackers and viruses frequently target. (Nessus, n.d.)

What Nessus Does

To comprehend how Nessus and other port-scanning security solutions work, it is necessary to understand how different services, such as a web server, SMTP server, and FTP server, are accessible on a remote server. Most high-level network traffic, including email and web pages, is conveyed securely by a TCP stream, a high-level protocol.

For instance, when communicating with a web server or SMTP server, one connects to port #80 or #25 as a computer divides its physical connection to the network into thousands of logical pathways or ports.

Nessus examines each port on a computer to determine if any services are listening on them. The software identifies the service running on each port and then assesses that service for any vulnerabilities that hackers could exploit to launch an attack. Nessus is called a "remote scanner" because it can test a machine without installation. This means the software can be installed on a single machine and then used to test as many other machines as required.

BENEFITS OF NESSUS

The advantages of Nessus over other network vulnerability scanners are numerous. One important point is that unlike other scanners, Nessus does not make assumptions about server configurations. Other scanners may overlook actual vulnerabilities by assuming, for example, that port 80 is the only web server. (Wendlandt, 2023)

Nessus is also highly expandable, with a scripting language that allows users to write tests particular to their system. In addition, the Nessus plug-in interface enables many free plug-ins, which often focus on identifying specific infections or vulnerabilities. To reduce the time between an exploit appearing in the wild and detecting it with Nessus, the team frequently refreshes the list of vulnerabilities to check for. (Wendlandt, 2023)

DRAWBACKS

One drawback of Nessus is that it is not a comprehensive security solution, but merely a tool that scans systems for potential vulnerabilities. The system administrator is responsible for patching these vulnerabilities to form a sound security strategy. (Wendlandt, 2023)

Despite its limitations, Nessus is an open-source tool, allowing users to view and alter the source code. It also frequently provides recommendations for mitigating vulnerabilities when they are discovered. (Wendlandt, 2023)

SSL Scan

SSLScan is a tool that operates through the command line and is used to run various tests on a target. This tool generates a comprehensive list of protocols and cyphers that an SSL/TLS server accepts and other relevant data that can be utilized in security testing. To scan SSL on the IP address 10.7.7.5, SSLScan can be used.(O'Reilly (2023))

Use

Using a collection of secure web servers, the SSL Certificate Scanner Tool may swiftly retrieve and examine SSL Certificate expiration dates and other details. Additionally, it verifies whether weak SSLv2/SSLv3 connections (if your OS permits this) are accepted by the web server itself.

Benefits

Depending on an organization's needs, the advantages may be innumerable and rather ambiguous. Most businesses move backwards from the "need" or "business case" of SSL inspection. Here, we'll concentrate on some of the more common requirements:

- improved application control and URL classification
- Virus/Malware analysis (i.e., sandboxing, AV scanning, etc.)
- DLP
- monitoring and troubleshooting of advanced networks.
- improved awareness of cloud applications
- Send visitors to other tools.
- increased effectiveness of current security measures

All-in-one devices can perform most of the tasks, however size is necessary if SSL inspection is performed on the same device. Organizations will inevitably need to size not only the amount of SSL inspection but also the tasks it will perform and the anticipated load for those operations. The expense of oversizing might not matter because the device might be the sole place where SSL inspection is required. All-in-one devices often don't work that way because most firms prefer the advantage of using additional tools to view unencrypted traffic. Although the functionality of some all-in-one devices to send decrypted traffic to a mirror port is currently poor.

Sizing is based solely on SSL inspection because purpose-built devices almost never provide any other functionality beyond those described in the "How it Works" section. The other hardware that examines unencrypted communication is sized for the functions it will perform and the load it is expected to carry. Once more, many businesses will desire to monitor unencrypted data using different techniques (current or future). Decrypt once and send to many is a benefit of the purpose-built solutions (e.g., unencrypted SSL traffic is simultaneously sent to a web proxy and a network IDS).

Many of these functions are also offered by cloud providers, however with some of them, the data is kept in their environment for logging and reporting. When information is combined for a comprehensive understanding of security, however, this might lead to a disconnect. (Cardwell, 2018)

Drawbacks

Users may not anticipate the same level of end-to-end security from SSL and TLS. There are issues with how effectively browsers are communicating SSL information to users, even in the absence of SSL inspection. The mere existence of the term "SSL inspection" should serve as a clear warning that what you believe SSL to be doing for you is fundamentally flawed. The errors committed by the creators of SSL inspection tools exacerbate the issue.

The decision to implement SSL inspection capabilities in their environment may need to be re-evaluated by system administrators. You can utilize CERT Tapioca to confirm that the SSL inspection solution is taking all necessary precautions to reduce the users' increased risk. System administrators might at the very least get in touch with the providers of SSL inspection tools to ask them to confirm the appropriate configuration settings and behavior. (Dormann, 2015)

Hydra

Hydra is a software tool designed for ethical hackers and penetration testers to help them acquire network service credentials through brute-force attacks. The software can execute rapid dictionary attacks on over 50 protocols, including Telnet, FTP, HTTP, HTTPS, SMB, databases, and other services. The Hacker's Choice, a hacker collective, developed Hydra in 2000 as a proof-of-concept program to demonstrate how network logon services could be attacked. *Hydra* is a parallelized login cracker, meaning it can have multiple active connections simultaneously. It contrasts sequential brute-forcing, which speeds up the password cracking process. To work with Hydra, it is important to understand its standard formats and the options available for brute-forcing usernames and passwords. There are three categories of attacks: dictionary attacks, password spraying, and single username/password assaults. These methods can be employed to obtain the desired network credentials depending on the situation.

\$ sudo apt-get install hydra.

If Hydra is already installed, use the help command to get started.

\$ hydra -h

All the options available in Hydra are shown below:

```
Syntax: hydra [[-l LOGIN][-L FILE] [-p PASS][-P FILE]] | [-C FILE]] [-e nsr] [-o FILE] [-t TASKS] [-M FILE [-T TASKS]] [-w TIME] [-W TIME] [-f] [-s PORT] [-x MIN:MAX:CHARSET] [-c TIME] [-ISOUvVd46] [-m MODULE_OPT] [service://server[:PORT][:OPT]]

Options:
  -R      restore a previous aborted/crashed session
  -I      ignore an existing restore file (don't wait 10 seconds)
  -S      perform an SSL connect
  -s PORT if the service is on a different default port, define it here
  -l LOGIN or -L FILE login with LOGIN name, or load several logins from FILE
  -p PASS or -P FILE try password PASS, or load several passwords from FILE
  -x MIN:MAX:CHARSET password bruteforce generation, type "-x -h" to get help
  -y      disable use of symbols in bruteforce, see above
  -t      use a non-random shuffling method for option -x
  -e nsr  try "n" null password, "s" login as pass and/or "r" reversed login
  -u      loop around users, not passwords (effective! implied with -x)
  -C FILE colon separated "login:pass" format, instead of -L/-P options
  -M FILE list of servers to attack, one entry per line, ':' to specify port
  -o FILE write found login/password pairs to FILE instead of stdout
  -b FORMAT specify the format for the -o FILE: text(default), json, jsonv1
  -f / -F exit when a login/pass pair is found (-M: -f per host, -F global)
  -t TASKS run TASKS number of connects in parallel per target (default: 16)
  -T TASKS run TASKS connects in parallel overall (for -M, default: 64)
  -w / -W TIME wait time for a response (32) / between connects per thread (0)
  -c TIME wait time per login attempt over all threads (enforces -t 1)
  -4 / -6 use IPv4 (default) / IPv6 addresses (put always in [] also in -M)
  -v / -V / -d verbose mode / show login/pass for each attempt / debug mode
  -O      use old SSL v2 and v3
  -K      do not redo failed attempts (good for -M mass scanning)
  -g      do not print messages about connection errors
  -U      service module usage details
  -m OPT  options specific for a module, see -U output for information
  -h      more command line options (COMPLETE HELP)
  server the target: DNS, IP or 192.168.0.0/24 (this OR the -M option)
  service the service to crack (see below for supported protocols)
  OPT     some service modules support additional input (-U for module help)

Supported services: adam6500 asterisk cisco cisco-enable cobaltstrike cvs ftp[s] http[s]--[head|get|post] http[s]--[get|post]--form http-proxy http-proxy-urlemum icq imap[s] irc ldap2[s] l
dap3[s]--[cram|digest|md5][s] mssql mysql nntp oracle-listener oracle-sid pcanywhere pcnfs pop3[s] redis rexec rlogin rpcap rsh rtsp s7-300 sip smb smtp[s] smtp-enum snmp socks5 ssh sshkey
teamspeak telnet[s] vmauthd vnc xmp

Hydra is a tool to guess/crack valid login/password pairs.
Licensed under AGPL v3.0. The newest version is always available at:
https://github.com/vanhauser-thc/thc-hydra
Please don't use in military or secret service organizations, or for illegal
purposes. (This is a wish and non-binding - most such people do not care about
laws and ethics anyway - and tell themselves they are one of the good ones.)
These services were not compiled in: nfp firebird memcached mongodb ncp oracle postgres radmin2 rdp sapr3 svn smb2.
```

Output: `$ hydra -h`

Hydra is a networking tool that can be used for brute-force attacks on services such as FTP and SSH. It is highly adaptable and can be easily modified to include new protocols and services due to its modular design and parallelization capabilities. It is widely recognized as a valuable addition to a penetration tester's toolkit (Shivanandhan, 2022).

Nmap

Nmap (Network Mapper) is a well-known tool often referred to as the "sysadmin's Swiss Army knife" (Hegde, 2019). When managing a business network, it is common to encounter issues that require debugging and troubleshooting. These issues can accumulate in the work log over time. Nmap is a valuable tool that can help with these tasks by allowing us to scan our network and gather information about each connected host.

Nmap supports a variety of scanning methods, including TCP connect, TCP SYN (half-open), FTP, and UDP. It also provides a range of scan types, such as ICMP (ping sweep), Proxy (bounce attack), Null scan, Xmas, FIN, SYN sweep, and IP Protocol. With Nmap, we can discover what services each host is running, the number of hosts connected, and other details.

In addition to these features, Nmap offers advanced capabilities such as OS detection via TCP/IP fingerprinting, stealth scanning, parallel scanning, direct (non-portmapper) RPC scanning, flexible target and port specification, fragmentation scanning, and down host detection via parallel pings. These advanced features make Nmap a versatile tool for network management.

Overall, Nmap is a valuable tool for system administrators to manage their networks. By utilizing Nmap's various scanning methods and advanced features, administrators can identify and troubleshoot issues within their networks.

The man page for Nmap specifies the following syntax:

```
nmap [ <Scan Type> ... ] [ <Options> ] { <target specification> }
```

While Nmap has several switch choices available, let's concentrate on the most useful one. Use the `Nmap hostname>` command as follows to scan the hostname for this use case:

```
root@localhost$ nmap localhost.example.com
```

```
Starting Nmap 6.40 ( http://nmap.org ) at 2019-07-21 16:37 EDT
```

```
Nmap scan report for localhost.example.com(127.0.0.1)
```

```
Host is up (0.0000080s latency).
```

```
Not shown: 991 closed ports
```

| PORT | STATE | SERVICE |
|------|-------|---------|
|------|-------|---------|

| | | |
|--------|----------|-----|
| 22/tcp | filtered | ssh |
|--------|----------|-----|

| | | |
|--------|------|------|
| 25/tcp | open | smtp |
|--------|------|------|

| | | |
|--------|------|------|
| 80/tcp | open | http |
|--------|------|------|

| | | |
|---------|------|---------|
| 111/tcp | open | rpcbind |
|---------|------|---------|

| | | |
|---------|------|-----|
| 631/tcp | open | ipp |
|---------|------|-----|

| | | |
|----------|------|-----|
| 3000/tcp | open | ppp |
|----------|------|-----|

| | | |
|----------|------|----------------|
| 4000/tcp | open | remoteanything |
|----------|------|----------------|

| | | |
|----------|------|------------|
| 8080/tcp | open | http-proxy |
|----------|------|------------|

| | | |
|----------|------|------------|
| 9090/tcp | open | zeus-admin |
|----------|------|------------|

```
Nmap done: 1 IP address (1 host up) scanned in 0.11 seconds
```

Summary

During this unit, we conducted a study on the tools that were previously identified in the Literature Review section. Our primary focus was on security-related tools, and we examined both their benefits and limitations. We aimed to determine the purpose of each tool and how it can be utilized to conduct security testing of an API. The outcomes of this research will be utilized in subsequent sections to create a detailed methodology for API Security Testing. These tests will be performed on OSCAR EMR SOAP API later.

Methodology

In this unit, the major vulnerabilities were identified for the hypothesis to declare an API secure or insecure. The below sections in this document include details of the tools which would run on an API to test its security. A few API vulnerabilities are described below:

1. **Lack of Input Validation:** APIs that do not validate input data effectively can be vulnerable to attacks such as injection attacks, where an attacker sends malicious input data that can execute unwanted code or reveal sensitive information.
2. **Insufficient Authentication and Authorization:** APIs that lack strong authentication and authorization mechanisms can be vulnerable to attacks where attackers can gain access to sensitive data or perform unauthorized actions.
3. **Broken Access Controls:** APIs that have broken access control mechanisms can be vulnerable to attacks where attackers can access sensitive data or perform unauthorized actions.
4. **Insufficient Rate Limiting:** APIs that do not limit the rate of requests from a client can be vulnerable to denial-of-service attacks, where an attacker sends many requests to the API, overwhelming it and making it unavailable to other clients.
5. **Insufficient Encryption:** APIs that do not use encryption or use weak encryption can be vulnerable to attacks where an attacker can intercept and read sensitive data transmitted over the API.
6. **Inadequate Error Handling:** APIs that do not handle errors effectively can be vulnerable to attacks where attackers can use error messages to gain information about the system or exploit vulnerabilities.

These are just a few hypotheses, and API vulnerabilities can result from a wide range of factors, including both technical and non-technical issues. It's important to thoroughly assess the security of an API to identify potential vulnerabilities and take appropriate measures to mitigate them.

Declaring SOAP API secure or insecure is a complex task. It requires the implementation of testing the API at multiple levels for different vulnerabilities. In the earlier unit we identified the principles and tools which could be used to test API. As part of this unit OSCAR was installed on a VM. Please refer to **Appendix A** for details about the installation of OSCAR.

Following vulnerabilities were identified as most critical for security testing of API:

- Broken Object Level Authorization
- Broken User Authentication
- Excessive Data Exposure
- Lack of Resource & Rate Limiting
- Broken Function Level Authorization
- Mass Assignment
- Security Misconfiguration
- Injection
- Improper Assets Management
- Insufficient Logging and Monitoring
- SQL Injection Scan
- XPath Injection Scan
- Malformed XML

Broken Object Level Authorization (BOLA)

Object level authorization is controlling user access to resources. A user with administrator privileges can access more resources than a normal user. This vulnerability is currently at the top of the OWASP top 10 list (Apisec, 2023).

Broken User Authentication

Application functions related to authentication are not often correctly implemented. It often allows hackers to exploit compromise passwords, keys, or sessions tokens.

Sensitive Data Exposure

It is often seen that applications often do not properly protect sensitive data such as financial and health care. This may result in hackers stealing such sensitive data. In our case study project OSCAR, this vulnerability is extremely relevant as it is used in the medical field.

Lack of Resource & Rate Limiting

These vulnerabilities occur when developers forget to limit the size of objects and fail to implement controls for the number of inbound requests and access requests from a single client.

Broken Function Level Authorization

Often complex access control policies in different groups, roles and some times unclear separation between them can lead to broken authorization flaws.

Mass Assignment

Binding input data from client without proper properties filtering based on the whitelist, usually leads to mass assignment. The possibility for a user not appropriately authorized and equipped can tamper the agronomics of the permissions and levels of accessibility is usually seen as a threat possible via Mass Assignment.

Security Misconfiguration

It is usually the result of unsecure default configurations, incomplete or ad-hoc configurations, open cloud storage, misconfigured HTTP headers, unnecessary HTTP methods, allowing Cross-Origin resource sharing (CORS).

SQL Injection

It occurs when a SQL query is sent to an interpreter as part of a command or query. The attackers send malicious data as part of query which can trick the interpreter in executing unintended command and accessing data without proper authorization.

Improper Assets Management

API Expose a lot of endpoints and that makes proper management and documentation a tough job. Exposing deprecated and debug endpoints might result in this vulnerability.

Insufficient Logging and Monitoring

Insufficient logging and monitoring along with ineffective integration with incident response allow attackers to conduct attacks undetected.

XPath Injection

The attackers use XPath injection to find out the structure of XML data or access sensitive data. The API should be tested for its ability to handle incorrect or malicious XPath queries.

Malformed XML

The malformed xml is usually sent by attackers to crash a vulnerable server or to execute arbitrary commands. The server should be tested how it responds to the malformed XML.

Tools Used for API Testing

As identified, we will use security tools to test the vulnerabilities discussed above. In this unit, the tools were installed, in the next unit, they will be run on OSCAR-EMR API. The details about how to use these tools for API testing are as follows:

HYDRA

Hydra is a potent command-line API testing tool that supports several different protocols, including HTTP, HTTPS, and HTTP/2. The fundamental actions for utilizing Hydra for API testing are as follows:

Identifications of API Endpoints: The API's endpoints should be identified. These are the URLs the API makes available. You can retrieve this information by using a tool like Postman or the API documentation. We started preparing a list of OSCAR EMR API endpoints.

Attack types: Hydra can be used to conduct a variety of attack types, including dictionary attacks, brute-force attacks, and more, are supported by Hydra. You can select the type of assault that best suits your requirements based on your use case.

Customize the Hydra settings: You can tailor your attack by configuring the many Hydra options. The number of threads, the timeout, the user-agent, and other variables are examples of frequent choices. To view all available options, use the `hydra -h` command.

Run the assault: After setting up Hydra, you can launch the attack by selecting the attack type and API endpoint. An example command to check the administrator's password on the endpoint "`https://example.com/login`" is provided below:

```
$ hydra -l admin -P /path/to/password/list.txt https://example.com/login http-post-form "/login.php:user=^USER^&pass=^PASS^:Invalid username or password."
```

The password of the user "admin" is being tested by Hydra using a dictionary attack (-P option) on the endpoint "`https://example.com/login`." When Hydra sends the login information via the http-post-form method, it anticipates seeing the message "Invalid username or password."

Examine the findings: After the attack is over, Hydra will present a summary of the findings. Using this knowledge, you may spot any security holes in the API and take the necessary steps to close them. Overall, Hydra is a strong API testing tool, but it should only be used under responsible conditions and with the proper authorizations.

Nmap

Nmap is usually used for network research and security auditing; however, it can occasionally be applied to API testing. Nmap can be used for testing APIs in the following ways:

Check for open ports: On a server providing an API, Nmap can be used to check for open ports. This can provide information about the active services and the open communication ports.

Locate web servers: Nmap can assist in locating the web servers using the ports listed above if the API is delivered over HTTP or HTTPS. This can provide information on the software being used to host the API and any potential vulnerabilities it may have.

Sending personalized requests to a web server is possible with Nmap. This can be used to test different API endpoints and methods as well as to check for security flaws like cross-site scripting and SQL injection.

Nmap may be used to determine the version numbers of web servers and other software that is running on a server. Finding vulnerabilities unique to software versions may be possible using this data.

Examine response headers for security headers such as Content-Security-Policy and Strict-Transport-Security using Nmap. These headers can shield an API from several assaults.

SSL Scan

SSLScan is a command-line tool who can perform a variety of tests and it returns comprehensive list of the protocols and ciphers accepted by an SSL/TLS server.

sslscan 10.7.7.5

SSLScan's color code provides us a quick reference about the severity, in terms of security, of the displayed results. Red (allowing SSLv3 and using DES and RC4 ciphers) indicates an insecure configuration, while green or white is a recommended one.

```
root@kali:~# sslscan 10.7.7.5
Version: 1.11.10-static
OpenSSL 1.0.2-chacha (1.0.2g-dev)

Testing SSL server 10.7.7.5 on port 443 using SNI name 10.7.7.5

  TLS Fallback SCSV:
Server does not support TLS Fallback SCSV

  TLS renegotiation:
Secure session renegotiation supported

  TLS Compression:
Compression enabled (CRIME)

  Heartbleed:
TLS 1.2 not vulnerable to heartbleed
TLS 1.1 not vulnerable to heartbleed
TLS 1.0 not vulnerable to heartbleed

Supported Server Cipher(s):
Preferred TLSv1.0 256 bits DHE-RSA-AES256-SHA DHE 1024 bits
Accepted TLSv1.0 256 bits AES256-SHA
Accepted TLSv1.0 128 bits DHE-RSA-AES128-SHA DHE 1024 bits
Accepted TLSv1.0 128 bits AES128-SHA
Accepted TLSv1.0 128 bits RC4-SHA
Accepted TLSv1.0 128 bits RC4-MD5
Accepted TLSv1.0 112 bits EDH-RSA-DES-CBC3-SHA DHE 1024 bits
Accepted TLSv1.0 112 bits DES-CBC3-SHA
Preferred SSLv3 256 bits DHE-RSA-AES256-SHA DHE 1024 bits
Accepted SSLv3 256 bits AES256-SHA
Accepted SSLv3 128 bits DHE-RSA-AES128-SHA DHE 1024 bits
Accepted SSLv3 128 bits AES128-SHA
Accepted SSLv3 128 bits RC4-SHA
Accepted SSLv3 128 bits RC4-MD5
Accepted SSLv3 112 bits EDH-RSA-DES-CBC3-SHA DHE 1024 bits
Accepted SSLv3 112 bits DES-CBC3-SHA

SSL Certificate:
Signature Algorithm: sha1WithRSAEncryption
RSA Key Strength: 1024
```

SSL Scan sample output (SSL Scan, 2023)

Nessus

Nessus is a great tool for testing API. Following are important characteristics of Nessus:

- Logging in to Nessus
- There are four navigation tabs at the top: Reports, Scans, Policies, and users.
- Choose the Report tab. The results of any scans you have ran, are running, or have imported will be displayed.
- The Scans tab list currently running scans, scan templates and scheduled scans.
- The Scans tab lists planned scans, scan templates, and scans that are presently executing.
- The scan configurations offered for scans are listed on the Policy tab.
- Users can be added, removed, or edited from the Users tab's list of users.

Nessus can be run in following two steps:

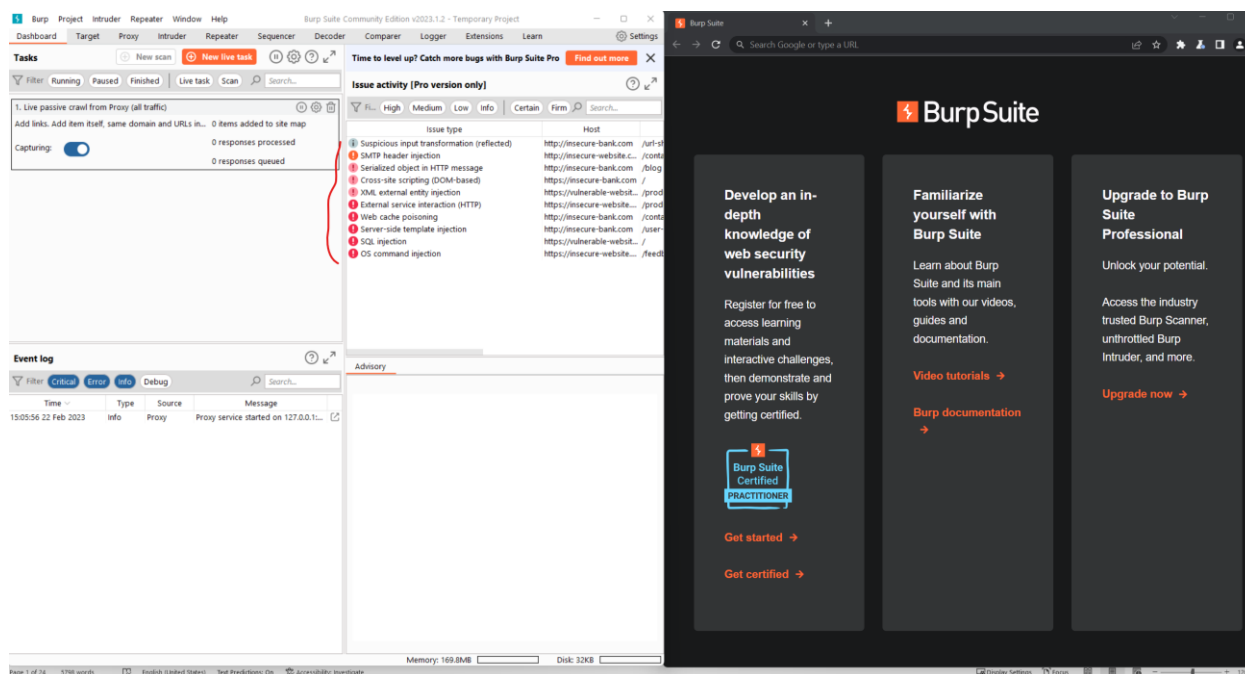
Step 1: Under the Scans Tab, choose your Basic Scan Template.

Step 2: Click the Start Button. If the template has been properly launched, a progress bar and a "running copy" of your scan should be displayed in the list.

Burp Suite

Burp suite is an immensely powerful tool to test SOAP API's security. It requires the client to use Burp as proxy and continue testing it in a normal way. The first step for testing the API is to map the attack surface. The burp suite can also be used to intercept the response and identify information.

When used as proxy, Burp suite can find common vulnerabilities like SQL Injection, Cross-site request forgery (CSRF). Burp Scanner's crawl engine can cut through obstacles like CSRF token, stateful functionality, XML external entity injection, server-side request forgery and overloaded or volatile URLs.



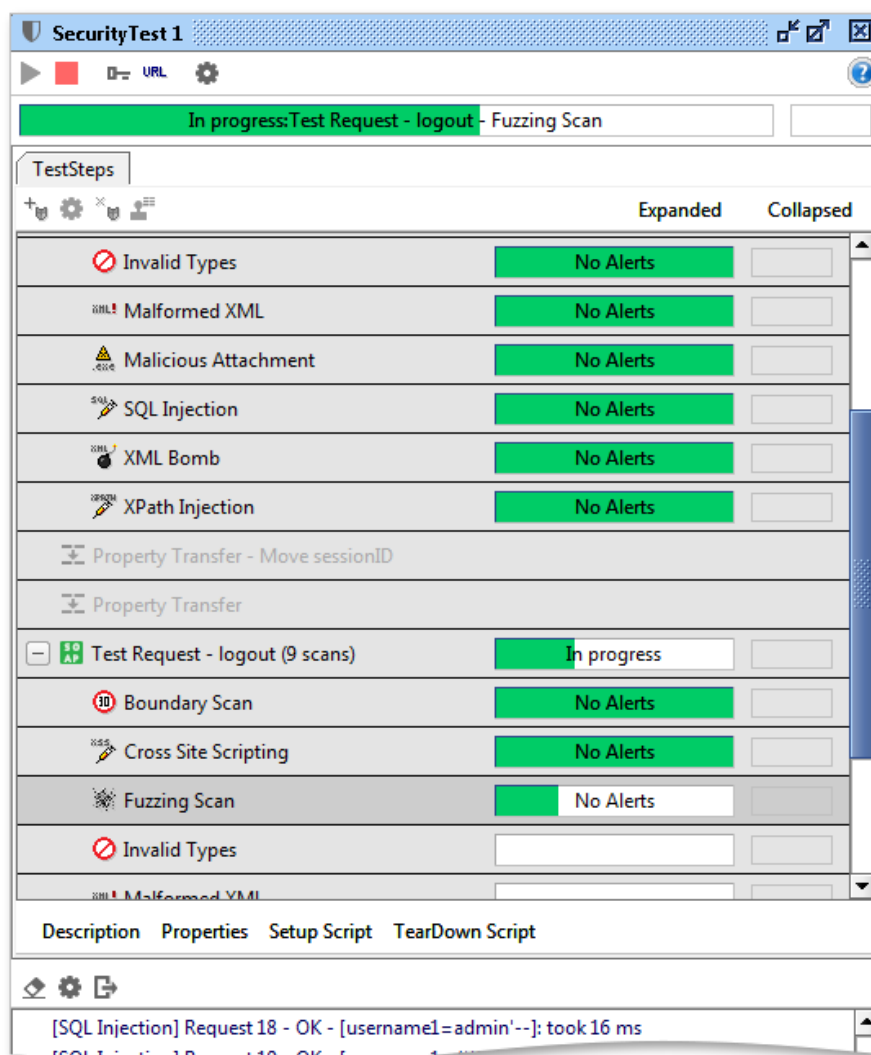
Burp Suite Dashboard

SOAP UI PRO

SOAP UI is an open-source tool for running security tests on SOAP APIs. It includes features to carry out inspection, invoking, development, mocking, functional testing, load and compliance testing. Specially for SOAP APIs it has features like WSDL coverage, WSDL refactoring, composite projects, table inspector, schema inspector, XPath wizards, and API discovery. These features are even more relevant for testing our case study project OSCAR EMR.

Following vulnerabilities can be tested with SOAP UI:

- SQL Injection: It tries to exploit bad database integration coding
- XPath Injection: It tries to exploit bad XML processing inside your target service.
- Boundary Scan: tries to exploit bad handling of values that are outside of defined ranges.
- Invalid Types: It tries to exploit handling of invalid input data.
- Malformed XML: tries to exploit bad handling of invalid XML on your server or in your service.
- XML Bomb: tries to exploit bad handling of malicious XML request (be careful).
- Malicious Attachment: tries to exploit bad handling of attached files.
- Cross Site Scripting: tries to find cross-site scripting vulnerabilities.
- Custom Script: allows you to use a script for generating custom parameter fuzzing values.



Run a security test SOAP UI (SOAP UI, 2023)

Summary

In this unit OSCAR EMR was installed on a virtual machine. Details can be found in Appendix A. The major vulnerabilities were identified who are necessary to declare an API secure. The tools identified in were installed and executed in the Methodology as documented above.

Results

As elaborated and identified the previous units we proposed the methodologies of doing secure API testing. In this unit, we carried out API security testing and took OSCAR EMR API as a case study. We have listed the criteria which should be met to declare an API secure. We have listed each vulnerability and tools we used to test this vulnerability. The last section of this unit includes a summary of test reports which we acquired from different tools. Detailed reports are provided in Appendix B. We unfortunately did not get access to a business environment installation of Oscar, we had to conduct our testing on virtual machine. It meant some of the infrastructure testing might not be as accurate as we wanted it to be. That meant use of tools like SSL Scan and Metasploitable made no sense. It is also pertinent to mention that the OSCAR installation we had and tools we used, covered most of the attack surface of OSCAR EMR.

Security testing of an API (Application Programming Interface) is extremely important. APIs are increasingly being used by organizations to enable communication between different applications, systems, and devices. However, if an API is not properly secured, it can pose a serious security risk, potentially leading to data breaches, information leakage, and other security vulnerabilities. APIs are a potential target for cybercriminals, who may try to exploit vulnerabilities to steal sensitive data, launch attacks, or cause disruption to services. Security testing helps identify and mitigate these threats. Organizations must comply with various regulatory standards such as GDPR, HIPAA, and PCI-DSS. Security testing helps ensure that the API meets these standards and is compliant with relevant regulations. A data breach or security incident can result in significant financial losses, reputational damage, and legal liabilities. Security testing helps prevent such losses by identifying and addressing vulnerabilities before they can be exploited by attackers. Customers expect their data to be secure when interacting with an organization's services or applications. Security testing helps demonstrate that an organization takes security seriously, which can enhance customer trust and confidence. This is particularly important for OSCAR EMR. As it is used by health care professional and contains health related data of Canadian residents.

Vulnerabilities Identified

As described in previous section, following vulnerabilities were identified as most critical for security testing of API:

- Broken Object Level Authorization
- Broken User Authentication
- Excessive Data Exposure
- Lack of Resource & Rate Limiting
- Broken Function Level Authorization
- Mass Assignment
- Security Misconfiguration
- Injection
- Improper Assets Management
- Insufficient Logging and Monitoring
- SQL Injection Scan
- XPath Injection Scan
- Malformed XML

Broken Object Level Authorization (BOLA)

It is a security vulnerability that occurs when an API does not properly enforce access controls on individual objects or resources. This can allow an attacker to access resources they are not authorized to access, potentially leading to data breaches, information leakage, and other security issues.

Following steps were followed for testing BOLA:

Identify the resources and objects: Start by identifying the resources and objects that are accessible via the API. These may include user accounts, documents, files, or other data.

Test access controls: Test whether the API enforces access controls on individual resources and objects. This may involve attempting to access resources using different authorization levels and verifying that access is appropriately restricted.

Test resource enumeration: Attempt to enumerate all available resources and objects by using different request parameters or by brute-forcing resource identifiers. Verify that the API only returns resources that are authorized for the user making the request.

Test object manipulation: Attempt to manipulate objects or resources by modifying their identifiers or request parameters. Verify that the API enforces proper access controls and prevents unauthorized access or modification.

Test privilege escalation: Attempt to escalate privileges by manipulating request parameters or object identifiers to gain access to resources that are normally restricted to higher-level users or roles.

Test error handling: Test the API's error handling capabilities to determine if error messages leak information about unauthorized resources or objects.

Perform penetration testing: Conduct penetration testing to simulate real-world attacks and identify potential security vulnerabilities related to BOLA.

We used Burp Suite to test OSCAR EMR for BOLA. Details can be found below in Appendix B.

Broken User Authentication

This vulnerability occurs when an API fails to properly authenticate users, which can lead to unauthorized access to sensitive data or functionality.

Here are steps followed to test for Broken User Authentication:

Identify authentication mechanisms: Identify the authentication mechanisms used by the API, such as username and password, token-based authentication, or other forms of authentication.

Test for weak passwords: Attempt to use weak passwords or easily guessable credentials to authenticate with the API. Verify that the API enforces password complexity requirements and prevents the use of easily guessable passwords.

Test for credential stuffing: Attempt to use a list of known usernames and passwords to authenticate with the API. Verify that the API enforces rate limiting and other mechanisms to prevent credential-stuffing attacks.

Test session management: Test the API's session management mechanisms, such as cookie or token management. Verify that sessions are properly invalidated and that session cookies are not vulnerable to session hijacking attacks.

Test logout functionality: Verify that the API properly logs out users when requested and that session tokens are invalidated after logout.

Test password reset functionality: Test the API's password reset functionality to ensure that it properly authenticates users and prevents unauthorized password resets.

Test for authentication bypass: Attempt to bypass authentication by manipulating request parameters or other mechanisms. Verify that the API properly enforces authentication and prevents unauthorized access to sensitive data or functionality.

We used Zap Proxy, Burp suite and hydra to test this vulnerability. **Zap Proxy** included a range of features for testing authentication mechanisms and was used to simulate attacks and identify potential vulnerabilities. **Burp Suite** was used to test for Broken User Authentication vulnerabilities. It included a range of features for identifying and exploiting vulnerabilities, including authentication testing and session management. **Hydra** is a password cracking tool that can be used to test for weak passwords and credential stuffing attacks. It was used to automate the process of attempting to authenticate with an API using a list of known credentials. Appendix B includes details.

Sensitive Data Exposure

Sensitive Data Exposure is a security vulnerability that can occur when an API fails to properly protect sensitive data, such as passwords, financial data, or personal information. Here are steps we followed to test for sensitive data exposure:

Identify sensitive data: Identify the types of sensitive data that are transmitted or stored by the API, such as passwords, financial data, or personal information.

Test data transmission: Test the API's data transmission mechanisms, such as HTTPS, to ensure that sensitive data is properly encrypted during transmission.

Test data storage: Test the API's data storage mechanisms, such as databases or file systems, to ensure that sensitive data is properly encrypted and protected from unauthorized access.

Test data leakage: Test the API for potential data leakage vulnerabilities, such as error messages that contain sensitive data or debug information that reveals sensitive data.

Test for secure password storage: Verify that the API properly stores passwords using secure password storage mechanisms, such as bcrypt or PBKDF2.

Test for proper authentication and authorization: Test the API's authentication and authorization mechanisms to ensure that only authorized users can access sensitive data.

Test for security misconfigurations: Test the API for potential security misconfigurations, such as default passwords or open ports, that could lead to sensitive data exposure.

We used Zap Proxy, Burp suit and Nmap to test this vulnerability. Details can be found in Appendix B.

Lack of Resource & Rate Limiting

Lack of Resource & Rate Limiting is a security vulnerability that can occur when an API fails to properly limit the number of resources that can be requested by a user or fails to limit the number of requests that can be made within a given period of time. Following steps were followed to test for Lack of Resource & Rate Limiting:

Identify API endpoints: Identify the API endpoints that are used to access resources or perform actions.

Test resource limits: Test the API's resource limits by attempting to request more resources than allowed by the API's rate limiting mechanism. Verify that the API properly limits the number of resources that can be requested and returns appropriate error messages when resource limits are exceeded.

Test rate limiting: Test the API's rate limiting mechanism by sending many requests within a short period of time. Verify that the API properly limits the number of requests that can be made within a given period and returns appropriate error messages when rate limits are exceeded.

Test for authentication bypass: Test the API for potential authentication bypass vulnerabilities that could be used to circumvent rate limiting or resource limits.

Test for denial-of-service attacks: Test the API for potential denial-of-service vulnerabilities that could be used to overload the API's resources and bypass rate limiting or resource limits.

Test for client-side caching: Test the API's client-side caching mechanisms to ensure that cached resources are properly invalidated and that clients cannot circumvent rate limiting or resource limits by caching resources.

Zap Proxy and Burp suite were used to test this vulnerability in OSCAR EMR. Details can be found in Appendix B.

Broken Function Level Authorization

Broken Function Level Authorization is a security vulnerability that can occur when an API does not properly enforce access controls on specific functions or operations, allowing unauthorized access to restricted functionality. Steps followed to test for Broken Function Level Authorization:

Identify privileged functions: Identify the functions or operations within the API that require privileged access and should only be available to authorized users.

Test function access: Test the API's access controls by attempting to access privileged functions or operations without the required authorization. Verify that the API properly enforces access controls and returns appropriate error messages when access is denied.

Test for authentication bypass: Test the API for potential authentication bypass vulnerabilities that could be used to circumvent access controls and gain unauthorized access to privileged functionality.

Test for authorization bypass: Test the API for potential authorization bypass vulnerabilities that could be used to circumvent access controls and gain unauthorized access to privileged functionality.

Test for privilege escalation: Test the API for potential privilege escalation vulnerabilities that could be used to elevate the privileges of an authorized user and gain unauthorized access to privileged functionality.

Test for business logic vulnerabilities: Test the API for potential business logic vulnerabilities that could be used to exploit privileged functionality and gain unauthorized access to sensitive data or resources.

Here are some tools we used to test for Broken Function Level Authorization, the details are given in **Appendix B:**

OWASP ZAP: OWASP ZAP (Zed Attack Proxy) is a free and open-source web application security testing tool that can be used to test for Broken Function Level Authorization vulnerabilities. It includes a range of features for identifying and exploiting access control vulnerabilities, including testing function access, and identifying potential authentication and authorization bypass vulnerabilities.

Burp Suite: Burp Suite is another popular web application security testing tool that can be used to test for Broken Function Level Authorization vulnerabilities. It includes a range of features for identifying and exploiting access control vulnerabilities, including testing function access, and identifying potential authentication and authorization bypass vulnerabilities.

Nessus: Nessus is a vulnerability scanner that can be used to identify potential access control vulnerabilities within an API. It includes a range of plugins that can be used to test for Broken Function Level Authorization vulnerabilities and identify potential privilege escalation vulnerabilities.

Nmap: Nmap is a free and open-source port scanner that can be used to identify potential access control vulnerabilities within an API. It includes a range of features for identifying open ports and testing network access controls, which can be used to identify potential access control vulnerabilities within an API.

Mass Assignment

This vulnerability occurs when an API allows users to submit more data than necessary in a request, allowing them to modify fields they should not have access to. Steps followed to test the mass assignment vulnerabilities:

Identify sensitive data: Identify the data fields that should be protected and not allowed to be modified by unauthorized users.

Test data modification: Test the API by submitting a request with additional data fields and values to see if any sensitive data can be modified.

Test data creation: Test the API by submitting a request to create a new object and including additional fields in the request to see if sensitive data can be added to the object.

Test data deletion: Test the API by submitting a request to delete an object and including additional fields in the request to see if sensitive data can be deleted along with the object.

Test nested objects: Test the API by submitting a request with nested objects and verify that the API properly restricts access to sensitive data within the nested objects.

Test PUT and PATCH requests: Test the API by submitting PUT and PATCH requests and verify that the API properly restricts access to sensitive data fields that should not be modified.

We used Zap Proxy and Burp Suite to test for this vulnerability. The details are included in Appendix B.

Security Misconfiguration

Security misconfiguration is a common vulnerability that occurs when an API or application is not properly configured, leading to potential security issues. Steps followed to test for security misconfigurations:

Review configuration files: Review configuration files for the API or application and ensure that all settings are set securely. This includes settings related to authentication, access controls, encryption, and logging.

Check for default passwords: Check for default or weak passwords in configuration files, database tables, and other places where passwords might be stored.

Test error handling: Test the API by submitting requests with incorrect parameters or malformed input to see how the API handles errors. Improper error handling can provide attackers with useful information that can be used to exploit the system.

Verify SSL/TLS configuration: Verify that SSL/TLS is properly configured and that the appropriate certificates are installed.

Check for unnecessary services and ports: Check for unnecessary services and ports that are open on the server, as they may provide attackers with a means to gain access to the system.

Check for outdated software and components: Check for outdated software and components, as they may contain known vulnerabilities that can be exploited by attackers.

We used Zap Proxy, Nessus, and Burp Suite to test for this vulnerability. The details are included in Appendix B.

SQL Injection

SQL injection is a common vulnerability that occurs when user input is not properly sanitized or validated, allowing attackers to inject malicious SQL code into the application's database. Following steps were followed to test for SQL Injection

Identify user input: Identify any user input fields in the application, including search bars, login forms, and other input fields.

Inject malicious SQL code: Attempt to inject malicious SQL code into these fields, such as the use of the "OR" operator or a comment marker to bypass authentication. For example, in a login form, try entering " ' OR 1=1 -- " in the username or password field.

Check for errors or unexpected behavior: If the application is vulnerable to SQL injection, you may see error messages or unexpected behavior, such as being able to bypass authentication or access unauthorized data.

Try other injection techniques: Try other injection techniques, such as using a UNION operator to combine two SELECT statements or using a sleep function to delay the response from the server.

ZAP Proxy, Nmap and Burp suite was used to test SQL Injection. Details are included in Appendix B.

Improper Assets Management

Improper asset management is a security vulnerability that occurs when an application does not properly manage its resources or assets, such as files, databases, or network connections. Here are some steps followed to test for improper asset management:

Identify assets: Identify all the assets or resources that the application uses, including files, databases, network connections, and other resources.

Check for access controls: Check that access controls are in place to restrict access to these resources. This includes ensuring that authentication and authorization mechanisms are implemented correctly and that users only have access to the resources they need.

Check for encryption: Check that sensitive data is properly encrypted, both in storage and in transit.

Check for data leakage: Check that the application does not leak sensitive data or resources to unauthorized users.

Check for cleanup: Check that the application cleans up resources properly after they are no longer needed.

Insufficient Logging and Monitoring

Insufficient logging and monitoring is a security vulnerability that occurs when an application does not properly log and monitor events and activities, making it difficult to detect and respond to security incidents. Here are steps followed to test for insufficient logging and monitoring:

Identify critical events: Identify critical events and activities that need to be logged and monitored, such as authentication attempts, data access, and administrative actions.

Check for logging: Check that the application logs events and activities correctly, including timestamps, user IDs, and other relevant data.

Check for monitoring: Check that the application is actively monitoring the logs for suspicious activities, such as failed login attempts or unusual data access patterns.

Check for alerting: Check that the application can alert administrators or security personnel when suspicious activities are detected.

Test incident response: Test the application's incident response procedures to ensure that they are effective in responding to security incidents.

Due to lack of proper infrastructure, we were not able to test this vulnerability.

XPath Injection

XPath Injection is a type of injection attack that occurs when an attacker manipulates an application's XPath query to gain unauthorized access to sensitive information or execute malicious code. Steps followed to test for XPath Injection:

Identify XPath queries: Identify the XPath queries used in the application, including those used for authentication, data access, and other critical functions.

Modify queries: Attempt to modify the XPath queries used in the application to see if it is possible to inject malicious code or gain unauthorized access to sensitive information.

Test for error messages: Test for error messages that may reveal sensitive information about the application's XPath queries, such as XPath syntax errors or error messages that reveal the structure of the XPath query.

Test for input validation: Test the application's input validation to see if it is possible to bypass or manipulate input validation checks and inject malicious code into the XPath query.

Tools used to test XPath Injection were Burp Suite and Zap Proxy. The details can be found in Appendix B.

Malformed XML

Malformed XML is a type of vulnerability that occurs when an application fails to properly handle or validate XML input, which can lead to a range of security issues, including denial of service attacks, data theft, and code execution. Steps followed to test for Malformed XML:

Identify input sources: Identify the input sources in the application that accept XML input, such as forms, APIs, and database queries.

Input malformed XML: Input malformed XML data into the application to see if it can handle or validate it correctly. Examples of malformed XML input include missing or mismatched tags, invalid character encoding, and overly long or complex XML data.

Test for error messages: Test for error messages that may reveal information about the application's handling of malformed XML input, such as XML syntax errors or error messages that reveal the structure of the XML input.

Test for code injection: Test if it is possible to inject malicious code into the application using malformed XML input.

We used Burp suite to test for this vulnerability. Details can be found in **Appendix B**.

Summary of Test results

We used different tools to perform security testing on OSCAR EMAR installation. Tools and summary of test results obtained from them are described below:

Zap Proxy

It is a free and open-source web application security testing tool that can be used to test vulnerabilities. It includes a range of features for identifying and exploiting vulnerabilities related to assets and resources, including automated testing and manual testing tools. Please find below a summary of test results obtained from Zap Proxy. It did not provide any High Alerts, however, four medium, five low and three informational alerts were found. Details are described in Appendix B. The Brief table is described below:

| Risk Level | Number of Alerts |
|---------------|------------------|
| High | 0 |
| Medium | 4 |
| Low | 5 |
| Informational | 3 |

Issues found

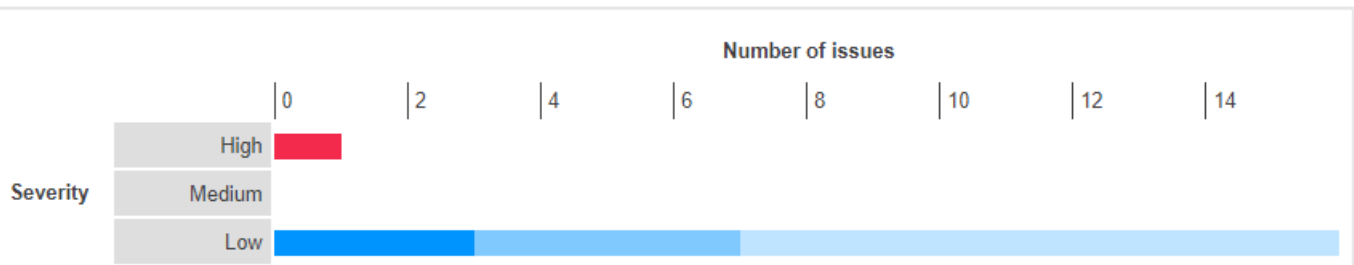
| Name | Risk Level | Number of Instances |
|--------------------------------------------------------------|---------------|---------------------|
| Absence of Anti-CSRF Tokens | Medium | 2 |
| Content Security Policy (CSP) Header Not Set | Medium | 4 |
| Missing Anti-clickjacking Header | Medium | 1 |
| Session ID in URL Rewrite | Medium | 2 |
| Cookie No HttpOnly Flag | Low | 1 |
| Cookie Without Secure Flag | Low | 1 |
| Cookie without SameSite Attribute | Low | 2 |
| Strict-Transport-Security Header Not Set | Low | 8 |
| X-Content-Type-Options Header Missing | Low | 5 |
| Information Disclosure - Suspicious Comments | Informational | 2 |
| Loosely Scoped Cookie | Informational | 2 |
| User Agent Fuzzer | Informational | 12 |

Burp Suite

Burp Suite is a web application security testing tool that includes features for testing vulnerabilities. It includes a range of tools for identifying vulnerabilities, including a vulnerability scanner and a proxy server for intercepting and modifying requests. The table below shows the numbers of issues identified in different categories. Issues are classified according to severity as High, Medium, Low, Information or False Positive. This reflects the likely impact of each issue for a typical organization. Issues are also classified according to confidence as Certain, Firm or Tentative. This reflects the inherent reliability of the technique that was used to identify the issue.

| | | Confidence | | | |
|----------|----------------|------------|------|-----------|-------|
| | | Certain | Firm | Tentative | Total |
| Severity | High | 1 | 0 | 0 | 1 |
| | Medium | 0 | 0 | 0 | 0 |
| | Low | 3 | 4 | 9 | 16 |
| | Information | 4 | 2 | 0 | 6 |
| | False Positive | 0 | 0 | 0 | 0 |

The chart below shows the aggregated numbers of issues identified in each category. Solid colored bars represent issues with a confidence level of Certain, and the bars fade as the confidence level falls.



Issues Found

| Name | Risk Level | Issues found |
|----------------------------------|------------|--------------|
| Cleartext submission of password | High | 1 |
| Vulnerable JavaScript dependency | Low | 9 |
| Content type incorrectly stated | Low | 4 |
| Unencrypted communications | Low | 2 |

Nessus

Nessus is a popular vulnerability scanner that helps to identify and assess security vulnerabilities in computer systems, networks, and applications. It is developed and maintained by Tenable, Inc. Nessus can perform various security assessments, including host discovery, port scanning, service identification, configuration auditing, and vulnerability scanning. It uses a database of known vulnerabilities and exploits to test for security weaknesses in target systems. Nessus can be used by security professionals, network administrators, and penetration testers to identify security vulnerabilities in their systems and prioritize remediation efforts. It is available as both a commercial and open-source product, with different levels of features and capabilities.

We ran a Nessus scan on OSCAR EMR. It found four high, 3 medium and 1 low vulnerability.



Vulnerabilities assessed.

Issues Found

| Name | Risk Level |
|------------------------------------------------------------------------------------|------------|
| Apache Tomcat 9.0.0.M1 < 9.0.71 | HIGH |
| Apache Tomcat 9.0.13 < 9.0.63 vulnerability | HIGH |
| Apache Tomcat 9.0.40 < 9.0.69 | HIGH |
| Apache Tomcat 9.0.30 < 9.0.65 vulnerability | Medium |
| Apache Tomcat Default Files | Medium |
| Apache Tomcat 9.0.0.M1 < 9.0.72 | Medium |
| Apache Tomcat 9.0.0.M1 < 9.0.62 Spring4Shell (CVE-2022-22965) Mitigations | Low |
| Apache Tomcat Detection | INFO |
| Apache Tomcat 9.0.0.M1 < 9.0.71 | INFO |
| Apache Tomcat Detection | INFO |
| HSTS Missing From HTTPS Server | INFO |
| HTTP Methods Allowed (per directory) | INFO |
| HyperText Transfer Protocol (HTTP) Information | INFO |
| Missing or Permissive Content-Security-Policy frame-ancestors HTTP Response Header | INFO |
| Missing or Permissive X-Frame-Options HTTP Response Header | INFO |
| Nessus SYN scanner | INFO |
| Nessus Scan Information | INFO |
| Patch Report | INFO |
| Web Application Sitemap | INFO |

Hydra

Hydra is a popular password-cracking tool that is used to perform brute-force attacks against remote systems or web applications. It is a command-line tool that supports a wide range of protocols, including HTTP, HTTPS, FTP, SSH, Telnet, and many others. Hydra is designed to test the strength of passwords used to secure remote services, by trying out a large number of possible password combinations. It works by attempting to log in to a target system or application using a list of usernames and passwords and can be configured to use various techniques to speed up the process, such as parallel requests and optimized network communications.

Hydra is often used by security professionals to test the security of their systems and applications, as well as by attackers attempting to gain unauthorized access to systems or steal sensitive data. It is important to note that the use of Hydra or any other password-cracking tool on systems or applications without explicit permission is illegal and can result in serious consequences.

We ran hydra on OSCAR EMR. The results are described in Appendix B.

Nmap

Nmap (Network Mapper) is a powerful open-source tool used for network exploration, management, and security auditing. It is a command-line utility that can be used to scan networks, identify hosts and services, and detect security vulnerabilities. Nmap is commonly used by network administrators, security professionals, and hackers to gather information about networked devices, including operating systems, open ports, and network services. The tool can also be used to identify potential security weaknesses and misconfigurations that may exist within a network or on individual hosts. Nmap supports a variety of scanning techniques, including ping scanning, TCP scanning, UDP scanning, and OS detection. It can also be used to perform advanced scans, such as version detection, traceroute analysis, and network mapping. Additionally, Nmap can be extended with custom scripts, allowing for

advanced network analysis and automation. While Nmap is a powerful tool, it should only be used on networks or systems that you own or have explicit permission to test. Using Nmap on unauthorized systems or networks without permission can result in serious legal and ethical consequences. We ran Nmap on OSCAR EMR. The results are described in Appendix B.

Summary

In this unit we described the criteria and process with the help of which we declare an API Secure or insecure. We tested OSCAR EMR based on these criteria using various tools. The output of test results is attached below. We found in total **5 high, 7 medium, 9 low** and **17 info vulnerabilities**. In the next section we shall describe upon fulfilling which criteria we declare an API secure or insecure. We will also draw conclusions about OSCAR EMR's API.

Conclusion

In Results, we provided details related to API security testing and explained how different kinds of security testing should be performed and the tools which could be used to test those vulnerabilities. In this unit we include findings, attack scenarios and overall assessment of the API's security posture. It includes a summary of vulnerabilities, their potential impact, and recommendations for remediation. It is important to note that a penetration test may identify few vulnerabilities, it does not necessarily mean that the API is completely insecure. The goal of the report is to provide actionable insights and recommendations to improve the security of the API, and to help the organization better understand and manage their security risks. Ultimately, the success of a REST API depends on the organization's willingness to implement the recommended security controls and to continually monitor and assess their API's security posture. We would analyze results obtained in earlier unit. We will carry on with the vulnerabilities identified earlier. At the end of each vulnerability, we will provide information about that vulnerability's presence in OSCAR EMR. At the end we will give our recommendations about the status of security of API of OSCAR EMR (our case study project).

Vulnerabilities

Broken Object Level Authorization (BOLA)

It is important to conduct careful testing of all resources for BOLA vulnerability, as a single privilege escalation by an attacker can result in a critical vulnerability that can render the entire API insecure, potentially granting access to sensitive data or functionality that the attacker should not be able to access (OWASP, 2021).

There are several possible scenarios where BOLA vulnerabilities may occur. In the case of Insecure Direct Object Reference, an attacker can pass a URL or parameter to access objects they should not have permission to access, such as accessing the details of a different account in a banking application (OWASP, 2021).

Predictable Object References can occur when applications use predictable object references like sequential numbers or names, allowing attackers to guess valid object references and access objects they should not have permission to access, like guessing file names in a file-sharing application (OWASP, 2021). Insufficient Authorization Checks occur when an application performs authorization checks at the wrong level, allowing attackers to access objects they should not have permission to access, such as not performing additional checks to ensure that a user can access specific data on a page (OWASP, 2021).

In some cases, Privilege Escalation may occur where an attacker can upload a file with elevated permissions and access it to gain additional privileges if an application allows file uploads but fails to enforce object-level authorization rules properly (OWASP, 2021).

Cross-Site Request Forgery (CSRF) can occur when an attacker tricks a user into clicking a malicious link or visiting a malicious website, allowing them to perform actions on behalf of the user if the application fails to enforce object-level authorization rules (OWASP, 2021) properly. For example, an attacker can trick a user into clicking a link that performs a transfer of funds from their account to the attacker's account.

We used Burpsuite and Zaproxy to test BOLA in OSCAR EMR. We identified all resources from wadl available at

<OscarHost/oscar/ws/rs? wadl>

The Web Application Description Language (WADL) is a machine-readable XML description of HTTP-based web services. Extract of OSCAR Wadl is shown below:

```
<?xml version='1.0' encoding='UTF-8'>
<wadl:application xmlns:wadl="http://www.w3.org/2005/01/wadl" xmlns:xs="http://www.w3.org/2001/XMLSchema" attributeFormDefault="unqualified" elementFormDefault="unqualified">
  <wadl:grammar>
    <xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" attributeFormDefault="unqualified" elementFormDefault="unqualified">
      <xs:element name="List" type="oscarSearchResponse"/>
      <xs:element name="OscarJobType" type="oscarJobTypeTol"/>
      <xs:element name="ProductLocation" type="productLocationTol"/>
      <xs:element name="addressTol" type="addressTol"/>
      <xs:element name="allergyResponse" type="allergyResponse"/>
      <xs:element name="app" type="appDefinitionTol"/>
      <xs:element name="appointmentTextResponse" type="appointmentTextResponse"/>
      <xs:element name="appointmentStatusTol" type="appointmentStatusTol"/>
      <xs:element name="consultationRequestTol" type="consultationRequestTol"/>
      <xs:element name="consultationResponseTol" type="consultationResponseTol"/>
      <xs:element name="consultationServiceTol" type="consultationServiceTol"/>
      <xs:element name="demographicHeaderTol" type="demographicHeaderTol"/>
      <xs:element name="demographicResponse" type="demographicResponse"/>
      <xs:element name="diagnosisTol" type="diagnosisTol"/>
      <xs:element name="drug" nillable="true" type="drugTol"/>
      <xs:element name="drugLookupResponse" type="drugLookupResponse"/>
      <xs:element name="drugProduct" type="drugProductTol"/>
      <xs:element name="drugProductResponse" type="drugProductResponse"/>
      <xs:element name="drugProductTemplateResponse" type="drugProductTemplateResponse"/>
      <xs:element name="drugResponse" type="drugResponse"/>
      <xs:element name="drugSearchResponse" type="drugSearchResponse"/>
      <xs:element name="encounterNote" type="noteTol"/>
      <xs:element name="encounterTemplateResponse" type="encounterTemplateResponse"/>
      <xs:element name="favorite" type="favoriteTol"/>
      <xs:element name="favoriteResponse" type="favoriteResponse"/>
      <xs:element name="formlist" type="formlistTol"/>
      <xs:element name="genericRESTResponse" type="genericRESTResponse"/>
      <xs:element name="groupNoteExt" type="noteExtTol"/>
      <xs:element name="inbox" type="inboxTol"/>
      <xs:element name="inboxResponse" type="inboxResponse"/>
      <xs:element name="issue" type="issueTol"/>
      <xs:element name="item" type="menuItemTol"/>
      <xs:element name="labResponse" type="labResponse"/>
      <xs:element name="letterheadTol" type="letterheadTol"/>
      <xs:element name="measurementResponse" type="measurementResponse"/>
      <xs:element name="menu" type="menuItemTol"/>
      <xs:element name="message" type="messageTol"/>
      <xs:element name="messageInResponse" type="messagingResponse"/>
      <xs:element name="navBarMenu" type="navBarMenuTol"/>
      <xs:element name="navBarResponse" type="navBarResponse"/>
      <xs:element name="noteIssue" type="noteIssueTol"/>
      <xs:element name="noteSelection" type="noteSelectionTol"/>
      <xs:element name="notification" type="notificationTol"/>
      <xs:element name="oscarJobResponse" type="oscarJobResponse"/>
      <xs:element name="oscarJobTypeResponse" type="oscarJobTypeResponse"/>
      <xs:element name="patientDetailStatusTol" type="patientDetailStatusTol"/>
      <xs:element name="patientListApptBean">
        <xs:complexType>
          <xs:sequence>
            <xs:element minOccurs="0" name="template" type="xs:string"/>
            <xs:element maxOccurs="unbounded" minOccurs="0" name="patients" nillable="true" type="patientListApptItemBean"/>
          </xs:sequence>
        </xs:complexType>
      </xs:element>
      <xs:element name="patientListApptItemBean" type="patientListApptItemBean"/>
      <xs:element name="personResponse" type="personResponse"/>
      <xs:element name="personRightResponse" type="personRightResponse"/>
      <xs:element name="post" type="rssItem"/>
      <xs:element name="prescription" type="prescriptionTol"/>
      <xs:element name="prescriptionResponse" type="prescriptionResponse"/>
      <xs:element name="preventiveResponse" type="preventiveResponse"/>
      <xs:element name="productLocationResponse" type="productLocationResponse"/>
      <xs:element name="programProviderTol" type="programProviderTol"/>
      <xs:element name="providerApptsCountResponse" type="providerApptsCountResponse"/>
      <xs:element name="providerPeriodAppResponse" type="providerPeriodAppResponse"/>
      <xs:element name="referralResponse" type="referralResponse"/>
      <xs:element name="response" type="response"/>
      <xs:element name="schedulingResponse" type="schedulingResponse"/>
      <xs:element name="securityObject" type="securityObjectTol"/>
      <xs:element name="statusValueTol" type="statusValueTol"/>
      <xs:element name="summary" type="summaryTol"/>
      <xs:element name="summaryItem" type="summaryItemTol"/>
      <xs:element name="tickler" type="ticklerTol"/>
      <xs:element name="ticklerComment" type="ticklerCommentTol"/>
      <xs:element name="ticklerLink" type="ticklerLinkTol"/>
      <xs:element name="ticklerNote" type="ticklerNoteTol"/>
      <xs:element name="ticklerNoteResponse" type="ticklerNoteResponse"/>
      <xs:element name="ticklerResponse" type="ticklerResponse"/>
      <xs:element name="ticklerUpdate" type="ticklerUpdateTol"/>
    </xs:schema>
  </wadl:grammar>
</wadl:application>
```

Wadl file available at OscarHost/oscar/ws/rs? wadl

As OSCAR uses Oauth 1.0 and we did not find this vulnerability. It is however recommended that for each rest service there should be the right role management system. The administrator should be able to decide to which methods he wants to give access to a particular user. Currently It is for all methods.

Manage Clients

| Name | Client Key | Client Secret | URI | Token TTL | Actions |
|-----------|------------------|------------------|----------------|-----------|---------|
| conestoga | gilphg3dbd7a1vry | de9oqxz2l0335d4y | localhost:8081 | 123 | ✗ |

Add New

Oscar keys generated.

Broken User Authentication

This can allow an attacker to impersonate a legitimate user or bypass authentication altogether. Scenario which should be looked for:

Weak Password Policies: In this scenario, an application may allow weak or easily guessable passwords. This can allow an attacker to easily guess a user's password and gain access to their account. For example, an application may allow passwords that are too short, do not require a combination of letters, numbers, and symbols, or do not have any complex requirements.

Password Reuse: In some cases, users may reuse the same password across multiple accounts. If one of these accounts is compromised, an attacker can use the same password to access the user's account on other sites or applications. This is especially dangerous if the user has used the same email address for multiple accounts.

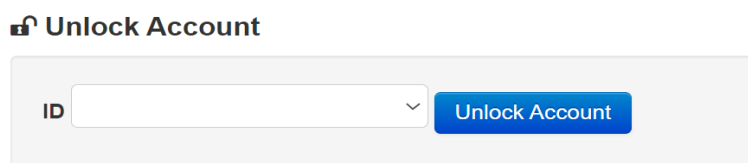
Brute Force Attacks: In a brute force attack, an attacker tries many possible passwords in order to guess the correct one. If an application does not properly limit the number of failed login attempts, an attacker can use this technique to gain access to a user's account. For example, an attacker may use a script or tool to automatically try many passwords until they find the correct one.

Session Hijacking: In a session hijacking attack, an attacker intercepts a user's session ID or authentication token and uses it to impersonate the user. This can occur if an application does not properly secure session IDs or if an attacker is able to obtain the session ID through cross-site scripting (XSS) or other means.

Credential Stuffing: In this scenario, an attacker uses a list of known username and password combinations to attempt to gain access to an application. This can be successful if users have reused passwords across multiple accounts or if the application does not properly lock out users after a certain number of failed login attempts.

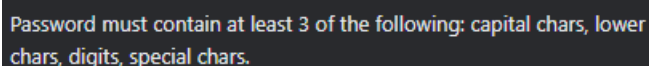
Insecure Password Recovery: If an application's password recovery mechanism is insecure, an attacker may be able to reset a user's password and gain access to their account. This can occur if the application allows password resets based on easily obtainable information, such as a user's email address or date of birth.

We used Hydra to test this vulnerability in OSCAR EMR. OSCAR EMR has the option of blocking IP addresses of users who enter repeated wrong passwords.



Account Locked of user who entered wrong password 5 times.

Oscar has the following Password security policy which we found adequate. A possible recommendation would be to have a minimum specified set of characters for a password required to be accepted by the system.



Oscar password policy

Oscar EMR uses JSP and uses SESSION_ID in URL rewrite as deduced by Zaproxy. We recommend fixing this vulnerability as soon as possible.

Excessive Data Exposure

Excessive Data Exposure is a security vulnerability that occurs when sensitive data is exposed to unauthorized parties due to insufficient security measures. Here are a few possible scenarios:

Unencrypted Data: If sensitive data is not properly encrypted, it can be easily intercepted and read by unauthorized parties. For example, if a web application stores passwords in plain text, an attacker who gains access to the application's database can easily obtain the passwords.

Weak Encryption: Even if data is encrypted, if the encryption algorithm is weak or the encryption keys are not properly protected, an attacker may be able to decrypt the data. For example, if an application uses a weak encryption algorithm such as MD5, an attacker may be able to easily decrypt the data using a brute-force attack.

Lack of Access Controls: If an application does not properly control access to sensitive data, unauthorized users may be able to view it. For example, if an application stores sensitive data in a publicly accessible directory on the web server, anyone who knows the URL of the file can access it.


Insecure APIs: If an application exposes sensitive data through an API, and the API does not properly authenticate and authorize requests, attackers may be able to access the data by sending unauthorized requests to the API.

Logging of Sensitive Data: If an application logs sensitive data, such as passwords or credit card numbers, in plaintext, anyone who has access to the logs can view the data. This can occur if an application developer accidentally logs sensitive data or if the application's logging mechanism is not properly configured.

Improperly Configured Databases: If a database is not properly configured, it may expose sensitive data to unauthorized users. For example, if a database is configured to allow remote access without proper authentication and authorization, attackers may be able to access sensitive data stored in the database.

It's important for applications to properly encrypt sensitive data, implement access controls to restrict access to sensitive data, and properly configure APIs and databases to prevent excessive data exposure. Most of the OSCAR EMR installations use TLS/SSL protocol which is adequate for most cases. Burp suite found one vulnerability as described in results "Cleartext submission of password". We recommend fixing it.

1- Cleartext submission of password

| | | |
|-------------------------------------------------------------------------------------|------------|------------------------|
|  | Severity | High |
| | Confidence | Certain |
| | Path | <i>oscar/index.jsp</i> |

Lack of Resource & Rate Limiting

Lack of Resource & Rate Limiting is a security vulnerability that occurs when an application does not properly limit the number of requests or resources that can be used by a user or an attacker. Here are a few possible scenarios:

API Abuse: If an application exposes an API, an attacker can use it to make a large number of requests to the application. If the API does not have proper rate limiting, an attacker can easily overwhelm the application's resources.

Denial of Service (DoS) Attacks: In a DoS attack, an attacker sends a large number of requests to an application with the goal of overwhelming its resources and making it unavailable to legitimate users.

If an application does not properly limit the number of requests that can be made by a user or an IP address, it can be easily overwhelmed by a DoS attack.

Scraping Attacks: In a scraping attack, an attacker uses an automated script or tool to scrape large amounts of data from an application. If an application does not properly limit the number of requests that can be made, an attacker can scrape large amounts of data in a short period of time, which can cause performance issues or even crash the application.

CPU and Memory Exhaustion: If an application does not properly limit the amount of CPU and memory that can be used by a user, an attacker can cause the application to crash by sending requests that use many resources.

As OSCAR EMR does not allow public registrations, we feel **CAPTCHAs are not required** but we recommend prevention against DoS attacks by using services like Cloudflare. It was however possible to write an automated script to scrap large amounts of data. We recommend limiting the number of requests a user can make.

Broken Function Level Authorization

Broken Function Level Authorization is a security vulnerability that occurs when an application does not properly restrict access to specific functions or actions based on the user's role or permissions. Here are a few possible scenarios:

Access to Admin Functions: If an application does not properly restrict access to admin functions, an attacker who gains access to a regular user account may be able to access admin functions and perform actions that should only be performed by an administrator. For example, an attacker may be able to change settings or delete data.

Access to User Data: If an application does not properly restrict access to user data, an attacker may be able to access sensitive user data that should only be accessible to the user or authorized personnel. For example, an attacker may be able to view a user's personal information or financial data.

Access to Payment Functions: If an application allows users to make payments or financial transactions, it's important to properly restrict access to these functions. If an attacker is able to access payment functions, they may be able to steal money or perform fraudulent transactions.

Access to API Functions: If an application exposes API functions, it's important to properly restrict access to these functions based on the user's role or permissions. If an attacker is able to access API functions that should only be accessible to authorized users, they may be able to steal data or perform malicious actions.

Access to Backup and Restore Functions: If an application allows backup and restore functions, it's important to properly restrict access to these functions. If an attacker can access backup or restore functions, they may be able to delete or modify data or perform other malicious actions.

Access to Debugging Functions: If an application allows debugging functions in production, it's important to properly restrict access to these functions. If an attacker can access debugging functions, they may be able to view sensitive data or modify the application's behavior.

It's important for applications to properly restrict access to specific functions or actions based on the user's role or permissions to prevent these types of vulnerabilities. This can be done through various techniques such as role-based access control (RBAC), attribute-based access control (ABAC), or implementing proper authentication and authorization mechanisms.

OSCAR EMR offers limited functionality via the API. We checked privilege elevation for different functions, but we did not find this problem. Sensitive functions like payment functions are not reachable via the API. We did not find this vulnerability in OSCAR EMR. We however recommend

implementation of a rights and role system for OSCAR where admin can control which functions can be accessed by an API user.

Mass Assignment

Mass Assignment is a security vulnerability that occurs when an application allows an attacker to modify more fields than intended by modifying a single input parameter. Here are a few possible scenarios:

User Profile Modification: An attacker may modify more fields than intended when modifying their user profile. For example, an attacker may be able to modify their user account to gain administrator access or modify sensitive data.

Data Leakage: An attacker may be able to modify more fields than intended when creating or modifying records in a database. For example, an attacker may be able to modify a user's account record to gain access to sensitive data or modify records that are not intended to be modified.

Unintended Privileges: An attacker may be able to modify more fields than intended when creating or modifying records that include role or permission data. For example, an attacker may be able to modify a user's role to gain additional privileges or modify permission data to gain access to restricted data.

Financial Transactions: An attacker may be able to modify more fields than intended when creating or modifying financial transactions. For example, an attacker may be able to modify the amount of a transaction to steal money or modify the recipient of a transaction to redirect funds.

Product Pricing: An attacker may be able to modify more fields than intended when creating or modifying product pricing data. For example, an attacker may be able to modify the price of a product to obtain a discount or change the pricing of multiple products to affect the overall pricing of an application.

It's important for applications to properly restrict the fields that can be modified by input parameters to prevent these types of vulnerabilities. This can be done through various techniques such as implementing proper validation and sanitization of input parameters, using attribute-based access control (ABAC), or only allowing specific fields to be modified by specific roles or permissions.

The above provided cases are a must for security of an API. In OSCAR this vulnerability was not present.

Security Misconfiguration

Security Misconfiguration is a security vulnerability that occurs when an application or system is not properly configured to securely handle data or prevent unauthorized access. Here are a few possible scenarios:

Default Passwords: An attacker may be able to gain access to a system or application by using default passwords that were not changed during the configuration process. This can happen if the system or application was not properly configured to enforce password changes or prevent the use of default passwords.

Server Misconfiguration: An attacker may be able to exploit vulnerabilities in server configurations to gain unauthorized access to the server or data. For example, if the server is not properly configured to handle HTTPS traffic, an attacker may be able to intercept sensitive data being transmitted over the network.

Improper Access Controls: An attacker may be able to bypass access controls if the application or system is not properly configured to enforce access control policies. For example, if a user is able to

access sensitive data without being authorized to do so, it may be due to a misconfiguration in the access control policies.

Incorrect File Permissions: An attacker may be able to access or modify sensitive data if the file permissions are not properly set. For example, if a file containing sensitive data is set to be publicly accessible, an attacker may be able to access the file without authorization.

Unnecessary Services: An attacker may be able to exploit vulnerabilities in services that are not necessary for the application or system to function. For example, if a system has unnecessary services enabled, an attacker may be able to use those services to gain unauthorized access.

Outdated Software: An attacker may be able to exploit vulnerabilities in outdated software that has not been properly configured with security updates or patches. For example, if a system is running outdated software that has known security vulnerabilities, an attacker may be able to exploit those vulnerabilities to gain unauthorized access.

OSCAR EMR uses a default password and access key for initial login. It uses an old version of tomcat. A network scan with Nessus revealed this information.

| | |
|---------------------------------------------|------|
| Apache Tomcat 9.0.0.M1 < 9.0.71 | HIGH |
| Apache Tomcat 9.0.13 < 9.0.63 vulnerability | HIGH |
| Apache Tomcat 9.0.40 < 9.0.69 | HIGH |

We recommend upgrading the tomcat version and generation of password in a file on server instead of always using the same password.

Injection

Injection is a type of security vulnerability that occurs when an attacker can inject malicious code or data into an application or system, which can lead to data loss, data corruption, or unauthorized access. Here are a few possible injection scenarios:

SQL Injection: An attacker may be able to inject SQL code into an application's input fields, such as login forms or search bars, to gain unauthorized access to data stored in a database. For example, an attacker could inject SQL code into a login form to bypass authentication and gain access to sensitive data.

Command Injection: An attacker may be able to inject malicious code into an application's input fields to execute arbitrary commands on the system. For example, an attacker could inject code into a search bar to execute system commands, such as deleting files or modifying system configurations.

Cross-site Scripting (XSS): An attacker may be able to inject malicious scripts into an application's input fields to steal data, such as cookies or session tokens, from other users. For example, an attacker could inject a script into a comment field on a website to steal a user's session token and gain unauthorized access to their account.

LDAP Injection: An attacker may be able to inject LDAP code into an application's input fields to gain unauthorized access to data stored in a directory service. For example, an attacker could inject LDAP code into a search field to bypass authentication and gain access to sensitive data.

XML Injection: An attacker may be able to inject malicious code into an application's XML input fields to modify or delete data in a system. For example, an attacker could inject XML code into a form field to delete a user's account or modify their permissions.

Email Injection: An attacker may be able to inject malicious code into an email message to bypass spam filters or execute arbitrary code on the recipient's system. For example, an attacker could inject code into an email message to steal data from the recipient's system or gain unauthorized access to their accounts.

In OSCAR EMR, we did find one vulnerability of this type.

| Medium | Absence of Anti-CSRF Tokens |
|-------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| | <p>No Anti-CSRF tokens were found in an HTML submission form.</p> <p>A cross-site request forgery is an attack that involves forcing a victim to send an HTTP request to a target destination without their knowledge or intent in order to perform an action as the victim. The underlying cause is application functionality using predictable URL /form actions in a repeatable way. The nature of the attack is that CSRF exploits the trust that a website has for a user. By contrast, cross-site scripting (XSS) exploits the trust that a user has in a website. Like XSS, CSRF attacks are not necessarily cross-site, but they can be. Cross-site request forgery is also known as CSRF, XSRF, one-click attack, session riding, confused deputy, and sea surf.</p> <p>CSRF attacks are effective in a number of situations, including:</p> <ul style="list-style-type: none">* The victim has an active session on the target site. |
| Description | |

We recommend fixing this.

Insufficient Logging and Monitoring

Insufficient logging and monitoring is a type of security vulnerability that occurs when an organization fails to properly monitor and log security events, making it difficult to detect and respond to security incidents. Here are a few possible serious scenarios:

Data Breach: An organization may fail to detect and respond to a data breach due to insufficient logging and monitoring. For example, an attacker may gain unauthorized access to sensitive data and exfiltrate it without being detected, resulting in a serious data breach.

Insider Threats: An organization may fail to detect and respond to insider threats due to insufficient logging and monitoring. For example, an employee may steal sensitive data or sabotage systems without being detected, resulting in a serious security incident.

Malware Infection: An organization may fail to detect and respond to malware infections due to insufficient logging and monitoring. For example, malware may be installed on a system and remain undetected, allowing attackers to steal data or use the system for malicious purposes.

Advanced Persistent Threats (APTs): An organization may fail to detect and respond to APTs due to insufficient logging and monitoring. For example, an attacker may gain access to a system and remain undetected for an extended period of time, allowing them to steal data or carry out other malicious activities.

Fraudulent Activities: An organization may fail to detect and respond to fraudulent activities due to insufficient logging and monitoring. For example, an employee may engage in financial fraud or other fraudulent activities without being detected, resulting in financial losses for the organization.

Compliance Violations: An organization may fail to detect and respond to compliance violations due to insufficient logging and monitoring. For example, an organization may fail to detect and respond to unauthorized access to personal data, resulting in a violation of data protection regulations such as GDPR or HIPAA.

As OSCAR EMR runs on tomcat, we found logging information in catalina.out. We did not find any evidence of log file monitoring. We recommend use of IDS (Intrusion Detection System) and proper monitoring of log files. We recommend use of proper alerting based on the information available log files.

Malformed XML

Malformed XML is a type of security vulnerability that occurs when an application does not properly validate user input in XML format, which can result in various types of attacks. Here are a few possible serious scenarios:

XML External Entity (XXE) Injection: An attacker may inject malicious XML code into an application, which can allow them to read sensitive files or execute arbitrary code on the server. For example, an attacker may include an external entity reference that points to a file containing sensitive data, which could be leaked to the attacker.

XML Injection: An attacker may inject malicious XML code into an application, which can allow them to modify or delete data on the server. For example, an attacker may modify an XML document in a way that changes the behavior of the application or deletes important data.

Denial-of-Service (DoS) Attack: An attacker may send malformed XML requests to an application, which can cause the server to crash or become unresponsive. For example, an attacker may send an XML request with an excessively large payload, causing the server to run out of memory.

Authentication Bypass: An attacker may use malformed XML requests to bypass authentication mechanisms in an application. For example, an attacker may modify an XML document to include a forged authentication token, allowing them to access restricted areas of the application.

Information Disclosure: An attacker may use malformed XML requests to obtain sensitive information about an application or its users. For example, an attacker may include an entity reference that reveals information about the server's configuration or the application's internal structure.

Cross-Site Scripting (XSS) Attack: An attacker may use malformed XML requests to inject malicious scripts into an application, which can allow them to steal sensitive information or perform other malicious activities. For example, an attacker may inject a script that steals user credentials or redirects the user to a malicious website.

We attempted attack scenarios above on OSCAR SOAP API. OSCAR does a good job protecting itself against attacks using XML injection.

Summary

In the above unit we drew conclusion that an API cannot be declared secure unless it can handle above-described attack scenarios. Please note we have only identified critical attack scenarios in this unit. Our security testing of OSCAR EMR API found no significant vulnerabilities in the tested system. Our testing team conducted a thorough analysis of the system and performed a variety of tests to identify any potential security weaknesses. We used both automated and manual testing techniques, including vulnerability scanning, port scanning, and password cracking. The team concluded that the system appeared to be well-secure and that there were no major issues that required immediate attention. The report recommended that the system be regularly monitored and tested to ensure that it remains secure over time. At the end of this section, we are giving a few recommendations for further improving the system's security posture, such as implementing additional access controls and reviewing user permissions. Overall, we found that the system was secure and met the necessary security standards. We make however following few recommendations to secure OSCAR EMR further:

| | |
|------------------------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Use Secure cookies | Secure cookies use the HTTPS protocol to encrypt and protect sensitive information, preventing unauthorized access and enhancing the security of web applications. |
| Rights/role System | Implement a rights and role system in the software system to manage user permissions and access control based on predefined roles and privileges. |
| Implement JWT | Consider switching to Json Web Tokens (JWT) from OAuth 1.0 as JWT uses digital signatures to ensure the authenticity and integrity of the data, whereas OAuth 1.0 relies on plaintext signatures, which can be more susceptible to attacks. |
| Remove URL Rewriting | Not doing session URL rewriting helps to prevent session hijacking and enhances the security of the web application. |
| Add CSP Header | Content Security policy (CSP) is used to mitigate and prevent Cross-Site Scripting (XSS) attacks by allowing web developers to specify which sources of content are trusted and which are not, thus preventing malicious code from being executed on a web page. |
| Add Cookie SameSite attribute | Add samesite attribute for a cookie. The SameSite attribute is an effective counter measure to cross-site request forgery, cross-site script inclusion, and timing attacks. |
| Add Strict-Transport-Security Header | HTTP Strict Transport Security (HSTS) is a web security policy mechanism whereby a web server declares that complying user agents (such as a web browser) are to interact with it using only secure HTTPS connections (i.e. HTTP layered over TLS/SSL). HSTS is an IETF standards track protocol and is specified in RFC 6797 |
| Clear text submission | Clear text submission should be avoided as it can compromise the confidentiality and integrity of sensitive information. |
| Use A WAF | Consider using a solution like Cloudflare which is a cloud-based network security and performance platform that provides a range of services including content delivery, DDoS mitigation, DNS management, web application firewall (WAF), and bot management. |
| Use an IDS (Intrusion Detection System) | An IDS is a software application that monitors network traffic or system activity for suspicious behavior and alerts security personnel or administrators. |
| Upgrade Tomcat | Upgrading Tomcat involves installing a newer version of the software, which can provide security patches, performance enhancements, new features, and improved compatibility and support. |
| Use Anti CSRF Token | An anti-CSRF token is a security measure used to protect web applications against Cross-Site Request Forgery attacks by validating the origin of a request and ensuring that it was generated by a legitimate user. |
| Monitor log files | Log file monitoring is important for detecting security incidents, identifying system errors, troubleshooting issues, ensuring compliance with regulations, and improving system performance and reliability. |
| | |

References

1. Oscar (2000, January 1). What is Oscar? Oscarcanada. Retrieved January 23, 2023, from <http://oscarcanada.org/about-oscar/brief-overview/index.html#what-is-oscar>
2. Oscar (2022, January 1). SOAP. IBM. Retrieved January 23, 2023, from <https://www.ibm.com/docs/en/wasdtfe?topic=applications-soap>
3. API (2022, January 1). What is an API? Redhat. Retrieved January 23, 2023, from <https://www.redhat.com/en/topics/api/what-are-application-programming-interfaces>
4. API security (2022, January 1). API security. Redhat. Retrieved January 23, 2023, from <https://www.redhat.com/en/topics/security/api-security>
5. Chawla, R. (2021, 10 1). API Testing Strategy and Documentation. Retrieved from QAonCloud: <https://www.qaoncloud.com/api-testing-strategy/>
6. Frye, M.-K. (n.d.). What is an API? Retrieved from MuleSoft: <https://www.mulesoft.com/resources/api/what-is-an-api>
7. Gentoo Linux. (n.d.). Nmap. Retrieved from Gentoo Linux: <https://wiki.gentoo.org/wiki/Nmap>
8. Gianchandani, P. (2012, 01 20). Burp suite walkthrough. Retrieved from INFOSEC: <https://resources.infosecinstitute.com/topic/burp-suite-walkthrough/#:~:text=Burp%20also%20has%20option%20of%20presenting>
9. marcus lantaler. (n.d.). Hydra: Hypermedia-Driven Web APIs. Retrieved from Hydra: Hypermedia-Driven Web APIs: <https://www.markus-lantaler.com/hydra/>
10. Metasploit. (n.d.). Metasploit. Retrieved from Gentoo Linux: <https://wiki.gentoo.org/wiki/Metasploit#:~:text=Wikipedia%20GitHub%20The%20Metasploit%20Project%20is%20a%20computer,framework%20is%20maintained%20by%20Rapid7%20and%20the%20community>
11. Pp_Pankaj. (2019, May 08). Software Testing | Functional Testing. Retrieved from geeksforgeeks: <https://www.geeksforgeeks.org/software-testing-functional-testing/>
12. Pp_Pankaj. (2023, 01 24). Software Testing | Security Testing. Retrieved from geeksforgeeks: <https://www.geeksforgeeks.org/software-testing-security-testing/>
13. Pp_Pankaj. (2023, 02 06). Unit Testing. Retrieved from Geeksforgeeks: <https://www.geeksforgeeks.org/unit-testing-software-testing/>
14. Sanjay. (2023, 02 01). Software Engineering | Integration Testing. Retrieved from geeksforgeeks: <https://www.geeksforgeeks.org/software-engineering-integration-testing/>
15. SOAPUI. (2023). SoapUI Open-Source Features. Retrieved from SoapUI: <https://www.soapui.org/tools/soapui/>
16. Tesauro, M., & kingthorin. (n.d.). API SECURITY TOOLS. Retrieved from OWASP: https://owasp.org/www-community/api_security_tools
17. TestingEmpire. (31, 12 2022). Best API Testing Tools (Free and Paid) For 2023. Retrieved from TestingEmpire: <https://testingempire.com/api-testing-tools/#:~:text=List%20of%20Best%20API%20Testing%20Tools%201%20%231,...%208%20%238.%20Karate%20DSL%20...%20More%20items>
18. Cardwell, D. (2018, Feb 06). Retrieved from <https://www.optiv.com/explore-optiv-insights/blog/what-ssl-web-inspection-and-where-should-it-occur-part-3>
19. Dormann, W. (2015, Mar 13). Retrieved from <https://insights.sei.cmu.edu/blog/the-risks-of-ssl-inspection/>
20. Ooreilly. (2023). Retrieved from <https://learning.oreilly.com/library/view/web-penetration-testing/9781788623377/285990a3-9992-40b0-ac36-69adc6fb47ce.xhtml>

21. Wendlandt, D. (2023). Retrieved from <https://www.cs.cmu.edu/https://www.cs.cmu.edu/~dwendlan/personal/nessus.html>
22. Shivanandhan, M. (2022, November 18). How to Use Hydrato Hack Passwords – Penetration Testing Tutorial. FreeCodeCamp.org. Retrieved February 5, 2023, from <https://www.freecodecamp.org/news/how-to-use-hydra-pentestingtutorial/#:~:text=Hydra%20is%20a%20brute%2Dforcing,databases%2C%20and%20several%20other%20services.>
23. Hegde, S. N. (2019, August 19). Practical use cases for Nmap. Red hat. Retrieved February 5, 2023, from [https://www.redhat.com/sysadmin/use-casesnmap#:~:text=Nmap%20allows%20you%20to%20scan,%2Dopen\)%2C%20and%20FTP](https://www.redhat.com/sysadmin/use-casesnmap#:~:text=Nmap%20allows%20you%20to%20scan,%2Dopen)%2C%20and%20FTP)
24. Shivanandhan, M. (2022, November 18). How to Use Hydrato Hack Passwords – Penetration Testing Tutorial. Free Code Camp. Retrieved February 20, 2023, from <https://www.freecodecamp.org/news/how-to-use-hydra-pentesting-tutorial/#:~:text=Hydra%20is%20a%20brute%2Dforcing,databases%2C%20and%20several%20other%20services.>
25. 2 - P. (2022, November 23). A Complete Guide to Nmap – Nmap Tutorial. Edureka. Retrieved February 20, 2023, from <https://www.edureka.co/blog/nmap-tutorial/>
26. Apisec (2023, April 22). What is Broken Object Level Authorization (BOLA) and How to Fix It. Retrieved February 22, 2023, from <https://www.apisec.ai/blog/broken-object-level-authorization>
27. Sslscan (2023). SSL Scan [Photograph]. <https://www.oreilly.com/api/v2/epubs/9781788623377/files/assets/4b9f1d5e-2492-4250-b800-ea603ab38254.png>
28. Soapui (2023). SOAP UI [Photograph]. https://www.soapui.org/soapui/media/images/stories/security/getting-started/running_security_test_os.png
29. Gillis, A. S. (2023, March 1). DEFINITION API testing. Tech Target. Retrieved March 27, 2023, from <https://www.techtarget.com/searchapparchitecture/definition/API-testing#:~:text=API%20testing%20is%20a%20type,as%20part%20of%20integration%20testing.>
30. OWASP. (n.d.). OWASP Zed Attack Proxy Project. Retrieved September 14, 2021, from <https://owasp.org/www-project-zap/>
31. Postman. (n.d.). API Development Environment. Retrieved September 14, 2021, from <https://www.postman.com/>
32. SSL Labs. (n.d.). SSL Server Test. Retrieved September 14, 2021, from <https://www.ssllabs.com/ssltest/>

Appendix A

Install OSCAR on VM for SOAP API testing

The Oscar installation is grouped into following phases:

1 – Installing the Infrastructure Packages

The following three commands should be executed on the Ubuntu shell:

```
$ sudo add-apt-repository universe
```

```
$ sudo apt update
```

```
$ sudo apt upgrade
```

2 – SSL Configuration

```
$ sudo apt install certbot
```

```
$ sudo certbot certonly --standalone -d FQDN
```

```
$ sudo apt upgrade
```

3 – Manual Configuration

```
$ keytool -genkey -alias tomcat -keyalg RSA -keysize 2048 -keystore YOURDOMAIN.jks
```

```
<!DOCTYPE html>
```

```
<Connector port="8443" maxHttpHeaderSize="8192" protocol="HTTP/1.1" SSLEnabled="true"
maxThreads="150" minSpareThreads="25" maxSpareThreads="75" acceptCount="100"
scheme="https" secure="true" enableLookups="false",disableUploadTimeout="true"
clientAuth="false" keystoreFile= "LOCATIONTOYOURFILE/DOMAIN.jks" keystorePass=
"PASSWORDTHATYOUWILLYSE" sslProtocol= "TLS" keyAlias= "tomcat">
```

4 – Java 8 or 11

```
$ sudo apt install openjdk-11-jre-headless
```

5 – MariaDB 10.3

```
$ sudo apt install mariadb-client libmariadb-java
```

```
$ sudo mysql_secure_installation
```

```
$ sudo systemctl status mariadb
```

```
$ sudo mysql -uroot -p
```

6 - Java 8 or 11

```
$ sudo apt install tomcat9
```

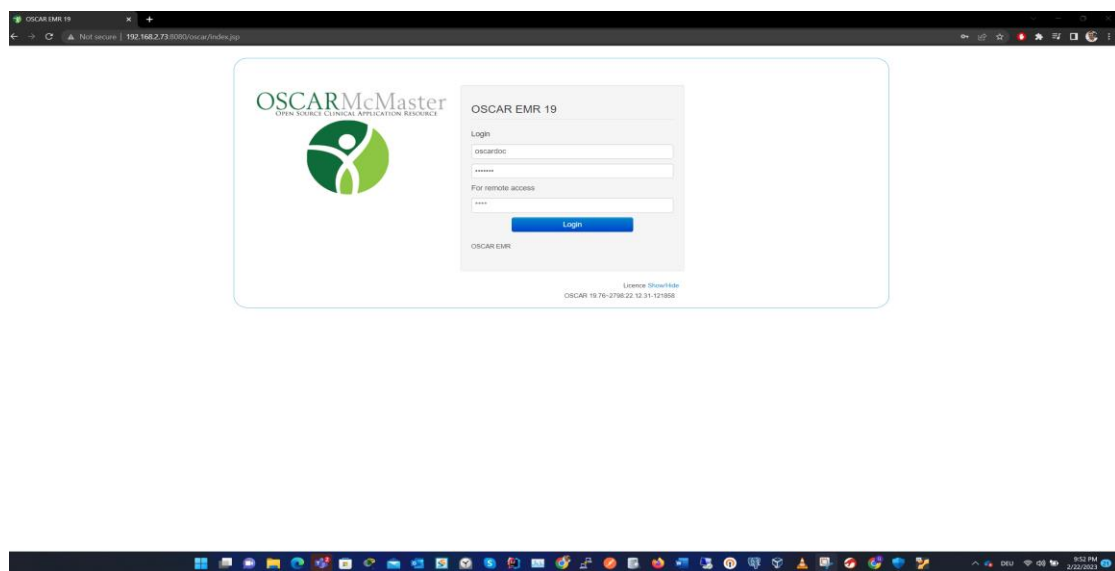
7 - Installing OSCAR

```
$ sudo wget http://sourceforge.net/projects/oscardcmaster/files/Oscar\ Debian\+Ubuntu\ deb\ Package/oscar_emr19-49~1508.deb
```

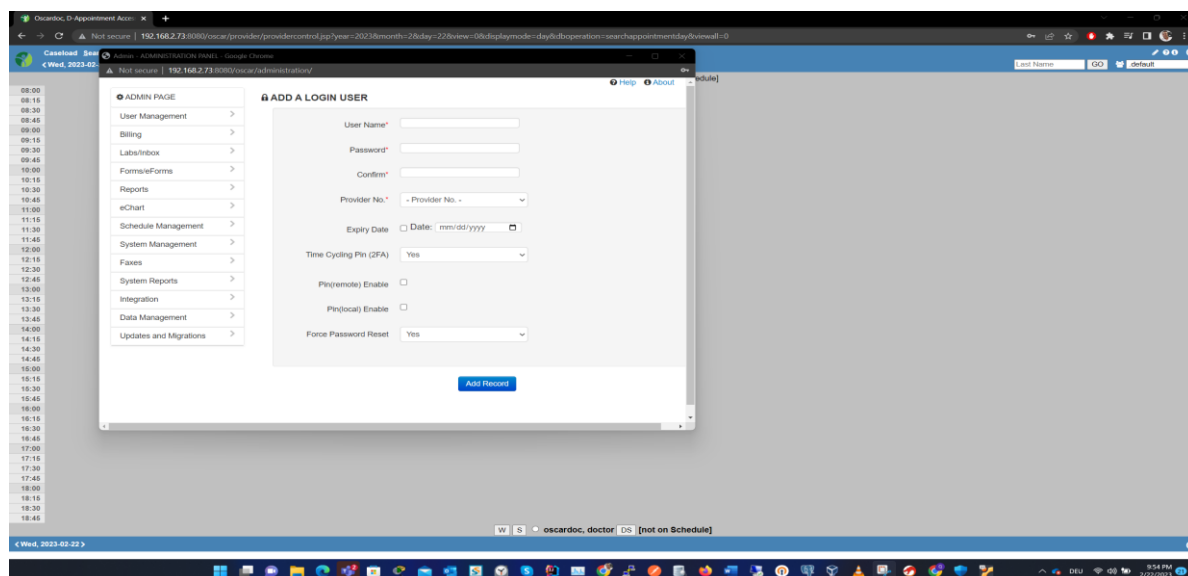
```
$ sudo dpkg -i oscar_emr19-49~1508.deb
```

```
$ tail -f /usr/share/oscar-emr/Oscar19install.log
```

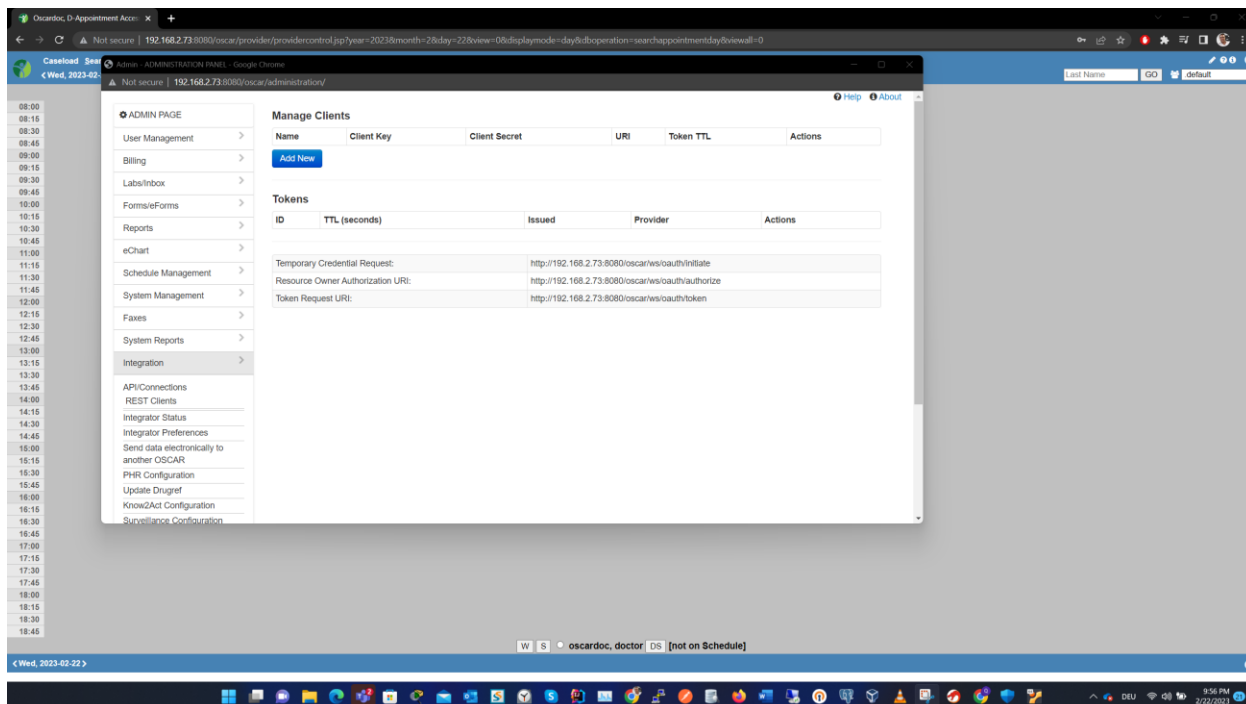
Oscar Screenshots



OSCAR EMR - Login Screen



OSCAR EMR - Add A login User Screen




OSCAR EMR – Add API Client.

APPENDIX B

Security Issues Identified in OSCAR API

Burp Suite

1 - Cleartext submission of password

| | | |
|-----------------------------------------------------------------------------------|------------|------------------------|
|  | Severity | High |
| | Confidence | Certain |
| | Path | <i>oscar/index.jsp</i> |

Issue detail

The page contains a form with the following action URL, which is submitted over clear-text HTTP:

- http://server_host/oscar/login.do;jsessionid=2ED715360486E91AA56D5EA8DAAAD532

The form contains the following password fields:

- password
- pin

Issue background

Some applications transmit passwords over unencrypted connections, making them vulnerable to interception. To exploit this vulnerability, an attacker must be suitably positioned to eavesdrop on the victim's network traffic. Common defenses such as switched networks are not sufficient to prevent this. An attacker situated in the user's ISP or the application's hosting infrastructure could also perform this attack. Note that an advanced adversary could potentially target any connection made over the Internet's core infrastructure. Vulnerabilities that result in the disclosure of users' passwords can result in compromises that are extremely difficult to investigate due to obscured audit trails. Even if the application itself only handles non-sensitive information, exposing passwords puts users who have re-used their password elsewhere at risk.

Issue remediation

Applications should use transport-level encryption (SSL or TLS) to protect all sensitive communications passing between the client and the server. Communications that should be protected include the login mechanism and related functionality, and any functions where sensitive data can be accessed or privileged actions can be performed. These areas should employ their own session handling mechanism, and the session tokens used should never be transmitted over unencrypted communications. If HTTP cookies are used for transmitting session tokens, then the secure flag should be set to prevent transmission over clear-text HTTP.

Vulnerability classifications

- CWE-319: Cleartext Transmission of Sensitive Information
- CAPEC-117: Interception

Request

```
GET /oscar_war_exploded/index.jsp HTTP/1.1
Host: localhost:8080
sec-ch-ua: "Chromium";v="111", "Not(A:Brand";v="8"
sec-ch-ua-mobile: ?0
sec-ch-ua-platform: "Windows"
Upgrade-Insecure-Requests: 1
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko)
Chrome/111.0.5563.65 Safari/537.36
Accept:
text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q
=0.8,application/signed-exchange;v=b3;q=0.7
Sec-Fetch-Site: none
Sec-Fetch-Mode: navigate.
Sec-Fetch-User: ?1
Sec-Fetch-Dest: document
Accept-Encoding: gzip, deflate.
Accept-Language: en-US,en;q=0.9
Connection: close
```

Response

```
HTTP/1.1 200
Cache-Control: no-store, no-cache, must-revalidate
Set-Cookie: JSESSIONID=2ED715360486E91AA56D5EA8DAAAD532; Path=/oscar_war_exploded; HttpOnly
Set-Cookie: oscprvid=; Max-Age=0; Expires=Thu, 01 Jan 1970 00:00:10 GMT; Path=/
X-FRAME-OPTIONS: SAMEORIGIN
Content-Type: text/html;charset=UTF-8
Date: Thu, 23 Mar 2023 00:14:56 GMT
Connection: close
Content-Length: 15356
<!DOCTYPE html>
<html lang="en">
<head>
<title>          OSCAR EMR
  </title>
<link rel="shortcut icon"
...[SNIP]...
<div class="leftinput" ng-app="indexApp" ng-controller="indexCtrl">
  <form name="loginForm" method="post"
action="/oscar_war_exploded/login.do;jsessionid=2ED715360486E91AA56D5EA8DAAAD532">

    <div class="form-group ">
...[SNIP]...
<div class="form-group ">
  <input type="password" name="password" placeholder="Enter your password"
value="" size="15" maxlength="32" autocomplete="off"
class="form-control" ng-model="password" required/>
</div>
...[SNIP]...
<div class="form-group ">
  <input type="password" name="pin" placeholder="Enter your PIN" value=""
size="15" maxlength="15" autocomplete="off" class="form-control" ng-model="pin"/>
  <span class="extrasmall">
...[SNIP]...
```

2 - Vulnerable JavaScript dependency

| | | |
|--|------------|------------------------------------|
| | Severity | Low |
| | Confidence | Tentative |
| | Path | <i>oscar/admin/api/clients.jsp</i> |

Issue Background

There are 9 instances of this issue:

<http://oscarhost/oscar/admin/api/clients.jsp>

<http://oscarhost/oscar/administration/>

<http://oscarhost/oscarhost/oscar/index.jsp>

<http://oscarhost/oscar/js/jquery-1.7.1.min.js>

<http://oscarhost/oscar/js/jquery-1.9.1.js>

<http://oscarhost/oscar/library/jquery/jquery-1.12.0.min.js>

<http://oscarhost/oscar/provider/providercontrol.jsp>

<http://oscarhost/oscar/share/javascript/prototype.js>

http://oscarhost/oscar/portal_javascripts/Old%20Plone%203%20Custom%20Theme/resourceplone.app.jquery-cachekey-471b79589b8859d7aa21d44dd428c1ca.js

Issue background

The use of third-party JavaScript libraries can introduce a range of DOM-based vulnerabilities, including some that can be used to hijack user accounts like DOM-XSS.

Common JavaScript libraries typically enjoy the benefit of being heavily audited. This may mean that bugs are quickly identified and patched upstream, resulting in a steady stream of security updates that need to be applied. Although it may be tempting to ignore updates, using a library with missing security patches can make your website exceptionally easy to exploit. Therefore, it's important to ensure that any available security updates are applied promptly.

Some library vulnerabilities expose every application that imports the library, but others only affect applications that use certain library features. Accurately identifying which library vulnerabilities apply to your website can be difficult, so we recommend applying all available security updates regardless.

Issue remediation

Develop a patch-management strategy to ensure that security updates are promptly applied to all third-party libraries in your application. Also, consider reducing your attack surface by removing any libraries that are no longer in use.

Vulnerability classifications

- [CWE-1104: Use of Unmaintained Third-Party Components](#)
- [A9: Using Components with Known Vulnerabilities](#)

Issue detail

We observed a vulnerable JavaScript library.

We detected jQuery version 1.7.1.min, which has the following vulnerabilities:

- [CVE-2012-6708](#): Selector interpreted as HTML.
- [CVE-2015-9251](#): 3rd party CORS request may execute.

- [CVE-2019-11358](#): jQuery before 3.4.0, as used in Drupal, Backdrop CMS, and other products, mishandles jQuery.extend(true, {}, ...) because of Object. Prototype pollution.
- [CVE-2020-11022](#): Regex in its jQuery.htmlPrefilter sometimes may introduce XSS.
- [CVE-2020-11023](#): Regex in its jQuery.htmlPrefilter sometimes may introduce XSS.

3 - Content type incorrectly stated

There are 4 instances of this issue:

[/oscar war exploded/css/font/Roboto/Roboto-Bold.woff2](#)

[/oscar war exploded/css/font/Roboto/Roboto-Regular.woff2](#)

[/oscar war exploded/css/font/fontawesome-webfont.woff](#)

[/oscar war exploded/library/bootstrap/3.0.0/fonts/glyphicons-halflings-regular.woff](#)

Issue background

If a response specifies an incorrect content type then browsers may process the response in unexpected ways. If the content type is specified to be a renderable text-based format, then the browser will usually attempt to interpret the response as being in that format, regardless of the actual contents of the response. Additionally, some other specified content types might sometimes be interpreted as HTML due to quirks in particular browsers. This behavior might lead to otherwise "safe" content such as images being rendered as HTML, enabling cross-site scripting attacks in certain conditions.

The presence of an incorrect content type statement typically only constitutes a security flaw when the affected resource is dynamically generated, uploaded by a user, or otherwise contains user input. You should review the contents of affected responses, and the context in which they appear, to determine whether any vulnerability exists.

Issue remediation

For every response containing a message body, the application should include a single Content-type header that correctly and unambiguously states the MIME type of the content in the response body. Additionally, the response header "X-content-type-options: nosniff" should be returned in all responses to reduce the likelihood that browsers will interpret content in a way that disregards the Content-type header.

References

[Web Security Academy: Cross-site scripting](#)

Vulnerability classifications

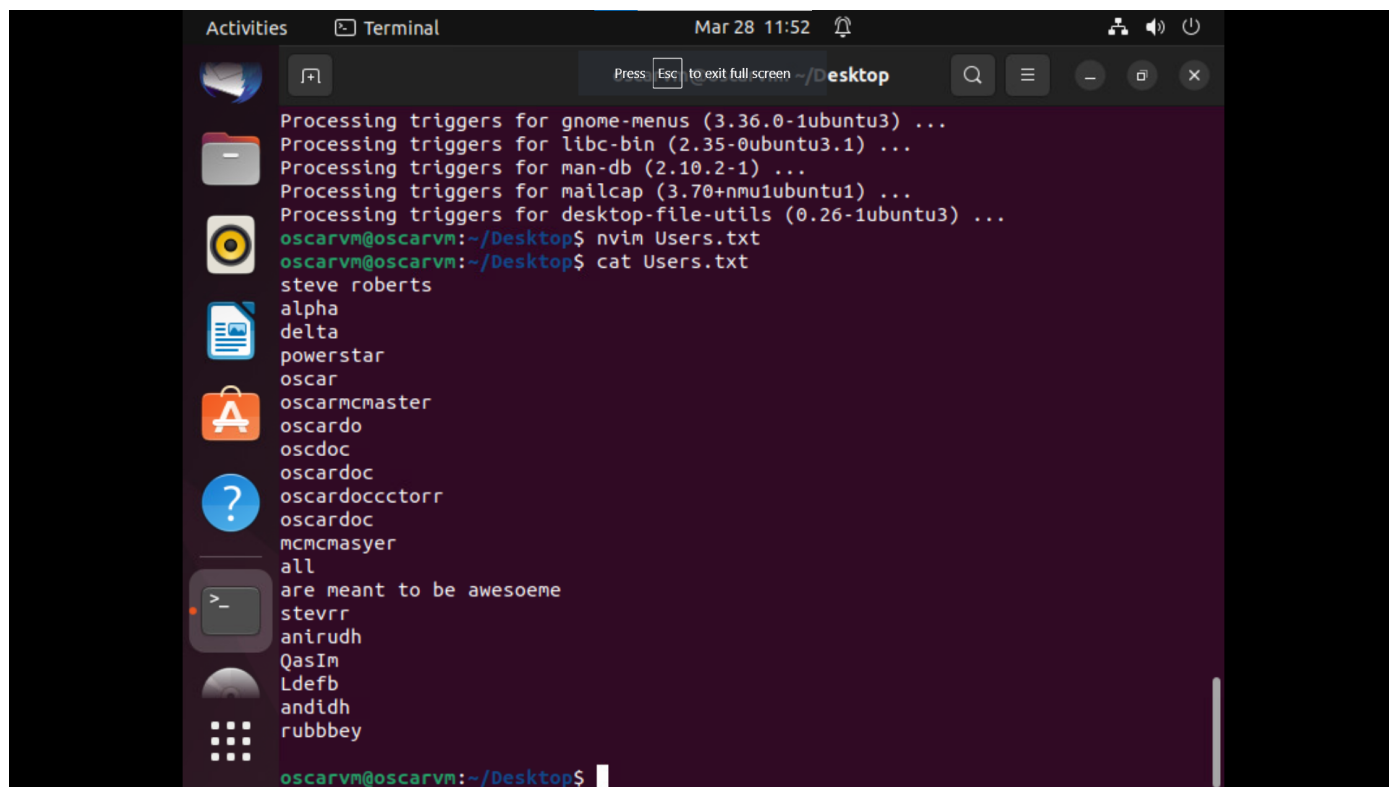
[CWE-16: Configuration](#)

[CWE-436: Interpretation Conflict](#)

[CAPEC-63: Cross-Site Scripting \(XSS\)](#)

HYDRA

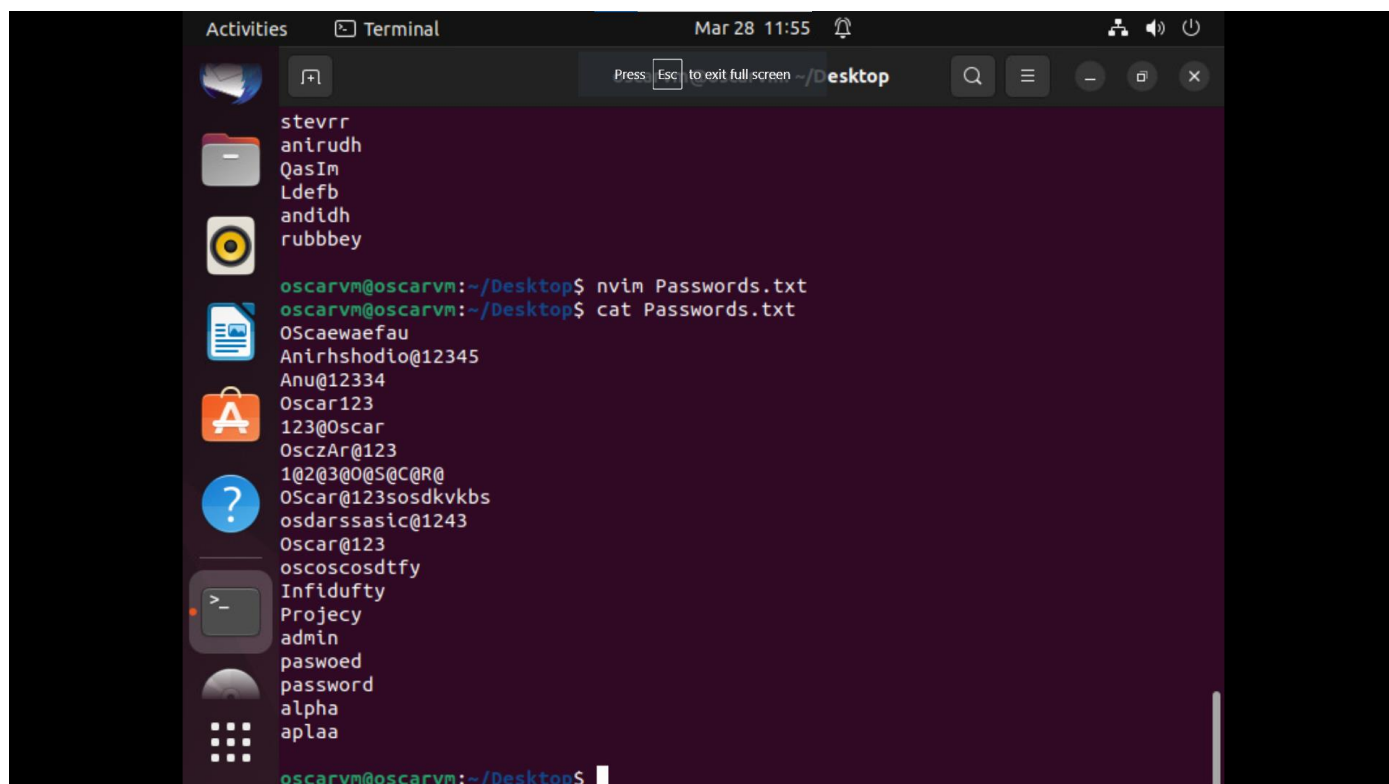
Below is the list of Users that have been potentially identified as possible usernames that may be used as Usernames for the Oscar Website.

A terminal window titled 'Terminal' with a date and time of 'Mar 28 11:52'. The window shows the output of a Hydra command. The first part of the output lists system triggers being processed for various packages like gnome-menus, libc-bin, man-db, mailcap, and desktop-file-utils. Then, it shows the command 'nvm Users.txt' and 'cat Users.txt'. The output of 'cat Users.txt' is a list of usernames: steve roberts, alpha, delta, powerstar, oscar, oscarmcmaster, oscardo, oscdoc, oscardoc, oscardocctorr, oscardoc, mcmcmasyer, all, are meant to be awesome, stevrr, anirudh, QasIm, Ldefb, andidh, and rubbbey. The prompt 'oscarvm@oscarvm:~/Desktop\$' is visible at the bottom.

```
Processing triggers for gnome-menus (3.36.0-1ubuntu3) ...
Processing triggers for libc-bin (2.35-0ubuntu3.1) ...
Processing triggers for man-db (2.10.2-1) ...
Processing triggers for mailcap (3.70+nmu1ubuntu1) ...
Processing triggers for desktop-file-utils (0.26-1ubuntu3) ...
oscarvm@oscarvm:~/Desktop$ nvm Users.txt
oscarvm@oscarvm:~/Desktop$ cat Users.txt
steve roberts
alpha
delta
powerstar
oscar
oscarmcmaster
oscardo
oscdoc
oscardoc
oscardocctorr
oscardoc
mcmcmasyer
all
are meant to be awesome
stevrr
anirudh
QasIm
Ldefb
andidh
rubbbey
oscarvm@oscarvm:~/Desktop$
```

Hydra User inputs for API password cracking.

Below is the list of words that have been potentially identified as possible passwords that may be used as a validation credential for the Oscar Website.

A terminal window titled 'Terminal' with a date and time of 'Mar 28 11:55'. The window shows the output of a Hydra command. The first part of the output lists system triggers being processed for various packages like gnome-menus, libc-bin, man-db, mailcap, and desktop-file-utils. Then, it shows the command 'nvm Passwords.txt' and 'cat Passwords.txt'. The output of 'cat Passwords.txt' is a list of passwords: stevrr, anirudh, QasIm, Ldefb, andidh, and rubbbey. The prompt 'oscarvm@oscarvm:~/Desktop\$' is visible at the bottom.

```
Processing triggers for gnome-menus (3.36.0-1ubuntu3) ...
Processing triggers for libc-bin (2.35-0ubuntu3.1) ...
Processing triggers for man-db (2.10.2-1) ...
Processing triggers for mailcap (3.70+nmu1ubuntu1) ...
Processing triggers for desktop-file-utils (0.26-1ubuntu3) ...
oscarvm@oscarvm:~/Desktop$ nvm Passwords.txt
oscarvm@oscarvm:~/Desktop$ cat Passwords.txt
stevrr
anirudh
QasIm
Ldefb
andidh
rubbbey
oscarvm@oscarvm:~/Desktop$
```

Password inputs for the given user-names utilized to do the scan.

Password cracked using hydra.

#HYDRA RESULTS

Hydra v9.2 run at 2023-03-28 13:20:18 on 127.0.0.1 http-get (hydra -L Users.txt -P Passwords.txt -o hydra.txt http-get://127.0.0.1:8080)

| | | | |
|------------------|-----------------|----------------------|------------------------------|
| [8080][http-get] | host: 127.0.0.1 | login: steve roberts | password: osdarssasic@1243 |
| [8080][http-get] | host: 127.0.0.1 | login: steve roberts | password: OScaewaefau |
| [8080][http-get] | host: 127.0.0.1 | login: steve roberts | password: Anirhshodio@12345 |
| [8080][http-get] | host: 127.0.0.1 | login: steve roberts | password: Anu@12334 |
| [8080][http-get] | host: 127.0.0.1 | login: steve roberts | password: Oscar123 |
| [8080][http-get] | host: 127.0.0.1 | login: steve roberts | password: 123@Oscar |
| [8080][http-get] | host: 127.0.0.1 | login: steve roberts | password: OszAr@123 |
| [8080][http-get] | host: 127.0.0.1 | login: steve roberts | password: 1@2@3@0@S@C@R@ |
| [8080][http-get] | host: 127.0.0.1 | login: steve roberts | password: OScar@123sosdkvkbs |
| [8080][http-get] | host: 127.0.0.1 | login: steve roberts | password: Oscar@123 |
| [8080][http-get] | host: 127.0.0.1 | login: steve roberts | password: oscoscodtfy |
| [8080][http-get] | host: 127.0.0.1 | login: steve roberts | password: Infidulty |
| [8080][http-get] | host: 127.0.0.1 | login: steve roberts | password: |

```
Projecy
[8080][http-get] host: 127.0.0.1    login: steve roberts    password:
admin
[8080][http-get] host: 127.0.0.1    login: steve roberts    password:
paswoed
[8080][http-get] host: 127.0.0.1    login: steve roberts    password:
password
[8080][http-get] host: 127.0.0.1    login: alpha      password:
Oscaewaefau
.....
.....
.....
.....
.....
.....
```

Nmap

```
nmap -Pn -sn 192.168.2.73/24
Starting Nmap 7.93 ( https://nmap.org ) at 2023-03-29 21:11 EDT
Nmap scan report for qasim-VirtualBox (192.168.2.73)
Host is up.
```

ZAPPROXY- SCANNING, REPORT AND SOLUTIONS

After installing ZAP Proxy, launch the application and configure the proxy settings.

To do this, go to "Tools" > "Options" > "Local Proxy" and set the listening address to 127.0.0.1 and the port number to 8080.

To scan the OSCAR website using ZAP Proxy, we must configure your browser to use the Proxy. To do this, go to the browser settings and set the proxy settings to use the IP address 127.0.0.1 and port number 8080.

Open the OSCAR website in your browser and ensure it loads over HTTPS.

Return to ZAP Proxy and start a new scan by clicking the "New Session" button. Then, enter the URL of the OSCAR website and make sure that the "Attack Mode" is set to "Safe."

Once the scan is complete, ZAP Proxy displays the results of the scan. Look for any vulnerabilities related to HTTPS, such as weak ciphers, expired or invalid certificates, or mixed content.

After identifying any HTTPS vulnerabilities, remediate them by following best practices for HTTPS, such as using strong ciphers, ensuring that certificates are valid and not expired, and ensuring that all content is loaded over HTTPS.

ZAP Scanning Report

Site: https://localhost:8443

Generated on Tue, 28 Mar 2023 11:04:47

Summary of Alerts

| Risk Level | Number of Alerts |
|---------------|------------------|
| High | 0 |
| Medium | 4 |
| Low | 5 |
| Informational | 3 |

Alerts overview

Alerts

| Name | Risk Level | Number of Instances |
|--------------------------------------------------------------|---------------|---------------------|
| Absence of Anti-CSRF Tokens | Medium | 2 |
| Content Security Policy (CSP) Header Not Set | Medium | 4 |
| Missing Anti-clickjacking Header | Medium | 1 |
| Session ID in URL Rewrite | Medium | 2 |
| Cookie No HttpOnly Flag | Low | 1 |
| Cookie Without Secure Flag | Low | 1 |
| Cookie without SameSite Attribute | Low | 2 |
| Strict-Transport-Security Header Not Set | Low | 8 |
| X-Content-Type-Options Header Missing | Low | 5 |
| Information Disclosure - Suspicious Comments | Informational | 2 |
| Loosely Scoped Cookie | Informational | 2 |
| User Agent Fuzzer | Informational | 12 |

Alert Details

| Medium | Absence of Anti-CSRF Tokens |
|-------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| | No Anti-CSRF tokens were found in an HTML submission form. |
| Description | <p>A cross-site request forgery is an attack that involves forcing a victim to send an HTTP request to a target destination without their knowledge or intent in order to perform an action as the victim. The underlying cause is application functionality using predictable URL /form actions in a repeatable way. The nature of the attack is that CSRF exploits the trust that a website has for a user. By contrast, cross-site scripting (XSS) exploits the trust that a user has in a website. Like XSS, CSRF attacks are not necessarily cross-site, but they can be. Cross-site request forgery is also known as CSRF, XSRF, one-click attack, session riding, confused deputy, and sea surf.</p> <p>CSRF attacks are effective in a number of situations, including:</p> <ul style="list-style-type: none">* The victim has an active session on the target site. |
| | <p>The victim is authenticated via HTTP auth on the target site.</p> <p>The victim is on the same local network as the target site.</p> <p>CSRF has primarily been used to perform an action against a target site using the victim's privileges, but recent techniques have been discovered to disclose information by gaining access to the response. The risk of information disclosure is dramatically increased when the target site is vulnerable to XSS</p> |

| | |
|----------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| | because XSS can be used as a platform for CSRF, allowing the attack to operate within the bounds of the same-origin policy. |
| URL | https://localhost:8443/oscar/index.jsp |
| Method | GET |
| Attack | |
| Evidence | <form name="loginForm" method="post" action="/oscar/login.do;jsessionid=B3FA96D4A02F448BE14CC9BB33F73C51"> |
| URL | https://localhost:8443/oscar/index.jsp |
| Method | GET |
| Attack | |
| Evidence | <form name="loginForm" method="post" action="/oscar/login.do;jsessionid=E0E30C96B95FCDCD368E19EAE3F5DCBD"> |
| Instances | 2 |
| Solution | <p>Phase: Architecture and Design</p> <p>Use a vetted library or framework that does not allow this weakness to occur or provides constructs that make this weakness easier to avoid.</p> <p>For example, use anti-CSRF packages such as the OWASP CSRFGuard. Phase: Implementation</p> <p>Ensure that your application is free of cross-site scripting issues, because most CSRF defenses can be bypassed using attacker-controlled script.</p> <p>Phase: Architecture and Design</p> <p>Generate a unique nonce for each form, place the nonce into the form, and verify the nonce upon receipt of the form. Be sure that the nonce is not predictable (CWE-330).</p> <p>Note that this can be bypassed using XSS.</p> <p>Identify especially dangerous operations. When the user performs a dangerous operation, send a separate confirmation request to ensure that the user intended to perform that operation.</p> <p>Note that this can be bypassed using XSS. Use the ESAPI Session Management control. This control includes a component for CSRF.</p> <p>Do not use the GET method for any request that triggers a state change. Phase: Implementation</p> <p>Check the HTTP Referer header to see if the request originated from an expected page. This could break legitimate functionality, because users or proxies may have disabled sending the Referer for privacy reasons.</p> <p>http://projects.webappsec.org/Cross-Site-Request-Forgery http://cwe.mitre.org/data/definitions/352.html 352</p> |
| Reference | Reference |
| CWE Id WASC Id | CWE Id WASC Id |
| Plugin Id | 10202 |
| Medium | Content Security Policy (CSP) Header Not Set |
| Description | <p>Content Security Policy (CSP) is an added layer of security that helps to detect and mitigate certain types of attacks, including Cross Site Scripting (XSS) and data injection attacks. These attacks are used for everything from data theft to site defacement or distribution of malware. CSP provides a set of standard HTTP headers that allow website owners to declare approved sources of content that browsers should be allowed to load on that page — covered types are JavaScript, CSS, HTML frames, fonts, images and embeddable objects such as Java applets, ActiveX, audio and video files.</p> |
| URL | https://localhost:8443/oscar/index.jsp |
| Method | GET |
| Attack | |
| Evidence | |
| URL | https://localhost:8443/robots.txt |
| Method | GET |
| Attack | |
| Evidence | |
| URL | https://localhost:8443/sitemap.xml |
| Method | GET |
| Attack | |

| | |
|----------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Evidence | |
| URL | https://localhost:8443/sitemap.xml/sitemap.xml |
| Method | GET |
| Attack | |
| Evidence | |
| Instances | 4 |
| Solution | Ensure that your web server, application server, load balancer, etc. is configured to set the Content-Security-Policy header. https://developer.mozilla.org/en-US/docs/Web/Security/CSP/Introducing_Content_Security_Policy https://cheatsheetseries.owasp.org/cheatsheets/Content_Security_Policy_Cheat_Sheet.html |
| Reference | http://www.w3.org/TR/CSP/ http://w3c.github.io/webappsec/specs/content-security-policy/csp-specification.dev.html http://www.html5rocks.com/en/tutorials/security/content-security-policy/ http://caniuse.com/#feat=contentsecuritypolicy http://content-security-policy.com/ 693 |
| CWE Id WASC Id | 15 |
| Plugin Id | 10038 |
| Medium | Content Security Policy (CSP) Header Not Set |
| Description | Content Security Policy (CSP) is an added layer of security that helps to detect and mitigate certain types of attacks, including Cross Site Scripting (XSS) and data injection attacks. These attacks are used for everything from data theft to site defacement or distribution of malware. CSP provides a set of standard HTTP headers that allow website owners to declare approved sources of content that browsers should be allowed to load on that page — covered types are JavaScript, CSS, HTML frames, fonts, images and embeddable objects such as Java applets, ActiveX, audio and video files. |
| URL | https://localhost:8443/oscar/index.jsp |
| Method | GET |
| Attack | |
| Evidence | |
| URL | https://localhost:8443/robots.txt |
| Method | GET |
| Attack | |
| Evidence | |
| URL | https://localhost:8443/sitemap.xml |
| Method | GET |
| Attack | |
| Evidence | |
| URL | https://localhost:8443/sitemap.xml/sitemap.xml |
| Method | GET |
| Attack | |
| Evidence | |
| Instances | 4 |
| Solution | Ensure that your web server, application server, load balancer, etc. is configured to set the Content-Security-Policy header. https://developer.mozilla.org/en-US/docs/Web/Security/CSP/Introducing_Content_Security_Policy https://cheatsheetseries.owasp.org/cheatsheets/Content_Security_Policy_Cheat_Sheet.html |
| Reference | http://www.w3.org/TR/CSP/ http://w3c.github.io/webappsec/specs/content-security-policy/csp-specification.dev.html http://www.html5rocks.com/en/tutorials/security/content-security-policy/ http://caniuse.com/#feat=contentsecuritypolicy http://content-security-policy.com/ 693 |
| CWE Id WASC Id | 15 |
| Plugin Id | 10038 |
| Medium | Missing Anti-clickjacking Header |
| Description | The response does not include either Content-Security-Policy with 'frame-ancestors' directive or X-Frame-Options to protect against 'ClickJacking' attacks. |
| URL | https://localhost:8443/oscar/index.jsp |
| Method | GET |
| Attack | |

| | |
|------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Evidence | |
| Solution | Modern Web browsers support the Content-Security-Policy and X-Frame-Options HTTP headers. Ensure one of them is set on all web pages returned by your site/app. |
| Reference CWE Id | https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/X-Frame-Options 1021 |
| WASC Id | 15 |
| Plugin Id | 10020 |
| Medium | Session ID in URL Rewrite |
| Description | URL rewrite is used to track user session ID. The session ID may be disclosed via cross- site referrer header. In addition, the session ID might be stored in browser history or server logs. |
| URL | https://localhost:8443/oscar/images/OSCAR-LOGO.gif;jsessionid=E0E30C96B95FCDCD368E19EAE3F5DCBD |
| Method | GET |
| Attack | |
| Evidence | jsessionid=E0E30C96B95FCDCD368E19EAE3F5DCBD https://localhost:8443/oscar/login.do;jsessionid=E0E30C96B95FCDCD368E19EAE3F5DCBD |
| URL | POST |
| Method | |
| Attack | |
| Evidence | jsessionid=E0E30C96B95FCDCD368E19EAE3F5DCBD |
| Instances | 2 |
| Solution | For secure content, put session ID in a cookie. To be even more secure consider using a combination of cookie and URL rewrite. |
| Reference CWE Id | http://seclists.org/lists/webappsec/2002/Oct-Dec/0111.html 200 |
| WASC Id | 13 |
| Plugin Id | 3 |
| Low | Cookie No HttpOnly Flag |
| Description | A cookie has been set without the HttpOnly flag, which means that the cookie can be accessed by JavaScript. If a malicious script can be run on this page then the cookie will be accessible and can be transmitted to another site. If this is a session cookie then session hijacking may be possible. |
| URL | https://localhost:8443/oscar/index.jsp |
| Method | GET |
| Attack | |
| Evidence | Set-Cookie: oscprvid |
| Instances | 1 |
| Solution | Ensure that the HttpOnly flag is set for all cookies. |
| Reference CWE Id | https://owasp.org/www-community/HttpOnly 1004 |
| WASC Id | 13 |
| Plugin Id | 10010 |
| Low | Cookie Without Secure Flag |
| Description | A cookie has been set without the secure flag, which means that the cookie can be accessed via unencrypted connections. |
| URL | https://localhost:8443/oscar/index.jsp |
| Method | GET |
| Attack | |
| Evidence | Set-Cookie: oscprvid |
| Instances | 1 |
| Solution | Whenever a cookie contains sensitive information or is a session token, then it should always be passed using an encrypted channel. Ensure that the secure flag is set for cookies containing such sensitive information. |
| Reference | https://owasp.org/www-project-web-security-testing-guide/v41/4-Web_Application_Security_Testing/06-Session_Management_Testing/02-Testing_for_Cookies_Attributes.html |
| CWE Id WASC Id | 614 13 |
| Plugin Id | 10011 |
| Low | Cookie without SameSite Attribute |

| | |
|------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Description | A cookie has been set without the SameSite attribute, which means that the cookie can be sent as a result of a 'cross-site' request. The SameSite attribute is an effective counter measure to cross-site request forgery, cross-site script inclusion, and timing attacks. |
| URL | https://localhost:8443/oscar/index.jsp |
| Method | GET |
| Attack | |
| Evidence | Set-Cookie: JSESSIONID |
| URL | https://localhost:8443/oscar/index.jsp |
| Method | GET |
| Attack | |
| Evidence | Set-Cookie: oscprvid |
| Instances | 2 |
| Solution | Ensure that the SameSite attribute is set to either 'lax' or ideally 'strict' for all cookies. |
| Reference CWE Id | https://tools.ietf.org/html/draft-ietf-httpbis-cookie-same-site 1275 |
| WASC Id | 13 |
| Plugin Id | 10054 |
| Low | Strict-Transport-Security Header Not Set |
| Description | HTTP Strict Transport Security (HSTS) is a web security policy mechanism whereby a web server declares that complying user agents (such as a web browser) are to interact with it using only secure HTTPS connections (i.e. HTTP layered over TLS/SSL). HSTS is an IETF standards track protocol and is specified in RFC 6797. |
| URL | https://localhost:8443/oscar/css/bootstrap-responsive.css |
| Method | GET |
| Attack | |
| URL | https://localhost:8443/oscar/css/bootstrap.css |
| Method | GET |
| Attack | |
| Evidence | https://localhost:8443/oscar/images/OSCAR-LOGO.gif;jsessionid=E0E30C96B95FCDCD368E19EAE3F5DCBD |
| URL | GET |
| Method | |
| Attack | |
| Evidence | |
| URL | https://localhost:8443/oscar/images/Oscar.ico |
| Method | GET |
| Attack | |
| Evidence | |
| URL | https://localhost:8443/oscar/index.jsp |
| Method | GET |
| Attack | |
| Evidence | |
| URL | https://localhost:8443/robots.txt |
| Method | GET |
| Attack | |
| Evidence | |
| URL | https://localhost:8443/sitemap.xml |
| Method | GET |
| Attack | |
| Evidence | |
| URL | https://localhost:8443/sitemap.xml/sitemap.xml |
| Method | GET |
| Attack | |
| Evidence | |
| Instances | 8 |
| Solution | Ensure that your web server, application server, load balancer, etc. is configured to enforce Strict-Transport-Security. |
| Reference | https://cheatsheetseries.owasp.org/cheatsheets/HTTP_Strict_Transport_Security_Cheat_Sheet.html https://owasp.org/www-community/Security-Headers http://en.wikipedia.org/wiki/HTTP_Strict_Transport_Security |

| | |
|----------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| | http://caniuse.com/stricttransportsecurity http://tools.ietf.org/html/rfc6797 |
| CWE Id WASC Id | 319 15 |
| Plugin Id | 10035 |
| Low | X-Content-Type-Options Header Missing |
| Description | The Anti-MIME-Sniffing header X-Content-Type-Options was not set to 'nosniff'. This allows older versions of Internet Explorer and Chrome to perform MIME-sniffing on the response body, potentially causing the response body to be interpreted and displayed as a content type other than the declared content type. Current (early 2014) and legacy versions of Firefox will use the declared content type (if one is set), rather than performing MIME- sniffing. |
| URL | https://localhost:8443/oscar/css/bootstrap-responsive.css |
| Method | GET |
| Attack | |
| Evidence | |
| URL | https://localhost:8443/oscar/css/bootstrap.css |
| Method | GET |
| Attack | |
| Evidence | |
| URL | https://localhost:8443/oscar/images/OSCAR-LOGO.gif;jsessionid=E0E30C96B95FCDCD368E19EAE3F5DCBD |
| Method | GET |
| Attack | |
| Evidence | |
| URL | https://localhost:8443/oscar/images/Oscar.ico |
| Method | GET |
| Attack | |
| Evidence | |
| URL | https://localhost:8443/oscar/index.jsp |
| Method | GET |
| Attack | |
| Evidence | |
| Instances | 5 |
| Solution | Ensure that the application/web server sets the Content-Type header appropriately, and that it sets the X-Content-Type-Options header to 'nosniff' for all web pages. If possible, ensure that the end user uses a standards-compliant and modern web browser that does not perform MIME-sniffing at all, or that can be directed by the web application /web server to not perform MIME-sniffing. http://msdn.microsoft.com/en-us/library/ie/gg622941%28v=vs.85%29.aspx https://owasp.org/www-community/Security-Headers 693 15 |
| Reference | |
| CWE Id WASC Id | |
| Plugin Id | 10021 |
| Informational | Information Disclosure - Suspicious Comments |
| Description | The response appears to contain suspicious comments which may help an attacker. Note: Matches made within script blocks or files are against the entire content not only comments. |
| URL | https://localhost:8443/oscar/index.jsp |
| Method | GET |
| Attack | |
| Evidence | username |
| URL | https://localhost:8443/oscar/index.jsp |
| Method | GET |
| Attack | |
| Evidence | from |
| Instances | 2 |
| Solution | Remove all comments that return information that may help an attacker and fix any underlying problems they refer to. |

| | |
|----------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Reference | |
| CWE Id | 200 |
| WASC Id | 13 |
| Plugin Id | 10027 |
| Informational | Loosely Scoped Cookie |
| Description | Cookies can be scoped by domain or path. This check is only concerned with domain scope. The domain scope applied to a cookie determines which domains can access it. For example, a cookie can be scoped strictly to a subdomain e.g. www.nottrusted.com, or loosely scoped to a parent domain e.g. nottrusted.com. In the latter case, any subdomain of nottrusted.com can access the cookie. Loosely scoped cookies are common in mega- applications like google.com and live.com. Cookies set from a subdomain like app.foo.bar are transmitted only to that domain by the browser. However, cookies scoped to a parent- level domain may be transmitted to the parent, or any subdomain of the parent. |
| URL | https://localhost:8443/oscar/index.jsp |
| Method | GET |
| Attack | |
| Evidence | |
| URL | https://localhost:8443/oscar/index.jsp |
| Method | GET |
| Attack | |
| Evidence | |
| Instances | 2 |
| Solution | Always scope cookies to a FQDN (Fully Qualified Domain Name). https://tools.ietf.org/html/rfc6265#section-4.1 https://owasp.org/www-project-web-security-testing-guide/v41/4-Web_Application_Security_Testing/06-Session_Management_Testing/02-Testing_for_Cookies_Attributes.html http://code.google.com/p/browsersec/wiki/Part2#Same-origin_policy_for_cookies |
| Reference | 565 |
| CWE Id WASC Id | 15 |
| Plugin Id | 90033 |
| Informational | User Agent Fuzzer |
| Description | Check for differences in response based on fuzzed User Agent (eg. mobile sites, access as a Search Engine Crawler). Compares the response statuscode and the hashcode of the response body with the original response. |
| URL | https://localhost:8443/oscar/index.jsp |
| Method | GET |
| Attack | Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1) |
| Evidence | |
| URL | https://localhost:8443/oscar/index.jsp |
| Method | GET |
| Attack | Mozilla/4.0 (compatible; MSIE 7.0; Windows NT 6.0) |
| Evidence | |
| URL | https://localhost:8443/oscar/index.jsp |
| Method | GET |
| Attack | Mozilla/4.0 (compatible; MSIE 8.0; Windows NT 6.1) |
| Evidence | |
| URL | https://localhost:8443/oscar/index.jsp |
| Method | GET |
| Attack | Mozilla/5.0 (Windows NT 10.0; Trident/7.0; rv:11.0) like Gecko |
| Evidence | |
| URL | https://localhost:8443/oscar/index.jsp |
| Method | GET |
| Attack | Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/75.0.3739.0 Safari/537.36 Edg/75.0.109.0 |
| Evidence | |
| URL | https://localhost:8443/oscar/index.jsp |
| Method | GET |
| Attack | Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/91.0.4472.124 Safari/537.36 |
| Evidence | |
| URL | https://localhost:8443/oscar/index.jsp |
| Method | GET |

| | |
|----------|--------------------------------------------------------------------------------|
| Attack | Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:93.0) Gecko/20100101 Firefox/91.0 |
| Evidence | |
| | |