

Sri Lanka Institute of Information Technology

**Bug Bounty Review Report**

**Individual Assignment**

IE2062 - Web Security

Submitted by:

|  |  |
| --- | --- |
| Student Registration Number | Student Name |
| IT21022142 | Abeysinghe Y.P. P |

**Acknowledgement**

Firstly, I would like to grant my sincere gratitude specially for the lecturer in charge of the Web Security module Dr. Lakmal Rupasinghe for his invaluable guidance and advices throughout the degree program which encouraged us on having a deep understanding on web security concepts, secure programming and bug bounty.

And also, I would like to express my deep gratuity for Ms. Chethana Liyanapathirana and Ms. Tharsha Soloman who provided us the proper guidance and in-depth knowledge on understanding the web security concepts withing the lectures and the lab series of the module.

Table of Contents

[1. Broken Access Control 4](#_Toc119311727)

[2. Blind SSRF 8](#_Toc119311728)

[3. SQL Injection 13](#_Toc119311729)

[4. Violation of Secure Design Principals 20](#_Toc119311730)

[5. Insecure Deserialization 23](#_Toc119311731)

[6. Video explanation 27](#_Toc119311732)

[7. Portswigger progress 27](#_Toc119311733)

# Broken Access Control

* **Program**: - HackerOne (https://hackerone.com/reports/1472721)
* **Area**: - A01:2021-Broken Access Control
* **Bounty hunter**: -   
  This vulnerability is been detected by Toby Davenport who is currently working as an IT and Cyber Security Engineer at Cetsat partaking at United Kingdom. Throughout his bug hunting journey, he has aided number of platforms and organizations namely TVA.gov, Microsoft, BBC Tribepad, Ipaper, JotForm, Department of Defense United Nations etc. In addition, he has found out CVE-2022-35863, CVE-2022-35863, CVE-2022-3625.

LinkedIn profile: <https://www.linkedin.com/in/toby-davenport>

* **Organization and Assets**:

This is a website <https://demo.sftool.gov/> which is considered to be an official website managed by United States government.

* **Vulnerability**: -

This is an access control vulnerability caused due to IDOR (Insecure Direct Object Reference). This is a common web security vulnerability caused due to horizontal privilege escalation since the user-supplied inputs are been used by the application to directly access the objects.

Here it is possible to access the contents of another user tampering the URL parameters without any issues.

Root causes for IDOR:

* URL for form parameters can be modified by the clients which will be sufficient enough to change the direct reference.
* Web applications directly refers to internal resources and operations of the organization.
* Despite not checking whether the particular client is authorized on not, the web application grants the access to the internal objects.
* **How vulnerability is discovered**: -

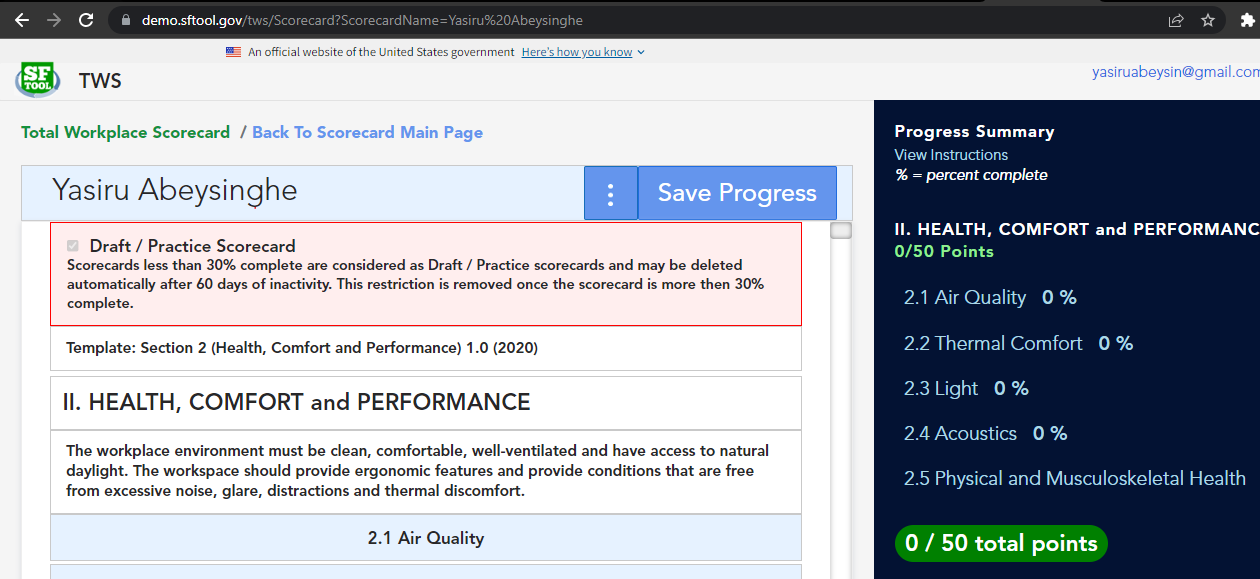
Here Toby has created two legitimate accounts. And he has checked whether he is possible to intercept the access controls of an account by logging to the other account.

* **How it is performed: -**

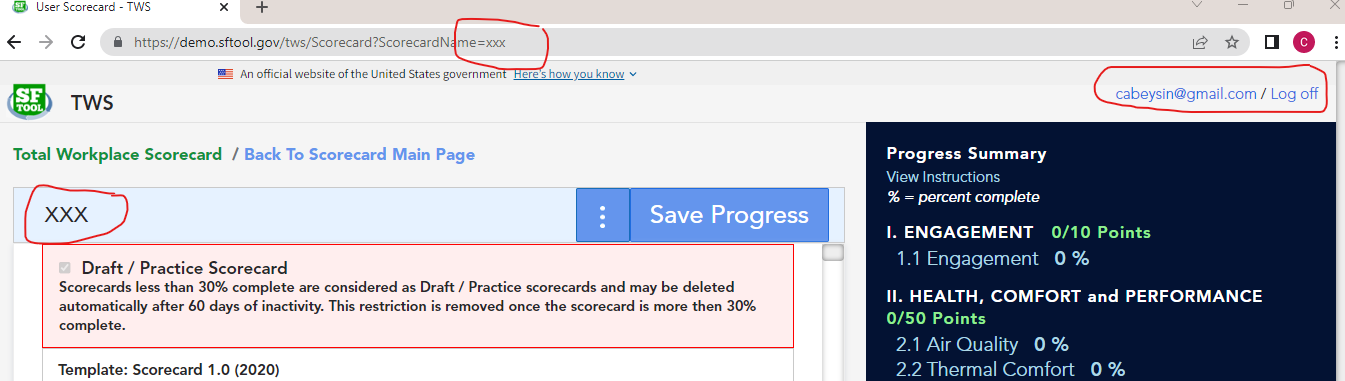
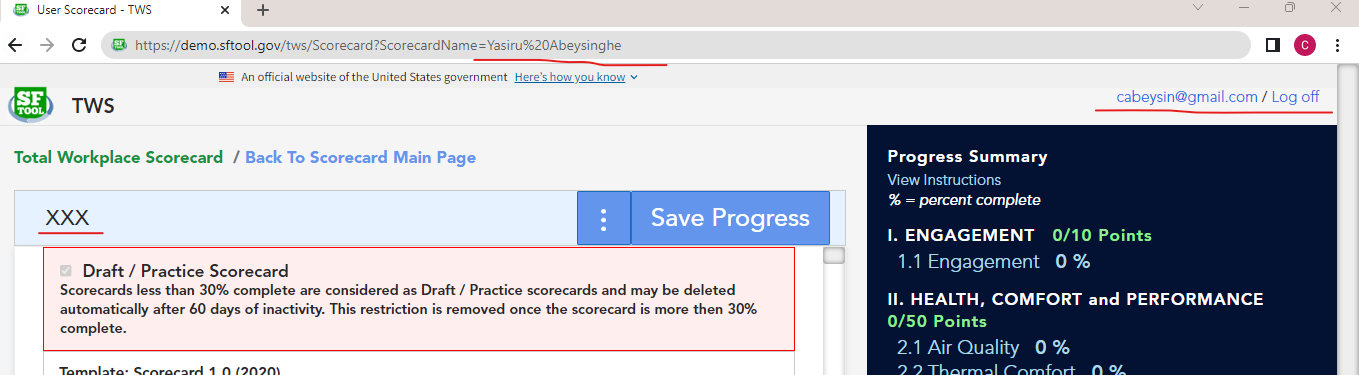
Initially he has submitted the details to a single account (namely testdsfdfsf) and later he had logged into the other account (let’s consider it as the attacker’s account) and replaced the URL with https://demo.sftool.gov/TwsHome/ScorecardManage/testdsfdfsf “. And he was able to view the previous account without any issue and even he was possible to make alternations and edit and download the contents of that profile. And later after logging into the testdsfdfsf account he was able to recognized the changes that have been done to the account initially.

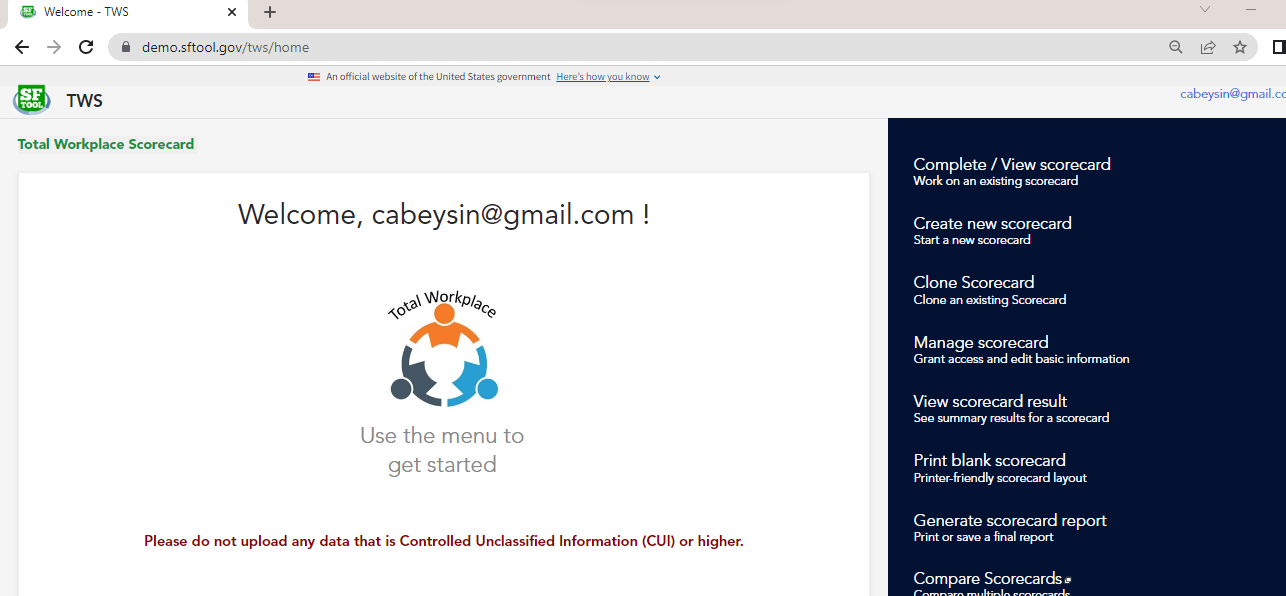
* **Repeatability:** Not repeatable

Initially I created yasiruabeysin@gmail.com account within the respective site using.



* And again, created another account using [cabeysin@gmail.com](mailto:cabeysin@gmail.com)
* And tried to log to Yasiru Abeysinghe scorecard using “<http://demo.sftool.gov/tws/Scorecard?ScorecardName=Yasiru%20Abeysinghe>”.



* But it was unable to login to that account since it redirected to the home page
* **Solutions**: -
* It always recommended as a best practice to validate the user inputs gained via HTTP request using the whitelisting in order to verify whether requested values meet the expectations of the application. This can be done by parameterizing the maximum and minimum length, characters, patterning the set of expected inputs.
* Implementing proper access control mechanisms which defines the intended functions to be done by authorized users. This can be performed by regulating the applications to check the authorization databases and access control lists per each user request to verify whether the particular user/subject is permitted to perform a certain operation on the resources/objects. And also, the resource values in the session should be checked with stored values on the server at defined timed intervals.
* And also, these sorts of vulnerabilities can be prevented by implementing defense- in-depth approaches by thoroughly auditing and doing test controls, denying access by default, using single application-wide mechanisms in order to enforce access controls.

# Blind SSRF

* **Program**: - Hacker one (https://hackerone.com/reports/1220688)
* **Area**: - Blind Server-side request forgery
* **Bounty hunter**: -

The above vulnerability is been identified by S. Rahul who is an information security analyst working at NUK 9 Auditors and involved in security management and operations, security testing, vulnerability scanning and bug bounty hunting having well-rounded 2+ years experience within the IT profession. And also, he is involved in auditing and implementing GRC standard such as GDRP, ISMS, BCMS, DSS etc.

LinkedIn profile: <https://www.linkedin.com/in/7srambo/>

* **Organization and Assets**: -

This is a website <https://mtngbissau.com/> handled by MTN group whom the Africa’s largest mobile network operator. So, this can be considered as the web interface where lots of services are being handled on MTN Guinea and also connected with numerous backend resources and services.

* **Vulnerability: -**

This a vulnerability that allows an intruder to force on a server-side of a web application to make HTTP requests to reach towards a location where users aren’t permit to make requests. So, by this way attackers can connect with the internal-only services which belongs organization handling the web application as well can get connected with external systems linked with the organization.

So specifically, when it’s coming to blind SSRF though back-end HTTP requested are been easily sent via the application the returning responses can be identified by analyzing the front end.

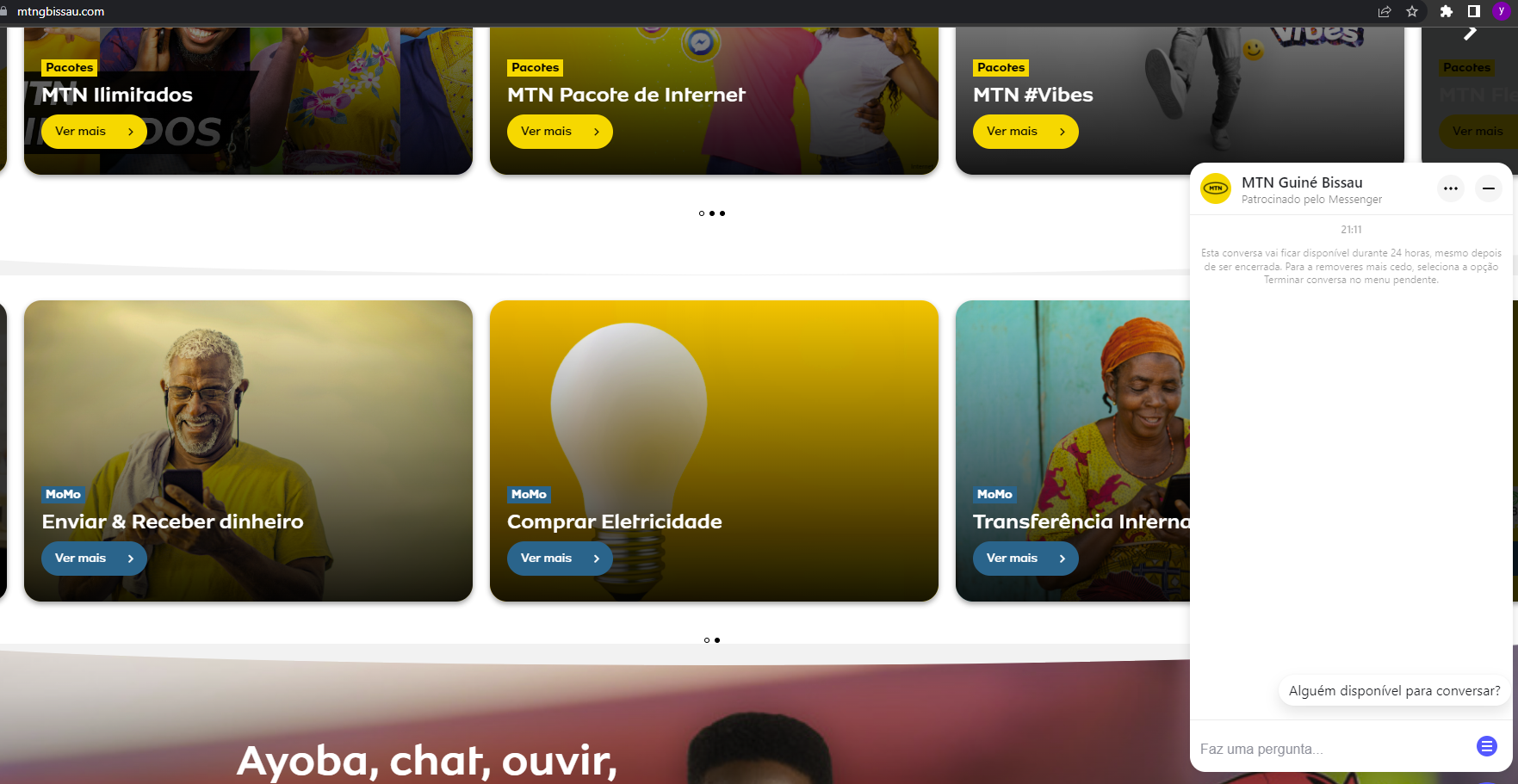
* **How vulnerability is discovered**: -

Here the hunter has used the OAST technique (Out-of-band Application Security Testing) in order to detect whether website is vulnerable to SSRF. This can be easily performed using the Burp Collaborator which is function provided by the burp professional edition.

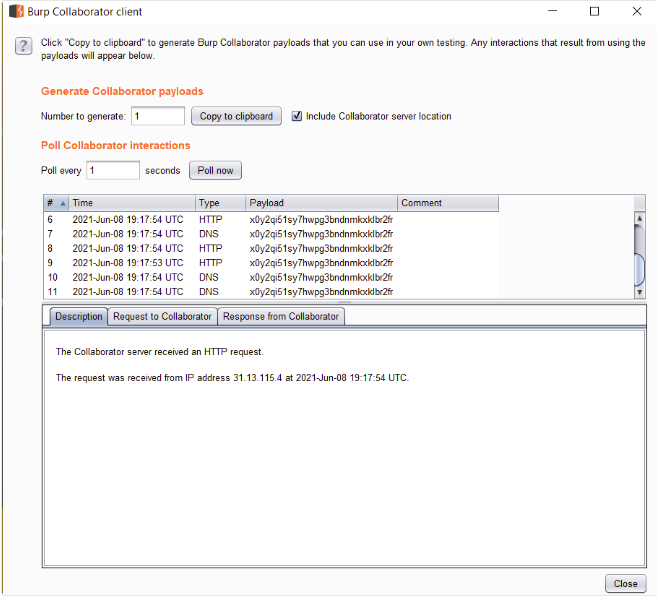
Here by the burp collaborator unique domain names are been generated while after sending the payload to the application we can monitor the interaction of the backend servers with that collaborator generated domains. So, the responsiveness of the web application for SSRF can be easily identified whether incoming HTTP requests are been reached from the application.

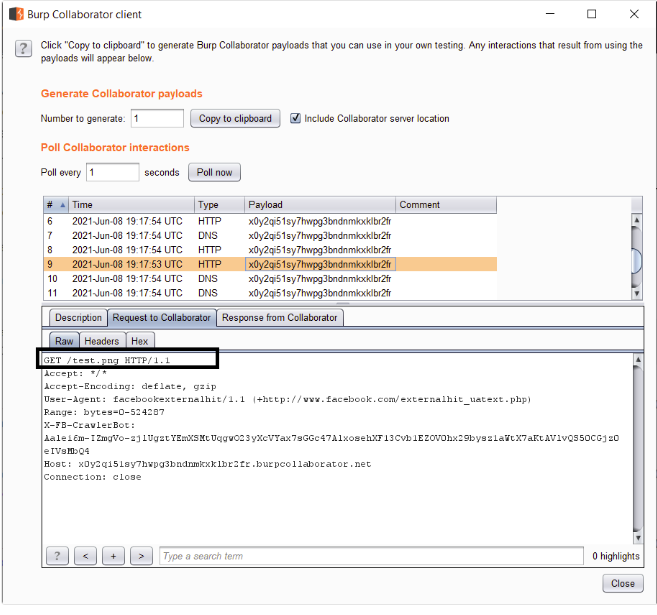
* **How it is performed: -**

After navigating to the website, a messenger box is available and the URL generated using the Burp collaborator is been pasted on it and sent.



After clicking on poll, he has been able to see the HTTP and DNS interactions of the web application resources with collaborator generated domains.

So here the hunter has been able to gain the internal IP and some other information of the website.



* **Solutions:**

In order to prevent these sorts of vulnerabilities which are server-side vulnerability defense in depth should be approached from the server end which includes network layer defenses as well as application layer defenses.

* So, the web the application should enforce the URL schemes, port and destinations where users are been allowed to access.
* User inputs should be properly sanitized before been sent towards the back-end servers and the services. In order to implement this whitelisting and using of regex is not recommended since a well skilled attacker can intercept this sort of a security barrier easily by encoding the inputs which are been sent.
* No responses which mainly include private information should be sent towards the end user in raw format.
* HTTP redirections to internal as well as external services should be disable.
* **Alternative methods:**

In addition to the portswigger collaborator web application vulnerability scanners such as WAPITI, Acunetix, arachini, w3af can be used.

# 3. SQL Injection

* **Program**: - Hacker one (https://hackerone.com/reports/1081145), <https://karmainsecurity.com/impresscms-from-unauthenticated-sqli-to-rce>

https://github.com/ImpressCMS/impresscms/blob/48af29c6b8150fbf4220bb5cc4f3c57bcd818384/misc.php#L181

* **Area**: - Injection
* **Bounty hunter**: -  
  This vulnerability is detected by Egidio Romano (EgiX) who is graduate of University of Catania on BSc in computer science. He is highly passionate on computer and web application security. Currently he is working as IT security consult performing code reviewing and penetration testing on web applications.

LinkedIn profile: <https://www.linkedin.com/in/romanoegidio/>

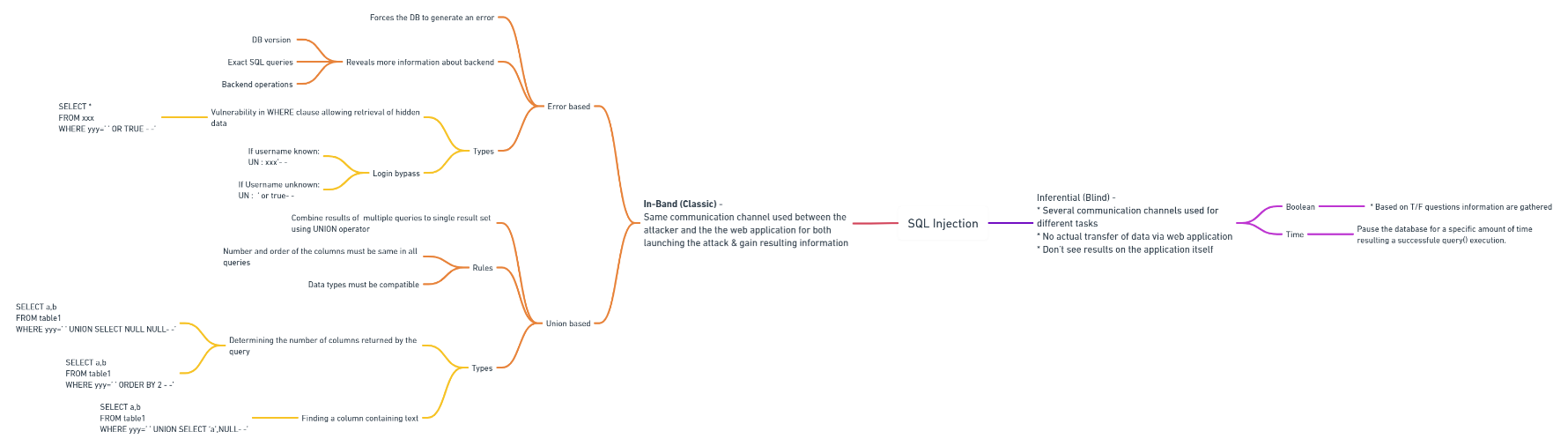
* **Organization and Assets:**

This is an open source Content Management System namely ImpressCMS which can securely manage web contents. This tool is been widely used by range of community users and enterprises who need to used it as a blogging tool.

In addition, this application has been implemented with an inbuilt security mechanism namely Protector which provides an overall security against XSS and SQL injections up to a certain extent.

* **Vulnerability:**

Here the hunter has been capable of identifying 2 vulnerabilities within the location /include/findusers.php page of the systems namely SQL injection and access control by pass followed by RCE (Remote code execution).

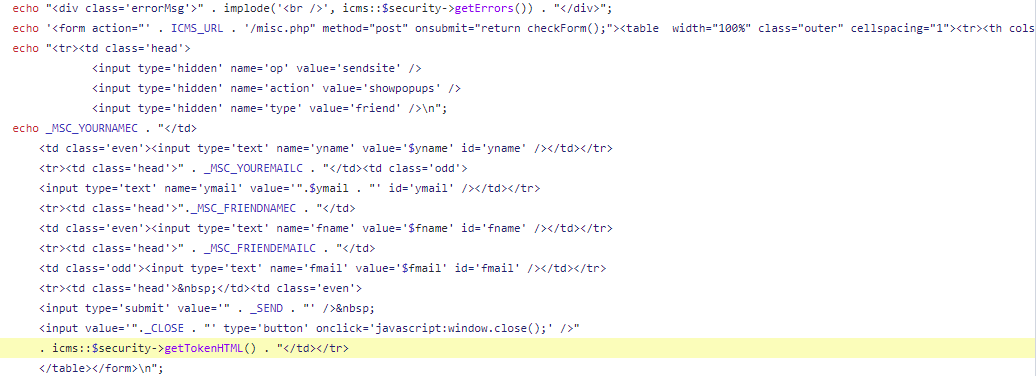


<https://whimsical.com/sql-injection-3DEfUfLfPnaYFJUyS9ewWG>

* **How vulnerability is discovered: -**

According to this portion of code this is checking whether the provided token is a valid one or not. But here the issue is that the system has no methodology to ensure whether the current user is properly authenticated or not. In simple terms even, an attacker can provide a valid token by resulting the unauthorized access to the attacker.

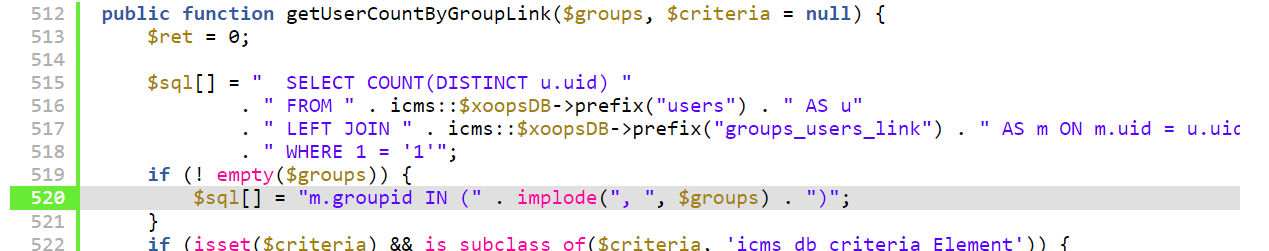
So, within the script it has been identified that the valid security tokens are been easily generated without any user authentication as below,

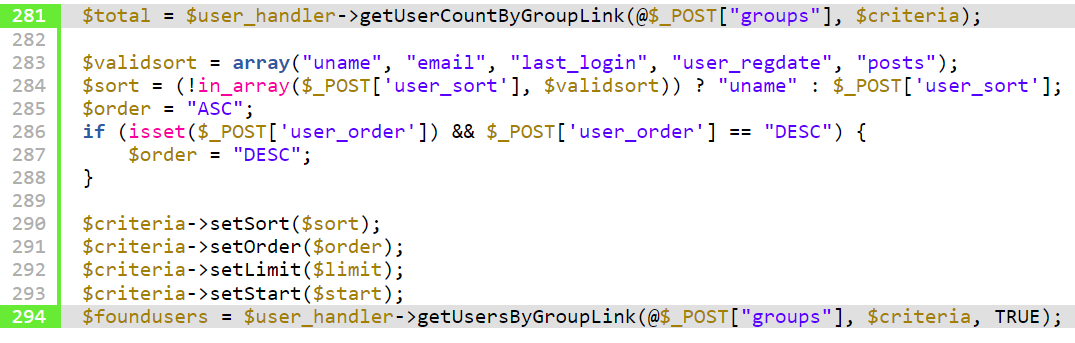


🡨 /misc.php

And also, the hunter has identified another set of implementations within the SQL query segments of the php code which lacks proper validations that can be easily vulnerable for SQL injections.

So according to this it is clear that an attacker who is remotely authenticated can execute SQL queries to disclose sensitive information in the user’s database by implementing attacks such as Boolean-based SQL injection.





* **How it is performed:**

As it was previously mentioned that a security mechanism namely Protector is been implemented which includes some anti-SQL injection methods some suspicious elements such as SELECT, UNION are been detected and blocked. But in the same time the application has used PHP Data Objects (PDO) to access databases using php which provides the capability to execute stacked SQL queries which are been separated by semicolons.

So first of all, the hunter has tried to understand the table prefixes in order to create a new user. So, in order to do that attacker has inserted the following query,

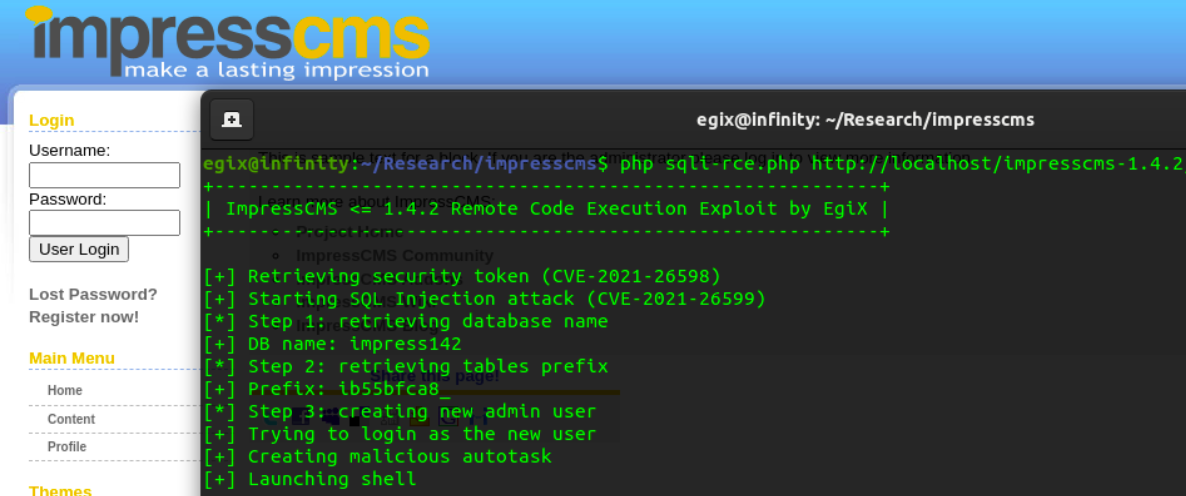
**AND ORD (SUBSTR (SELECT table name FROM information\_schema. tables WHERE table\_schema=ímpresscms’ AND table name LIKE ‘%users’),1,1) =XX #**

Since protector module has detected the suspicious terms such as SELECT that attacker need to bypass it signing variables in hex format using statements such as SET, PREPARE, EXECUTE. So here the hunter has used as follows.

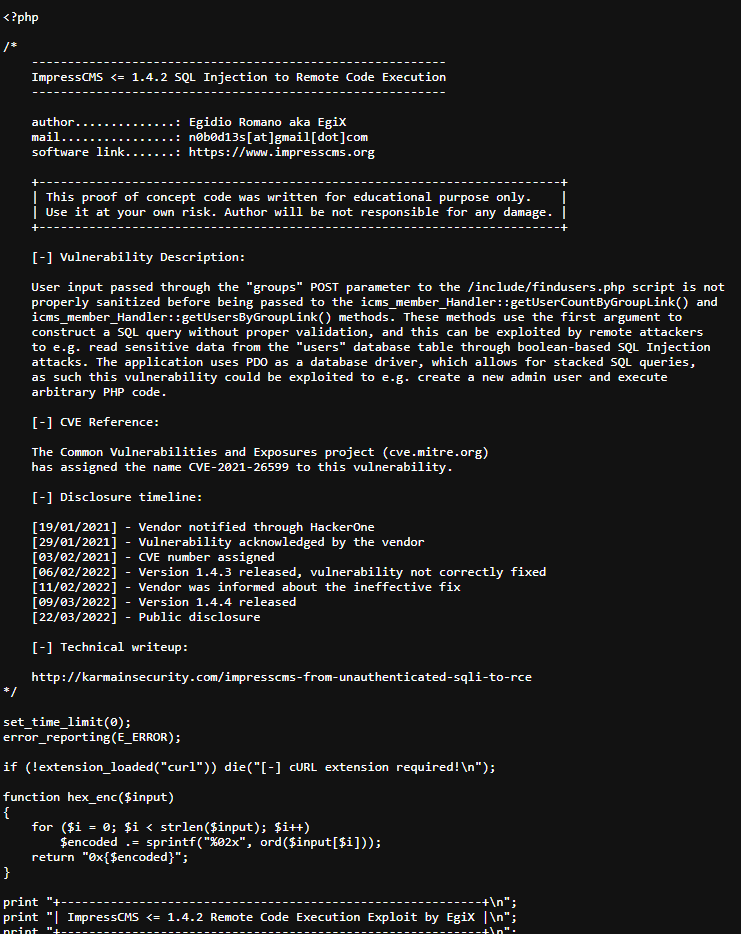
  
And ultimately a new user created,



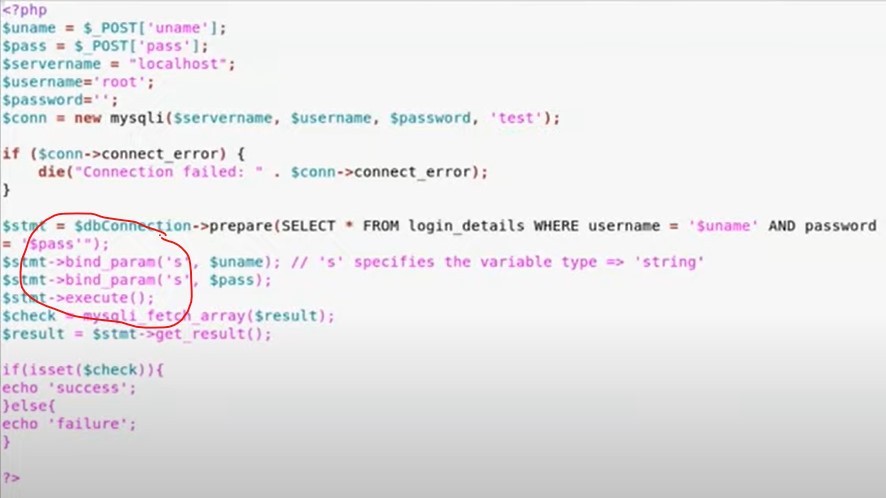
And now that attacker has the capability to RCE as a result of unauthorized SQL injection. First of all, a valid token is been gained for the newly inserted users using the methods implemented in /misc.php? as mentioned before. And by using that token unauthorized access to the /include/findusers.php is been gained. And by implementing some additional SQL injections an admin user is been created (fetch the database names using Boolean-based injection, fetch the tables by passing the protector module using time-based injection). And finally, now it is possible to login as admin to the system and execute arbitrary PHP in order to gain RCE capabilities as follows.



Arbitrary php code used:



* **Solutions:**



* So most of the SQLIs takes place since the web application is unable to consider the closing ‘(single quotation) as a string as similar as the rest of the characters which are been entered. So, in order to prevent that bind\_param () method can be used which bind all the parameters of the user input as consider the whole user input as a single string
* And specifically, when it comes to this web application proper authorization mechanisms should be implemented as front line defense before validating the tokens.

# 4. Violation of Secure Design Principals

* **Program**: - HackerOne (<https://hackerone.com/reports/1428385>)

https://infosecwriteups.com/mail-server-misconfiguration-leads-to-sending-a-fax-from-anyones-account-on-hellofax-dropbox-bbp-aab3d97ab4e7

* **Area**: - A04:2021-Insecure Design, A05:2021-Security Misconfiguration.
* **Bounty hunter**: -   
  This vulnerability is been detected by Sayaan Alam who is security researcher and red-team member who is working for Synack, California. He has done bounty hunting for nearly 5 years contributing 100+ companies in order to fix the critical security issues via Synack and bug bounty platforms such as Bug Crowd and HackerOne.

LinkedIn profile: <https://www.linkedin.com/in/sayaanalam/>

* **Organization and Assets: -**

The vulnerable asset is the HelloFax-Dropbox BBP option which is provided by the dropbox which is a leading file hosting service that provides services on file sharing, storing which helps in project collaboration protecting the shared workspace.

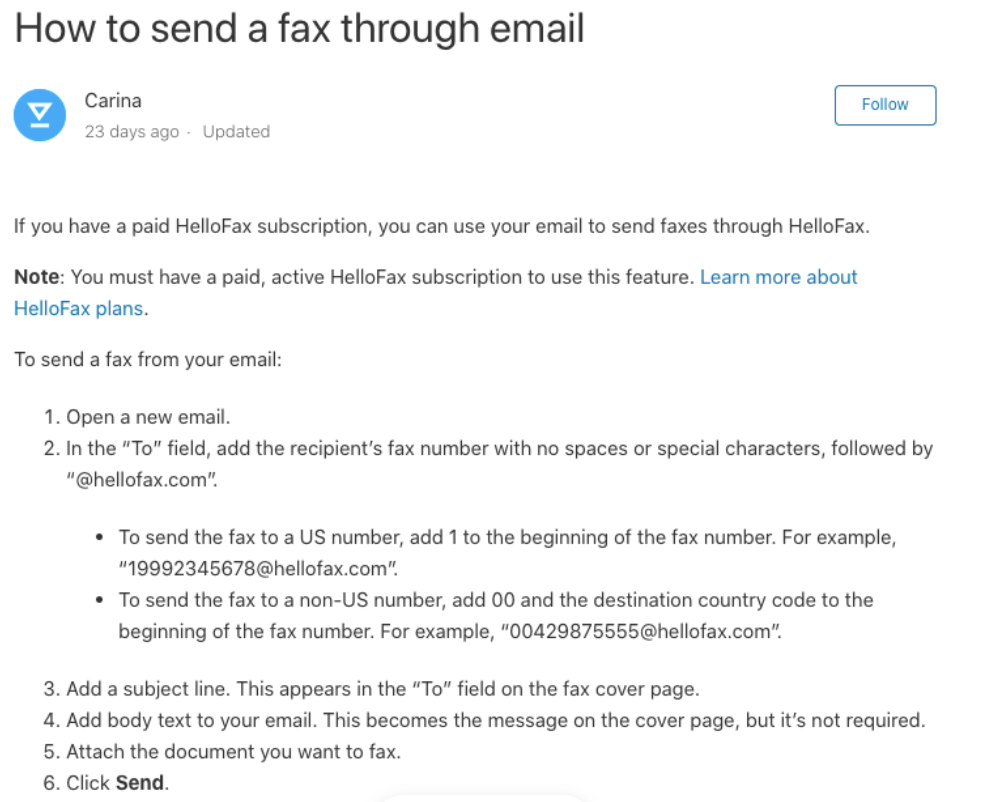
* **Vulnerability: -**

Here the hunter has found that the Dropbox BBP allows users to send a fax by using their emails even without logging into their accounts managed by HelloFax which can be considered as a severe security misconfiguration of mail server.

Through this vulnerability an attacker can enter a victim HelloFax line number and send forging email requests to a HelloFax users by using a third-party mail server.

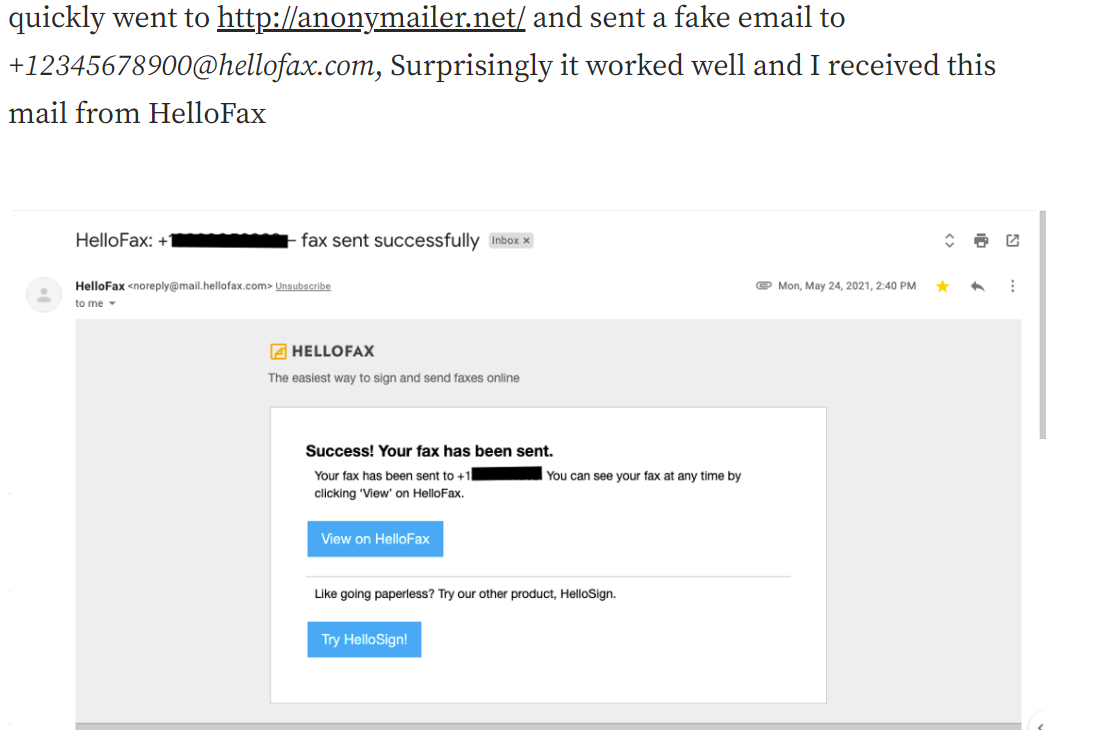
Based on this the HackerOne had decided to categorize this in high severity range since this vulnerability has capability to trick application even by using valid HelloFax customer email and HelloFax number.

<https://whimsical.com/violation-of-secure-design-principals-JtWmqGrJQz3KAjKBfKw7RP>



* **How vulnerability is discovered and exploited: -**

Firstly, the hunter has sent a fake email to HelloFax user by using [faxnumber@hellofax.com](mailto:faxnumber@hellofax.com) as the email address and putting a victim email address to the FROM field of the email.



Here the expected target successfully even without verifying the authentication of the sender while considering the sender as a legitimate user.

Ultimately the dropbox BBP application has process the email and performed the required actions which had finally achieved via authentication bypass.

* **Repeatability:** Not repeatable
* **Solutions:**
* Performing threat modelling and vulnerability assessments on each of the unit/integration stages of the software development life cycle.
* Implementing proper access control mechanisms which differentiates tasks which each user can perform.
* Implementing proper security controls in way which ensures the defense-in-depth.

# 5. Insecure Deserialization

* **Program**: - HackerOne (https://hackerone.com/reports/838196)
* **Area**: - A08:2017-Insecure Deserialization

A08:2021-Software and Data integrity failures

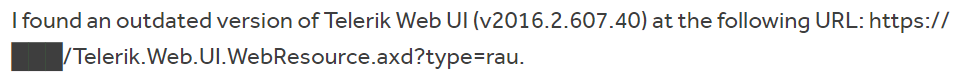
A06:2021-Vulnerable and Outdated Components

* **Bounty hunter**: -

This vulnerability is been identified by an independent security researcher named Paolo Arnolfo who have been awarded with 2020 VDP Researcher of the year. He is well experienced in doing bug bounty hunting since self employed as a hacker in platforms such as HackerOne, Bugcrowd, Synack.

LinkedIn profile: <https://www.linkedin.com/in/paolo-arnolfo/>

* **Organization and Assets: -**

****This particular path is belonging to a website which is been managed by Department of Defense at United States.

* **Vulnerability: -**

Here this website is using an outdated Telerik Web User Interface (v2016.2.607.40) which makes an attacker possible to gain complete remote code execution through uploading arbitrary files on the backend server and finally leads to deserialization of untrusted data.

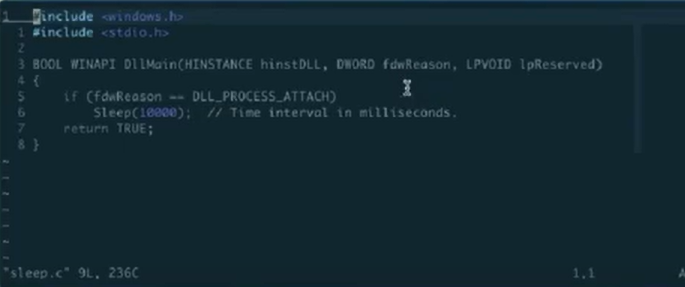
Associated CVEs : CVE-2017-11317, CVE-2019-18935

The path way to identify this sort of vulnerability is been started by the CVE-2017-11317 vulnerability which is unrestricted file upload via weak encryption. Here the “AsyncUploadHandler” of “RadAsyncUpload” which is a single/multi file upload control which can provide features such as integrated validation, automatic file saving, uploading file asynchronously etc. Through this vulnerability which is been identified in 2017 attacker could be able to craft request on file upload on “/Telerik.Web.Ui.WebResource.axd?type=rau” of “rauPostData” POST parameter if the encryption key which is “PrivateKeyForEncryptionOfRadAsyncUploadConfiguration” remains unchanged.

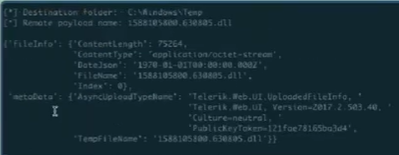
By closely analyzing this vulnerability Markus Wulftange has identified that the rauPostData contains serialized configuration object together with its object type. So here intruders have the possibility of deserialize the objects which means convert JSON-formatted strings in to objects using the JavaScriptSerializer.Deserialize() method if gadgets are been set in a way to support remote code execution of arbitrary codes. So ultimately attackers can submit POST request which specifically specifies the RCE gadget instead of submitting the “AsyncUploadConfiguration” as usual with rauPostData.

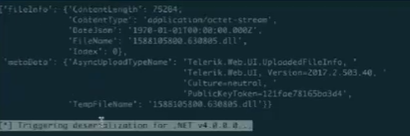
* **How vulnerability is discovered**: -

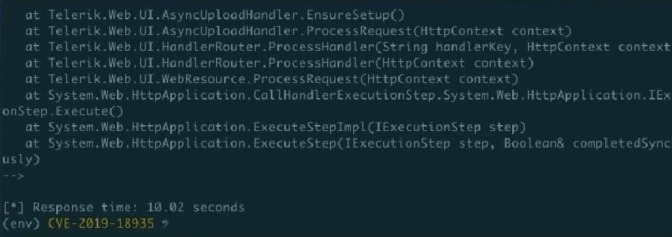
Here the hunter has initially checking whether it is possible to do an arbitrary code execution within the respective platform by compiling a payload which makes application sleep for 10s.



Here we can see that the payload is been successfully uploaded to the back end waiting for the response.



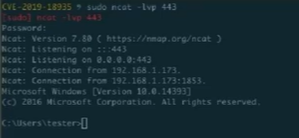




So finally, the response is been received indicating that the application has been slept for 10.02 seconds.

* **How it is performed: -**

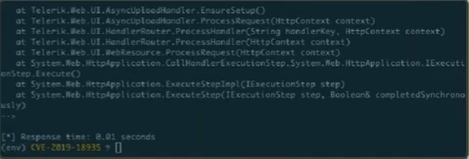
So now the hunter has checked to initiate a reverse shell TCP connection back to the intruder server in the similar manner.

Firstly, the hunter has setup a netcat listener

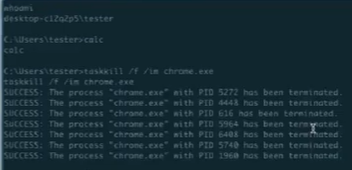
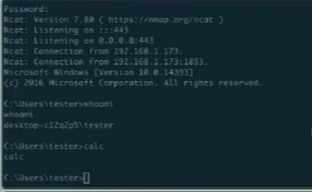
And the payload is been sent to the application



So finally, an interactive remote shell have been gained,





And finally, hunter has gained the ability to do remote code execution.



* Repeatability: Not repeatable

Here the respective website is not properly disclosed in the HackerOne page based on the security concerns while assume that the respective security patches are been added since it’s a web site handled by the U.S government and in the comments, they have mentioned that they have identified the criticality of the vulnerability since they have vulnerable and outdated components.

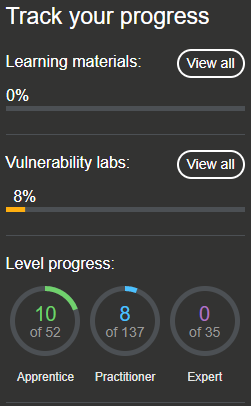
* **Solutions: -**

The best option is to upgrade version of Telerik for ASP.NET Ajax to R1 2022 SP1 which is the latest version.

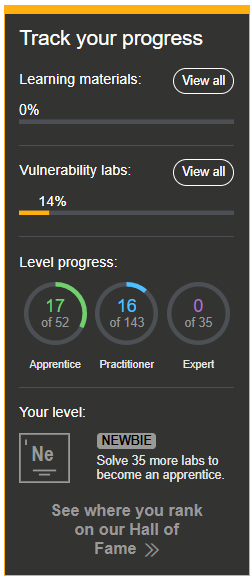
# 6. Video explanation

<https://mysliit-my.sharepoint.com/:v:/g/personal/it21022142_my_sliit_lk/EZa6l_ySiDBOkATE4IEBp5UBXMJUBN6PLmdr9XvnyeL7qA?e=44yN2O>

# 7. Portswigger progress



Previous status



Latest status