Information Sciences Letters
An International Journal

http://dx.doi.org/10.18576/isl/120319

Enhance Linux Security Server Misconfigurations and hardening Methods

Belal Ayyoub[1,*], *Ashraf Abu-Ein*[2], *Bilal Zahran*[3], *Jihad Nader*[1] *and Obaida Al-Hazaimeh*[4]

[1]Department of Electrical Engineering, Faculty of Engineering Technology, Al-Balqa Applied University, Amman, Jordan
[2]Department of Electrical Engineering, Al-Hoson University College, Al-Balqa Applied University, Irbid, Jordan
[3]Department of Engineering and AI, Al-Salt Technical College, Al-Balqa Applied University, Al-Salt, Jordan
[4]Department of Computer Science and Information Technology, Al-Hoson University College, Al-Balqa Applied University, Irbid, Jordan

Received: 4 Aug. 2022, Revised: 20 Oct. 2022, Accepted: 25 Oct. 2022
Published online: 1 Mar. 2023

Abstract: The calamity begins if an attacker successfully compromises a system and gains access to high-level privileges. This paper presents and addresses a vulnerable Linux server with typical flaws and configuration errors. This paper aims to show how these widespread vulnerabilities might be used by an attacker to compromise the server. In order to prevent building and setting up a Linux server with risks and low security, as well as to guarantee the integrity and confidentiality of user and customer information, this paper instructs aspiring system administrators and developers on how to avoid making such errors in their initial configuration for this servers set of examples.

Keywords: Linux server, Metasploitable, Vagrant file, Docker, Netcat, NMAP

1 Introduction

Computer and server security can be improved through a process known as ”system hardening,” which involves removing or reducing potential entry points and other points of vulnerability. It’s a method of defense against cyber attacks that entails plugging the vulnerabilities that hackers use to access private information [1,2,3,4,5,6,7,8,9,10]. The process of closing loopholes and disabling unused services in an effort to make a system more resistant to intrusion is called ”system hardening” [11,12,13,14,15]. Several research studies have been conducted in the topic of hardening servers and utilizing penetration testing [15-19].This paper aimed to show how an attacker can leverage weak server implementations to compromise and ex-filtrate PII (Personally Identifiable Information) data. A short description of a common vulnerable and penetration testing machines:

1.1 Metasploitable 2

Metasploitable is a Linux virtual machine that has been made purposely vulnerable so that it can be used for penetration testing, security tool testing, and training purposes [4,5,6]. This server discusses the following misconfigurations and vulnerabilities:

1. **vsFTPd 2.3.4 backdoor:** There is a hidden backdoor in the coding of this version. With the backdoored version, an attacker can gain access to the server by sending a username that ends in the sequence and then using the listening shell on port 6200 to carry out their orders.
2. **UnrealRCD 3.2.8.1:** This version contains an undiscovered backdoor triggered by sending ”AB” followed by a system command to any listening server port.
3. **Web services to practice web vulnerabilities exploitation:**

* Corresponding author e-mail: belal_ayyoub@bau.edu.jo

–tikiwiki-old
–phpMyAdmin
–Mutillidae (NOWASP Mutillidae 2.1.19)
–dav (”WebDAV”)
–tikiwiki (”TWiki”)
–DVWA (”Damn Vulnerable Web Application”)

It is quite hard to fairly compare between Metasploitable 2 and our machine even though their purpose is the same, to gain and expand one's knowledge about security vulnerabilities and the risks that are caused by them and to practice and sharpen one's penetration testing and security skills.

Metasploitable 2 is more focused on providing as many vulnerabilities as possible to practice on them all but without having approaches or scenarios that are close to what's happening in real life and it's more beginner friendly than our machine.

1.2 Cap machine

Cap is a virtual machine that contains several vulnerabilities which could let an attacker gain administrative privileges (root privileges) on the machine with simple yet high-impact vulnerabilities, this machine is provided from HackTheBox platform, it's an online penetration testing training platform that is aimed for intermediate and high-skilled penetration testing students and security engineers.

The attacker can then exploit this to execute a python command that can let them maintain privileged access by manipulating its own process UID to 0, which is the root's UID, thus having root privileges on the system.

While Cap and our machine have the same purpose and a similar approach; which is gaining a foothold by exploiting a vulnerability that is exposed to the public, logging in as one of the users on the machine and finding a way to escalate their privilege to root, which means that they have full access on the machine, Cap machine have a much simpler approach than ours.

Compared to our machine, Cap is an easy machine with a scenario that is rare to happen in real life, unless there were a really lazy system administrator that would expose their server and production environment to danger in such a way.

1.3 H4cked machine

H4cked is a machine that shows the impact of having bad password policies such as using weak passwords and reusing the same password on multiple accounts/services. Yet using weak and simple passwords is a recipe for data breaches, account takeovers, and other forms of cyberattack. This machine is provided by TryHackMe, which is a website that teaches you by doing instructions and lessons unlike HackTheBox, where you're on your own.

The first thing the attacker did was simple port scanning using the NMAP tool. He found three open ports 21, 22 and 80. Then the attacker tried to crack the FTP password using Hydra tool, which is a famous tool used to crack passwords by brute-forcing. He used the built-in wordlist “rockyou.txt” which contains 14,341,564 unique passwords, used in 32,603,388 accounts. He managed to get a password for the user ”jenny”.Now he will login into the FTP server.

Now after logging in to the FTP server, the attacker needs to get a reverse shell to get access to the server. So he will upload a malicious file (shell.php) which is a script that will open an outbound TCP connection from the webserver to the attacker IP address. Now he had to open the web server homepage to execute the script. Now he got a reverse shell.

Now he has access to the server, but it's limited because it's a webserver user (www-data). So, he tried to switch to the root user, and luckily the root password was the same as the FTP password. This shows the high impact of using the same password on multiple accounts/services. While H4cked and our machine have a similar approach, which is gaining a foothold by exploiting a vulnerability that is exposed to the public, logging in as one of the users on the machine and finding a way to escalate their privilege to root, which means that they have full access on the machine, this machine is mainly targeted for beginners who just started their cybersecurity learning path by teaching them how to penetrate their way in by asking them questions that will guide them to compromise the machine while they're learning.

Compared to our machine, H4cked is a beginners-targeted machine with a situation that you mostly won't find in real life since the whole environment had been set up for learning purposes on basic vulnerabilities to get students started, and there aren't as many common vulnerabilities as there are on the internet in this machine.

2 Settng Up the Environment

This chapter will show how the vulnerable and misconfigured environment is made; all of the files are in the project's Github repository.

2.1 Vagrant

We used Vagrant, is an Open-source software for developing virtual software development environments (Journal, n.d.) to create our environment in a virtual machine with apps and services automatically installed and configured based on our needs for this thesis to demonstrate the risks of running vulnerable or misconfigured apps and services. By simply running "vagrant up" it will start creating the virtual machine based on the provided vagrant file in the current working directory by default.

It will start by downloading the operating system if there's no available image does already exist for Vagrant to use, it will then boot up and install the operating system and configure it. Here we configure Vagrant to assign the IP 192.168.77.105 for the virtual machine to use, and to map and sync our local folders "configuration" and "Setup-Project" with the virtual machine

```
# Create a public network, which generally matched to bridged network.
# Bridged networks make the machine appear as another physical device on
# your network.
config.vm.network "public_network", ip: "192.168.77.105"

# Share an additional folder to the guest VM. The first argument is
# the path on the host to the actual folder. The second argument is
# the path on the guest to mount the folder. And the optional third
# argument is a set of non-required options.
config.vm.synced_folder "./configuration", "/.Setup-Project"
```

Fig. 1: Network and Folders settings in the Vagrant file

And finally, we configure how will Vagrant provision the virtual machine, here we are configuring it by running a bash script in which it will setup and configure all of the vulnerable and misconfigured apps and services for this thesis.

2.2 Configuration

In the configuration folder there are two other folders that are needed to create the vulnerable server, they contain the

2.2.1 Server Folder

The server folder contains those files that are needed for the applications and users to work as intended, note that all of them are misconfigured and using them will compromise the server.

- **–backup.sh:** This file is being run by a cron job from by user *baker* to take a backup of all the files inside the source directory and compress it to a file named *webapp.tgz* located in the destination directory.
- **–configs.sh:** This file will run after booting up the machine (by *Vagrant*) from the root user and it'll setup requirements, configurations, and network settings for the applications to run on the server.
- **–debug.service:** This is a service file that is owned by the developers group and can be used by the user *baker*.
- **–restart_services:** This file is the user-privilege based restart service script and its purpose is to prevent users from restarting services that are managed by the root user and permit restarting if otherwise.
- **–sudoers:** This file is a pre-configured *sudoers* file, note that the user *nabil* can run *docker* with *root* privileges as well as everyone in the *developers* group can execute the file *restart_services* with root privileges.

2.2.2 Folder webserver

The web server contains the WordPress application, which has the user interface for a security company, and the vulnerable plugin, which is affected by an LFI vulnerability which we will use later to exploit the server.

And now we will see how to configure the webserver by doing these steps:

Step 1: Installing Apache

To get started, refresh the cache of the package manager. If you've never used sudo in this session before, you'll be prompted for your password before being granted access to apt package management.

Step 2: Installing MySQL

Your website's data can't be stored and managed without a database system, which you'll need now that your web server is up and operating. MySQL, a database management system, is widely used in PHP contexts.

Step 3: Installing PHP

The next step is to set up PHP, the server-side used to run the code that renders dynamic web pages for the user. We require php-mysql, a PHP extension that bridges the gap between PHP and MySQL-based databases, in addition to the core PHP distribution. Additionally, we require libapache2-mod-php to make Apache capable of processing PHP scripts. (See Figure 2)

Fig. 2: sudoers file

Step 4: Virtual Host creation

We were able to use a custom domain because we set up a virtual host, which isolates configuration details and allows many domains to share a single server.and it will be threats.int. (See Figure 3)

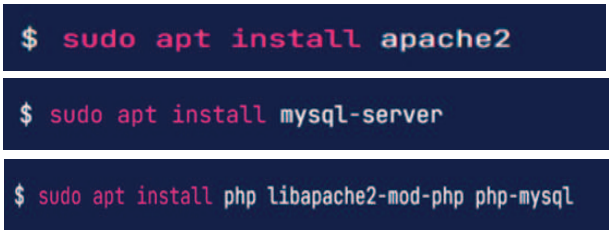

Fig. 3: the first three steps of web server

3 Implementation and Results

3.1 Enumeration and information collecting

The term "enumeration" refers to the procedure of obtaining information about a system, such as its user names, machines, resources, shares, and services. At this stage, the attacker has established a live connection to the system and is gathering intelligence via targeted requests. In the System gaining phase, the collected data is used to locate and exploit the system's weaknesses.(Campus, n.d.). (See Figure 4). For Nmap scan results

Nmap scan

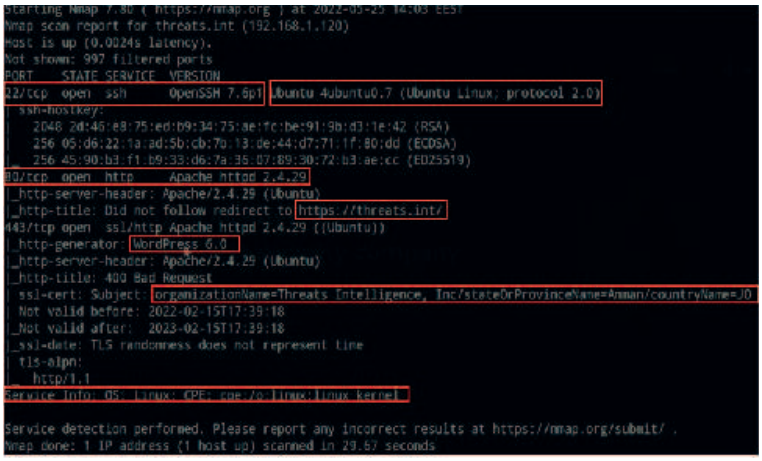

Fig. 4: Nmap scan results

Nmap "Network Mapper" is a free and open source utility for network discovery and security auditing. (NMAP, n.d.). We'll use *"nmap"* to get information about the server, including the open ports and the services that are running on it. This is the "nmap" command that we'll be using: (See Figure 4)

–-sC: Performs a script scan using the default set of scripts. (NMAP, n.d.).
–-sV: Extensive version detection. (NMAP, n.d.).
–-T4: Set timing template (higher is faster) so more threads (NMAP, n.d.).
–-p-: To scan all ports. (NMAP, n.d.).
–192.168.1.120: The IP of the server we want to gather its information.
–-oN: Output scan in normal text file. (NMAP, n.d.).
–NmapScan.txt: The name of output file

In figure 4 We can see that the server is running Linux Ubuntu OS, an Apache web server with the version 2.4.29, WordPress version 6.0. And we can also see that the organization name is *"Threats Intelligence"* located in Amman, Jordan. We'll start by enumerating more on WordPress for vulnerable plugins since it's more interesting than other findings.

WPSCAN

WordPress security pros and blog owners can use the free WPScan CLI tool to scan their sites for vulnerabilities at no cost for non-commercial purposes. WPScan is a command-line interface that scans for 28,737 known vulnerabilities in WordPress..(Scan, n.d.). We'll use this command to scan for any vulnerabilities on WordPress website.

-e ap:0

This option is for enumerating all plugins (Scan, n.d.). We can see some interesting findings; we already know that the web server is Apache but there are other information that *wpscan* have found including *XML-RPC, Astra* theme and a vulnerable plugin named *wp-with-spritz*, notice that it says that it's on the latest version but the last update was on 2015 which was a long time ago so it's old and outdated now.(See figure 5)

There's a file inclusion vulnerability in the plugin, and they are vulnerabilities often affect web applications that rely on a scripting run time and can allow users to read or execute code on the victim server (Sec, n.d.), we can exploit an LFI (Local File Inclusion) on this server. See Figure 6 and 7)

3.2 Initial Compromise

As soon as an attacker is able to execute malicious code on one or more systems, whether by exploiting a vulnerability in an Internet-facing system or some other method, they have successfully compromised those systems.(Center, n.d.). The initial compromise here will be due to the LFI vulnerability in the *wp-with-spritz* plugin.

Exploiting LFI to read admin credentials

Attackers can use Local File Inclusion (LFI) to run or expose files on a web server by tricking a web application into doing so. A loss of integrity (LFI) assault can result in the disclosure of private data. (Sec, n.d.). According to the exploit

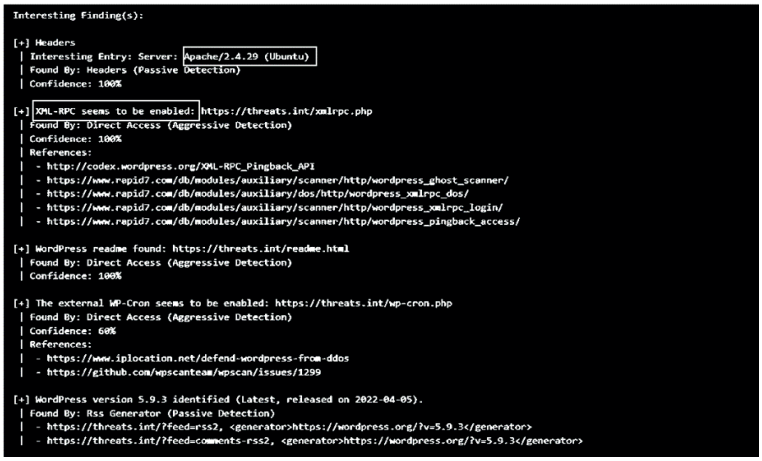

Fig. 5: WPScan tool results (1)

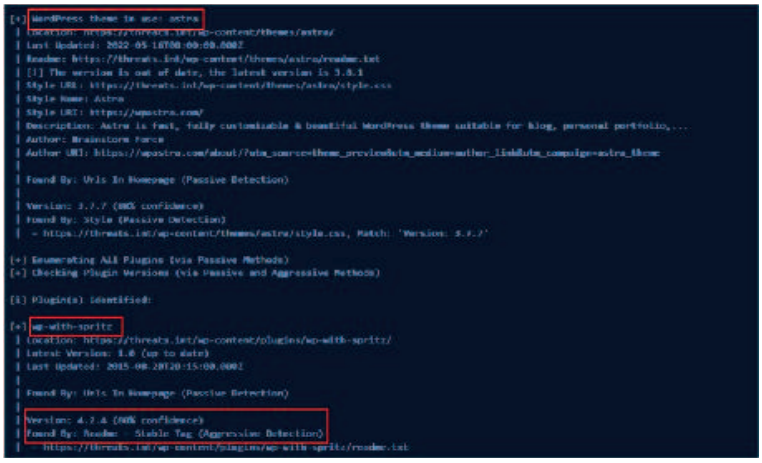

Fig. 6: WPScan tool results (2)

we've seen in the previous figure; we can test the existence of the LFI vulnerability in the *wp-with-spritz* plugin by reading the */etc/passwd* file located in the server by going to this path on the website: (See Figure 8)

Checking if the plugin is vulnerable to LFI

We'll do this by issuing a *curlcommand*,which is a tool for transferring data from or to a server. It supports a variety of protocols including HTTP and HTTPS (Se, n.d.). The curl command would be as follows: (See figure 9)

-k: This option makes curl skip the certificate verification step and proceed without checking. (Se, n.d.).

The *curl* command will issue an HTTP GET request to the specified path to read the file named *passwd* located under the */etc* directory on the server. It also can read the file on a web browser instead of using *curl* by requesting the same file path in the address bar: (See Figure 10)

We can see that we've successfully were able to read the */etc/passwd* file, thus the plugin *wp-with-spritz* is vulnerable and can allow attackers to read arbitrary files on the server that they shouldn't be allowed or able to read.

Reading wp-config.php

The *wp-config.php* file is crucial to the operation of your WordPress installation. This file is located in the WordPress file directory root and contains your website's base configuration details. (WordPress, n.d.). By exploiting the LFI vulnerability we request the path of the *wp-config.php* file and examine its content: (See Figure 11)

We can see that there's a database user named *mohammad* and his password *"Mohammad_threats2022!"*, we were able to read the credentials of the database user *mohammad* and will proceed logging in with those credentials to establish a foothold on the server.

Fig. 7: WP with spritz exploit

Fig. 8: website Path

Fig. 9: Curl command

3.3 Establishing Foothold

In this step, the attacker makes sure he still has full control of the system once he's breached it. This happens right after the first concession is made. An attacker can gain a foothold by putting a backdoor on the victim's computer or by downloading additional utilities or malicious software.(Center, n.d.). In our case we will log in as the user *mohammad* and backdoor the server with a malicious custom plugin that we created to let us have a way to access the server.

Logging in as the user mohammad

We will first log in to the WordPress website as the user *mohammad* with the credentials that we previously found in the *wp-config.php* file. Logging in has been successfully done.

Getting a reverse shell

This *php* code, when executed, will connect from the victim server to the IP address *192.168.1.104*, the attacker IP address, on the port *9001* with an interactive bash shell, hence the name reverse shell. Note that we are not constrained by the port *9001* only, and the attacker IP address might be different from the one in our crafted backdoor. We must save this code and *zip* it to be able to upload it as a plugin to WordPress: (See Figure 12)

Now on the attacker machine a listener must be running to accept connections from the victim server, or else we can't communicate with the victim server. We will use Ncat to listen for a connection from the victim server and interact with it, Ncat is a powerful networking tool that can be used from the command line to read and write data across networks.(7, n.d.)

Fig. 10: The results of reading /etc/passwd file using the browser

Fig. 11: wp-config.php contents

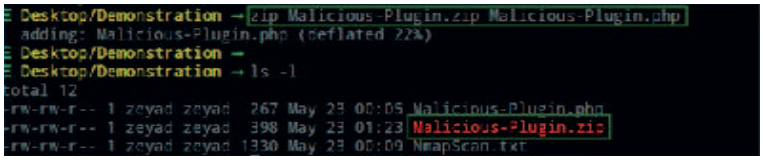

Fig. 12: saving and Zipping

–-n: (”Do not resolve hostnames”), Turning off hostname resolution in Ncat means it won’t work regardless of where the connection is coming from or where the source address goes in the routing process. The use of numerical formats for addressing is mandatory.(7, n.d.).
–-l: To listen for connections on TCP. (7, n.d.).
–-v: (”Be verbose”) With the -v option passed to Ncat, it will show detailed information about the current connection.(7, n.d.).
–-p: (”Specify source port”) Adjust the Ncat binding port. (7, n.d.).

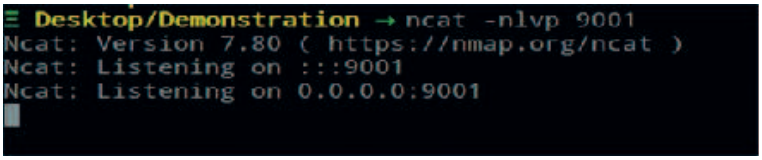

Fig. 13: Netcat listening on port 9001

In Figure 13 We can see that Ncat is now listening for connections from port 9001 from any IP address. Now the attacker machine is ready to receive connections, we want to make a connection happen from the victim server to the attacker machine using our backdoor, we will first upload it to WordPress by navigating to *Plugins ¿ Add New*.

Then by clicking the *Add New* button, and *Upload Plugin*, we can browse for our crafted backdoor and upload and install it to WordPress. Now right after activating the plugin the attacker will receive a reverse shell, you can notice in Figure 14 the IP address *192.168.1.120* of the victim server and the output of the *whoami* command (7, n.d.). The result is the www-data user, which is the user that Ubuntu web servers (Apache, nginx, and so on) use by default. Any file that www-data has access to is accessible to the web server process. The user www-data is used by Ubuntu web servers (nginx, apache, and so on) during normal operation. Each and every file that www-data may read is also accessible to the web server process.., and since it has limited permissions, the attacker needs to escalate their privileges from the user *www-data* to another user with higher privileges. (See figure 14)

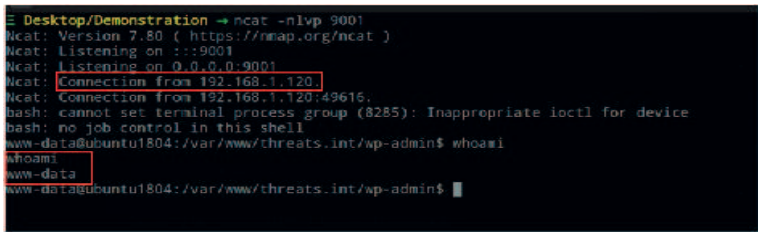

Fig. 14: receive reverse shell

3.4 Escalating Privileges

Now after we've gained a foothold on the victim server, we will proceed to the next phase which is escalating our privileges, and what we mean by that is having an access or being able to read, write and execute in the operating system as another user with higher privileges. This phase requires a lot of internal enumeration inside the server to find potential vulnerabilities to exploit them in order to gain root privileges.

From www-data to baker

After internal enumeration we find an interesting cron job running every minute in the crontab file in the path */etc/crontab* the cron job is being executed by the user baker and it's an execution of a file named *backup* located in the path */usr/bin/backup*. By examining the contents of */usr/bin/backup*, we find that it's a bash script that backs-up the entire content of the */var/www/threats*. *int* directory and compress it to a file named *webapp.tgz* under the directory */home/baker/backups/* it seems like a normal behavior but the interesting part is the asterisk in the tar command acting as a wildcard to include all files in the */var/www/threats*.int directory.

And it's not a good practice to use and asterisk with *tar* as we can exploit this to make *tar* execute the intended command but with extra options due to the asterisk including all files in the directory and if there was a file named like a *tar* option, *tar* will treat it as an option not as a file, and that's called a wildcard injection, what the attacker can do is using some interesting *tar* options:

—checkpoint: Display progress messages every Nth record (7, n.d.).
—checkpoint-action: Run ACTION on each checkpoint (7, n.d.).

The attacker could abuse those two options to make *tar* run a command after the checkpoint is reached, therefore we can execute a command as the user *baker* if we made two files that hold the name of the above options and the command will be running a script that will authorize us to login as the user *baker*, but first we must craft the malicious script in order for *tar* to run it.

The goal of the script is to add our SSH public key to the *authorized_keys* file for the user *baker*, the file is located at */home/baker/.ssh/authorized_keys*, this will allow the attacker to login as the user *baker* without even knowing his password because we would be authorized to login since our SSH public key exist in the *authorized_key* shere's the public key of the attacker: (See figure 15)

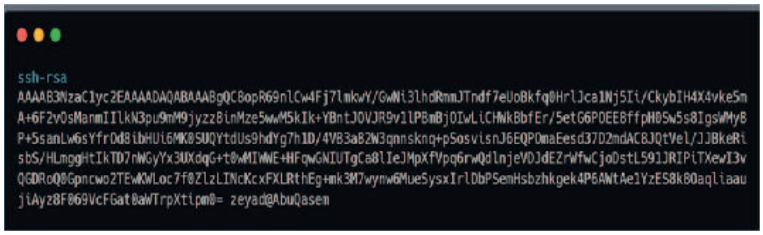

Fig. 15: The RSA key for the attacker machine

As we see in figure (16), we will name the script as *malicious-script.sh*, and the content of the malicious script would be as follows: (See Figure 16)

Fig. 16: malicious-script.sh file contents

Now we will create the files in the directory where *tar* is using the asterisk so we could exploit the wildcard injection in *tar*; inside the source directory */var/www/threats.int* we will issue those two commands to create the files, it doesn't matter what's inside the files, in fact we are creating empty files here, what's important is their names to look exactly like valid *tar* options. (See Figure 17)

Fig. 17: Directory file creations

This will allow the attacker to execute the command *sh malicious-script.sh* which by its turn will add our SSH public key to the *authorized_keys* file. After creating those two files, we will wait for one minute for the cron job start, since it's running every minute, and then we can simply SSH login to the user *baker* without being asked to provide a password.

From baker to nabil

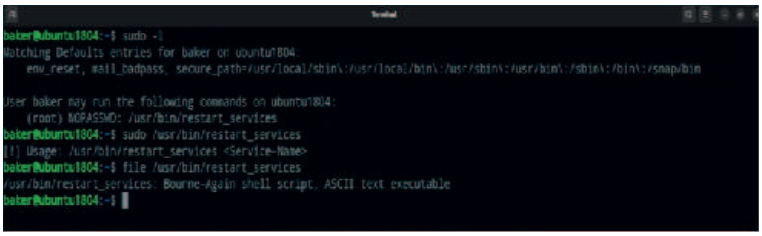

Fig. 18: sudo-l command results

Now since we are logged in as *baker*, we will try to escalate our privileges to another user with higher privileges, that means re-enumerate again for what interesting things *baker* can do. By issuing the command *sudo -l* we can see that *baker* can execute the file */usr/bin/restart_services* with root privileges without providing a password, the file is a *bash* script and it's used by executing the script with *sudo* and providing a service name as an argument to restart that service. (See Figure 18)

It contains a function called *PreventRoot* and it checks if the service file contains the string *"User=root"* and if this condition was true, it will print the message *"[!] Running service as root isn't allowed, this will be reported!"* and exit with the status of 1, but if the condition was false, meaning that the service file did not contain the string *"User=root"*, it will reload the daemon and restart the service then print the message *"[+] Done"*. That means if there's a service that isn't owned by *root* then *baker* can restart it, and by viewing the content of the */etc/systemd/system/* directory we can see that there's a service named *debug.service* that is owned by the *developers* group with read and write permissions, meaning anyone in that group can read the file and write to it. (See Figure 19)

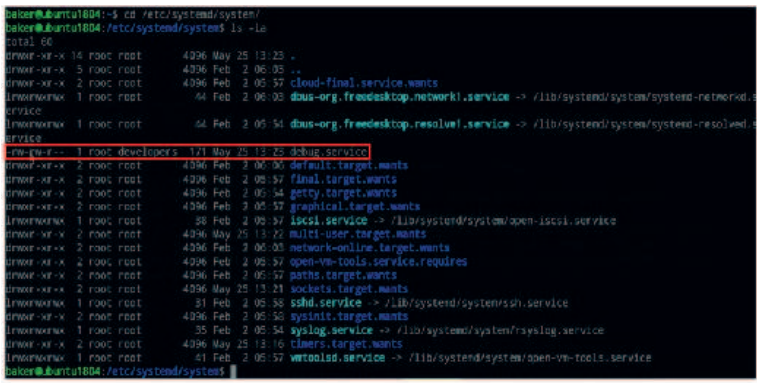

Fig. 19: root developers screen

User *baker* is in the *developers* group, this means that he can read and write to the *debug.service* file. (See Figure 20)

The *debug.service* is for debugging purposes for developers when they create services, by examining the content of the *debug.service* file we can see that we can specify a user to run the service in their permissions and what to execute when the service is started. (See Figure 21)

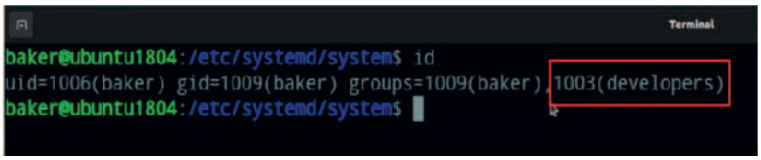

Fig. 20: baker user authentakation

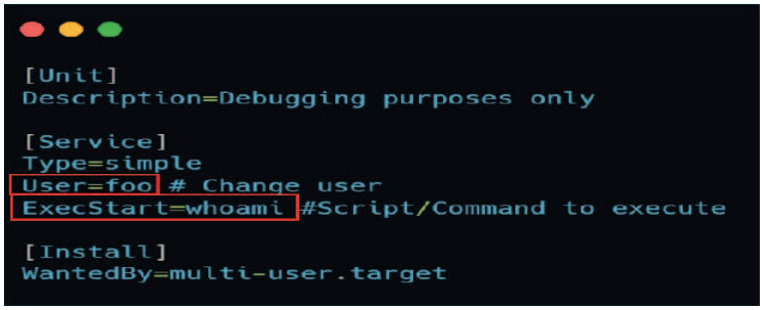

Fig. 21: debug.service file content

We can run the service and specify any user except for *root* because of the *PreventRoot* function, so we will specify another user and get another reverse shell to the port *9999* with their permissions, and we'll run it as the user *nabil*. (See Figure 22)

Fig. 22: user nabeel running operation

Starting another *Ncat* listener on port *9999* the attacker machine to accept connections from the victim server. (See Figure 23)

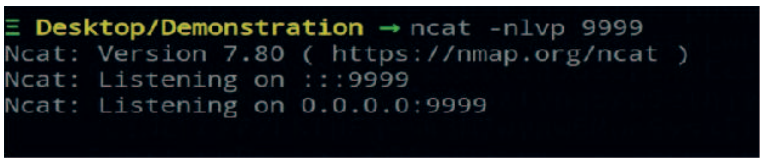

Fig. 23: Ncat listening

And then restarting the *debug.service* by the user *baker* will get us a reverse shell to the user *nabil*. (See Figure 24)

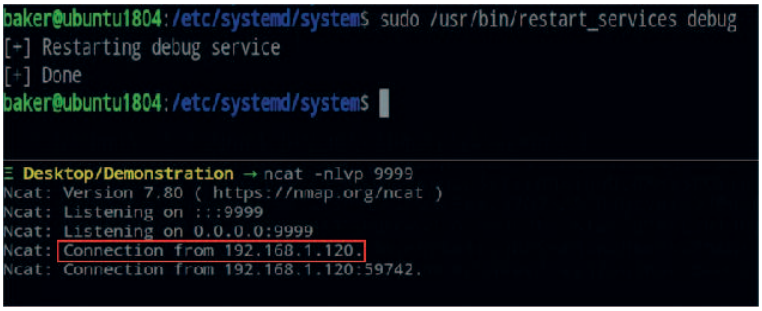

Fig. 24: connection from ip 192.168.1.120

From nabil to root

User *nabil* got higher privileges than *baker* but we still need to escalate our privileges to the highest privileged user, which is *root*, we now need to enumerate again for interesting privileges that *nabil* has. By examining the output of *sudo -l* we can notice that *nabil* can also execute the *restart_service* script with *root* permissions but the most interesting privilege is that he can execute *docker* commands with *root* privileges.

Having the rights to run *docker* with *sudo* is dangerous as we can run a container and mount the whole OS file system to the container as a volume, meaning the whole server files and directories can be read from inside the container, thus *nabil* can read any file he's not allowed to read on the main host OS, such as the files in the */root* directory, the home directory of the user *root*, therefore *nabil* having the same permissions as the *root* user on the host server. To do that we can run the container as the following command:

–-v: Bind mount a volume. (Docker, n.d.)
—rm: Clean up, automatically clean up the container and remove the file system when the container exits. (Docker, n.d.)
–-it: For interactive processes (like a shell). (Docker, n.d.)
–alpine: Alpine Linux-based Docker image that is only 5 MB in size and has a full package index. (Docker, n.d.).
–chroot /MountedFSsh: Changes the root directory to the directory that we named "MountedFS" and run a child shell that runs as a separate process from your original shell. (IBM, n.d.)

We can see that we are the *root* user inside the container and since the whole server file system is mounted to the container we then have full access to the server. Changing our directory to the */root* we find the *proof.txt* file, a file we added in the home directory of the *root* user that can only be read by root, being able to read its content *"[+] Congratulations on Successfully Hacking this Server!"* means that we have the highest privilege on the server.

4 Conclusions

Although it will guarantee the availability, integrity, and confidentiality of the Personally Identifiable Information (PII), conducting regular penetration tests, vulnerability assessments on network components and infrastructure, providing cyber awareness training, and ensuring that the development process pipeline is monitored and subject to security checks are crucial for enterprises to try to avoid threats as much as possible.

References

[1] A. Ibor and J. Obidinnu. System Hardening Architecture for Saver Access to Critical Business. *Nigerian Journal of Technology*,**34**, 788–792 (2015).
[2] P. Gallus and P.Frantis. *Security analysis of the Raspbian Linux operating system and its settings to increase resilience against attacks via network interface*. In 2021 International Conference on Military Technologies (ICMT), IEEE (2021).
[3] R. Dimov, L. Nikolov and D. Dimov. Vulnerability Analysis in Server Systems. *security & futur*,**5**, 141-146 (2021).
[4] R. Hertzog and J. O'Gorman. *Kali Linux Revealed: Mastering the Penetration Testing*. Offsec Press, New York (2017).
[5] L. Yuchong and L. Qinghui. A comprehensive review study of cyber-attacks and cyber security; Emerging trends and recent developments. *Energy Reports*,**7**, 8176-8186 (2021).
[6] F. Salahdine and N. Kaabouch. Social Engineering Attacks: A Survey. *Future Internet*,**11**, 89 (2019).

[7] W.R. Flores and M. Eksted. *Countermeasures for Social Engineering-based Malware Installation Attacks*. In The International Conference on Information Resources Management (Conf-IRM), Natal, Brazil (2013).
[8] J. Shahid, M.K. Hameed, I.T. Javed, K.N. Qureshi, M. Ali and N. Crespi. A Comparative Study of Web Application Security Parameters: Current Trends and Future Directions. *Applied Sciences*,**12**, 4077 (2022).
[9] M. Kourtesis. *Creating a Secure Server Architecture and Policy for Linux-based Systems*. Thesis for Bachelor's Degree, Linnaeus University (2015).
[10] O. M. A. Al-hazaimeh. *New cryptographic algorithms for enhancing security of voice data*. Doctoral dissertation, Universiti Utara Malaysia (2010).
[11] A. Nepal. *Linux server & hardening security*. Doctoral dissertation (2014).
[12] M. Mattetti, A. Shulman-Peleg, Y. Allouche, A. Corradi, S. Dolev and L. Foschini. *Securing the infrastructure and the workloads of linux containers*. in 2015 IEEE Conference on Communications and Network Security (pp. 559-567), IEEE (2015).
[13] S. Niu, J. Mo, Z. Zhang and Z. Lv. *Overview of linux vulnerabilities*. in 2nd International Conference on Soft Computing in Information Communication Technology (pp. 225,228), Atlantis Press, France (2014).
[14] P. Ranaweera, A.D. Jurcut and M. Liyanage. Survey on multi-access edge computing security and privacy. *IEEE Communications Surveys & Tutorials*,**23**, 1078-1124 (2021).
[15] J. Stanger and P.T. Lane. *Hack proofing Linux: a guide to open source security*. Syngress Publishing Inc., United States (2001).
[16] I.S. Al-Qasrawi and O.M. Al-Hazaimeh. A Pair-Wise Key Establishment Scheme for Ad Hoc Networks. *International Journal of Computer Networks & Communications*,**5**, 125 (2013).
[17] N. Tahat, A. Alomari, O.M. Al-Hazaimeh and M.F. Al-Jamal. An efficient self-certified multi-proxy signature scheme based on elliptic curve discrete logarithm problem. *Journal of Discrete Mathematical Sciences and Cryptography*,**23**, 935-948 (2020).
[18] N. Tahat, A. Alomari, A. Al–Freedi, O.M. Al-Hazaimeh and M.F. Al–Jamal. An efficient identity-based cryptographic model for Cheby hev chaotic map and integer factoring based cryptosystem. *Journal of Applied Security Research*,**14**, 257-269 (2019).
[19] O.M.A. Al-Hazaimeh. A new approach for complex encrypting and decrypting data. *International Journal of Computer Networks & Communications*,**5**, 95 (2013)