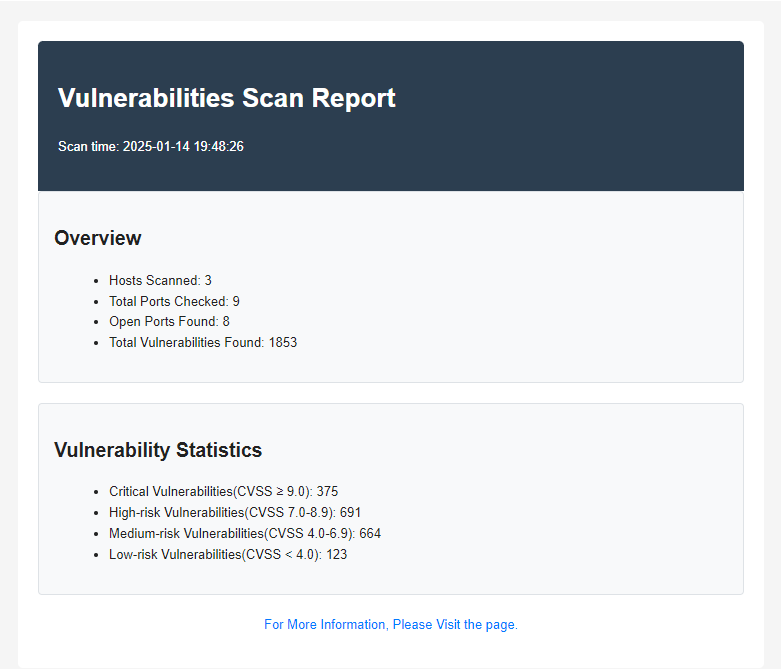
**Github repository:**

[**lc4545333/WQE7002\_Alternative-Assessment**](https://github.com/lc4545333/WQE7002_Alternative-Assessment/tree/main)

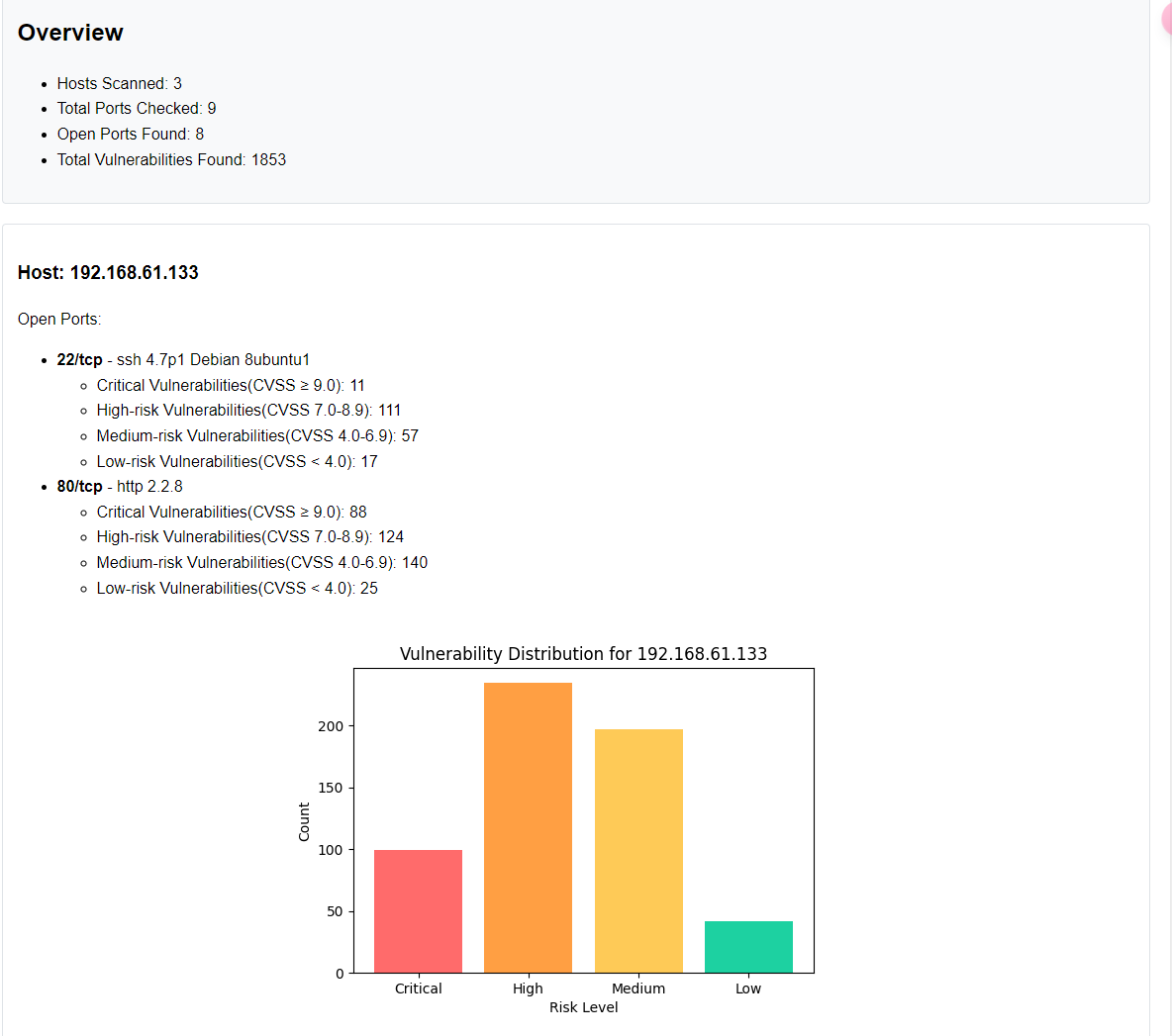
**Introduction:**

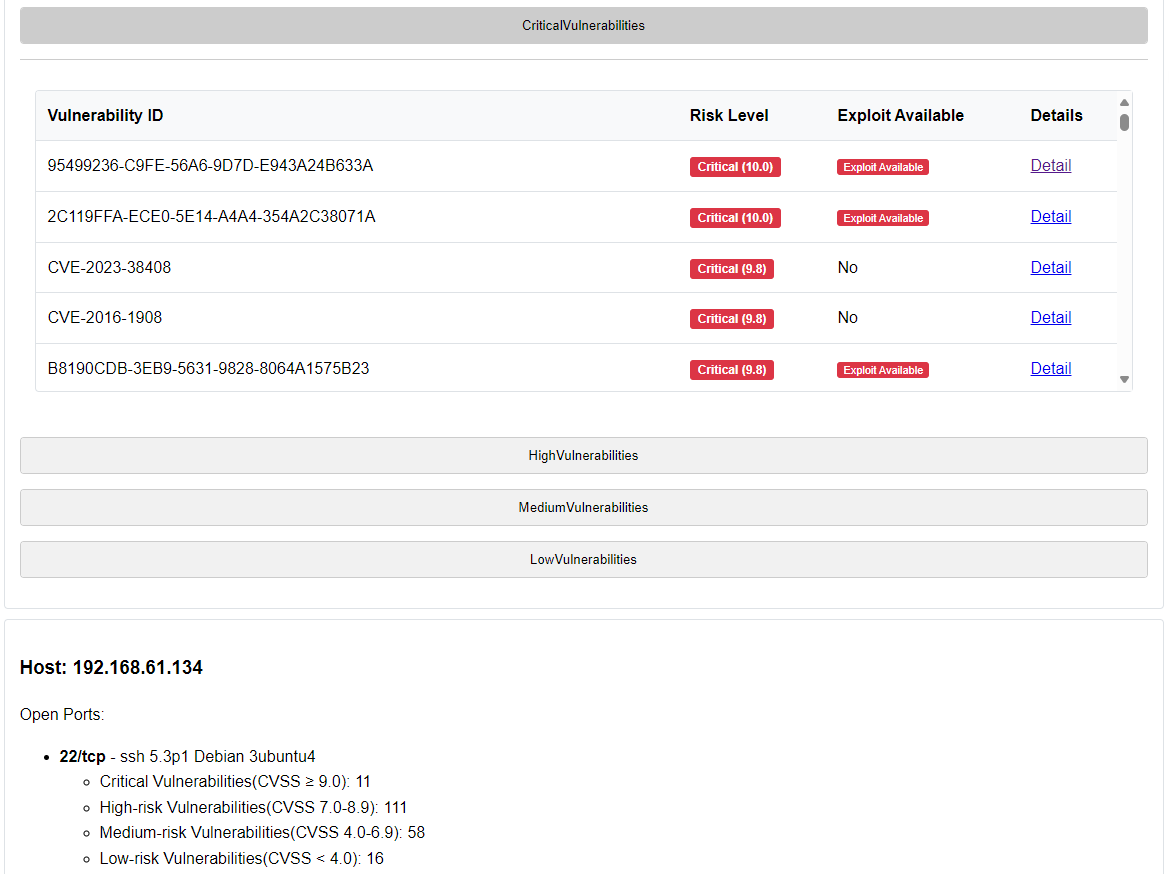
The first picture is the scan results sent to my email.



There are some simple statistics like number of hosts, number of vulnerabilities. Since the email doesn't support javascript in HTML file, I choose to only show the brief information in the email. Instead, the details of each vulnerability are displayed in another web page

**By clicking “For more infor…”, Detailed information will be shown**. We can see here is a bar chart which is used to represent the number of vulnerabilities of different severities. I have categorized the vulnerabilities according to different severities. We can click DETAIL to get the vulnerability details.





My experimental environment contains three virtual target virtual machines, each of which contains many vulnerabilities.

My program includes four steps. The First step is host discovery. I achieved a host scanner using scapy. Of course，it's far less powerful than scanning tools like Nmap. Please note that I have achieved ARP scan, ICMP scan, TCP scan and UDP scan, and I can choose any of these methods to scan the network. Also, when designing the scanner, factory pattern is used.

The first step returns a list showing the live hosts in the network. The second step is vulnerability scan, I use the python nmap library to implement this function. The results are saved in a json file.

In step3, I will generate two html files. One of them is used to show the details and will be deployed to the website, another file does not contain JavaScript code, so it can be sent by email to alert the user. The code to send the email is in step4. In step4, the sender and recipient address formats are checked before sending the email. The email sender can also send the email to multiple recipients at the same time.

**The advantage of the program is that it can run automatically. This can be done in a very simple way by using a while loop and a sleep() function.**

**Program Details:**

**Network structure**:

My Computer 192.168.61.1

Vmware-DVWA 192.168.61.134

Vmware-Metasploitable2-linux 192.168.61.133

Vmware-OWASP Broken Wen Apps VM 192.168.61.132

**Step1: Host discovery**:

input: a network address (e.g. 192.168.61.0/24)

output: a list [IP addresses of living hosts]

The scapy library is used to bulild scanning packages. I hav achieved ARP scan, ICMP scan, TCP scan and UDP scan, and I can choose any of these methods to scan the network. Also, when designing the scanner, I used the factory pattern, which achieves good scalability of the program.

Regarding the factory approach: Since scapy's packet building process is hierarchical, e.g. a TCP scan packet is built as: IP/TCP, two factories are constructed: the ScanPacketPart factory and the ScanPacket factory.

class ScanPacketPart(metaclass=ABCMeta):  
 @abstractmethod  
 def create\_part(self):  
 pass

class ScanPacket(metaclass=ABCMeta):  
 @abstractmethod  
 def create\_packet(self):  
 pass

The factory pattern brings good extensibility to programs. For example, I can add TCP Xmas scan, just add a new class and set urg, psh and fin flag.

**Key Code:**

class TCP\_packet(ScanPacket):  
 def \_\_init\_\_(self,ip\_addr\_and\_mask,dport=443,flags='S'):  
 self.ip\_addr\_and\_mask = ip\_addr\_and\_mask  
 self.dport = dport  
 self.flags = flags  
 def create\_packet(self):  
 tcp\_part = TCP\_Part(self.dport,self.flags).create\_part()  
 ip\_part = IpList(self.ip\_addr\_and\_mask).create\_part()  
 TCP\_packet\_list = ip\_part/tcp\_part  
 return TCP\_packet\_list

class TCP\_packet(ScanPacket): **#add XMAS scan**  
def \_\_init\_\_(self,ip\_addr\_and\_mask,dport=443,flags='FPU'):

pass

def create\_packet(self):

pass

**Step2: vulnerability scan:**

input: a list [IP addresses of living hosts]

output: scan\_results.json

The library used to achieve the function is python-nmap. It is a call to an existing tool nmap, so the implementation is very simple.

**Key Code：**

scanner.scan(ip, ports=ports, arguments='-sV --script vulners')

#Use the -sV --script vulners command to scan the hosts found in step one

results = {  
 'host': ip,  
 'timestamp': datetime.datetime.now().isoformat(),  
 'ports': {}  
 }

The scan results are eventually saved in scan\_result.json, which is structured roughly like this:

{  
 "192.168.61.133": {  
 "host": "192.168.61.133",  
 "timestamp": "2025-01-14T19:48:02.206627",  
 "ports": {  
 "22": {  
 "state": "open",  
 "service": "ssh",  
 "version": "4.7p1 Debian 8ubuntu1",  
 "vulnerabilities": [  
 {  
 "cpe": "cpe:/a:openbsd:openssh:4.7p1",  
 "id": "95499236-C9FE-56A6-9D7D-E943A24B633A",  
 "severity": 10.0,  
 "url": "https://vulners.com/githubexploit/95499236-C9FE-56A6-9D7D-E943A24B633A",  
 "is\_exploit": true  
 },

Please note that after completing step2, I have all the data I need to build the web page.

**Step3: Generate two html files**

input: scan\_results.json

output: vulnerability\_report\_email.html and vulnerability\_report.html

First I count the vulnerabilities discovered in STEP2 and categorize them according to their severity.

def get\_severity\_level(score):

pass

Then the vulnerabilities are analyzed and the number of vulnerabilities of different severity is counted, and the number of vulnerabilities of different severity is shown in the form of a bar chart. Based on the analysis results, dynamic html files are generated. Please note that a new html file is generated for each scan, and the content of the web page changes according to the scanning result.

def generate\_chart\_image(severity\_counts: Dict[str, int], host: str):

pass

*def process\_scan\_results(json\_file\_path):*

pass

def generate\_html\_report(scan\_results)：

pass

the different between these two html files is that one of them is used to show the details and will be deployed to a website, another file does not contain JavaScript code, so it can be sent by email to alert the user.

**Step4: Send email**

input: vulnerability\_report\_email.html and vulnerability\_report.html

output: A webpage and an email.

The final step of the program involves sending the vulnerability scan report via email. The report is generated as an HTML file (vulnerability\_report\_email.html) in Step 3, and the step ensures that the report is delivered to the intended recipients. The email functionality includes:

Validating sender and recipient email addresses.

EMAIL\_REGEX = re.compile(r'^[a-zA-Z0-9.\_%+-]+@[a-zA-Z0-9.-]+\.[a-zA-Z]{2,}$')

def validate\_email(email):

pass

Sending the HTML report as the email body and saving a copy of the email content locally for record-keeping.

def send\_email(sender, recipients, subject, content, smtp\_server, smtp\_port, username, password, is\_html=True):

pass

def save\_email\_to\_file(email\_data):

pass

There are also features such as Handling errors during the email-sending process.