Finding Exploitable Services on a Network

Roger Tang, Rosary Wang, Johahn Wu, Andrew Zoellner
University of Texas
Department of Computer Science
rogertang, rosarywang, johahnwu, andrewzoellner@{cs.utexas.edu}

***Abstract*—*NMAP* is a security scanner that has an extremely large number of features that may be confusing for first time users who are not familiar with the program. The use of our tool is to target a specific application (in our case to find exploitable services on a network) and then automate the process. This was accomplished through the use of various scripts to run commands and parse output as well as checking results with an open source vulnerabilities database.**

***Index Terms*—NMAP, FTP, SSH, SMTP, MSSQL, Vulnerabilities**

I. Introduction

The purpose of our tool is to find exploitable versions of FTP, SSH, SMTP and MSSQL Server running on machines on a network. To accomplish this, three different parts of the assessment methodology were utilized to first ping the desired network range for live hosts, then scan the commonly associated ports for each service to resolve if they are open and finally perform banner grabbing to determine the versions of the service being run. As a final step, the version numbers are cross referenced with a database of known vulnerable versions to see if the host is running a vulnerable service. Fig. 1 depicts the flow of data through our tool. The entire process is automated using a BASH script and various python programs to parse the output of the NMAP commands[1].

II. Pinging Network Range for Live Hosts

The first step scans a range of IP addresses and retrieves all the addresses that have a live or active host. A linux shell script was used to automate this step by first running an NMAP search over the given range of IP addresses. This command can be seen in Fig. 2. The shell script takes in an address range in the same format as NMAP. The output of the scan is piped into a text document which is then read by a python program to grab the IP addresses out of the NMAP output and create a text file to be used in step 2. This step eliminates a large number of addresses that are inactive thus reducing the search time of step 2. It also breaks the range of addresses into single addresses which allows for further NMAP scans to be done individually yielding incremental vulnerability information.

```
>nmap -sn <ip range> > liveHosts
```

Fig. 2. NMAP command for first step

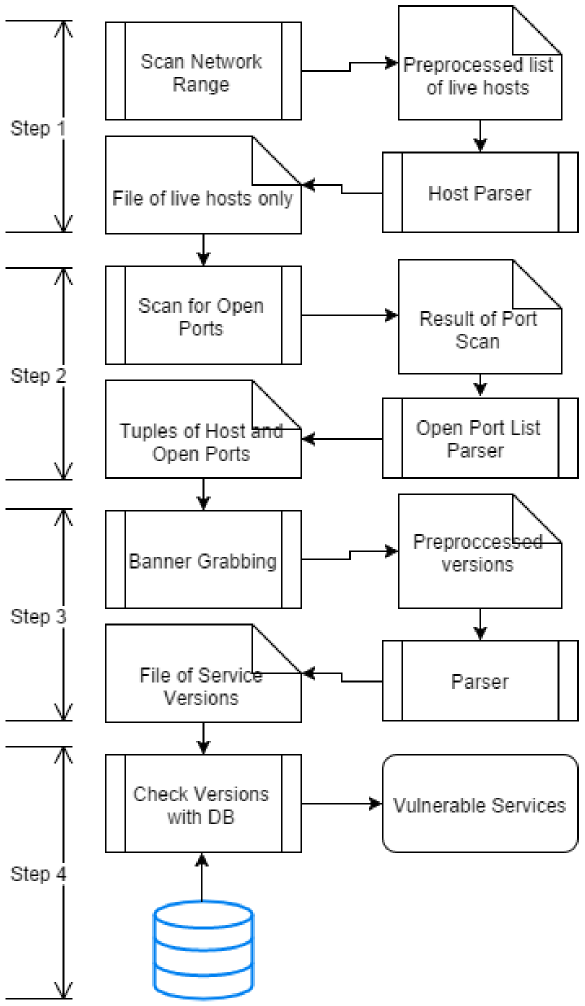

Fig. 1. Flow Chart depicting the tool

III. Port Scanning for Targeted Services

The second step of the automation tool is to perform an NMAP scan of the live hosts obtained from the first step on the following ports to see if they are open: FTP(21), SSH(22), SMTP(25) and MS SQL Server(1433)[2]. Much like step 1, the purpose of this step is to narrow down the targets we have to scan in order to minimize run time. The reason these ports were picked were because they are commonly found

```
>nmap -n --open -p21,25,1433 -iL liveHosts > portList
```

Fig. 3. NMAP command for second step

services on a network and at least for some are very prone to vulnerabilities (e.g., MSSQL Server)[3]. This step consisted of two different steps. First, an NMAP command is called using the following flags: -n for no DNS resolution, --open to look only for open ports, -p21,22,25,1433 to only scan ports associated with our selected services and -iL to pass in target hosts from a file created by the previous step. The entire command can be seen in Fig. 3. Next, the output is parsed with a python script to return a file with a host and port tuple of all the open ports on the network corresponding to the services we are interested in. This file is then passed onto the 3rd step for banner grabbing.

IV. Banner Grabbing

Our third and final NMAP step investigates each IP address from the second step's result to find out the version of software installed on the targeted port. The automation process utilizes the knowledge of IP address and its ports status, then uses NMAP to only search for the software version of known open ports. The result of this step is then parsed to become reasonable search phrases that can be used for the final step. The general logic behind this step can be seen in Fig. 4.

```
for all IP and Port Tuples in portList do
    >nmap -sV <ip> -p<port>
    Parse Output
    Append to output file
end for
```

Fig. 4. Pseudo-code for third step

V. Cross-referencing with Vulnerability Database

The fourth step cross-references the versions of the services running over the scanned port with the vulnerability databases through the vFeed Framework[5]. vFeed Framework was used due to it being built on open source technologies and relies on the main Open Standard CPE. The framework also supports correlation with IAVA, OSVDB, OVAL, Nessus, Exploit-DB, Redhat, Microsoft, and more. Using vFeed's search by CPE function, each service version enumerated in step 3 is queried against the vFeed database. Any matches, which indicates a possible vulnerable version, are added to our final list of vulnerable services on the network. The pseudo-code for this step can be found in Fig. 5.

```
for all Banner/Service versions in versionList do
    Query vFeed Database for service version
    if version found in DB then
        Append to vulnerable list
    else
        continue to next banner
    end if
end for
```

Fig. 5. Psuedo-code for fourth step

VI. Complications

While creating the tool and testing it on various networks we ran across a few problems. When checking against a database of known vulnerabilities it was difficult to match the version of the service ran on a certain port with the same service and version listed in a given database due to the lack of convention used to name these services. Similarly, some services that were listed as vulnerable included a range of versions (e.g. Version 5.0 and older) which contained the vulnerability. This in particular made it difficult for our tool to work correctly because there was no way to tell from the database which other versions also contained the vulnerability. Fortunately, Common Platform Enumeration (CPE) provides a standard machine-readable format for encoding names of IT products and platforms. This allowed us to narrow down search results.

One unfortunate flaw with our tool is that it isn't entirely conclusive. Banner grabbing does not account for patches that may have been applied to fix security vulnerabilities thus any service determined to be vulnerable by our tool doesn't necessarily mean it's exploitable. However, given how often vulnerable services are left unpatched[4], it may still be prudent to treat these services as if they are still vulnerable.

VII. Future Work

One obvious way the tool can be improved is simply increasing the number of services that we scan for. Currently, we are only looking for vulnerabilities for FTP, SSH, SMTP and MSSQL Server. However adding additional services is trivial and can be done by additionally scanning the default port the service is run on. With the obvious improvements stated, we can now move on to less trivial future enhancements.

Another way the tool could be improved would be to give it the option of scanning the network on a trusted level. This may yield more accurate vulnerabilities as a scan done at an administrator level would get more results. In the experimentation of this tool there weren't a large number of vulnerabilities found, but in the cases where large networks and ranges are being scanned there could be thousands of vulnerabilities listed. Sorting the results based on the potential threat for each vulnerability would give the user a better way to sift through the data for real threats.

For some services, we have run into the problem where the banner grabbing step doesn't provide a conclusive version (e.g., postfix for SMTP). Thus queries for the service in the vFeed database results in all known vulnerabilities for the service. A definite improvement would be finding a more concrete way to find version numbers so the resulting vulnerability list is more accurate.

To tackle our tools inability to determine if a security patch has been applied, we can integrate Nessus' Patch Management feature [6] between the third and fourth step of our tool. This would of course require us to revise our cross-referencing step and either find a more comprehensive database that accounts for patches or adapt our current implementation so patches take precedence over version numbers.

Finally, while our tool is functional, the final step could be further optimized. We are currently relying on vFeed's search script which takes a very long time to search the entire database and searching one service at a time. This can be simply fixed by multithreading the search portion of our tool since the database changes relatively infrequently. If time is of an issue, we can pre-process the databse using various information retrieval concepts to further optimize the search function.

References

[1] G. Lyon. (2015). *Nmap: the Network Mapper - Free Security Scanner*. [Online] Available: https://nmap.org/

[2] J. Reynolds. (1994). *Assigned Numbers*. [Online] Available: http://www.ietf.org/rfc/rfc1700.txt?number=1700

[3] CVE Details. (2014). *Microsoft SQL Server Security Vulnerabilities*. [Online] Available: https://www.cvedetails.com/vulnerability-list/vendor_id-26/product_id-251/Microsoft-Sql-Server.html

[4] B. Schneier. (2014). *The Internet of Things Is Wildly Insecure And Often Unpatchable*. [Online] Available: http://www.wired.com/2014/01/theres-no-good-way-to-patch-the-internet-of-things-and-thats-a-huge-problem/

[5] ToolsWatch. (2015). *vFeed*. [Online] Available: http://www.toolswatch.org/2015/08/vfeed-correlated-vulnerability-database-api-major-update-0-6-released/

[6] Tenable Network Security. (2015). *Patch Management*. [Online] Available: http://www.tenable.com/solutions/patch-management