Penetration Testing Report

TESTING (MAPPING AND SCANNING): Mapping the target environment and conducting a vulnerability scan

This is one of the initial and crucial parts of penetration testing, this is part of the recon and analysis portion of penetration testing. To begin the mapping and scanning portion of the penetration test, I ran a NetDiscovery command utilizing the NetDiscovery tool in the Kali Linux environment. This helped me discover the vulnerable systems and their arp packets were captured as well. I ran a nmap for each host to confirm which one was exploitable. The host that I knew was metasploitable was not up and running at the time, so NMAP failed to scan it. Finally, I chose to do a NMAP scan of the host 203.0.113.100, in the screenshots below it shows the ports that are open and can be exploited on the system. This can easily result in privilege escalation as well. Thus, I came to the conclusion that 203.0.113.100 is the most vulnerable on the network.

A picture containing calendar

Description automatically generated

Text

Description automatically generated

Exploitation: Gaining Access through A vulnerability identified during the vuln scan

After completing the successful vulnerability scan, I noticed quite a few exploitable ports and services running. One of which was the rmiregistry, I immediately noted down the port it was running on. I then launched Metasploit and began the process of exploitation. Initially, I searched for the Java\_rmi modules, to ensure the service could be exploitable, I ran the scanner auxiliary/scanner/misc/java\_rmi\_server as shown in the screenshot below:

Text

Description automatically generated

This shows the Class Loader is enabled, I then went ahead and proceeded with running the exploit module as well, which can be seen in the screenshot above. I also went ahead and set the URIPATH, this is because the java rmi when told to load a class, it needs to retrieve it from somewhere. The machine needs to retrieve it, in this case it uses an http request.

Text

Description automatically generated

Next the SRVPORT is set, which I set to 80. When the show targets command or info command is used, it shows the java payload listed as can be seen in the above screenshot. Next, I used the Set PAYLOAD command, and since it was defaulted to 0, I simply hit tab and it showed the different payloads available for the specific target. Next, I set the specific PAYLOAD to Java/Meterpreter/reverse\_tcp, and I then set the LHOST (local host) and LPORT.

Text

Description automatically generated

Text

Description automatically generated

I then typed the exploit command, which resulted in the session being established, and I had gotten root access utilizing the Java rmi vulnerability. I typed getuid to confirm I was in as root.

Text

Description automatically generated

I then went ahead and typed the LS and PS commands, to list the different processes and directories on the machine.

Graphical user interface

Description automatically generatedText

Description automatically generated

I also ran the ifconfig command to further confirm root access to the system:

Text

Description automatically generated

From the above screenshots you can see how it only took a few steps to exploit this java rmi vulnerability. This shows the importance of penetration testing, and how it helps with finding these vulnerabilities in a system to be fixed before they are exploited by malicious actors.

Analysis and Reporting: Communicating findings and providing mitigation recommendation

Finally, after having exploited the vulnerability in my target system, I came to the realization that this vulnerability is something that very much could have been corrected. With regards to the rmiregistry or Java RMI (remote method invocation) vulnerability, it is due to the way the application is created. The whole point of the Java RMI application is that it should be able to communicate to another program remotely, which is greatly beneficial for its usability purposes. However, when it comes to security, such an application comes with vulnerabilities that can be exploited if not secured. Essentially, Java RMI allows an object that exists in one VM to access and call methods that are contained in another Java VM (DRD\_, 2018). A Major advantage of Java RMI is that it allows for remote objects to load new classes that aren’t explicitly defined, which extends functionality and behavior of an object. RMI applications are made up of a client and a server, when the server is initially setup, the methods become available to the client. The communication is handled by a stub and skeleton, the stub is client side and sends information, which includes relevant parameters. While the skeleton is on the server and passes the client request to the remote object. Now that we know the way Java RMI works, both client side and server side, we can understand where the vulnerability arises. The most immediate vulnerability which can be noted, is a misconfigured or insecurely configured server. This results in the possibility of classes to be loaded from any URL, because method calls do not require authentication, it can be exploited easily.

Although I could not find the NIST NVD specifically for this vulnerability itself, I found a couple other NIST NVD CVEs for the RMI vulnerability. In both CVEs the Java RMI vulnerability is listed as critical and has a score of 9.8 for its criticality. The first CVE being CVE-2021-37578 and the second being CVE-2019-2684. This is certainly a vulnerability which would need immediate attention given to it, considering the fact that it can easily attain root access to system. This vulnerability, although it is isolated to the Java RMI, it can be exploited using different methods. The two CVEs listed above, one of them shows it being exploited using a Oracle Java SE (NIST, NVD, 2021), a subcomponent of the RMI. The other shows it being exploited using Apache jUDDI, where malicious serialized objects can be sent to the RMI entries(NIST, NVD, 2019).

The risk that this vulnerability poses to an organization is quite vast, whether it be internal or external facing servers. If a malicious actor were to gain access to an organization using this Java RMI vulnerability, they have access to the root of the server. This leads to access to all the different processes, directories, and they can easily delete data, alter data, and setup backdoors. If a backdoor is established without the organizations knowledge, it results in further security risks. In some cases, they may even be able to jump from one host on the network to another. They can run commands to find network information, such as ifconfig to find information about the network. There is no shortage of issues that can be caused when privileges are escalated, or root access is gained.

The best way for this vulnerability to be mitigated, is to first change the default configuration of the server, to ensure it is more secure. Part of that process would also be ensuring that the Java RMI has the latest patches and updates for it installed. As at this time the vulnerability has been patched. However, in the situation that the patch wouldn’t have been deployed yet, the class loader would need to be disabled (Progress, 2018). I would also ensure that the rmiregistry is not displayed so easily when a scan of the host is done, that port should be closed when not in use as well to prevent unwanted access and execution. Just as when securing any system on the network, unwanted ports or unused ports should always be blocked and closed. Consistent patching and penetration testing of vulnerabilities within an organization can help prevent such vulnerabilities from being exploited. The importance of penetration testing and the purpose it servers cannot be emphasized enough.

References

NIST, NVD.GOV, *CVE-2019-2684 Detail*. (2019) https://nvd.nist.gov/vuln/detail/CVE-2019-2684.

NIST, NVD.GOV, *CVE-2021-37578 Detail*. (2021) <https://nvd.nist.gov/vuln/detail/CVE-2021-37578#:~:text=RMI%20uses%20the%20default%20Java%20serialization%20mechanism%20to,deserialized%20without%20any%20check%20on%20the%20incoming%20data>.

Progress Community, *How to Prevent Java RMI Class Loader Exploit on AdminServer.* (2018, January 23). <https://community.progress.com/s/article/How-to-prevent-Java-RMI-class-loader-exploit-with-AdminServer>*.*

Wonder How To Null Byte, *Exploit Java Remote Method Invocation to Get Root*. (2018, November 5) <https://null-byte.wonderhowto.com/how-to/exploit-java-remote-method-invocation-get-root-0187685/#:~:text=Scanning%20for%20Java%20RMI%20Start%20Metasploit%20by%20typing,type%20search%20rmi%20and%20locate%20the%20%22auxiliary%2Fscanner%2Fmisc%2Fjava_rmi_server%22%20module>.