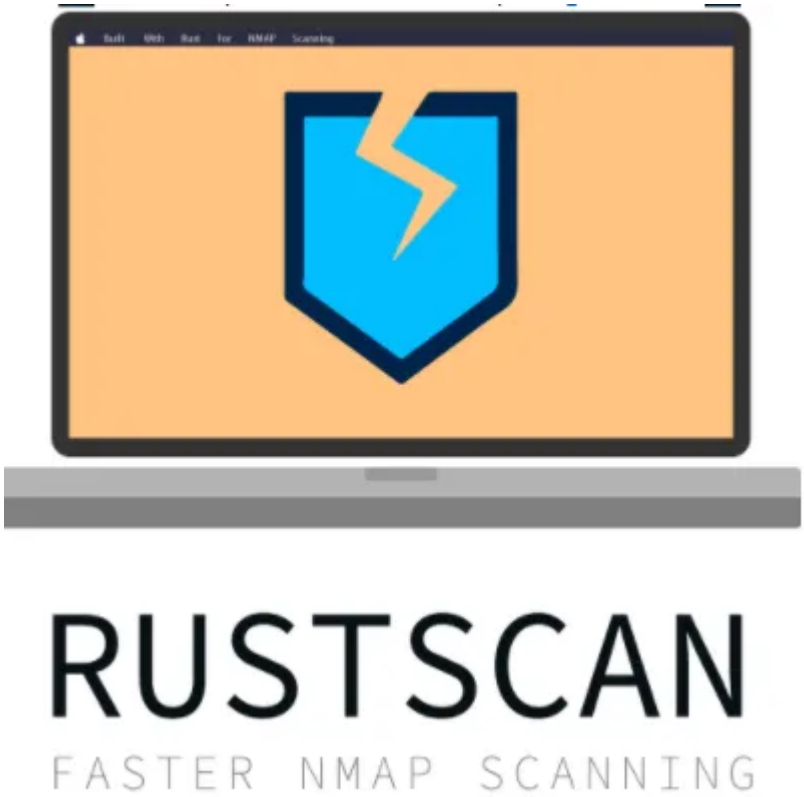
RustScan

Ayush Bagde · Follow
8 min read · Jan 11, 2021



Click **here** to get the room access.

Hey Guys, I'm Ayush Bagde aka Overide and in this we're gonna learn how to solve RustScan. Its a very easy machine to solve. So without further wasting any time let's jump into it.

RustScan is the modern day port scanner. Capable of scanning targets in less than a second, extensible scripting language allowing you to write scripts in Python, and more. This room will teach you all there is need to know about

This room will teach you all there is need to know about RustScan.

You can find RustScan's GitHub repository here.

Task 2 Installing RustScan

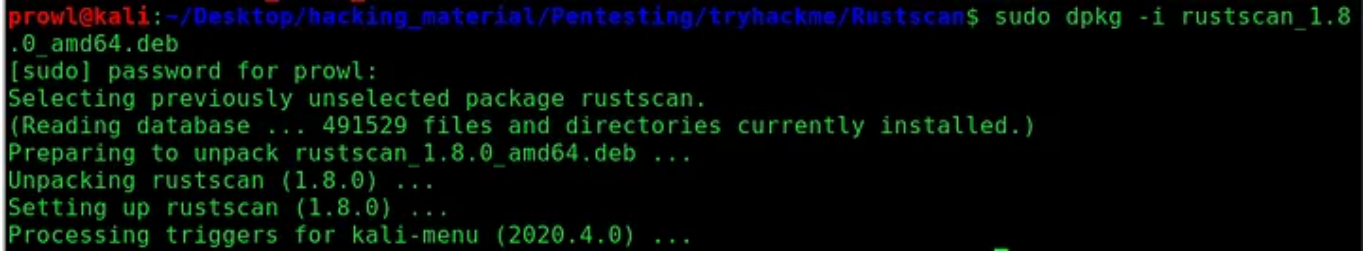
The installation procedure of RustScan is very easy.

Note: If you use Mac OS, Arch, Docker, Nix OS or any other operating system than Debian more install instructions can be found on the repository.

Download the .deb file for Kali Linux. (rustscan_1.8.0_amd64.deb)

Go to the folder where the package is then type the following command

```
sudo dpkg -i rustscan_1.8.0_amd64.deb
```



Task 3: Accessible

RustScan is actively accessible. That means we:

- Use continuous integration testing to ensure accessibility needs are met
- Manually test accessibility

This is important because accessibility is important in hacking.

Hacking isn't just inaccessible, it is the opposite — it is actively discluding members from the community because of some issues they were born or developed that they cannot help.

It is a basic human right to extend everything we do to be accessible to everyone. In the same way, it is right for you to get healthcare, to get an education.

Around 15% of all people suffer from some sort of disability. See this blog post for more info on why accessibility is important or the world report on disability for statistics.

For more information about accessibility in infosec, I wrote this blog post

Task 4: Fast

RustScan is fast. But, why? In short:

- Low-level kernel networking
- Written in a fast language (Rust)

- Asynchronous scanning. Multi-threading is slow due to the context switching cost. Async is fast. See my Rust room for more information on this.

At its fastest settings, the bottleneck is not the program. The bottleneck is your computer, the network link between your computer and the target, and finally the target itself.

If your OS/hardware isn't the best, that'll be the bottleneck.

This sounds bad, but some other port scanners have their bottlenecks in the program itself. RustScan is as fast as your system, the network, and the target is.

Being so fast is bad, but RustScan can be slowed down to however slow you want. But just know, if speed is your thing — RustScan is where it's at.

How do you ensure speed?

As more features are added into RustScan, slowness tends to creep in unless noticed.

As we add more features, the program generally tends to slow down. This is a rule for all programs. More complexity will often mean it is slower (but not always).

RustScan deals with this by:

- Manual testing on targets
- Continuous integration that fails if the program is too slow
- Benchmarking the program and graphing the results
- Keeping all features outside of the scanning itself. Unless it is absolutely needed, RustScan will not run code before or during the scan — only

after.

Task 5: Extensible

Now for the largest feature of RustScan. Thanks to Bergabman for working on this one.

RustScan is extensible by the RustScan Scripting Engine. This allows you to write a script which runs after the scan has completed, taking inputs of open ports and their respective IPs.

RSE supports these languages:

- Python
- Shell
- Perl
- Any program which is a binary and in $PATH

Scripting Engine Arguments

RustScan's scripting engine can be altered using the " — scripts" argument.

There are 3 possible arguments:

- None (don't run any scripts)
- Custom (run all scripts in the scripts folder)
- Default (runs Nmap script, or whatever script is in the config file. Default does not need to be enabled, it is on by default.)

Python Custom Scripts

To execute a custom script, we need a rustscan_scripts.toml file located at $HOME/ .rustscan_scripts.toml

The script file should look like:

```
# Test/Example ScriptConfig file

# Tags to filter on scripts. Only scripts containing all these tags
will run.
tags = ["core_approved", "example"]

# If it's present then only those scripts will run which has a tag
ports = "80". Not yet implemented.
#
# ex.:
# ports = ["80"]
# ports = ["80","81","8080"]
ports = ["80"]

# Only this developer(s) scripts to run. Not yet implemented.
developer = ["example"]
```

Let's walk through this.

Firstly, for reference, this is a basic Python script.

```
#!/usr/bin/python3
#tags = ["core_approved", "example",]
#developer = [ "example", "https://example.org" ]
#trigger_port = "80"
#call_format = "python3 {{script}} {{ip}} {{port}}"

# Scriptfile parser stops at the first blank line with parsing.
# This script will run itself as an argument with the system
installed python interpreter, only scanning port 80.
# Unused filed: ports_separator = ","

import sys

print('Python script ran with arguments', str(sys.argv))
```

Note: the metadata of scripts is stored as comments. The first line is always a shebang.

Tags

Tags are categories of scripts. For example, we may have these categories:

- HTTP
- SSH
- Tomcat

And only wish to run scripts that match these categories. Our config file will only execute the scripts with matching categories.

Developer

This tag issues who the developer of the script is.

Trigger Point

This tag states at what port should the script trigger? For HTTP it would be "80". For HTTP and HTTPS it would be "80, 443"

Call Format

RustScan uses a templating library called text_placeholder.

This allows us to enclose variables in `{{variable}}` doubly curly braces. RustScan supports 3 variables:

- The script name
- The IP address
- The port(s)

```
#call_format = "python3 {{script}} {{ip}} {{port}}"
```

The Code itself

Now everything after this metadata is the code itself.

The script will receive arguments via `sys.argv` in the format specified in the `call_format` variable.

Now with this data, we run the script, doing whatever we please!

Contributing / Making Scripts

We have a folder of example scripts here.

If you make a script, please consider contributing to RustScan. Right now you can submit a pull request to this folder and we'll include your script.

Running Other Tools with RustScan

Any tool installed in the system (like Nmap, GoBuster, etc) can be run with RustScan.

We do this by default with Nmap.

To execute another program, create a shell script which calls that program. So to call Nmap, create a shell script with our RustScan Scripting Engine and then for the function:

```
nmap -vvv -p {{port}} {{ip}}
```

You can replace this with GoBuster or any program at all. So long as the program is installed and reachable in the environment $PATH.

1. What is the scripting file config called?
Answer: rustscan_scripts.toml

2. Can you run other binaries with RustScan? (T)rue/ (F)alse
Answer: T

3. Does RutScan support scripts in Javascript? (T)rue / (F)alse.
Answer: F

TASK 6: Adaptive

RustScan is **adaptive**. That means it changes how it works to better suit its environment. We call this the “adaptive learning” feature set.

Some of these features included (or are being worked on) are:

- Adaptive Outbound SYN timing to optimize the speed of scanning

While RustScan scans the target, it learns how it reacts, How fast is it to scan? Does it respond quickly? How far away?

By using this data RustScan moulds itself so it is the fastest scanner for any target.

- Custom Top Ports

You may be familiar with the top ports feature of other scanners. It’ll let you scan the top 1000 ports on the internet.

But, the top 1000 ports on the internet are not often the top 1000 ports you might come across. Corporate networks may have unusual ports open, capture the flag events may have unusual ports. As an example, port 31137 is used a lot in CTFs because “l33t”.

This port is not in any top 1000 ports list.

RustScan learns what the most commonly open ports are **for you** and adapts itself.

- Operating System Adaption

Your computer is not the same as my computer. So why run the same scan settings? Mac OS devices have an open file limit of around 250. That means they can only make 250 connections at any given time.

Kali Linux has around 90,000 open files.

If we built this for Kali Linux, it would break on a Mac which does not support that many open files.

RustScan learns about your operating system and adapts itself to better suit you and your computer, as well as the networks it is scanning.

- Configuration File

All of this information is stored in a configuration file. Onboarding a new pentesting intern? Send them your config file and their RustScan will be optimal from day 1.

1. **I understand this.**
   No Answers needed

TASK 7: Scanning Time!

The tool is really amazing in terms of scanning. It can scan all the ports really fast and then pipe the output to the Nmap. Now in this room, we'll scan our vulnerable machine.
Basic format for RustScan is **rustscan -r ports -a <Target-ip> — <nmap cmds>**

Here's a full list of things you can do.

Multiple IP Scanning

You can scan multiple IPs using a comma-separated list like so:

```
rustscan -a 127.0.0.1,0.0.0.0
```

Host Scanning

RustScan can also scan hosts, like so:

```
➜ rustscan -a www.google.com, 127.0.0.1
Open 216.58.210.36:1
Open 216.58.210.36:80
Open 216.58.210.36:443
Open 127.0.0.1:53
Open 127.0.0.1:631
```

CIDR support

RustScan supports CIDR:

```
➜ rustscan -a 192.168.0.0/30
```

Hosts file as input

The file is a new line separated list of IPs / Hosts to scan:

hosts.txt

```
192.168.0.1
192.168.0.2
google.com
192.168.0.0/30
127.0.0.1
```

The argument is:

```
rustscan -a 'hosts.txt'
```

Individual Port Scanning

RustScan can scan individual ports, like so:

```
➜ rustscan -a 127.0.0.1 -p 53
53
```

Multiple selected port scanning

You can input a comma-separated list of ports to scan:

```
➜ rustscan -a 127.0.0.1 -p 53,80,121,65535
53
```

Ranges of ports

To scan a range of ports:

To run:

```
➜ rustscan -a 127.0.0.1 --range 1-1000
53,631
```

Adjusting the Nmap arguments

RustScan, at the moment, runs Nmap by default.

You can adjust the arguments like so:

```
rustscan -a 127.0.0.1 -- -A -sC
```

To run:

```
nmap -Pn -vvv -p $PORTS -A -sC 127.0.0.1
```

Random Port Ordering

If you want to scan ports in a random order (which will help with not setting off firewalls) run RustScan like this:

```
➜ rustscan -a 127.0.0.1 --range 1-1000 --scan-order "Random"
53,631
```

1. **Try running the scan for all ports.**
   No Answers needed
2. **After scanning this, how many ports do we find open under 1000?**
   2
3. **Perform a service version detection scan, what is the version of the software running on port 22?**
   6.6.1p1
4. **Perform an aggressive scan, what flag isn't set under the results for port 80?**
   httponly
5. **Using this tool in scanning can save a lot of time! Make sure to use it in your pentest.**
   No Answers needed

TASK 8: RustScan Quiz

1. **First, how do you access the help menu?**

   -h

2. **Often referred to as “quiet” mode, What switch can do this?**

   -q

3. **Which switch can help us to scan for a particular Range?**

   -r

4. **What switch would you use to find out RustScan’s version?**

   -v

5. **Which switch will help us to select batch size?**

   -b

6. **Which switch can set timeout?**

   -t

This completes our room and that was it from me. If you enjoyed reading this, do give it a clap and follow me on medium. If you face any problem regarding any solution, feel free to reach me out. Hope you enjoyed reading my work. If you really liked this article, then follow me on medium and connect with me on **LinkedIn**. Till then goodbye from my side and Happy Hacking.

Tryhackme Writeup

Written by Ayush Bagde

Follow

76 Followers

Cybersecurity Associate at ACPL Systems | MTA Security Fundamentals | Junior Pentester | DLP | Brand Monitoring | Android Pentest | Seclore | DRM

More from Ayush Bagde

Ayush Bagde

TShark TryHackMe Writeup

Learn how to use TShark to accelerate your pcap analysis!

May 5, 2021

Ayush Bagde

The Ultimate Ethical Hacking Roadmap: A Comprehensive Guide

Ethical hacking, also known as penetration testing or white-hat hacking, plays a crucial...

Dec 3, 2023

Ayush Bagde

History of Malware TryHackMe Writeup

Ayush Bagde

MAL: REMnux-The Redux TryHackMe Writeup

Join this room to learn about the first forms of malware and how they turned into the...

Mar 9, 2021

A revitalised, hands-on showcase involving analysing malicious macro's, PDF's and...

Mar 11, 2021

See all from Ayush Bagde

Recommended from Medium

Md.Maruf Ahmed

REmux The Tmux | TryHackMe Walkthrough

Updated, how to use tmux guide. Defaults and customize your workflow.

Apr 25

xocybersec in Infosec WatchTower

TryHackMe—ColddBox: Easy Walkthrough

A walkthrough with my tactics, techniques, and procedures.

Jan 23

Lists

Staff Picks

678 stories · 1100 saves

Stories to Help You Level-Up at Work

19 stories · 675 saves

Self-Improvement 101
20 stories · 2215 saves

Productivity 101
20 stories · 1968 saves

Rahul Kumar in System Weakness

Metasploit: Exploitation | Tryhackme Walkthrough

Using Metasploit for scanning, vulnerability assessment and exploitation.

Feb 2

Royall Researchers in InfoSec Write-ups

Windows Fundamentals 1 | TryHackMe Walk-Through

Royall Researchers

Mar 22

Yusif Yagubzadeh

TryHackMe: Linux Privilege Escalation

Today we will take look at TryHackMe: Linux Privilege Escalation. This is a one of the...

Feb 13

Cyber Sierra

TryHackMe—Room # 2—Intro to Defensive Security

Dear audience, this is Cyber Sierra speaking. Today we will be working over a room from...

Jan 1