Network research Project

Name:Yaniv Juravliov

Project code:NX201

Student code:s26

Table of Contents

Overview of the Script

This Bash script serves as a comprehensive, automated framework designed to

facilitate secure, anonymous, and fully documented remote network scanning operations, integrating multiple subsystems into a single streamlined workflow. The script begins by establishing a controlled environment through the automatic installation and verification of essential reconnaissance and connectivity tools, including sshpass for password-based SSH automation, Tor for anonymity routing, Nmap for host and service enumeration, Whois for domain information retrieval, and GeoIP utilities for external IP geolocation. By silently checking each dependency and installing missing components with color-coordinated output. Following this preparation stage, the script proceeds to deploy **Nipe**, a specialized tool that forces all outbound traffic through the Tor network, thereby converting the machine into a fully anonymized scanning host. This installation includes fetching the Nipe repository from GitHub, installing Perl dependencies using cpanminus, initiating the Tor service, and enabling Nipe to route all system traffic through a Tor exit node. Critically, the script incorporates an operational-security safeguard by performing an anonymity verification step: it retrieves the current external IP address, determines its country of origin via GeoIP lookup, and immediately terminates execution if the system still exposes a real, non-anonymous (e.g., Israeli) IP address. This ensures that no scans are conducted unless the environment is fully anonymized. Once anonymity is confirmed, the script transitions into its core operational phase, where it orchestrates remote scanning on a VPS via automated SSH execution. It prompts the user for remote server credentials and the target to be analyzed, then securely invokes a remote scanning script (agent_tasks.sh) located on the VPS. All output from the remote server is captured, stored locally, and then saved into a structured reporting directory with timestamped filenames for long-term reference. Alongside generating detailed result files, the script maintains a persistent audit log that records every scan event, including timestamps, the remote server used, the scanning target, the precise location of the generated report, and the Tor-obfuscated IP address active at the time of execution. This log ensures accountability, repeatability, and traceability—qualities essential in penetration testing, SOC investigations, and digital forensics workflows. Ultimately, the script integrates environment preparation, anonymity enforcement, remote automation, result capturing, and audit logging into a unified system. It significantly reduces operator workload, eliminates repetitive manual tasks, strengthens operational security, and produces standardized, well-organized documentation suitable for both technical analysis and professional reporting.

How the Project contributed me

From this project I have learned a lot and also encountered few challenges on the way, first and foremost it contributed me a lot to the understanding of anonymity and awareness of OPSEC, understanding how the tools such as nipe, tor, nmap work through the network and also alternative tools with each one has its own unique capabilities. The importance of automation process and how it contributes both to reliability and speed to the purpose of the specific task. I really think I've done my "network research" and understood there's a lot more to investigate and gain knowledge of as it is a world of it's own .On the way making this script I encountered some challenges such as making it automated as possible in which I had to write a code and then test it and constantly creating new snapshots on the VM and make corrections to adapt a new code and do the same process over again , create more prototypes , using different techniques .

further more writing the script increased my bash script a lot as I had to dive in deeper into "advanced bash scripting content" in order to understand better and write better which allowed me to use less assistance of AI tools and even work better with AI .

To sum up I feel more skilled both in bash scripting and I feel I understand network better in terms how it operates when anonymity functions applied

Breaking Down The Script

I will break down the script into 6 sections

1.Dependency Installation & Environment Setup

(tools,service)
Installs required tools (sshpass, nmap, whois, geoip, tor/nipe)
the verification of the installation (just for the doublecheck)
these will be the foundation of the script

2. Tor/Nipe Anonymity Setup
Why anonymity is checked , how nipe works and how the script validates our spoofed IP + country *this is important part of OPSec awareness

3. User Input & Target Selection
which is asking for the remote serv credentials and information (ip address , user , password)

4.SSH Automation & Remote Command Execution
how sshpass and ssh work and how to utilize it for automation, what are the commands executed remotely and what information they bring to the table

5. Data Collection

6. final output & results
What the user sees at the end , Where results are stored and how the script finishes execution

1.Dependency Installation & Environment Setup (tools,service)

```
function APP_INSTALL()
{
    echo -e "${CYAN}🔍 Checking and installing required applications...${RESET}"
    echo -e "$LINE"

    APPS=( "sshpass" "tor" "nmap" "whois" )
    for A in "${APPS[@]}" ; do
        if ! command -v "$A" >/dev/null 2>&1 ; then
            echo -e "${YELLOW}Installing $A...${RESET}"
            sudo apt-get install -y "$A" >/dev/null 2>&1
        else
            echo -e "${GREEN}✅ $A is already installed.${RESET}"
        fi
    done

    if ! command -v geoiplookup >/dev/null 2>&1; then
        echo -e "${YELLOW}Installing Geoiplookup...${RESET}"
        sudo apt-get install geoip-bin -y >/dev/null 2>&1
    else
        echo -e "${GREEN}✅ Geoiplookup is already installed.${RESET}"
    fi
}
APP_INSTALL
```

Begins with defining an Array APPS=(the list of the tools) and proceeds with a for loop that iterates through each item in the APPS array.
This allows the script to check and install each tool one by one.
command -v returns the path to the executable if it exists.
If the command is **not (!)** found, then it proceeds to install it with the command **sudo apt-get install -y "$A"** Output is hidden **(>/dev/null 2>&1)** to keep the script clean when it is executed . If the program **is** already installed, the script prints a green checkmark confirming it . now geoiplookup is a little bit more tricky,because we need to install the package which is geoip-bin if we add it to the array command -v geoip-bin, it will fail because there is no such command . so while its still possible to create a script that will check the command and install the correct package, requires a deeper understanding of bash which im not there but i'll definitely acquire it latter . so same concept
if ! command -v geoiplookup if the path to the command does NOT exist
then **sudo apt-get install geoip-bin -y** . and we are done.

1.1 Verification of the installation

```
function IF_INSTALLED() {
    echo -e "${CYAN}🔍 Verifying installed tools...${RESET}"
    echo -e "$LINE"

    TOOLS=( "sshpass" "tor" "nmap" "whois" "geoiplookup" )
    for T in "${TOOLS[@]}"; do
        if ! command -v "$T" >/dev/null 2>&1; then
            echo -e "${RED}❌ $T is NOT installed! Install manually.${RESET}"
        else
            echo -e "${GREEN}✔️  $T is installed successfully.${RESET}"
        fi
    done
}
IF_INSTALLED
```

Honestly it wasn't that necessary but i did it for the sports and the double check . it's basically the same concept as the first function with the array and for loop following the if ! command -v , but here i actually added the geoiplookup as there is no sudo apt install -y command just and echo msg that says if the tool is installed or not
but few notes :
*most of the tools are already pre-installed on kali
*different packaging of linux distros this will work on Debian based distros
such as kali,parrot,ubuntu,etc..
*Other distros use **different package managers**, for example:
RedHat / CentOS uses yum
Fedora uses dnf
arch linux uses pacman
servies on some distros:
*ssh is called sshd
File Paths Differ Across Distributions
Root Permission Requirements Differ
So to summarize , double check with your self in order for the script to run successfully

2. Tor/Nipe Anonymity Setup

```
function NIPE_INSTALL() {
    echo -e "${CYAN}📥 Installing and setting up Nipe...${RESET}"

    git clone https://github.com/htrgouvea/nipe >/dev/null 2>&1 && cd nipe
    sudo apt install cpanminus -y >/dev/null 2>&1
    sudo cpanm --notest --quiet --sudo --installdeps . >/dev/null 2>&1
    sudo perl nipe.pl install >/dev/null 2>&1
    sudo systemctl start tor
    sudo systemctl start ssh
    echo -e "${YELLOW}Starting nipe service...${RESET}"
    sudo perl nipe.pl start
}
NIPE_INSTALL
echo -e "$LINE"
```

The NIPE installation section begins by cloning the official Nipe repository from GitHub using git clone, which downloads the project into the current directory. After navigating into the nipe folder, the script installs cpanminus
which is a Perl module installer required to handle Nipe's dependencies. Next, the script uses cpanm to quietly install all necessary Perl modules for Nipe to function. This ensures that the underlying Perl environment has everything it needs before Nipe can run. Once all dependencies are set up, the script executes **sudo perl nipe.pl install** which initiates the internal installation process After that, the script starts two essential services: Tor and SSH **(sudo systemctl start tor , sudo systemctl start ssh)**. Tor is required because Nipe routes all system traffic through the Tor network, and SSH is needed for secure remote server communication later in the script. Finally, the script starts Nipe itself using **perl nipe.pl start**, which activates the anonymous routing engine. When this command completes successfully, the system's outbound traffic should now be routed through a Tor exit node, enabling the anonymity checks that come later in the script.

2.1 Check Anonymity function

```
function CHECK_ANONYMITY() {
    echo -e "$LINE"
    echo -e "${CYAN}🔎 Checking network anonymity...${RESET}"

    IP=$(curl -s ifconfig.co)
    COUNTRY=$(geoiplookup "$IP" | awk -F ': ' '{print $2}')

    if echo "$COUNTRY" | grep -iqE "Israel|IL"; then
        echo -e "${RED}❌ You're NOT anonymous. Exiting now.${RESET}"
        exit 1
    else
        echo -e "$LINE"
        echo -e "${GREEN}✅ You're anonymous!${RESET}"
        echo -e "${YELLOW}🌍 IP: $IP${RESET}"
        echo -e "${YELLOW}📌 Spoofed Country: $COUNTRY${RESET}"
    fi
}
CHECK_ANONYMITY
```

IP=$(curl -s ifconfig.co) curl -s ensures the request is silent, and ifconfig.co returns only the public-facing IP of the machine. When Nipe and Tor are active, this IP should be different from the user's real one. ifconfig.co is a web service that displays information about the internet connection **COUNTRY=$(geoiplookup "$IP" | awk -F ': ' '{print $2}')**
GeoIP lookup is a process that determines the geographic location of an IP address by mapping it to real-world coordinates and associated data such as country, region, city, latitude, longitude, ISP, domain, and connection type. With some text manipulation we are able to carve out the 2 letter code of the country and the full name

```
❯ geoiplookup 149.202.79.101 | awk -F ': ' '{print $2}'
FR, France
```

-F stands for field separator we are separating the colon and the space to print the FR and FRANCE with $2 , followed then we set a condition **if echo "$COUNTRY" | grep -iqE "Israel|IL"; then**
if it displays the country of our origin it will print "You're not anonymous" and the script will exit that can indicate that nipe isn't working properly or the anonymity we seek for , couldn't be achieved .
but if it doesn't match then it will display a green confirmation message indicating that anonymity is successfully established.
finally displaying the spoofed country location and **public** ip address

```
echo -e "${YELLOW}🌍 IP: $IP${RESET}"
echo -e "${YELLOW}📌 Spoofed Country: $COUNTRY${RESET}"
```

2.2 About Nipe and Tor & Alternatives

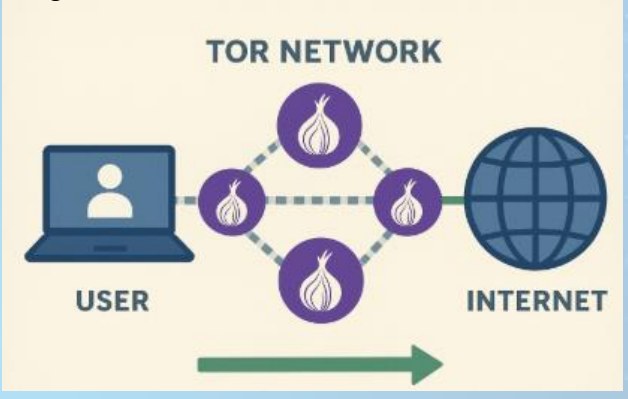
Nipe is an engine, developed in Perl, that aims on making the Tor network your default network gateway. Nipe can route the traffic from your machine to the Internet through Tor network, so you can surf the Internet having a more formidable stance on privacy and anonymity in cyberspace.
Despite its utility, Nipe has several **important technical limitations**:

- **IPv6 is not supported** — Only IPv4 traffic is routed through Tor. Systems relying on IPv6 may leak data outside the encrypted tunnel.
- **DNS leak protection is partial** — While most DNS queries are protected, some local/loopback requests are excluded from Tor routing.
- **All non-local UDP/ICMP traffic is blocked** — This can disrupt services requiring real-time communication or network diagnostics.
- **No built-in obfuscation** — Unlike Tor Browser with pluggable transports, Nipe does not support **snowflake, obfs4**, or other censorship-circumvention methods (though this is a requested feature).
- **Iptables conflicts** — If custom firewall rules exist, they may interfere with Nipe's operation during startup or shutdown.

What is TOR and how does it work?
Tor (The Onion Router) is a free, open-source network designed to enable anonymous communication over the internet. It protects users' identities and online activity from surveillance and traffic analysis by routing internet traffic through a global, decentralized network of volunteer-operated servers known as *relays* or *nodes*.
Tor works by **encrypting and routing your internet traffic through at least three randomly selected relays(a.k.a nodes)** before it reaches its final destination. This process prevents any single point on the network from knowing both the origin (your device) and the destination (the website you're visiting)

Important notes:

- **Entry node**: Knows your real IP address but not your destination.
- **Middle node**: Knows neither your IP nor the final destination.
- **Exit node**: Knows the destination but not your original IP.



2.3 About Anonsurf

Anonsurf is a privacy protection module developed by the ParrotSec team, designed to provide system-wide anonymization by routing all network traffic through the Tor network. It is integrated into Parrot OS and functions as a script that configures IP packet filter rules using Tor, enabling users to achieve near-untraceable online activity simply by starting the service. Unlike Tor's browser-only solution, Anonsurf anonymizes all traffic from the entire system, not just web browsing, ensuring that applications, services, and even background processes are routed through Tor.

Anonsurf's functionality lies in its **comprehensive and automated approach to privacy**. It not only routes traffic through Tor but also supports the Invisible Internet Project (I2P) for additional anonymity, clears user disk traces, and employs a "Pandora bomb" feature to terminate potentially dangerous applications that could compromise anonymity. This all-in-one functionality simplifies the process of maintaining privacy, eliminating the need for users to manually configure proxies, browsers, or other tools. The tool is designed for ease of use, with simple commands like start, stop, and restart to manage anonymization sessions, and even offers a graphical interface in Parrot OS, allowing users to activate anonymity with a single click.

Furthermore, Anonsurf enhances security by disabling IPv6, shutting down services that could leak information, and ensuring that all traffic is encrypted and routed through Tor's layered network. This system-wide approach provides a more robust privacy solution compared to tools that only anonymize specific applications or require complex manual setup. While it does not guarantee absolute anonymity—especially if users log into personal accounts while connected—it significantly reduces the digital footprint and makes tracking much more difficult.

```
┌─[yanivslab@parrot]─[~]
└──╼ $curl -s ifconfig.co
77.137.77.6
┌─[yanivslab@parrot]─[~]
└──╼ $sudo anonsurf start
[sudo] password for yanivslab:
Do you want to kill dangerous apps? [Y/n] n
[*] AnonSurf is activated
  AnonSurf started. Your traffic goes through Tor
```

```
┌─[yanivslab@parrot]─[~]
└──╼ $geoiplookup 45.84.107.74
GeoIP Country Edition: DE, Germany
```

2.4 tcpdump before and after

After running anonsurf tcpdump shows no internet-bound packets because they are encrypted and routed through TOR internally.”

```
┌[✗]-[yanivslab@parrot]-[~]
└─► $sudo tcpdump -i ens33
tcpdump: verbose output suppressed, use -v[v]... for full protocol decode
listening on ens33, link-type EN10MB (Ethernet), snapshot length 262144 bytes
03:37:25.594743 IP 100.14.71.202.nl7.servers.guru.9000 > 192.168.188.138.37164: Flags [P.], seq 284668093:284668629, ack 2144088424, win 64240, length 536
03:37:25.594823 IP 192.168.188.138.37164 > 100.14.71.202.nl7.servers.guru.9000: Flags [.], ack 536, win 65535, length 0
03:37:25.687248 IP 192.168.188.138.37164 > 100.14.71.202.nl7.servers.guru.9000: Flags [P.], seq 1:537, ack 536, win 65535, length 536
03:37:25.687442 IP 100.14.71.202.nl7.servers.guru.9000 > 192.168.188.138.37164: Flags [.], ack 537, win 64240, length 0
03:37:27.239234 IP 100.14.71.202.nl7.servers.guru.9000 > 192.168.188.138.37164: Flags [P.], seq 536:1072, ack 537, win 64240, length 536
03:37:27.239329 IP 192.168.188.138.37164 > 100.14.71.202.nl7.servers.guru.9000: Flags [.], ack 1072, win 65535, length 0
03:37:27.637283 IP 192.168.188.138.47044 > 188.165.0.43.tproxy: Flags [P.], seq 2140453091:2140453627, ack 1300711005, win 65535, length 536
03:37:27.637477 IP 188.165.0.43.tproxy > 192.168.188.138.47044: Flags [.], ack 536, win 64240, length 0
03:37:27.654087 IP 192.168.188.138.37164 > 100.14.71.202.nl7.servers.guru.9000: Flags [P.], seq 537:1073, ack 1072, win 65535, length 536
03:37:27.654357 IP 100.14.71.202.nl7.servers.guru.9000 > 192.168.188.138.37164: Flags [.], ack 1073, win 64240, length 0
03:37:27.793686 IP 192.168.188.138.47044 > 188.165.0.43.tproxy: Flags [P.], seq 536:1072, ack 1, win 65535, length 536
03:37:27.793867 IP 188.165.0.43.tproxy > 192.168.188.138.47044: Flags [.], ack 1072, win 64240, length 0
03:37:27.909470 IP 188.165.0.43.tproxy > 192.168.188.138.47044: Flags [P.], seq 1:537, ack 1072, win 64240, length 536
03:37:27.909723 IP 192.168.188.138.47044 > 188.165.0.43.tproxy: Flags [.], ack 537, win 65535, length 0
03:37:28.045422 IP 188.165.0.43.tproxy > 192.168.188.138.47044: Flags [P.], seq 537:1073, ack 1072, win 64240, length 536
03:37:28.045592 IP 192.168.188.138.47044 > 188.165.0.43.tproxy: Flags [.], ack 1073, win 65535, length 0
03:37:28.596836 IP 100.14.71.202.nl7.servers.guru.9000 > 192.168.188.138.37164: Flags [P.], seq 1072:1608, ack 1073, win 64240, length 536
03:37:28.596965 IP 192.168.188.138.37164 > 100.14.71.202.nl7.servers.guru.9000: Flags [.], ack 1608, win 65535, length 0
03:37:28.883428 IP 192.168.188.128.43644 > vpn.tdr.li.9001: Flags [P.], seq 3129011043:3129011579, ack 323345535, win 65535, length 536
03:37:28.883436 IP vpn.tdr.li.9001 > 192.168.188.128.43644: Flags [.], ack 536, win 64240, length 0
03:37:28.901191 IP 192.168.188.138.37164 > 100.14.71.202.nl7.servers.guru.9000: Flags [P.], seq 1073:1609, ack 1608, win 65535, length 536
03:37:28.901456 IP 100.14.71.202.nl7.servers.guru.9000 > 192.168.188.138.37164: Flags [.], ack 1609, win 64240, length 0
03:37:29.115605 IP 192.168.188.128.58572 > 96.9.98.41.https: Flags [P.], seq 964662218:964662754, ack 292332159, win 65535, length 536
03:37:29.115613 IP 96.9.98.41.https > 192.168.188.128.58572: Flags [.], ack 536, win 64240, length 0
```

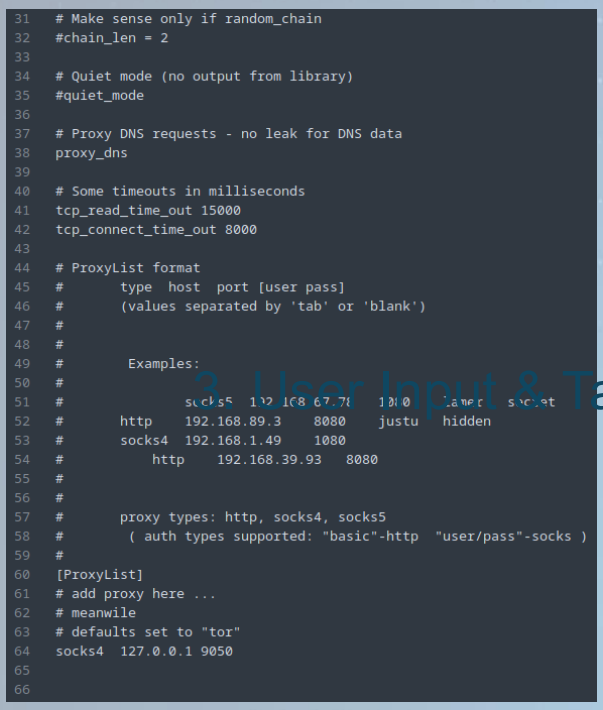
Before running anonsurf tcpdump exposes all outbound traffic, including DNS.

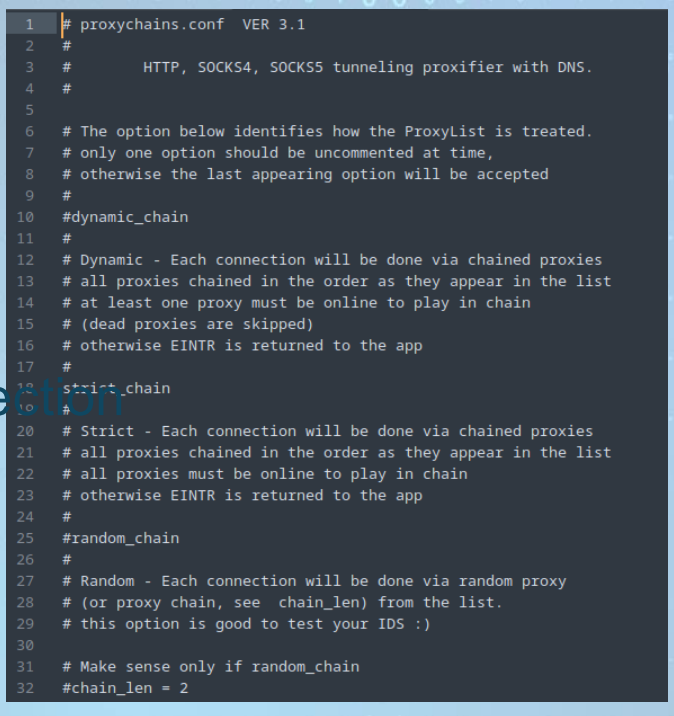
```
┌[yanivslab@parrot]-[~/Desktop]
└─► $sudo tcpdump -i ens33
tcpdump: verbose output suppressed, use -v[v]... for full protocol decode
listening on ens33, link-type EN10MB (Ethernet), snapshot length 262144 bytes
03:45:11.885324 IP 192.168.188.1.mdns > mdns.mcast.net.mdns: 0 PTR (QM)? _spotify-connect._tcp.local. (45)
03:45:11.885900 IP6 fe80::b239:7bc6:af53:cd93.mdns > ff02::fb.mdns: 0 PTR (QM)? _spotify-connect._tcp.local. (45)
03:45:11.901301 IP 192.168.188.138.57986 > 192.168.188.2.domain: 36615+ PTR? 251.0.0.224.in-addr.arpa. (42)
03:45:11.913880 IP 192.168.188.2.domain > 192.168.188.138.57986: 36615 1/0/0 PTR mdns.mcast.net. (70)
03:45:11.914371 IP 192.168.188.138.45701 > 192.168.188.2.domain: 21604+ PTR? 1.188.168.192.in-addr.arpa. (44)
03:45:11.937390 IP 192.168.188.2.domain > 192.168.188.138.45701: 21604 NXDomain 0/1/0 (99)
03:45:11.938092 IP 192.168.188.138.53353 > 192.168.188.2.domain: 38465+ PTR? b.f.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.2.0.f.f.ip6.arpa. (90)
03:45:11.950410 IP 192.168.188.2.domain > 192.168.188.138.53353: 38465 NXDomain 0/1/0 (166)
03:45:11.950913 IP 192.168.188.138.34799 > 192.168.188.2.domain: 11038+ PTR? 3.9.d.c.3.5.f.a.6.c.b.7.9.3.2.b.0.0.0.0.0.0.0.0.0.0.0.0.0.0.8.e.f.ip6.arpa. (90)
03:45:11.972489 IP 192.168.188.2.domain > 192.168.188.138.34799: 11038 NXDomain 0/1/0 (139)
03:45:12.003943 IP 192.168.188.138.60156 > 192.168.188.2.domain: 44059+ PTR? 2.188.168.192.in-addr.arpa. (44)
03:45:12.026101 IP 192.168.188.2.domain > 192.168.188.138.60156: 44059 NXDomain 0/1/0 (99)
03:45:12.026536 IP 192.168.188.138.36993 > 192.168.188.2.domain: 39270+ PTR? 138.188.168.192.in-addr.arpa. (46)
03:45:12.048642 IP 192.168.188.2.domain > 192.168.188.138.36993: 39270 NXDomain 0/1/0 (101)
03:45:13.195432 IP 192.168.188.138.ntp > 107.150.179.160.ntp: NTPv4, Client, length 48
03:45:13.209339 IP 107.150.179.160.ntp > 192.168.188.138.ntp: NTPv4, Server, length 48
03:45:13.246942 IP 192.168.188.138.54473 > 192.168.188.2.domain: 46400+ PTR? 160.179.150.107.in-addr.arpa. (46)
03:45:13.259408 IP 192.168.188.2.domain > 192.168.188.138.54473: 46400 NXDomain 0/1/0 (116)
03:45:13.528429 IP 192.168.188.1.63607 > 239.255.255.250.1900: UDP, length 125
03:45:13.556974 IP 192.168.188.138.52789 > 192.168.188.2.domain: 59779+ PTR? 250.255.255.239.in-addr.arpa. (46)
03:45:13.568977 IP 192.168.188.2.domain > 192.168.188.138.52789: 59779 NXDomain 0/1/0 (119)
03:45:15.195447 IP 192.168.188.138.ntp > 107.150.179.160.ntp: NTPv4, Client, length 48
03:45:15.208774 IP 107.150.179.160.ntp > 192.168.188.138.ntp: NTPv4, Server, length 48
03:45:17.195457 IP 192.168.188.138.ntp > 107.150.179.160.ntp: NTPv4, Client, length 48
03:45:17.208098 IP 107.150.179.160.ntp > 192.168.188.138.ntp: NTPv4, Server, length 48
```

Proxychains is **a UNIX-based tool designed to force any TCP connection made by an application to pass through a series of proxy servers**, enhancing anonymity and privacy by masking the user's original IP address. It works by intercepting network-related library calls in dynamically linked programs using a preloaded shared library (LD_PRELOAD), redirecting traffic through configured proxies such as SOCKS4a/5 or HTTP, and supporting only TCP traffic. The tool allows users to create a "chain" of proxies, where each connection request is routed sequentially through multiple proxy servers, making it difficult for the final destination to trace the request back to the original source.

The most common implementation is proxychains-ng, a maintained successor to the original unmaintained project, which is pre-installed on distributions like Kali Linux. Users can configure the proxy chain in the /etc/proxychains4.conf file, choosing from different chaining modes like dynamic, strict, random, or round-robin, which determine how proxies are selected and handled during connection attempts. It supports integration with various proxy types, including Tor (default on many systems), and can be used with tools like Nmap, Metasploit, SSH, and web browsers to route traffic securely. For example, running proxychains4 firefox forces the browser to use the configured proxy chain.

While proxychains offers strong privacy benefits, especially for penetration testers and red team operations, it requires careful setup and reliable proxy servers to avoid performance issues or connection failures. It is primarily designed for Linux systems, though Windows alternatives like proxychains-windows or Proxifier exist, though with limited functionality or requiring paid licenses. The tool is particularly effective for bypassing geo-restrictions, evading firewalls, and conducting secure network operations, but users must be aware of potential DNS leaks and the need for proper configuration to ensure complete anonymity.





3. User Input & Target Selection

```
function RMT_VPS ()
{
    RemoteScanResults="$HOME/Desktop/RemoteScanReport"
    mkdir -p "$RemoteScanResults"

    echo -e "$LINE"
    read -p "PW of the Remote Server: " RMSPW
    read -p "USER of the Remote Server: " RMSUS
    read -p "IP of the Remote Server: " RMSIP
    read -p "Enter target domain/IP for scanning: " TARGET

    echo -e "${CYAN}[+] Running scan on remote VPS...${RESET}"
```

Starting with defining two variables that are going to save the results into a directory **$HOME/Desktop/RemoteScanReport**

A directory that will contain the outputs of the remote Nmap scan (reports, logs, results, etc.

A single log file that records script actions, timestamps, and possibly errors.

mkdir -p "$RemoteScanResults" Creates the directory if it doesn't exist , Does **not throw errors** if the folder already exists The -p flag means “make parent directories as needed”

```
-p, --parents     no error if existing, make parent directories as needed,
                  with their file modes unaffected by any -m option
```

read -p "PW of the Remote Server: " RMSPW
reads an input for the Password of the remote server
read -p "USER of the Remote Server: " RMSUS
reads an input for the User of the remote server
read -p "IP of the Remote Server: " RMSIP
reads the input for the Remote server **private ip** address
these variables will be used latter with sshpass , ssh

***Note**: Password input is visible because there is **no -s flag** (silent).
If you want it hidden, you can use:
read -s -p "PW of the Remote Server: " RMSPW
echo
read -p “Enter target domain/IP for scanning:” Target

4. SSH Automation & command execution

```
sshpass -p "$RMSPW" ssh -o StrictHostKeyChecking=no "$RMSUS@$RMSIP" "
    mkdir -p ~/scan_reports
    REPORT=~/scan_reports/${TARGET}_report_$(date +%F_%H-%M).txt

    echo '========== SYSTEM INFO ==========' >> \"\$REPORT\"
    echo 'Hostname:' \$(hostname) >> \"\$REPORT\"
    echo 'Uptime:' >> \"\$REPORT\"
    uptime >> \"\$REPORT\"

    echo '========== PUBLIC IP ==========' >> \"\$REPORT\"
    IP=\$(curl -s ifconfig.co)
    echo \"IP: \$IP\" >> \"\$REPORT\"
    geoiplookup \"\$IP\" >> \"\$REPORT\"

    echo '========== WHOIS ($TARGET) ==========' >> \"\$REPORT\"
    whois \"$TARGET\" >> \"\$REPORT\"

    echo '========== NMAP ($TARGET) ==========' >> \"\$REPORT\"
    nmap -sV -T4 --top-ports 200 \"$TARGET\" >> \"\$REPORT\"

    echo \"===== REPORT SAVED ON VPS: \$REPORT =====\"
"
 sshpass -p "$RMSPW" scp -o StrictHostKeyChecking=no \
"$RMSUS@$RMSIP:~/scan_reports/${TARGET}_report_*" \
"$RemoteScanResults/"
```

this entire block means that we gonna log into a remote server using sshpass -p with the credentials we provide it from the previous inputs which are the RMSPW , RMSUS and RMSIP then run all the commands the are between the quote “ “ and save them into a txt file to inspect them latter

Normally SSH does NOT allow passing a password automatically.

It wants you to type it manually.

sshpass bypasses that by injecting the password non-interactively.

Which is perfect for automation task as we want as minimum interaction as possible with the terminal we just want to connect to the remote server and run the commands

so basically wit sshpass -p “RMSPW” we tell the ssh when it asks for password , respond with this one

By default, SSH will STOP the connection the first time you connect to a new server and show this message:

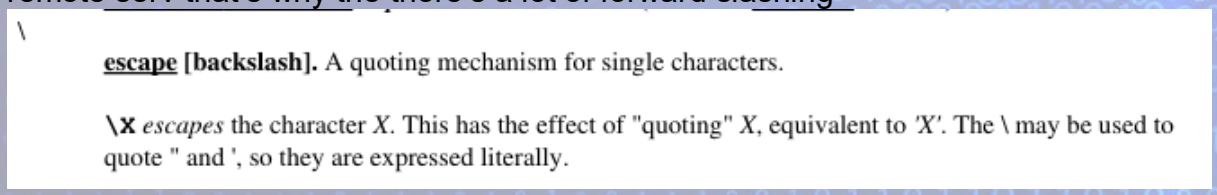
```
~/Desktop
❯ ssh yan@192.168.188.134
The authenticity of host '192.168.188.134 (192.168.188.134)' can't be established.
ED25519 key fingerprint is: SHA256:oO2W4HNIo7oIaN+NadPdF6ceqeedo5j5IkEMOCZqaaY
This key is not known by any other names.
Are you sure you want to continue connecting (yes/no/[fingerprint])? 
```

StrictHostKeyChecking=no disables this prompt.

```
~/Desktop
❯ sshpass -p "1" ssh -o StrictHostKeyChecking=no yan@192.168.188.134
Welcome to Ubuntu 22.04.5 LTS (GNU/Linux 6.8.0-87-generic x86_64)

 * Documentation:  https://help.ubuntu.com
 * Management:     https://landscape.canonical.com
 * Support:        https://ubuntu.com/pro

Expanded Security Maintenance for Applications is not enabled.

0 updates can be applied immediately.

10 additional security updates can be applied with ESM Apps.
Learn more about enabling ESM Apps service at https://ubuntu.com/esm

Last login: Fri Nov 14 20:27:59 2025 from 192.168.188.1
yan@soc:~$ 
```

4.1 SSH & Command execution

notice the entire remote code is inside double quote “ “
and inside that code we want to the remote server shell to interpret variables like $IP , $(hostname) , $TARGET .
if we don’t escape them , the local bash will expand them before sending to the remote serv that’s why the there’s a lot of forward slashing



\

escape [backslash]. A quoting mechanism for single characters.

\X *escapes* the character *X*. This has the effect of "quoting" *X*, equivalent to *'X'*. The \ may be used to quote " and ', so they are expressed literally.

if we won’t escape them, our local machine will replace it with its own local hostname and send that to the remote serv instead of letting the remote serv run the command.
\$IP means:

“Send the literal characters $IP to the remote server so it expands them.”
\$(hostname) means:
“Tell the remote server to run hostname, NOT the local machine.”
First block focuses on system info with commands like $(hostname) , uptime .
Second block will provide us a public ip by using external service **$(curl -s ifconfig.co)**
and print the geographical location of the printed public ip with **geoiplookup \”\$IP\”**
Both of these blocks give us the information of the remote server
Third block is going to enumerate the Target with the command whois followed by **\”$Target\”**
and finally the fourth block which is the nmap scanning (network mapper)
which is going to perform an advanced scanning
nmap -sV -Pn \”$TARGET\”
-sV — detects what service is running , version , product name
-T4 — faster timing, suitable for speed vs stealth balance
--top-ports 200 — scan the most commonly used 200 TCP ports
another alternative is :
--script=vuln — which is an nmap scripting engine (NSE) it runs all vulnerability detection scripts
like smb-vuln-ms17-010 , ftp-anon , etc etc etc it checks for **CVE** (common vulnerabilities and exposures) which is an important information and finally
> remote_results.txt to take everything that is printed within the double quotes “ “ and save it locally on a text file

*for further acquiring knowledge on nmap i recommend on visiting their site which provided me a lot of information and understanding of how it works
https://nmap.org/book/man.html -- this is a manual book

5. Data collection

1.

```
# ================================
# AUDIT LOGGING FUNCTION
# ================================
log_action() {
    local action="$1"
    local comment="$2"
    echo "$(date '+%Y-%m-%d %H:%M:%S') INFO: $action - $comment" >> "$AuditLog"
}
```

2.

```
    echo -e "${YELLOW}============= Saving results =============${RESET}"
    {
        echo "============================================="
        echo " Scan Audit Record"
        echo " Timestamp        : $(date)"
        echo " Remote Server IP : $RMSIP"
        echo " Remote Username  : $RMSUS"
        echo " Target Scanned   : $TARGET"
        echo " Report Saved To  : $RemoteScanResults/${TARGET}_report_$(date +%F_%H-%M).txt"
    } >> "$AuditLog"

    echo -e "${GREEN}================== Done ==================${RESET}"
}
RMT_VPS
```

```
AuditLog="$HOME/Desktop/audit.log"
touch "$AuditLog"
```

log_action "install $A" "$A installed"
log_action "Check $A" "$A is already installed"
log_action "Check Anonymity" "Real IP detected: $IP"
log_action "Check Anonymity" "Spoofed country: $COUNTRY (IP: $IP)"
log_action "Retrieve Remote Details" "Server IP: $RMSIP"
log_action "Set Scan Target" "Target: $TARGET"
log_action "Save Scan Report" "Saved results for $TARGET"
log_action "WHOIS Lookup" "Completed WHOIS for $TARGET"
log_action "Nmap Scan" "Completed Nmap scan for $TARGET"

All these are part of the data collection , log (style auth.log) but just for the script and results of the scanning + enumeration of the target
log_action() — defines a Bash function named **log_action**
local action="$1" — stores the **first argument** into a local variable called action
local comment="$2" — stores the **second argument** into a variable called comment
date '+%Y-%m-%d %H:%M:%S' — generates a timestamp like “2025-11-21 23:38:03”

6. Results

```
❯ bash TMagen773638.s26.NX201.sh.sh
🔍 Checking and installing required applications ...
═══════════════════════════════════════

✅ sshpass is already installed.
✅ tor is already installed.
✅ nmap is already installed.
✅ whois is already installed.
✅ Geoiplookup is already installed.
═══════════════════════════════════════

🔍 Verifying installed tools ...
═══════════════════════════════════════

✔ sshpass is installed successfully.
✔ tor is installed successfully.
✔ nmap is installed successfully.
✔ whois is installed successfully.
✔ geoiplookup is installed successfully.
═══════════════════════════════════════

📥 Installing and setting up Nipe ...
[sudo] password for kali:
Starting nipe service ...

[!] ERROR: sorry, it was not possible to establish a connection to the server.

═══════════════════════════════════════
═══════════════════════════════════════

🔎 Checking network anonymity ...
═══════════════════════════════════════

✅ You're anonymous!
🌍 IP: 45.141.215.88
📌 Spoofed Country: PL, Poland
═══════════════════════════════════════
═══════════════════════════════════════

PW of the Remote Server: kali
USER of the Remote Server: kali
IP of the Remote Server: 192.168.188.137
Enter target domain/IP for scanning: 192.168.188.130
[+] Running scan on remote VPS ...
═══ REPORT SAVED ON VPS: /home/kali/scan_reports/192.168.188.130_report_2025-11-22_00-10.txt ═══
═══════════ Saving results ═══════════
═══════════════ Done ═══════════════
```

Audit log : which is saved in **desktop**

audit.log | 192.168.188.130_report_2025-11-22_00-10.txt | TMagen773638.s26.NX201.sh.sh

```
1  2025-11-22 00:08:54 INFO: Check sshpass - sshpass is already installed
2  2025-11-22 00:08:54 INFO: Check tor - tor is already installed
3  2025-11-22 00:08:54 INFO: Check nmap - nmap is already installed
4  2025-11-22 00:08:54 INFO: Check whois - whois is already installed
5  2025-11-22 00:09:57 INFO: Check Anonymity - Spoofed country: PL, Poland (IP: 45.141.215.88)
6  2025-11-22 00:10:06 INFO: Retrieve Remote Details - Server IP: 192.168.188.137
7  2025-11-22 00:10:06 INFO: Set Scan Target - Target: 192.168.188.130
8  2025-11-22 00:11:42 INFO: Save Scan Report - Saved results for 192.168.188.130
9  2025-11-22 00:11:42 INFO: WHOIS Lookup - Completed WHOIS for 192.168.188.130
10 2025-11-22 00:11:42 INFO: Nmap Scan - Completed Nmap scan for 192.168.188.130
11 ==========================================
12  Scan Audit Record
13  Timestamp        : Sat Nov 22 12:11:42 AM EST 2025
14  Remote Server IP : 192.168.188.137
15  Remote Username  : kali
16  Target Scanned   : 192.168.188.130
17  Report Saved To  : /home/kali/Desktop/RemoteScanReport/192.168.188.130_report_2025-11-22_00-11.txt
18
```

6.1 Scanning & enumeration results

Which is saved in desktop -> RemoteScanReport

audit.log | 192.168.188.130_report_2025-11-22_00-10.txt | TMagen773638.s26.NX201.sh.sh

```
========== SYSTEM INFO ==========
Hostname: kali
Uptime:
 00:10:06 up 1 day, 1 min,  2 users,  load average: 0.00, 0.01, 0.00
========== PUBLIC IP ==========
IP: 77.137.77.6
GeoIP Country Edition: IL, Israel
========== WHOIS (192.168.188.130) ==========

#
# ARIN WHOIS data and services are subject to the Terms of Use
# available at: https://www.arin.net/resources/registry/whois/tou/
#
# If you see inaccuracies in the results, please report at
# https://www.arin.net/resources/registry/whois/inaccuracy_reporting/
#
# Copyright 1997-2025, American Registry for Internet Numbers, Ltd.
#


NetRange:       192.168.0.0 - 192.168.255.255
CIDR:           192.168.0.0/16
NetName:        PRIVATE-ADDRESS-CBLK-RFC1918-IANA-RESERVED
NetHandle:      NET-192-168-0-0-1
Parent:         NET192 (NET-192-0-0-0-0)
NetType:        IANA Special Use
OriginAS:
Organization:   Internet Assigned Numbers Authority (IANA)
RegDate:        1994-03-15
Updated:        2024-05-24
Comment:        These addresses are in use by many millions of independently operated networks, which might be as small as a single computer con
and are automatically configured in hundreds of millions of devices.  They are only intended for use within a private context  and traffic that
will need to use a different, unique address.
Comment:
Comment:        These addresses can be used by anyone without any need to coordinate with IANA or an Internet registry.  The traffic from these
ICANN or IANA.  We are not the source of activity you may see on logs or in e-mail records.  Please refer to http://www.iana.org/abuse/answers
Comment:
Comment:        These addresses were assigned by the IETF, the organization that develops Internet protocols, in the Best Current Practice docume
found at:
Comment:        http://datatracker.ietf.org/doc/rfc1918
Ref:            https://rdap.arin.net/registry/ip/192.168.0.0
```

audit.log | 192.168.188.130_report_2025-11-22_00-10.txt | TMagen773638.s26.NX201.sh.sh

```
OrgAbuseName:   ICANN
OrgAbusePhone:  +1-310-301-5820
OrgAbuseEmail:  abuse@iana.org
OrgAbuseRef:    https://rdap.arin.net/registry/entity/IANA-IP-ARIN


#
# ARIN WHOIS data and services are subject to the Terms of Use
# available at: https://www.arin.net/resources/registry/whois/tou/
#
# If you see inaccuracies in the results, please report at
# https://www.arin.net/resources/registry/whois/inaccuracy_reporting/
#
# Copyright 1997-2025, American Registry for Internet Numbers, Ltd.
#

========== NMAP (192.168.188.130) ==========
Starting Nmap 7.95 ( https://nmap.org ) at 2025-11-22 00:10 EST
Nmap scan report for 192.168.188.130
Host is up (0.0016s latency).
Not shown: 182 closed tcp ports (reset)
PORT     STATE SERVICE     VERSION
21/tcp   open  ftp         vsftpd 2.3.4
22/tcp   open  ssh         OpenSSH 4.7p1 Debian 8ubuntu1 (protocol 2.0)
23/tcp   open  telnet      Linux telnetd
25/tcp   open  smtp        Postfix smtpd
53/tcp   open  domain      ISC BIND 9.4.2
80/tcp   open  http        Apache httpd 2.2.8 ((Ubuntu) DAV/2)
111/tcp  open  rpcbind     2 (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)
513/tcp  open  login?
514/tcp  open  tcpwrapped
2049/tcp open  nfs         2-4 (RPC #100003)
2121/tcp open  ftp         ProFTPD 1.3.1
3306/tcp open  mysql       MySQL 5.0.51a-3ubuntu5
5432/tcp open  postgresql  PostgreSQL DB 8.3.0 - 8.3.7
5900/tcp open  vnc         VNC (protocol 3.3)
6000/tcp open  X11         (access denied)
8009/tcp open  ajp13?
MAC Address: 00:0C:29:FB:C0:BA (VMware)
Service Info: Host:  metasploitable.localdomain; OSs: Unix, Linux; CPE: cpe:/o:linux:linux_kernel

Service detection performed. Please report any incorrect results at https://nmap.org/submit/ .
Nmap done: 1 IP address (1 host up) scanned in 95.17 seconds
```