Network Security Assessment Tool

User Manual

Ayşe Simge ÖZGER

Emre ÖVÜNÇ

Umut BAŞARAN

Table of Contents

I. Introduction to NSAT

Network Security is an important field that is getting more and more attention as the internet expands. There is a large amount of personal, commercial, military and government information on networking infrastructures worldwide. Network security is becoming of great importance because of intellectual property that can be easily acquired through the internet. The security threats and internet protocols should be analyzed to determine the necessary security technology. This project covers as a very large and important area of computer security: networks and distributed applications. As the world becomes more connected by networks, the significance of network security will certainly continue to grow.

The purpose of the project is detecting and analyzing network threats by sniffing network traffics, reading Pcap files, automatic & manual packet generation and scan & attack target systems. We will develop a set of tools to decode and analyze protocols headers and contents. Following threat identification and classification performs in order to build a dictionary of threats and attacks.

II. Descriptions of NSAT Modules

A. ATTACK MODULE

A.1. Fuzzer

Fuzzer attack is designed for bruteforcing Web Applications, it can be used for finding resources not linked (directories, servlets, scripts, etc), bruteforce GET and POST parameters for checking different kind of injections (SQL, XSS, LDAP, etc), bruteforce Forms parameters (User/Password), Fuzzing,etc.

A.2. ARP Spoof

In computer networking, ARP spoofing, ARP cache poisoning, or ARP poison routing, is a technique by which an attacker sends (spoofed) Address Resolution Protocol (ARP) messages onto a local area network. Generally, the aim is to associate the attacker's MAC address with the IP address of another host, such as the default gateway, causing any traffic meant for that IP address to be sent to the attacker instead.

ARP spoofing may allow an attacker to intercept data frames on a network, modify the traffic, or stop all traffic. Often the attack is used as an opening for other attacks, such as denial of service, man in the middle, or session hijacking attacks.

A.3. SSH Overload

SSH Overload is one of the DoS attack methods. A denial-of-service attack can exploit vulnerabilities in a physical infrastructure to deny legitimate users access to computer or network resources. While many service interruptions are disrupted by network attacks, the same result can also be achieved by physically severing wiring or preventing power and cooling resources from being accessed. In such cases, the physical attack may also be referred to as sabotage.

A.4. DoS

Denial-of-service (DoS) attacks typically flood servers, systems or networks with traffic in order to overwhelm the victim resources and make it difficult or impossible for legitimate users to use them. While an attack that crashes a server can often be dealt with successfully by simply rebooting the system, flooding attacks can be more difficult to recover from.

A.5. TCP Flood

TCP SYN floods are one of the oldest yet still very popular Denial of Service (DoS) attacks. The most common attack involves sending numerous SYN packets to the victim.
The attack in many cases will spoof the SRC IP meaning that the reply (SYN+ACK packet) will not come back to it.

The intention of this attack is overwhelm the session/connection tables of the targeted server or one of the network entities on the way (typically the firewall). Servers need to open a state for each SYN packet that arrives and they store this state in tables that have limited size. As big as this table may be it is easy to send sufficient amount of SYN packets that will fill the table, and once this happens the server starts to drop a new request, including legitimate ones. Similar effects can happen on a firewall which also has to process and invest in each SYN packet.

Unlike other TCP or application level attacks the attacker does not have to use a real IP; this is perhaps the biggest strength of the attack.

A.6. ICMP Flood

Internet Control Message Protocol (ICMP) is a connectionless protocol used for IP operations, diagnostics, and errors. An ICMP Flood - the sending of an abnormally large number of ICMP packets of any type (especially network latency testing “ping” packets) - can overwhelm a target server that attempts to process every incoming ICMP request, and this can result in a denial-of-service condition for the target server.

The attack involves flooding the victim's network with request packets, knowing that the network will respond with an equal number of reply packets. Additional methods for bringing down a target with ICMP requests include the use of custom tools or code, such as hping and scapy.

A.7. SYN Flood

A SYN flood is a form of denial-of-service attack in which an attacker sends a succession of SYN requests to a target's system in an attempt to consume enough server resources to make the system unresponsive to legitimate traffic.

A SYN flood attack works by not responding to the server with the expected ACK code. The malicious client can either simply not send the expected ACK, or by spoofing the source IP address in the SYN, causing the server to send the SYN-ACK to a falsified IP address - which will not send an ACK because it "knows" that it never sent a SYN.

The server will wait for the acknowledgement for some time, as simple network congestion could also be the cause of the missing ACK. However, in an attack, the *half-open connections* created by the malicious client bind resources on the server and may eventually exceed the resources available on the server. At that point, the server cannot connect to any clients, whether legitimate or otherwise. This effectively denies service to legitimate clients. Some systems may also malfunction or crash when other operating system functions are starved of resources in this way.

B. GENERATION MODULE

B.1. ARP Packet

Address Resolution Protocol (ARP) is a protocol for mapping an Internet Protocol address (IP address) to a physical machine address that is recognized in the local network. For example, in IP Version 4, the most common level of IP in use today, an address is 32 bits long. In an Ethernet local area network, however, addresses for attached devices are 48 bits long. (The physical machine address is also known as a Media Access Control or MAC address.) A table, usually called the ARP cache, is used to maintain a correlation between each MAC address and its corresponding IP address. ARP provides the protocol rules for making this correlation and providing address conversion in both directions.

Systems keep an ARP look-up table where they store information about what IP addresses are associated with what MAC addresses. When trying to send a packet to an IP address, the system will first consult this table to see if it already knows the MAC address. If there is a value cached, ARP is not used.

B.2. TCP Packet

TCP (Transmission Control Protocol) is a standard that defines how to establish and maintain a network conversation via which application programs can exchange data. TCP works with the Internet Protocol (IP), which defines how computers send packets of data to each other. Together, TCP and IP are the basic rules defining the Internet. TCP is defined by the Internet Engineering Task Force (IETF) in the Request for Comment (RFC) standards document number 793.

TCP is a connection-oriented protocol, which means a connection is established and maintained until the application programs at each end have finished exchanging messages. It determines how to break application data into packets that networks can deliver, sends packets to and accepts packets from the network layer, manages flow control, and—because it is meant to provide error-free data transmission—handles retransmission of dropped or garbled packets as well as acknowledgement of all packets that arrive. In the Open Systems Interconnection (OSI) communication model, TCP covers parts of Layer 4, the Transport Layer, and parts of Layer 5, the Session Layer.

B.3. UDP Packet

The User Datagram Protocol offers only a minimal transport service -- non-guaranteed datagram delivery -- and gives applications direct access to the datagram service of the IP layer. UDP is used by applications that do not require the level of service of TCP or that wish to use communications services (e.g., multicast or broadcast delivery) not available from TCP.

UDP is almost a null protocol; the only services it provides over IP are checksumming of data and multiplexing by port number. Therefore, an application program running over UDP must deal directly with end-to-end communication problems that a connection-oriented protocol would have handled -- e.g., retransmission for reliable delivery, packetization and reassembly, flow control, congestion avoidance, etc., when these are required. The fairly complex coupling between IP and TCP will be mirrored in the coupling between UDP and many applications using UDP.

B.4. IP Packet

An IP header is header information at the beginning of an IP packet which contains information about IP version, source IP address, destination IP address, time-to-live, etc.

Two different versions of IP headers have been defined, IPv4 and IPv6. The IPv6 header uses IPv6 addresses and thus offers a much bigger address space, but is not backwards compatible with IPv4

IPv4 is the fourth version in the development of the Internet Protocol (IP), and routes most traffic on the Internet. IPv4 is described in IETF publication RFC 791, replacing an earlier definition.

IPv4 is a connectionless protocol for use on packet-switched networks. It operates on a best effort delivery model, in that it does not guarantee delivery, nor does it assure proper sequencing or avoidance of duplicate delivery. These aspects, including data integrity, are addressed by an upper layer transport protocol, such as the Transmission Control Protocol (TCP).

IPv6, the successor to IPv4, has been defined and is in various stages of production deployment, and has a different header layout. An IPv6 packet is the smallest message entity exchanged via the Internet Protocol across an IPv6 network. Packets consist of control information for addressing and routing, and a payload consisting of user data. The control information in IPv6 packets is subdivided into a mandatory fixed header and optional extension headers. The payload of an IPv6 packet is typically a datagram or segment of the higher-level Transport Layer protocol, but may be data for an Internet Layer (e.g., ICMPv6) or Link Layer instead.

B.5. ICMP Packet

ICMP (Internet Control Message Protocol) is an error-reporting protocol network devices like routers use to generate error messages to the source IP address when network problems prevent delivery of IP packets. ICMP creates and sends messages to the source IP address indicating that a gateway to the Internet that a router, service or host cannot be reached for packet delivery. Any IP network device has the capability to send, receive or process ICMP messages.

ICMP is *not* a transport protocol that sends data between systems.

While ICMP is not used regularly in end-user applications, it is used by network administrators to troubleshoot Internet connections in diagnostic utilities including ping and traceroute.

One of the main protocols of the Internet Protocol suite, ICMP is used by routers, intermediary devices or hosts to communicate error information or updates to other routers, intermediary devices or hosts. The widely used IPv4 (Internet Protocol version 4) and the newer IPv6 use similar versions of the ICMP protocol (ICMPv4 and ICMPv6, respectively).

B.6. DNS Packet

The Domain Name System (DNS) is a hierarchical decentralized naming system for computers, services, or other resources connected to the Internet or a private network. It associates various information with domain names assigned to each of the participating entities. Most prominently, it translates more readily memorized domain names to the numerical IP addresses needed for locating and identifying computer services and devices with the underlying network protocols. By providing a worldwide, distributed directory service, the Domain Name System is an essential component of the functionality of the Internet, that has been in use since 1985.

The DNS protocol uses two types of DNS messages, queries and replies, and they both have the same format. Each message consists of a header and four sections: question, answer, authority, and an additional space. A header field (*flags*) controls the content of these four sections.

The header section contains the following fields: *Identification*, *Flags*, *Number of questions*, *Number of answers*, *Number of authority resource records* (RRs), and *Number of additional RRs*. The identification field can be used to match responses with queries. The flag field consists of several sub-fields. The first is a single bit which indicates if the message is a query (0) or a reply (1). The second sub-field consists of four bits; if the value is 1, the present packet is a reply; if it is 2, the present packet is a status; if the value is 0, the present packet is a request.

C. PACKET ANALYZING MODULE

A packet analyzer (also known as a network analyzer, protocol analyzer or packet sniffer—or, for particular types of networks, an Ethernet sniffer or wireless sniffer) is a computer program or piece of computer hardware that can intercept and log traffic that passes over a digital network or part of a network. As data streams flow across the network, the sniffer captures each packet and, if needed, decodes the packet's raw data, showing the values of various fields in the packet, and analyzes its content according to the appropriate RFC or other specifications.

D. SNIFFER MODULE

Packet sniffers work by intercepting and logging network traffic that they can 'see' via the wired or wireless network interface that the packet sniffing software has access to on its host computer.

On a wired network, what can be captured depends on the structure of the network. A packet sniffer might be able to see traffic on an entire network or only a certain segment of it, depending on how the network switches are configured, placed, etc. On wireless networks, packet sniffers can usually only capture one channel at a time unless the host computer has multiple wireless interfaces that allow for multichannel capture.

Once the raw packet data is captured, the packet sniffing software must analyze it and present it in human-readable form so that the person using the packet sniffing software can make sense of it. The person analyzing the data can view details of the 'conversation' happening between two or more nodes on the network.

Sniffer module prints the contents of network packets. It can read packets from a network interface card or from a previously created saved packet file. It can write packets to standard output or a file. It is also possible to use this module for the specific purpose of intercepting and displaying the communications of another user or computer. A user with the necessary privileges on a system acting as a router or gateway through which unencrypted traffic such as Telnet or HTTP passes can use tcpdump to view login IDs, passwords, the URLs and content of websites being viewed, or any other unencrypted information.

E. RECON MODULE

E.1. Subdomain Discovery

The Domain Name System (DNS) has a tree structure or hierarchy, with each non-RR (resource record) node on the tree being a domain name. A subdomain is a domain that is part of a larger domain; the only domain that is not also a subdomain is the root domain.[1] For example, west.example.com and east.example.com are subdomains of the example.com domain, which in turn is a subdomain of the com top-level domain (TLD). A "subdomain" expresses relative dependence, not absolute dependence: for example, example.org comprises a subdomain of the org domain, and en.example.org comprises a subdomain of the domain example.org. In theory this subdivision can go down to 127 levels deep, and each DNS label can contain up to 63 characters, as long as the whole domain name does not exceed a total length of 255 characters, but in practice most domain registries limit at 253 characters.

E.2. Banner Detection

Banner grabbing is a technique used to glean information about a computer system on a network and the services running on its open ports. Administrators can use this to take inventory of the systems and services on their network. However, an intruder can use banner grabbing in order to find network hosts that are running versions of applications and operating systems with known exploits.

Some examples of service ports used for banner grabbing are those used by Hyper Text Transfer Protocol (HTTP), File Transfer Protocol (FTP), and Simple Mail Transfer Protocol (SMTP); ports 80, 21, and 25 respectively. Tools commonly used to perform banner grabbing are Telnet, which is included with most operating systems, and Netcat.

To prevent this, network administrators should restrict access to services on their networks and shut down unused or unnecessary services running on network hosts.

E.3. Traceroute

Traceroute is a utility that records the route (the specific gateway computers at each hop) through the Internet between your computer and a specified destination computer. It also calculates and displays the amount of time each hop took. Traceroute is a handy tool both for understanding where problems are in the Internet network and for getting a detailed sense of the Internet itself. Another utility, PING, is often used prior to using traceroute to see whether a host is present on the network.

The traceroute utility comes included with a number of operating systems, including Windows and UNIX-based operating systems (such as IBM's AIX/6000) or as part of a TCP/IP package. If your system doesn't include the utility, you can install it. There are freeware versions that you can download.

When you enter the traceroute command, the utility initiates the sending of a packet (using the Internet Control Message Protocol or ICMP), including in the packet a time limit value (known as the "time to live" (TTL) that is designed to be exceeded by the first router that receives it, which will return a Time Exceeded message. This enables traceroute to determine the time required for the hop to the first router. Increasing the time limit value, it resends the packet so that it will reach the second router in the path to the destination, which returns another Time Exceeded message, and so forth.

E.4. Port Scan

Port Scanning is the name for the technique used to identify open ports and services available on a network host. It is sometimes utilized by security technicians to audit computers for vulnerabilities, however, it is also used by hackers to target victims. It can be used to send requests to connect to the targeted computers, and then keep track of the ports which appear to be opened, or those that respond to the request.

When a criminal targets a house for a burglary, typically the first thing he or she checks is if there is an open window or door through which access to the home can be gained. A Port scan is similar, only the windows and doors are the ports of the individual's personal computer. While a hacker may not decide to "break in" at that moment, he or she will have determined if easy access is available. Many people feel this activity should be illegal, which it is not, however, due to the fact that the potential attacker is merely checking to see if a possible connection could be made, in most areas, it is not considered a crime. However, if repetitive port scans are made, a denial of service can be created.

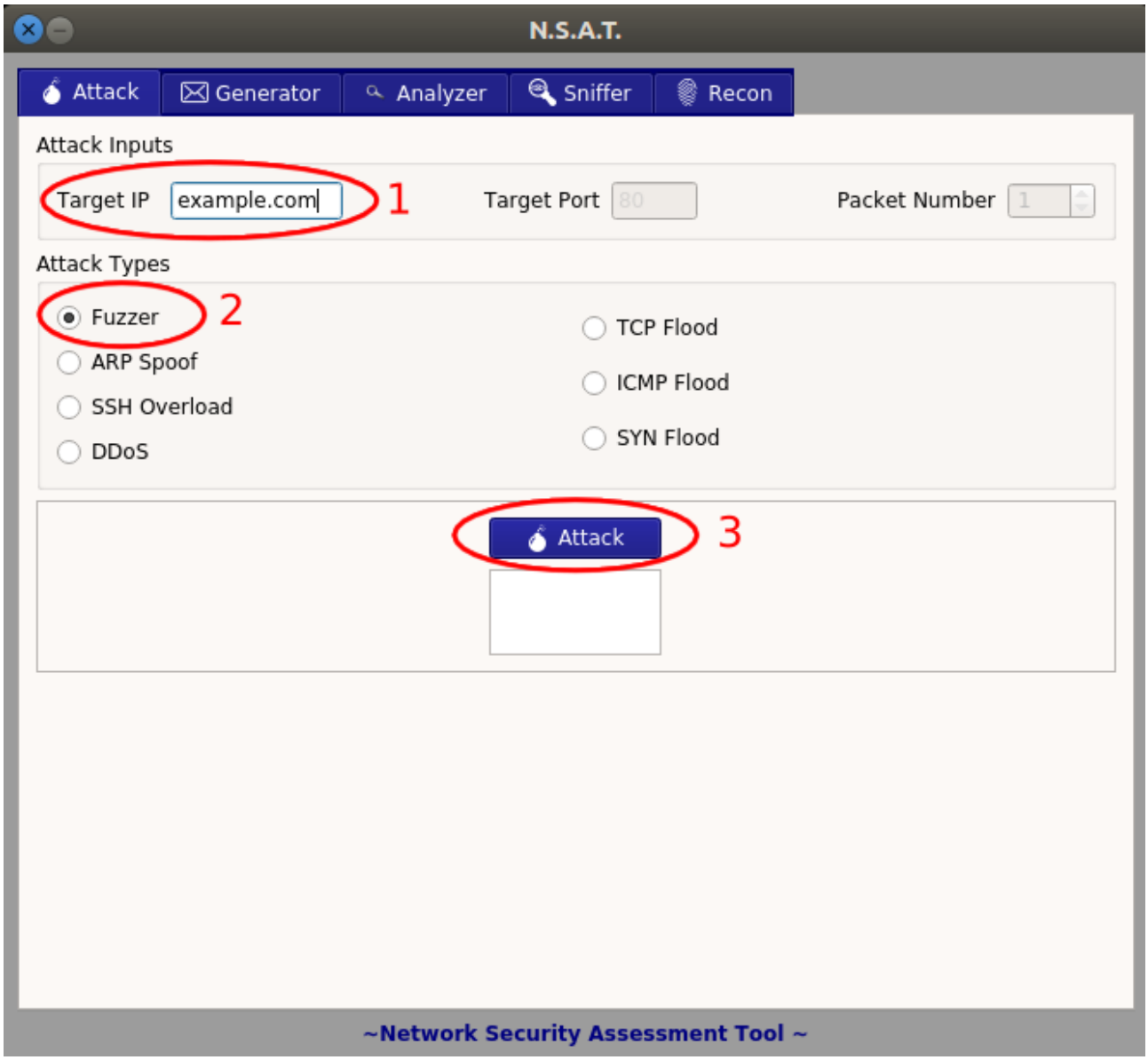
III. Usage of NSAT

A. GUI

A.1. Attack Module

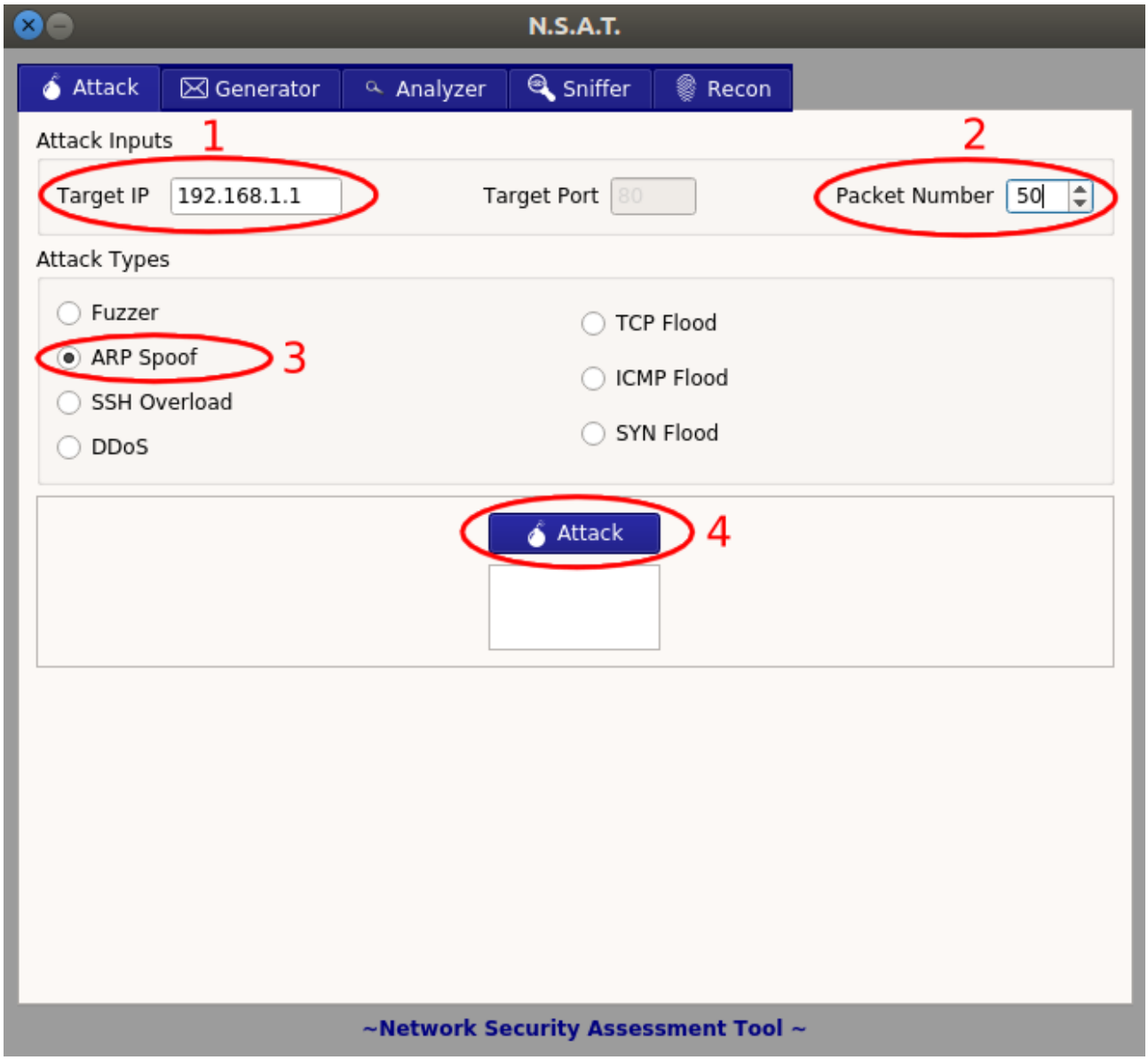
A.1.1. Fuzzer

- The user enters IP address or domain name.
- Press ‘Attack’ button to start Fuzzer attack.



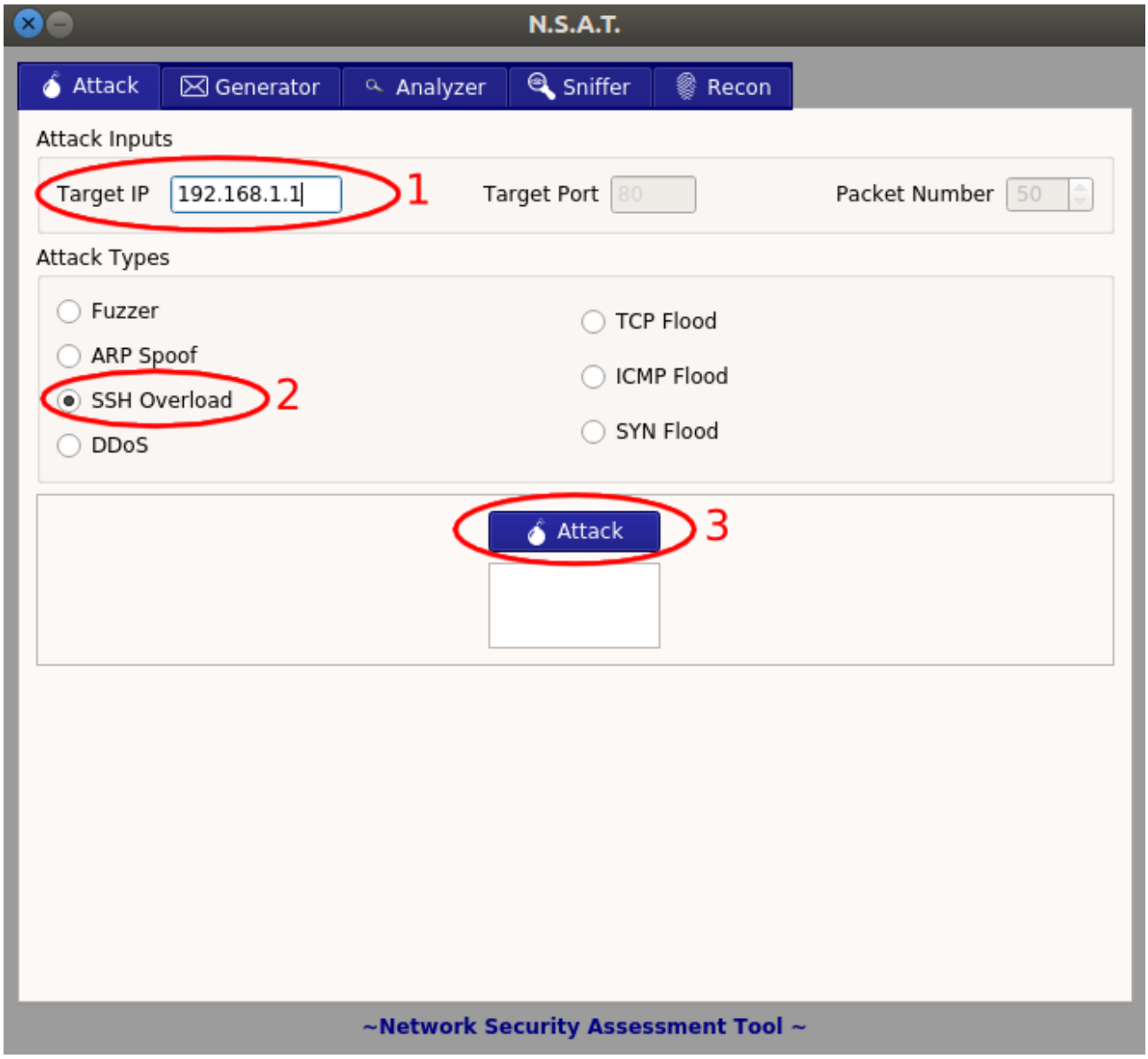
A.1.2. ARP Spoof

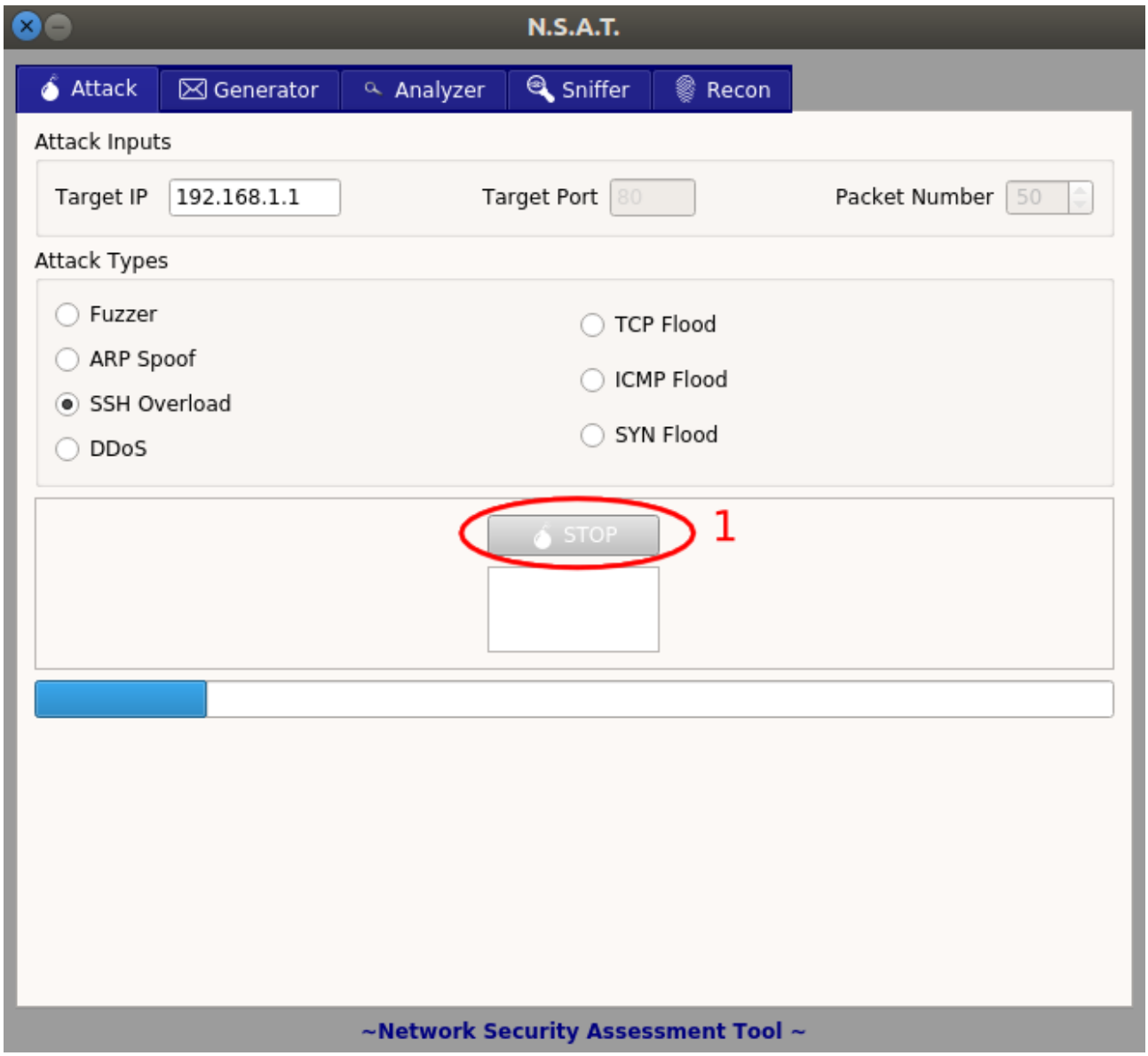
- The user enters an IP address.
- Selects packet number to send given IP address.
- Press ‘Attack’ button to start arp spoofing attack.



A.1.3. SSH Overload

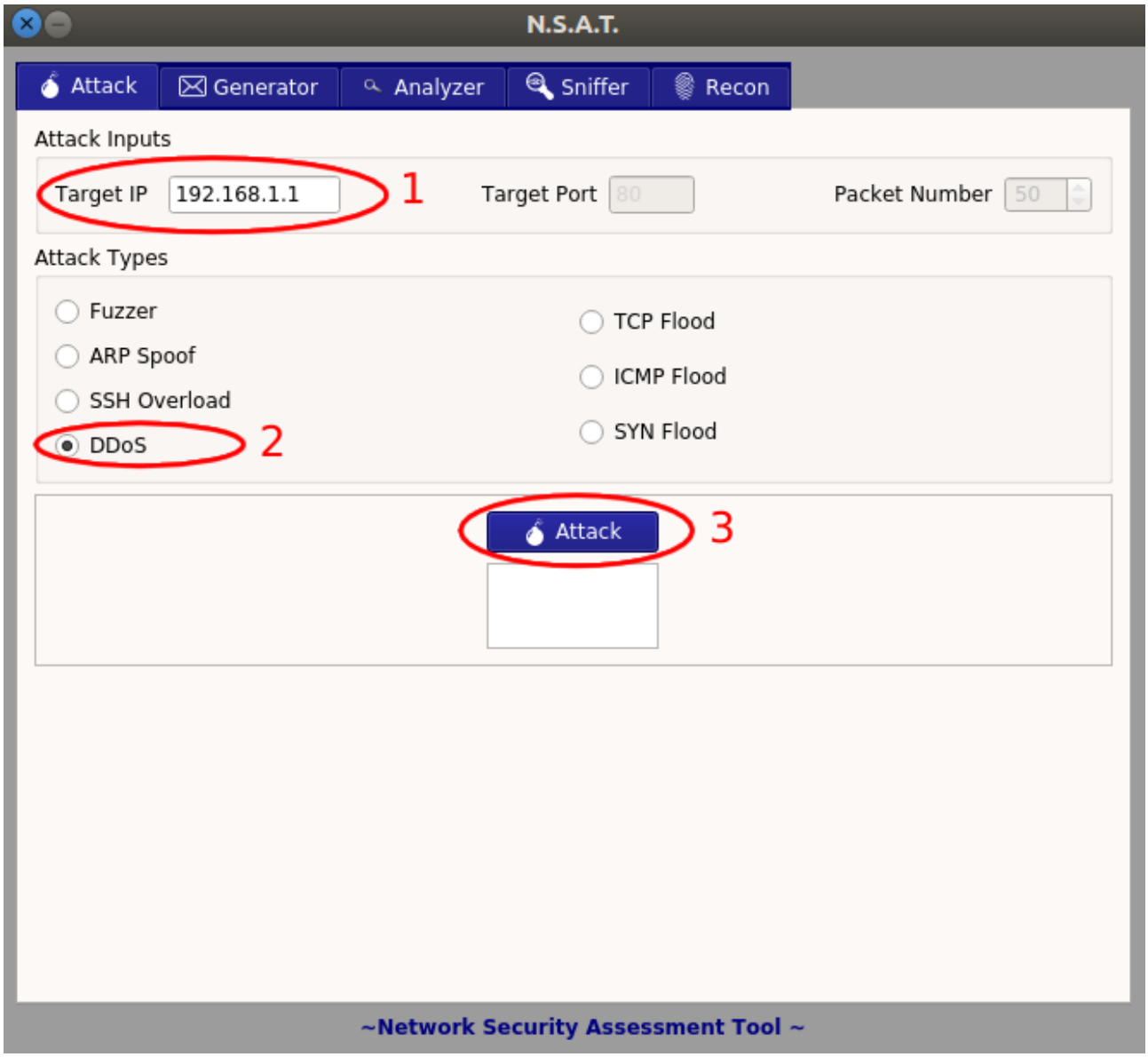
- The user enters an IP address.
- Press ‘Attack’ button to start ssh overloading attack.
- Press ‘Stop’ button to terminate attack.

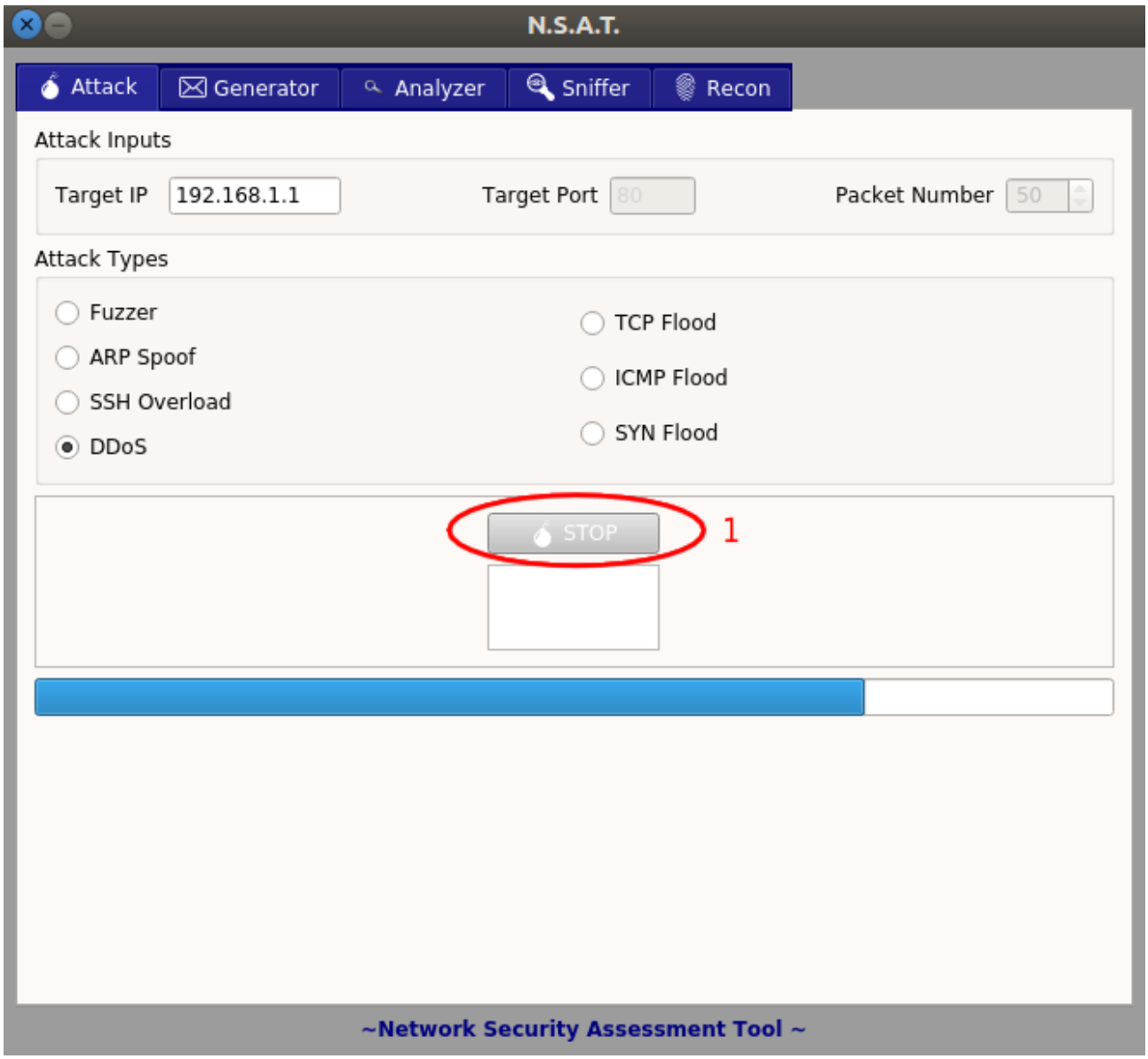



N.S.A.T.
Attack
Generator
Analyzer
Sniffer
Recon
Attack Inputs
Target IP
192.168.1.1
Target Port
80
Packet Number
50
Attack Types
Fuzzer
ARP Spoof
SSH Overload
DDoS
TCP Flood
ICMP Flood
SYN Flood
STOP
1
~Network Security Assessment Tool ~

A.1.4. DoS

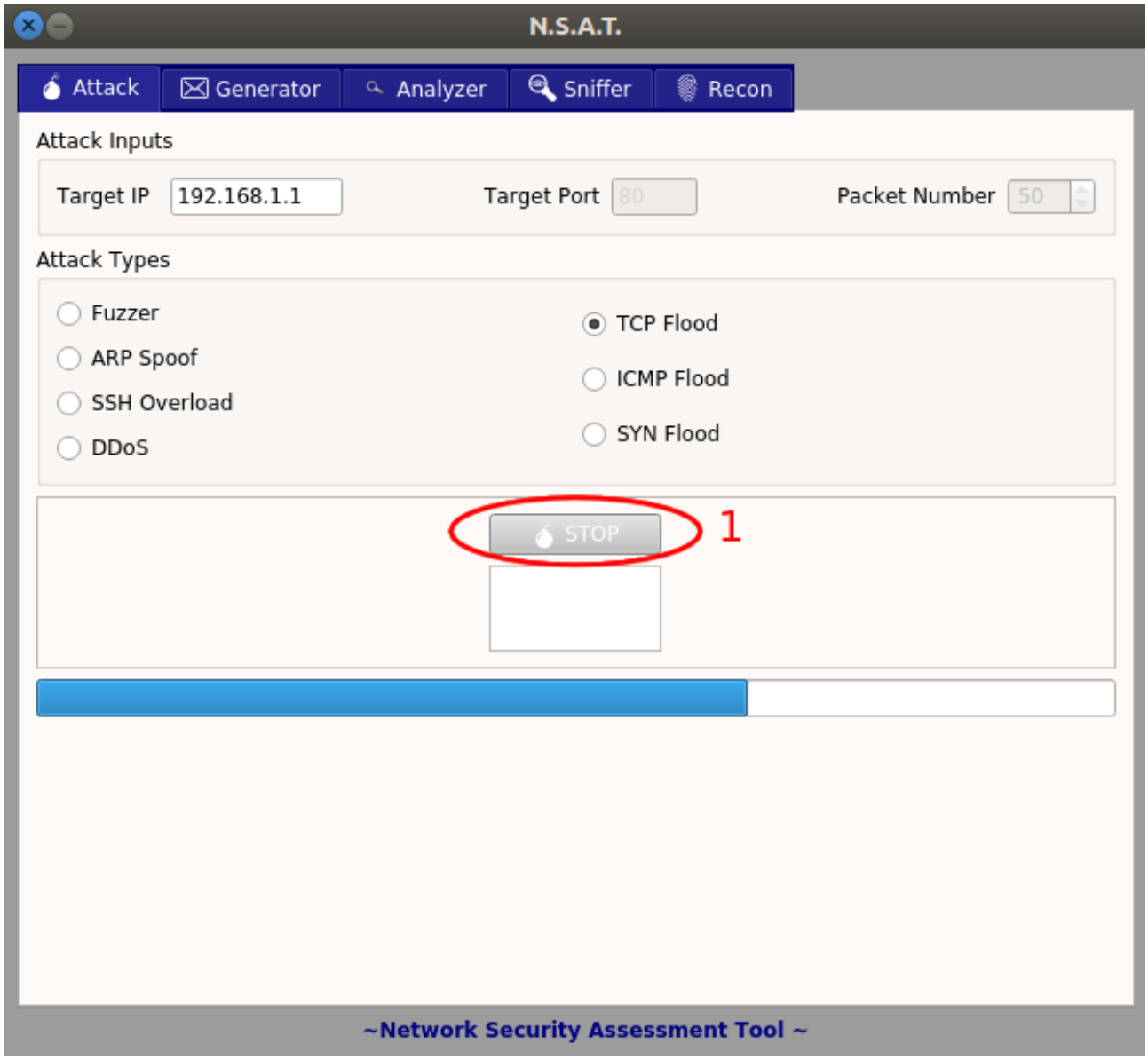
- The user enters an IP address.
- Press ‘Attack’ button to start DoS attack.
- Press ‘Stop’ button to terminate attack.

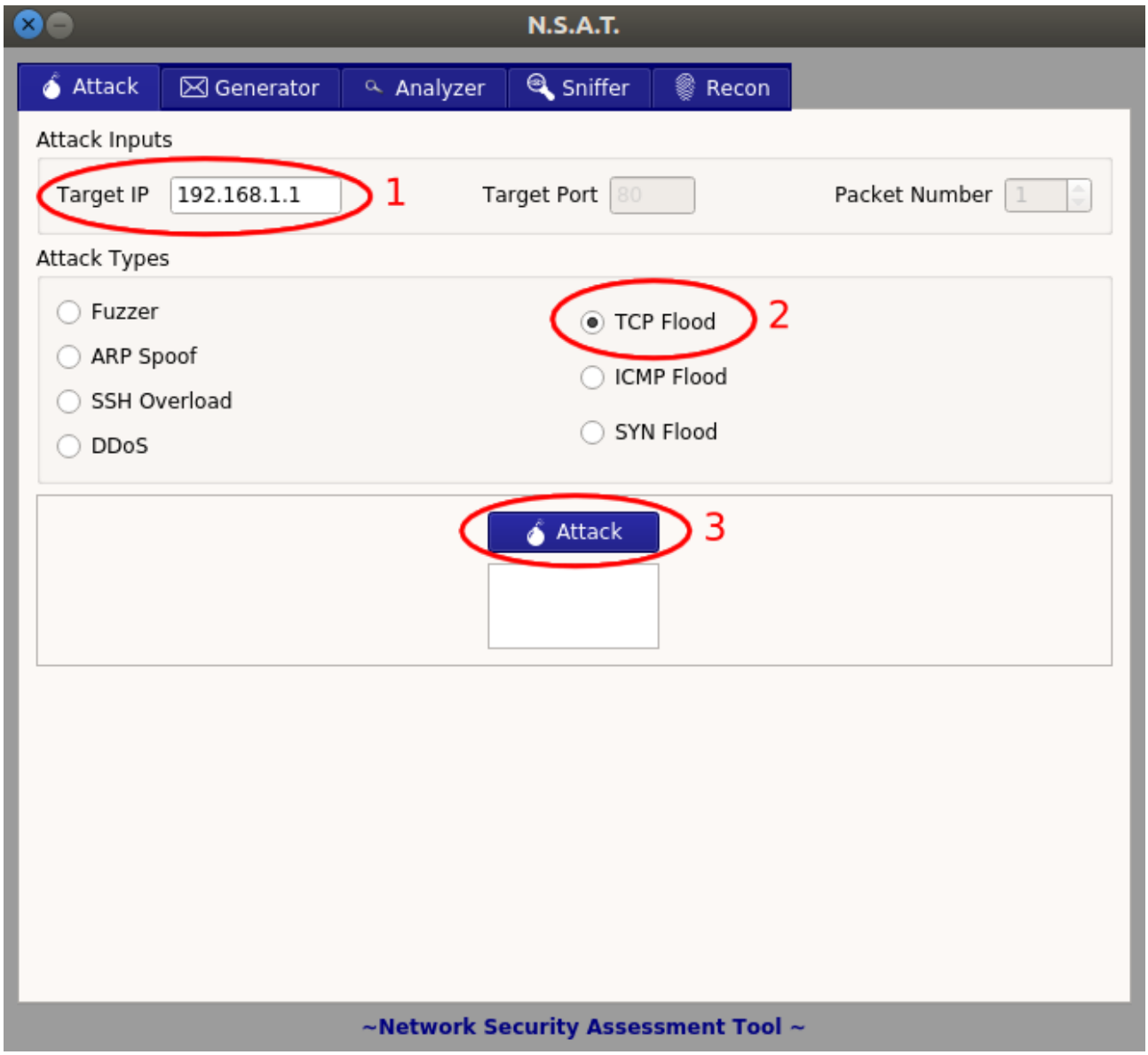



N.S.A.T.
Attack
Generator
Analyzer
Sniffer
Recon
Attack Inputs
Target IP
192.168.1.1
Target Port
80
Packet Number
50
Attack Types
Fuzzer
ARP Spoof
SSH Overload
DDoS
TCP Flood
ICMP Flood
SYN Flood
STOP
1
~Network Security Assessment Tool ~

A.1.5. TCP Flood

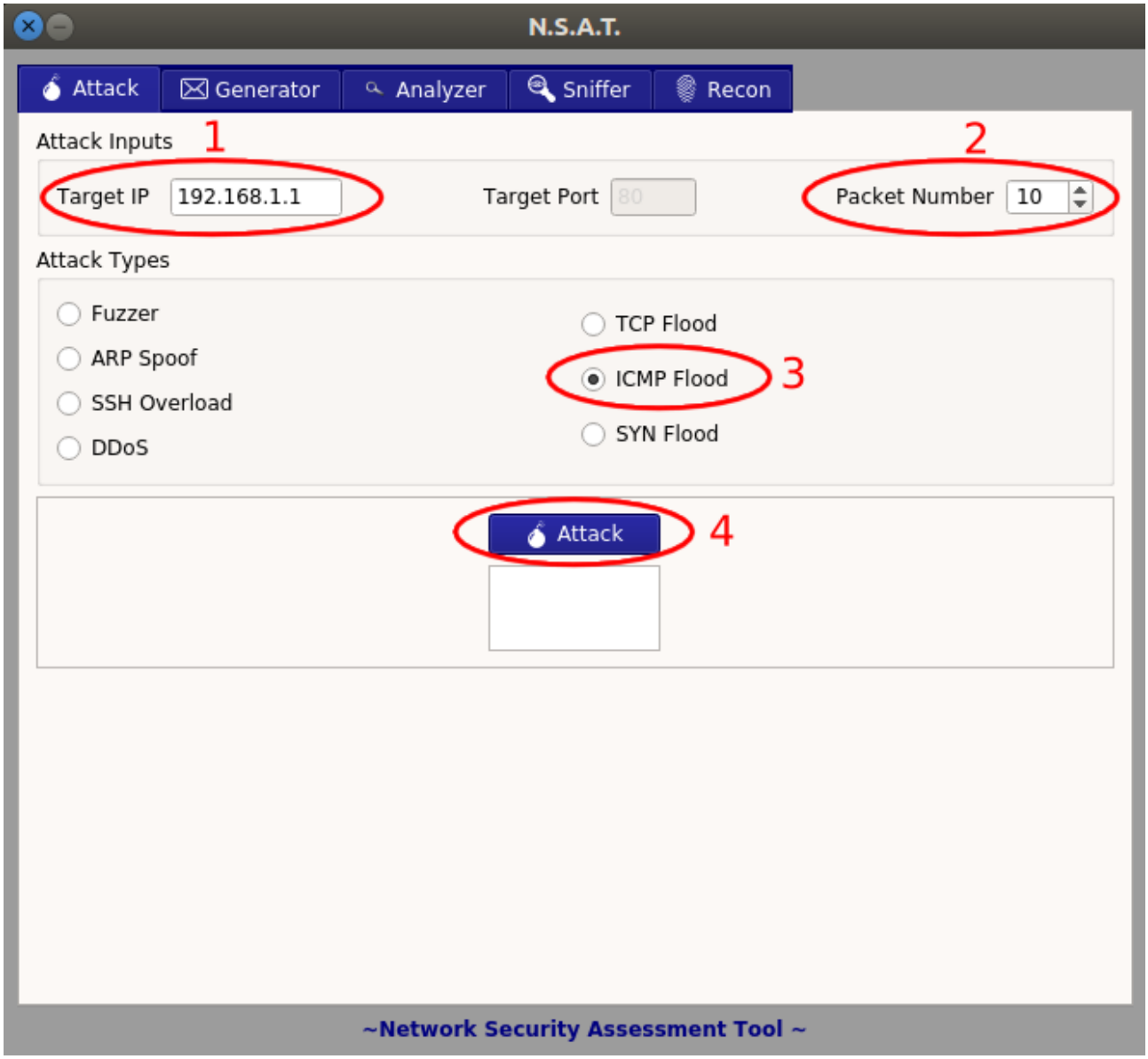
- The user enters an IP address.
- Press ‘Attack’ button to start tcp flooding attack.
- Press ‘Stop’ button to terminate attack.




N.S.A.T.
Attack
Generator
Analyzer
Sniffer
Recon
Attack Inputs
Target IP
192.168.1.1
1
Target Port
80
Packet Number
1
Attack Types
Fuzzer
ARP Spoof
SSH Overload
DDoS
TCP Flood
2
ICMP Flood
SYN Flood
Attack
3
~Network Security Assessment Tool ~

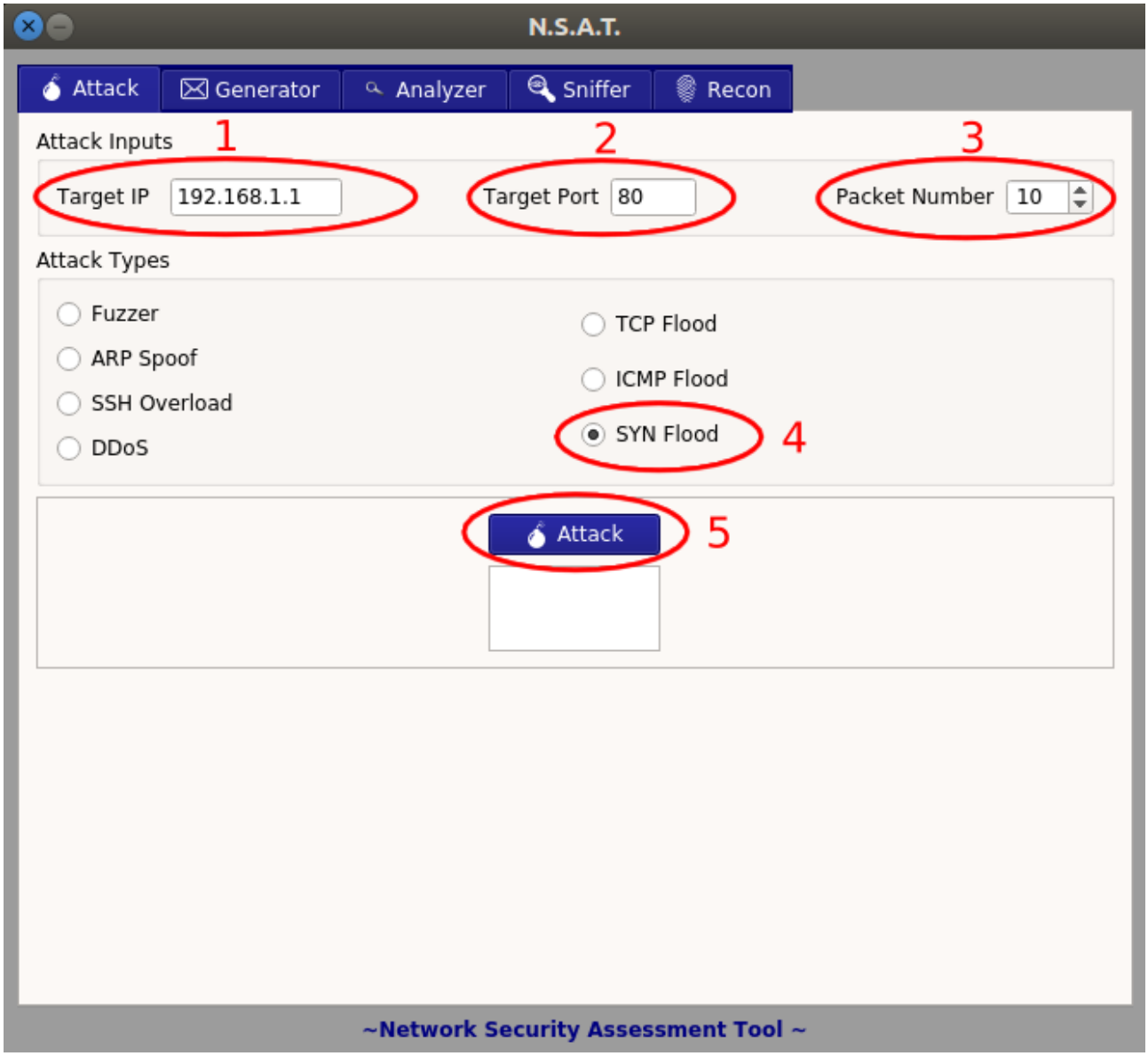
A.1.6. ICMP Flood

- The user enters an IP address.
- Selects packet number to send given IP address.
- Press ‘Attack’ button to start icmp flooding attack.



A.1.7. SYN Flood

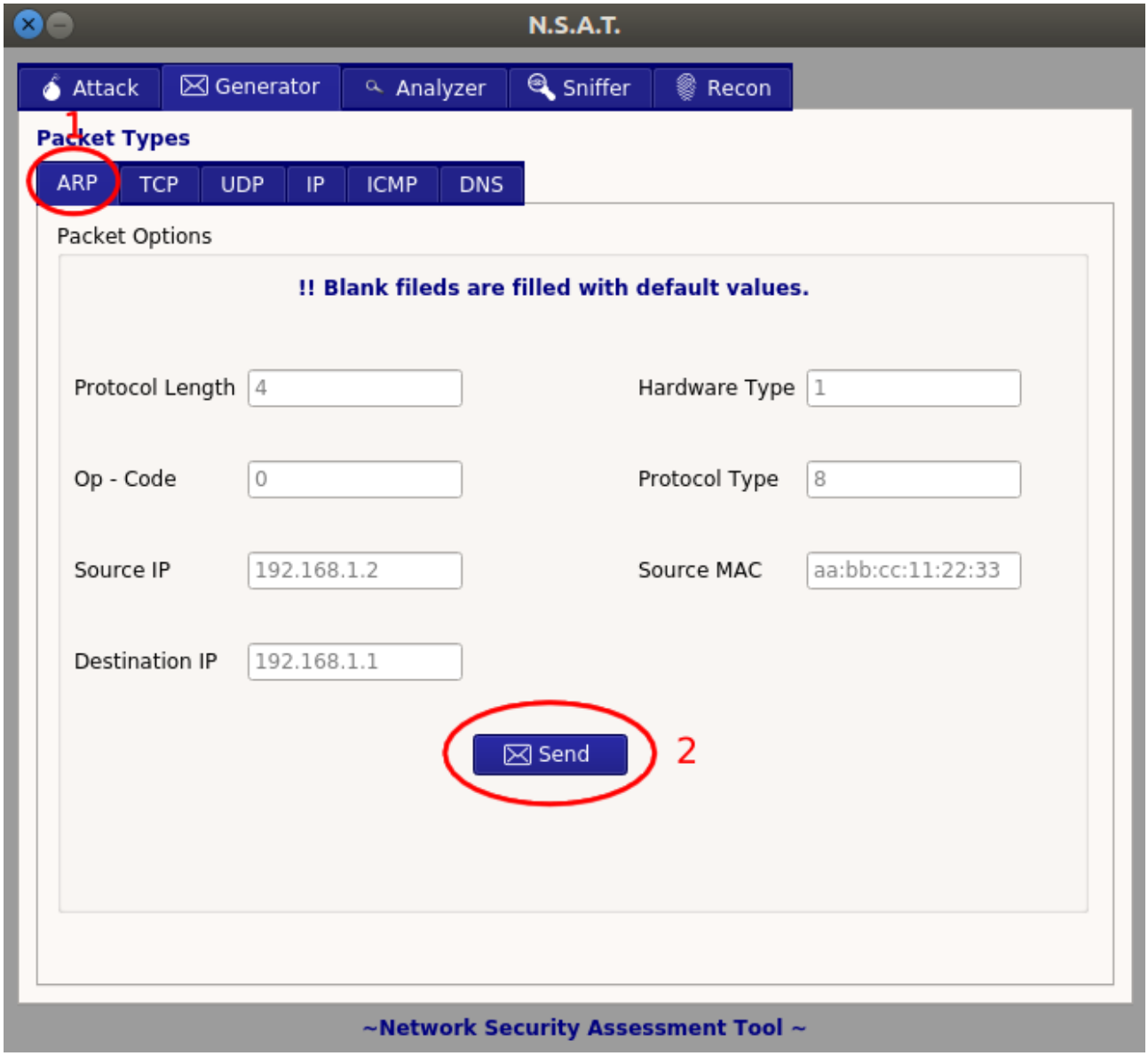
- The user enters an IP address
- Enters a port number
- Selects packet number to send given IP address.
- Press ‘Attack’ button to start syn flooding attack.



A.2. Generation Module

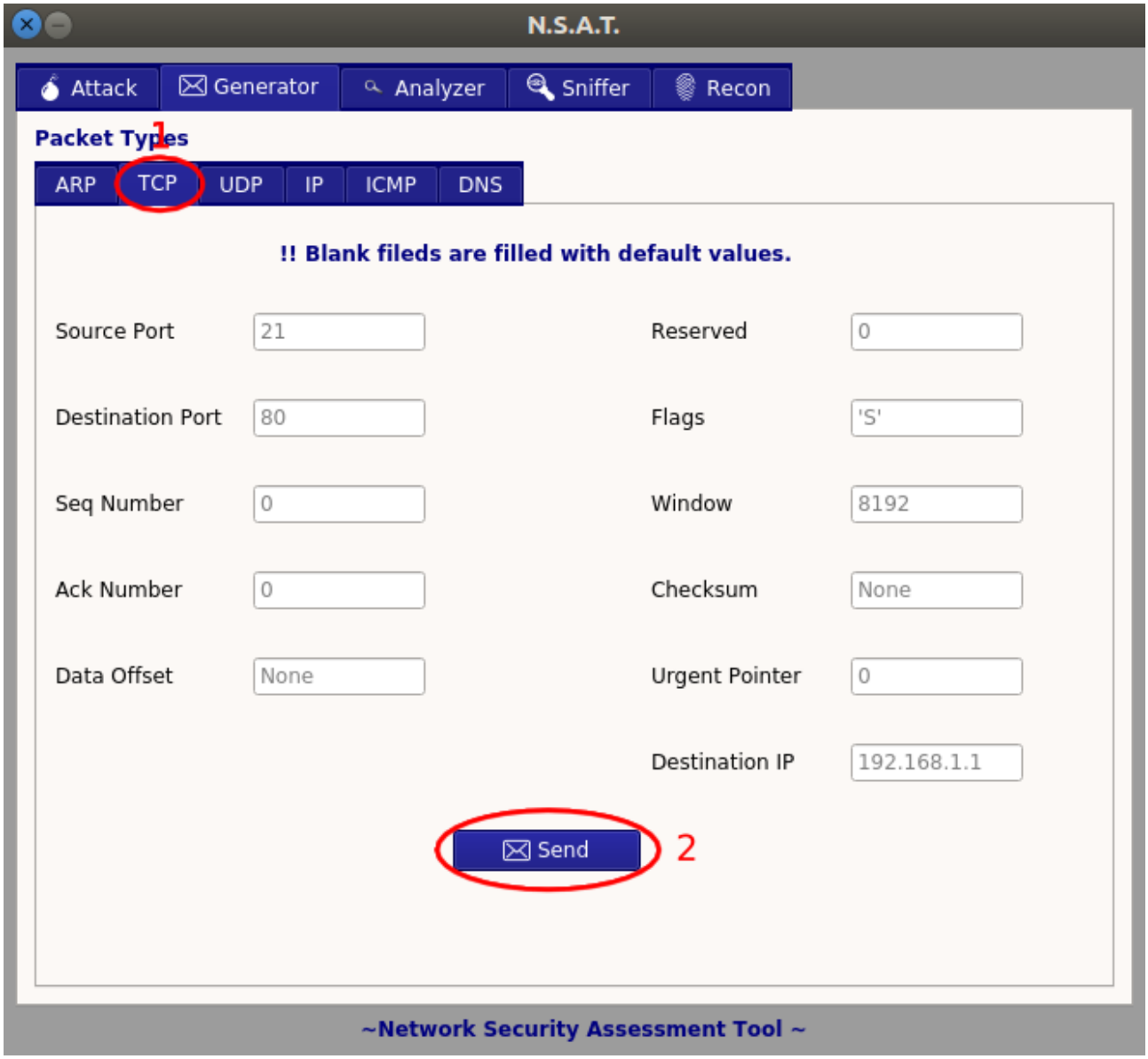
A.2.1. ARP Packet

- The user selects ‘ARP’ from the generation module.
- The user can fill the blanks to change packet values.
- Press ‘Send’ button to generate and send the packet.



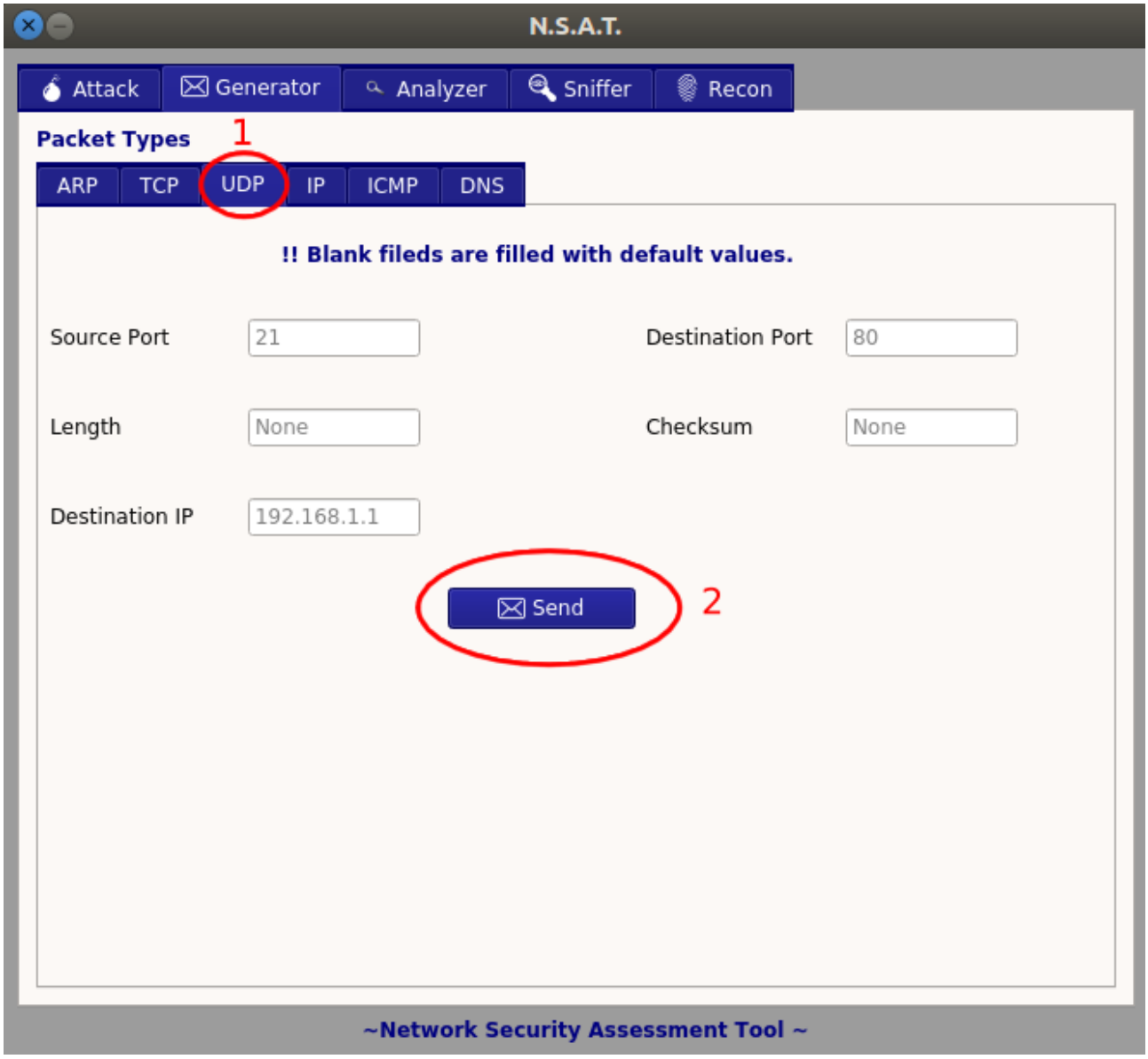
A.2.2. TCP Packet

- The user selects ‘TCP’ from the generation module.
- The user can fill the blanks to change packet values.
- Press ‘Send’ button to generate and send the packet.



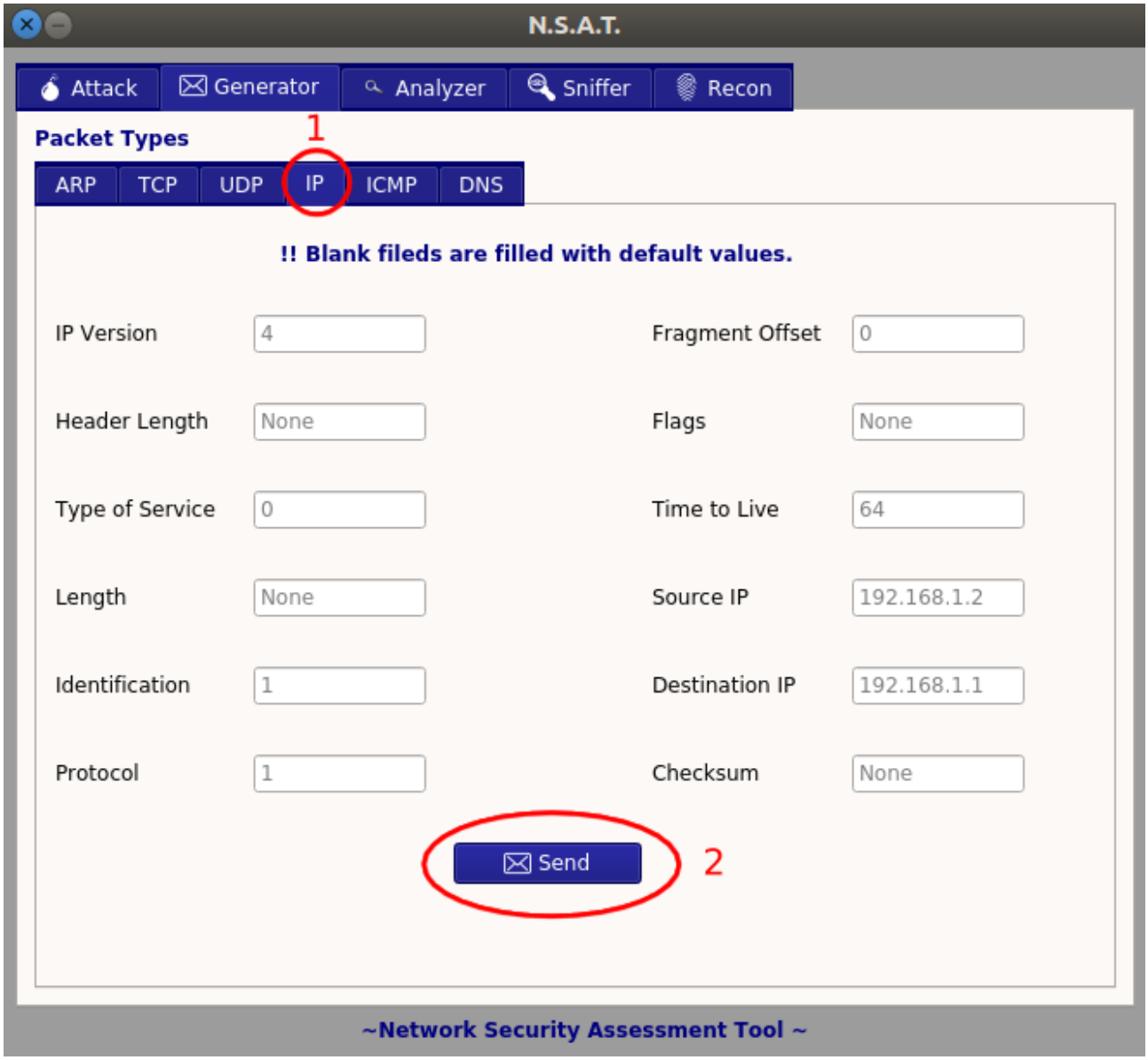
A.2.3. UDP Packet

- The user selects ‘UDP’ from the generation module.
- The user can fill the blanks to change packet values.
- Press ‘Send’ button to generate and send the packet.



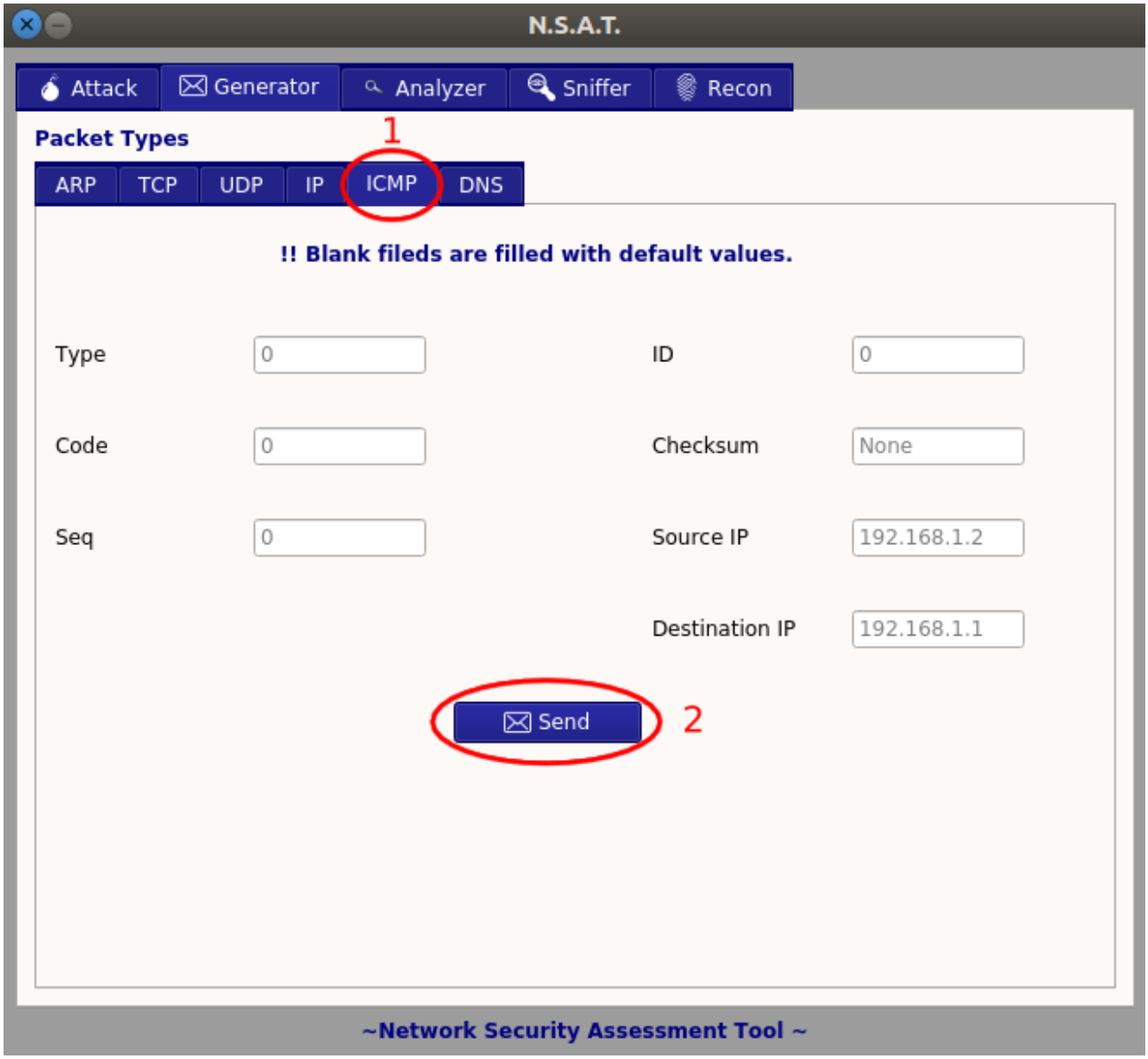
A.2.4. IP Packet

- The user selects ‘IP’ from the generation module.
- The user can fill the blanks to change packet values.
- Press ‘Send’ button to generate and send the packet.



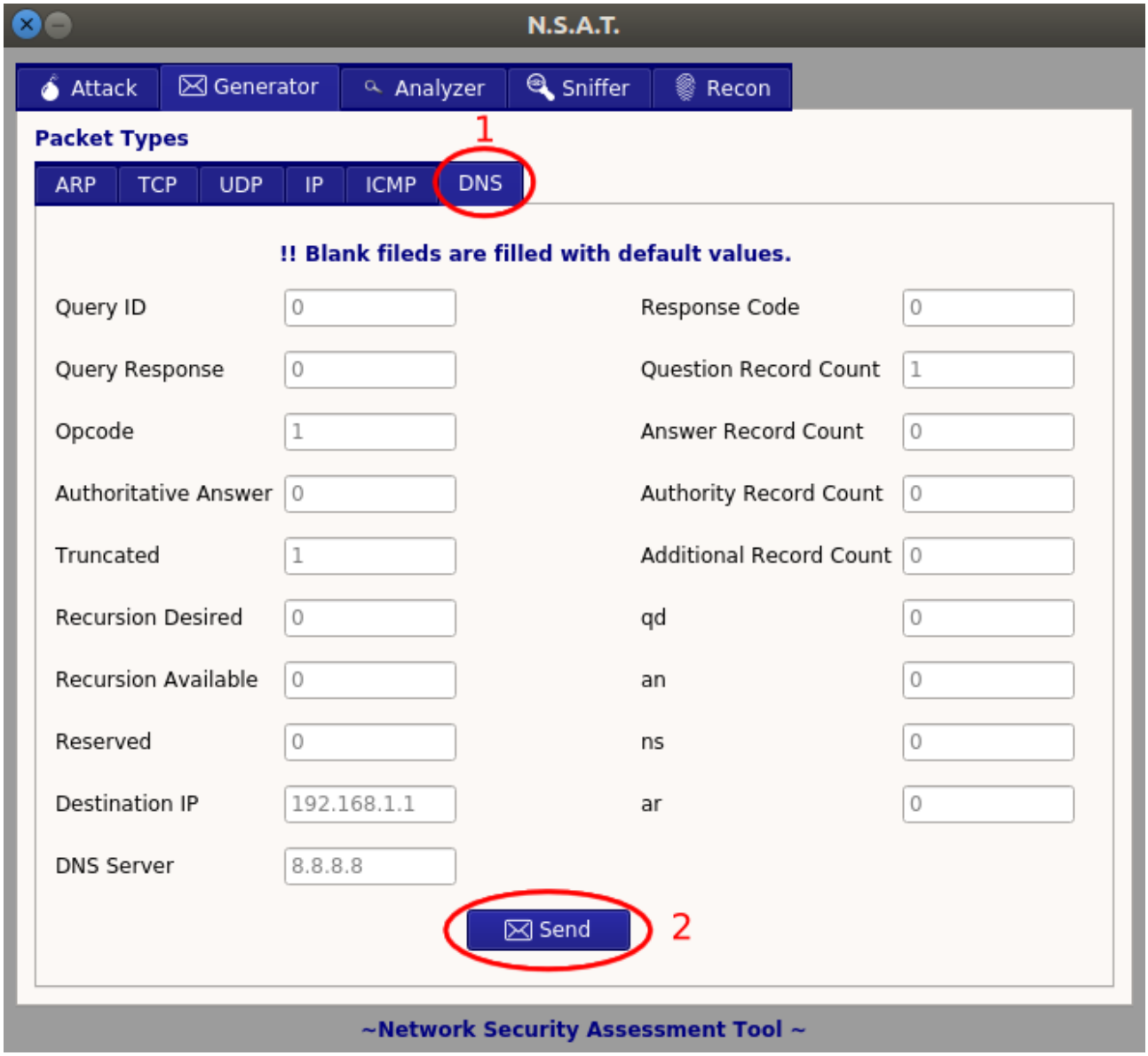
A.2.5. ICMP Packet

- The user selects ‘ICMP’ from the generation module.
- The user can fill the blanks to change packet values.
- Press ‘Send’ button to generate and send the packet.



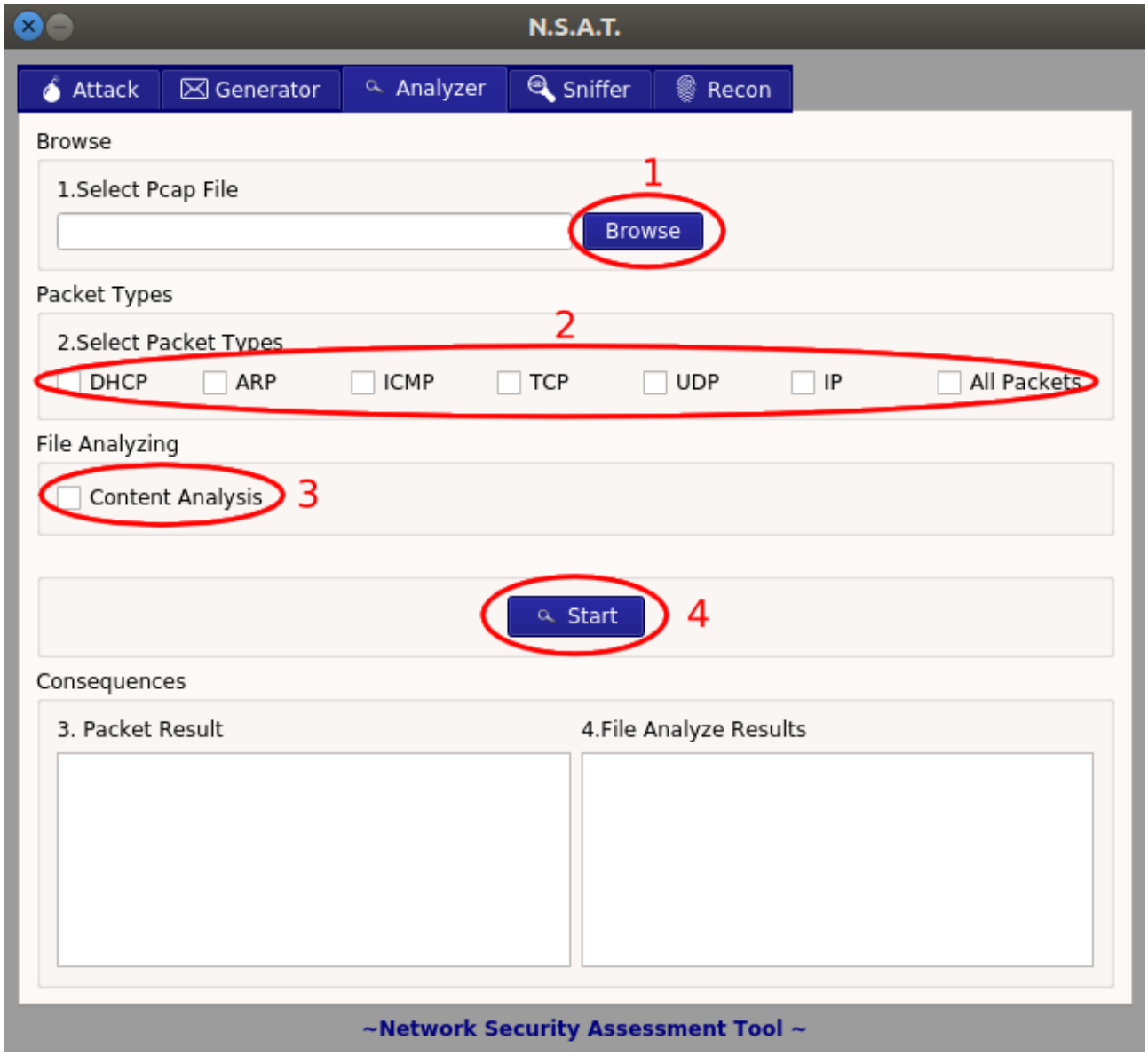
A.2.6. DNS Packet

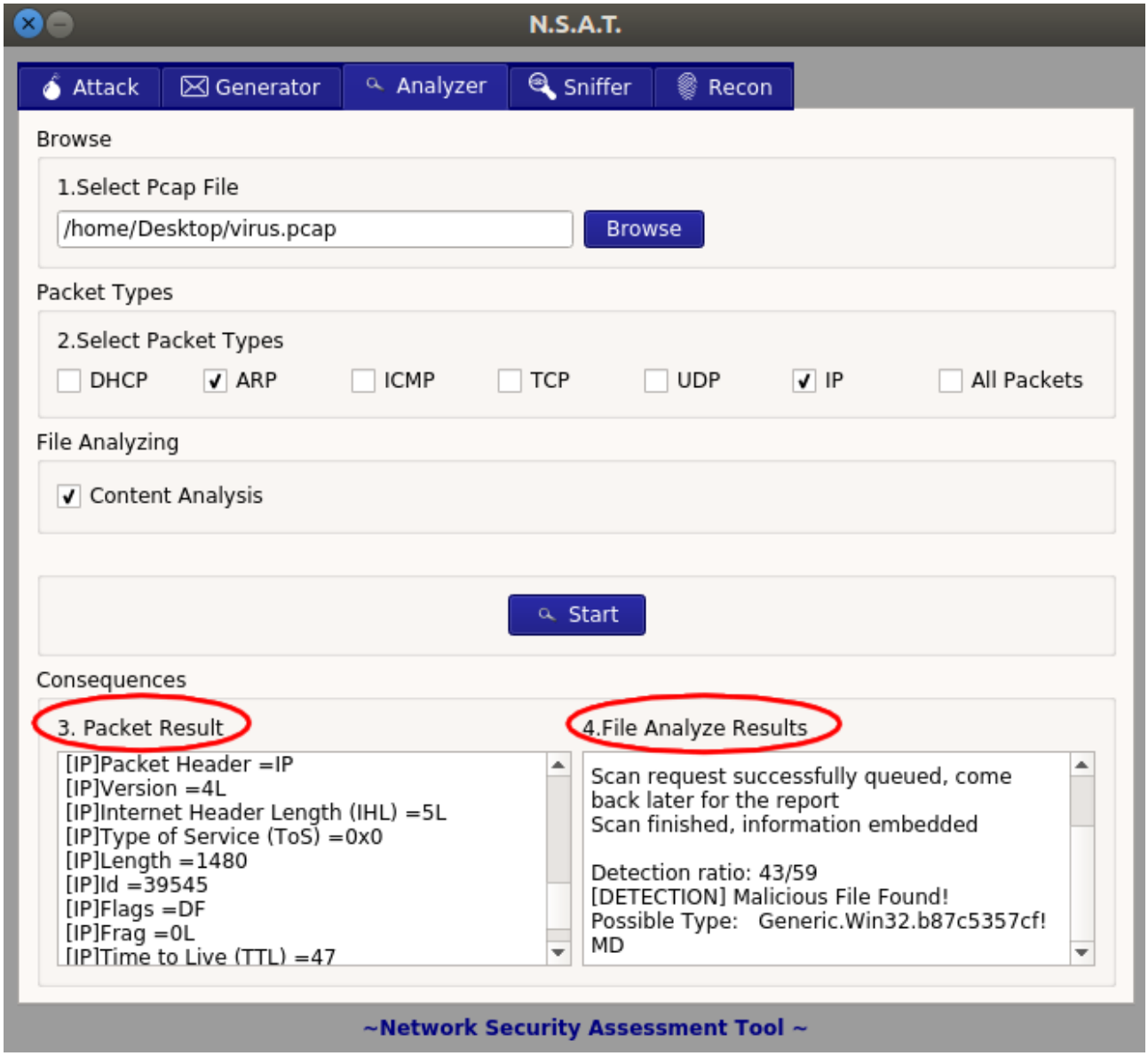
- The user selects ‘DNS’ from the generation module.
- The user can fill the blanks to change packet values.
- Press ‘Send’ button to generate and send the packet.



A.3. Packet Analyzer Module

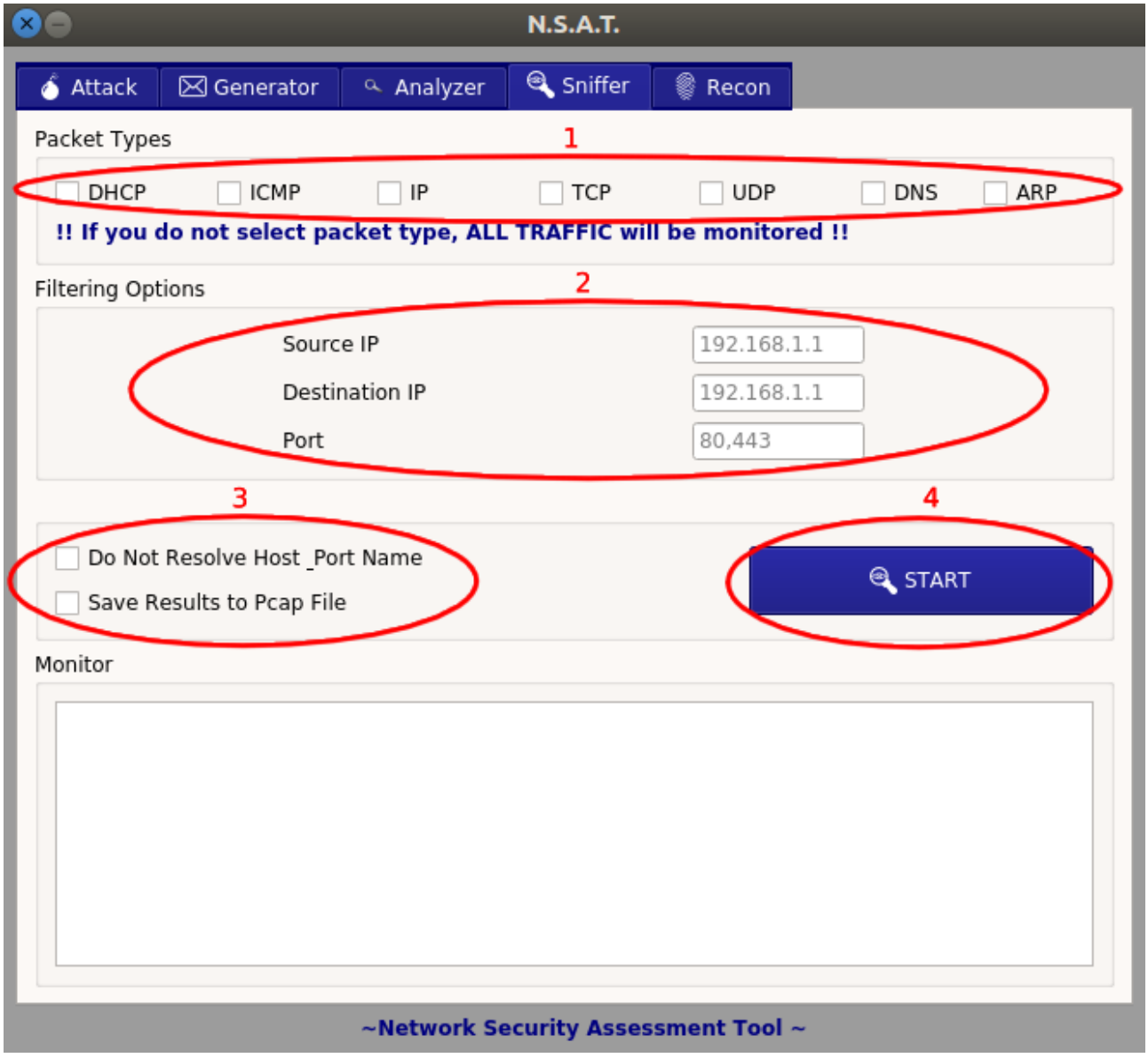
- The user selects a pcap file from the computer via ‘Browse’ button.
- Selects packet type(s) to analyze given pcap file.
- The user selects ‘Content Analysis’, if wants to detect malicious file in given pcap.
- Press ‘Start’ button to analyze packet results.

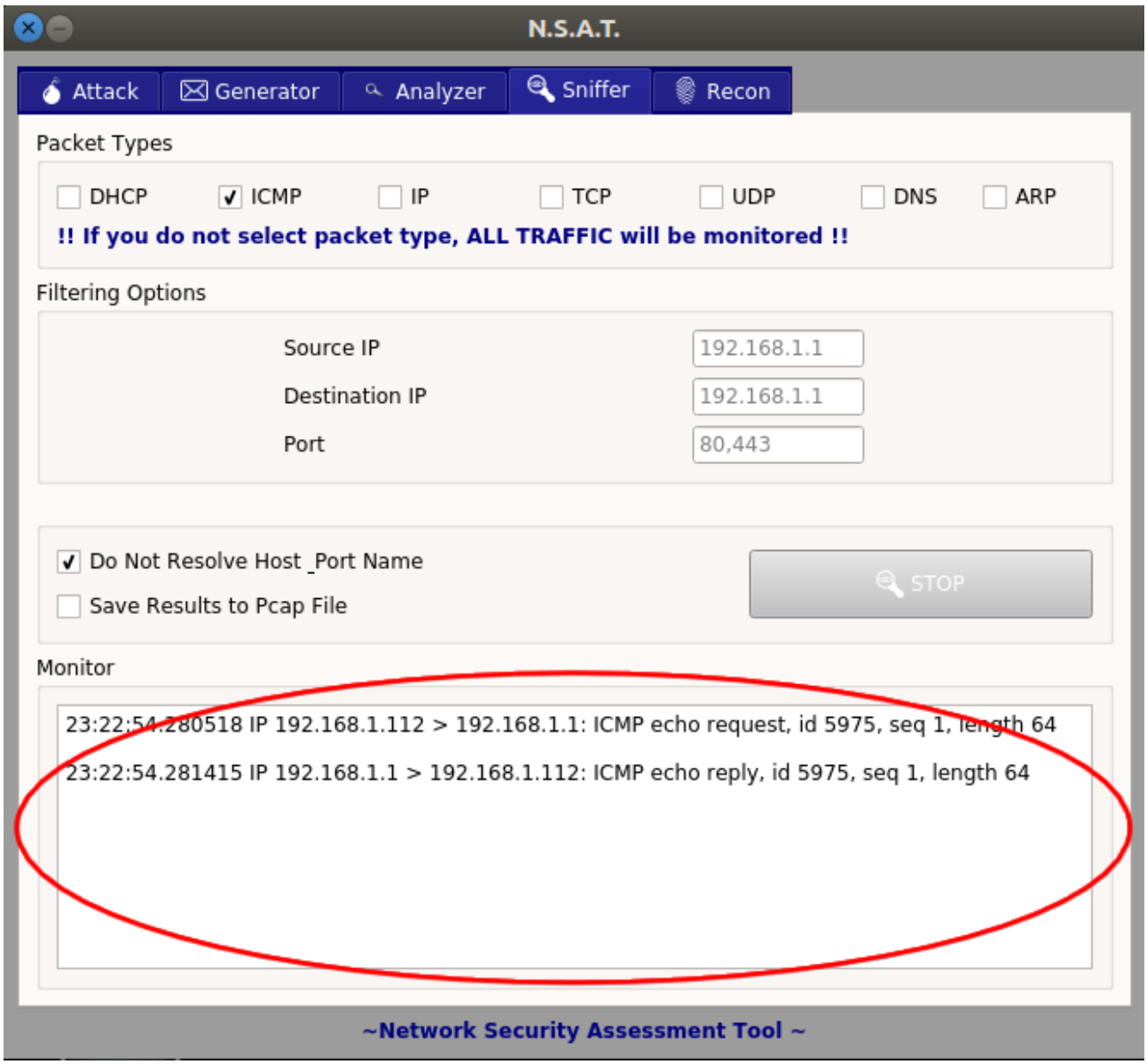



N.S.A.T.
Attack
Generator
Analyzer
Sniffer
Recon
Browse
1.Select Pcap File
/home/Desktop/virus.pcap
Browse
Packet Types
2.Select Packet Types
DHCP
ARP
ICMP
TCP
UDP
IP
All Packets
File Analyzing
Content Analysis
Start
Consequences
3. Packet Result
[IP]Packet Header =IP
[IP]Version =4L
[IP]Internet Header Length (IHL) =5L
[IP]Type of Service (ToS) =0x0
[IP]Length =1480
[IP]Id =39545
[IP]Flags =DF
[IP]Frag =0L
[IP]Time to Live (TTL) =47
4.File Analyze Results
Scan request successfully queued, come back later for the report
Scan finished, information embedded
Detection ratio: 43/59
[DETECTION] Malicious File Found!
Possible Type: Generic.Win32.b87c5357cf!MD
~Network Security Assessment Tool ~

A.4. Sniffer Module

- The user selects network protocols that wants to monitor.
- The user can fill the blank options to identify specific target.
- Select ‘Save Results’ option to save all traffic into pcap file.
- Press ‘ Start’ button to sniff network traffics.
- Press ‘Stop’ button to terminate monitoring.

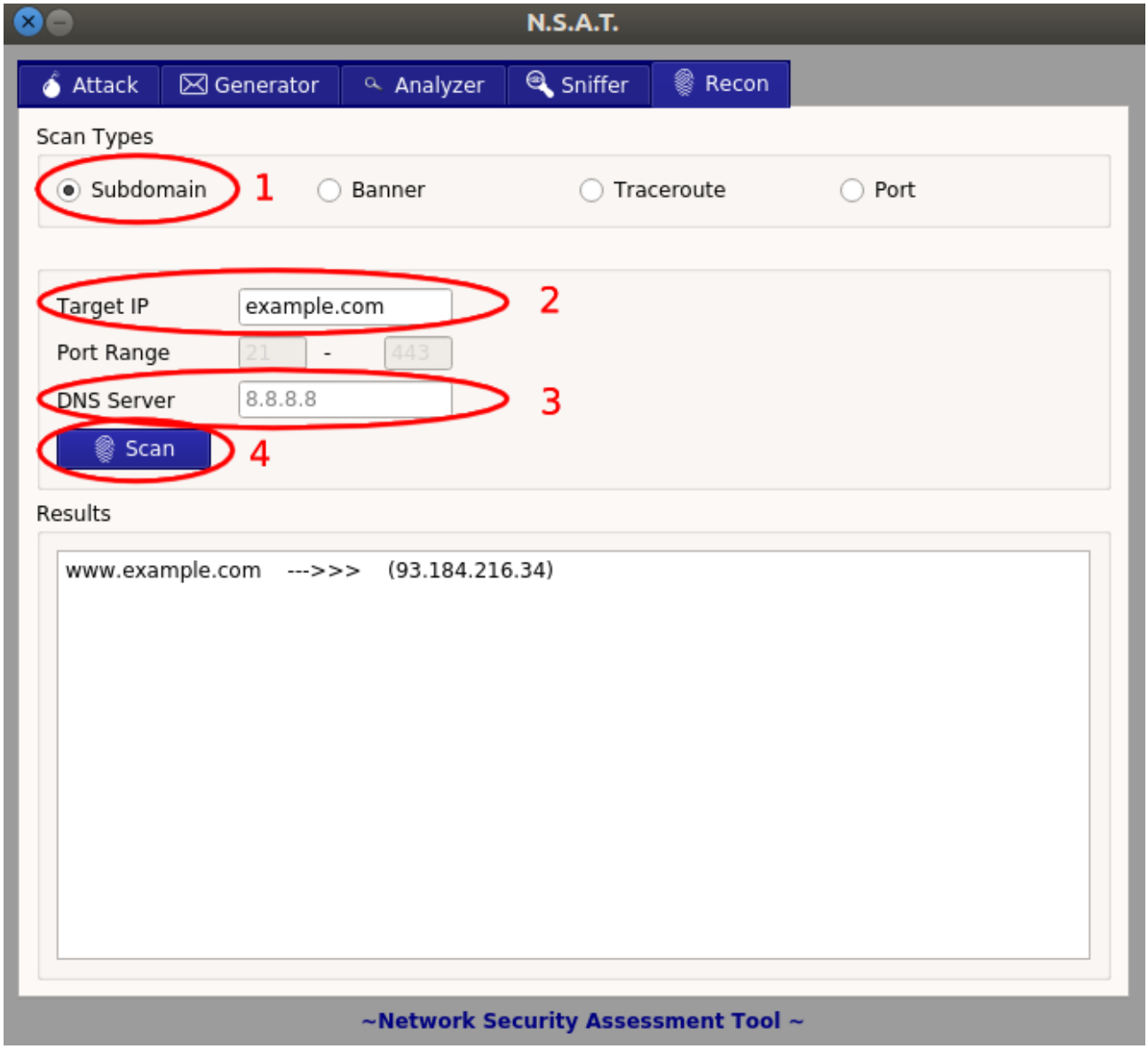



N.S.A.T.
Attack
Generator
Analyzer
Sniffer
Recon
Packet Types
DHCP
ICMP
IP
TCP
UDP
DNS
ARP
!! If you do not select packet type, ALL TRAFFIC will be monitored !!
Filtering Options
Source IP
192.168.1.1
Destination IP
192.168.1.1
Port
80,443
Do Not Resolve Host _Port Name
Save Results to Pcap File
STOP
Monitor
23:22:54.280518 IP 192.168.1.112 > 192.168.1.1: ICMP echo request, id 5975, seq 1, length 64
23:22:54.281415 IP 192.168.1.1 > 192.168.1.112: ICMP echo reply, id 5975, seq 1, length 64
~Network Security Assessment Tool ~

A.5. Recon Module

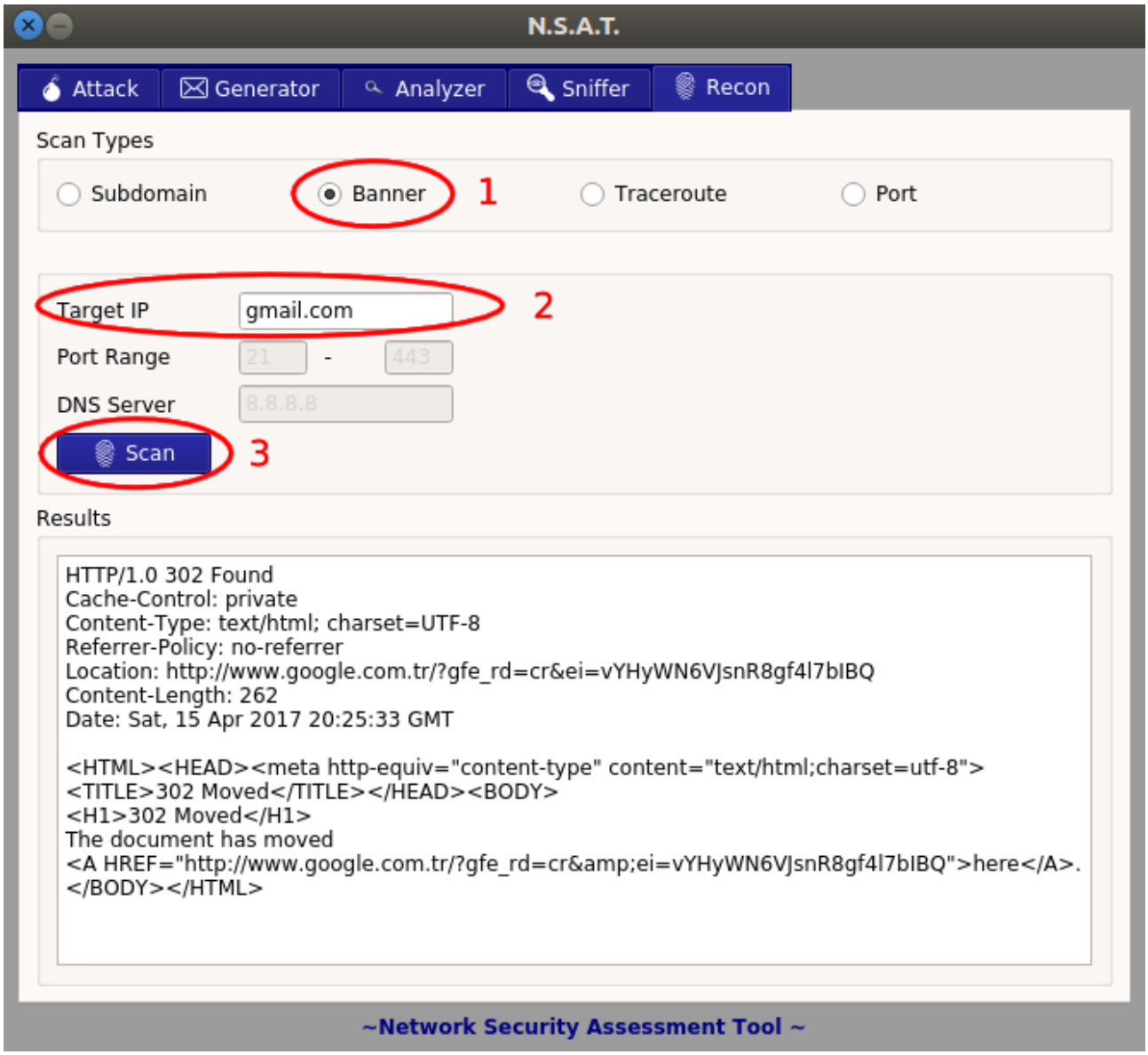
A.5.1. Subdomain Finder

- The user selects ‘Subdomain’ from the recon module.
- Enters target IP or domain name.
- The user can enter different dns servers.
- Press ‘Scan’ button to find all subdomains.



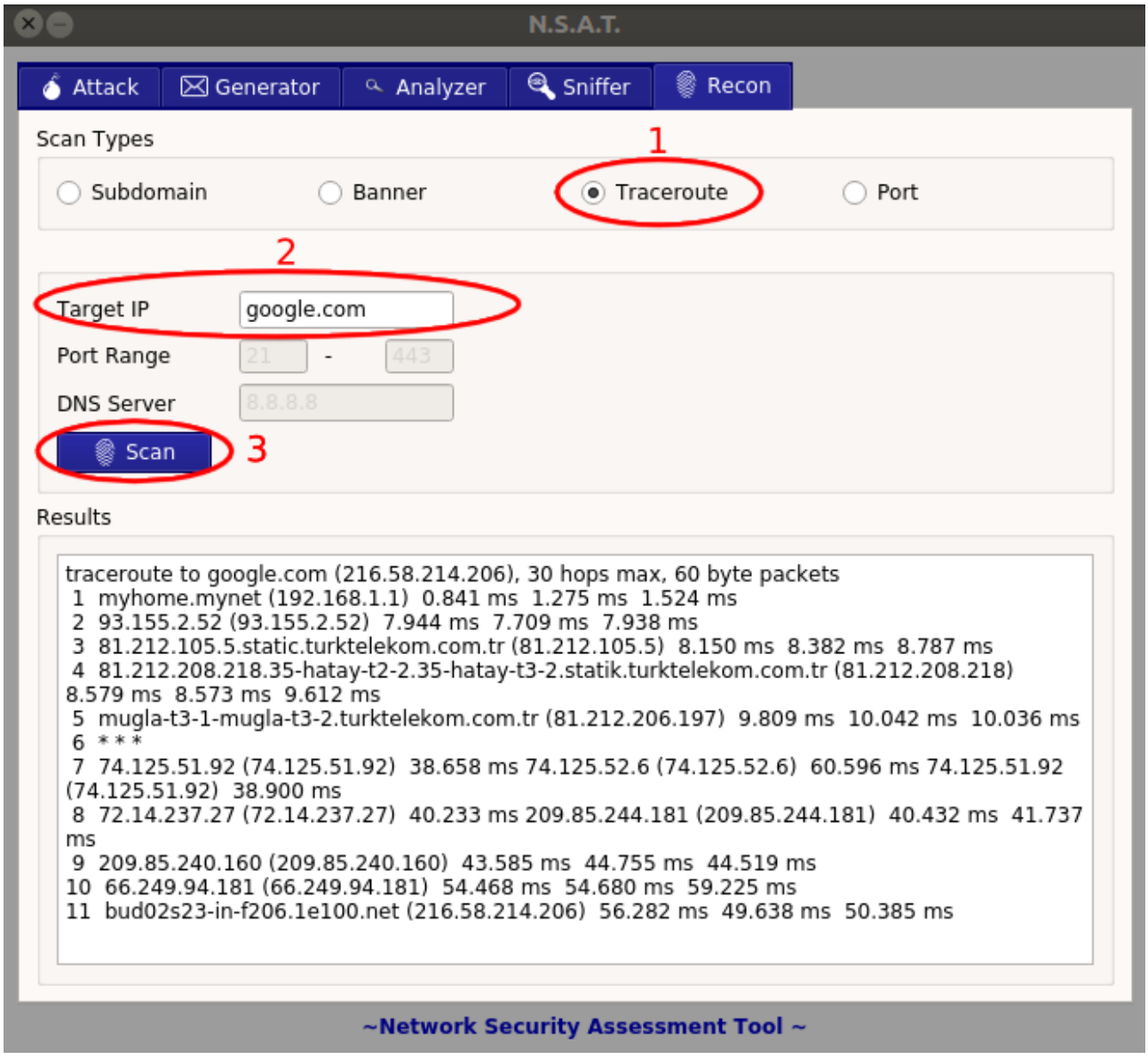
A.5.2. Banner Detection

- The user selects ‘Banner’ from the recon module.
- Enters target IP or domain name.
- Press ‘Scan’ button to detect server banner.



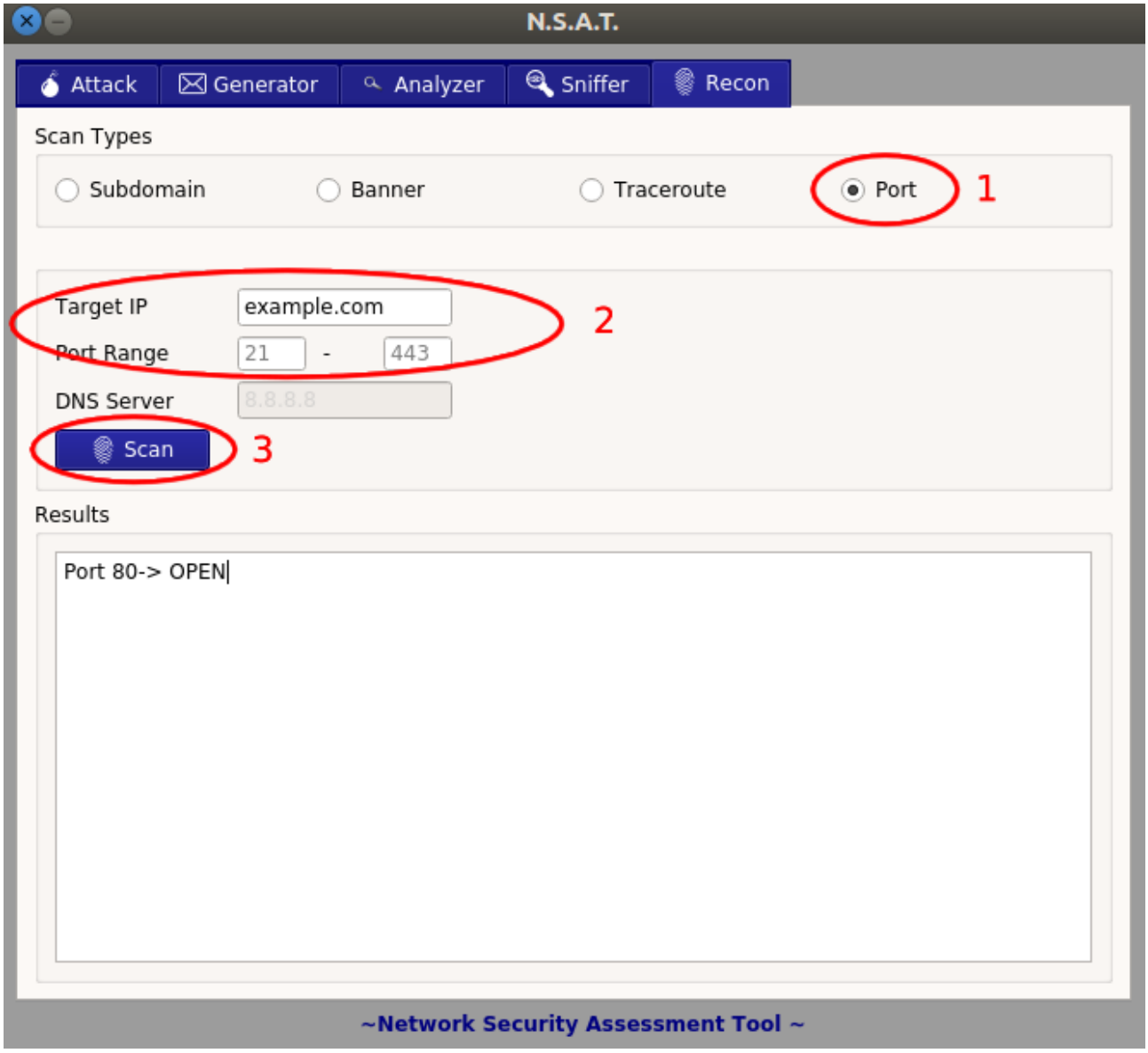
A.5.3. Traceroute

- The user selects ‘Traceroute’ from the recon module.
- Enters target IP or domain name.
- Press ‘Scan’ button to find routing paths.



A.5.4. Port Scan

- The user selects ‘Traceroute’ from the recon module.
- Enters target IP or domain name.
- The user can give port range.
- Press ‘Scan’ button to find open ports.



B. TERMINAL

B.1. Main Commands

B.1.1. Help

 # Engine.py --help

```
root@NSAT:~# Engine.py --help
usage: Engine.py [-h] [--file FILE] [--virus-scan]
                 [--packet-type {arp,ip,icmp,tcp,udp,dns}] [--auto] [--manual]
                 [--attack-type {ddos,arpspoof,icmpflood,tcpflood,synflood,fuzzer,sshoverload}]
                 [--recon-type {subdomain,portscan,banner,traceroute}]
                 [--sniff-type {arp,ip,tcp,udp,icmp,dhcp} [{arp,ip,tcp,udp,icmp,dhcp} ...]]
                 [--resolve] [--save] [--multi] --module
                 {attack,generate,pcap,recon,sniffer} [--version]

Network Security Assessment Tool [EmreOvunc-AyseSimgeOzger-UmutBasaran]

optional arguments:
  -h, --help            show this help message and exit
  --module {attack,generate,pcap,recon,sniffer}
                        select a module to run
  --version             show program's version number and exit

[Pcap Analysis Module]:
  --file FILE           give a pcap file to read
  --virus-scan          scan files in pcap to detect threads

[Packet Generation Module]:
  --packet-type {arp,ip,icmp,tcp,udp,dns}
                        select a packet type to generate
  --auto                generate auto packets
  --manual              generate manual packets

[Attack Module]:
  --attack-type {ddos,arpspoof,icmpflood,tcpflood,synflood,fuzzer,sshoverload}
                        select a type of attack

[Recon Module]:
  --recon-type {subdomain,portscan,banner,traceroute}
                        select a type of recon

[Sniffer Module]:
  --sniff-type {arp,ip,tcp,udp,icmp,dhcp} [{arp,ip,tcp,udp,icmp,dhcp} ...]
                        select a type of sniffing filters
  --resolve             do not resolve protocol and port numbers
  --save                to save all traffics in a pcap file
  --multi               to select more sniffing filters

Ex: Engine.py --module attack --attack-type icmpflood
```

B.1.2. Version

 # Engine.py --version

```
root@NSAT:~# Engine.py --version
NSAT-Project v1.4.2
```

B.2. Attack Module

B.2.1. Fuzzer

 # Engine.py --module attack --attack-type fuzzer

```
root@NSAT:~# Engine.py --module attack --attack-type fuzzer
Enter the URL: 192.168.1.1
```

```
root@NSAT:~# cat Fuzz-2017-04-16/192.168.1.1-WFuzz.txt
********************************************************
* Wfuzz 2.1.3 - The Web Bruteforcer                    *
********************************************************

Target: http://192.168.1.1/FUZZ
Total requests: 950

=====================================================================
ID      Response   Lines      Word         Chars          Request
=====================================================================

00360:  C=200    412 L       601 W       12871 Ch        "help"
00388:  C=200    278 L       527 W       10131 Ch        "index"
00571:  C=403     56 L        90 W        1713 Ch        "phone"
00704:  C=403     56 L        90 W        1713 Ch        "setup"
00750:  C=200    270 L       474 W        8840 Ch        "status"
00784:  C=403     56 L        90 W        1713 Ch        "test"
00794:  C=403     56 L        90 W        1713 Ch        "tools"
00931:  C=403     56 L        90 W        1713 Ch        "account"

Total time: 25.67532
Processed Requests: 950
Filtered Requests: 942
Requests/sec.: 37.00050
```

B.2.2. ARP Spoof

 # Engine.py --module attack --attack-type arpspoof

```
root@NSAT:~# Engine.py --module attack --attack-type arpspoof
Destination IP: 192.168.1.1
How many packets do you sent [Press enter for 100]: 3
.
Sent 1 packets.
.
Sent 1 packets.
.
Sent 1 packets.
.
Sent 1 packets.
.
Sent 1 packets.
.
Sent 1 packets.
```

B.2.3. SSH Overload

 # Engine.py --module attack --attack-type sshoverload

```
root@NSAT:~# Engine.py --module attack --attack-type sshoverload
Destination IP: 192.168.1.1
using eth0, addr: 192.168.1.235, MTU: 1500
HPING 192.168.1.1 (eth0 192.168.1.1): S set, 40 headers + 120 data bytes
hping in flood mode, no replies will be shown
```

B.2.4. DoS

 # Engine.py --module attack --attack-type ddos

```
root@NSAT:~# Engine.py --module attack --attack-type ddos
Destination IP: 10.0.0.1
using eth0, addr: 192.168.1.235, MTU: 1500
HPING 10.0.0.1 (eth0 10.0.0.1): S set, 40 headers + 120 data bytes
hping in flood mode, no replies will be shown
^C
--- 10.0.0.1 hping statistic ---
90889 packets transmitted, 0 packets received, 100% packet loss
round-trip min/avg/max = 0.0/0.0/0.0 ms
```

B.2.5. TCP Flood

 # Engine.py --module attack --attack-type tcpflood

```
root@NSAT:~# Engine.py --module attack --attack-type tcpflood
Destination IP: 10.0.0.1

Starting Nping 0.7.40 ( https://nmap.org/nping ) at 2017-04-16 16:42 +03
^CMax rtt: N/A | Min rtt: N/A | Avg rtt: N/A
TCP connection attempts: 13554 | Successful connections: 0 | Failed: 13554 (100.00%)
Nping done: 1 IP address pinged in 5.50 seconds
```

B.2.6. ICMP Flood

 # Engine.py --module attack --attack-type icmpflood

```
root@NSAT:~# Engine.py --module attack --attack-type icmpflood
Destination IP: 10.0.0.1
How many packets do you sent [Press enter for 100]: 5
.
Sent 1 packets.
.
Sent 1 packets.
.
Sent 1 packets.
.
Sent 1 packets.
.
Sent 1 packets.
```

B.2.7. SYN Flood

 # Engine.py --module attack --attack-type synflood

```
root@NSAT:~# Engine.py --module attack --attack-type synflood
Destination IP: 10.0.0.1
Destination Port: 443
How many packets do you sent [Press enter for 100]: 5
.
Sent 1 packets.
.
Sent 1 packets.
.
Sent 1 packets.
.
Sent 1 packets.
.
Sent 1 packets.
```

B.3. Generation Module

B.3.1. ARP Packet

B.3.1.1. Auto ARP Packet

 # Engine.py --module generate --packet-type arp --auto

```
root@NSAT:~# Engine.py --module generate --packet-type arp --auto
Destination IP: 192.168.1.1
Do you want to send your packet? (Y/n): Y
.
Sent 1 packets.
```

B.3.1.2. Manual ARP Packet

 # Engine.py --module generate --packet-type arp --manual

```
root@NSAT:~# Engine.py --module generate --packet-type arp --manual
[ARP] Hardware Type (hwtype) [Press enter for 0x1]: 0x1
[ARP] Protocol Type (ptype) [Press enter for 0x800]: 0x800
[ARP] Protocol Length (plen) [Press enter for 4]:
[ARP] Operation (op) [Press enter for who-has]:
Source IP: 10.0.0.1
Destination IP: 10.0.0.2
Source Mac Address: aa:bb:cc:11:22:33
Do you want to send your packet? (Y/n): Y
.
Sent 1 packets.
```

B.3.2. TCP Packet

B.3.2.1. Auto TCP Packet

 # Engine.py --module generate --packet-type tcp --auto

```
root@NSAT:~# Engine.py --module generate --packet-type tcp --auto
Destination IP: 10.0.0.1
Source Port: 8080
Destination Port: 4444
.
Sent 1 packets.
```

B.3.2.2. Manual TCP Packet

 # Engine.py --module generate --packet-type tcp --manual

```
root@NSAT:~# Engine.py --module generate --packet-type tcp --manual
[TCP] Source Port [Press enter for 21]:
[TCP] Destination Port [Press enter for 80]: 8080
[TCP] Seq Number [Press enter for 0]: 1
[TCP] Ack Number [Press enter for 0]: 132
[TCP] Dataoffset [Press enter for None]:
[TCP] Reserved [Press enter for 0]:
[TCP] Flags [Press enter for 'S']: S
[TCP] Window [Press enter for 8192]:
[TCP] Checksum [Press enter for None]:
[TCP] Urgent Pointer [Press enter for 0]:
Destination IP: 10.0.0.1
.
Sent 1 packets.
```

B.3.3. UDP Packet

B.3.3.1. Auto UDP Packet

 # Engine.py --module generate --packet-type udp --auto

```
root@NSAT:~# Engine.py --module generate --packet-type udp --auto
Destination IP: 10.0.0.1
Destination Port: 53
Source Port: 50
.
Sent 1 packets.
```

B.3.3.2. Manual UDP Packet

 # Engine.py --module generate --packet-type udp --manual

```
root@NSAT:~# Engine.py --module generate --packet-type udp --manual
[UDP] Source Port [Press enter for 53]:
[UDP] Destination Port [Press enter for 53]: 50
[UDP] Length [Press enter for None]:
[UDP] Checksum [Press enter for None]:
Destination IP: 10.0.0.1
.
Sent 1 packets.
```

B.3.4. IP Packet

B.3.4.1. Auto IP Packet

 # Engine.py --module generate --packet-type ip --auto

```
root@NSAT:~# Engine.py --module generate --packet-type ip --auto
Source IP: 10.0.0.1
Destination IP: 10.0.0.2
Do you want to send your packet? (Y/n): Y
.
Sent 1 packets.
```

B.3.4.2. Manual IP Packet

 # Engine.py --module generate --packet-type ip --manual

```
root@NSAT:~# Engine.py --module generate --packet-type ip --manual
[IP] Version [Press enter for 4]:4
[IP] IP Header Length (ihl) [Press enter for None]:
[IP] Type of Service (tos) [Press enter for 0x0]:
[IP] Length (len) [Press enter for None]:
[IP] Identification (id) [Press enter for 1]: 1
[IP] Fragmentation Offset (frag) [Press enter for 0]:
[IP] Flag number [Press enter for 0 - Reserved]: 1
[IP] Time to Live (ttl) [Press enter for 64]: 128
[IP] Protocol (proto) [Press enter for hopopt]:
[IP] Checksum [Press enter for None]:
Source IP: 10.0.0.1
Destination IP: 10.0.0.2
Do you want to send your packet? (Y/n): y
.
Sent 1 packets.
```

B.3.5. ICMP Packet

B.3.5.1. Auto ICMP Packet

 # Engine.py --module generate --packet-type icmp --auto

```
root@NSAT:~# Engine.py --module generate --packet-type icmp --auto
Destination IP: 10.0.0.1
.
Sent 1 packets.
```

B.3.5.2. Manual ICMP Packet

 # Engine.py --module generate --packet-type icmp --manual

```
root@NSAT:~# Engine.py --module generate --packet-type icmp --manual
[ICMP] Type [Press enter for echo-request]:1
[ICMP] Code [Press enter for 0]:
[ICMP] Checksum [Press enter for None]:
[ICMP] ID [Press enter for 0x0]:0x1
[ICMP] Seq [Press enter for 0x0]:0x2
Source IP: 10.0.0.1
Destination IP: 10.0.0.2
Do you want to send your packet? (Y/n): y
.
Sent 1 packets.
```

B.3.6. DNS Packet

B.3.6.1. Auto DNS Packet

 # Engine.py --module generate --packet-type dns --auto

```
root@NSAT:~# Engine.py --module generate --packet-type dns --auto
Target Domain: google.com
DNS Ans "172.217.17.174"
```

B.3.6.2. Manual DNS Packet

 # Engine.py --module generate --packet-type dns --manual

```
root@NSAT:~# Engine.py --module generate --packet-type dns --manual
[DNS] Query ID [Press enter for 0]:
[DNS] Query Response [Press enter for 0]:
[DNS] Opcode [Press enter for 'QUERY']:
[DNS] Authoritative Answer [Press enter for 0]:
[DNS] Truncated [Press enter for 1]:
[DNS] Recursion Desired [Press enter for 0]:
[DNS] Recursion Available [Press enter for 0]:
[DNS] Reserved [Press enter for 0]:
[DNS] Response Code [Press enter for 'ok']:
[DNS] Question Record Count [Press enter for 1]:
[DNS] Answer Record Count [Press enter for 0]:
[DNS] Authority Record Count [Press enter for 0]:
[DNS] Additional Record Count [Press enter for 0]:
[DNS] qd [Press enter for 0]:
[DNS] an [Press enter for 0]:
[DNS] ns [Press enter for 0]:
[DNS] ar [Press enter for 0]:
Target Domain: google.com
DNS Server: 8.8.8.8
```

B.4. Packet Analyzer Module

B.4.1. Pcap File

 # Engine.py --module pcap --file /your/pcap/file.pcap

```
root@NSAT:~# Engine.py --module pcap --file traffic.pcap
################## Packet[1] ##################

[Ether]Destination MAC = 00:  Mac Address  :6c
[Ether]Source MAC = 44:  Mac Address  :d1
[Ether]Type = 0x800

[IP]Packet Header = IP
[IP]Version = 4L
[IP]Internet Header Length (IHL) = 5L
[IP]Type of Service (ToS) = 0x0
[IP]Length = 59
[IP]Id = 17132
[IP]Flags = DF
[IP]Frag = 0L
[IP]Time to Live (TTL) = 64
[IP]Protocol = udp
[IP]Checksum = 0x7387
[IP]Source IP = 192.168.1.237
[IP]Destionation IP = 192.168.1.1

[UDP]Transport Protocol = UDP
[UDP]Source Port = 50878
[UDP]Destination Port = domain
[UDP]Length = 39
[UDP]Checksum = 0x75ab
```

B.4.2. Content Analysis

 # Engine.py --module pcap --file /your/pcap/file.pcap --virus-scan

```
root@NSAT:~# Engine.py --module pcap --file traffic.pcap --virus-scan
Running as user "root" and group "root". This could be dangerous.
tshark: Lua: Error during loading:
 [string "/usr/share/wireshark/init.lua"]:44: dofile has been disabled due to running Wiresha
rk as superuser. See https://wiki.wireshark.org/CaptureSetup/CapturePrivileges for help in ru
nning Wireshark as an unprivileged user.
Scan request successfully queued, come back later for the report
Scan finished, information embedded

Detection ratio: 41/58
[DETECTION] Malicious File Found!
Possible Type:   Generic.Win32.b87c5357cf!MD
```

B.5. Sniffer Module

B.5.1. ARP Sniffing

 # Engine.py --module sniffer --sniff-type arp

```
root@NSAT:~# Engine.py --module sniffer --sniff-type arp
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on eth0, link-type EN10MB (Ethernet), capture size 262144 bytes
18:37:15.073559 ARP, Request who-has myhome.mynet tell NSAT.mynet, length 28
18:37:15.073983 ARP, Reply myhome.mynet is-at 00:02:61:83:a5:6c (oui Unknown), length 46
18:37:20.066632 ARP, Request who-has NSAT.mynet tell myhome.mynet, length 46
18:37:20.066657 ARP, Reply NSAT.mynet is-at 08:00:27:5f:cf:aa (oui Unknown), length 28
```

B.5.2. IP Sniffing

 # Engine.py --module sniffer --sniff-type ip

```
root@NSAT:~# Engine.py --module sniffer --sniff-type ip
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on eth0, link-type EN10MB (Ethernet), capture size 262144 bytes
18:38:31.270981 IP NSAT.mynet.ssh > Monster.mynet.33670: Flags [P.], seq 3620382290:3620382398
, ack 220239113, win 272, options [nop,nop,TS val 330793 ecr 426091], length 108
18:38:31.271091 IP Monster.mynet.33670 > NSAT.mynet.ssh: Flags [.], ack 108, win 1444, options
 [nop,nop,TS val 426261 ecr 330793], length 0
18:38:31.271187 IP NSAT.mynet.ssh > Monster.mynet.33670: Flags [P.], seq 108:144, ack 1, win 2
72, options [nop,nop,TS val 330793 ecr 426261], length 36
18:38:31.271233 IP Monster.mynet.33670 > NSAT.mynet.ssh: Flags [.], ack 144, win 1444, options
 [nop,nop,TS val 426261 ecr 330793], length 0
18:38:31.271301 IP NSAT.mynet.ssh > Monster.mynet.33670: Flags [P.], seq 144:252, ack 1, win 2
72, options [nop,nop,TS val 330793 ecr 426261], length 108
18:38:31.271346 IP Monster.mynet.33670 > NSAT.mynet.ssh: Flags [.], ack 252, win 1444, options
 [nop,nop,TS val 426261 ecr 330793], length 0
```

B.5.3. TCP Sniffing

 # Engine.py --module sniffer --sniff-type tcp

```
root@NSAT:~# Engine.py --module sniffer --sniff-type tcp
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on eth0, link-type EN10MB (Ethernet), capture size 262144 bytes
18:39:05.208571 IP NSAT.mynet.ssh > Monster.mynet.33670: Flags [P.], seq 3621316794:3621316902
, ack 220241329, win 272, options [nop,nop,TS val 339277 ecr 434573], length 108
18:39:05.208775 IP Monster.mynet.33670 > NSAT.mynet.ssh: Flags [.], ack 108, win 1444, options
 [nop,nop,TS val 434745 ecr 339277], length 0
18:39:05.208856 IP NSAT.mynet.ssh > Monster.mynet.33670: Flags [P.], seq 108:144, ack 1, win 2
72, options [nop,nop,TS val 339278 ecr 434745], length 36
18:39:05.208916 IP Monster.mynet.33670 > NSAT.mynet.ssh: Flags [.], ack 144, win 1444, options
 [nop,nop,TS val 434745 ecr 339278], length 0
18:39:05.208996 IP NSAT.mynet.ssh > Monster.mynet.33670: Flags [P.], seq 144:252, ack 1, win 2
72, options [nop,nop,TS val 339278 ecr 434745], length 108
18:39:05.209050 IP Monster.mynet.33670 > NSAT.mynet.ssh: Flags [.], ack 252, win 1444, options
 [nop,nop,TS val 434745 ecr 339278], length 0
```

B.5.4. UDP Sniffing

 # Engine.py --module sniffer --sniff-type udp

```
root@NSAT:~# Engine.py --module sniffer --sniff-type udp
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on eth0, link-type EN10MB (Ethernet), capture size 262144 bytes
18:40:00.928844 IP NSAT.mynet.bootpc > myhome.mynet.bootps: BOOTP/DHCP, Request from 08:00:27:
5f:cf:aa (oui Unknown), length 300
18:40:00.929757 IP myhome.mynet.bootps > NSAT.mynet.bootpc: BOOTP/DHCP, Reply, length 548
18:40:00.930316 IP NSAT.mynet.48192 > myhome.mynet.domain: 60837+ PTR? 1.1.168.192.in-addr.arp
a. (42)
18:40:00.931648 IP myhome.mynet.domain > NSAT.mynet.48192: 60837* 1/0/0 PTR myhome.mynet. (68)
18:40:00.931738 IP NSAT.mynet.57025 > myhome.mynet.domain: 31416+ PTR? 235.1.168.192.in-addr.a
rpa. (44)
18:40:00.932856 IP myhome.mynet.domain > NSAT.mynet.57025: 31416* 1/0/0 PTR NSAT.mynet. (68)
```

B.5.5. ICMP Sniffing

 # Engine.py --module sniffer --sniff-type icmp

```
root@NSAT:~# Engine.py --module sniffer --sniff-type icmp
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on eth0, link-type EN10MB (Ethernet), capture size 262144 bytes
18:42:10.776108 IP NSAT.mynet > myhome.mynet: ICMP echo request, id 2757, seq 1, length 64
18:42:10.776942 IP myhome.mynet > NSAT.mynet: ICMP echo reply, id 2757, seq 1, length 64
18:42:11.779178 IP NSAT.mynet > myhome.mynet: ICMP echo request, id 2757, seq 2, length 64
18:42:11.780044 IP myhome.mynet > NSAT.mynet: ICMP echo reply, id 2757, seq 2, length 64
```

B.5.6. DHCP Sniffing

 # Engine.py --module sniffer --sniff-type dhcp

```
root@NSAT:~# Engine.py --module sniffer --sniff-type dhcp
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on eth0, link-type EN10MB (Ethernet), capture size 262144 bytes
18:42:45.202217 IP 0.0.0.0.bootpc > 255.255.255.255.bootps: BOOTP/DHCP, Request from 88:e8:7f:
c1:b7:15 (oui Unknown), length 300
18:42:48.228906 IP NSAT.mynet.bootpc > myhome.mynet.bootps: BOOTP/DHCP, Request from 08:00:27:
5f:cf:aa (oui Unknown), length 300
18:42:48.230117 IP myhome.mynet.bootps > NSAT.mynet.bootpc: BOOTP/DHCP, Reply, length 548
```

B.5.7. Multi Packets Sniffing

 # Engine.py --module sniffer --sniff-type arp icmp --multi

```
root@NSAT:~# Engine.py --module sniffer --sniff-type arp icmp --multi
arp or icmp
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on eth0, link-type EN10MB (Ethernet), capture size 262144 bytes
20:02:45.075146 IP Monster.mynet > NSAT.mynet: ICMP echo request, id 2954, seq 1, length 64
20:02:45.075176 IP NSAT.mynet > Monster.mynet: ICMP echo reply, id 2954, seq 1, length 64
20:02:50.092836 ARP, Request who-has NSAT.mynet tell myhome.mynet, length 46
20:02:50.092857 ARP, Reply NSAT.mynet is-at 08:00:27:5f:cf:aa (oui Unknown), length 28
```

B.5.8. Resolve Protocols

 # Engine.py --module sniffer --sniff-type ip --resolve

```
root@NSAT:~# Engine.py --module sniffer --sniff-type ip --resolve
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on eth0, link-type EN10MB (Ethernet), capture size 262144 bytes
18:43:31.200876 IP 192.168.1.235.22 > 192.168.1.112.33670: Flags [P.], seq 3622478406:36224785
14, ack 220245837, win 272, options [nop,nop,TS val 405775 ecr 501065], length 108
18:43:31.201048 IP 192.168.1.112.33670 > 192.168.1.235.22: Flags [.], ack 108, win 1444, optio
ns [nop,nop,TS val 501237 ecr 405775], length 0
18:43:31.201125 IP 192.168.1.235.22 > 192.168.1.112.33670: Flags [P.], seq 108:144, ack 1, win
 272, options [nop,nop,TS val 405776 ecr 501237], length 36
18:43:31.201165 IP 192.168.1.112.33670 > 192.168.1.235.22: Flags [.], ack 144, win 1444, optio
ns [nop,nop,TS val 501237 ecr 405776], length 0
18:43:31.201207 IP 192.168.1.235.22 > 192.168.1.112.33670: Flags [P.], seq 144:252, ack 1, win
 272, options [nop,nop,TS val 405776 ecr 501237], length 108
18:43:31.201242 IP 192.168.1.112.33670 > 192.168.1.235.22: Flags [.], ack 252, win 1444, optio
ns [nop,nop,TS val 501237 ecr 405776], length 0
18:43:31.201276 IP 192.168.1.235.22 > 192.168.1.112.33670: Flags [P.], seq 252:288, ack 1, win
 272, options [nop,nop,TS val 405776 ecr 501237], length 36
```

B.5.9. Save Traffics

 # Engine.py --module sniffer --sniff-type arp --save

```
root@NSAT:~# Engine.py --module sniffer --sniff-type arp --save
tcpdump: listening on eth0, link-type EN10MB (Ethernet), capture size 262144 bytes
^C13 packets captured
13 packets received by filter
0 packets dropped by kernel
```

No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000	Tilgin_83:a5:6c	Broadcast	ARP	60	Who has 192.168.1.112? Tell 192
2	13.895165	Apple_	Broadcast	ARP	60	Who has 192.168.1.236? Tell 0.0
3	14.220604	Apple_	Broadcast	ARP	60	Who has 192.168.1.236? Tell 0.0
4	14.545967	Apple_	Broadcast	ARP	60	Who has 192.168.1.236? Tell 0.0
5	14.871486	Apple_	Broadcast	ARP	60	Gratuitous ARP for 192.168.1.23
6	15.194465	Apple_	Broadcast	ARP	60	Gratuitous ARP for 192.168.1.23
7	15.522391	Apple_	Broadcast	ARP	60	Gratuitous ARP for 192.168.1.23
8	15.525788	Apple_	Broadcast	ARP	60	Who has 192.168.1.1? Tell 192.1
9	16.333947	Apple_	Broadcast	ARP	60	Who has 192.168.1.1? Tell 192.1
10	18.136391	Micro-St_45:b5:d1	Broadcast	ARP	60	Who has 192.168.1.235? Tell 192

B.6. Recon Module

B.6.1. Subdomain Finder

 # Engine.py --module recon --recon-type subdomain

```
root@NSAT:~# Engine.py --module recon --recon-type subdomain
Target Domain: ieu.edu.tr
DNS Server IP [Press enter for '8.8.8.8']:
ns1.ieu.edu.tr 193.255.108.3
ns2.ieu.edu.tr 193.255.108.4
pop.ieu.edu.tr 193.255.108.79
mail.ieu.edu.tr 193.255.108.79
imap.ieu.edu.tr 193.255.108.79
gateway.ieu.edu.tr 193.255.108.5
```

B.6.2. Banner Detection

 # Engine.py --module recon --recon-type banner

```
root@NSAT:~# Engine.py --module recon --recon-type banner
Target Domain: gmail.com
HTTP/1.0 302 Found
Cache-Control: private
Content-Type: text/html; charset=UTF-8
Referrer-Policy: no-referrer
Location: http://www.google.com.tr/?gfe_rd=cr&ei=CpLzWN_LD4qg8wfL0ZjIAQ
Content-Length: 262
Date: Sun, 16 Apr 2017 15:47:22 GMT

<HTML><HEAD><meta http-equiv="content-type" content="text/html;charset=utf-8">
<TITLE>302 Moved</TITLE></HEAD><BODY>
<H1>302 Moved</H1>
The document has moved
<A HREF="http://www.google.com.tr/?gfe_rd=cr&amp;ei=CpLzWN_LD4qg8wfL0ZjIAQ">here</A>.
</BODY></HTML>
```

B.6.3. Traceroute

 # Engine.py --module recon --recon-type traceroute

```
root@NSAT:~# Engine.py --module recon --recon-type traceroute
Target Domain: google.com
traceroute to google.com (216.58.205.238), 30 hops max, 60 byte packets
 1  myhome.mynet (192.168.1.1)  0.762 ms  1.119 ms  1.594 ms
 2  93.155.2.52 (93.155.2.52)  7.970 ms  8.201 ms  8.162 ms
 3  81.212.105.5.static.turktelekom.com.tr (81.212.105.5)  8.360 ms  8.602 ms  8.692 ms
 4  81.212.208.218.35-hatay-t2-2.35-hatay-t3-2.statik.turktelekom.com.tr (81.212.208.218)  8.9
57 ms  8.906 ms  8.982 ms
 5  mugla-t3-1-mugla-t3-2.turktelekom.com.tr (81.212.206.197)  9.685 ms  10.101 ms  10.127 ms
 6  * * *
 7  74.125.52.6 (74.125.52.6)  78.359 ms  76.533 ms 74.125.51.92 (74.125.51.92)  108.499 ms
 8  72.14.237.27 (72.14.237.27)  39.811 ms 209.85.244.181 (209.85.244.181)  41.100 ms  42.141
ms
 9  209.85.240.160 (209.85.240.160)  108.187 ms  108.460 ms  108.138 ms
10  216.239.57.244 (216.239.57.244)  57.553 ms  63.009 ms 108.170.236.249 (108.170.236.249)  6
2.949 ms
11  108.170.232.180 (108.170.232.180)  58.878 ms 209.85.243.130 (209.85.243.130)  68.975 ms 10
8.170.232.180 (108.170.232.180)  67.622 ms
12  216.239.47.87 (216.239.47.87)  66.766 ms 216.239.47.53 (216.239.47.53)  57.507 ms 72.14.23
5.245 (72.14.235.245)  53.816 ms
13  216.239.48.45 (216.239.48.45)  57.424 ms 216.239.48.43 (216.239.48.43)  53.358 ms 216.239.
48.45 (216.239.48.45)  58.914 ms
14  fra15s24-in-f14.1e100.net (216.58.205.238)  82.579 ms  53.412 ms  54.055 ms
```

B.6.4. Port Scan

 # Engine.py --module recon --recon-type portscan

```
root@NSAT:~# Engine.py --module recon --recon-type portscan
Target Domain: 192.168.1.1
Port 53 -> OPEN
Port 80 -> OPEN
```

III. Conclusion

As internet has become a huge part of our daily life, the need of network security has also increased exponentially from the last decade. Network security is an important filed that is increasingly gaining attention as the internet expands. The security and internet protocol were analyzed to determine the necessary security technology.
In conclusion attacks on network security are a constant threat. Analyzing and detecting threats are a necessity for a complete network security system. Security is everybody's business, and only with everyone's cooperation, an intelligent policy, and consistent practices, will it be achievable.