applied sciences

Article

TRAP: A Three-Way Handshake Server for TCP Connection Establishment

Fu-Hau Hsu [1], Yan-Ling Hwang [2], Cheng-Yu Tsai [3], Wei-Tai Cai [1], Chia-Hao Lee [1,*] and KaiWei Chang [1]

1 Department of Computer Science and Information Engineering, National Central University, Taoyuan 32001, Taiwan; hsufh@csie.ncu.edu.tw (F.-H.H.); taya86334@gmail.com (W.-T.C.); popdata520@gmail.com (K.C.)
2 School of Applied Foreign Languages, Chung Shan Medical University, Taichung 40201, Taiwan; yanling@csmu.edu.tw
3 CyberTrust Technology Institute, Institute for Information Industry, Taipei 10622, Taiwan; josephtsai@iii.org.tw
* Correspondence: diolee@csie.ncu.edu.tw; Tel.: +886-978-765-069

Academic Editor: Christos Bouras
Received: 9 September 2016; Accepted: 9 November 2016; Published: 16 November 2016

Abstract: Distributed denial of service attacks have become more and more frequent nowadays. In 2013, a massive distributed denial of service (DDoS) attack was launched against Spamhaus causing the service to shut down. In this paper, we present a three-way handshaking server for Transmission Control Protocol (TCP) connection redirection utilizing TCP header options. When a legitimate client attempted to connect to a server undergoing an SYN-flood DDoS attack, it will try to initiate a three-way handshake. After it has successfully established a connection, the server will reply with a reset (RST) packet, in which a new server address and a secret is embedded. The client can, thus, connect to the new server that only accepts SYN packets with the corrected secret using the supplied secret.

Keywords: TCP; DDoS; three-way handshake; SYN flood attack; Netfilter; TCP options

1. Introduction

With the growing number of personal computers (PCs) connected to the Internet, malicious attackers try to break into and make good use of them. The term "botnet" is a network which is comprised of infected end-hosts under the control of a human operator, commonly known as the botmaster [1]. In Q1 2015, 23,095 distributed denial of service (DDoS) attacks were reported, targeting web resources in 76 countries, and SYN DDoS and Hypertext Transfer Protocol (HTTP) DDoS were the most common scenarios for botnet-assisted DDoS attacks [2].

If the attack originates from a single host on the network, it is defined as a denial of service (DoS) attack. If the attack originates from multiple devices across multiple networks, we call it a DDoS attack.

Botnets usually participate in performing DDoS attacks. DoS/DDoS attacks have a long history, back to the 1990s, and have become more and more vicious. In 2013, a massive DDoS attack was launched against Spamhaus, a non-profit anti-spam mail organization. Up to 75 Gbps of Domain Name System (DNS) reflection traffic were directed to Spamhaus' servers [3] causing the service to overload. Another example is the famous code-sharing website, GitHub (San Francisco, CA, USA), suffered from a massive DDoS attack in March 2015 [4]. A DDoS was caused by some nefarious JavaScript which was injected by "a certain device at the border of China's inner network and the Internet" when people visited the Baidu search engine [5].

In this paper, we propose a TCP three-way handshaking server, called TRAP hereafter, which utilizes the options field in the TCP headers to redirect legitimate client traffic to the real service server to handle the DoS/DDoS attacks. When a DoS/DDoS attack occurs, the following results usually appear on the attached system. First, the attacked service server may crash. Second, the attacked service server may no longer be able to provide service to any client. TRAP wants to solve the first problem and mitigate the second problem. First, only after a normal user completes a three-way handshake with our TRAP server, our TRAP server will dynamically notify the user of the IP of the service server. Then the user connects to the service server directly through IP translation without the need to connect to the service server through TRAP. Since the target of the attacked host is our TRAP, not the service server, the service server will not crash. Additionally, because the only job TRAP needs to do is to complete three way handshakes with clients, TRAP can handle a request quickly and can recover quickly after it crashes.

Second, because TRAP can service each user quickly and it can recover from a crash quickly, when a DoS/DDoS attack occurs on a TRAP some legitimate users still can obtain the service of the TRAP to connect to the real service server. Hence, the service server still works. However, without TRAP, legitimate users may not be able to obtain any service from the service server.

The above solution may bring the following concerns: first, an attacker may saturate the bandwidth of the TRAP so that it cannot connect to the Internet; and second, an attacker may disguise themself as a legitimate user to obtain the real IP of a service server from the TRAP. For the first concern, we think the possible solutions should be provided by Internet Service Providers (ISPs). After all, only they can control the traffic following into a host. For the second concern, TRAP can solve the problem easily using a stateful firewall, which is a network firewall that tracks the operating state and characteristics of network connections and can easily drop packets that do not belong to any established TCP connection and limit the amount of traffic a TCP connect can create. Additionally, because three handshakes are handled by TRAP, a service server can drop any SYN packet directly. Thus, attackers cannot launch a SYN flood against a service server protected by a TRAP.

In this paper, we demonstrate the basic ability of TCP connection redirection. In future works, more functionality can be implemented. To demonstrate the basic ability of simple DDoS attack mitigation, we perform SYN-flood attacks on the test server.

The rest of this paper is organized as follows: Section 2 explains the related background knowledge for understanding our work, TRAP; Section 3 discusses related work; Section 4 describes our system mechanisms; Section 5 displays the experimental results; and Section 6 concludes this paper.

2. Background

In this chapter, we will discuss the related background knowledge for understanding our work, TRAP, in detail.

2.1. Transmission Control Protocol

Transmission Control Protocol (TCP), as defined in Request For Comments (RFC) 793, is a connection-oriented, end-to-end reliable protocol. Originally, it was designed to fit into a layered hierarchy that supports multi-network applications [6]. The specifications including connection establishment, error-recovery, flow control, and window size negotiations.

In a typical TCP connection, each device maintains its state and the corresponding sequence number to track the incoming sequences of packets. When an end-host device receives a new packet, it will send back an ACK (acknowledgement) packet, containing an acknowledgement number, which means the device has successfully received the data and is awaiting further incoming data as the number indicates.

In the establishment of a TCP connection between a client and a server, a TCP three-way handshake process is performed. The process is as illustrated in Figure 1.

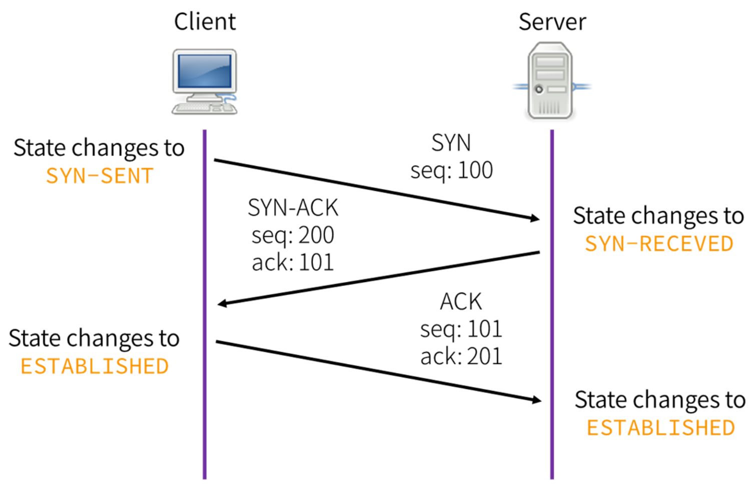


Figure 1. TCP three-way handshake diagram. TCP, Transmission Control Protocol; SYN, Synchronize; ACK, Acknowledgement.

1. The client sends a SYN (synchronize) packet to the server, which has a random sequence number.
2. The server sends back a SYN-ACK packet, containing a random sequence number and an ACK number acknowledging the client's sequence number.
3. The client sends an ACK number to the server, acknowledging the server's sequence number.
4. The sequence numbers on both ends are synchronized. Both ends can now send and receive data independently.

2.2. *Denial of Service Attacks*

A denial-of-service attack is characterized by an explicit attempt to prevent the legitimate use of a service [7]. For example, a Teardrop attack is defined as when a client sends a malformed packet containing an erroneous fragment number, certain servers will not be able to parse the packet, causing its network stack to crash [8]. On the other hand, for the security of websites, a Slow HTTP DoS attack takes advantage of a vulnerability in thread-based web servers which wait for entire HTTP headers to be received before releasing the connection [9]. An attacker sends HTTP requests in pieces slowly, one at a time to a Web server. If an HTTP request is not complete, or if the transfer rate is very low, the server keeps its resources busy waiting for the rest of the data. When the server's concurrent connection pool reaches its maximum, this creates a DoS [10].

Botnets or zombie armies usually participate in such attacks, being controlled and coordinated by a bot master.

2.3. *SYN Flood*

In the classic server-client TCP connection establishment during the three-way handshaking process, the client will have to send a SYN packet to the server. When a server receives a SYN packet from a client, it reserves its memory for the establishment of a TCP connection with the client, and then sends back a SYN-ACK packet.

As the resource (e.g., kernel memory, network adapter cache) on the server is limited; if a client never sends back an ACK packet, the allocated memory cannot be recycled or reused by any other clients before the timeout occurs. If a server receives a large amount of SYN packets from many other clients, but not any SYN-ACK packets, it will exceeds a server's resource limit, causing it to be unable to handle any new clients. We call this scenario a SYN flood attack. Figure 2 is an example illustration of a SYN flood attack scenario.

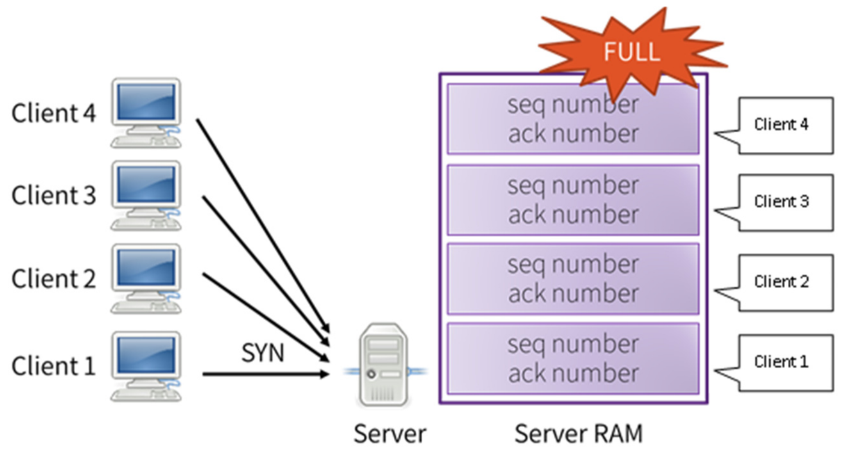


Figure 2. Illustration of SYN flood attacks.

Moreover, a malicious attacker can send forged SYN packets with fake IP addresses (IP spoofing), hiding it from the server. With its source IP address changing, traditional methods like blocking the offending IP address will have little effects.

Among all forms of DDoS attacks, a SYN-flood attack is the most prevalent one, because both are easy to implement and have a low cost to perform [11]. Moreover, it is, by standard, difficult to mitigate due to the design flaws in the TCP standard.

2.4. TCP Options

Internet Assigned Numbers Authority (IANA) has defined various TCP options in the TCP connection headers, including window scale, selective ACK (SACK) implementation, and maximum segment sizes (MSS) in RFC2780 [12].

As Figure 3, TCP options is defined in three fields: a type number, option length, and option data. In total, TCP options can store up to 40 bytes of data.

Some of the commonly used option numbers are maximum segment size (0x02), windows scale (0x03), selective acknowledgement number (0x04), and TCP timestamps (0x08).

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|          Source Port          |       Destination Port        |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                        Sequence Number                        |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                    Acknowledgment Number                      |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|  Data |           |U|A|P|R|S|F|                               |
| Offset| Reserved  |R|C|S|S|Y|I|            Window             |
|       |           |G|K|H|T|N|N|                               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|           Checksum            |         Urgent Pointer        |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                    Options                    |    Padding    |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                             data                              |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Figure 3. TCP header format (reference source: RFC-793) [6].

2.5. Netfilter

Netfilter is a Linux kernel networking filter framework, usually compiled as a Loadable Kernel Module (LKM), offering various hooks in the network layer of the packet manipulation. Kernel modules are pieces of code that can be loaded and unloaded into the kernel upon demand [13]. It also provides various functions, like altering the packet's source or destination Media Access Control (MAC) address and filtering based on certain criteria like TCP port and redirecting it to a new IP address.

Netfilter provides a set of tables, which indicates the traversal of packets throughout the kernel space and the user space. Inside these tables are sets of pre-defined chains, which ultimately contain sets of rules. By using the user-space program iptables, a user is freely to modify these rules to control the behavior of how network packets are processed by a Linux kernel [14].

Netfilter provides following hook points for a packet as depicted in Figure 4:

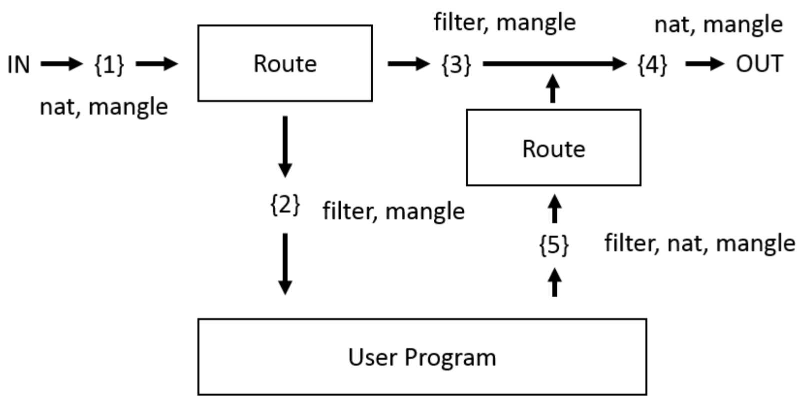


Figure 4. Netfilter hooks diagram (reference source: netfilter.org) [11].

1. Having passed simple sanity checks (i.e., not truncated, IP checksum, etc.), a network packet is passed to the NF_INET_PRE_ROUTING {1} hook.
2. Next, it enters the routing decision to determine if the packet is for the local machine. If it is true, then it is passed to the NF_INET_LOCAL_IN {2} hook, before going to the user-space program.
3. If the packet is for another interface or device, it is passed to the NF_INET_FORWARD {3} hook.
4. The packet then passes a final hook, the NF_INET_POST_ROUTING {4} hook, before being put on the wire again.
5. For any outgoing packet from the user-space program, NF_INET_LOCAL_OUT {5} hook is called first, before going to the routing decision.

A kernel module can register to listen at any of these hooks. When a packet arrives at the specified hook, the kernel module gets the raw packet buffer, and is free to manipulate the packet's content or header.

The return value can be one of the five value:

1. NF_ACCEPT: continue the packet traversal as normal.
2. NF_DROP: drop the packet and free the buffer.
3. NF_STOLEN: stop the traversal but not freeing the memory, the kernel module will take control afterwards.
4. NF_QUEUE: queue the packet.
5. NF_REPEAT: call this hook again.

3. Related Work

Since the emergence of DDoS attacks, many defense solutions have been proposed, but not any effective solutions have been discovered, yet. Most of the solutions focus on how to mitigate the impact of the attack temporarily.

In this chapter, we will examine current approaches on how to mitigate the problem and their related work.

3.1. SYN Cookie

In order to prevent a SYN flood attack from exhausting all of a server's resources, when receiving a SYN packet, the client information required for future connection like the source IP address and the TCP port number are encoded directly into the replying SYN-ACK packet's initial sequence number (ISN).

When a client replies with an ACK packet, the packet's acknowledge number are extracted, and compared with a newly-generated hash based on the client's information to verify if the packet is a continuation from a previous connection and a previous verified host. In addition to the client information, a time-window value is also checked to verify the integrity of the cookie.

A server adopting SYN cookies does not save any state of the client in the memory prior to its establishment, thus will not be facing resource outages when a SYN flood occurs. Figure 5 shows a simplified illustration of the procedure of SYN cookies.

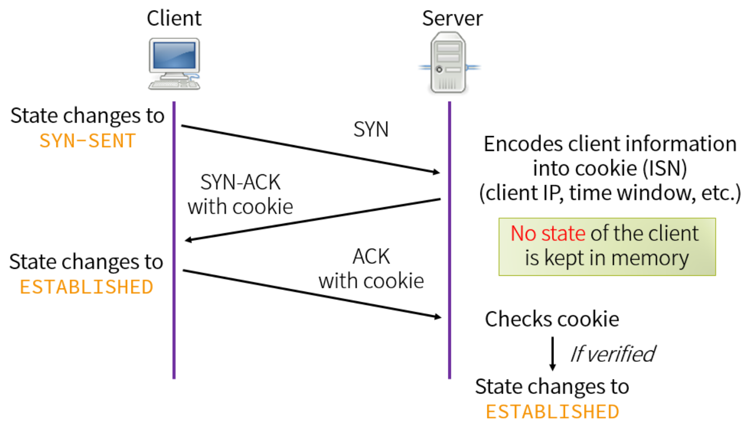

Client
Server
State changes to SYN-SENT
SYN
Encodes client information into cookie (ISN) (client IP, time window, etc.)
SYN-ACK with cookie
No state of the client is kept in memory
State changes to ESTABLISHED
ACK with cookie
Checks cookie
If verified
State changes to ESTABLISHED

Figure 5. Illustration of TCP Cookies.

However, as the TCP protocol specific requires the retransmission of unacknowledged data, after applying SYN cookies, this will be impossible since no state of a client is kept on the server [15]. This may also prevent a legitimate client from connecting to the server.

The hash generation process may also cost many CPU cycles under heavy network load, defecting its purpose of mitigating a DDoS attack.

3.2. SYN Proxy

In order to distinguish legitimate clients and attacking clients, we can setup an intermediate server to filter the connection. If a client completes a TCP three-way handshake with the intermediate server, we consider this client as a legitimate client, and then the intermediate server will hand off the established socket to the backend servers. The intermediate server will translate the sequence number and acknowledgement number between the clients and the backend servers. We call this infrastructure a SYN proxy.

SHAK [16] is an implementation based on this approach. By installing a separate front-end server between the back-end servers and the clients, the front-end server not only acts as a SYN proxy server, but also a load balancer. When a client connects to the front-end node, it chooses a back-end server to handle the request and reply to the clients with a "address remap" ACK packet. The client will then modify its stored socket destination address to the new one, handing off the socket.

Similar socket handoff schemes were proposed by Wenting Tang, in which a SYN-ACK and TCP state are translated via an intermediate proxy server [17]. In the implementation, the TCP/IP protocol stack was heavily modified to accommodate the project needs.

However, these methods require extensive modification to the kernel and the TCP/IP stack, limiting its appliance across devices. On situations that modify the clients' kernel is not appropriate, this causes implementation difficulties.

In 2006, Zhijun Wu and Zhifeng Chen proposed a three-layer defense model against DDoS attacks. When a client attempts to connect to the SYN-Proxy enabled server, a SYN is sent to the intermediate firewall. The firewall will then send back a SYN-ACK packet, finishing the TCP three way handshake with the client [18]. If a client successfully established a connection with the firewall (thus proving itself not being a SYN flood attacker), the firewall then connects to a real server. The firewall acts as a transparent proxy server afterwards, only translating the TCP sequence number; hence, the name SYN Proxy.

This method was implemented in the Netfilter framework, and is released along with the Linux kernel version 3.12 (2013) as SYNPROXY. SYNPROXY depends on SYN cookies and TCP timestamps [19].

This architecture can mitigate moderate SYN flood attacks; however, the added components will make the server harder to maintain due to the complex design. The sequence number translating process cannot be load-balanced using multiple firewalls due to the nature of sequence number synchronization.

4. System Design

In this chapter, we will describe our work—TRAP—in terms of design and implementation.

4.1. System Overview

Figure 6 shows the overview of our system in operation.

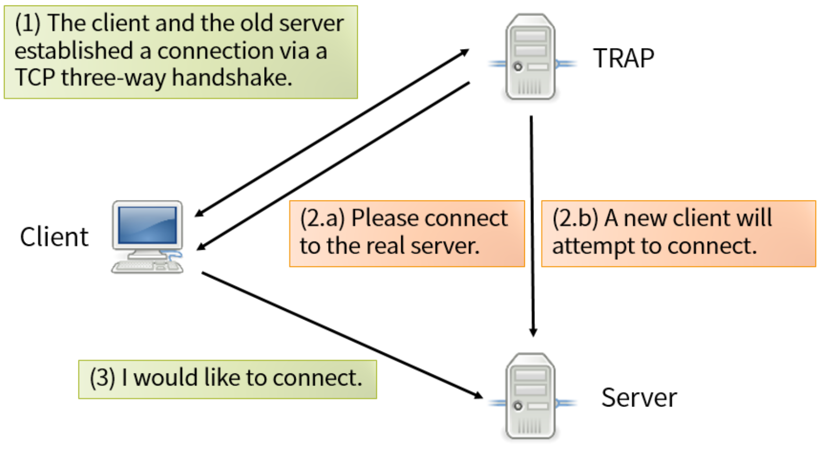

(1) The client and the old server established a connection via a TCP three-way handshake.
TRAP
Client
(2.a) Please connect to the real server.
(2.b) A new client will attempt to connect.
(3) I would like to connect.
Server

Figure 6. System overview.

TRAP is comprised of three components:

1. TRAP server
2. Client module
3. Server module

In this section, we will describe these three components.

4.1.1. TRAP Server

A TRAP server is a server program listening for incoming requests. When TRAP receives incoming SYN packets, it will attempt to establish a TCP connection via the TCP three-way-handshake with the incoming client via a TCP three-way handshaking process by sending a SYN-ACK packet.

If a client successfully established a TCP connection with the TRAP server, the TRAP server will then send back a redirect command packet (TCP reset (RST) packet in our implementation) to redirect the client to a new server IP address and a secret hash utilizing the TCP header options.

The TRAP server will also send a control message contained in a TCP RST packet to the new server, but the IP address in the TCP header option are replaced with the client's IP address, indicating a new client will attempt to connect soon.

The TRAP server will reject any packets other than packets required for TCP connection establishment (SYN packets and ACK packets).

4.1.2. Client Module

The client module will act as a daemon listening in the background, installed on the legitimate client machine. If it receives a TRAP command packet, it will attempt to parse the TCP header options to see if it contains the specified TCP option type number and a server IP address.

If the packet is successfully parsed with optional data extracted, the client module then alternates the Netfilter table using iptables, the user space program, to redirect any new connection from the old server IP to the new server IP.

4.1.3. Server Module

In order to process the packet efficiently without sacrificing performance, the server module is designed as a loadable kernel module written using C and various Netfilter hooks.

The new server module will listen for any incoming TCP connection requests. If a packet originates from the TRAP server IP address, which is configured at runtime, it will then parse the IP address stored inside the TCP header option with the secret. It will then check the secret to see if it matches the secret defined at the server. If the secret hash matches, the kernel module will then save the new client IP address in a array allocated by kmalloc().

For an incoming new packet, a lookup is performed. If the server module finds a match in the stored IP array, then the packet is accepted and passed to the user-space client program. If the IP address does not match any saved record, then the packet is discarded. Figure 7 shows an overview of the TRAP's modules.

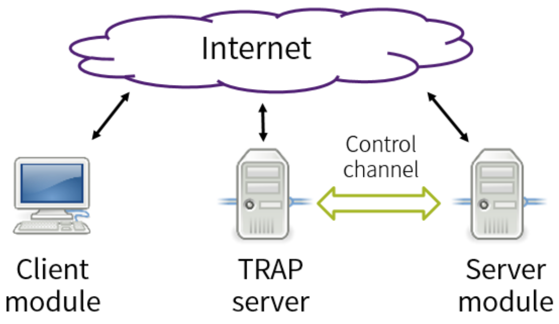


Figure 7. System design.

4.2. System Implementation

In this section, we will describe how we implement TRAP and its components.

4.2.1. TRAP Server and Client Module

Libpcap is an open source library providing various packet dumping Application Programming Interfaces (APIs) [20]. It can capture packets in lower system level utilizing system calls in the Linux kernel, providing raw packet buffers for programmers in user space. Figure 8 shows a sample function code of the libpcap library.

```
void get_packet(char *interface, u_long dstip, struct seqack *sa){
    pcap_t              *pt;
    char                ebuf[PCAP_ERRBUF_SIZE];
    u_char              *buf;
    struct ip           iph;
    struct tcphdr       tcph;
    int                 ethrhdr;

    pt = pcap_open_live(interface, 65535, 1, 60, ebuf);
    ethrhdr = 14;

    for (;;) {
        struct pcap_pkthdr pkthdr;
        buf = (u_char *) pcap_next(pt, &pkthdr);
        if (!buf)
            continue;

        // do something to the packet.
        pcap_close(pt);
    return;
    }
}
```

Figure 8. Acquiring packet buffer using libpcap.

Libnet is an open source library providing various functions and macros, and is capable of constructing packets and sending them using raw sockets [21]. During the packet construction, one can specify many custom settings to the packet, such as window scaling size, sequence numbers, and the TCP options.

Both the TRAP server and the client module are developed using C, libpcap, and libnet.

4.2.2. Server Module

We developed the server module as a Linux kernel module (LKM) using C and Netfilter hooks. By implementing the NF_INET_LOCAL_IN Netfilter hook, we can intercept incoming packets in the Linux kernel, thus performing our designated filtering work.

Figure 9 shows an example code of filtering IP packets and printing the addresses to the /proc/kmsg.

```
static unsigned int hook_func(
const struct nf_hook_ops *ops,
struct sk_buff *skb,
const struct net_device *in,
const struct net_device *out,
int (*okfn)(struct sk_buff *))
{

sock_buff = skb;
if (!sock_buff)
return NF_ACCEPT;

struct ethhdr *mac_header = (struct ethhdr *)skb_mac_header(skb);
struct iphdr *ip_header = (struct iphdr *)skb_network_header(skb);
if (!ip_header)
return NF_ACCEPT;

if(ip_header->protocol == IPPROTO_TCP)
{
int src_port = ntohs(tcp_header->source);
int dst_port = ntohs(tcp_header->dest);

printk(KERN_INFO "src_ip: %pI4:%d\n", &ip_header->saddr, src_port);
printk(KERN_INFO "dst_ip: %pI4:%d\n", &ip_header->daddr, dst_port);
}

return NF_ACCEPT;
}
```

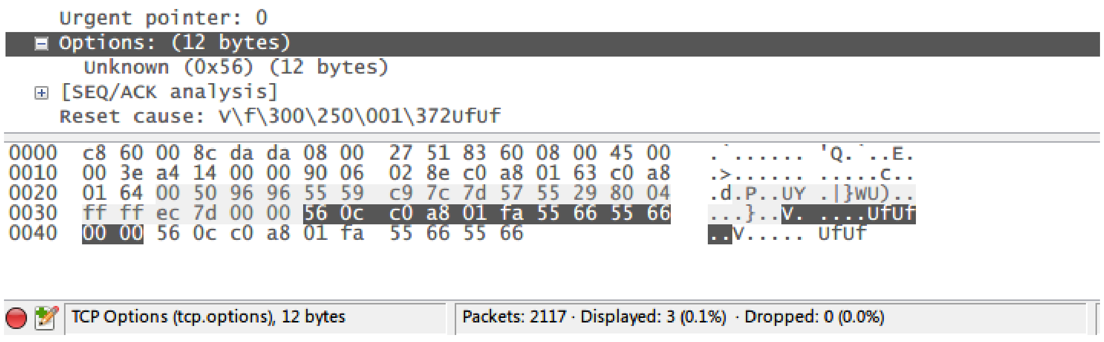
Figure 9. Example code of acquiring packets using Netfilter hooks.

4.2.3. TCP Option Types

In order not to conflict with any existing implementation of TCP stacks for compatibility reasons, we chose a reserved number, 0x56, as our TCP option type number.

Both the server module and the client module rely on this option number to check if the packet contains a TRAP command. If a packet does not contain such option number, it is simply ignored as not being a TRAP command.

We utilized Wireshark, the packet-dumping program, to monitor the packet flow [22]. Figure 10 shows an example of viewing the raw packet using Wireshark.



```
    Urgent pointer: 0
 ⊟  Options: (12 bytes)
      Unknown (0x56) (12 bytes)
 ⊞  [SEQ/ACK analysis]
    Reset cause: V\f\300\250\001\372UfUf

0000  c8 60 00 8c da da 08 00  27 51 83 60 08 00 45 00   .`...... 'Q.`..E.
0010  00 3e a4 14 00 00 90 06  02 8e c0 a8 01 63 c0 a8   .>...... .....c..
0020  01 64 00 50 96 96 55 59  c9 7c 7d 57 55 29 80 04   .d.P..UY .|}WU)..
0030  ff ff ec 7d 00 00 56 0c  c0 a8 01 fa 55 66 55 66   ...}..V. ....UfUf
0040  00 00 56 0c c0 a8 01 fa  55 66 55 66               ..V..... UfUf

TCP Options (tcp.options), 12 bytes    Packets: 2117 · Displayed: 3 (0.1%) · Dropped: 0 (0.0%)
```

Figure 10. Packet dump viewed by Wireshark.

5. Evaluation

In this chapter, we will discuss how we evaluate the TRAP's implementation overhead, and perform real-life evaluation tests to see if the overhead matches our standards.

5.1. Lab Environment

Our lab environment includes is a desktop PC (ASUS, Taipei, Taiwan) acts as a virtual machine (VM) host, hosting three virtual machines.

The virtual machine host has an Intel Xeon E3-1231v3 (3.4 GHz, 4 physical cores) CPU, 16 GB of RAM, and a 1 TB hard disk. The virtual machine emulator is Oracle VM VirtualBox version 4.3.28 r100309 (Santa Clara, CA, USA, 2015). We configured three virtual machines to have a single-cored CPU and 768 MB of RAM each. We configure all of the virtual machines to bridge with a physical gigabit network adapter (providing 1 Gb/s of Internet speed).

5.2. Evaluation Results

We wrote a simple Python script to test a TRAP-enabled server and compare the results with the results without TRAP. Hence, we can compute the percentages of time that our module costs.

The script will attempt to download a file hosted on a TRAP-enabled server, and the connection will be directed to the real server instead. On the same environment, we will record the time needed to download the file directly from the server to compare the differences. The files used for the test ranges from five megabytes to 100 megabytes of sizes. These files contain random contents generated from /dev/urandom. Table 1 shows the results of the overhead evaluation.

Table 1. Evaluation of overhead.

File Size (M)	With TRAP (ms)	Without TRAP (ms)	Overhead (ms)
5	225.70	77.92	147.78
10	380.93	214.78	166.15
50	748.70	567.42	181.28
100	1360.13	1137.39	222.74

In addition to overhead evaluation, we also conduct a performance evaluation simulating an ongoing DDoS attack.

We use the hping program, an open-source network tool able to send custom TCP/IP packets. Hping is a command-line TCP/IP packet assembler/analyzer [23]. The tool is able to randomize each packet's source IP address and set its SYN flag and, thus, can be used to simulate SYN flood attacks. As an action, "hping3 –S –P –p 80 192.168.1.99 –i u10 –rand-source", the parameter we used for hping to simulate a DDoS attack. "–rand-source" is used to spoof the packet's source IP, and "-s" is used to indicate the packet has a SYN flag. We sent a file of 5 MB with and without TRAP. It spent 253.03 ms with TRAP and spent 2935.06 ms without TRAP which gives an evident evaluation results while a DDoS simulated attack is undergoing.

On the other hand, we processed an effective test among SYN cookies, SYN proxy, and our TRAP system. To calculate the average time period, we conducted a three-round TCP connecting and disconnecting experiment. Twenty connections were conducted in every round; hence, there were a total of 60 connection attempts for each of the three defense mechanisms. In this experiment, each target TCP server installed with one of the three defense mechanisms sent a response as soon as it received a connection request. We calculated the time period starting at the time a client sent a TCP connection request and ending at the time that the client received the connection response from the server. The averages of one TCP connection time are: 7683.6 µs with TRAP, 7820.3 µs with SYN Cookies, and 7978.3 µs with SYN Proxy, from an experiment of 60 TCP connections for each security scheme. We found that TRAP is more effective than the others.

5.3. Discussion

The performance overhead occurs during the time a client receives a command, and the TCP connection being redirected to the specified server. Compared to regular TCP connection establishment, a TRAP-enabled server requires the TCP three-way handshake process one more time. This may be affected by network latency, which we are unable to measure in our evaluation.

The redirection process affects all new future connections, and all new TCP connections towards the specified resource are automatically being handed off to the server. Hence, this overhead only happens once per client. This one-time overhead averages to 179.49 ms.

As for the evaluation performed while undergoing a simulated DDoS attack, the results are much more promising. While it takes a long time for regular servers to transport a file to a client (2935 ms), a TRAP-enabled server and client only needs 253 ms for the connection redirection.

6. Conclusions

In this paper, we propose a TRAP, a TCP three-way handshake connection-establishing server.

As any TCP connection establishment must go through the three-way handshake server, one can implement many functionalities on the TRAP. In this thesis, we configure the TRAP to defend against SYN flood DDoS attacks, but we believe that, with more extension modules, we can enhance the TRAP's ability to handle more functionality, like traffic filtering, user authentication, and more.

We designed the TRAP as a modular component, so that each component can be replaced or updated independently, without sacrificing the overall performance.

More importantly, a TRAP server can introduce more traffic filtration functionality, like host separation, authentication, and connection redirection, based on certain client information. We leave this for future works.

With a tolerable one-time overhead, we can direct any future TCP connection to a remote server under many situations including, but not limited to, under a DDoS attack, virtual machine migration, and resource change. Additionally, since a DDoS attack is under the control of malicious users or unconscious bots, the above mentioned DDoS attack will not be interrupted from happening for some time. Still, it is also possible to protect our whole system from such threats.

An attacker may redirect his payload and directly target our TRAP server, which may make our solution ineffective. However, when a packet originates from the TRAP server IP, it will then parse the client IP stored inside the TCP header option and the secret hash. If verified as a legitimate request, the server module will then store the IP address and allowing its further connection. The server module will drop any unauthorized packets. Additionally, we can put a stateful inspection firewall in the frontend of the network structure to drop some abnormal SYN packets, which is common in commercial use.

Note that this solution cannot stop a DDoS attack from happening, but when an attack happens, we can effectively redirect authorized traffic to a remote, secure server to unload the burden of the original server being unable to provide service to any legitimate clients. If a DDoS attack deprecated all of the server's available bandwidth, clients may still be unable to connect to the TRAP. However, in a situation like this, we think any possible defense mechanism will have to rely on ISP-grade hardware or routers.

Acknowledgments: Our work is funded by Ministry of Science and Technology of Taiwan (ROC), and the numbers of the Projects are Most 105-2221-E-008 -074 -MY3 and MOST 104-2221-E-008-056-.

Author Contributions: All authors contributed equally to this work. Fu-Hau Hsu designed the study. Fu-Hau Hsu, Yan-Ling Hwang, Cheng-Yu Tsai, Wei-Tai Cai, Chia-Hao Lee and KaiWei Chang developed the methodology, collected data, and performed the analysis. Fu-Hau Hsu, Wei-Tai Cai and Chia-Hao Lee prepared the manuscript. Wei-Tai Cai and Chia-Hao Lee performed the experiments. Chia-Hao Lee communicated with the journal and coauthors and ensured the accuracy of all content in the proof.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Abu Rajab, M.; Zarfoss, J.; Monrose, F.; Terzis, A. A multifaceted approach to understanding the botnet phenomenon. In Proceedings of the 6th ACM SIGCOMM Conference on Internet Measurement, Rio de Janeiro, Brazil, 25–27 October 2006; pp. 41–52.
2. Kaspersky Lab. Statistics on Botnet-Assisted DDoS Attacks in Q1 2015. Available online: https://securelist.com/blog/research/70071/statistics-on-botnet-assisted-ddos-attacks-in-q1--2015/ (accessed on 11 November 2016).
3. Prince, M. The DDoS that Knocked Spamhaus Offline (and How We Mitigated It). Available online: https://blog.cloudflare.com/the-ddos-that-knocked-spamhaus-offline-and-ho (accessed on 11 November 2016).
4. Graham, R. Pin-Pointing China's Attack against GitHub. Available online: http://blog.erratasec.com/2015/04/pin-pointing-chinas-attack-against.html (accessed on 11 November 2016).
5. Sebastian, A. GitHub Battles "Largest DDoS" in Site's History, Targeted at Anti-Censorship Tools. Available online: http://arstechnica.com/security/2015/03/github-battles-largest-ddos-in-sites-history-targeted-at-anti-censorship-tools/ (accessed on 11 November 2016).
6. RFC 793—Transmission Control Protocol. Available online: https://tools.ietf.org/html/rfc793 (accessed on 11 November 2016).
7. Mirkovic, J.; Reiher, P. A taxonomy of DDoS attack and DDoS defense mechanisms. *ACM SIGCOMM Comput. Commun. Rev.* **2004**, *34*, 39–53. [CrossRef]
8. Juniper Networks, Inc. Understanding Teardrop Attacks. Available online: https://www.juniper.net/techpubs/software/junos-es/junos-es92/junos-es-swconfig-security/understanding-teardrop-attacks.html (accessed on 11 November 2016).
9. Muscat, I. How to Mitigate Slow HTTP DoS Attacks in Apache HTTP Server. Available online: http://www.acunetix.com/blog/articles/slow-http-dos-attacks-mitigate-apache-http-server/ (accessed on 11 November 2016).
10. Shekyan, S. How to Protect against Slow HTTP Attacks. Available online: https://blog.qualys.com/securitylabs/2011/11/02/how-to-protect-against-slow-http-attacks (accessed on 12 November 2016).
11. Miao, L.; Ding, W.; Gong, J. A real-time method for detecting internet-wide SYN flooding attacks. In Proceedings of the IEEE International Workshop Local and Metropolitan Area Networks (LANMAN), Beijing, China, 22–24 April 2015; pp. 1–6.
12. Transmission Control Protocol (TCP) Parameters. Available online: http://www.iana.org/assignments/tcp-parameters/tcp-parameters.xhtml#tcp-parameters-1 (accessed on 12 November 2016).
13. Salzman, P. The Linux Kernel Module Programming Guide. Available online: http://www.tldp.org/LDP/lkmpg/2.6/html/lkmpg.html#AEN40 (accessed on 12 November 2016).
14. Linux netfilter Hacking HOWTO: Netfilter Architecture. Available online: http://www.netfilter.org/documentation/HOWTO/netfilter-hacking-HOWTO-3.html (accessed on 13 November 2016).
15. Lemon, J. Resisting SYN Flood DoS Attacks with a SYN Cache. In Proceedings of the BSD Conference 2002 on BSD Conference, Amsterdam, The Netherlands, 15–17 November 2002; pp. 89–97.
16. Jin, H.; Tang, D.; Zhang, Y.; Chen, H. SHAK: Eliminating faked three-way handshaking in socket handoff. In Proceedings of the 18th International Parallel and Distributed Processing Symposium, Santa Fe, Mexico, 26–30 April 2004; p. 184.
17. Tang, W.; Cherkasova, L.; Russell, L.; Mutka, M.W. Modular TCP handoff design in STREAMS–based TCP/IP implementation. In *Networking—ICN 2001*; Lorenz, P., Ed.; Springer: Berlin/Heidelberg, Germany, 2001; pp. 71–81.
18. Wu, Z.; Chen, Z. A three-layer defense mechanism based on web servers against distributed denial of service attacks. In Proceedings of the First International Conference on Communications and Networking in China, Beijing, China, 25–27 October 2006; pp. 1–5.
19. McHardy, P. Netfilter: Implement Netfilter SYN Proxy. Available online: https://lwn.net/Articles/563151/ (accessed on 13 November 2016).
20. TCPDUMP/LIBPCAP public repository. Available online: http://www.tcpdump.org/ (accessed on 24 April 2005).
21. GitHub-Sam-Github/Libnet: Libnet Provides a Portable Framework for Low-Level Network Packet Construction. Available online: https://github.com/sam-github/libnet (accessed on 13 November 2016).

22. Wireshark Go Deep. Available online: https://www.wireshark.org/ (accessed on 13 November 2016).
23. Hping—Active Network Security Tool. Available online: http://www.hping.org (accessed on 13 November 2016).