Addressing Shortcomings of Existing DDoS Protection Software Using Software-Defined Networking

SSP 2018, Hildesheim

Lukas Iffländer, Stefan Geissler, Jürgen Walter, Lukas Beierlieb, Samuel Kounev

08.11.2018

http://se.informatik.uni-wuerzburg.de/
Motivation

- No definite defense possible, only mitigation
- Long time security threat
- More dangerous than ever:
  - Increasing number of IoT devices
  - Generally lower security level
- Marginal performance increase of defense systems hardware

Need for more effective mitigation approaches
SYN Flood

EXPLANATION
TCP is a reliable transport protocol:
- Retransmission of lost packets
- Sorting of out-of-order packets

Sequence number on every packet is necessary

Initial sequence numbers are established in a three-way handshake
TCP Handshake

**SYN packet**
- SYN set, seqn=client_isn

**SYN-ACK packet**
- SYN set, ACK set, seqn=server_isn, ackn = client_isn+1

**ACK packet**
- ACK set, seqn=client_isn+1, ackn = server_isn+1
SYN Flood

Client

SYN packet
spoofed source IP 1

SYN packet
spoofed source IP 2

SYN packet
spoofed source IP 3

SYN packet
spoofed source IP 4

Server

Backlog:

TCB
sIP1

TCB
sIP2

TCB
sIP3

TCB
sIP4

Addressing Shortcomings of Existing DDoS Protection Software Using Software-Defined Networking

Lukas Iffländer, Stefan Geissler, Jürgen Walter, Lukas Beierlieb, Samuel Kounev
SYN Flood Explanation

- Attacker can spoof any source IP address
- Server has to create TCB and keep it for a while
- SYN packets are small:
  - 14 byte (Ethernet header) + 20 byte (IP header) + 20 byte (TCP header) = 54 byte
  - 1 Mbit/s can transport 2314 pps
  - 1 Mpps requires 432 Mbit/s
SYN Flood

EXISTING DEFENSE MECHANISMS
SYN Cookies

**Client**
- **SYN packet**
  - SYN set, seqn=client_isn

**Server**
- **SYN-ACK packet**
  - SYN set, ACK set,
    - seqn=cookie,
    - ackn = client_isn+1

- **ACK packet**
  - ACK set, seqn=client_isn+1,
    - ackn = cookie+1

Calculate expected cookie, compare with ackn-1
SYN Cookies

- Amount of half-open connections not limited by backlog
- CPU is burdened with hash calculations
- TCP options are restricted
- Only active when backlog is full
Addressing Shortcomings of Existing DDoS Protection Software Using Software-Defined Networking

SYN PROXY – Connection Establishment

**Client**

- **SYN packet**
  - `seqn=client_isn`

**SYN PROXY**

- **SYN-ACK packet**
  - `seqn=proxy_isn`
- **ACK packet**

**Server**

- **SYN packet**
  - `seqn=client_isn`
- **SYN-ACK packet**
  - `seqn=server_isn`
- **ACK packet**
SYN PROXY – Data Transfer

Client

SYN PROXY

data packet
seqn=proxy_isn+X

data packet
ackn=proxy_isn+Y

data packet
seqn=server_isn+X

data packet
ackn=server_isn+Y

Server
SYN PROXY

- Implemented as an IPtables module
- Does not have to run on target machine
- Only complete handshakes reach target
- Proxy cannot predetermine server’s ISN
- seqn/ackn translation always necessary
Limitations

SYN Cookies
- Has to run on service host
- No independent scaling

SYN PROXY
- Stateful
- Network bottle neck
- Independent scaling complex
Problem:
- Existing solutions can not easily be scaled independently from the service host

Idea:
- Complete handshake in proxy
- Route subsequent packets directly

Benefit:
- Server handles only established connections
- Proxy can specialize on handshake handling

Action:
- Develop proxy network function, utilize SDN, modify server kernel
SYN Flood

SDN/NFV APPROACH
SDN and NFV

Software Defined Networking

- SDN switches’ behavior is determined by a set of flows
- SDN controller modify and monitor flow sets of connected switches
- A flow consists of:
  - Match
  - Action
  - Stats

Network Function Virtualization

- Network functions modify packets not addressed to them
  - Firewall
  - Switch
  - IDS
- Virtualized NF is running on COTS hardware (instead of being an ASIC)
SDN/NFV Approach

Traditional network
Addressing Shortcomings of Existing DDoS Protection Software Using Software-Defined Networking

**SDN/NFV Approach**

![SDN enabled network diagram]

**Attacker**

**Client**

**Gateway**

**Controller**

**OF-switch**

**VNF**

**Server**
Addressing Shortcomings of Existing DDoS Protection Software Using Software-Defined Networking

**SDN/NFV Approach**

<table>
<thead>
<tr>
<th>Prio</th>
<th>Match</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>GW</td>
<td>VNF</td>
</tr>
<tr>
<td>0</td>
<td>SERV</td>
<td>VNF</td>
</tr>
<tr>
<td>0</td>
<td>VNF</td>
<td>GW</td>
</tr>
<tr>
<td>1</td>
<td>VNF, daddr=s_ip</td>
<td>SERV</td>
</tr>
</tbody>
</table>
Attacker sends SYN packets with spoofed addresses

### SDN/NFV Approach

<table>
<thead>
<tr>
<th>Prio</th>
<th>Match</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>GW</td>
<td>VNF</td>
</tr>
<tr>
<td>0</td>
<td>SERV</td>
<td>VNF</td>
</tr>
<tr>
<td>0</td>
<td>VNF</td>
<td>GW</td>
</tr>
<tr>
<td>1</td>
<td>VNF, daddr=s_ip</td>
<td>SERV</td>
</tr>
</tbody>
</table>
**Client opens connection**

<table>
<thead>
<tr>
<th>Prio</th>
<th>Match</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>GW</td>
<td>VNF</td>
</tr>
<tr>
<td>0</td>
<td>SERV</td>
<td>VNF</td>
</tr>
<tr>
<td>0</td>
<td>VNF</td>
<td>GW</td>
</tr>
<tr>
<td>1</td>
<td>VNF, daddr=s_ip</td>
<td>SERV</td>
</tr>
<tr>
<td>10</td>
<td>GW, from client</td>
<td>SERV</td>
</tr>
<tr>
<td>10</td>
<td>SERV, to client</td>
<td>GW</td>
</tr>
</tbody>
</table>
IMPLEMENTATION
Implementation: Kernel Modification

- Simple concept
- Only 8 lines of code
- Kernel recompilation necessary
- Complete handshake required
Implementation: VNF

VNF is split in DPDK application and Python application

DPDK:
- Handshaking

Python:
- REST requests
- (HTTP flood defense)
INITIAL EVALUATION
Addressing Shortcomings of Existing DDoS Protection Software Using Software-Defined Networking

Lukas Iffländer, Stefan Geissler, Jürgen Walter, Lukas Beierlieb, Samuel Kounev
Testing Methodology

- Attacker floods SYN packets with delay between each packet
- Client sends 50 SYN packets in 0.5s intervals
- Score is the amount of answered client SYN packets
### First Results

<table>
<thead>
<tr>
<th>AVG Successfull Connections out of 50</th>
<th>No Protection</th>
<th>SYN Cookies</th>
<th>Our Approach</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>0</td>
<td>35</td>
<td>30.2</td>
</tr>
</tbody>
</table>

- Promising results for first implementation
- Feasability of idea verified
Limitations

- Client starts sending packets right after SYN-ACK
- At that moment still outstanding
  - Connection establishment with server
  - Adaptation of SDN-Configuration
- First packets are lost and have to be retransmitted.
CONCLUSION & FUTURE WORK
Conclusion

- Approach to use SDN to mitigate SYN-Flood attacks
  - Only complete handshakes reach service
  - VNF only responsible for establishing new connection
- Performant because of DPDK
- Scalable, since there are no dependencies and stateless
Future Work

- Evaluation with hardware NICs
- Comparison between different architectures for multi core
- Move Kernel modification in standalone module
- Analyze how to circumvent packet-loss problem
- On demand enabling
- Click & Use deployment (Docker/VM/Ansible)
Thank you for your attention!

Phone: +49 (931) 31 89947
Mail: lukas.ifflaender@uni-wuerzburg.de
Web: https://go.uniue.de/ifflaender


...
Addressing Shortcomings of Existing DDoS Protection Software Using Software-Defined Networking

Lukas Iffländer, Stefan Geissler, Jürgen Walter, Lukas Beierlieb, Samuel Kounев
Implementation: Kernel Modification

from net/ipv4/tcp_ipv4.c:

```c
static u32 tcp_v4_init_seq(const struct sk_buff *skb)
{
    return secure_tcp_seq(ip_hdr(skb)->daddr,
        ip_hdr(skb)->saddr,
        tcp_hdr(skb)->dest,
        tcp_hdr(skb)->source);
}
```
static u32 tcp_v4_init_seq(const struct sk_buff *skb) {
    struct tcphdr *tcph;
    unsigned char *payload_start, *payload_end;
    tcph = tcp_hdr(skb);
    payload_start = (unsigned char *)((unsigned char *)tcph+(tcph->doff * 4));
    payload_end = skb_tail_pointer(skb);

    if (payload_end - payload_start == 4)
        return cpu_to_be32(*((u32 *)payload_start));

    return secure_tcp_seq(ip_hdr(skb)->daddr,
                           ip_hdr(skb)->saddr,
                           tcp_hdr(skb)->dest,
                           tcp_hdr(skb)->source);
}
DPDK Features

- Poll-mode, userspace NIC drivers
- Lockless ringbuffer
- Cryptographic library
- Packet processing library (e.g. Fragmentation)
- Kernel Network Interfaces
- Hugepage support (and requirement) for less TLB misses
- Thread-to-core pinning
- NUMA support
Addressing Shortcomings of Existing DDoS Protection Software Using Software-Defined Networking

Lukas Iffländer, Stefan Geissler, Jürgen Walter, Lukas Beierlieb, Samuel Kounev
Python application

SYN Flood Defense

- Translates address and port from pipe into flow modification REST request

HTTP Flood Defense

- Monitor flow statistics
- Aggregate per IP address
- Requeue suspicious addresses
NICs and Drivers

Addressing Shortcomings of Existing DDoS Protection Software Using Software-Defined Networking

Lukas Iffländer, Stefan Geissler, Jürgen Walter, Lukas Beierlieb, Samuel Kounev
Kernel Modification Validation

echo -n -e '\x01\x23\x45\x67' > payload
hping3 --count 1 --syn --dest-port 80 --data 4 --file payload 192.168.0.10
### Kernel Modification Validation

<table>
<thead>
<tr>
<th>No.</th>
<th>Time</th>
<th>Source</th>
<th>Destination</th>
<th>Protocol</th>
<th>Length Info</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.0000000000</td>
<td>10.0.0.30</td>
<td>192.168.0.10</td>
<td>TCP</td>
<td>58 2489 -&gt; 80 [SYN] Seq=2081396469 Win=512 Len=4</td>
</tr>
<tr>
<td>2</td>
<td>0.000241677</td>
<td>192.168.0.10</td>
<td>10.0.0.30</td>
<td>TCP</td>
<td>60 80 -&gt; 2489 [SYN, ACK] Seq=19088743 Ack=2081396470</td>
</tr>
</tbody>
</table>

- Frame 1: 58 bytes on wire (464 bits), 58 bytes captured (464 bits) on interface 0
- Ethernet II, Src: 0a:76:c6:e9:8f:43 (0a:76:c6:e9:8f:43), Dst: DecObsol_00:00:00 (aa:00:00:00:00:00)
- Internet Protocol Version 4, Src: 10.0.0.30, Dst: 192.168.0.10

Source Port: 2489
- Destination Port: 80
  - [Stream index: 0]
  - [TCP Segment Len: 4]
  - Sequence number: 2081396469
  - [Next sequence number: 2081396474]
  - Acknowledgment number: 510359386
  - 0101 .... = Header Length: 20 bytes (5)
- Flags: 0x002 (SYN)
  - Window size value: 512
  - [Calculated window size: 512]
  - Checksum: 0xe9b0 [unverified]
  - [Checksum Status: Unverified]
  - Urgent pointer: 0

TCP payload (4 bytes):
```
0000: aa 00 00 00 00 00 a0 76 c6 e9 8f 43 08 00 45 00
0010: 00 2c f9 93 00 00 3f 06 b7 68 0a 00 00 1e c0 a8
0020: 00 0a 09 b9 00 50 7c 0f 96 f5 1e 6b 77 5a 50 02
0030: 02 00 e9 b0 00 00 01 23 45 67
```

---

Addressing Shortcomings of Existing DDoS Protection Software Using Software-Defined Networking

Lukas Iffländer, Stefan Geissler, Jürgen Walter, Lukas Beierlieb, Samuel Kounev
Kernel Modification Validation

Addressing Shortcomings of Existing DDoS Protection Software Using Software-Defined Networking

Lukas Iffländer, Stefan Geissler, Jürgen Walter, Lukas Beierlieb, Samuel Kounev
### THREADS Validation

<table>
<thead>
<tr>
<th>iface</th>
<th>(\Delta t [\mu s])</th>
<th>source</th>
<th>dest</th>
<th>flags</th>
<th>datalen [byte]</th>
</tr>
</thead>
<tbody>
<tr>
<td>client0</td>
<td>-</td>
<td>10.0.0.20 : 55510</td>
<td>192.168.0.10 : 80</td>
<td>SYN</td>
<td>0</td>
</tr>
<tr>
<td>client0</td>
<td>369</td>
<td>192.168.0.10 : 80</td>
<td>10.0.0.20 : 55510</td>
<td>SYN,ACK</td>
<td>0</td>
</tr>
<tr>
<td>client0</td>
<td>77</td>
<td>10.0.0.20 : 55510</td>
<td>192.168.0.10 : 80</td>
<td>ACK</td>
<td>0</td>
</tr>
<tr>
<td>client0</td>
<td>122</td>
<td>10.0.0.20 : 55510</td>
<td>192.168.0.10 : 80</td>
<td>ACK</td>
<td>193</td>
</tr>
<tr>
<td>serv0</td>
<td>11</td>
<td>10.0.0.20 : 55510</td>
<td>192.168.0.10 : 80</td>
<td>SYN</td>
<td>4</td>
</tr>
<tr>
<td>serv0</td>
<td>98</td>
<td>192.168.0.10 : 80</td>
<td>10.0.0.20 : 55510</td>
<td>SYN,ACK</td>
<td>0</td>
</tr>
<tr>
<td>serv0</td>
<td>1212</td>
<td>10.0.0.20 : 55510</td>
<td>192.168.0.10 : 80</td>
<td>ACK</td>
<td>0</td>
</tr>
<tr>
<td>client0</td>
<td>195971</td>
<td>10.0.0.20 : 55510</td>
<td>192.168.0.10 : 80</td>
<td>ACK</td>
<td>193</td>
</tr>
<tr>
<td>serv0</td>
<td>91</td>
<td>10.0.0.20 : 55510</td>
<td>192.168.0.10 : 80</td>
<td>ACK</td>
<td>193</td>
</tr>
</tbody>
</table>