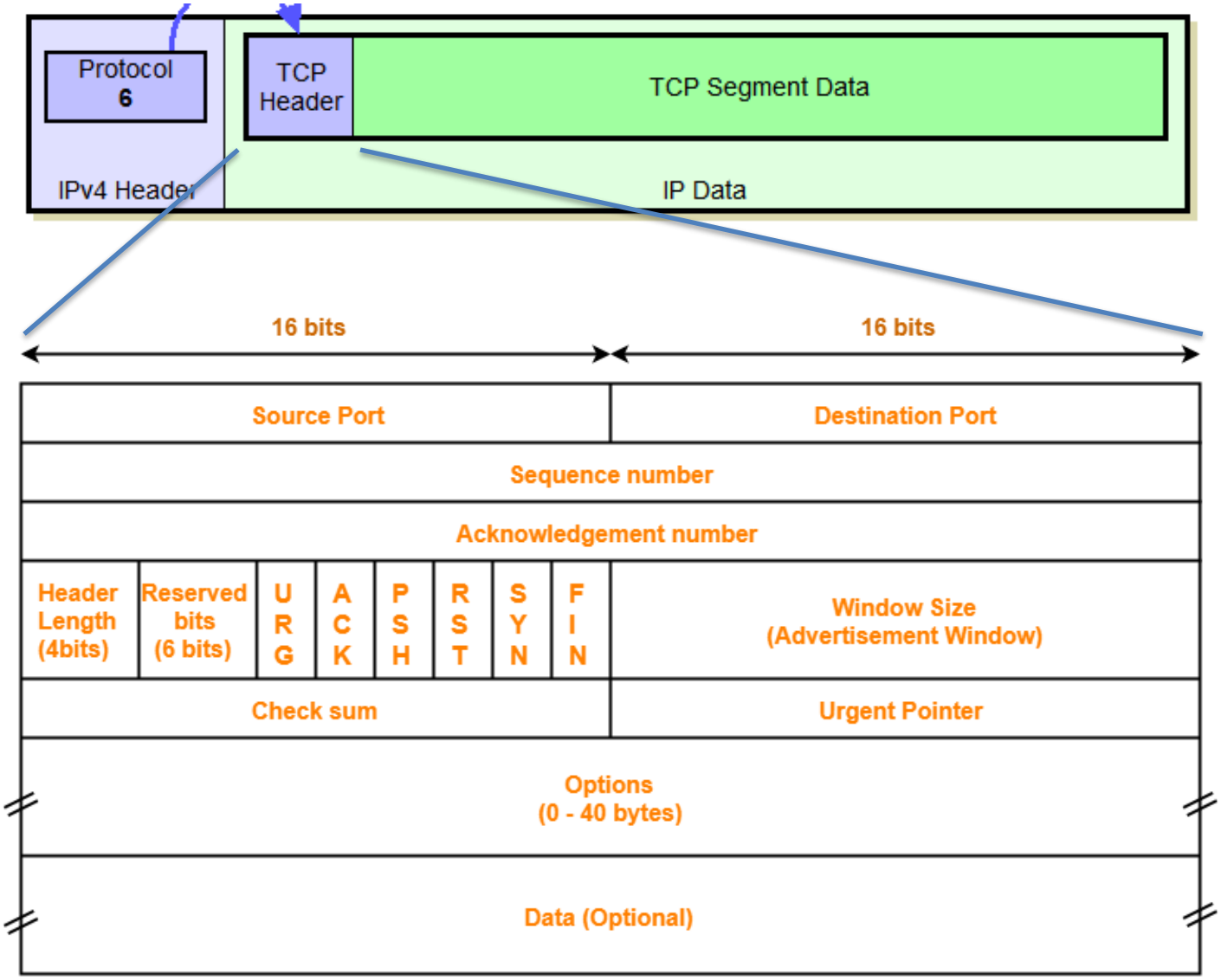
TCP Layer Security

Duan Haixin

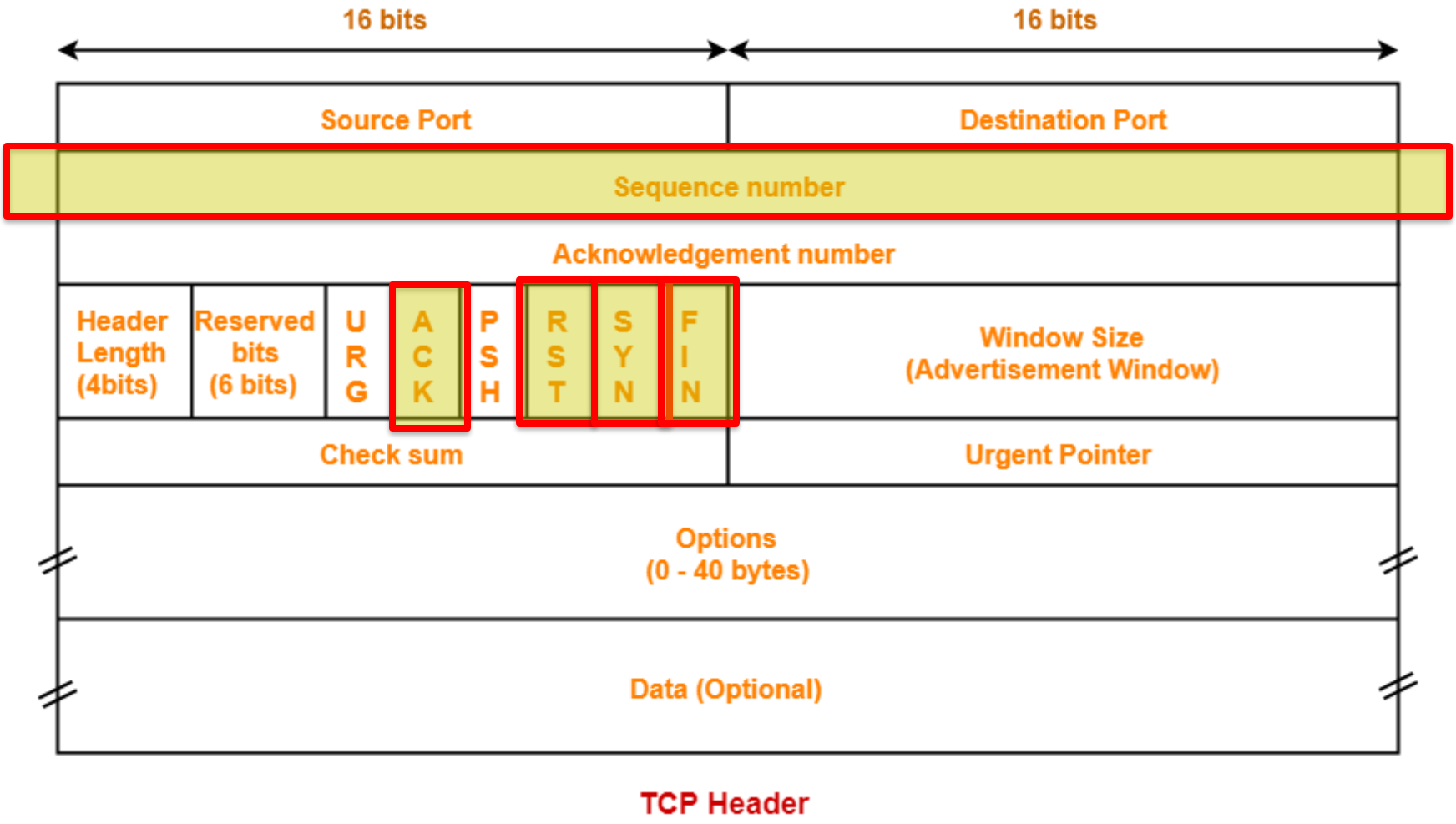
TCP Security

- TCB maintenance Problems
  - SYN Flood
  - Connection de-synchronize
- TCP Sequence Number Prediction
  - Reset, Hijack

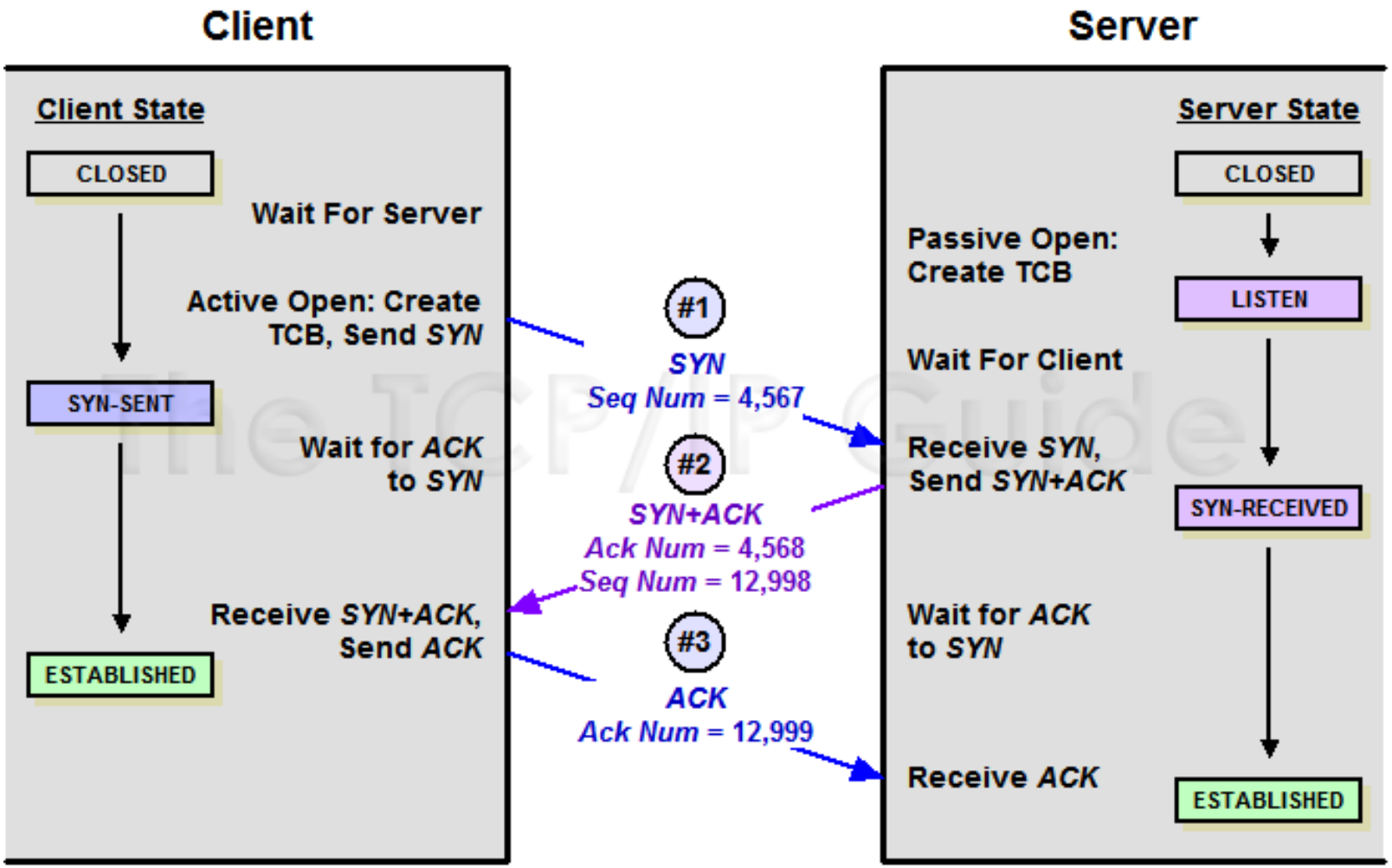

Protocol
6
TCP
Header
TCP Segment Data
IPv4 Header
IP Data
16 bits
16 bits
Source Port
Destination Port
Sequence number
Acknowledgement number
Header Length (4bits)
Reserved bits (6 bits)
URG
ACK
PSH
RST
SYN
FIN
Window Size (Advertisement Window)
Check sum
Urgent Pointer
Options (0 - 40 bytes)
Data (Optional)

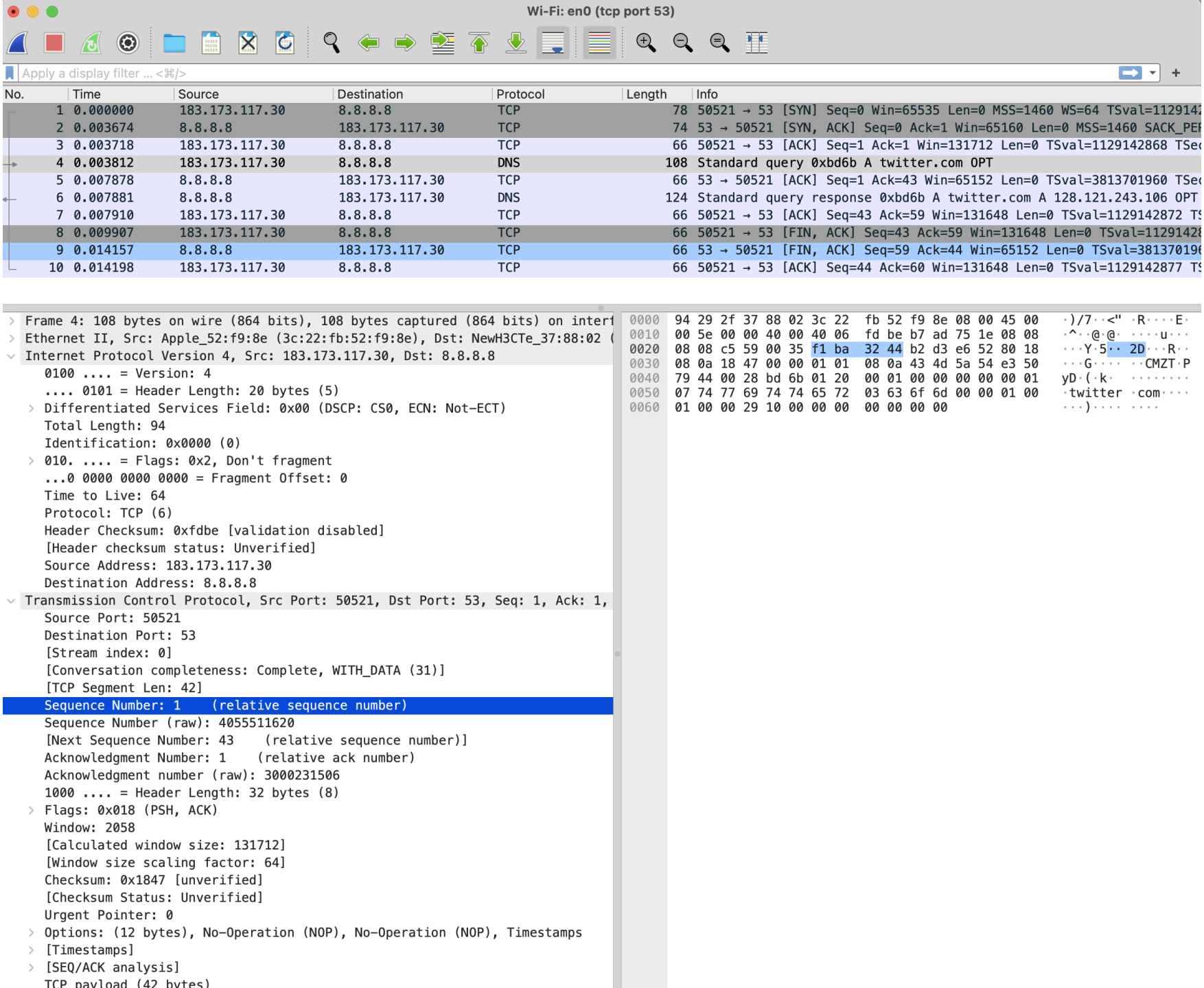
TCP Header

TCP Flags and Window Size

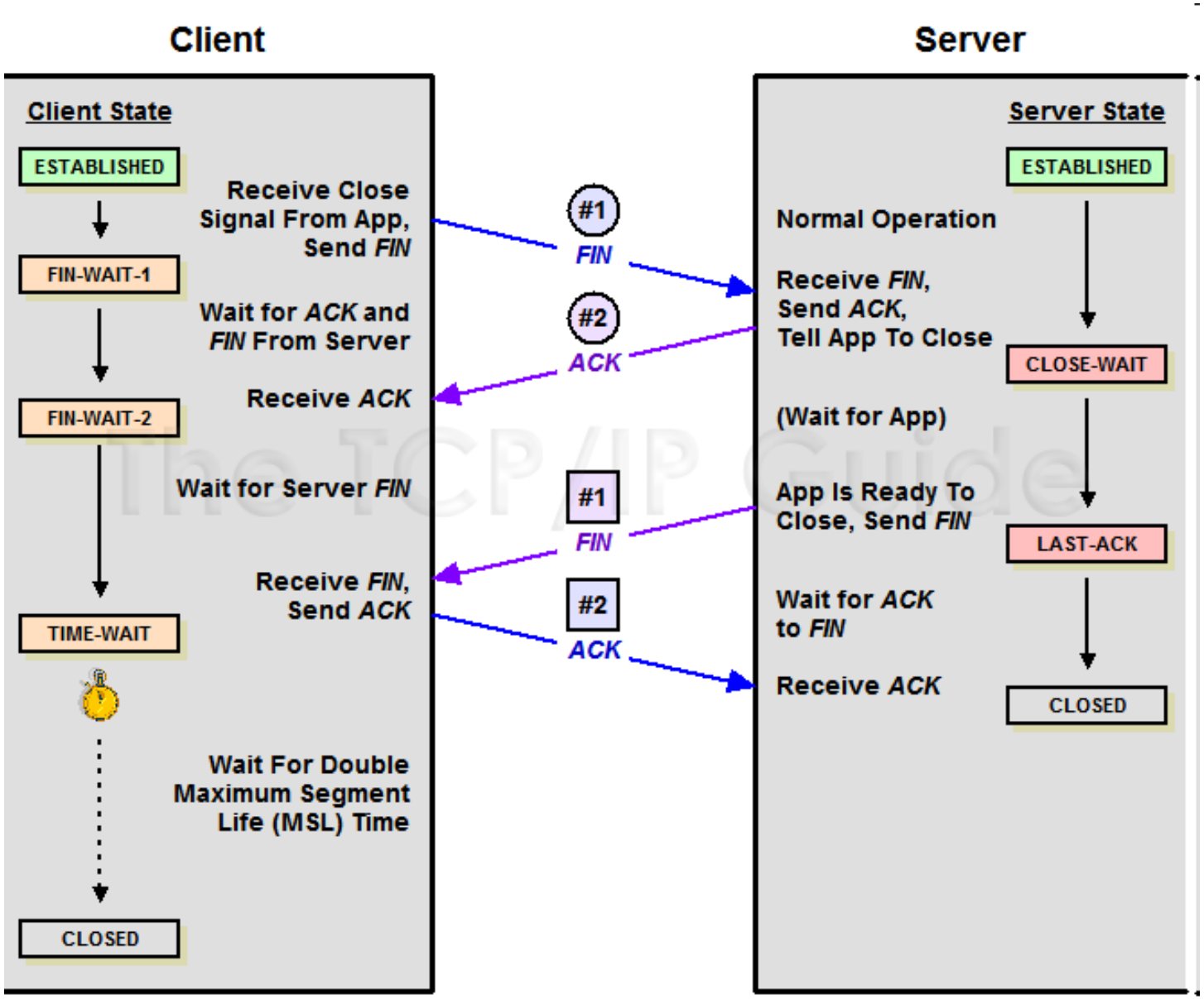


TCP Sequence Number Synchronization



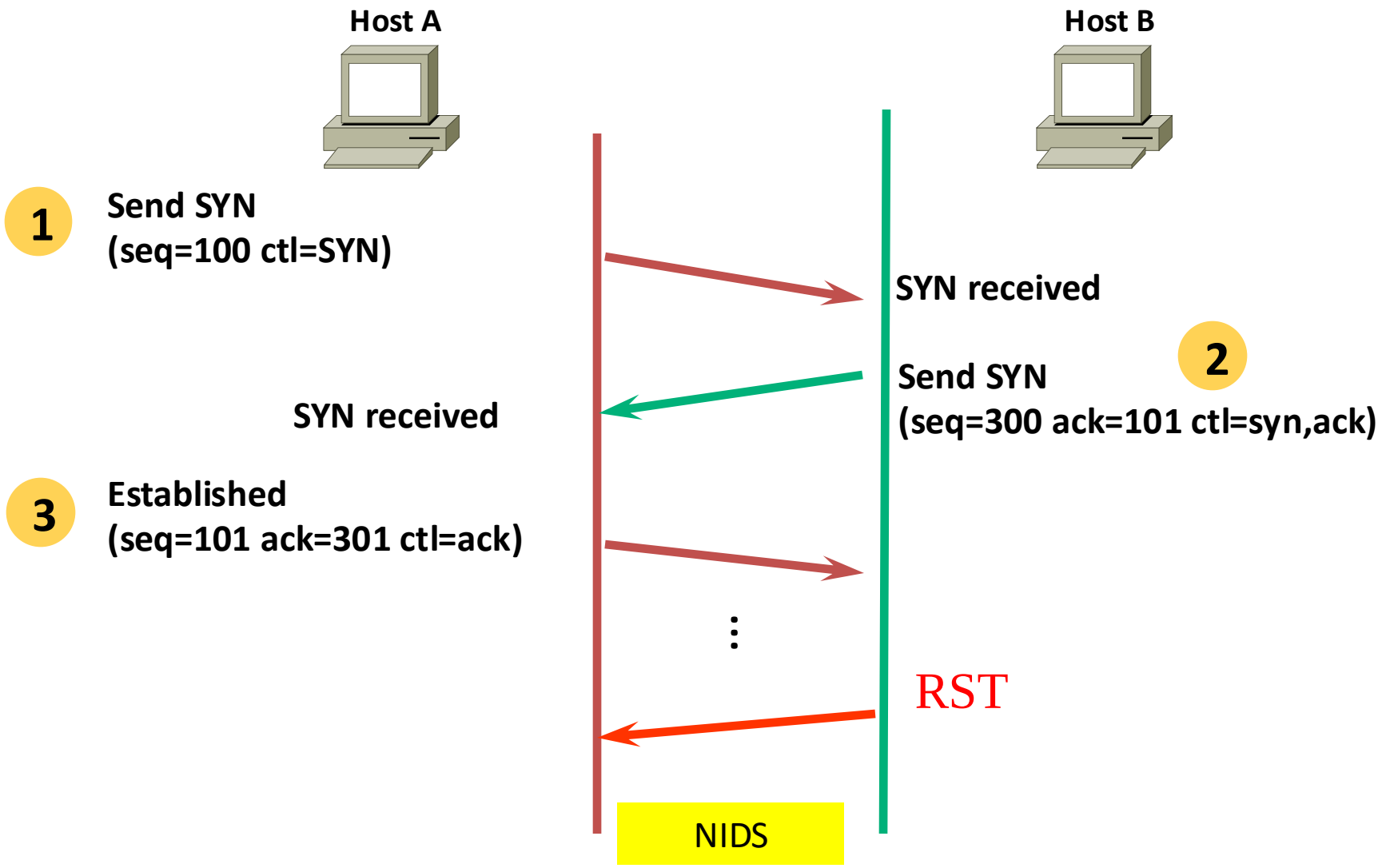

Wi-Fi: en0 (tcp port 53)
Apply a display filter ... <⌘/>
No. Time Source Destination Protocol Length Info
1 0.000000 183.173.117.30 8.8.8.8 TCP 78 50521 → 53 [SYN] Seq=0 Win=65535 Len=0 MSS=1460 WS=64 TSval=1129142
2 0.003674 8.8.8.8 183.173.117.30 TCP 74 53 → 50521 [SYN, ACK] Seq=0 Ack=1 Win=65160 Len=0 MSS=1460 SACK_PE
3 0.003718 183.173.117.30 8.8.8.8 TCP 66 50521 → 53 [ACK] Seq=1 Ack=1 Win=131712 Len=0 TSval=1129142868 TSe
4 0.003812 183.173.117.30 8.8.8.8 DNS 108 Standard query 0xbd6b A twitter.com OPT
5 0.007878 8.8.8.8 183.173.117.30 TCP 66 53 → 50521 [ACK] Seq=1 Ack=43 Win=65152 Len=0 TSval=3813701960 TSe
6 0.007881 8.8.8.8 183.173.117.30 DNS 124 Standard query response 0xbd6b A twitter.com A 128.121.243.106 OPT
7 0.007910 183.173.117.30 8.8.8.8 TCP 66 50521 → 53 [ACK] Seq=43 Ack=59 Win=131648 Len=0 TSval=1129142872 TS
8 0.009907 183.173.117.30 8.8.8.8 TCP 66 50521 → 53 [FIN, ACK] Seq=43 Ack=59 Win=131648 Len=0 TSval=1129142
9 0.014157 8.8.8.8 183.173.117.30 TCP 66 53 → 50521 [FIN, ACK] Seq=59 Ack=44 Win=65152 Len=0 TSval=381370196
10 0.014198 183.173.117.30 8.8.8.8 TCP 66 50521 → 53 [ACK] Seq=44 Ack=60 Win=131648 Len=0 TSval=1129142877 TS
Frame 4: 108 bytes on wire (864 bits), 108 bytes captured (864 bits) on interf
Ethernet II, Src: Apple_52:f9:8e (3c:22:fb:52:f9:8e), Dst: NewH3CTe_37:88:02 (
Internet Protocol Version 4, Src: 183.173.117.30, Dst: 8.8.8.8
0100 .... = Version: 4
.... 0101 = Header Length: 20 bytes (5)
Differentiated Services Field: 0x00 (DSCP: CS0, ECN: Not-ECT)
Total Length: 94
Identification: 0x0000 (0)
010. .... = Flags: 0x2, Don't fragment
...0 0000 0000 0000 = Fragment Offset: 0
Time to Live: 64
Protocol: TCP (6)
Header Checksum: 0xfdbe [validation disabled]
[Header checksum status: Unverified]
Source Address: 183.173.117.30
Destination Address: 8.8.8.8
Transmission Control Protocol, Src Port: 50521, Dst Port: 53, Seq: 1, Ack: 1,
Source Port: 50521
Destination Port: 53
[Stream index: 0]
[Conversation completeness: Complete, WITH_DATA (31)]
[TCP Segment Len: 42]
Sequence Number: 1 (relative sequence number)
Sequence Number (raw): 4055511620
[Next Sequence Number: 43 (relative sequence number)]
Acknowledgment Number: 1 (relative ack number)
Acknowledgment number (raw): 3000231506
1000 .... = Header Length: 32 bytes (8)
Flags: 0x018 (PSH, ACK)
Window: 2058
[Calculated window size: 131712]
[Window size scaling factor: 64]
Checksum: 0x1847 [unverified]
[Checksum Status: Unverified]
Urgent Pointer: 0
Options: (12 bytes), No-Operation (NOP), No-Operation (NOP), Timestamps
[Timestamps]
[SEQ/ACK analysis]
TCP payload (42 bytes)
0000 94 29 2f 37 88 02 3c 22 fb 52 f9 8e 08 00 45 00 ·)/7··<" ·R····E·
0010 00 5e 00 00 40 00 40 06 fd be b7 ad 75 1e 08 08 ·^··@·@· ····u···
0020 08 08 c5 59 00 35 f1 ba 32 44 b2 d3 e6 52 80 18 ···Y·5·· 2D···R··
0030 08 0a 18 47 00 00 01 01 08 0a 43 4d 5a 54 e3 50 ···G···· ··CMZT·P
0040 79 44 00 28 bd 6b 01 20 00 01 00 00 00 00 00 01 yD·(·k· ········
0050 07 74 77 69 74 74 65 72 03 63 6f 6d 00 00 01 00 ·twitter ·com····
0060 01 00 00 29 10 00 00 00 00 00 00 00 ···)···· ····

TCP Connection Termination



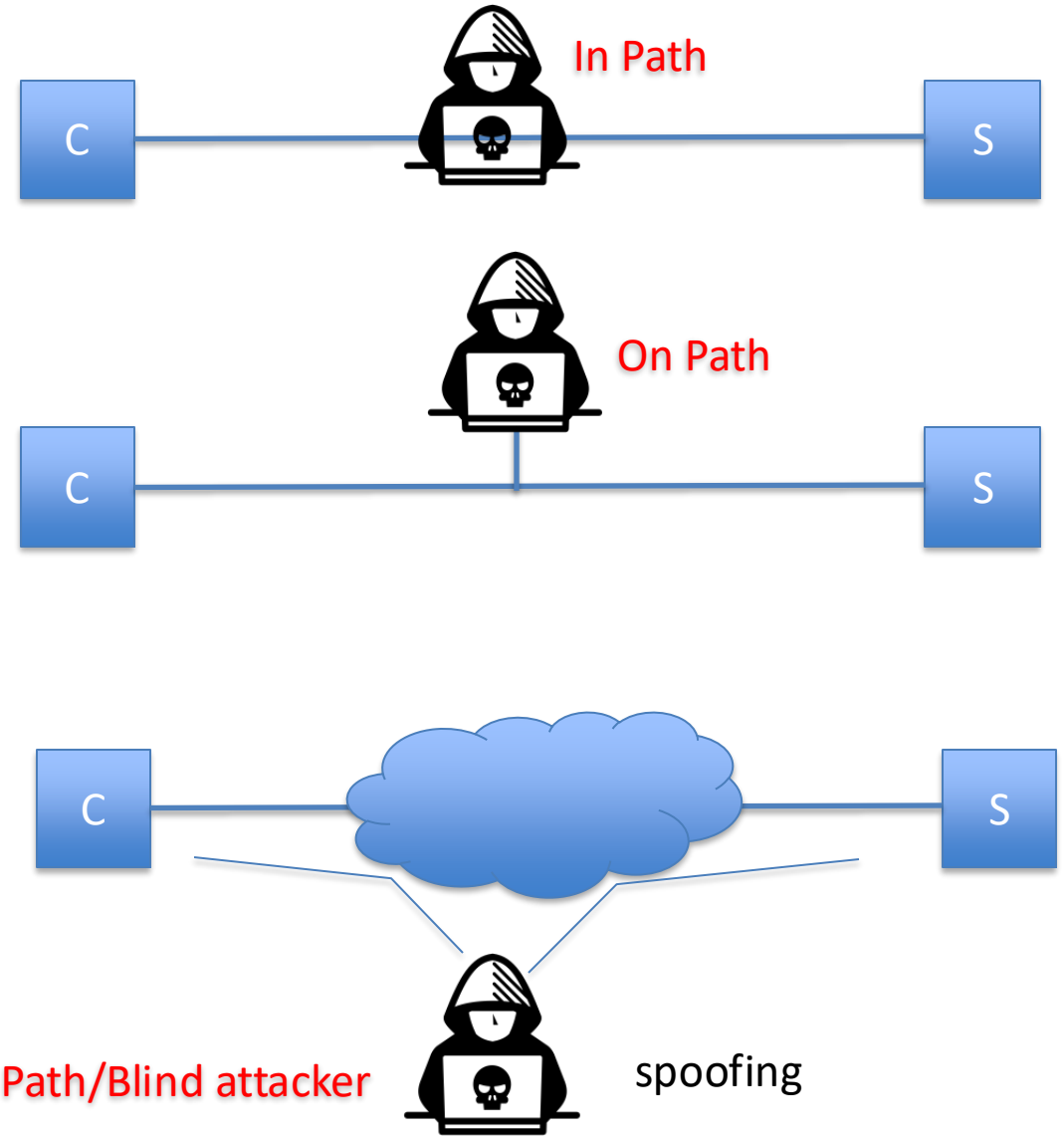
http://www.tcpipguide.com/free/t_TCPConnectionTermination-2.htm

TCP Reset

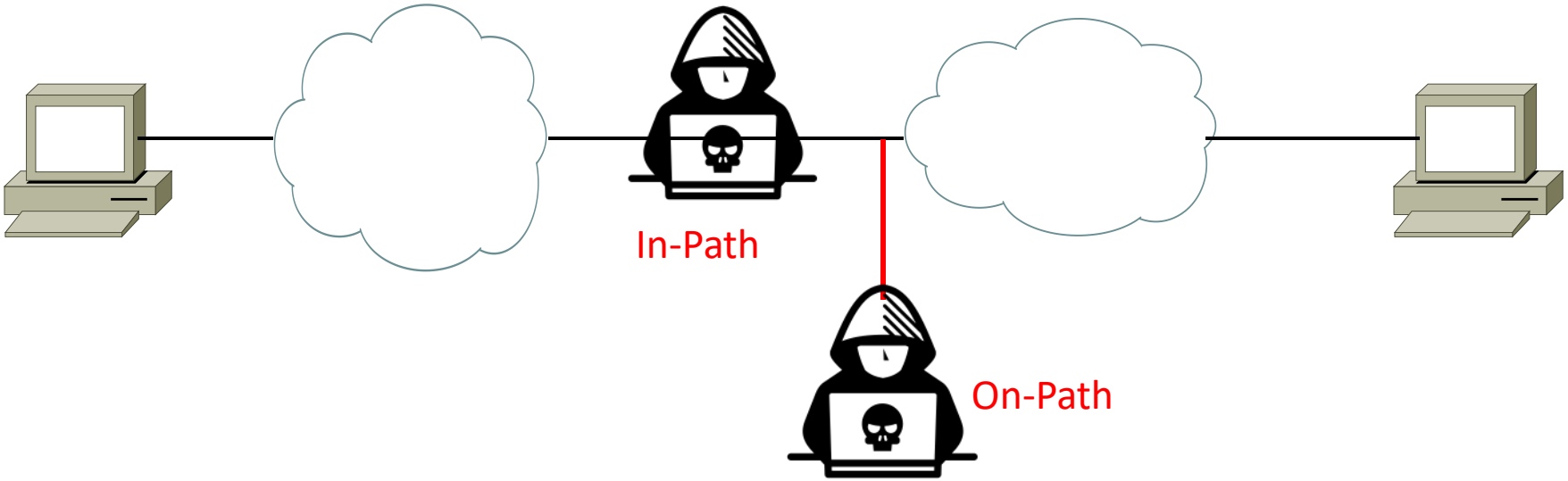

Host A
Host B
1
Send SYN
(seq=100 ctl=SYN)
SYN received
2
Send SYN
(seq=300 ack=101 ctl=syn,ack)
SYN received
3
Established
(seq=101 ack=301 ctl=ack)
RST
NIDS

Location of attacker

- In Path
- On Path
- Off Path

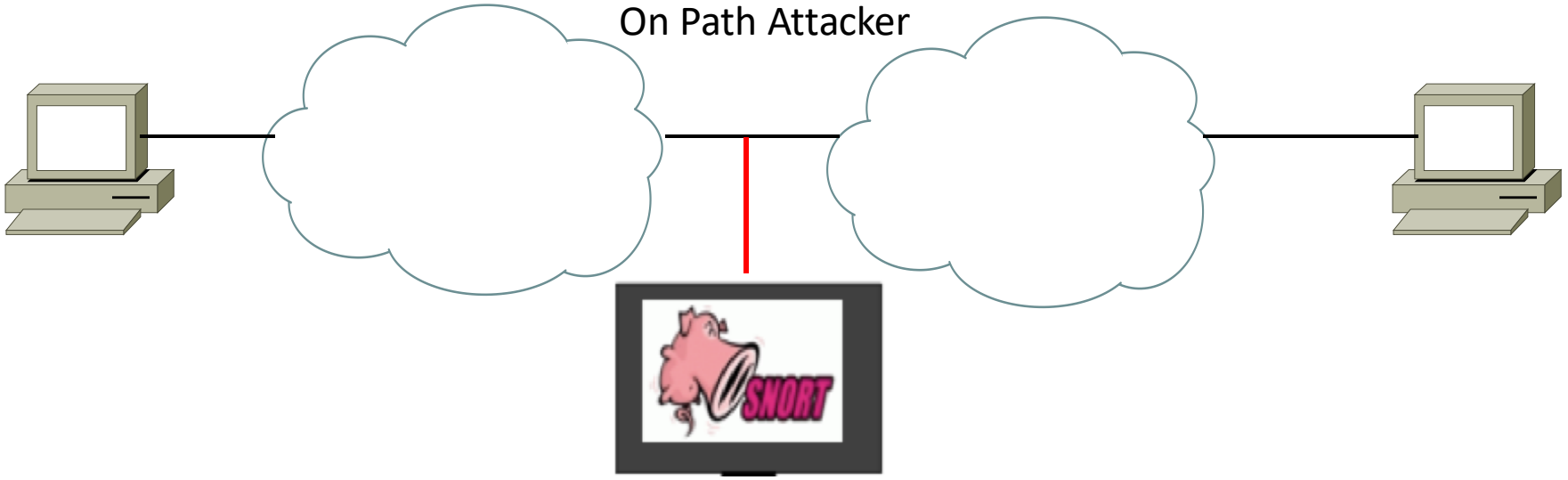


In-Path or On-Path Attack



- In Path: attacker can see any packet, and drop or modify them(or inject spoofed packets)
- On Path: attacker can see any packet, or inject spoofed packets(cannot drop or modify)

Network Intrusion Detection System(NIDS)



NIDS knows: sIP, sPort, dIP, dPort, Seq, ACK

NIDS can't : drop packet between client and server

Case study of NIDS

Reverse Engineering the NIDS

1. Ignoring the Great Firewall of China, PETS workshop 2006
2. Conceptdoppler: A weather tracker for internet censorship *14th ACM Conference on Computer and Communications Security, 2007*
3. "Empirical study of a national-scale distributed intrusion detection system: Backbone-level filtering of html responses in china," *Distributed Computing Systems* 2010.
4. West Chamber Project. (2010) https://code.google.com/p/scholarzhang/
5. Internet censorship in china: where does the filtering occur?, *PAM* 2011.
6. How the Great Firewall of China is Blocking Tor. FOCI '12
7. Towards Illuminating a Censorship Monitor's Model to Facilitate Evasion, Proc. USENIX Workshop FOCI, 2013
8. Anonymous, Towards a Comprehensive Picture of the Great Firewall's DNS Censorship, FOCI 2014
9. Examining How the Great Firewall Discovers Hidden Circumvention Servers. IMC '15.
10. SoK: Towards Grounding Censorship Circumvention in Empiricism. SP'16
11. Your state is not mine- a closer look at evading stateful Internet Censorship, IMC 2017

Ignoring the Great Firewall of China

Richard Clayton, Steven J. Murdoch, and Robert N. M. Watson
University of Cambridge, Computer Laboratory,

Abstract.
The so-called “Great Firewall of China” operates, in part, by inspecting TCP packets for keywords that are to be blocked. If the keyword is present, TCP reset packets (viz: with the RST flag set) are sent to both endpoints of the connection, which then close. However, because the original packets are passed through the firewall unscathed, if the endpoints completely ignore the firewall’s resets, then the connection will proceed unhindered.

Once one connection has been blocked, the firewall makes further easy-to-evade attempts to block further connections from the same machine. This latter behavior can be leveraged into a denial-of-service attack on third-party machines.

Privacy Enhancing Technologies workshop 2006.

TCP Reset attack from IDS or other systems

```
cam(53382)   → china(http) [SYN]
china(http)  → cam(53382)  [SYN, ACK]
cam(53382)   → china(http) [ACK]
cam(53382)   → china(http) GET / HTTP/1.0<cr><lf><cr><lf>
china(http)  → cam(53382)  HTTP/1.1 200 OK (text/html)<cr><lf> etc...
china(http)  → cam(53382)  ...more of the web page
cam(53382)   → china(http) [ACK]
               ...and so on until the page was complete
```

Normal Connection

TCP Reset attack from IDS or other systems

```
cam(53382)   → china(http) [SYN]
china(http)  → cam(53382)  [SYN, ACK]
cam(53382)   → china(http) [ACK]
cam(53382)   → china(http) GET / HTTP/1.0<cr><lf><cr><lf>
china(http)  → cam(53382)  HTTP/1.1 200 OK (text/html)<cr><lf> etc...
china(http)  → cam(53382)  ...more of the web page
cam(53382)   → china(http) [ACK]
               ...and so on until the page was complete
```

Normal Connection

We then issued a request which included a small fragment of text that we expected to cause the connection to be blocked, and this promptly occurred:

```
cam(54190)   → china(http) [SYN]
china(http)  → cam(54190)  [SYN, ACK] TTL=39
cam(54190)   → china(http) [ACK]
cam(54190)   → china(http) GET /?falun HTTP/1.0<cr><lf><cr><lf>
china(http)  → cam(54190)  [RST] TTL=47, seq=1, ack=1
china(http)  → cam(54190)  [RST] TTL=47, seq=1461, ack=1
china(http)  → cam(54190)  [RST] TTL=47, seq=4381, ack=1
china(http)  → cam(54190)  HTTP/1.1 200 OK (text/html)<cr><lf> etc...
cam(54190)   → china(http) [RST] TTL=64, seq=25, ack zeroed
china(http)  → cam(54190)  ...more of the web page
cam(54190)   → china(http) [RST] TTL=64, seq=25, ack zeroed
china(http)  → cam(54190)  [RST] TTL=47, seq=2921, ack=25
```

Sensitive keyword

TRANSMISSION CONTROL PROTOCOL, 1981
https://datatracker.ietf.org/doc/html/rfc793

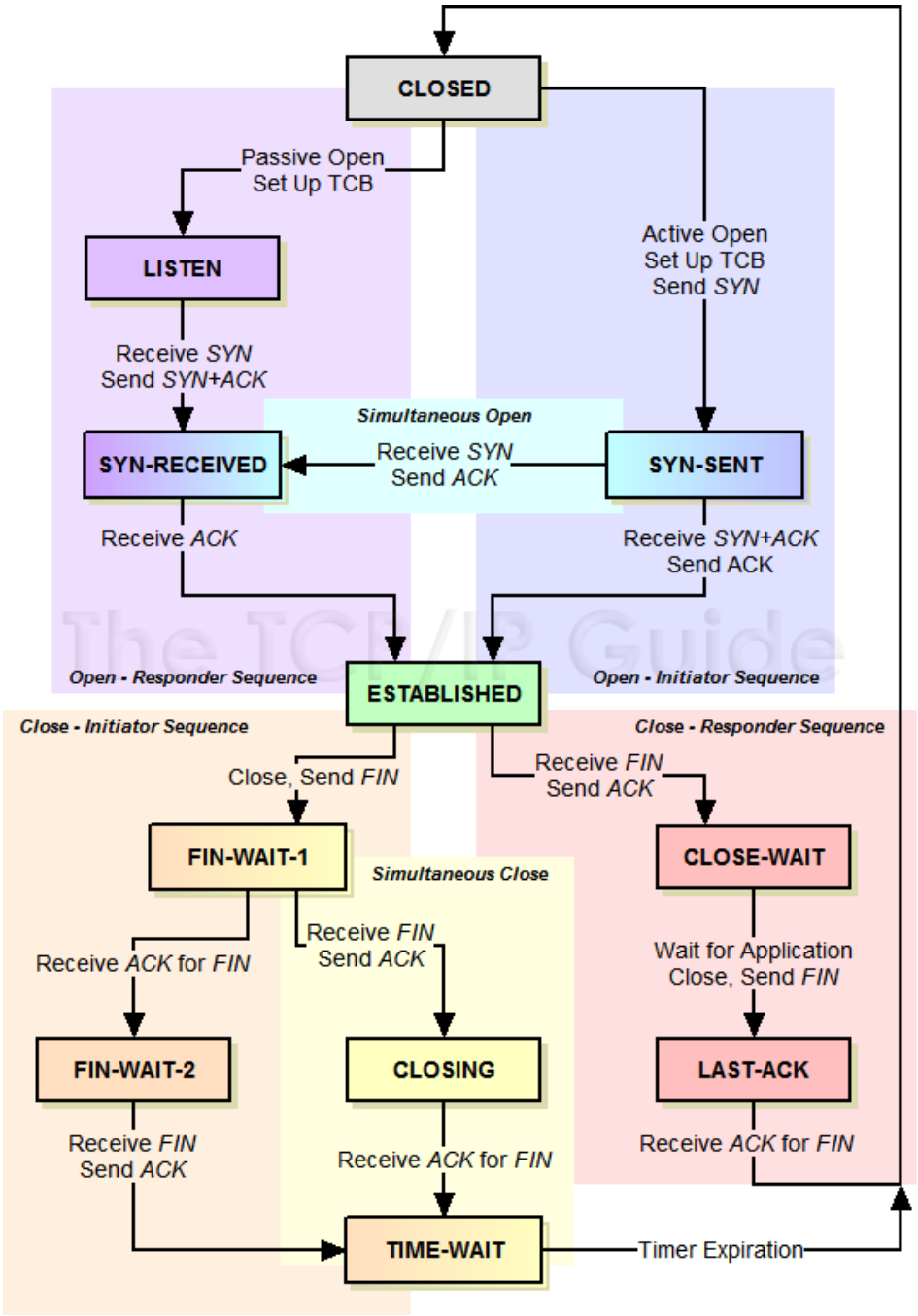
3. If the connection is in a synchronized state (ESTABLISHED, FIN-WAIT-1, FIN-WAIT-2, CLOSE-WAIT, CLOSING, LAST-ACK, TIME-WAIT), any unacceptable segment (out of window sequence number or unacceptible acknowledgment number) must elicit only an empty acknowledgment segment containing the current send-sequence number and an acknowledgment indicating the next sequence number expected to be received, and the connection remains in the same state.

If an incoming segment has a security level, or compartment, or precedence which does not exactly match the level, and compartment, and precedence requested for the connection,a reset is sent and connection goes to the CLOSED state. The reset takes its sequence number from the ACK field of the incoming segment.

Reset Processing

In all states except SYN-SENT, all reset (RST) segments are validated by checking their SEQ-fields. A reset is valid if its sequence number is in the window. In the SYN-SENT state (a RST received in response to an initial SYN), the RST is acceptable if the ACK field acknowledges the SYN.

The receiver of a RST first validates it, then changes state. If the receiver was in the LISTEN state, it ignores it. If the receiver was in SYN-RECEIVED state and had previously been in the LISTEN state, then the receiver returns to the LISTEN state, otherwise the receiver aborts the connection and goes to the CLOSED state. If the receiver was in any other state, it aborts the connection and advises the user and goes to the CLOSED state.


CLOSED
Passive Open
Set Up TCB
Active Open
Set Up TCB
Send SYN
LISTEN
Receive SYN
Send SYN+ACK
Simultaneous Open
SYN-RECEIVED
Receive SYN
Send ACK
SYN-SENT
Receive ACK
Receive SYN+ACK
Send ACK
The TCP/IP Guide
Open - Responder Sequence
ESTABLISHED
Open - Initiator Sequence
Close - Initiator Sequence
Close - Responder Sequence
Close, Send FIN
Receive FIN
Send ACK
FIN-WAIT-1
Simultaneous Close
CLOSE-WAIT
Receive ACK for FIN
Receive FIN
Send ACK
Wait for Application
Close, Send FIN
FIN-WAIT-2
CLOSING
LAST-ACK
Receive FIN
Send ACK
Receive ACK for FIN
Receive ACK for FIN
TIME-WAIT
Timer Expiration

TCP Reset attack from NIDS or other systems

```
cam(53382)   → china(http) [SYN]
china(http)  → cam(53382)  [SYN, ACK]
cam(53382)   → china(http) [ACK]
cam(53382)   → china(http) GET / HTTP/1.0<cr><lf><cr><lf>
china(http)  → cam(53382)  HTTP/1.1 200 OK (text/html)<cr><lf> etc...
china(http)  → cam(53382)  ...more of the web page
cam(53382)   → china(http) [ACK]
               ...and so on until the page was complete
```

Normal Connection

We then issued a request which included a small fragment of text that we expected to cause the connection to be blocked, and this promptly occurred:

```
cam(54190)   → china(http) [SYN]
china(http)  → cam(54190)  [SYN, ACK] TTL=39
cam(54190)   → china(http) [ACK]
cam(54190)   → china(http) GET /?falun HTTP/1.0<cr><lf><cr><lf>
china(http)  → cam(54190)  [RST] TTL=47, seq=1, ack=1
china(http)  → cam(54190)  [RST] TTL=47, seq=1461, ack=1
china(http)  → cam(54190)  [RST] TTL=47, seq=4381, ack=1
china(http)  → cam(54190)  HTTP/1.1 200 OK (text/html)<cr><lf> etc...
cam(54190)   → china(http) [RST] TTL=64, seq=25, ack zeroed
china(http)  → cam(54190)  ...more of the web page
cam(54190)   → china(http) [RST] TTL=64, seq=25, ack zeroed
china(http)  → cam(54190)  [RST] TTL=47, seq=2921, ack=25
```

Sensitive keyword

TCP Reset attack from IDS or other systems

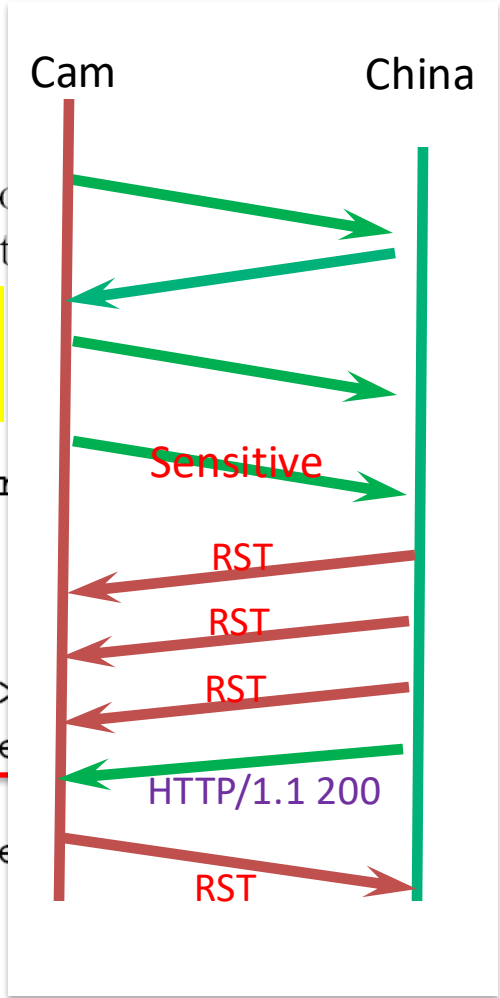
Normal Connection

```
cam(53382)   → china(http) [SYN]
china(http)  → cam(53382)  [SYN, ACK]
cam(53382)   → china(http) [ACK]
cam(53382)   → china(http) GET / HTTP/1.0<cr><lf><cr><lf>
china(http)  → cam(53382)  HTTP/1.1 200 OK (text/html)<cr><lf> etc...
china(http)  → cam(53382)  ...more of the web page
cam(53382)   → china(http) [ACK]
               ...and so on until the page was complete
```

We then issued a request which included a small fragment
expected to cause the connection to be blocked, and this prompt

Sensitive keyword

```
cam(54190)   → china(http) [SYN]
china(http)  → cam(54190)  [SYN, ACK] TTL=39
cam(54190)   → china(http) [ACK]
cam(54190)   → china(http) GET /?falun HTTP/1.0<cr><lf><cr
china(http)  → cam(54190)  [RST] TTL=47, seq=1, ack=1
china(http)  → cam(54190)  [RST] TTL=47, seq=1461, ack=1
china(http)  → cam(54190)  [RST] TTL=47, seq=4381, ack=1
china(http)  → cam(54190)  HTTP/1.1 200 OK (text/html)<cr>
cam(54190)   → china(http) [RST] TTL=64, seq=25, ack zeroe
china(http)  → cam(54190)  ...more of the web page
cam(54190)   → china(http) [RST] TTL=64, seq=25, ack zeroe
china(http)  → cam(54190)  [RST] TTL=47, seq=2921, ack=25
```



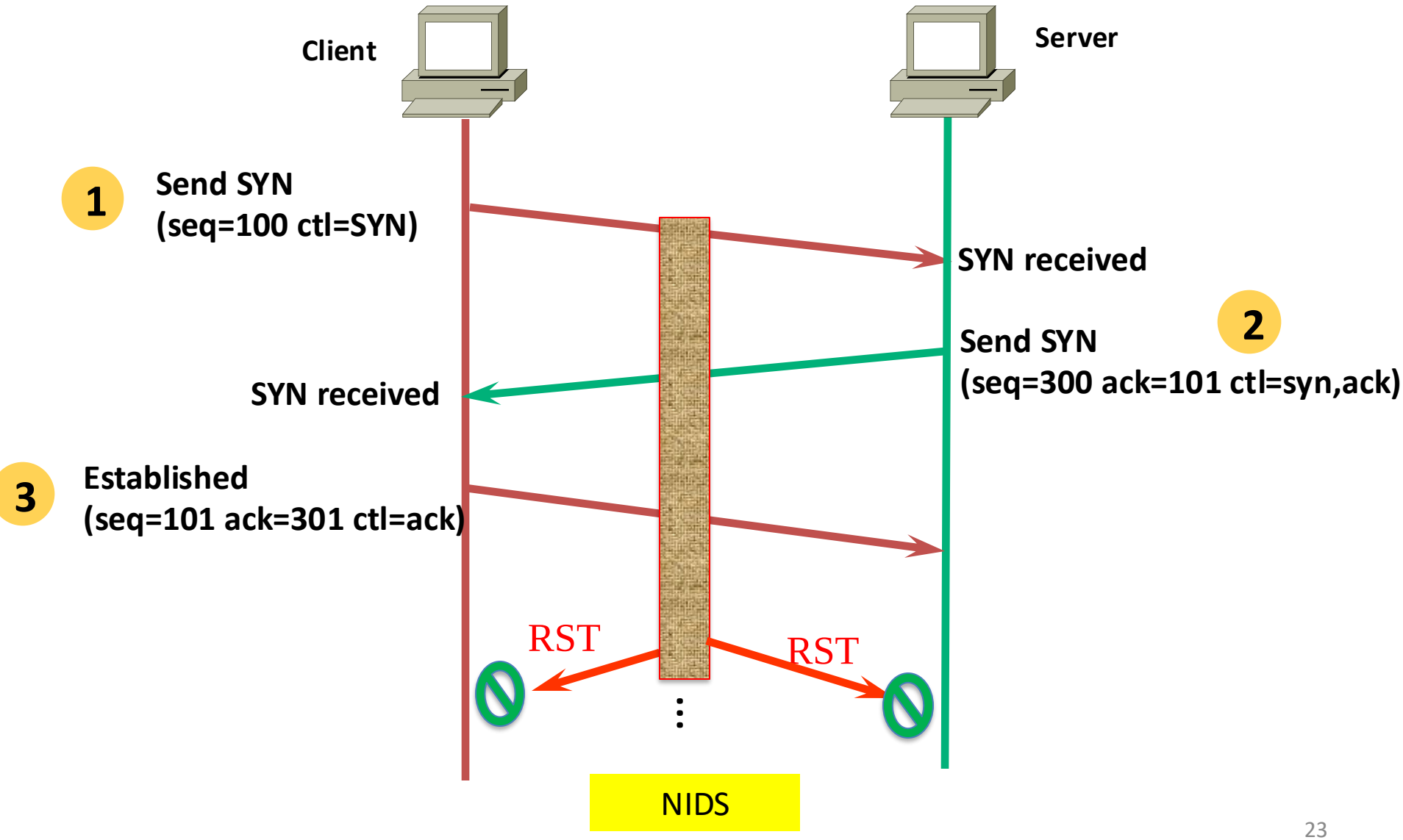
Ignoring RST packet

Linux we installed `iptables` and gave the command:

```
iptables -A INPUT -p tcp --tcp-flags RST RST -j DROP
```

```
cam(55817)  → china(http) [SYN]
china(http) → cam(55817)  [SYN, ACK] TTL=41
cam(55817)  → china(http) [ACK]
cam(55817)  → china(http) GET /?falun HTTP/1.0<cr><lf><cr><lf>
china(http) → cam(55817)  [RST] TTL=49, seq=1
china(http) → cam(55817)  [RST] TTL=49, seq=1
china(http) → cam(55817)  [RST] TTL=49, seq=1
china(http) → cam(55817)  HTTP/1.1 200 OK (text/html)<cr><lf> etc
china(http) → cam(55817)  ... more of the web page
cam(55817)  → china(http) [ACK] seq=25, ack=2921
china(http) → cam(55817)  ... more of the web page
china(http) → cam(55817)  [RST] TTL=49, seq=1461
china(http) → cam(55817)  [RST] TTL=49, seq=2921
china(http) → cam(55817)  [RST] TTL=49, seq=4381
cam(55817)  → china(http) [ACK] seq=25, ack=4381
china(http) → cam(55817)  [RST] TTL=49, seq=2921
china(http) → cam(55817)  ... more of the web page
china(http) → cam(55817)  ... more of the web page
cam(55817)  → china(http) [ACK] seq=25, ack=7301
china(http) → cam(55817)  [RST] TTL=49, seq=5841
```

Ignoring RST packet



Ignoring RST packet

Linux we installed `iptables` and gave the command:

```
iptables -A INPUT -p tcp --tcp-flags RST RST -j DROP
```

```
cam(55817)  → china(http) [SYN]
china(http) → cam(55817)  [SYN, ACK] TTL=41
cam(55817)  → china(http) [ACK]
cam(55817)  → china(http) GET /?falun HTTP/1.0<cr><lf><cr><lf>
china(http) → cam(55817)  [RST] TTL=49, seq=1
china(http) → cam(55817)  [RST] TTL=49, seq=1
china(http) → cam(55817)  [RST] TTL=49, seq=1
china(http) → cam(55817)  HTTP/1.1 200 OK (text/html)<cr><lf> etc
china(http) → cam(55817)  ...more of the web page
cam(55817)  → china(http) [ACK] seq=25, ack=2921
china(http) → cam(55817)  ...more of the web page
china(http) → cam(55817)  [RST] TTL=49, seq=1461
```

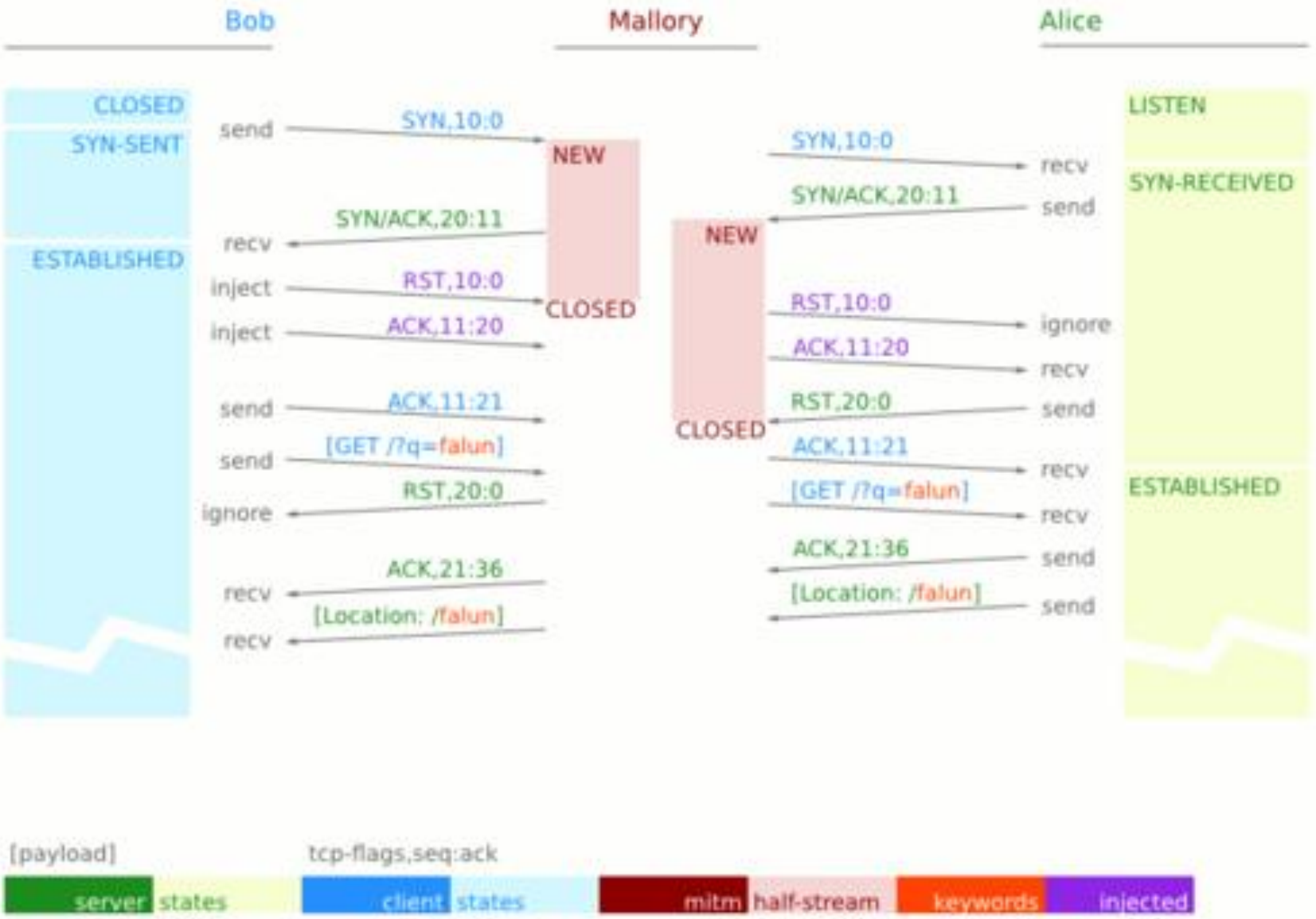
然而， 这需要两端的主机都过滤掉RST。

是否可能只在通信的一端过滤RST，就能让TCP连接继续呢？

```
china(http) → cam(55817)  ...more of the web page
cam(55817)  → china(http) [ACK] seq=25, ack=7301
china(http) → cam(55817)  [RST] TTL=49, seq=5841
```

张某工作原理示意图

2008-2010

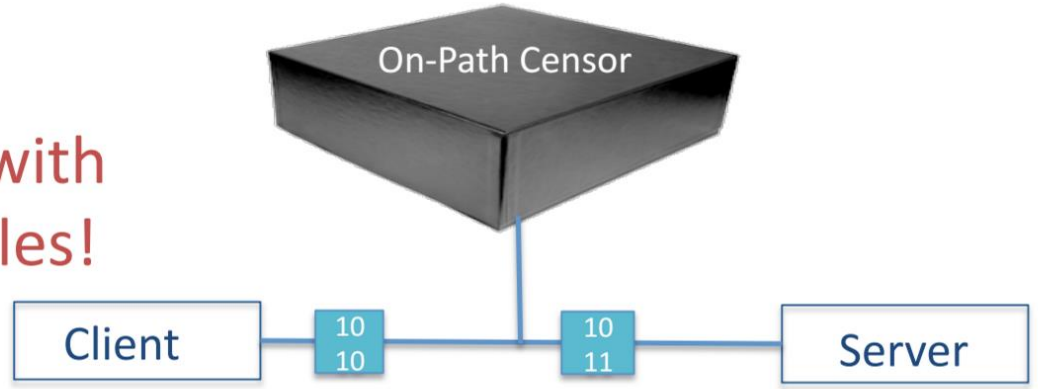


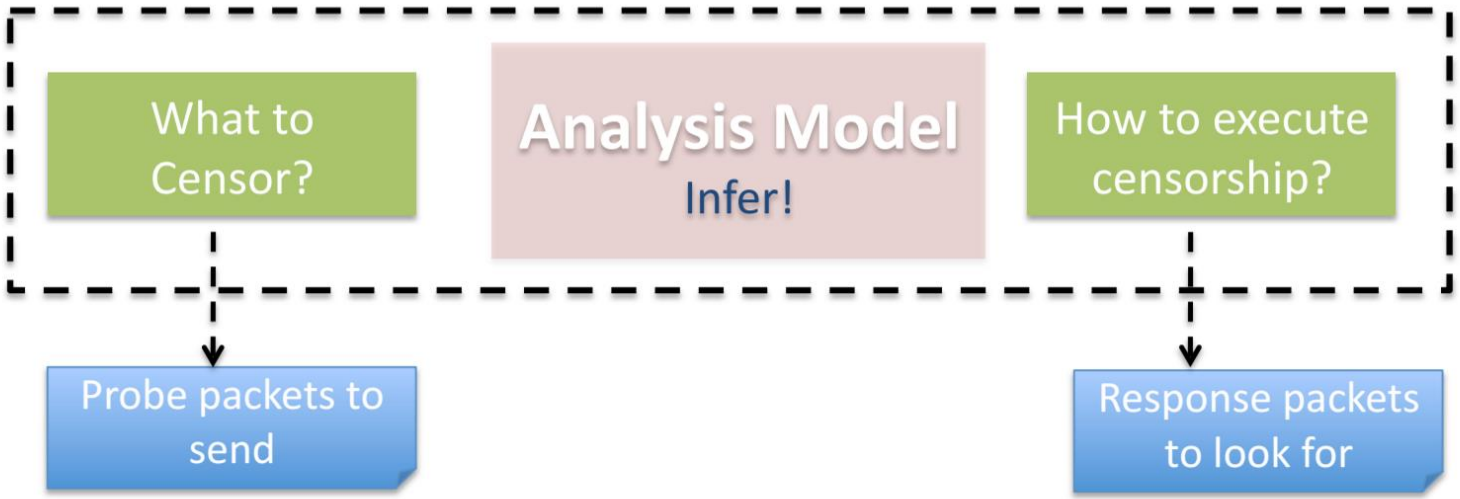
http://code.google.com/p/scholarzhang/

- 用伪造的RST和ACK干扰 NIDS的状态
- 有没有更系统化的方法，发现更多的逃逸技术呢？

Probing a Censor to infer model

A censor is a black-box, but with a few observables!





Towards Illuminating a Censorship Monitor's Model to Facilitate Evasion (FOCI 2013)

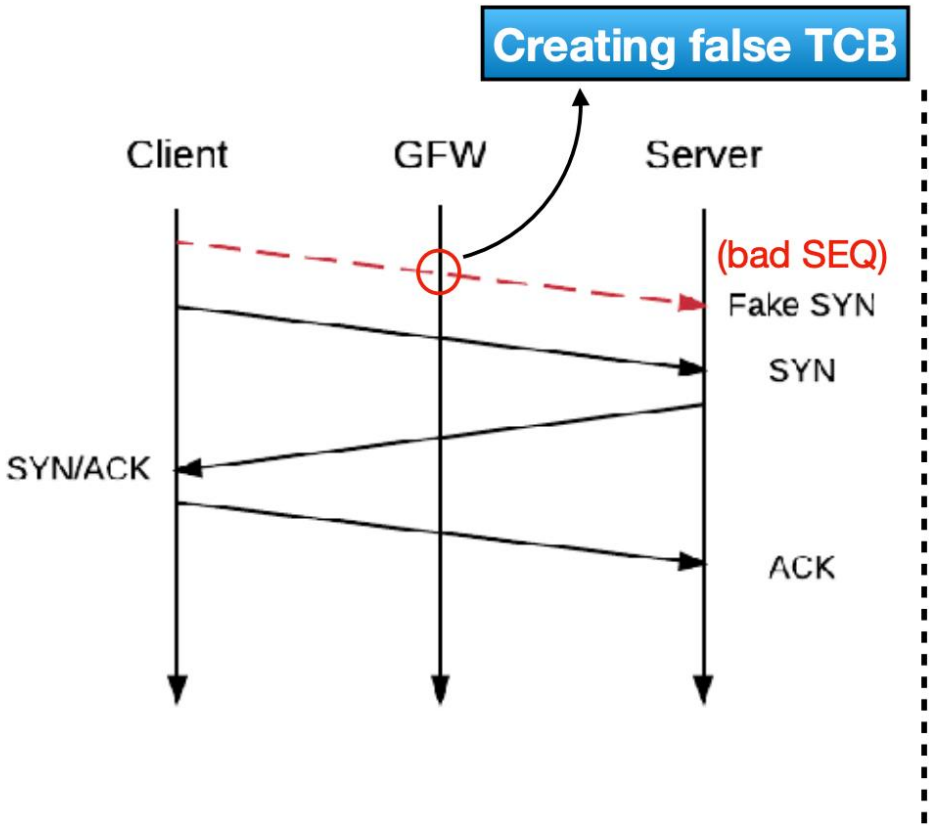
Model Elements to Probe

1. TCB Creation
2. Reassembly
3. State Management
4. TCB Teardown
5. Protocol Message Interpretation

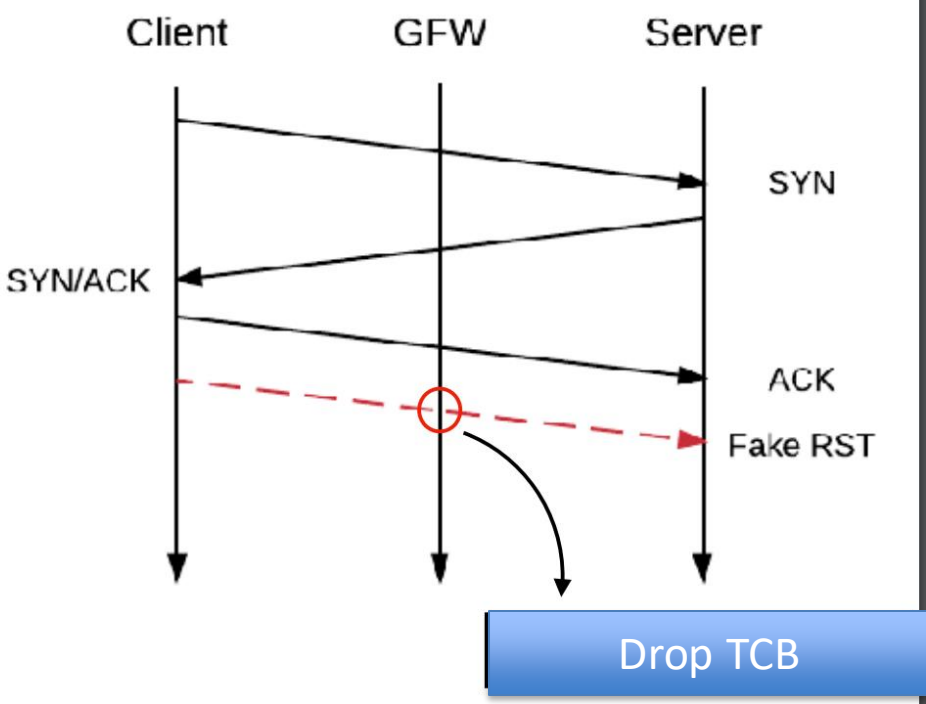
(Both network and higher layers)

https://www.usenix.org/conference/foci13/workshop-program/presentation/khattak

https://github.com/sheharbano/GFWProber



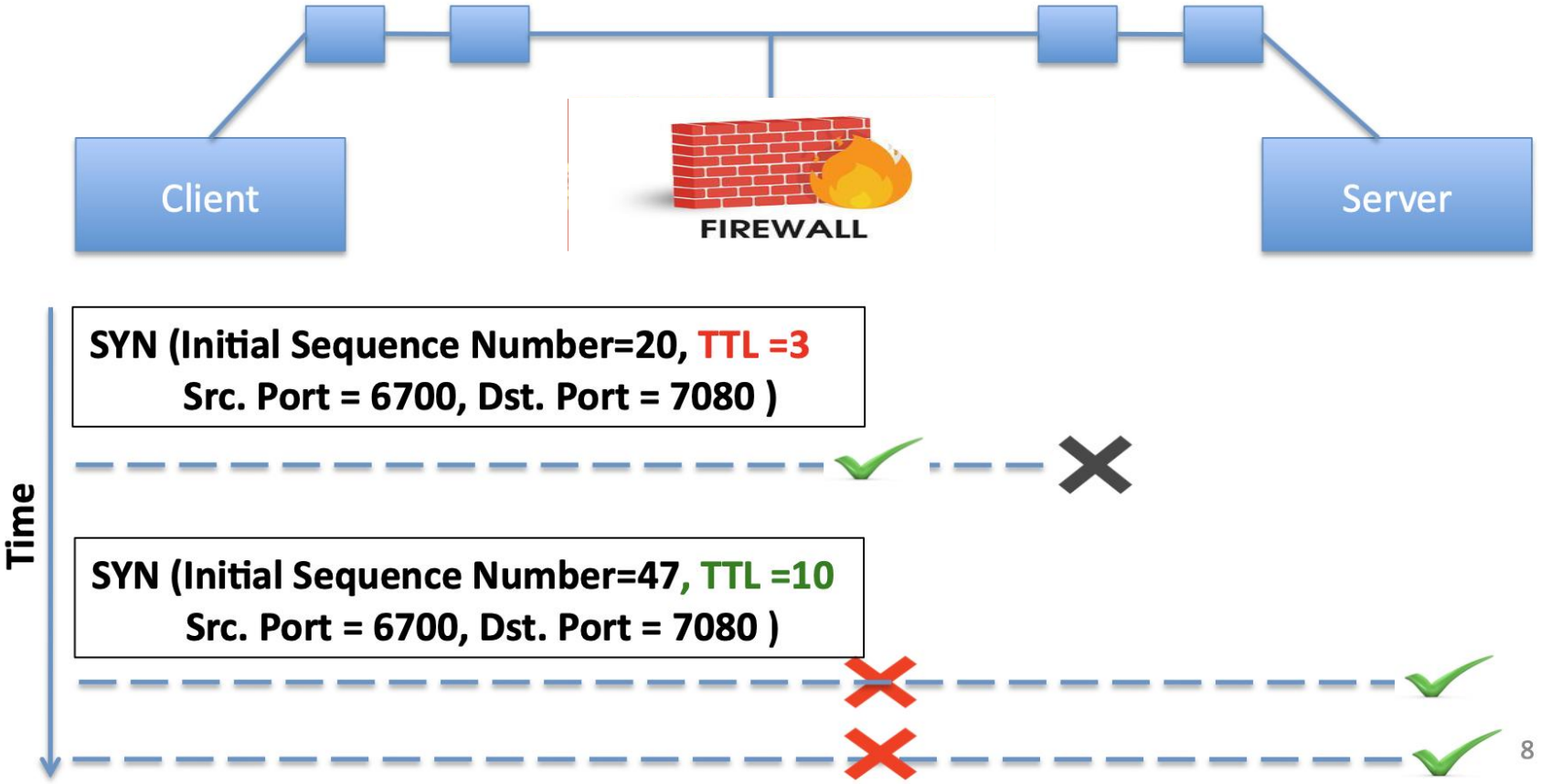
TCB Creation



TCB Teardown

1. TCB Creation (2)

Unsynchronized monitoring illustration



Test	Evasion Class	Description			
IP1	Ambiguity	$IP(TTL=<low>)p(Bad) \implies$			
IP2	Reassembly	Overlapping fragment proce			
TCP1	TCB creation	$IP(TTL=<low>)p_i^S, \quad p_{i+1}^S,$ $tuple(p_{i+1})) \wedge (seq(p_i) \neq s$			
TCP2	Incompleteness	$IP(ack=<bad>)p(Bad) \implies r$			
TCP3	Incompleteness	$IP(chksum=<bad>)p(Bad) \implies$			
TCP4	Incompleteness	$p^{\neg A}(Bad) \implies reset$			
TCP5	Reassembly	Overlapping segment proces			
TCP6[a]	TCB Teardown	$IP(TTL=<low>)p_i^{R(A)}, p_{i+1}(B$			
TCP6[b]	TCB Teardown	$IP(TTL=<low>)\ p_i^F, p_{i+1}(Bad$			
TCP7	State Management	$\tau(\leq \approx 10\text{ hr}), p_i(Bad) \implies$			
TCP8	State Management	$(p_i(Good)^+ \wedge \delta(Good) \leq \approx 1\text{ GB}), p_{i+1}(Bad) \implies reset$	State exhaust.	High	
TCP9	State Management	$hole, \ (p_i(Good)^+ \wedge \delta(Good) \geq 1\text{ KB} \wedge abovehole(p_i)),$ $p_{i+1}(Bad) \implies \neg reset$	State exhaust.	High	✓
TCP10	State Management	$hole, \tau(y) \geq 60\text{ min}, (p_i(Bad) \wedge abovehole(p_i)) \implies \neg reset$	State exhaust.	High	✓
HTTP1	Ambiguity	GET with > 1 space between method and URI $\implies \neg$ *reset*	Evasion	Low	
HTTP2	Incompleteness	GET with keyword at location > 2048 $\implies \neg$ *reset*	Evasion	Low	
HTTP3	Incompleteness	GET with keyword in $\geq$ 2nd of multiple requests in single segment $\implies \neg$ *reset*	Evasion	Low	
HTTP4	Incompleteness	GET with URL encoded (except %-encoding) $\implies \neg$ *reset*	Evasion	Low	✓

State management. We find that GFW maintains state for a connection for as long as 10 hours (TCP7), and for as much as 1 GB data transferred from client to server (TCP8). (Note, these values correspond to the maxima we tested, rather than precise thresholds. We obtained consistent results over repeated measurements for the values listed, but the boundaries used at a given time could depend on the monitor's operational load.)

We repeated these tests but with a preceding hole, finding that GFW abandons state more readily, doing so after having to buffer 1KB of data above the hole (TCP9). Likewise, GFW purges state after 60–90 minutes elapses for a connection that includes a hole (TCP10).

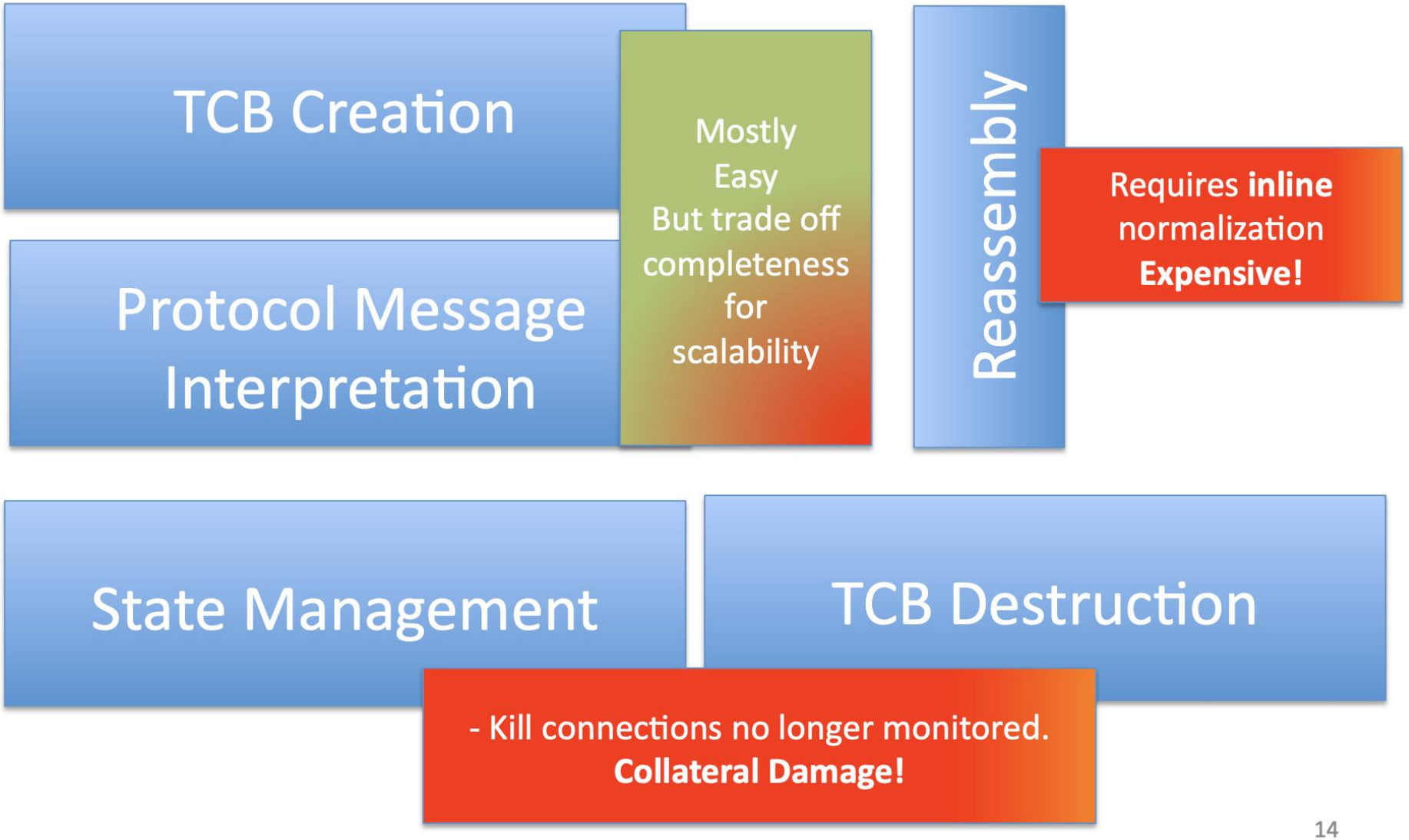
Table 1: Evasion opportunities in GFW's analysis of network traffic.

Test	Evasion Class	Description	Circumvention Opportunities	Fixing Cost	Receiver Dependent?
IP1	Ambiguity	$_{IP(TTL=<low>)}p(Bad) \implies reset$	Insertion	High	
IP2	Reassembly	Overlapping fragment processing	Insertion	High	✓
TCP1	TCB creation	$_{IP(TTL=<low>)}p_i^S,\ p_{i+1}^S,\ p_{i+2}(Bad) \wedge (tuple(p_i) = tuple(p_{i+1})) \wedge (seq(p_i) \neq seq(p_{i+1})) \implies \neg\, reset$	Insertion-Evasion	Low	
TCP2	Incompleteness	$_{IP(ack=<bad>)}p(Bad) \implies reset$	Insertion	Low	
TCP3	Incompleteness	$_{IP(chksum=<bad>)}p(Bad) \implies reset$	Insertion	Low	
TCP4	Incompleteness	$p^{\neg A}(Bad) \implies reset$	Insertion	Low	
TCP5	Reassembly	Overlapping segment processing	Insertion	High	✓
TCP6[a]	TCB Teardown	$_{IP(TTL=<low>)}p_i^{R(A)}, p_{i+1}(Bad) \implies \neg\, reset$	Insertion-Evasion	High	
TCP6[b]	TCB Teardown	$_{IP(TTL=<low>)}\ p_i^F, p_{i+1}(Bad) \implies \neg\, reset$	Insertion-Evasion	Low	
TCP7	State Management	$\tau(\leq \approx 10\text{ hr}), p_i(Bad) \implies reset$	State exhaust.	High	
HTTP1	Ambiguity	GET with > 1 space between method and URI $\implies \neg\, reset$	Evasion	Low	
HTTP2	Incompleteness	GET with keyword at location > 2048 $\implies \neg\, reset$	Evasion	Low	
HTTP3	Incompleteness	GET with keyword in $\geq$ 2nd of multiple requests in single segment $\implies \neg\, reset$	Evasion	Low	
HTTP4	Incompleteness	GET with URL encoded (except %-encoding) $\implies \neg\, reset$	Evasion	Low	✓

GET <SPACE>*2048 /path/to/url/bad_keyword/ HTTP/1.1
Host: allowed.host

Table 1: Evasion opportunities in GFW's analysis of network traffic. 2013

Cost of Fixing Evasion Bugs



How to find more insertion/evasion packets?

Your State is not mine: a closer look at evading stateful internet censorship, IMC 2017

Paper Reading this week

- Network Intrusion Detection: Evasion, Traffic Normalization, and End-to-End Protocol Semantics, Mark Handley, Christian Kreibich and Vern Paxson, USENIX Security 2001

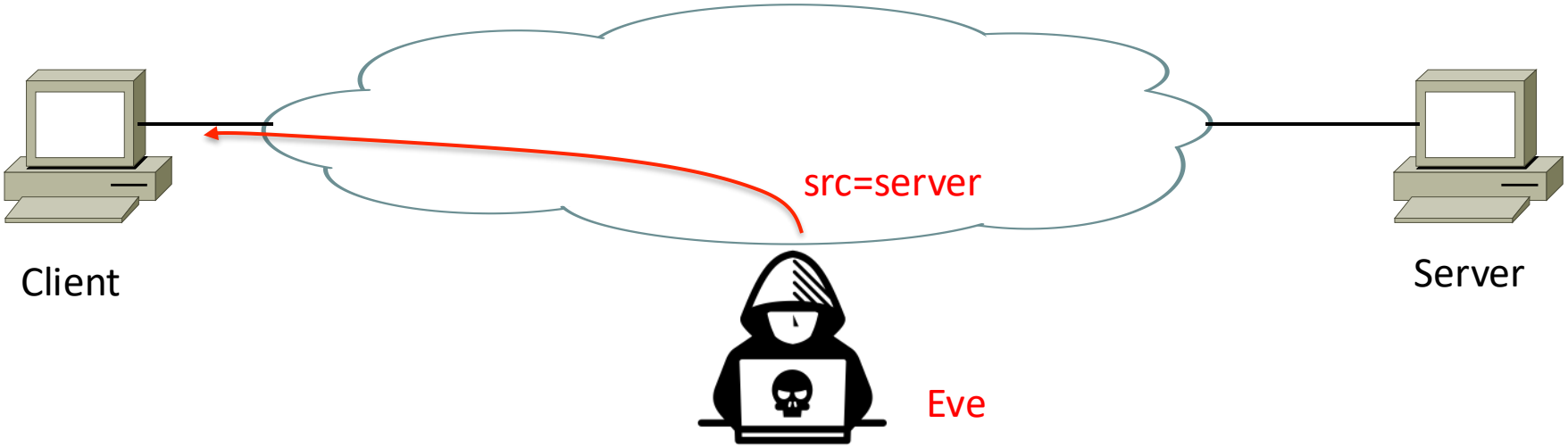
- Wang, Z., Cao, Y., Qian, Z., Song, C., & Krishnamurthy, S. V. Your state is not mine- a closer look at evading stateful internet censorship (pp. 114–127). IMC 2017.

https://docs.google.com/document/d/1pJMyvDcmd6WS_8N2ecULgx01iZQWPRL9-KixsmNSivA/edit?usp=sharing

CONTENTS

- IP Protocol Security
  - IDS evasion and Normalization
  - Fragment Problems
  - NIDS Confusion
- TCP Security
  - TCB maintenance Problems
  - TCP Hijacking and sequence number prediction
  - SYN Flood
  - Blind RST

Off-Path /Blind Attack： threat model



- Goal:
  - Interrupt or disconnect connection between victims
  - Inject data into the connection
- Conditions and constraints
  - Attacker can spoof the source IP address
  - Attacker cannot see any packets between victims

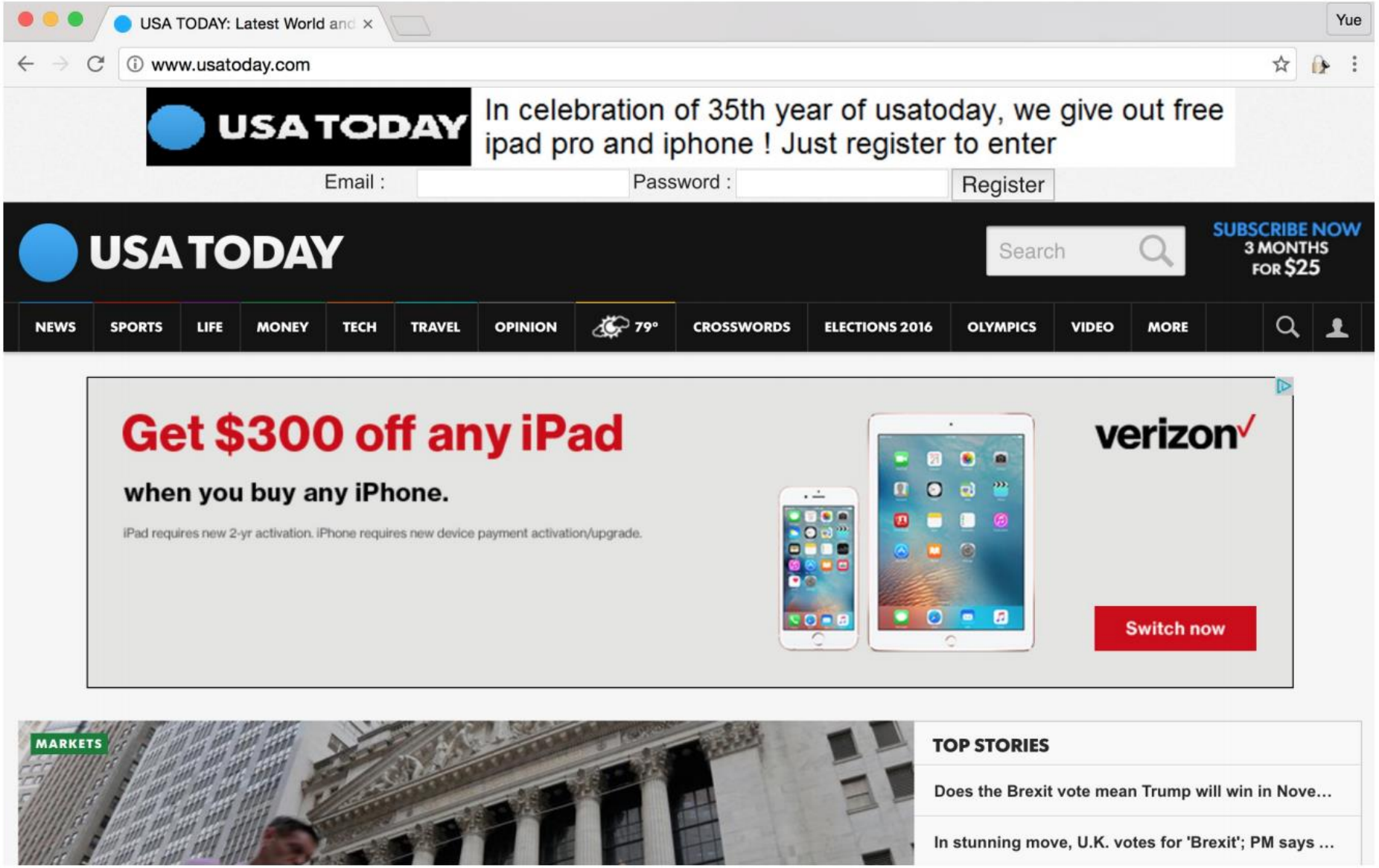
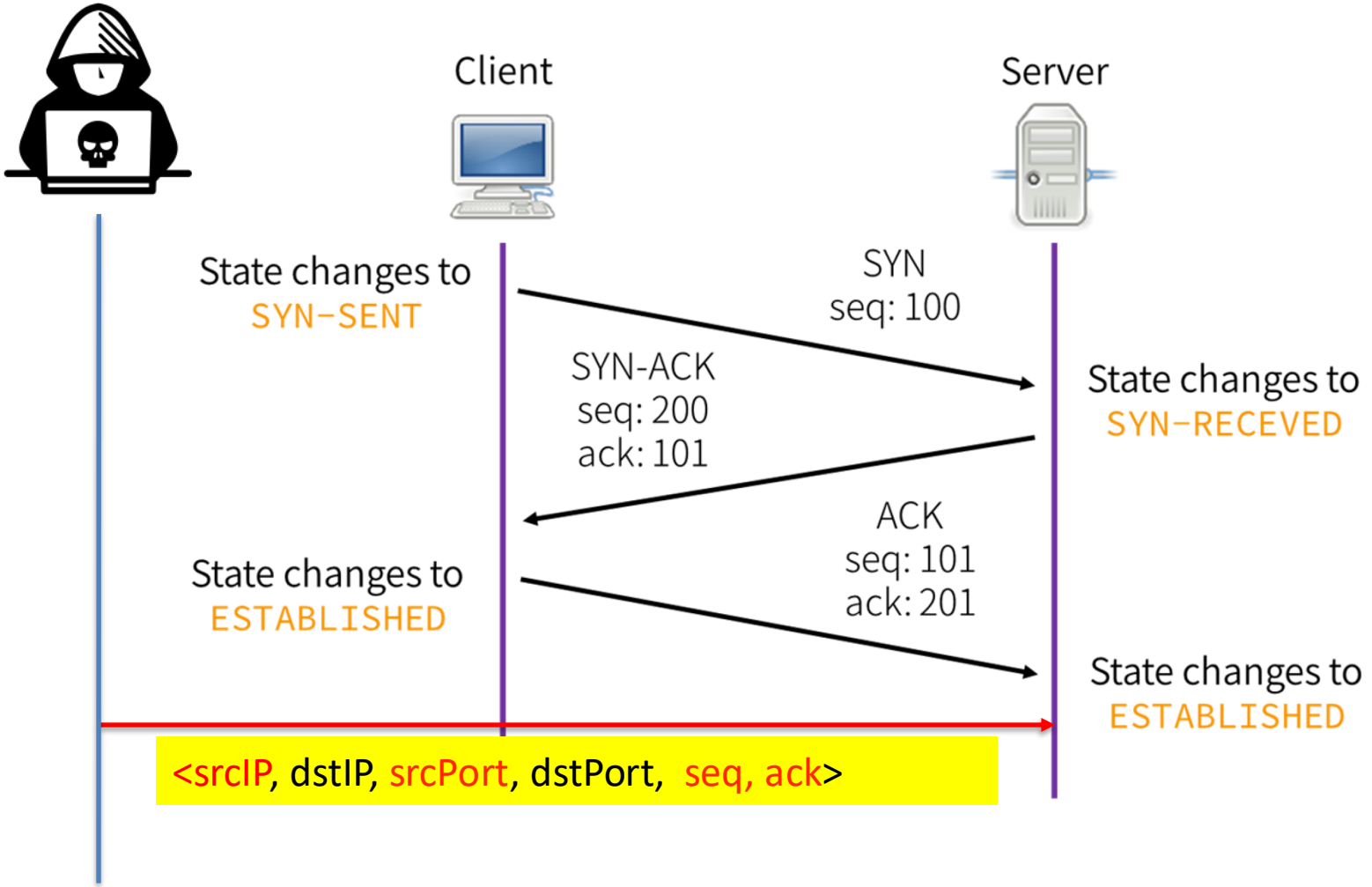
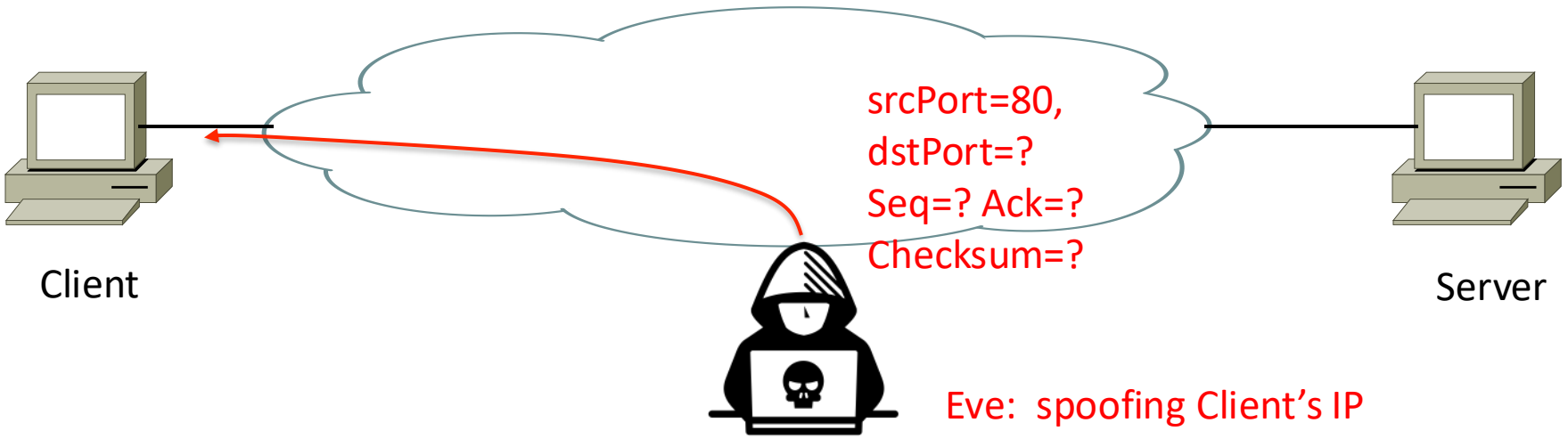


Figure 13: USAToday screenshot with phishing registration window

Off-Path /Blind Attack： threat model



Threat Model



TCP header:

00	01	02	03	04	05	06	07	08	09	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Source Port																Destination Port															
Sequence Number																															
Acknowledgment Number																															
Data Offset				reserved			ECN			Control Bits						Window															
Checksum																															
Urgent Pointer																															
Options and padding :::																															
Data :::																															

计算Checksum
包括 IP Header

预测Initial Sequence Number (ISN)

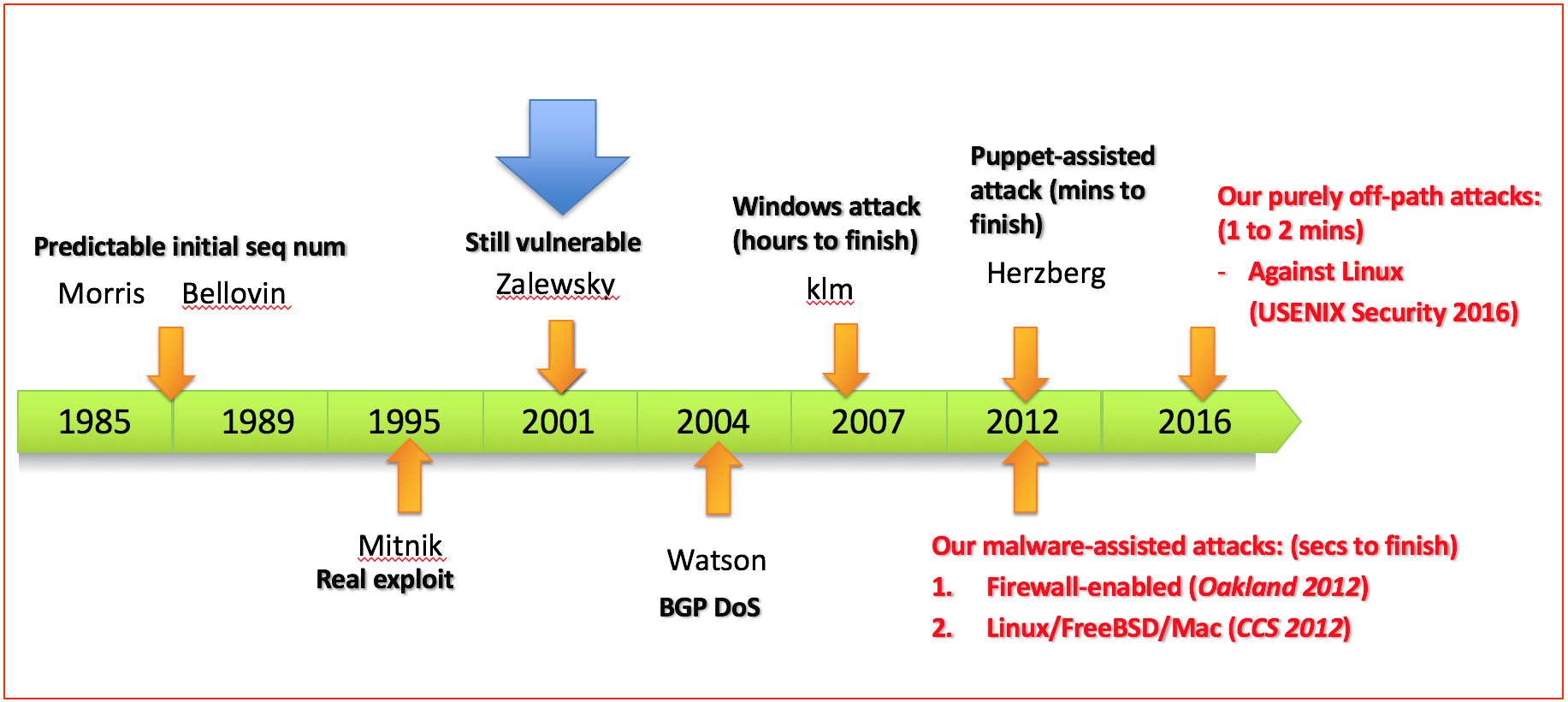
- ISN是怎么产生、维护的？

```
Initial Sequence Number Selection

The protocol places no restriction on a particular connection being
used over and over again.  A connection is defined by a pair of
sockets.  New instances of a connection will be referred to as
incarnations of the connection.  The problem that arises from this is
-- "how does the TCP identify duplicate segments from previous
incarnations of the connection?"  This pr
connection is being opened and closed in
connection breaks with loss of memory and

To avoid confusion we must prevent segments from one incarnation of a
connection from being used while the same sequence numbers may still
be present in the network from an earlier incarnation.  We want to
assure this, even if a TCP crashes and loses all knowledge of the
sequence numbers it has been using.  When new connections are created,
an initial sequence number (ISN) generator is employed which selects a
new 32 bit ISN.  The generator is bound to a (possibly fictitious) 32
bit clock whose low order bit is incremented roughly every 4
microseconds.  Thus, the ISN cycles approximately every 4.55 hours.
Since we assume that segments will stay in the network no more than
the Maximum Segment Lifetime (MSL) and that the MSL is less than 4.55
hours we can reasonably assume that ISN's will be unique.

For each connection there is a send sequence number and a receive
sequence number.  The initial send sequence number (ISS) is chosen by
the data sending TCP, and the initial rec
learned during the connection establishi
```

ISN: 32位的时钟，每4毫秒加1 (why?)

https://datatracker.ietf.org/doc/html/rfc793

TCP sequence number war timeline



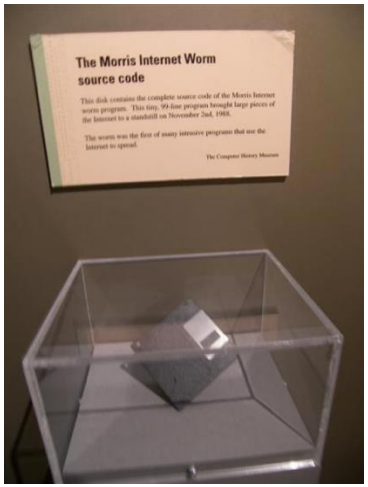
Slide from Prof. Zhiyun Qian @ UC Riverside

钱志云教授在清华大学的报告视频：When TCP Meets Side Channels

https://www.inforsec.org/wp/?p=1778

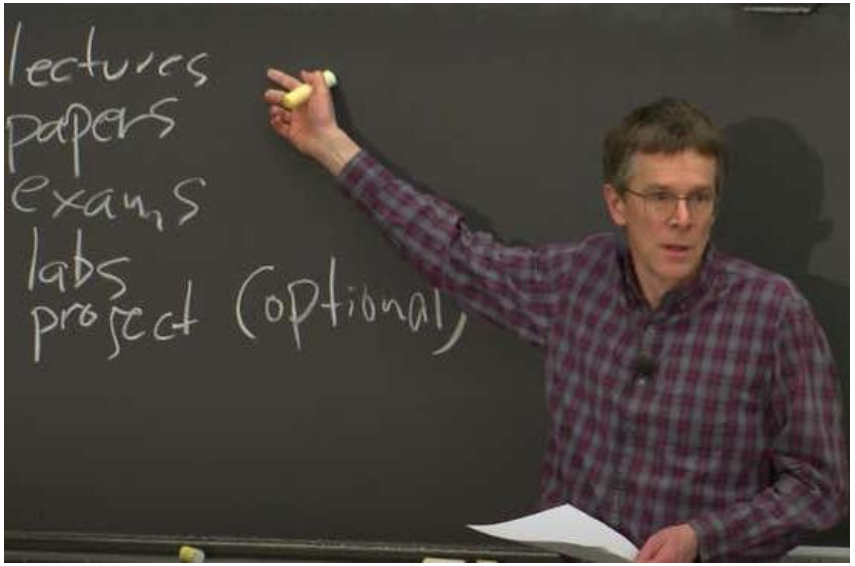
Robert Tappan Morris, 1965-

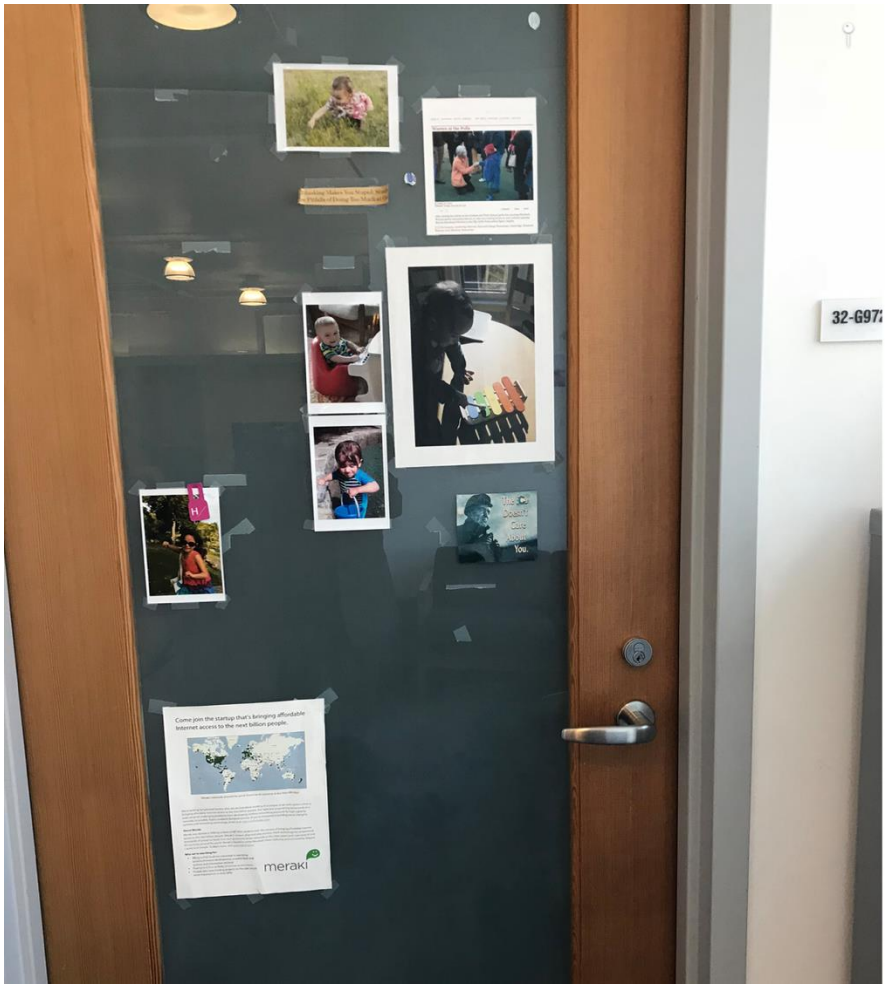
- Morris Worm(1988):
  - sendmail,
  - fingerd, rsh/rexec,
  - weak password
  - Infected 10% of Internet



computer scientist and entrepreneur
Professor of MIT, ACM Fellow
Cofounder of ViaWeb, $49 M acquired by Yahoo!

Professor Robert Morris





A Weakness in the 4.2BSD Unix† TCP/IP Software

Robert T. Morris

Feb,1985

4.2BSD maintains a global initial sequence number, which is incremented by 128 each second and by 64 after each connection is started; each new connection starts off with this number. When a SYN packet with a forged source is sent from a host, the destination host will send the reply to the presumed source host, not the forging host. The forging host must discover or guess what the sequence number in that lost packet was, in order to acknowledge it and put the destination TCP port in the ESTABLISHED state. Guessing the lost sequence number is easy when the destination runs 4.2BSD; one need only create a real connection, look in the kernel for the sequence number received, and add 64 to it. Once the forging program acknowledges this sequence number, the connection is fully set up and data may be sent, though not received, by the program.

Unfortunately, the SYN packet sent by the destination to the putative source does not just disappear. The supposed source sees it as a packet on a non-existent circuit, and sends a packet with a RST flag to the destination. This causes the destination to throw away the forged circuit. For instance: Host A sends a forged packet to B, claiming the source was C. B sends a SYN packet to C, and C sends a RST packet to B. B throws away the circuit that A is forging to it. The only ports on C that won't always generate RSTs in this situation are those which are waiting, or listening, for connections. Those listening ports have finite length queues of connections waiting to be set up; if this queue length is exceeded, the requesting SYN packet will be thrown away, but no reset will be generated. The originator is expected to resend the SYN packet after timing out. Note that original SYN packets and response SYN packets look the same. Thus it suffices for the forging process to claim that the packets are coming from a port on the supposed source that has a server listening for connections, and for the forger to flood that port with connection requests.

In summary, suppose the forging program is named A, its destination host is named B, the source to be forged is named C. The port on B involved is number 514, the remote execution server's port; A will forge packets from port 21 on host C, which is usually waiting for connections. The chain of events on A is as follows:

https://pdos.csail.mit.edu/~rtm/papers/117.pdf

黑客：凯文·米特尼克(Kevin Mitnick)

16岁入侵电信公司修改大量家庭电话被捕，后成为第一位少年网络在逃犯

入侵过IBM、摩托罗拉等40多家大公司和美国国防预警系统，曾是美国FBI通缉的“头号黑客”

因入侵下村努后被捕，判刑三年出狱后成为一名安全顾问，公共演说家和作家。

米特尼克安全公司

入侵的艺术

欺骗的艺术

线上幽灵

社会工程攻击
(social engineering)

1.冒充身份：假装成公司内部部门（如IT）打电话给目标用户，以维护为名要求目标用户提供登录信息或者安装恶意软件。

2.伪造电子邮件：伪造合法机构的电子邮件，要求收件人更新个人信息或者登录到一个伪造的网站，从而窃取登录凭证。

3.垃圾邮件钓鱼：群发垃圾邮件，声称用户赢得了奖品，要求他们提供个人信息或者支付运费。

4.心理攻击：通过电话与目标交流，利用心理操纵技巧，例如利用目标的同情心或紧急情况的恐惧，来获取机密信息。

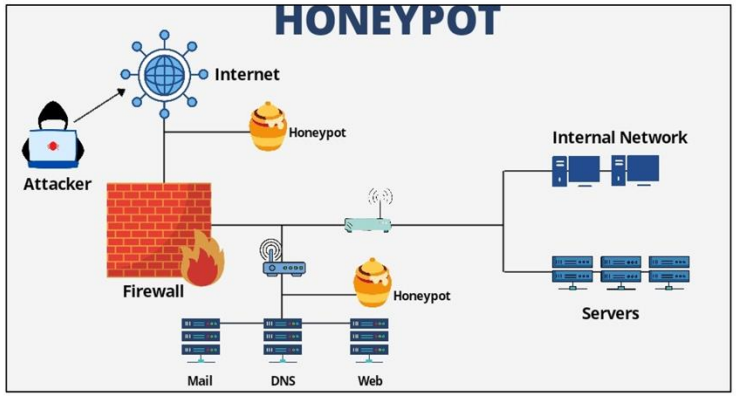
下村努（Shimomura Tsutomu）

日裔美籍物理学家、计算机安全专家，后来UCSD超级计算中心特别研究员，同时为NSA工作。

下村努推测：Mitnick为了获得入侵工具而入侵了自己的电脑

- 书《Takedown: The Pursuit and Capture of Kevin Mitnick, America's Most Wanted Computer Outlaw》, 1996
- 电影: Track Down, 2000, by Disney

蜜罐（Honeypot）吸引攻击者入侵的诱骗系统，用于观察、追踪攻击者的行为，收集威胁情报



The Mitnick attack

Computer security is an important factor in our information world with Internet and digitally owned materials. Over the past twenty years, network security has evolved continuously. More secure implementations are invented to replace old less secure implementations. Kevin Mitnick was able to hack into Tsutomu Shimomura's X–Terminal computer due to early implementation of TCP connection, which was not really secure at that time. With a huge desire of curiosity, Mitnick did something that no one has ever done before him. He exploited the trusted relationship between two computers by performing man–in–the–middle attack under a spoofed identity. His attack made him the most famous hacker in United States of America.

Contents [hide]

1 Who is Mitnick?
2 Three–way handshake
  2.1 Step 1: SYN request
  2.2 Step 2: SYN/ACK response
  2.3 Step 3: ACK or RESET response
3 The attack
  3.1 Step 1: Information gathering
    3.1.1 Determine the TCP sequence number generator's behavior
    3.1.2 Determine a trusted relationship between X–Terminal and Server
  3.2 Step 2: The flood
  3.3 Step 3: Trusted relationship hijacking
  3.4 Step 4: Remote command pump
  3.5 Step 5: Clean up
4 Detection
5 Prevention
6 Comments
7 References
8 See also
9 External links

Mitnick on FBI wanted list

http://wiki.cas.mcmaster.ca/index.php/The_Mitnick_attack

RSH(Remote Shell), Trust by IP

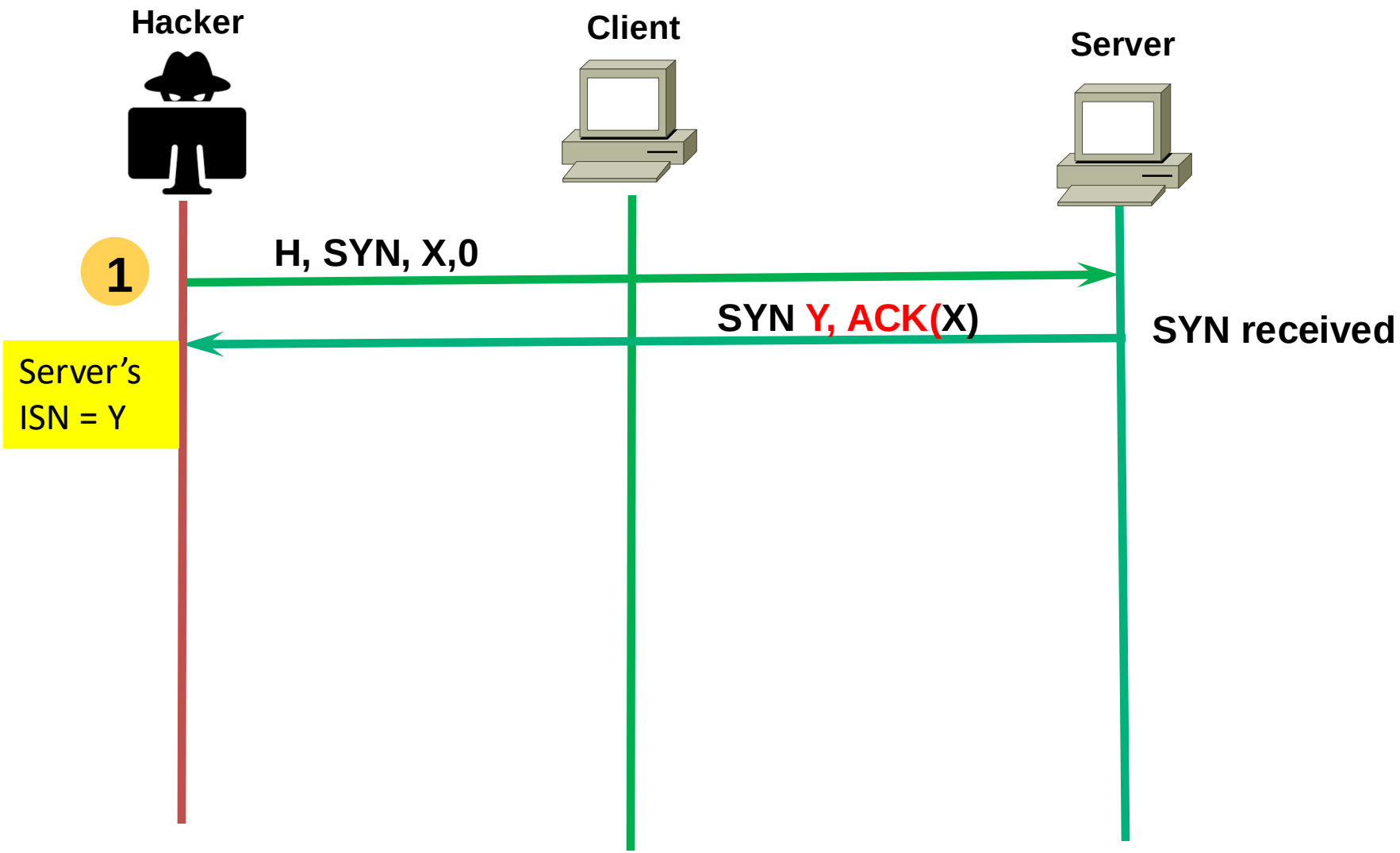
IP: 1.1.1.1

/root/.rhosts:

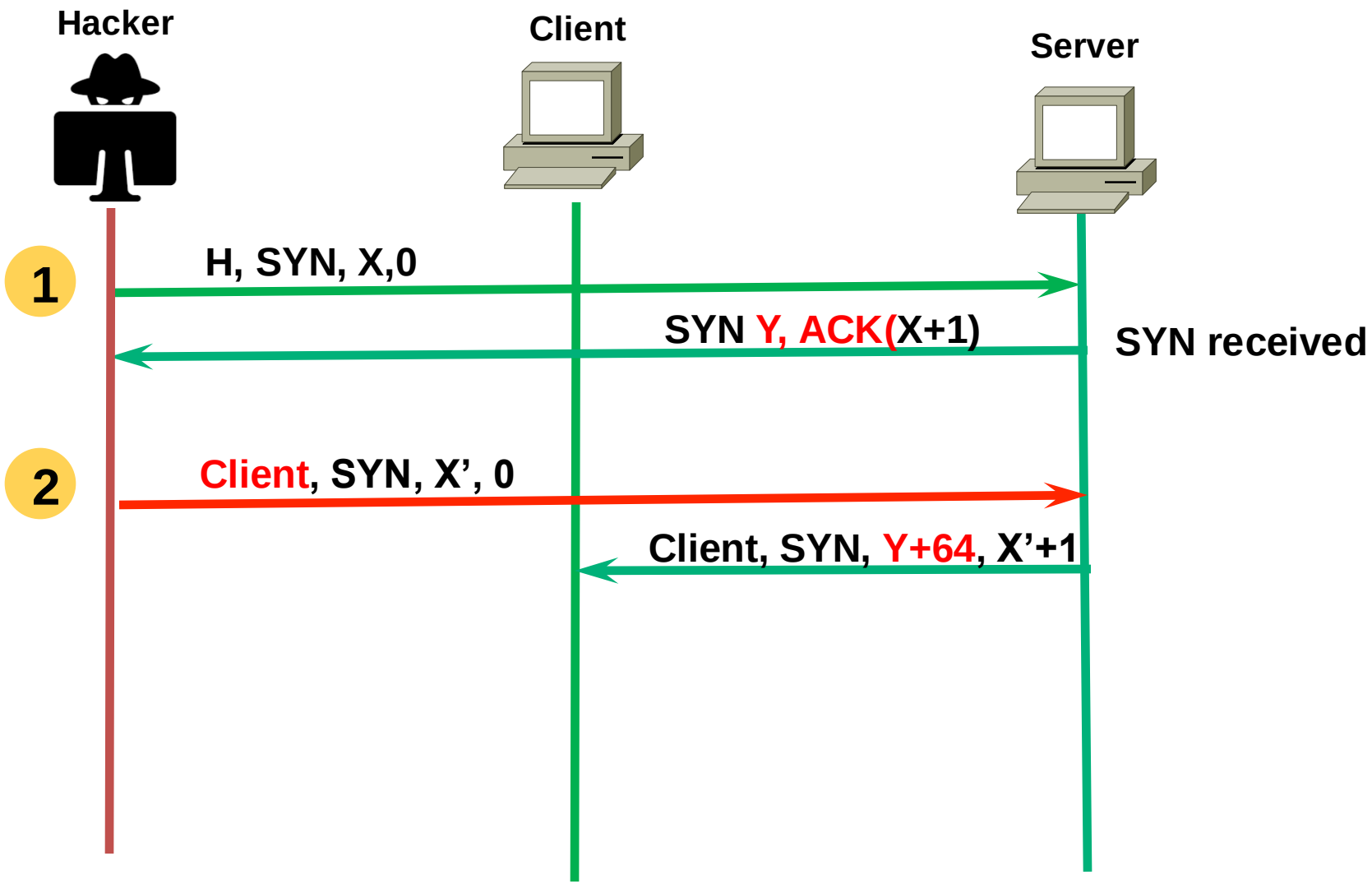
+ 1.1.1.1

```
$ rsh -l  root  host.example.com ”echo ‘++’ > /root/.rhosts"
```

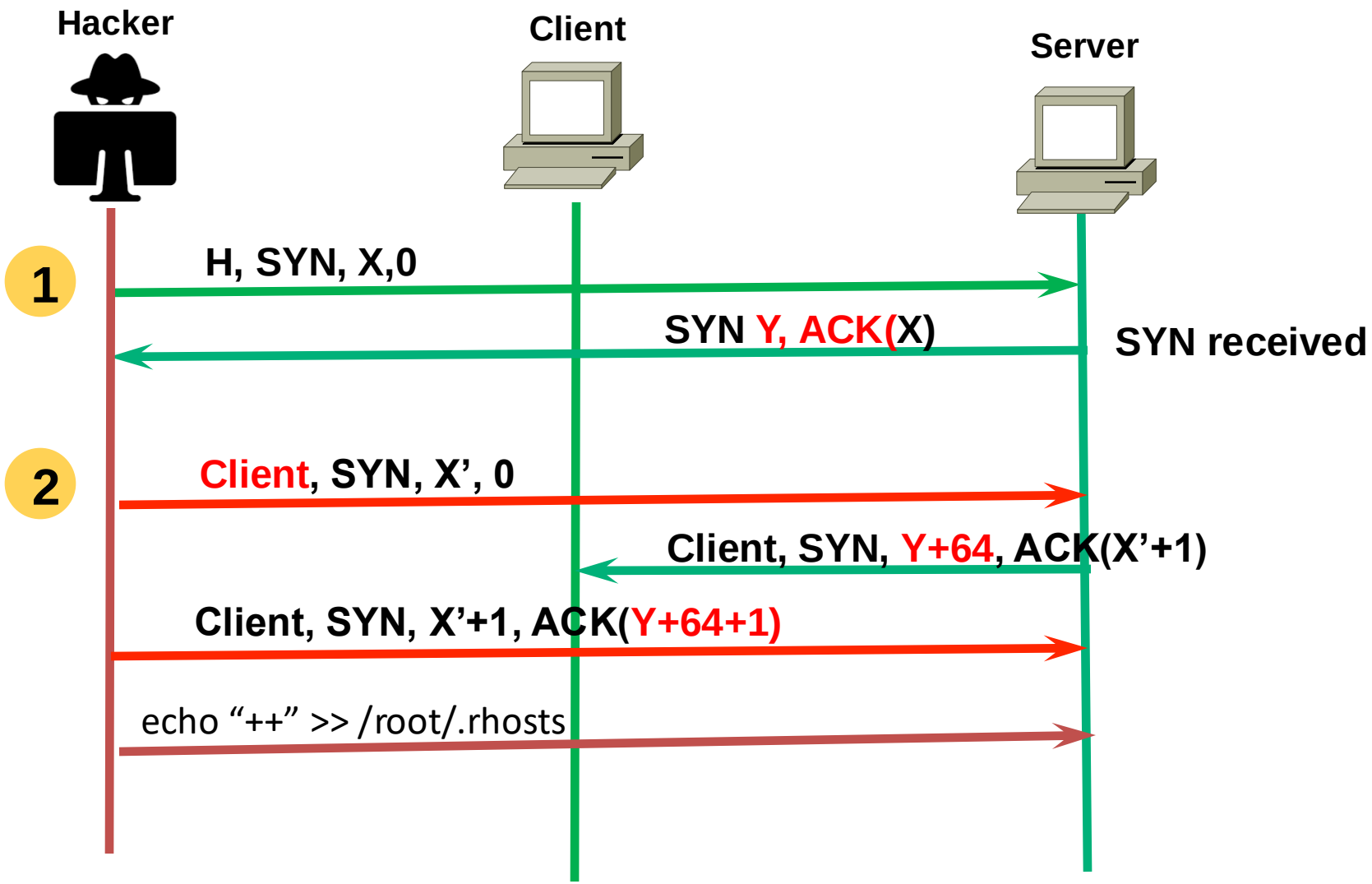
TCP Hijacking



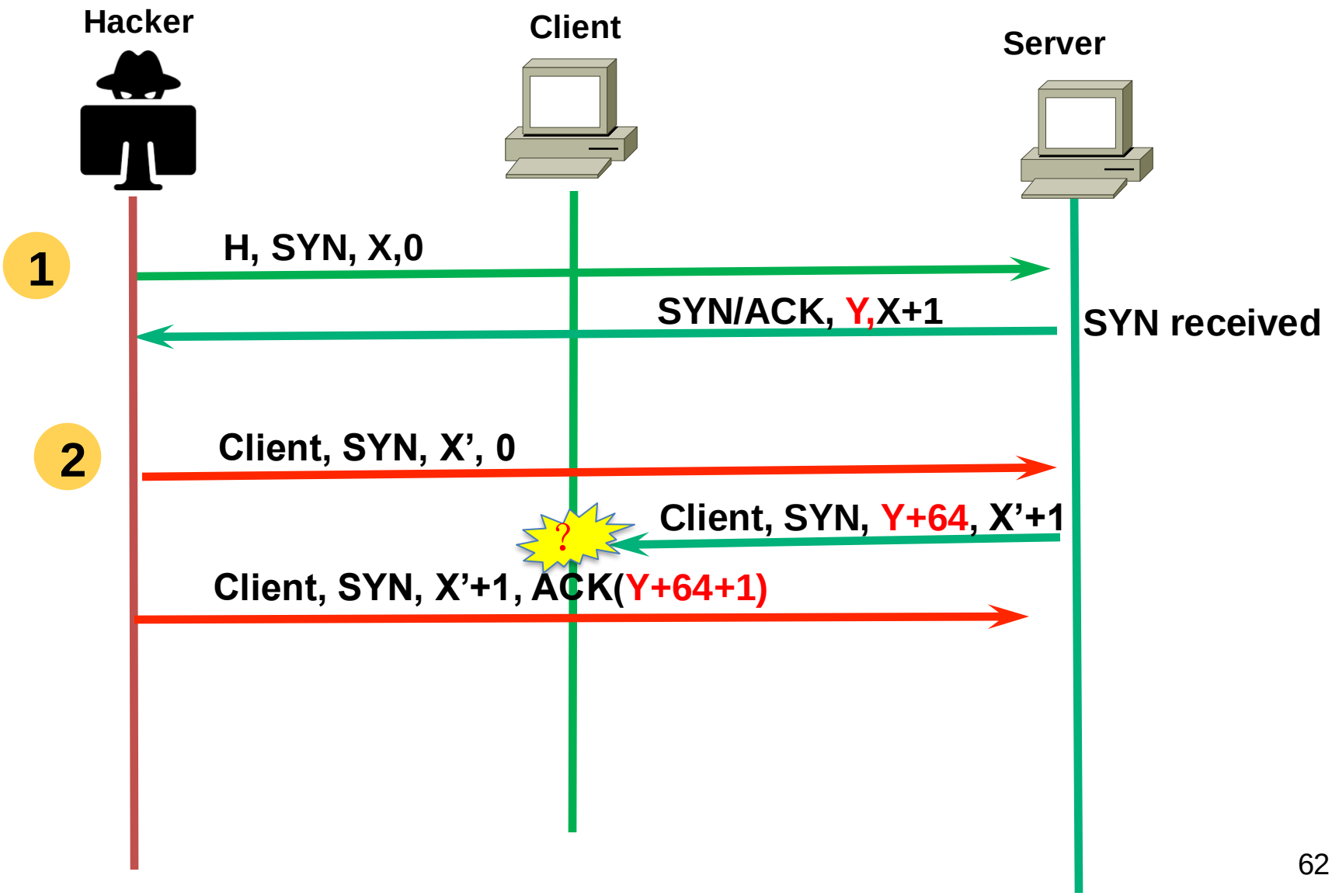
TCP Hijacking



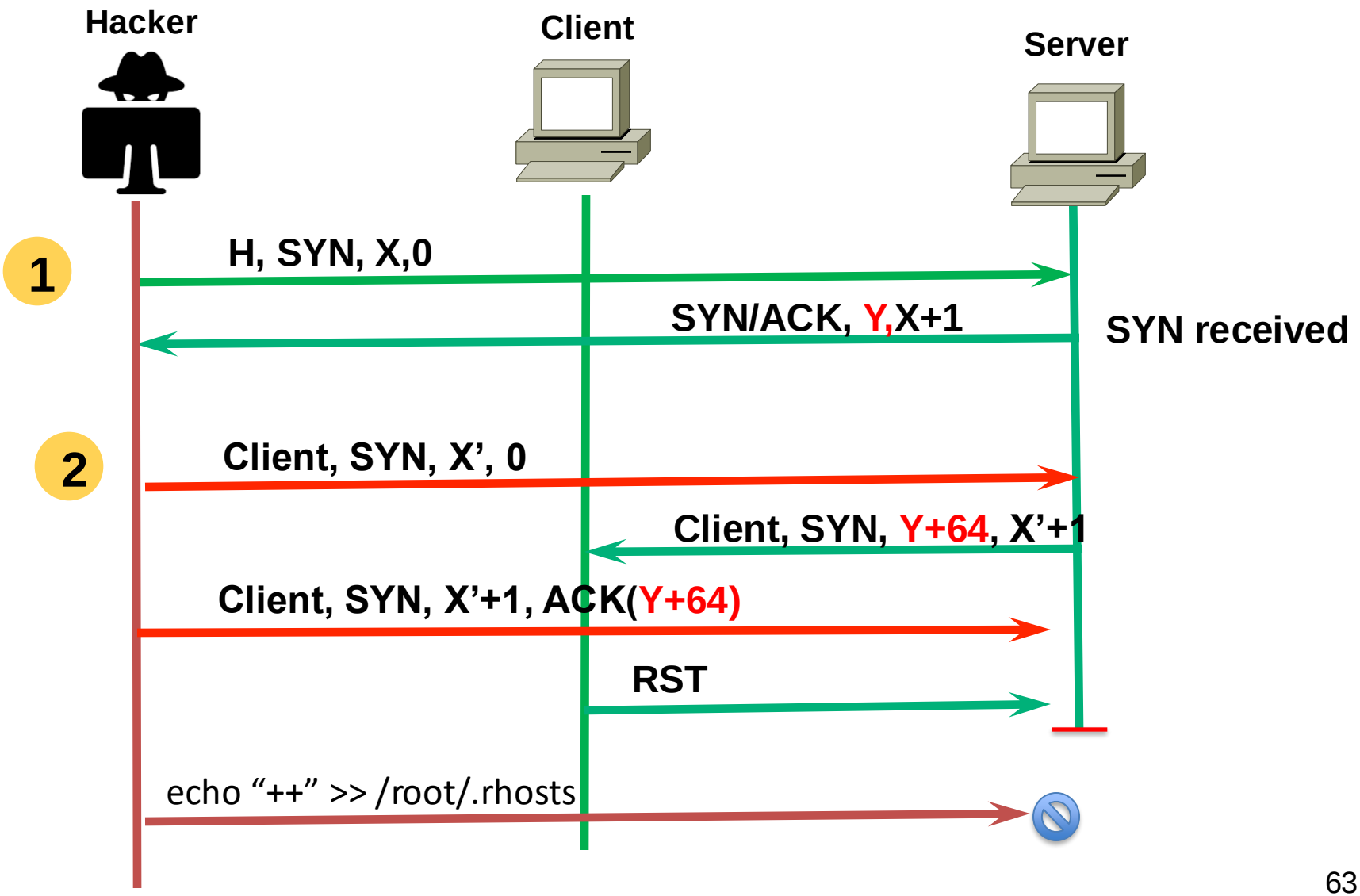
TCP Hijacking



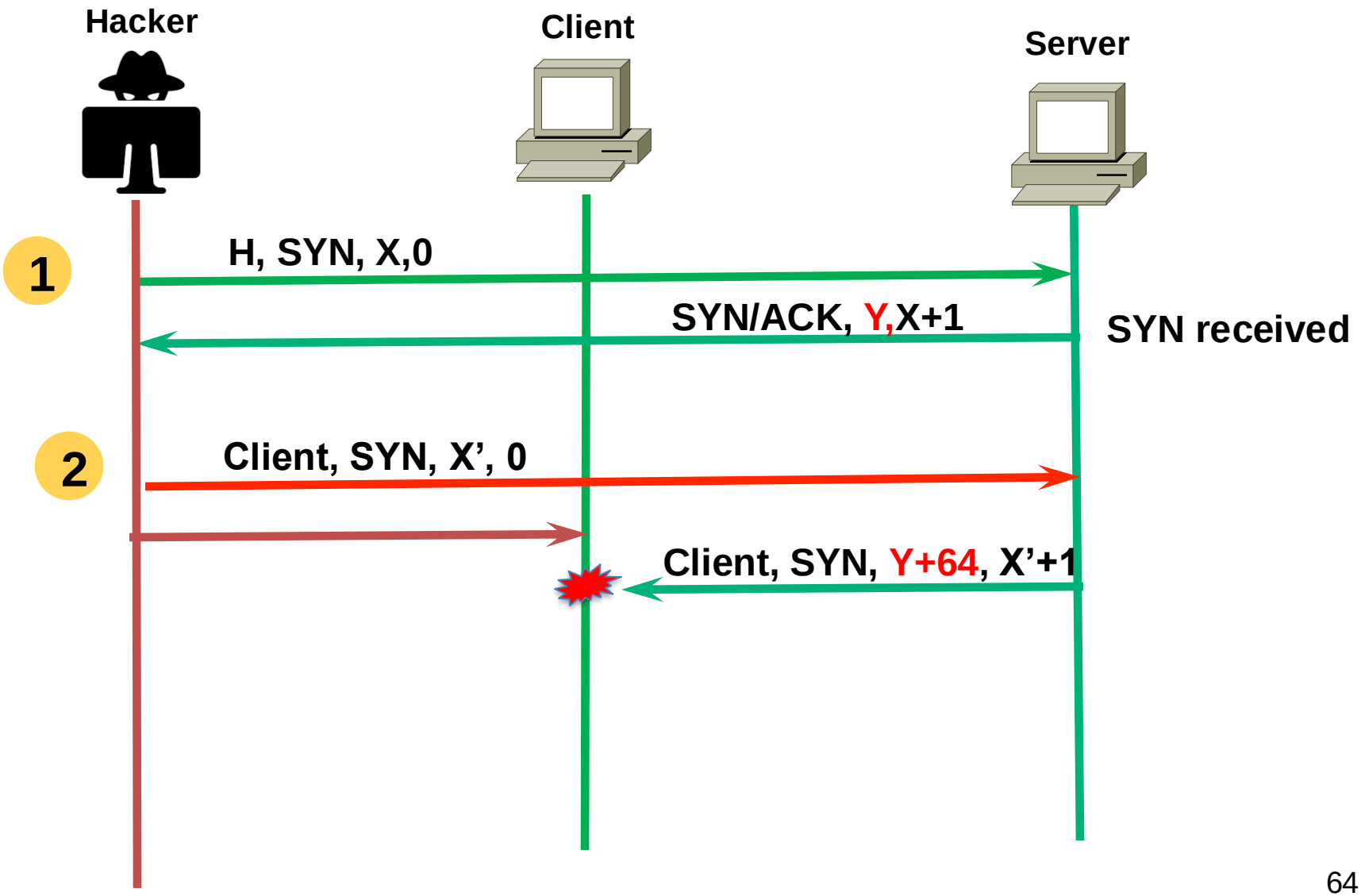
想的太简单了


Hacker
Client
Server
1
H, SYN, X,0
SYN/ACK, Y,X+1
SYN received
2
Client, SYN, X', 0
?
Client, SYN, Y+64, X'+1
Client, SYN, X'+1, ACK(Y+64+1)

想的太简单了


Hacker
Client
Server
1
H, SYN, X,0
SYN/ACK, Y,X+1
SYN received
2
Client, SYN, X', 0
Client, SYN, Y+64, X'+1
Client, SYN, X'+1, ACK(Y+64)
RST
echo "++" >> /root/.rhosts

如何让Client 不产生 RST?



Flood connection request to Client

FTP service port

Swamp port 21 on C with connection requests.
Create a real connection to a port on B, and record the
sequence number returned by B.
Create a raw IP socket, change its protocol to that of TCP,
and change its source to C (by writing in the kernel).
Send a SYN packet from port 21 (supposedly on C) to port 514 on B.
(A then sends a SYN to port 21 on C, which is silently ignored because
C's queue for 21 is full.)
Send an ACK packet to B with the acknowledgement number equal to the
sequence number previously recorded plus 64.
Send data to B, taking care to increment the sequence number each time
by the amount of data sent. Port 514 expects a null, followed
by a user name, followed by a command.
If all goes well, and B trusts C, B will execute the command.

A Weakness in the 4.2BSD Unix† TCP/IP Software

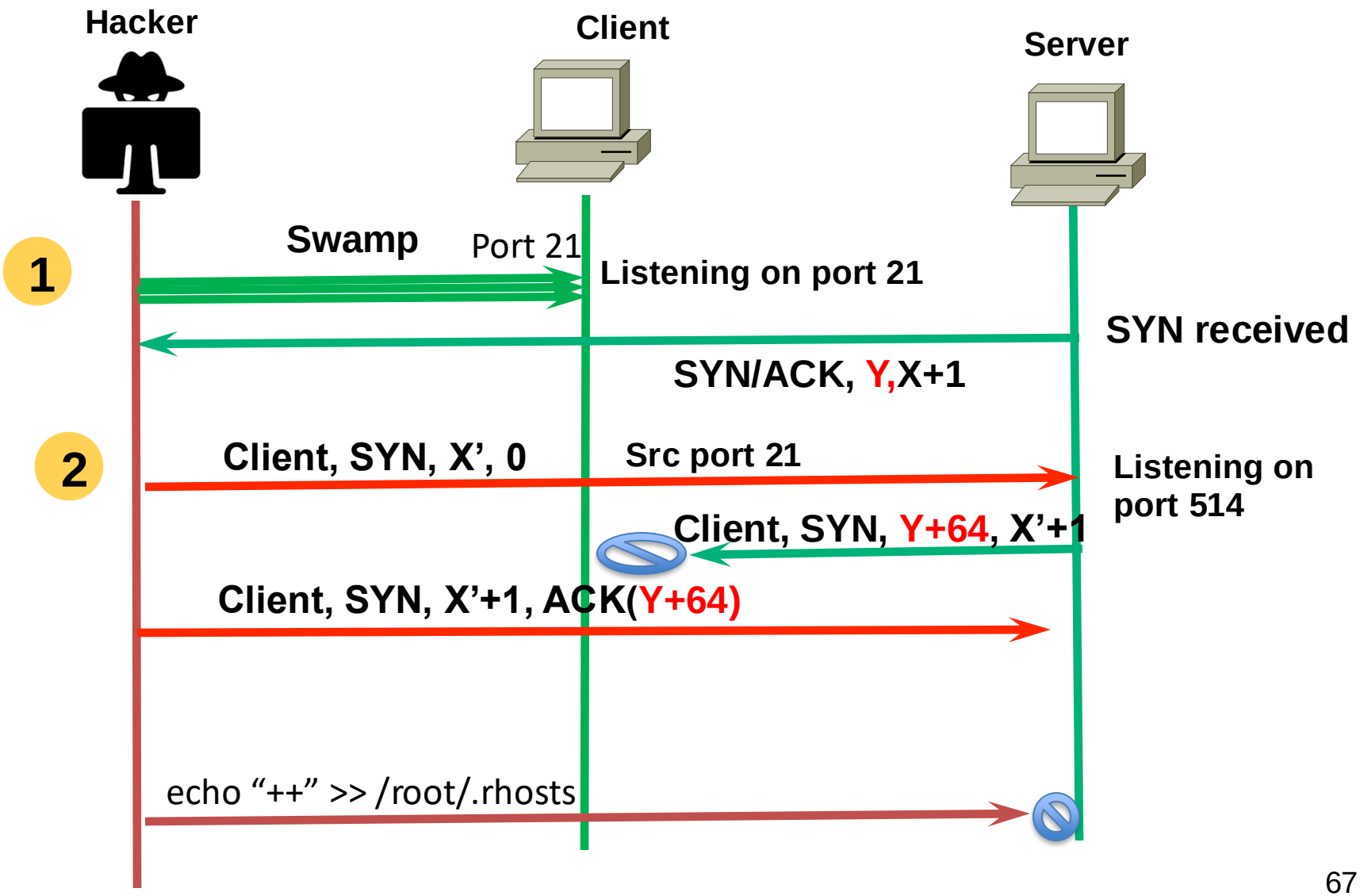
服务端口处于监听状态且队列已满

4.2BSD maintains a global initial sequence number, which is incremented by 128 each second and by 64 after each connection is started; each new connection starts off with this number. When a SYN packet with a forged source is sent from a host, the destination host will send the reply to the presumed source host, not the forging host. The forging host must discover or guess what the sequence number in that lost packet was, in order to acknowledge it and put the destination TCP port in the ESTABLISHED state. Guessing the lost sequence number is easy when the destination runs 4.2BSD; one need only create a real connection, look in the kernel for the sequence number received, and add 64 to it. Once the forging program acknowledges this sequence number, the connection is fully set up and data may be sent, though not received, by the program.

Unfortunately, the SYN packet sent by the destination to the putative source does not just disappear. The supposed source sees it as a packet on a non-existent circuit, and sends a packet with a RST flag to the destination. This causes the destination to throw away the forged circuit. For instance: Host A sends a forged packet to B, claiming the source was C. B sends a SYN packet to C, and C sends a RST packet to B. B throws away the circuit that A is forging to it. The only ports on C that won't always generate RSTs in this situation are those which are waiting, or listening, for connections. Those listening ports have finite length queues of connections waiting to be set up; if this queue length is exceeded, the requesting SYN packet will be thrown away, but no reset will be generated. The originator is expected to resend the SYN packet after timing out. Note that original SYN packets and response SYN packets look the same. Thus it suffices for the forging process to claim that the packets are coming from a port on the supposed source that has a server listening for connections, and for the forger to flood that port with connection requests.

In summary, suppose the forging program is named A, its destination host is named B, the source to be forged is named C. The port on B involved is number 514, the remote execution server's port; A will forge packets from port 21 on host C, which is usually waiting for connections. The chain of events on A is as follows:

TCP Hijacking



1.Bellovin, S. M. Security problems in the TCP/IP protocol suite. *ACM SIGCOMM Computer Communication Review* **19**, 1989

Security Problems in the TCP/IP Protocol Suite

*S.M. Bellovin**
smb@ulysses.att.com

AT&T Bell Laboratories
Murray Hill, New Jersey 07974

ABSTRACT

The TCP/IP protocol suite, which is very widely used today, was developed under the sponsorship of the Department of Defense. Despite that, there are a number of serious security flaws inherent in the protocols, regardless of the correctness of any implementations. We describe a variety of attacks based on these flaws, including sequence number spoofing, routing attacks, source address spoofing, and authentication attacks. We also present defenses against these attacks, and conclude with a discussion of broad-spectrum defenses such as encryption.

1. INTRODUCTION

The TCP/IP protocol suite[1][2], which is very widely used today, was developed under the sponsorship of the Department of Defense. Despite that, there are a number of serious security flaws inherent in the protocols. Some of these flaws exist because hosts rely on IP source address for authentication; the Berkeley "*r*-utilities"[3] are a notable example. Others exist because network control mechanisms, and in particular routing protocols, have minimal or non-existent authentication.

When describing such attacks, our basic assumption is that the attacker has more or less complete control over some machine connected to the Internet. This may be due to flaws in that machine's own protection mechanisms, or it may be because that machine is a microcomputer, and inherently unprotected. Indeed, the attacker may even be a rogue system administrator.

1.1 Exclusions

We are not concerned with flaws in particular implementations of the protocols, such as those used by the Internet "worm"[4][5][6]. Rather, we discuss generic problems with the protocols themselves. As will be seen, careful implementation techniques can alleviate or prevent some of these problems. Some of the protocols we discuss are derived from Berkeley's version of the UNIX® system; others are generic Internet protocols.

We are also not concerned with classic network attacks, such as physical eavesdropping, or altered or injected messages. We discuss such problems only in so far as they are facilitated or possible because of protocol problems.

For the most part, there is no discussion here of vendor-specific protocols. We do discuss some problems with Berkeley's protocols, since these have become de facto standards for many vendors, and not just for UNIX systems.

2. TCP SEQUENCE NUMBER PREDICTION

One of the more fascinating security holes was first described by Morris[7]. Briefly, he used TCP sequence number prediction to construct a TCP packet sequence without ever receiving any responses from the server. This allowed him to spoof a trusted host on a local network.

* Author's address: Room 3D-558, AT&T Bell Laboratories, 600 Mountain Avenue, Murray Hill, New Jersey 07974.

2.Bellovin, S. M. A look back at "Security Problems in the TCP/IP Protocol Suite." ACSAC 2004

A Look Back at "Security Problems in the TCP/IP Protocol Suite"

Steven M. Bellovin
AT&T Labs—Research
bellovin@acm.org

Abstract

About fifteen years ago, I wrote a paper on security problems in the TCP/IP protocol suite. In particular, I focused on protocol-level issues, rather than implementation flaws. It is instructive to look back at that paper, to see where my focus and my predictions were accurate, where I was wrong, and where dangers have yet to happen. This is a reprint of the original paper, with added commentary.

1. Introduction

The paper "Security Problems in the TCP/IP Protocol Suite" was originally published in *Computer Communication Review*, Vol. 19, No. 2, in April, 1989. It was a protocol-level analysis; I intentionally did not consider implementation or operational issues. I felt—and still feel—that that was the right approach. Bugs come and go, and everyone's operational environment is different. But it's very hard to fix protocol-level problems, especially if you want to maintain compatibility with the installed base.

This paper is a retrospective on my original work. New commentary is shown indented, in a sans serif font. The original text is otherwise unchanged, except for possible errors introduced when converting it from *troff* to LaTeX. I've left the references intact, too, even if there are better versions today. The reference numbers and pagination are, of course, different; the section numbers remain the same, except for a new "Conclusions" section. As a general rule, the commentary *follows* the section it's discussing.

This paper was presented at 20th Annual Computer Security Applications Conference (ACSAC), December 2004, in as part of the "classic papers" track.

It helps to understand where this paper came from. When I started work at Bell Labs Murray Hill in 1982, I assumed ownership of $1\frac{1}{2}$ of the first three pieces of Ethernet cable in all of AT&T, then a giant monopoly telephone company. My lab had one cable, another lab had a second, and a "backbone" linked the two labs. That backbone grew, as other labs connected to it. Eventually, we scrounged funds to set up links to other Bell Labs locations; we called the resuling network the "Bell Labs Internet" or the "R&D Internet", the neologism "Intranet" not having been invented.

Dedicated routers were rare then; we generally stuck a second Ethernet board into a VAX or a Sun and used it to do the routing. This mean that the routing software (we used Berkeley's *routed*) was accessible to system administrators. And when things broke, we often discovered that it was a routing problem: someone had misconfigured their machine. Once, we found that someone had plugged a new workstation into the Murray Hill backbone, rather than into his department's network; worse yet, the (proprietary) address assignment software on his machine didn't see any (proprietary) address assignment servers on that network, so it allocated .1—the gatewayy router—to itself. These two situations worried me; it was clear that anything that could happen by accident could be done deliberately, possibly with serious consequences.

Several other things focused my attention on security even more. One was Robert Morris' discovery of sequence number guessing attacks; these are discussed extensively below. Another was the "Shadow Hawk" incident—a teenager broke into var-

Assuming your opponents know it in detail

"Steven M. Bellovin" <smb@cs.columbia.edu>
Sat, 6 Jun 2009 22:21:01 -0400

The subject of security through obscurity comes up frequently. I think a lot of the debate happens because people misunderstand the issue.

It helps, I think, to go back to Kerckhoffs' second principle, translated as "The system must not require secrecy and can be stolen by the enemy without causing trouble", per http://petitcolas.net/fabien/kerckhoffs/). Kerckhoffs said neither "publish everything" nor "keep everything secret"; rather, he said that the system should still be secure *even if the enemy has a copy*.

In other words — design your system assuming that your opponents know it in detail. (A former official at NSA's National Computer Security Center told me that the standard assumption there was that serial number 1 of any new device was delivered to the Kremlin.) After that, though, there's nothing wrong with trying to keep it secret — it's another hurdle factor the enemy has to overcome. (One obstacle the British ran into when attacking the German Engima system was simple: they didn't know the unkeyed mapping between keyboard keys and the input to the rotor array.) But — *don't rely on secrecy*.

http://catless.ncl.ac.uk/Risks/25.71.html#subj19

Steven M. Bellovin

professor in Columbia University

Fellow at AT&T Labs Research

Network Working Group
Request for Comments: 1948
Category: Informational

S. Bellovin
AT&T Research

May 1996

Defending Against Sequence Number Attacks

Status of This Memo

This memo provides information for the Internet community. This memo does not specify an Internet standard of any kind. Distribution of this memo is unlimited.

Abstract

IP spoofing attacks based on sequence number spoofing have become a serious threat on the Internet (CERT Advisory CA-95:01). While ubiquitous crypgraphic authentication is the right answer, we propose a simple modification to TCP implementations that should be a very substantial block to the current wave of attacks.

Overview and Rational

In 1985, Morris [1] described a form of attack based on guessing what sequence numbers TCP [2] will use for new connections. Briefly, the attacker gags a host trusted by the target, impersonates the IP address of the trusted host when talking to the target, and completes the 3-way handshake based on its guess at the next initial sequence number to be used. An ordinary connection to the target is used to gather sequence number state information. This entire sequence, coupled with address-based authentication, allows the attacker to execute commands on the target host.

Clearly, the proper solution is cryptographic authentication [3,4]. But it will quite a long time before that is deployed. It has therefore been necessary for many sites to restrict use of protocols that rely on address-based authentication, such as rlogin and rsh. Unfortunately, the prevalence of "sniffer attacks" -- network eavesdropping (CERT Advisory CA-94:01) -- has rendered ordinary TELNET [5] very dangerous as well. The Internet is thus left without a safe, secure mechanism for remote login.

The obvious way to do this is to maintain state for dead connections, and the easiest way to do that is to change the TCP state transition diagram so that both ends of all connections go to TIMEWAIT state. That would work, but it's inelegant and consumes storage space. Instead, we use the current 4 microsecond timer M and set

```
ISN = M + F(localhost, localport, remotehost, remoteport).
```


It is vital that F not be computable from the outside, or an attacker could still guess at sequence numbers from the initial sequence number used for some other connection. We therefore suggest that F be a cryptographic hash function of the connection-id and some secret data. MD5 [9] is a good choice, since the code is widely available. The secret data can either be a true random number [10], or it can be the combination of some per-host secret and the boot time of the machine. The boot time is included to ensure that the secret is

sage would be lost. Alternatively, one could wait until T was down for routine maintenance or a reboot.

I mischaracterized Morris' paper on this point. While flooding can work—without explicitly stating it, I anticipated the denial of service attacks that started occurring in 1996—Morris in fact exploited an implementation error in the Berkeley kernel to accomplish his goal with many fewer packets. That flaw (described in [10] as well as in Morris' paper) received very little attention at the time, and was not fixed until many years later.

For that matter, sequence number attacks received little attention outside of my paper, until Kevin Mitnick reimplemented Morris' idea and used it to attack Tsutomu Shimomura [93]. Shimomura then proceeded to track down Mitnick.

SYN Flood 攻击的真实案例

Technology | CYBERTIMES

The New York Times ON THE WEB

New York's Panix Service Is Crippled by Hacker Attack

1996年，提供反垃圾邮件服务的公司遭受SYN Flood攻击

By ROBERT E. CALEM

Public Access Networks Corporation, a Manhattan-based Internet service provider popularly known as Panix, wanted to shield its customers from the junk bulk e-mailers known as "spammers." So, two weeks ago Panix installed a system for blocking junk bulk e-mail to its users -- an effort similar to the one that will send America Online to court in November.

But now the company is facing a new threat to its users that may have no solution, short of pulling the plug on Panix itself, experts said Friday.

Beginning Sept. 6 and continuing through at least last Tuesday, a hacker intent on shutting Panix down successfully did just that, by bombarding the service provider's servers with a flood of phony connection requests that prevented real requests by legitimate customers from getting through.

Speculation about the attacker's motive has focused on the company's newly installed system for locking out bulk e-mail spammers.

Okolo Schwinn-Clanton, director of corporate services at Panix, said that two weeks ago the ISP created a list of junk bulk e-mailers and a program that allows customers to instruct the company's mail servers to block all incoming messages from any sender on that list. Use of the blocking feature is entirely voluntary, Schwinn-Clanton emphasized. By editing their personal list, Panix subscribers can block sources of other e-mail they find annoying or restore sources from which they want to receive mail.

Company Statement

Statement from Public Access Networks Corporation explaining to subscribers how the attack took place and how it affected service.

> **"In principle, most of the denial-of-service attacks we see have no solution."**
>
> Peter G. Neumann, SRI International

Moreover, Simona Nass, a spokeswoman for Panix, said that the ISP did not automatically add any bulk e-mailer to the blacklist. But if any source of bulk e-mail is identified as a spammer by numerous customers, she said, Panix will exercise due diligence by contacting both the sender and the Internet provider that hosts the sender and request that the mass mailings be stopped.

Only then, if the junk bulk e-mail continues to flow, Schwinn-Clanton said, will all mail from the host site be blocked at Panix's gates -- whether it's from the accused sender or some other customer of the host.

Nass conceded that this solution might not be fair to the host server, which in all likelihood will have a number of other customers, all of whose e-mail will be blocked from Panix.

On the other hand, she said it was also "very unfair" for Panix customers "to have to spend time weeding through stuff."

As of Friday, Schwinn-Clanton said, Panix had 15 hosts on its blacklist, up from one, Moneyworld.com, two weeks ago. The second host to join the list was Capital Area Internet Service, or CAIS, whose downstream clients include Cyber Promotions, the junk bulk-emailer that is now also the focus of AOL's legal team. (Cyber Promotions is hosted by ServInt Corporation of McLean, Va., which buys its bandwidth from CAIS.) Cyber Promotions and AOL will meet in November in the United States Court of Appeals for the Third Circuit in Philadelphia.

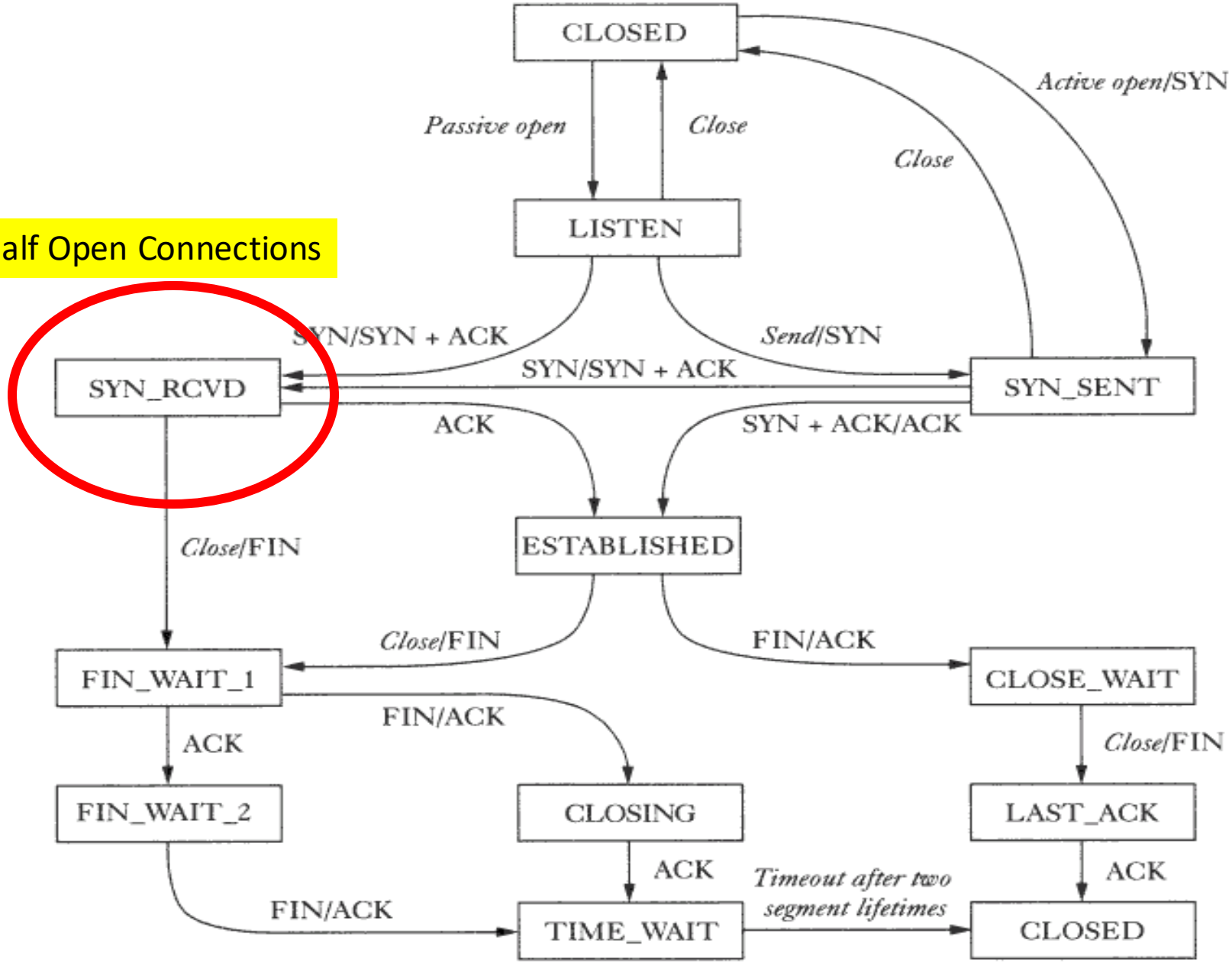
Whether Panix's strategy for blocking junk bulk e-mailers will be affected by the outcome of the AOL-Cyber Promotions case cannot be known now, said David Phillips, associate general counsel for America Online in Vienna, Va. "If the court held AOL did not have the right to block," then it could have "implications" for Panix, Phillips said.

Related Article

Judge Prevents AOL From Blocking E-Mail (September 7)

Online Service Blocks 'Junk' E-Mail Aimed at

Half Open Connections



maximum segment lifetime(MSL, 2munites)

关于SYN Flood 攻击的描述

A Common TCP Bug

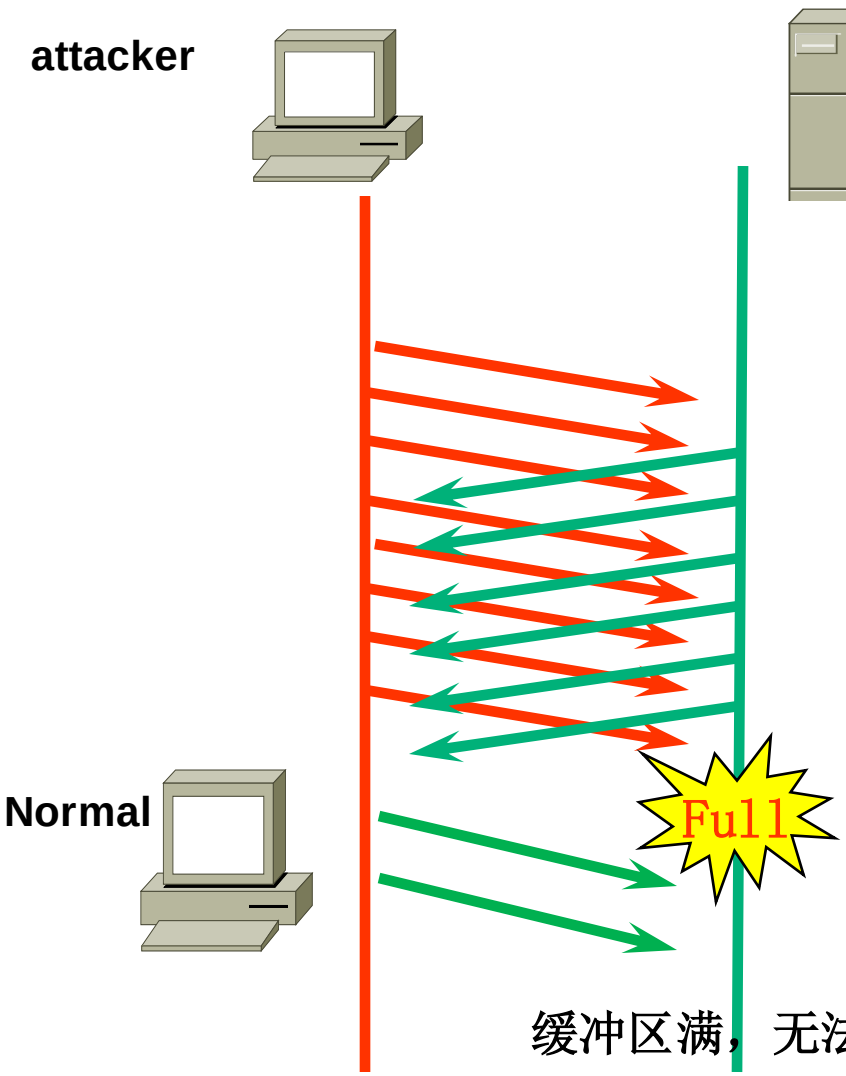
As mentioned earlier, attackers using sequence number guessing have to "gag" the trusted machine first. While a number of strategies are possible, most of the attacks detected thus far rely on an implementation bug.

When SYN packets are received for a connection, the receiving system creates a new TCB in SYN-RCVD state. To avoid overconsumption of resources, 4.2BSD-derived systems permit only a limited number of TCBs in this state per connection. Once this limit is reached, future SYN packets for new connections are discarded; it is assumed that the client will retransmit them as needed.

When a packet is received, the first thing that must be done is a search for the TCB for that connection. If no TCB is found, the kernel searches for a "wild card" TCB used by servers to accept connections from all clients. Unfortunately, in many kernels this code is invoked for any incoming packets, not just for initial SYN packets. If the SYN-RCVD queue is full for the wildcard TCB, any new packets specifying just that host and port number will be discarded, even if they aren't SYN packets.

https://datatracker.ietf.org/doc/html/rfc1948

SYN Flood Attack



SYN RCVD (Half-Open 状态)
TCB Queue exhausted !

The half-open connections data structure will eventually fill;
then the system will be unable to accept any new incoming connections until the table is emptied out.
However, the attacking system can simply continue sending IP-spoofed packets requesting new connections faster than the victim system can expire the pending connections.

缓冲区满，无法接收新的请求
没有任何回复

美国CERT/CC发布安全公告

Pages in the Historical section of this site are provided for historical purposes, they are no longer maintained. Links may not work.

CERT Advisory CA-1996-21 TCP SYN Flooding and IP Spoofing Attacks

由 Allen D. Householder创建于2021-10-07

Original issue date: September 19, 1996
Last revised: November 29, 2000
Updated vendor information for the Linux kernel.

A complete revision history is at the end of this file. **This advisory supersedes the IP spoofing portion of CA-95.01.**

Two "underground magazines" have recently published code to conduct denial-of-service attacks by creating TCP "half-open" connections. This code is actively being used to attack sites connected to the Internet. There is, as yet, no complete solution for this problem, but there are steps that can be taken to lessen its impact. Although discovering the origin of the attack is difficult, it is possible to do; we have received reports of attack origins being identified.

Any system connected to the Internet and providing TCP-based network services (such as a Web server, FTP server, or mail server) is potentially subject to this attack. Note that in addition to attacks launched at specific hosts, these attacks could also be launched against your routers or other network server systems if these hosts enable (or turn on) other TCP services (e.g., echo). The consequences of the attack may vary depending on the system; however, the attack itself is fundamental to the TCP protocol used by all systems.

If you are an Internet service provider, please pay particular attention to Section III and Appendix A, which describes step we urge you to take to lessen the effects of these attacks. If you are the customer of an Internet service provider, please encourage your provider to take these steps.

This advisory provides a brief outline of the problem and a partial solution. We will update this advisory as we receive new information. If the change in information warrants, we may post an updated advisory on comp.security.announce and redistribute an update to our cert-advisory mailing list. As always, the latest information is available at the URLs listed at the end of this advisory.

I. Description

When a system (called the client) attempts to establish a TCP connection to a system providing a service (the server), the client and server exchange a set sequence of messages. This connection technique applies to all TCP connections--telnet, Web, email, etc.

The client system begins by sending a SYN message to the server. The server then acknowledges the SYN message by sending SYN-ACK message to the client. The client then finishes establishing the connection by responding with an ACK message. The connection between the client and the server is then open, and the service-specific data can be exchanged between the client and the server. Here is a view of this message flow:

```
        Client                  Server
          ------                  ------
```

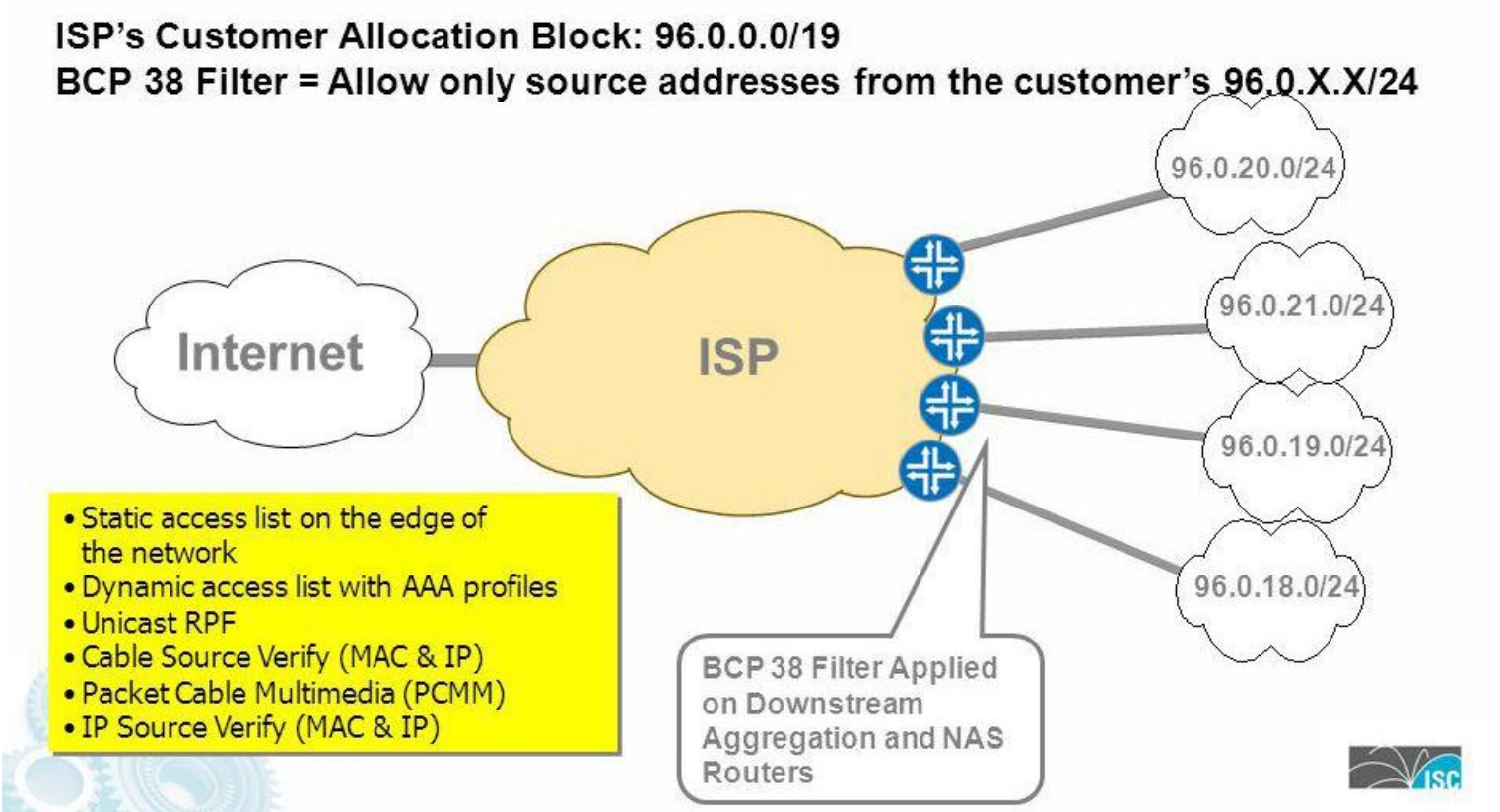
III. Solution

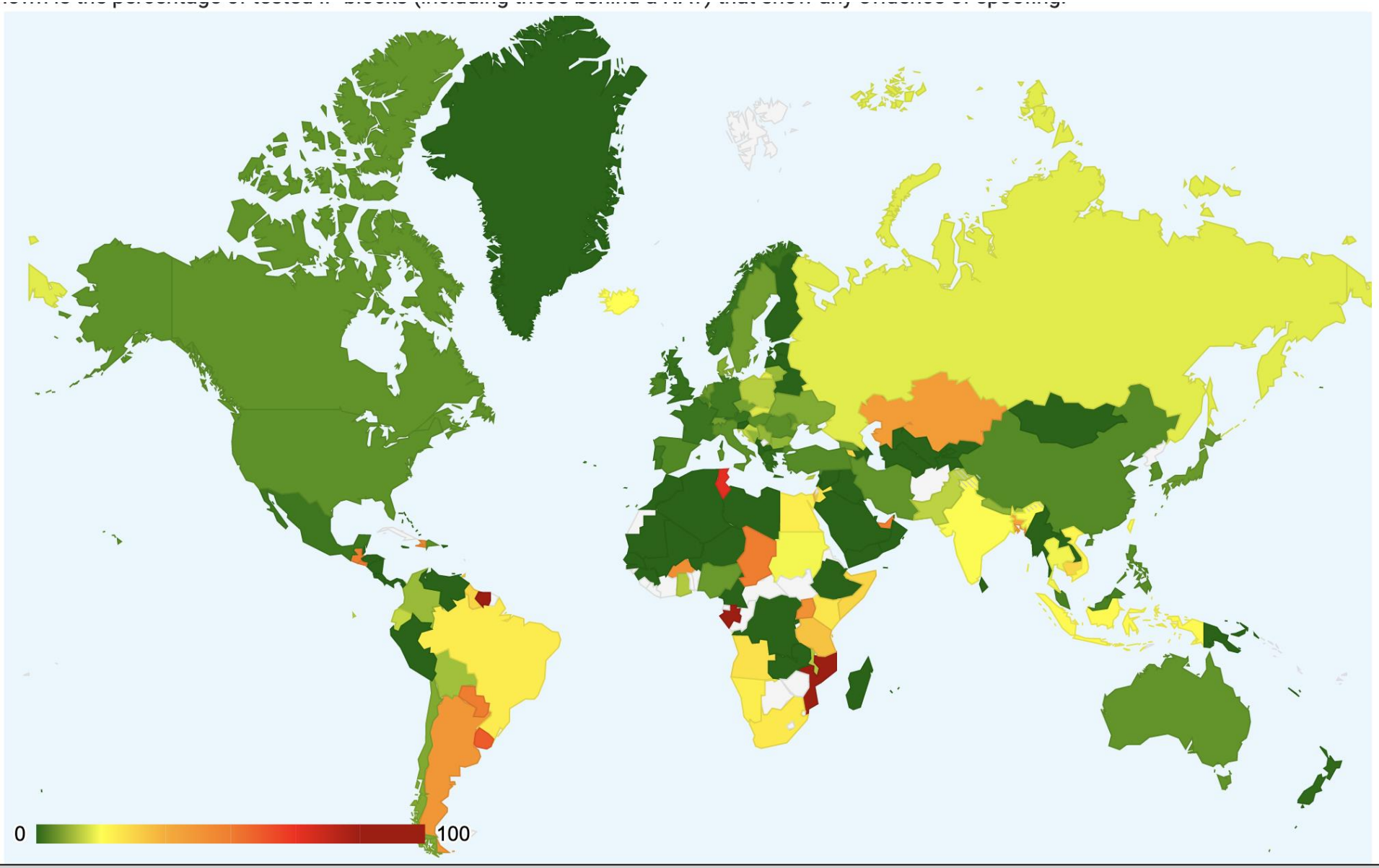
There is, as yet, **no generally accepted solution to this problem** with the current IP protocol technology.

The potential for abuse arises at the point where the server system has sent an acknowledgment (SYN-ACK) back to client but has not yet received the ACK message. This is what we mean by half-open connection. The server has built in its system memory a data structure describing all pending connections. This data structure is of finite size, and it can be made to overflow by intentionally creating too many partially-open

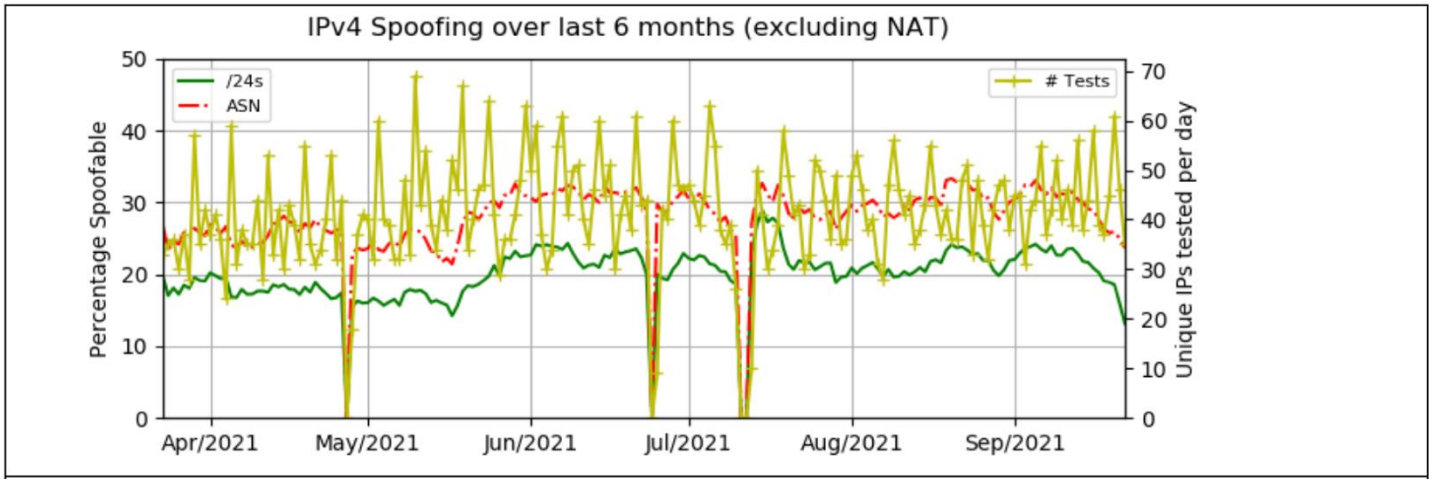
Solutions to SYN flooding

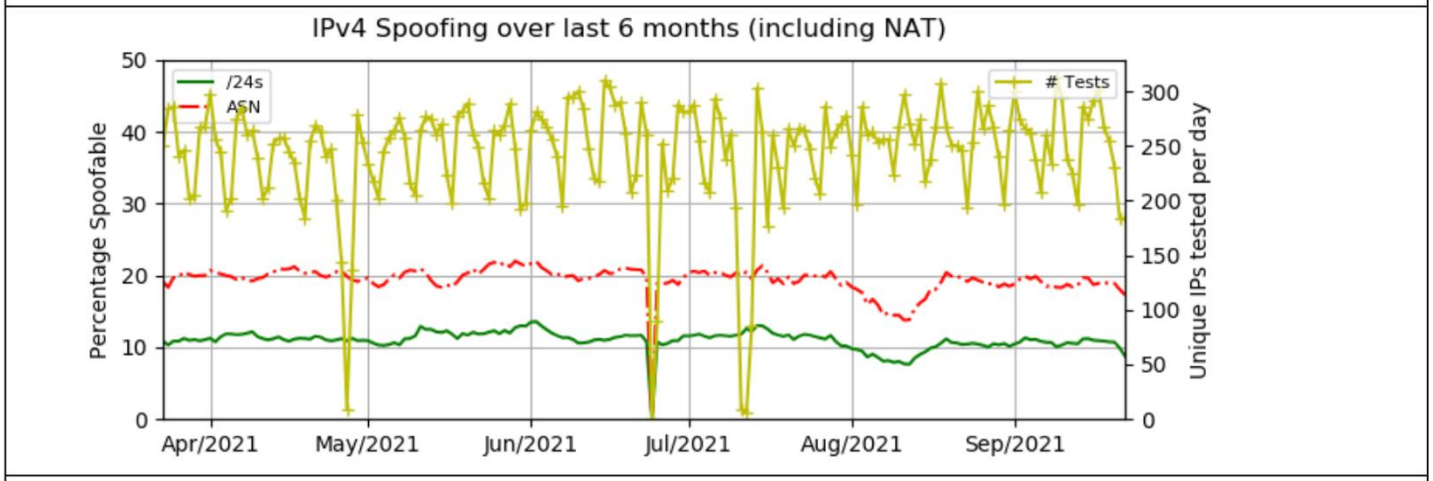
- **RFC 4987, TCP SYN Flooding Attacks and Common Mitigations, 2007**
  - **Filtering : BCP 38(RFC 2827, Ingress Filtering)**

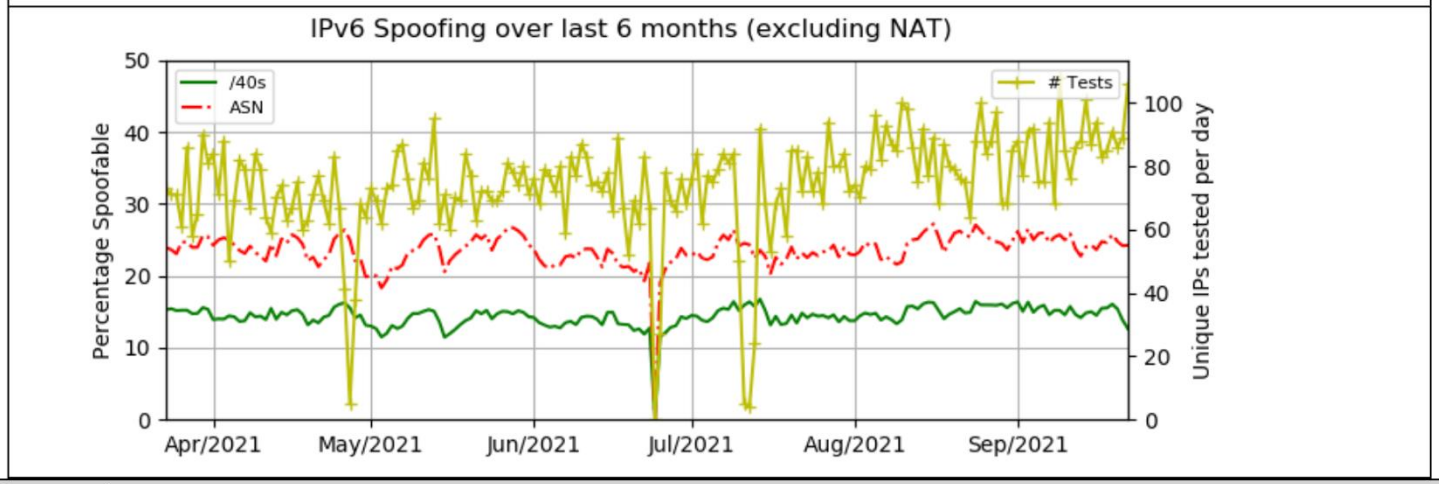




https://spoofer.caida.org/summary.php


IPv4 Spoofing over last 6 months (excluding NAT)
Percentage Spoofable
Unique IPs tested per day
/24s
ASN
Tests
0
10
20
30
40
50
0
10
20
30
40
50
60
70
Apr/2021
May/2021
Jun/2021
Jul/2021
Aug/2021
Sep/2021


IPv4 Spoofing over last 6 months (including NAT)
Percentage Spoofable
Unique IPs tested per day
/24s
ASN
Tests
0
10
20
30
40
50
0
50
100
150
200
250
300
Apr/2021
May/2021
Jun/2021
Jul/2021
Aug/2021
Sep/2021


IPv6 Spoofing over last 6 months (excluding NAT)
Percentage Spoofable
Unique IPs tested per day
/40s
ASN
Tests
0
10
20
30
40
50
0
20
40
60
80
100
Apr/2021
May/2021
Jun/2021
Jul/2021
Aug/2021
Sep/2021

Solutions to SYN flooding

- **RFC 4987, TCP SYN Flooding Attacks and Common Mitigations, 2007**
  - **Filtering (e.g. BCP 38 against spoofing)**
  - **Increasing Backlog (half-open queue)**
  - **Reducing SYN-RECEIVED Timer**

```
#define TCP_SYN_RETRIES  6      /* This is how many retries are done
                                 * when active opening a connection.
                                 * RFC1122 says the minimum retry MUST
                                 * be at least 180secs.  Nevertheless
                                 * this value is corresponding to
                                 * 63secs of retransmission with the
                                 * current initial RTO.
                                 */

#define TCP_SYNACK_RETRIES 5    /* This is how may retries are done
                                 * when passive opening a connection.
                                 * This is corresponding to 31secs of
                                 * retransmission with the current
                                 * initial RTO.
                                 */
```

Solutions to SYN flooding

- **RFC 4987, TCP SYN Flooding Attacks and Common Mitigations, 2007**
  - **Filtering (e.g. BCP 38 against spoofing)**
  - **Increasing Backlog (half-open queue)**
  - **Reducing SYN-RECEIVED Timer**
  - **Recycling the Oldest Half-Open TCB**
  - **SYN Cache**
  - **SYN Cookies**
  - **Hybrid Approaches**

Daniel J. Bernstein

18 languages

Article Talk | Read Edit View history Tools

From Wikipedia, the free encyclopedia

For the American businessman and activist, see Daniel J. Bernstein (businessman).

Daniel Julius Bernstein (sometimes known as **djb**; born October 29, 1971) is an American mathematician, cryptologist, and computer scientist. He is a visiting professor at CASA[2] at Ruhr University Bochum, as well as a research professor of Computer Science at the University of Illinois at Chicago. Before this, he was a visiting professor in the department of mathematics and computer science at the Eindhoven University of Technology.[*citation needed*]

Early life [edit]

Bernstein attended Bellport High School, a public high school on Long Island, graduating in 1987 at the age of 15.[3] The same year, he ranked fifth in the Westinghouse Science Talent Search.[4] In 1987 (at the age of 16), he achieved a Top 10 ranking in the William Lowell Putnam Mathematical Competition,[5] and was a member of the second–place team from Princeton University the following year.[6] Bernstein earned a B.A. in mathematics from New York University (1991) and a Ph.D. in mathematics from the University of California, Berkeley (1995), where he studied under Hendrik Lenstra.[*citation needed*]

Bernstein v. United States [edit]

The export of cryptography from the United States was controlled as a munition starting from the Cold War until recategorization in 1996, with further relaxation in the late 1990s.[7] In 1995, Bernstein brought the court case *Bernstein v. United States*. The ruling in the case declared that software was protected speech under the First Amendment, which contributed to regulatory changes reducing controls on encryption.[8] Bernstein was originally represented by the Electronic Frontier Foundation.[9] He later represented himself.[10]

Daniel J. Bernstein

Born	October 29, 1971 (age 53) East Patchogue, New York[1]
Citizenship	American, German[1]
Alma mater	University of California, Berkeley New York University
Known for	qmail, djbdns, Salsa20, ChaCha20, Poly1305, Curve25519
Scientific career	
Fields	Mathematics, Cryptography, Computer Security
Institutions	University of Illinois at Chicago, Eindhoven University of Technology, Ruhr University Bochum
Doctoral	Hendrik Lenstra

SYN Cookie by D. J. Bernstein

SYN cookies

Mail service for Panix, an ISP in New York, was shut down by a SYN flood starting on 6 September 1996. A week later the story was covered by the RISKS Digest, the Wall Street Journal, the Washington Post, and many other newspapers.

SYN flooding had been considered by security experts before. It was generally considered insoluble. See, for example, ``Practical UNIX and Internet Security,'' by Garfinkel and Spafford, page 778:

> The recipient will be left with multiple half-open connections that are occupying limited resources. Usually, these connection requests have forged source addresses that specify nonexistent or unreachable hosts that cannot be contacted. Thus, there is also no way to trace the connections back. ... There is little you can do in these situations. ... any finite limit can be exceeded.

Large SYN queues and random early drops make SYN flooding more expensive but don't actually solve the problem.

SYN cookies use cryptographic techniques to solve the problem. I pointed out how easy this was on 16 September 1996; Eric Schenk and I worked out the gory details over the next few weeks. Jeff Weisberg released a SunOS implementation in October 1996, and Eric Schenk released a Linux implementation in February 1997.

SYN cookies are now a standard part of Linux and FreeBSD. They are, unfortunately, not enabled by default under Linux. To enable them, add

```
echo 1 > /proc/sys/net/ipv4/tcp_syncookies
```

to your boot scripts.

What are SYN cookies?

SYN cookies are particular choices of initial TCP sequence numbers by TCP servers. The difference between the server's initial sequence number and the client's initial sequence number is

- top 5 bits: t mod 32, where t is a 32-bit time counter that increases every 64 seconds;
- next 3 bits: an encoding of an MSS selected by the server in response to the client's MSS;
- bottom 24 bits: a server-selected secret function of the client IP address and port number, the server IP address and port number, and t.

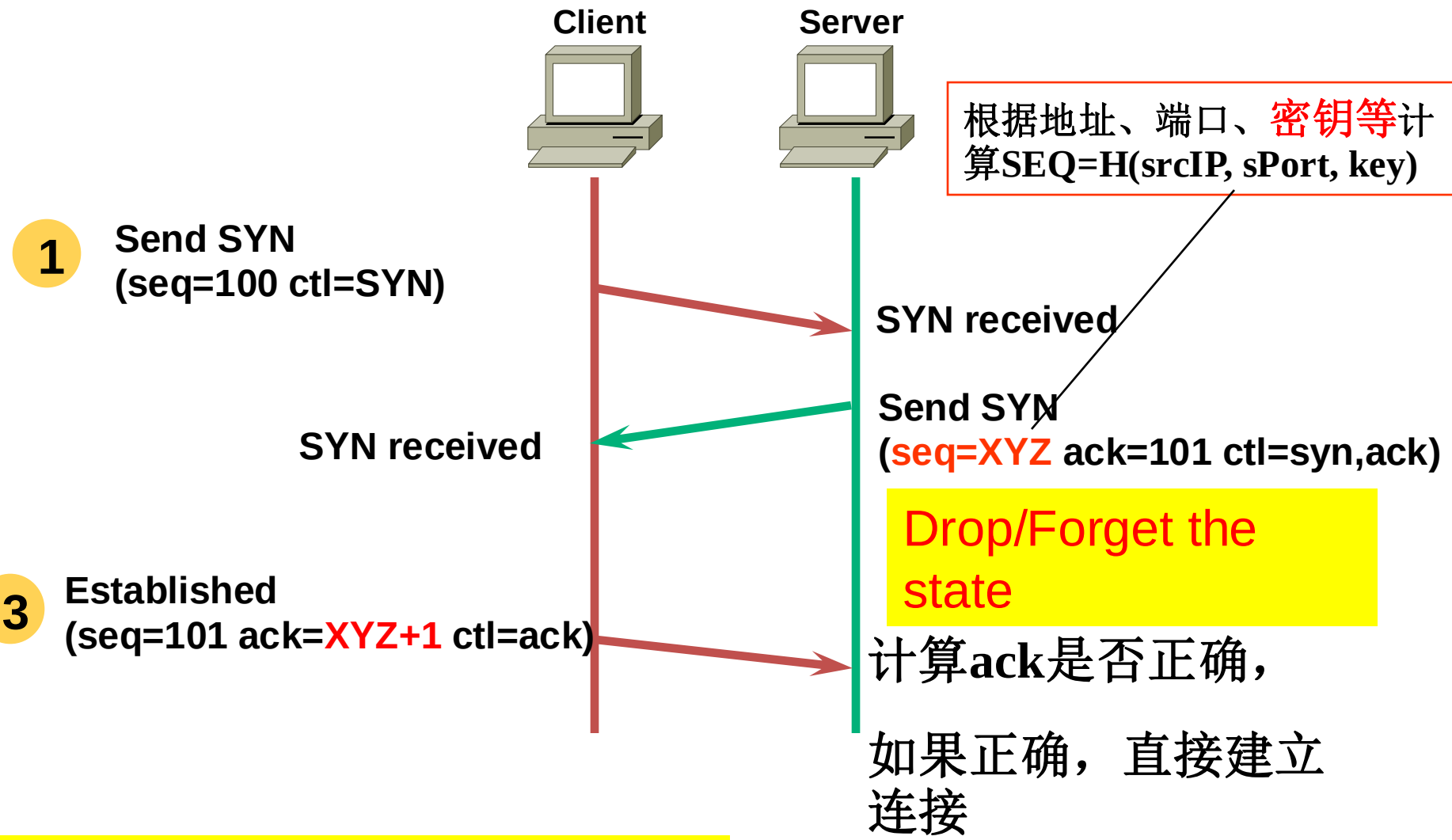
This choice of sequence number complies with the basic TCP requirement that sequence numbers increase slowly; the server's initial sequence number increases slightly faster than the client's initial sequence number.

A server that uses SYN cookies doesn't have to drop connections when its SYN queue fills up. Instead it sends back a SYN+ACK, exactly as if the SYN queue had been larger. (Exceptions: the server must reject TCP options such as large windows, and it must use one of the eight MSS values that it can encode.) When the server receives an ACK, it checks that the secret function works for a recent value of t, and then rebuilds the SYN queue entry from the encoded MSS.

A SYN flood is simply a series of SYN packets from forged IP addresses. The IP addresses are chosen randomly and don't provide any hint of where the attacker is. The SYN flood keeps the server's SYN queue full. Normally this would force the server to drop connections. A server that uses SYN cookies, however, will continue operating normally. The biggest effect of the SYN flood is to disable large windows.

https://cr.yp.to/syncookies.html

SYN Cookie, **Daniel J. Bernstein, http://cr.yp.to/syncookies.html**



echo 1 > /proc/sys/net/ipv4/tcp_syncookies

2007年，防范的建议

INFORMATIONAL

Network Working Group
Request for Comments: 4987
Category: Informational

W. Eddy
Verizon
August 2007

TCP SYN Flooding Attacks and Common Mitigations

Status of This Memo

This memo provides information for the Internet community. It does not specify an Internet standard of any kind. Distribution of this memo is unlimited.

Abstract

This document describes TCP SYN flooding attacks, which have been well-known to the community for several years. Various countermeasures against these attacks, and the trade-offs of each, are described. This document archives explanations of the attack and common defense techniques for the benefit of TCP implementers and administrators of TCP servers or networks, but does not make any standards-level recommendations.

Table of Contents

1. Introduction 2
2. Attack Description 2
2.1. History 3
2.2. Theory of Operation 3
3. Common Defenses 6
3.1. Filtering 6
3.2. Increasing Backlog 7
3.3. Reducing SYN-RECEIVED Timer 7

Maintain a counter that increases slowly over time and never repeats, such as "number of seconds since 1970, shifted right 6 bits".

When a SYN comes in from (saddr,sport) to (daddr,dport) with ISN x, find the largest i for which msstab[i] <= the incoming MSS. Compute

```
z = MD5(sec1,saddr,sport,daddr,dport,sec1)
    + x
    + (counter << 24)
    + (MD5(sec2,counter,saddr,sport,daddr,dport,sec2) % (1 << 24))
```

and then

```
y = (i << 29) + (z % (1 << 29))
```

Create a TCB as usual, with y as our ISN. Send back a SYNACK.

2012年，RFC6528更新RFC793()

tools.ietf.org/html/rfc6528#page-4

[Docs] [txt|pdf] [draft-ietf-tcpm...] [Tracker] [Diff1] [Diff2]

PROPOSED STANDARD

Internet Engineering Task Force (IETF) — F. Gont
Request for Comments: 6528 — SI6 Networks / UTN-FRH
Obsoletes: 1948 — S. Bellovin
Updates: 793 — Columbia University
Category: Standards Track — February 2012
ISSN: 2070-1721

Defending against Sequence Number Attacks

Abstract

This document specifies an algorithm for the generation of TCP Initial Sequence Numbers (ISNs), such that the chances of an off-path attacker guessing the sequence numbers in use by a target connection are reduced. This document revises (and formally obsoletes) RFC 1948, and takes the ISN generation algorithm originally proposed in that document to Standards Track, formally updating RFC 793.

each space, the ISN is incremented according to [RFC0793]; however, there is no obvious relationship between the numbering in different spaces.

An obvious way to prevent sequence number guessing attacks while not breaking the 4.4BSD heuristics would be to perform a simple random selection of TCP ISNs while maintaining state for dead connections (e.g. changing the TCP state transition diagram so that both end-points of all connections go to TIME-WAIT state). That would work but would consume system memory to store the additional state. Instead, we propose an improvement to the TCP ISN generation algorithm that does not require TCP to keep state for all recently terminated connections.

3. Proposed Initial Sequence Number Generation Algorithm

TCP SHOULD generate its Initial Sequence Numbers with the expression:

ISN = M + F(localip, localport, remoteip, remoteport, secretkey)

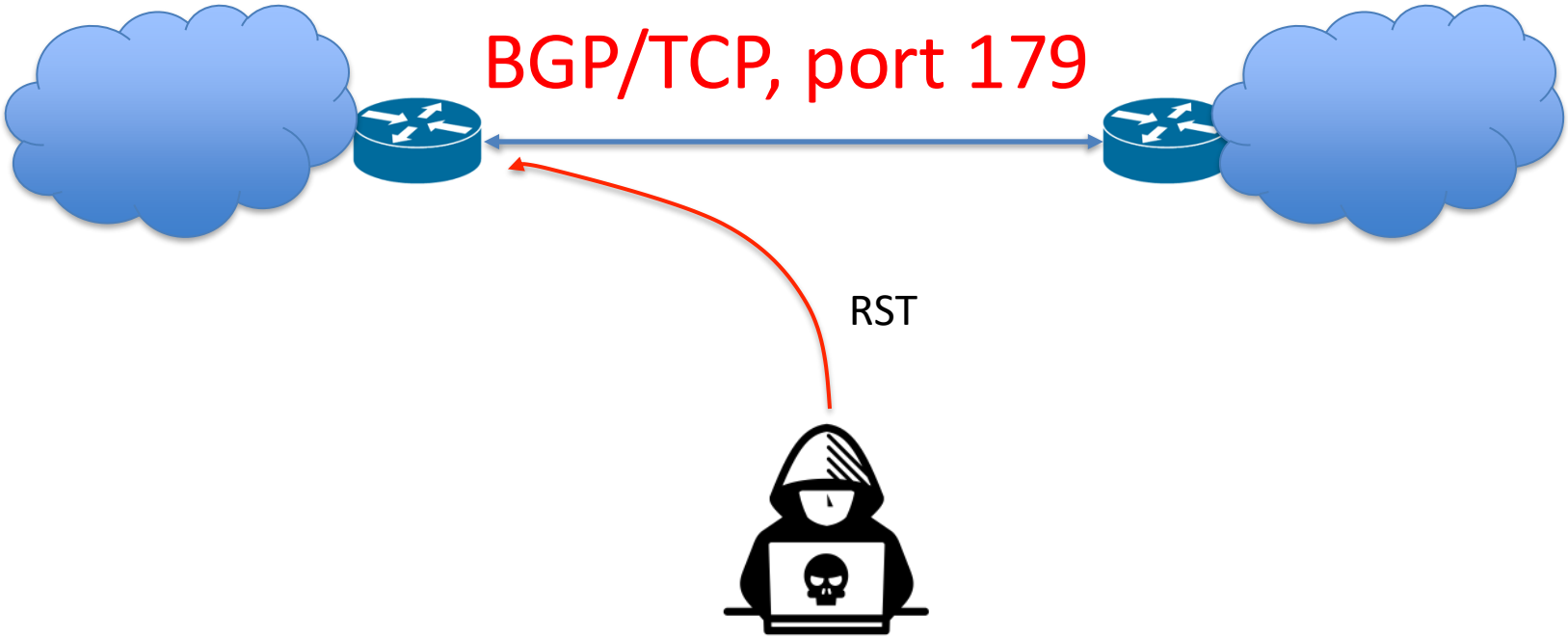
where M is the 4 microsecond timer, and F() is a pseudorandom function (PRF) of the connection-id. F() MUST NOT be computable from the outside, or an attacker could still guess at sequence numbers from the ISN used for some other connection. The PRF could be implemented as a cryptographic hash of the concatenation of the connection-id and some secret data; MD5 [RFC1321] would be a good choice for the hash function.

The result of F() is no more secure than the secret key. If an attacker is aware of which cryptographic hash function is being used by the victim (which we should expect), and the attacker can obtain enough material (i.e., ISNs selected by the victim), the attacker may simply search the entire secret-key space to find matches. To

基础协议演进的漫长过程

- 1981年发布TCP标准
- 1989年Robert Morris提出SYN Flood 攻击Idea
- 1996年D.J. Bernstein 提出防御方法
- 2012年RFC 6528 修改协议标准

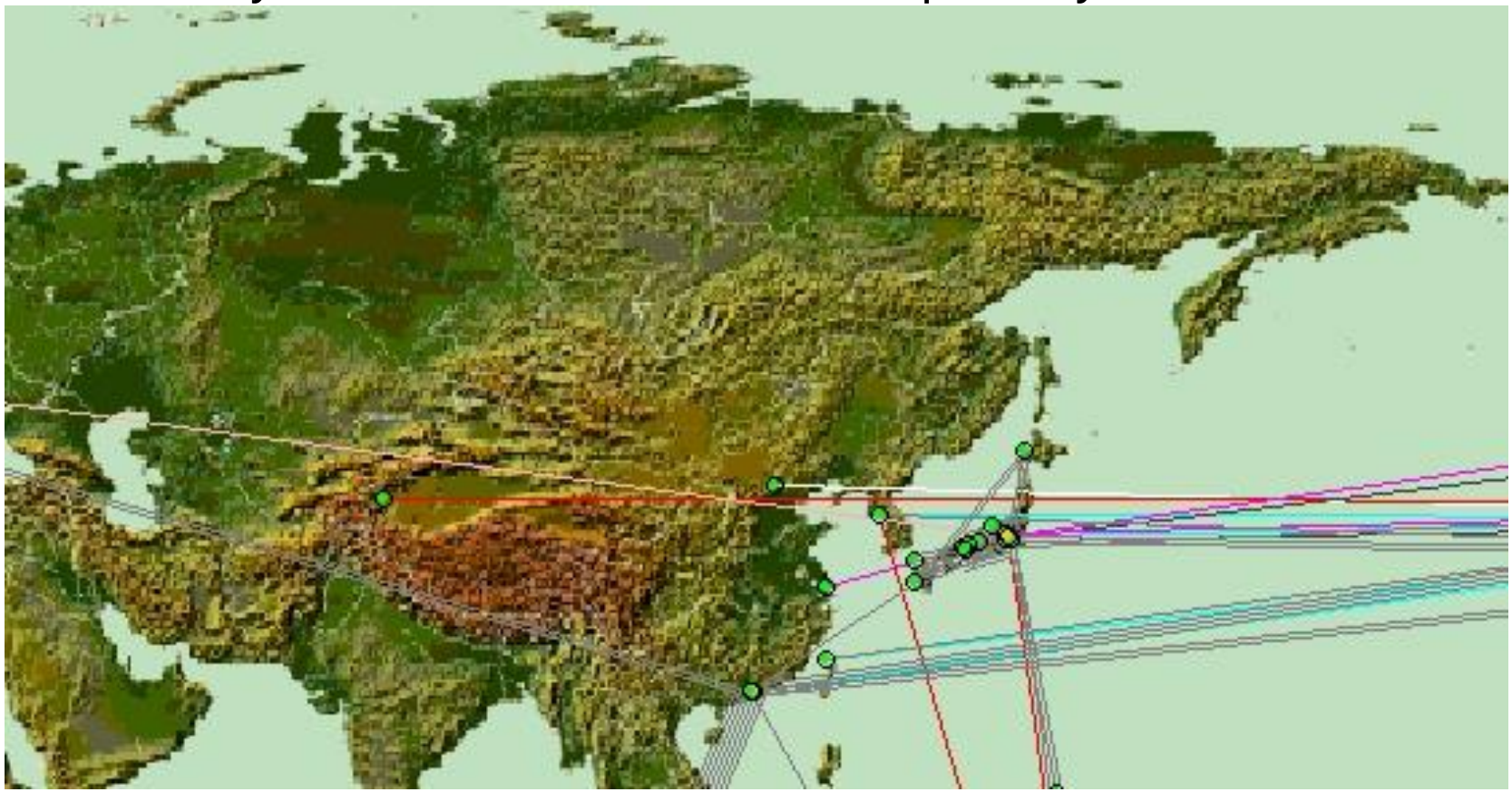
Blind RST



BGP as the target of TCP RST

http://www.caida.org/tools/visualization/mapnet/Backbones/

An analysis of backbone routes, shows a very limited amount of connectivity to China. This would be exceptionally vulnerable.



TCP Resets and Sequence Number Guessing

- Successful TCP resets require a valid 4-tuple and sequence number (not ttl)
- TCP Test Tool (ttt) is able to generate messages easily assuming local access to the wire:

```
18:22:59.328544 99.0.0.3.179 > 99.0.0.5.32324: P
272350230:272350249(19) ack 4142958006 win 15531: BGP
(KEEPALIVE) [tos 0xc0]  [ttl 1]
18:22:59.527079 99.0.0.5.32324 > 99.0.0.3.179: . ack
272350249 win 15543 [tos 0xc0]  [ttl 1]

# ./ttt -T 2 -D 99.0.0.5 -S 99.0.0.3 -x 179 -y 32324 -fR
-s 272350249

May  1 18:23:13.425: %BGP-5-ADJCHANGE: neighbor 99.0.0.3
Down Peer closed the session
```

- Nothing new here. Tcpkill (from dsniff) works, too.

Can you believe what you read?

If you can believe what you read…

- BGP is…highly vulnerable to a variety of attacks due to the lack of a scalable means of verifying the authenticity and authorization of BGP control traffic. - S-BGP Website[1]
- Any outsider can inject believable BGP messages into the communication between BGP peers and thereby inject bogus routing information or break the peer to peer connection. - draft-murphy-bgp-vuln-02.txt[2]
- outsider sources can also disrupt communications between BGP peers by breaking their TCP connection with spoofed RST packets. - draft-murphy-bgp-protect-01.txt[3]
- The border gateway protocol…is rife with security holes and needs to be replaced, a security consultant warned. - news.com[4]

FUD(Fear, Uncertainty, and Doubt)

Sean Convery, Matthew Franz@cisco.com
BGP Vulnerability Testing: Separating Fact from FUD v1.00, Blackhat 2003

TCP Resets Results

- The peer is fully reestablished in 50 seconds (test network) - several minutes (production network):

```
May 1 18:24:50: %BGP-5-ADJCHANGE: neighbor
   99.0.0.5 Up
```

- Various research [12], and [13] have found flaws in some implementations of TCP ISN selection. This should be a solved problem for most implementations though (did not repeat tests).
  - This research depends upon access to a range of initial sequence numbers from the router (we can prevent this with BCPs).
- If implementations went with pseudo-random source ports the number space moves from 2^{32} to 2^{48}.

初始序列号预测攻击问题已经解决了

可行的方法只有暴力猜测

TCP Resets Time Requirements

- A theoretical blind attack @ 1 million pps ~ 30 minutes to just the seq. number (assuming a correct guess after iterating through 50% of the space).

  $(2^{32}/2)/1{,}000{,}000$ = # of seconds

- Our tool was able to generate 62,500pps* ~ 9 hours
- Since the attacker won't know which side is 179 vs. a high port multiply these numbers by 2.
- With source port randomization, this goes to 4 years in the first example (1 mil. pps to guess 1 48 bit number and 142 years assuming 62,500pps and needing to guess both sides):

  $((2^{48}/2)/62{,}500)x2$ = # of seconds

*What sort of event is 62.5kpps on *your* router?

TCP Resets Results

- The peer is fully reestablished in 50 seconds (test network) - several minutes (production network):

```
May 1 18:24:50: %BGP-5-ADJCHANGE: neighbor
   99.0.0.5 Up
```

- Various research [12], and [13] have found flaws in some implementations of TCP ISN selection. This should be a solved problem for most implementations though (did not repeat tests).
  - This research depends upon access to a range of initial sequence numbers from the router (we can prevent this with BCPs).
- If implementations went with pseudo-random source ports the number space moves from 2^{32} to 2^{48}.

TCP Reset Conclusions

Ingress Filtering

- Blind TCP seq. guessing is operationally impossible with a router using BCPs –with proper RFC 2827[14] filtering, the packet won't even reach the destination.
- Even without BCPs, this is quite a lot of work for 50 seconds (up to 5 minutes?) of down-time
- A successful TCP reset attack would need to be constantly repeated to keep a session down and would need to be duplicated on many routers to cause substantial impact to the Internet's routing tables.
- Any TCP sequence number attack will require lots of packets potentially causing link saturation or other problems (routers should notice)

Slipping in the Window: TCP Reset Attacks

Paul (Tony) Watson
CISSP, CISM, CCSP, CCSE,
MCSE+Security, etc, etc…

paw@paw.org
www.terrorist.net

CanSecWest, 2004

https://research.google.com/pubs/PaulWatson.html

CBS interview with Paul (Tony) Watson : https://youtu.be/bo3iz5n3eyI

Facts: RFC-793 (TCP)

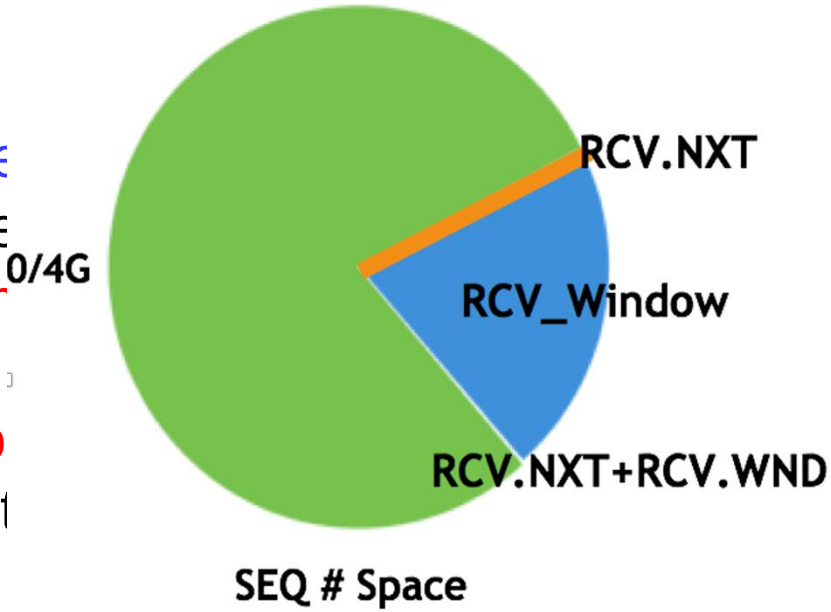
receive window

This represents the sequence numbers the local (receiving) TCP is willing to receive. Thus, the local TCP considers that segments overlapping the range RCV.NXT to RCV.NXT + RCV.WND - 1 carry acceptable data or control.

Segments containing sequence numbers entirely outside of this range are considered duplicates and discarded.

RST

A control bit (reset), occupying no se
the receiver should delete the conne
interaction. The receiver ***can*** deterr
number and acknowledgment fields
whether it should honor the reset co
case does receipt of a segment cont
in response.



Facts: RFC-793 (TCP)

receive window

This represents the sequence numbers the local (receiving) TCP is willing to receive. Thus, the local TCP considers that segments overlapping the range RCV.NXT to RCV.NXT + RCV.WND - 1 carry acceptable data or control.

Segments containing sequence numbers entirely outside of this range are considered duplicates and discarded.

RST

A control bit (reset), occupying no sequence space, indicating that the receiver should delete the connection without further interaction. The receiver ***can*** determine, based on the sequence number and acknowledgment fields of the incoming segment, whether it should honor the reset command or ignore it. In no case does receipt of a segment containing RST give rise to a RST in response.

Facts: RFC-793 Options

For a RST, a receiver **CAN** use sequence and acknowledgement to verify the packet. Unfortunately, **not a single TCP stack I have tested** verified the acknowledgement number. As a result, Reset attacks are easy to successfully execute….

理想与现实之间的差距..... 不是所有的实现都实现了标准的所有要求

Facts: Window size observations

Operating System	Initial Window Size	Packets Required
Efficient Networks 5861 (DSL Router) v5.3.20	4,096	1,048,575
Linux 2.4.18	5,840	735,439
Nokia IPSO 3.6-FCS6	16,384	262,143
Cisco 12.2(8)	16,384	262,143
Cisco 12.1(5)	16,384	262,143
Cisco 12.0(7)	16,384	262,143
Cisco 12.0(8)	16,384	262,143
Windows 2000 5.00.2195 SP1	16,384	262,143
Windows 2000 5.00.2195 SP3	16,384	262,143
HP-UX 11	32,768	131,071
Windows 2000 5.00.2195 SP4	64,512	66,576
Windows XP Home Edition SP1	64,240	66,858

攻击者需要发送的RST 个数： 2^32/2^16 = 2^16

Facts: Revised "FUD" Equation

- **At 1 million packets per second:**

(((4,294,967,295 / 1,000,000) / 16,384) / 60) /2 = .00218 (1/10 second)

  - **.00218 minutes (1/10th second)**

原来说需要30minutes

- **At 62,500 packets per second:**

(((4,294,967,295 / 62,500) / 16,384) / 60) / 2 = .0291 (approx. 1.7 seconds)

  - **.0291 minutes (1.7 seconds)**

原来说需要9hours

- **With TCP windows, we can reduce time requirements from 9 hours to 1.7 seconds (for 50% success rate)**

- **What kind of event is 60,000 packets for 1 second?**
  - **Would you even notice?**

Paul A. Watson, "Slipping In The Window: TCP Reset Attacks", CanSecWest 2004.

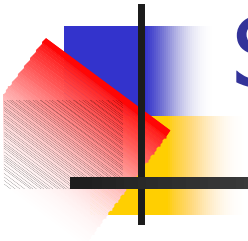
Live Testing: Results at 4,370pps

Target: Windows 2000	Start Time	Time of Reset	Time (sec.) Required
Test 1	180038	180046	8
Test 2	180724	180728	4
Test 3			10
Test 4			10
Test 5			11
Test 6			7
Test 7			10
Test 8	181539	181550	11
Test 9	202329	202332	3
Test 10	202410	202414	4

Anticipated Window Size	63,000 bytes
Packets Per Second	4,370

Targets

- BGP (the most obvious target)
- SSL, TLS, SSH (long lived connections, such as VPN's)
- IRC Servers (Netsplit anyone?)
- Domain Name System (DNS) over TCP
- Video Conferencing Systems
- SQL / Database Connections
- Remote Control (VNC, RDP, PCAnywhere)
- Online Games (Half-life tcp/27015, Diablo II tcp/4000, etc…)
- AOL (5190-5193)
- Others?



BGP Vulnerability Testing: Separating Fact from FUD v1.1

Sean Convery (sean@cisco.com)
Matthew Franz (mfranz@cisco.com)

Cisco Systems
Critical Infrastructure Assurance Group (CIAG)
http://www.cisco.com/go/ciag

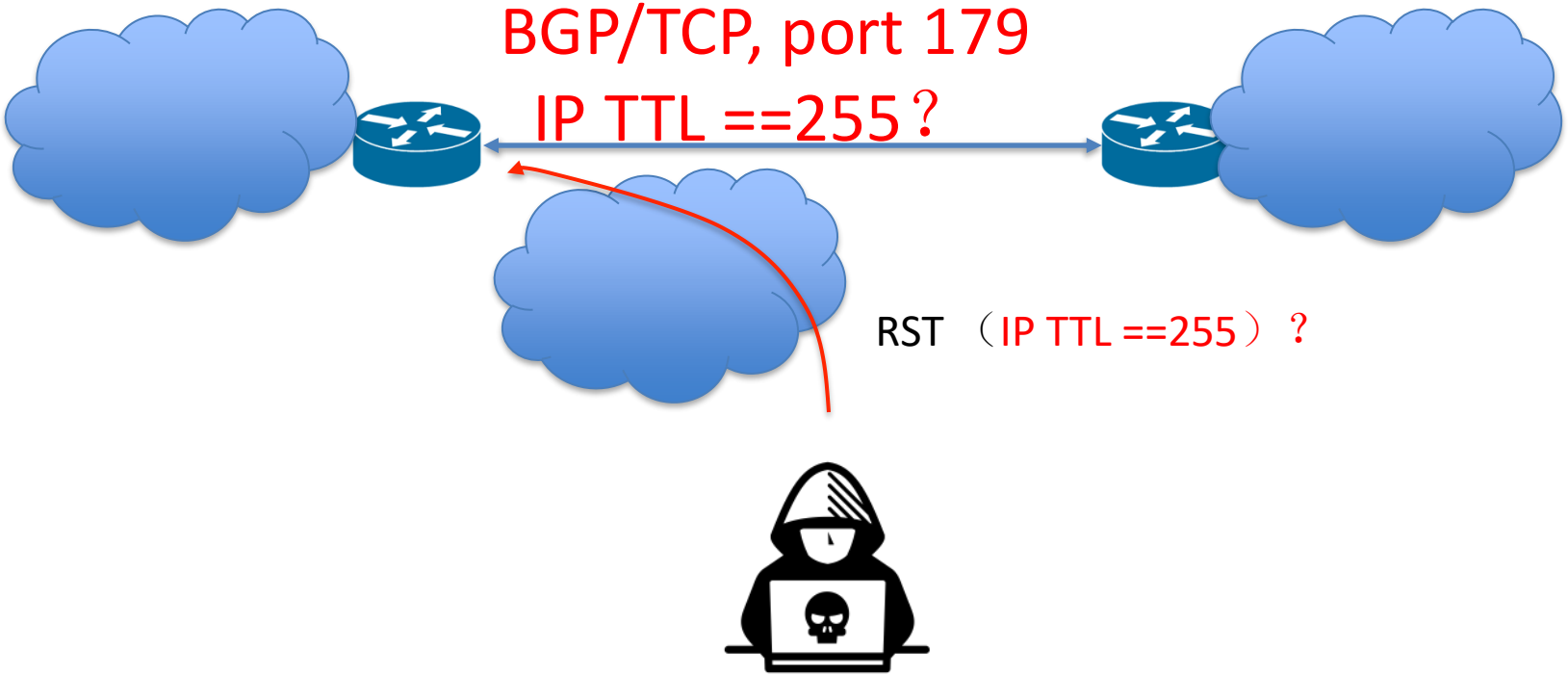
TCP Resets (1/2)

- Various research [12], and [13] have found flaws in some implementations of TCP ISN selection. This should be a solved problem for most implementations though (did not repeat tests).
- Recent research [24] has shown that the TCP window size significantly reduces the problem space to conduct a successful blind attack.
- draft-ietf-tcpm-tcpsecure-00.txt [25] describes new techniques for overcoming vulnerabilities due to the TCP window size in current TCP stacks.
  - The draft outlines an approach to increase their difficulty by implemented a challenge/response between client and server. These improvements have been implemented in shipping code from Cisco and Juniper and are under consideration by several other vendors.

Mitigation of TCP RST attacks

- RFC 1948(1996),RFC 6528(2012): Defending Against Sequence Number Attacks
- 1998, RFC 2385: Protection of BGP Sessions via the TCP MD5 Signature Option
- 2007, RFC 4953: Defending TCP Against Spoofing Attacks
- 2010, RFC5925: The TCP Authentication Option(Obsoletes: 2385)
- 2010, RFC 5961:Improving TCP's Robustness to Blind In-Window Attacks

TTL==255?

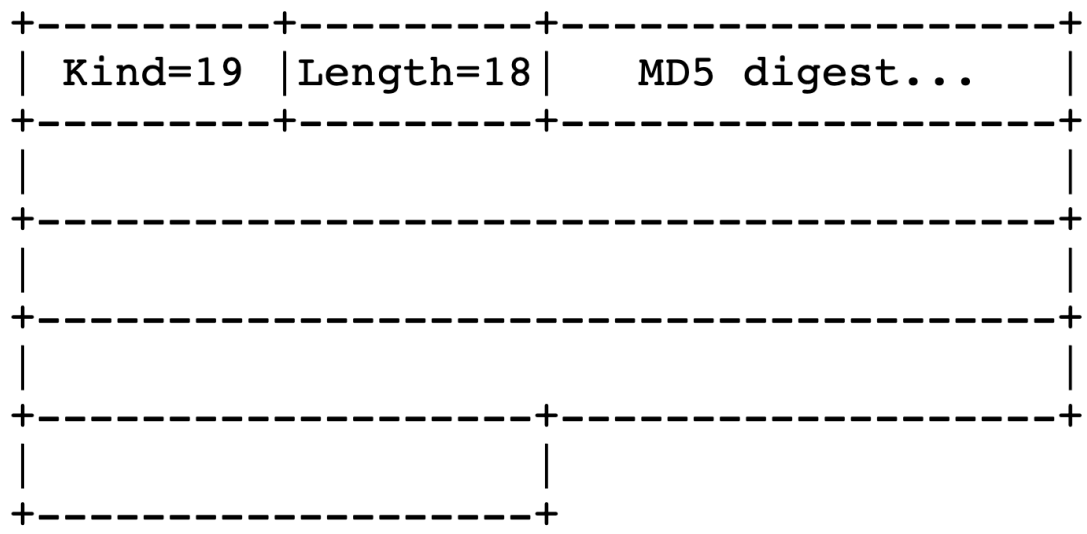


RFC 2385: Protection of BGP Sessions via the TCP MD5 Signature Option

3.0 Syntax

双方共享一个密钥

The proposed option has the following format:



The MD5 digest is always 16 bytes in length, and the option would appear in every segment of a connection.

Challenge ACK(RFC 5961, 2010)

RFC0793, 1981

1) If the RST bit is set and the sequence number is outside the current receive window (SEG.SEQ <= RCV.NXT || SEG.SEQ > RCV.NXT+ RCV.WND), silently drop the segment.

2) If the RST bit is set and the sequence number is acceptable, i.e., (RCV.NXT <= SEG.SEQ < RCV.NXT+RCV.WND), then reset the connection.

RFC 5961, 2010

1) If the RST bit is set and the sequence number is outside the current receive window, silently drop the segment.

2) If the RST bit is set and the sequence number exactly matches the next expected sequence number (RCV.NXT), then TCP MUST reset the connection.

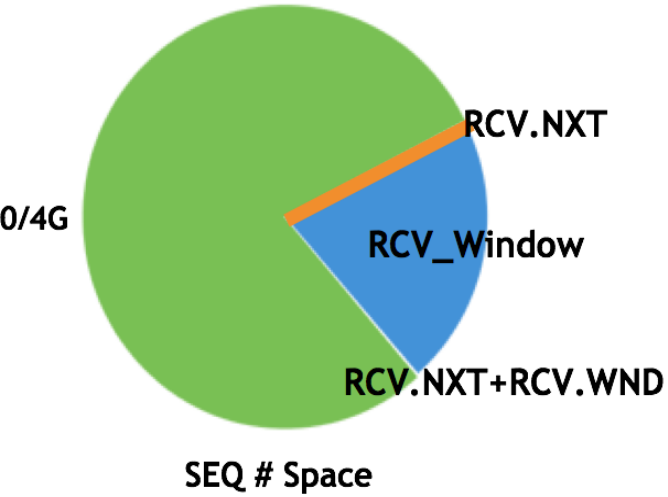
3) If the RST bit is set and the sequence number does not exactly match the next expected sequence value, yet is within the current receive window (RCV.NXT < SEG.SEQ < RCV.NXT+RCV.WND), TCP MUST send an acknowledgment (challenge ACK):

<SEQ=SND.NXT><ACK=RCV.NXT><CTL=ACK>

After sending the challenge ACK, TCP MUST drop the unacceptable segment and stop processing the incoming packet further. Further segments destined to this connection will be processed as normal.

RST Receiving Scheme

- Before RFC 5961: blind RST Attack by sending spoofed **RST** packet

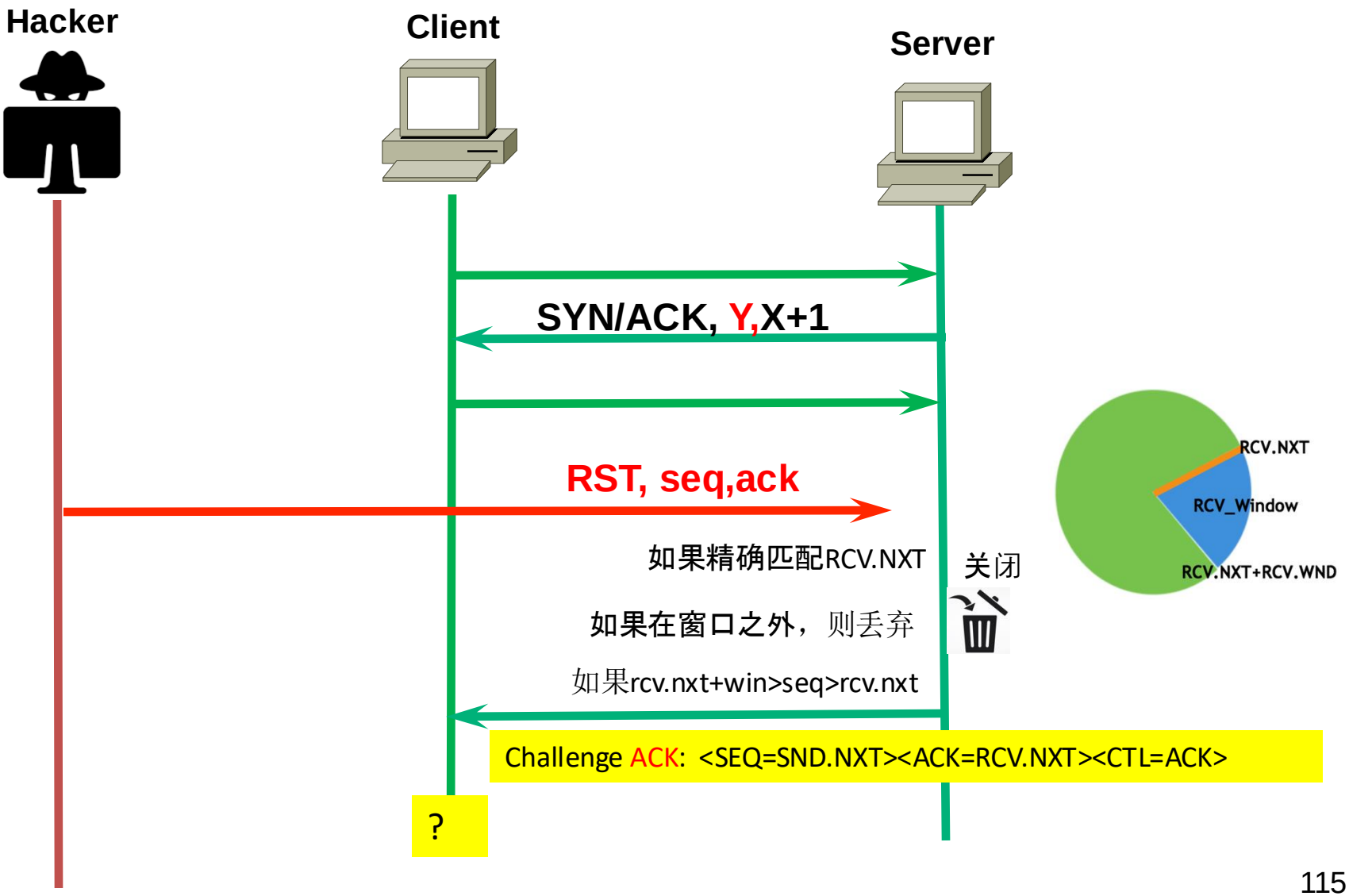


SEQ #:	Before RFC 5961	After RFC 5961
Out-of-Window	Drop the Packet	Drop the Packet
In-Window	Reset Connection	Challenge ACK
Exactly match		Reset Connection

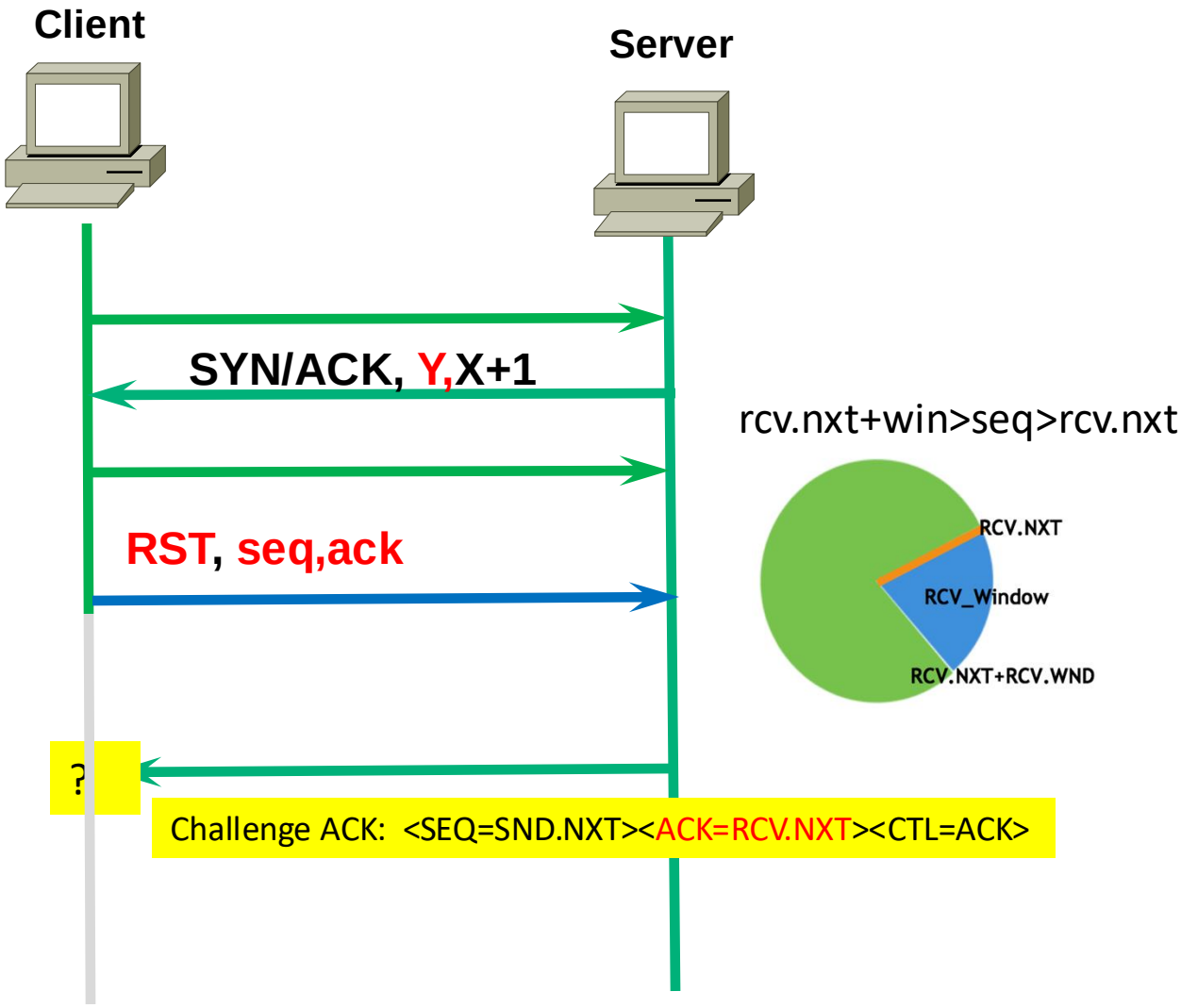
Challenge ACK: tell sender to confirm if it indeed terminated the connection

Slide from Prof. Zhiyun Qian

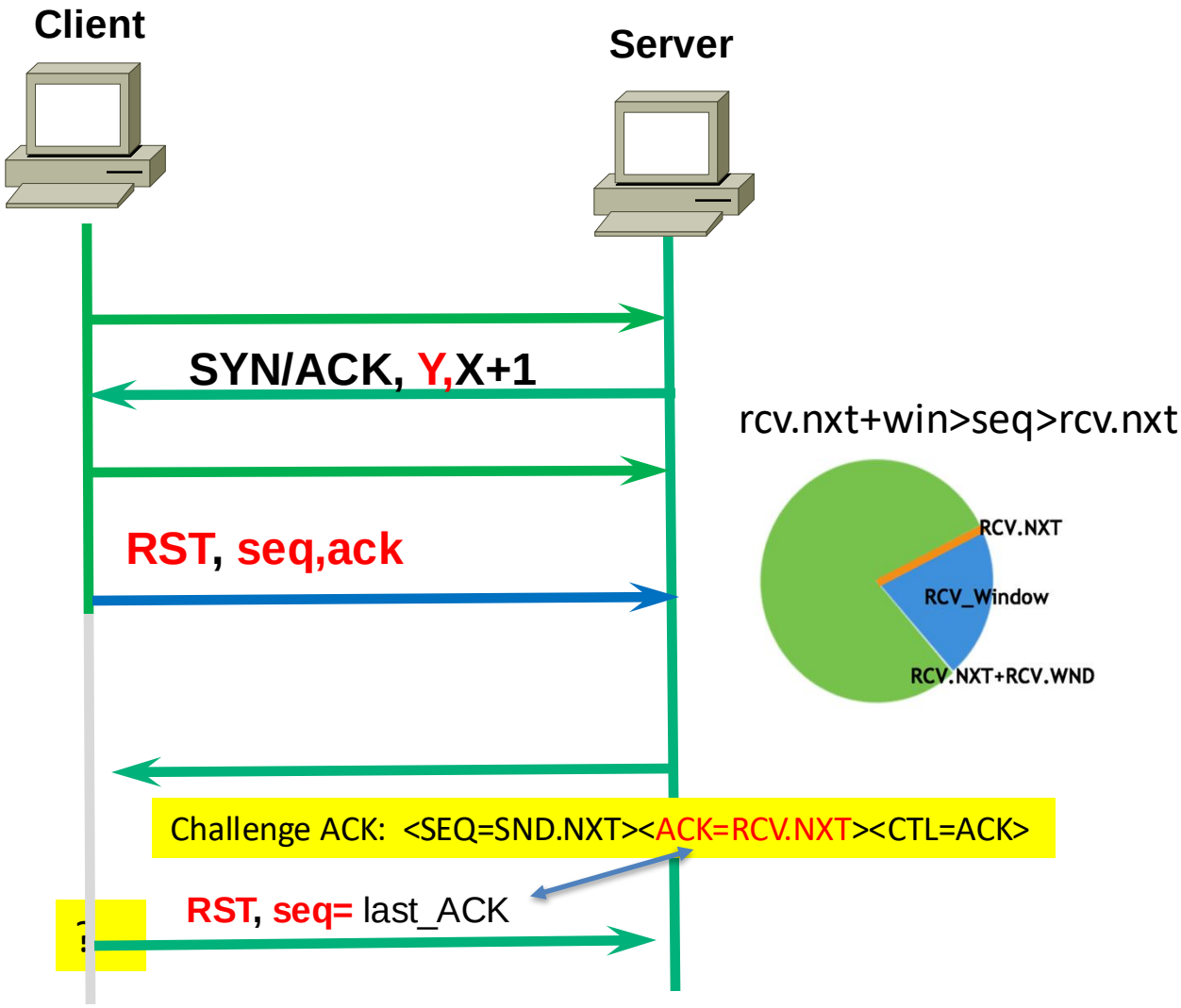
In-Window RST sent from Hacker



In-Window RST sent from Client



In-Window RST sent from Client



10. Security Considerations

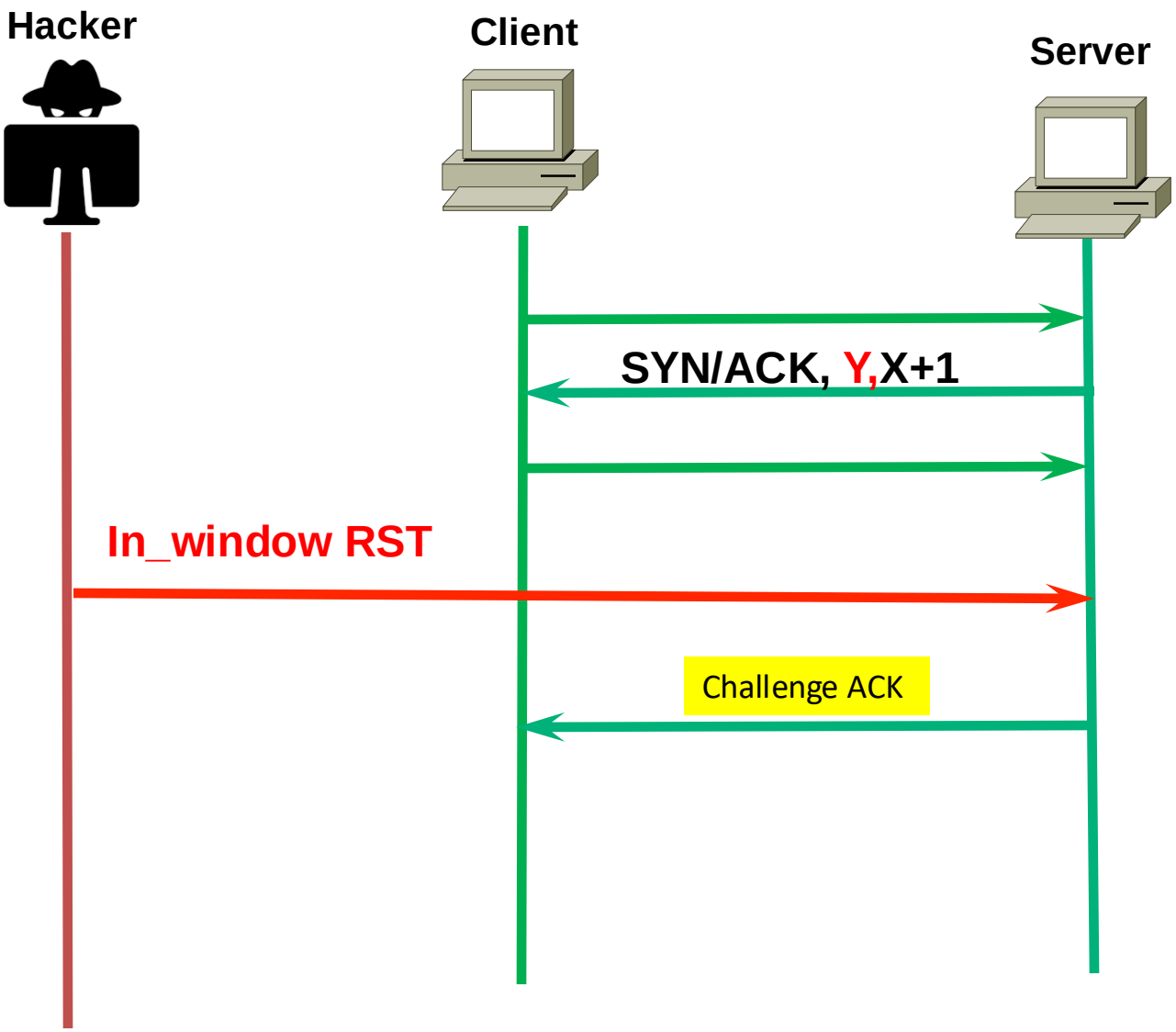
These changes to the TCP state machine do NOT protect an implementation from on-path attacks. It also needs to be emphasized that while mitigations within this document make it harder for off-path attackers to inject segments, it does NOT make it impossible. The only way to fully protect a TCP connection from both on- and off-path attacks is by using either IPsec Authentication Header (AH) [RFC4302] or IPsec Encapsulating Security Payload (ESP) [RFC4303].

Implementers also should be aware that the attacks detailed in this specification are not the only attacks available to an off-path attacker and that the counter measures described herein are not a comprehensive defense against such attacks.

反射攻击

Another notable consideration is that a reflector attack is possible with the required RST/SYN mitigation techniques. In this attack, an off-path attacker can cause a victim to send an ACK segment for each spoofed RST/SYN segment that lies within the current receive window of the victim. It should be noted, however, that this does not cause any amplification since the attacker must generate a segment for each one that the victim will generate.

Reflector attack



Off-Path TCP Exploits: Global Rate Limit Considered Dangerous (CVE-2016-5696)

Yue Cao, Zhiyun Qian, Zhongjie Wang, Tuan Dao, Srikanth V. Krishnamurthy, Lisa M. Marvel, *USENIX SECURITY 2016*

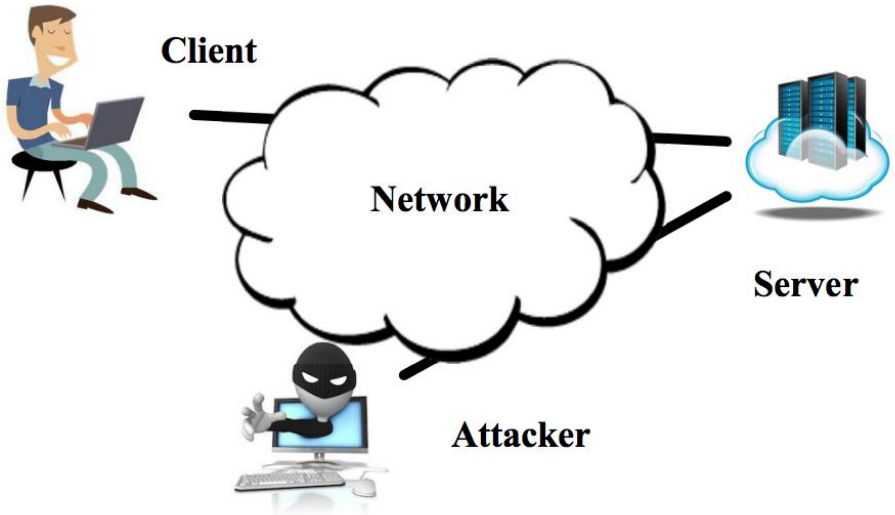


Figure 1: Threat model 1

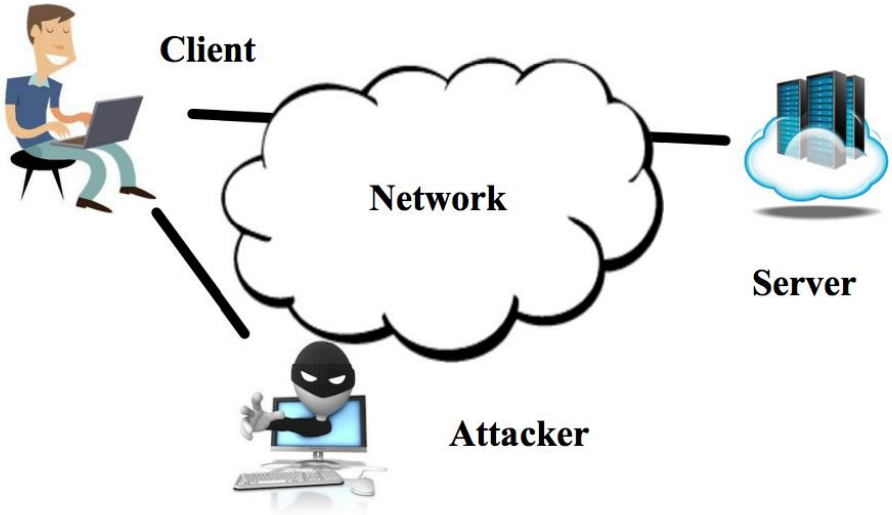


Figure 2: Alternative threat model

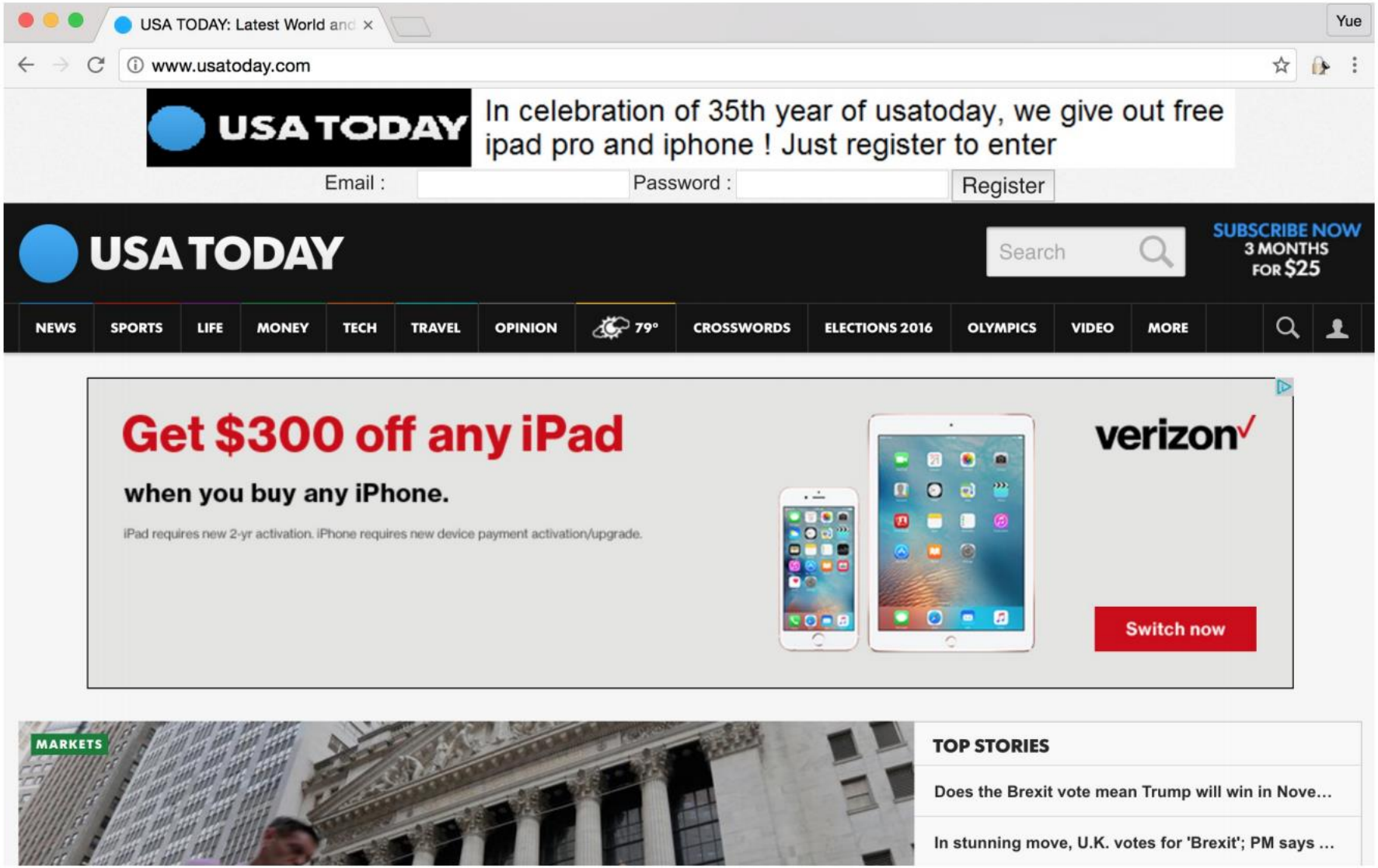


Figure 13: USAToday screenshot with phishing registration window

Reference

1. Security Assessment of the Internet Protocol Version 4, RFC 6274, 2011, https://tools.ietf.org/html/rfc6274
2. TCP/IP Guide, TCP/IP Transmission Control Protocol (TCP) , http://www.tcpipguide.com/free/t_TCPIPTransmissionControlProtocolTCP.htm
3. http://cr.yp.to/syncookies.html
4. Z. Durumeric, E. Wustrow, and J. A. Halderman, “ZMap: Fast Internet-wide Scanning and Its Security Applications.,” USENIX Security …, pp. 605–620, 2013.
5. Xu Zhang, Jefferey Knockel, Jed. Crandall. “Original SYN: Finding Machines Hidden Behind Firewalls,” INFOCOM, 2015
6. M. Vallentin, R. Sommer, J. Lee, C. Leres, V. Paxson, B. Tierney, The NIDS Cluster: Scalable, Stateful Network Intrusion Detection on Commodity Hardware, Proc. Symposium on Recent Advances in Intrusion Detection, 2007
7. W. YANG, B. Fang, X. C. YUN, and H. ZHANG, “A parallel cluster intrusion detection system for backbone network [J],” Journal of Harbin Institute of Technology, vol. 3, 2004.
8. M. Handley, V. Paxson, and C. Kreibich, “Network intrusion detection: Evasion, traffic normalization, and end-to-end protocol semantics,” Proceedings of the 10th conference on USENIX Security Symposium-Volume 10, pp. 9–9, 2001.
9. U Shankar, V Paxson. Active mapping: Resisting NIDS evasion without altering traffic, - Security and Privacy, 2003. Proceedings …, 2003 - ieeexplore.ieee.org
10. J. R. Crandall, D. Zinn, M. Byrd, E. Barr, and R. East, “Conceptdoppler: A weather tracker for internet censorship,” 14th ACM Conference on Computer and Communications Security, pp. 1–4, 2007.
11. X. Xu, Z. Mao, and J.Alex Halderman, “Internet censorship in china: where does the filtering occur?,” Passive and Active Measurement, pp. 133–142, 2011.
12. J. Park, “Empirical study of a national-scale distributed intrusion detection system: Backbone-level filtering of html responses in china,” Distributed Computing Systems 2010.
13. S. Khattak, M. Javed, P. D. Anderson, and V. Paxson, Towards Illuminating a Censorship Monitor's Model to Facilitate Evasion, Proc. USENIX Workshop on Free and Open Communications on the Internet (FOCI), August 2013
14. Anonymous, Towards a Comprehensive Picture of the Great Firewall’s DNS Censorship, FOCI 2014