SymTCP: Eluding Stateful Deep Packet Inspection with Automated Discrepancy Discovery

Zhongjie Wang, Shitong Zhu, Yue Cao, Zhiyun Qian, Chengyu Song, Srikanth Krishnamurthy, Kevin Chan, and Tracy Braun
What is DPI (Deep Packet Inspection)?

Censorship and Surveillance  
ISP Traffic Differentiation  
Modeling Users for Online Ads
How does DPI work?
Implementation-level discrepancy

// Linux TCP timestamp validation
if (((signed int)(last_tsval - current_tsval) <= 1) {
    // succeed
} else {
    // fail
}
lst_tsval - 1 <= current_tsval <= last_tsval + 2^{31}

// Snort TCP timestamp validation
if (((signed int)((current_tsval - last_tsval) + 1) < 0) {
    // fail
} else {
    // succeed
}
lst_tsval - 1 <= current_tsval <= last_tsval + 2^{31} - 2
Workflow of SymTCP

1. Symbolic Execution

2. Highly effective test cases

Huge search space!!!

Successful test cases
Problem with symbolic execution

All possible packets → All possible execution paths

Path explosion!!!
Pruning decisions

Labeling “drop” / “accept” points
In the program, we label where a packet gets dropped or accepted (i.e. TCP state changed). We try to cover these accept/drop points.

Bounding TCP options
We allow each TCP option to occur only once, and at most 5 different TCP options in a packet.

Pruning uninteresting TCP states
We terminate an execution path once it reaches any uninteresting TCP state (e.g., TIME_WAIT, CLOSED)
Differential testing DPI
Complete packet sequence

Packet triggering discrepancies

Test case

Follow-up packets

Packet triggering feedback

LISTEN state

ESTABLISHED state
Symbolic execution performance

- Linux kernel v4.9.3
- 72 core Intel Xeon CPU and 256GB memory
- 1/2/3 symbolic packets
- 20/40/60 byte length packet

<table>
<thead>
<tr>
<th># of pkts</th>
<th>20-byte TCP pkts</th>
<th>40-byte TCP pkts</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Time to cover</td>
<td>Covered drop points</td>
</tr>
<tr>
<td>1</td>
<td>5s</td>
<td>8</td>
</tr>
<tr>
<td>2</td>
<td>20s</td>
<td>16</td>
</tr>
<tr>
<td>3</td>
<td>50s</td>
<td>31</td>
</tr>
</tbody>
</table>

| 20m | 18m | 18 |
|     |     |    |
| 40m | 38  |    |

56,787 test cases
Sampled 10,000 test cases

Time cost could vary due to randomness in path selection of symbolic execution.
Zeek (formerly Bro)

- 6082 successful test cases, 9 strategies, 2 novel strategies

**TABLE IV. SUCCESSFUL STRATEGIES ON ZEEK V2.6**

<table>
<thead>
<tr>
<th>Strategy</th>
<th>TCP state</th>
<th>Insertion/Evasion packet</th>
<th>Linux</th>
<th>Zeek</th>
</tr>
</thead>
<tbody>
<tr>
<td>† SYN with data</td>
<td>L/SR/E</td>
<td>(I) SYN packet with data</td>
<td>Ignore data</td>
<td>Accept data</td>
</tr>
<tr>
<td>† Multiple SYN</td>
<td>SR/E</td>
<td>(I) SYN packet with out-of-window SEQ num</td>
<td>Discard and send ACK</td>
<td>Reset TCB</td>
</tr>
<tr>
<td>† Pure FIN</td>
<td>E</td>
<td>(I) Pure FIN packet without ACK flag</td>
<td>Discard (may send ACK)</td>
<td>Flush and reset receive buffer</td>
</tr>
<tr>
<td>† Bad RST/FIN</td>
<td>SR/E</td>
<td>(I) RST or FIN packet with out-of-window SEQ num</td>
<td>Discard (may send ACK)</td>
<td>Flush and reset receive buffer</td>
</tr>
<tr>
<td>† Data overlapping</td>
<td>SR/E</td>
<td>(I) Out-of-order data packet, then overlapping in-order data packet</td>
<td>Accept in-order data</td>
<td>Accept first data</td>
</tr>
<tr>
<td>† Data without ACK</td>
<td>SR/E</td>
<td>(I) Data packet without ACK flag</td>
<td>Discard</td>
<td>Accept</td>
</tr>
<tr>
<td>† Data bad ACK</td>
<td>E</td>
<td>(I) Data packet with ACK &gt; snd_nxt or &lt; snd_una - window_size</td>
<td>Discard</td>
<td>Accept</td>
</tr>
</tbody>
</table>

* Big gap         | SR/E      | (I) Data packet with SEQ > rcv_nxt + max_gap_size (16384)                               | Accept                       | Ignore later data             |
* SEQ < ISN       | SR/E      | (E) Data packet with SEQ num < client ISN and in-window data                            | Accept in-window data        | Ignore                         |

652 successful test cases, 11 strategies, 3 novel

<table>
<thead>
<tr>
<th>Strategy</th>
<th>TCP state</th>
<th>Insertion/Evasion packet</th>
<th>Linux</th>
<th>Snort</th>
</tr>
</thead>
<tbody>
<tr>
<td>Multiple SYN</td>
<td>E</td>
<td>(I) SYN packet with in-window SEQ num</td>
<td>Discard and send ACK</td>
<td>Teardown TCB</td>
</tr>
<tr>
<td>In-window FIN</td>
<td>E</td>
<td>(I) FIN packet with SEQ num in window but ≠ rcv_nxt</td>
<td>Ignore FIN (may accept data)</td>
<td>Cut off later data</td>
</tr>
<tr>
<td>FIN/ACK bad ACK</td>
<td>E</td>
<td>(I) FIN/ACK packet with ACK num &gt; snd_nxt or &lt; snd_una - window_size</td>
<td>Discard (may send ACK)</td>
<td>Cut off later data</td>
</tr>
<tr>
<td>FIN/ACK MD5</td>
<td>SR/E</td>
<td>(I) FIN/ACK packet with TCP MD5 option</td>
<td>Discard</td>
<td>Cut off later data</td>
</tr>
<tr>
<td>In-window RST</td>
<td>E</td>
<td>(I) RST packet with SEQ num ≠ rcv_nxt but still in window</td>
<td>Discard and send ACK</td>
<td>Teardown TCB</td>
</tr>
<tr>
<td>RST bad timestamp</td>
<td>SR/E</td>
<td>(I) RST packet with bad timestamp</td>
<td>Discard</td>
<td>Teardown TCB</td>
</tr>
<tr>
<td>RST MD5</td>
<td>SR/E</td>
<td>(I) RST packet with TCP MD5 option</td>
<td>Discard</td>
<td>Teardown TCB</td>
</tr>
<tr>
<td>RST/ACK bad ACK num</td>
<td>SR/E</td>
<td>(I) RST/ACK packet with ACK num ≠ server ISN + 1</td>
<td>Discard</td>
<td>Teardown TCB</td>
</tr>
<tr>
<td>Partial in-window RST</td>
<td>E</td>
<td>(I) RST packet with SEQ num &lt; rcv_nxt but partial data in window</td>
<td>Discard</td>
<td>Teardown TCB</td>
</tr>
<tr>
<td>Urgent data</td>
<td>SR/E</td>
<td>(E) Data packet with URG flag and urgent pointer set</td>
<td>Consume 1 byte urgent data</td>
<td>Ignore all data before urgent pointer</td>
</tr>
<tr>
<td>Time gap</td>
<td>SR/E</td>
<td>(E) Data packet timestamp = last timestamp + 0x7fzffff/0x80000000</td>
<td>Accept</td>
<td>Ignore</td>
</tr>
</tbody>
</table>

Great Firewall of China (GFW)

- 4587 successful test cases, 12 strategies, 9 novel

<table>
<thead>
<tr>
<th>Strategy</th>
<th>TCP state</th>
<th>Insertion/Evasion packet</th>
<th>Linux</th>
<th>GFW</th>
</tr>
</thead>
<tbody>
<tr>
<td>† Bad RST</td>
<td>SR/E</td>
<td>(I) RST packet with bad checksum or TCP MD5 option</td>
<td>Discard</td>
<td>Teardown TCB</td>
</tr>
<tr>
<td>† Bad data</td>
<td>SR/E</td>
<td>(I) Data packet with bad checksum or TCP MD5 option or bad timestamp</td>
<td>Discard</td>
<td>Accept</td>
</tr>
<tr>
<td>† Data without ACK</td>
<td>SR/E</td>
<td>(I) Data packet without ACK flag</td>
<td>Discard</td>
<td>Accept</td>
</tr>
<tr>
<td>* SEQ ≤ ISN</td>
<td>SR/E</td>
<td>(E) Data packet with SEQ num ≤ client ISN and in-window data</td>
<td>Accept in-window data</td>
<td>Ignore</td>
</tr>
<tr>
<td>* Small segments</td>
<td>SR</td>
<td>(E) Data packet with payload size ≤ 8 bytes</td>
<td>Ignore</td>
<td>Ignore</td>
</tr>
<tr>
<td>* FIN with data</td>
<td>SR/E</td>
<td>(I) FIN packet with data and without ACK flag</td>
<td>Discard</td>
<td>Teardown TCB</td>
</tr>
<tr>
<td>* Bad FIN/ACK data</td>
<td>E</td>
<td>(I) FIN/ACK packet with data and bad checksum or TCP MD5 option or bad timestamp</td>
<td>Discard</td>
<td>Teardown TCB</td>
</tr>
<tr>
<td>* FIN/ACK data bad ACK</td>
<td>E</td>
<td>(I) FIN/ACK packet with data and ACK num &gt; snd_nxt or &lt; snd_una - window_size</td>
<td>Discard</td>
<td>Teardown TCB</td>
</tr>
<tr>
<td>* Out-of-window SYN data</td>
<td>SR</td>
<td>(I) SYN packet with SEQ num out of window and data</td>
<td>Discard and send ACK</td>
<td>Desynchronized</td>
</tr>
<tr>
<td>* Retransmitted SYN data</td>
<td>SR</td>
<td>(I) SYN packet with SEQ num = client ISN and data</td>
<td>Discard</td>
<td>Desynchronized</td>
</tr>
<tr>
<td>* RST bad timestamp</td>
<td>SR</td>
<td>(I) RST packet with bad timestamp</td>
<td>Discard</td>
<td>Teardown TCB</td>
</tr>
<tr>
<td>* RST/ACK bad ACK num</td>
<td>SR</td>
<td>(I) RST/ACK packet with SEQ num ≠ server ISN + 1</td>
<td>Discard</td>
<td>Teardown TCB</td>
</tr>
</tbody>
</table>

Case study

1. Urgent Pointer (Snort)

Client → Snort → Server
SYN/ACK
ACK
Got junk
Data: "sensitive data & junk"
Urgent pointer: point to junk

2. Underflow SEQ (Zeek & GFW)

Client → Zeek/GFW → Server
SYN/ACK
SYN
SEQ=X
ACK
SEQ=X+1
Data: "AAsensitive data"
SEQ=X-1 (expected to be X+1)
Ignored
Key contributions

- A novel approach that combines whitebox and blackbox testing
  - Whitebox: Extract a reference model from server with symbolic execution
  - Blackbox: Infer internal states of DPI with follow-up packets
- First to run symbolic execution on full-fledged TCP implementation and send multiple symbolic packets
- Highly efficient and effective automated tool to unearth discrepancies between different TCP implementations
  - Facilitate DPI elusion
  - Help developers fix implementation bugs
Conclusion

- A novel approach combines whitebox and blackbox testing to automatically discover TCP implementation-level discrepancies
- Evaluated against 3 well-known DPI systems, Zeek (Bro), Snort, and the GFW, and found 14 novel strategies
- A significant step in testing and eluding DPI systems

I’m on the job market!
Homepage: https://zhongjie.me
Overview

- Symbolic Execution (whitebox) + Differential Testing (blackbox)
- Highly **efficient** and **effective** test case generation
  - >5k test cases in one hour
  - Cover **all** accept/drop points in Linux TCP implementation
- Found **14 novel** evasion strategies for Zeek (Bro), Snort, GFW
- An important step towards **automating** DPI evasion strategy discovery
Case study

Time Gap (Snort)

Client -> Snort
SYN/ACK

Snort -> Server
SYN

Server -> Client
ACK
TSVAL=T
Data: "sensitive data"
TSVAL=T+0x80000000

Ignored
Discrepancies between DPI and endhost

- **Gap between accept ranges**
  - Insertion packet: DPI accept, but endhost reject
  - Evasion packet: DPI reject, but endhost accept

- **Reasons for discrepancies**
  - Ambiguities in TCP specification
  - Evolution of specification
  - New features in TCP, e.g., MD5
  - Faulty implementation

- **Various TCP implementations, different OS, different version**
- **Almost impossible for DPI to behave exactly the same as endhosts**
Previous Work

• DPI evasion
  • Ptacek et al, 1998
  • S. Khattak et al, [FOCI’13]
  • Z. Wang et al, [IMC’17]
  • F. Li et al, [IMC’17]
• Automated evasion strategy discovery
  • K. Bock et al, [CCS’19]
• Symbolic execution in finding semantic bugs
  • S. Y. Chau et al, [S&P’17][NDSS’19]
System design
Symbolizing TCP header (including TCP options)
Online Probing

Test cases:
(symbolic execution result)

Example:

Pkt #1 is an evasion packet!
Early termination!
Online Probing

Test cases:

Example:

Pkt #2 is an evasion packet! Early termination!
Online Probing

Test cases:

Example:

Pkt #3 is an insertion packet!
Test case done!
Implementation

- Built upon S2E 2.0
- ~2.5KLOC C++ S2E plugins
- ~6.5KLOC Python analysis scripts
- Z3 constraint solver
Zeek (Bro) results

- 6082/10000 succeeded
- 5771 insertion packets
- 311 evasion packets
- Liberal in accepting incoming packets
  - Doesn't check SEQ and ACK number for control packets
  - Doesn't check ACK number for data packets
  - Reset connection when receiving multiple SYN packets
Snort results

- 652/10000 succeeded
- 432 insertion packets
- 220 evasion packets
- OS-specific policies, rigorous implementation
- Still more liberal than Linux
  - Accepts in-window SYN/RST/FIN
  - Doesn’t check ACK number for control packets
- Incomplete implementation
  - Doesn’t support TCP MD5 option
GFW results

- 4587/10000 succeeded
- 1435 insertion packets
- 3152 evasion packets
- Sophisticated state machine
- Re-synchronization: robust against de-synchronization attacks
- Strict in accepting packets
Symbolic Execution Tree

```c
int a = α, b = β, c = γ;
    // symbolic
int x = 0, y = 0, z = 0;
if (a) {
    x = -2;
}
if (b < 5) {
    if (!a && c) { y = 1; }
    z = 2;
}
assert(x+y+z!=3)
```
<table>
<thead>
<tr>
<th>Reason</th>
<th>Count</th>
</tr>
</thead>
<tbody>
<tr>
<td>TCP checksum error</td>
<td>5</td>
</tr>
<tr>
<td>TCP header length too small</td>
<td>1</td>
</tr>
<tr>
<td>TCP header length too large</td>
<td>4</td>
</tr>
<tr>
<td>MD5 option error</td>
<td>2</td>
</tr>
<tr>
<td>TCP flags invalid</td>
<td>7</td>
</tr>
<tr>
<td>SEQ number invalid</td>
<td>10</td>
</tr>
<tr>
<td>ACK number invalid</td>
<td>3</td>
</tr>
<tr>
<td>Challenge ACK</td>
<td>6</td>
</tr>
<tr>
<td>Receive window closed</td>
<td>2</td>
</tr>
<tr>
<td>Empty data packet</td>
<td>1</td>
</tr>
<tr>
<td>Data overlap in OFO queue</td>
<td>1</td>
</tr>
<tr>
<td>PAWS check failed</td>
<td>2</td>
</tr>
<tr>
<td>Embryonic reset</td>
<td>1</td>
</tr>
<tr>
<td>TCP_DEFER_ACCEPT drop bare ACK</td>
<td>1</td>
</tr>
<tr>
<td>TCP Fastopen check request failed</td>
<td>1</td>
</tr>
<tr>
<td>Total number</td>
<td>47</td>
</tr>
</tbody>
</table>
Extending to other DPIs, Linux kernel versions and OSes

- DPIs: re-run online probing
- Linux kernel versions: re-label accept/drop points
- OSes: re-visit pruning decisions, re-label accept/drop points