Finding Protocol Manipulation Attacks

Nupur Kothari

Ratul Mahajan, Todd Millstein,
Ramesh Govindan, Madanlal Musuvathii
Manipulation Attacks

Adversaries induce victim into undesirable behavior by lying in their messages

• Exploit partial information among participants
• Hard to detect because messages are protocol-compliant
Example: Optimistic ACKs in TCP [Savage ‘01]

**Regular TCP**
- TCP Sender
- TCP Receiver

**Optimistic ACKs**
- TCP Sender
- TCP Receiver

**Data Flow**
- Regular TCP
  - Data: 1:1461
  - ACK: 1461
  - Data: 1461:2921
  - ACK: 2921
  - Data: 2921:4381
  - ACK: 4381

**Optimistic ACKs**
- Data: 1:1461
- ACK: 4381
- ACK: 5841

**Receiver pretends packets are received earlier than they are**

**Sender fooled into sending data faster**
Other Examples

- **ECN (Explicit Congestion Notification)** [Ely ’01]
  - Adversarial receiver hides congestion by resetting the congestion bit in ACK
    ➔ Sender is fooled into ignoring congestion

- **802.11 MAC (WiFi)** [Bellardo ’03]
  - Adversary sets high duration value in data frames
    ➔ Neighboring nodes are starved
Goal and Contributions

Goal

- Automate finding manipulation attacks in protocol implementations

Contributions

- Novel use of program analysis techniques to find manipulation attacks
  - Guided by developer inputs

- MAX: A tool to analyze C protocol implementations
  - Finds all known attacks in TCP, 802.11 and ECN, and a new variant in SCTP
Manipulation Attack Characteristics

- **Lying**: Adversaries induce victim into exhibiting undesirable behaviors by modifying messages.

- **Protocol-compliance**: Messages are valid under other conditions.

- **Repetition**: Manipulations may need to be repeated to have significant impact.
Finding Manipulation Attacks: Challenge

Manipulations triggering undesirable behavior

All possible packet manipulations

Extremely Large!

Manipulation Attacks

Can be repeated

All possible message headers

Different protocol states

Different network conditions
Our Approach: Static + Dynamic Analysis

Symbolic execution
Statically computes conditions under which a code path is taken

Adversarial concrete execution
Emulate adversarial behavior during protocol execution

User Inputs
- Vulnerable statement(s)
- Network setup, Impact Metric(s)

Path constraints
- Messages that lead to vulnerable statement(s) given current victim state

Manipulation attack
- Sequence of manipulations to repeatedly trigger vulnerable statement(s)
**MAX: Symbolic Execution**

### Challenges

- **Scalability:** Prune uninteresting parts, work on partial programs
- **Accuracy:** Use domain-specific knowledge to improve approximations for pointers, type-casts
MAX: Adversarial Concrete Execution

Feasible Code Paths
Path Constraints

snd_nxt = 4323

seq == rcv_next
ack_seq <= snd_nxt
ack_seq >= snd_una1

Adversarial Module

Interceptor
Messages

Victim State

Adversary

Network

Victim
MAX: Outputs

Vulnerable Statement(s)
Network Setup
Impact Metric(s)

Set ack_seq to snd_nxt for all incoming ACKs

Sequence of manipulations that work

Comparison between adversarial and honest execution

Set ack_seq to snd_nxt for all incoming ACKs
MAX Implementation

- Uses CIL framework for C
- Uses Z3 constraint solver
- Can analyze real protocol implementations (user-level) or simulator code
## MAX Evaluation

<table>
<thead>
<tr>
<th>Protocol</th>
<th>LoC (K)</th>
<th>Attacks Found</th>
</tr>
</thead>
<tbody>
<tr>
<td>TCP (Daytona)</td>
<td>14.2</td>
<td>Optimistic ACKing</td>
</tr>
<tr>
<td>SCTP</td>
<td>12.5</td>
<td>Optimistic ACKing variant</td>
</tr>
<tr>
<td>802.11 MAC (Qualnet)</td>
<td>11.0</td>
<td>NAV, RTS</td>
</tr>
<tr>
<td>ECN (Qualnet)</td>
<td>7.6</td>
<td>ECN Echo</td>
</tr>
</tbody>
</table>
Exploring TCP with MAX

- Daytona
  - User-level port of Linux 2.2 kernel TCP
- Vulnerable Statement: Decrease the number of outstanding packets at the TCP Sender
- Network Setup: same LAN, background TCP traffic
Exploring TCP: Optimistic ACK Attack

- MAX manipulates the sequence number ACKed to last sequence number sent out by receiver
  - Successfully causes vulnerable statement to be executed

**ACKed Sequence Number**
- Increased Sequence Number
- No satisfying paths found

**TCP Throughput**
- Successful manipulation attack!
Exploring SCTP with MAX: Take 1

- Experimental setup identical to TCP
- Vulnerable Statement: Decrease the number of outstanding packets at the SCTP Sender
- MAX increases cumulative TSN ACKed

![Diagram showing TSN (Seq. No.) ACKed and SCTP Throughput with failed manipulation attack and sudden drop in throughput.]

- Rate of change of TSN ACK decreased
- Failed manipulation attack!
- Sudden drop in throughput
Why the attack failed

- Sending rate depends on outstanding packets and receiver window
  - # of packets for which the receiver currently has space
- Subtle difference between TCP and SCTP receiver window semantics
  - Decreasing outstanding packets does not capture the difference in semantics
Exploring SCTP with MAX: Take 2

- New Vulnerable Statement: sending messages
- MAX manipulates TSN ACKed and receiver window size

![Graphs showing TSN (Seq. No.) ACKed and SCTP Throughput]

**TSN (Seq. No.) ACKed**
- Honest Run
- Adversarial Run

**SCTP Throughput**
- Honest Run
- Adversarial Run

*Successful manipulation attack!*
Summary and future work

- Novel approach to find manipulation attacks in network protocol implementations
  - Found known attacks in TCP, 802.11, and ECN, and a new variant in SCTP
- Future work
  - Study other protocols and attacks
  - Mitigate manipulation attacks
  - Other forms of semantic analysis of protocols

enl.usc.edu/projects/max