Automated Attack Discovery in Data Plane Systems

Qiao Kang, Jiarong Xing, Ang Chen
Rice University
Motivation: A new class of attacks

- Attacks to emerging “data plane systems”
  - Network data planes are performing more functions today
  - Data plane systems: Enabled by “programmable data planes”
  - A general class of attacks to many of them
New trend: Programmable data planes

• Traditional data planes: **Fixed for routing**
• Programmable data planes: **Reconfigurable pipelines**
  • Using high-level languages like **P4**
  • Support sophisticated operations like arithmetic

Ingress {
  // ACL
  if (ACL[pkt] != Allow)
    drop();

  // Routing
  forward_to_port();

  // Traffic Engineering
  dst = least_util_link()
}
Data plane systems: High performance

- Data plane systems have **high performance**.
- Example: Link failure detection
  - Border Gateway Protocol (BGP): Periodic probing messages  \(\rightarrow O(\text{minutes})\)
  - Blink [NSDI’19]: Monitors data traffic  \(\rightarrow O(\text{seconds})\)
Open direction: Security risks

- Data plane systems react to **network packets**
- Anyone can inject **malicious packets** to cause **problematic outputs**
Example #1: Attacking a load balancer

If (TCP.sport % 2)
  Forward (0)
Else
  Forward (1)

- **Expected behavior**: Evenly splitting traffic
- **Malicious traffic**: TCP source port numbers = 1, 3, 5, 7...
- **Flipped behavior**: Load imbalance

- **Expected behavior**:
  - Path 0: 50%
  - Path 1: 50%

- **Flipped behavior**:
  - Path 0: 100%
  - Path 1: 0%
Example #2: Attacking Blink

// monitor TCP retrans
If (retrans > N)  
    Reroute()  
Else  
    NormalForward()

• Expected behavior: Only rerouting when link fails (very rare)
• Malicious traffic: Persistent TCP retransmissions
• Flipped behavior: Persistent re-routing and routing chaos
A general class of attacks

- Applies to many data plane systems!
- Different systems are vulnerable to different malicious patterns
Research question

Given a data plane system, can we discover all malicious traffic patterns and synthesize defenses in an automated manner?
A 3-step approach:

① Establish expected behaviors
② Generate attacks to flip the expected behaviors
③ Synthesize runtime monitors

Automated
Outline

• Motivation:
  • A new class of attacks to data plane systems
• Our system: Automated attack discovery and defense synthesis
  • System overview
    • Challenge #1: Establish expected behaviors
    • Challenge #2: Identifying equivalent classes
    • Challenge #3: Handling stateful programs
• Preliminary results
• Ongoing work and Conclusion
Outline

• Motivation:
  • A new class of attacks to data plane systems

• Our system: Automated attack discovery and defense synthesis
  • System overview
  • Challenge #1: Establish expected behaviors
  • Challenge #2: Identifying equivalent classes
  • Challenge #3: Handling stateful programs

• Preliminary results

• Ongoing work and Conclusion
Challenge #1: Establishing expected behaviors

• Problem: How to quantify the expected behaviors?
• Naïve solution: Feed random traffic traces and observe its outputs
  • Might not be comprehensive
• Proposed solution: Probabilistic Symbolic Execution (PSE)
  • An advanced version of Symbolic Execution
Probabilistic Symbolic Execution (PSE)

- Explore execution paths with per-path probabilities
- Model Counting: “number of solutions”
- Packet headers: Uniform distribution
Challenge #2: Identifying Equivalence Classes

- Problem: Number of paths might be very large
  - Hard to understand the expected behaviors.

- Proposed Solution: Equivalence Classes (ECs)
  - EC = a group of “equivalent” paths

Per-path probabilities

\[
\begin{align*}
P_1 & = 0.1 \\
P_2 & = 0.02 \\
P_3 & = 0.15 \\
P_4 & = 0.3
\end{align*}
\]

Per-EC probabilities

\[
\begin{align*}
\text{Pr}(\text{NormalFowarding}) & = 0.9 \\
\text{Pr}(\text{ReRouting}) & = 0.1
\end{align*}
\]
Challenge #3: Handling stateful programs

- Problem: Data plane systems can be stateful
  - Need a sequence of N packets to trigger a certain EC (e.g., Blink)

- Naïve solution: Explore all possible paths for N packets
  - Poor scalability

- Proposed solution: Directed Symbolic Execution (DSE)
  - Heuristic search: Prioritize the “closest” path

![Diagram showing packet flow and EC transitions](image-url)
Outline

• Motivation:
  • A new class of attacks to data plane systems

• Our system: Automated attack discovery and defense synthesis
  • System overview
  • Challenge #1: Establish expected behaviors
  • Challenge #2: Identifying equivalent classes
  • Challenge #3: Handling stateful programs

• Preliminary results

• Ongoing work and Conclusion
Setup

- Prototype implementation
  - Symbolic execution engine: P4pktgen [SOSR’18]
  - Model Counter: Python constraint library

- Experimental setup
  - P4 load balancer
  - Mininet simulator: 1 Bmv2 P4 switch + 3 hosts

```
If (TCP.sport % 2)
   Forward (0)
Else
   Forward (1)
```
Generated attack and defense

- Generated attack: Odd TCP source port numbers
- Generated defense: Per-EC packet counters + periodic tests
Link load results

- Normal traffic: 0~15s
- Attack starts at 15s
- Attack detected by the “patched” program

Alarm raised!
Ongoing work

• How to handle input packets that follow non-uniform distributions?
  • ”Distribution-aware” model counting

• How to group execution paths to ECs?
  • Too fined-grained: too many ECs
  • Too coarse-grained: lose useful information

• How to deal with switch resource constraints?
  • Adding monitors consumes switch resources
  • Compress monitors using sketches
Conclusion

• Motivation:
  • Data plane systems are emerging
  • Vulnerable to a new class of attacks

• Our system: Automated attack discovery
  • 1) Obtain expected behaviors
  • 2) Negate expected behaviors
  • 3) Synthesis runtime monitors

• Initial results:
  • ✔ Attack a simple 2-way load balancer
  • ✔ Detected by runtime monitors

Thank you!