Programmable In-Network Security for Context-aware BYOD Policies

Qiao Kang, Lei Xue*, Adam Morrison, Yuxin Tang, Ang Chen, Xiapu Luo*

Rice University *The Hong Kong Polytechnic University
BYOD: Bring Your Own Device

- BYOD devices: Less well managed and easier to be compromised
- Need to access control for BYOD clients
“Context-aware” policies for BYOD

- Making **precise** security decisions by **dynamically** adapting to security contexts
- How to enforce these policies?
State of the art: SDN-based defense

- Client modules collect client-side information
- BYOD policies are managed and enforced in an SDN “app”
Limitations of the SDN-based defense

- Low defense agility: Context updates need to traverse the software controller
- Vulnerable to control plane DoS attacks [AvantGuard - CCS’13]
- Root cause: Lower processing speed of the SDN controller software
Research question

Can we address the limitations of SDN-based BYOD defense?
Opportunity: Programmable data planes

• Switch features:
  • Programmable parser: Customized protocols
  • Stateful processing
  • Arithmetic operations
  • General-purpose control plane

• High performance: <1us delay for Tbps traffic
• Can we transform these hardware features to security benefits for BYOD?
P4: Language for data plane programming

- Reconfigures switch pipeline for header manipulation
- Has the potential to enforce BYOD policies at linespeed
- Downside: P4 is low-level, non-trivial to develop and maintain

```c
header myTunnel_t {
    bit<16> proto_id;
    bit<16> dst_id;
}

struct headers {
    ethernet_t ethernet;
    myTunnel_t myTunnel;
    ipv4_t ipv4;
}

table ipv4_lpm {
    key = {
        hdr.ipv4.dstAddr: lpm;
    }
    actions = {
        ipv4_forward;
        drop;
        NoAction;
    }
    size = 1024;
    default_action = NoAction();
}
```

// count the number of bytes seen since the last probe
register<bit<32>>(MAX_PORTS) byte_cnt_reg;
// remember the time of the last probe
register<time_t>(MAX_PORTS) last_time_reg;
Poise: Programmable In-network Security

- **Language**: An expressive language for defining BYOD policies
- **Compiler**: Generates device configurations and switch programs
- **P4 data plane design**: A dynamic and efficient security primitive
• Motivation
• Poise Design
  • The Poise language
  • Compiling Poise policies
  • Data plane design
• Evaluation
• Conclusion
The Poise language

- An expressive language for writing context-aware policies
  - Predicates on customized client contexts
  - Support pre-defined primitive actions

**Examples of Poise Policies**

```
Block access if SSL version <= 6.5.2

if match (sslver <= 6.5) then drop
```
Compiling Poise policies

- Contexts (sslver) are compiled to customized header fields
- Security actions (if-else) are compiled to match/action table entries
- Advanced features: Policy composition, resource optimizations, etc

```plaintext
if match (sslver <= 6.5) then drop

header ctx_t {
  sslver: 16
}

table decision_tab {
  key = {ctx.sslver: exact}
  entries = {
    <= 6.5.0: dec = DROP
    >  6.5.0: dec = ALLOW
  }
}
```
Outline

• Motivation
• Poise Design
  • The Poise language
  • Compiling Poise policies
  • Data plane design
• Evaluation
• Conclusion
An efficient in-network primitive

- **Problem**: How to spread context information from client to switch?
  - Strawman solution: Tag every packet with context -- high overhead!
- **Idea**: Periodic context packets per flow: Headers + context, no data
  - **Dynamic**: Decisions are based on the latest context
  - **Efficient**: Data packets unmodified (no embedded contexts)
  - **Adjustable accuracy**: Tunable context packet period
Key data structure: A per-flow table

- A match/action table to maintain the latest per-flow decision
- Technical challenges:
  - New flow insertion delay (~1ms)
  - Controlling the size of the table
  - Handling DoS attacks (e.g., many new flows)

See more details in our paper!
Outline

- Motivation
- Poise Design
  - The Poise language
  - Compiling Poise policies
  - Data plane design
- Evaluation
- Conclusion
Evaluation setup

- Prototype implementation
  - Compiler: Bison + Flex
  - Android client module: a kernel module on Linux 3.18.31
  - ~6000 LoC

- Evaluation setup
  - Tofino Wedge 100BF switch 32 X 100 Gbps = 3.2 Tbps
What we have evaluated

- **Correctness**: Can Poise enforce BYOD policies correctly?
- **Overhead**: How much delay or throughput degradation can Poise incur?
- **Scalability**: How complex/large policies can Poise support?
- **Poise vs. SDN**: Is Poise resilient to control plane saturation attacks?

- SDN-based solution: PBS – NDSS’16
  - Floodlight v1.2 + Open vSwitch v2.9.2
- Methodology:
  - DoS attacker: Launch frequent context changes
  - Measure how normal user traffic are affected
Poise vs. SDN: First packet delay

- SDN: Takes \(~1\) second for the first packet to arrive under heavy attacks
- Poise: Remains at a constant level
Poise vs. SDN: New flow installation

- **SDN**: Fails to install new flows under heavy attacks
- **Poise**: Almost always installs 100% new flows
- **Poise** is highly resilient to DoS attacks from malicious clients
Conclusion

• Motivation
  • SDN-based BYOD defense has limitations

• Poise: Programmable In-Network Security
  • An expressive policy language
  • Compiler for generating P4 programs
  • An efficient in-network security primitive

• Poise transforms the hardware features to security benefits

Thank you for listening!

Contact: qiaokang@rice.edu – Looking for 2021 summer internship
https://github.com/qiaokang92/poise