Enforcing Context-Aware BYOD Policies with In-Network Security

ADAM MORRISON*, LEI XUE†, ANG CHEN*, XIAPU LUO†

*RICE UNIVERSITY, †THE HONG KONG POLYTECHNIC UNIVERSITY
Background: BYOD

• BYOD: Bring Your Own Device
  - Allow personal devices to access enterprise network
  - Reduces cost
  - Increases employee satisfaction

• BYOD is everywhere, and it’s only growing*
  As of 2016:
  - 59% of companies used BYOD
  - 13% more planned to within a year

*Tech Pro Research:
Motivation: BYOD has Security Issues

- Used to access both external and internal resources
- Not subject to the same robust security measures found on company-only devices
- Chinese-wall style perimeter policies are becoming outdated
We Need **Context-Aware Policies!**

- **Context-aware policy:** Looks at the environment, not just the individual request
- **Examples:**
  - “Only allow access if an administrator is online”
  - “Only allow access if the device is inside the office”
  - “Conduct deep packet inspection if the camera is on”
Properties of Context-Aware Policies

- Access to network-wide context
  - “Only allow access if an administrator is online”

- Access to device-specific context
  - “Only allow access if the device is inside the office”

- Dynamically changing context
  - “Conduct deep packet inspection if the camera is on”
Existing Solutions

Both have pros and cons!
Server Side Solutions

Example: BeyondCorp
- “No-Perimeter” network security model
- “Unprivileged” network

Pros:
- Easy to update and manage
- Minimal client trust

Cons:
- Missing device-specific context
- Don’t always control server
- Missing network-wide context

Unsupported policy:
Conduct deep packet inspection if camera is on
Client Side Solutions

Example: PBS (NDSS 2016)
- Client-side enforcement of policies defined by the network

Pros:
- Has easy access to device context
- Tighter control over device

Cons:
- Requires lots of trust in client
- High overhead on mobile device
- Missing network-wide context

• Unsupported policy: Allow access only if Admin is online
Best of Both Worlds?

• Common limitation: No access to network-wide context

• Solution: Enforce policies in the network!

• Solves many issues with client/server solutions
  – Full control over network
  – Minimal trust in clients
  – Easy access to network context
  – Can get access to device context with client-side module*
Challenge: Traditional Networks are not Programmable

Existing in-network solutions:

- Fixed-function switches can’t recognize application-level contexts
  - Access control lists, blacklisting, ..

- Hardware middleboxes aren’t customizable
  - Deep packet inspection, traffic scrubbing, ..

Neither is flexible enough to meet our needs!
Opportunity: The Network is Becoming Programmable!

• Programmable dataplanes:
  – Define your own headers
  – Perform richer header operations
  – Ex: Barefoot Tofino, Cavium Xpliant

• Domain-specific languages for programmable switches (e.g. P4):
  – Define custom headers
  – Create and apply custom match/action tables
  – Support for read/write registers
POISE: Programmable In-Network Security

• Policy enforcement moved inside the network
  - Leverages programmable dataplanes

• Client devices only responsible for collecting context information (using client-side module)

• Best of both worlds:
  - Access to fine-grained device-specific context
  - Access to dynamic network state
  - Minimal trust in the client
Outline

✓ Motivation
✓ Existing Solutions
✓ POISE: Programmable In-Network Security
  - POISE Architecture
    - System Design
    - Policy Language
    - Compiler
    - Context Collector Module
  - Initial Results
  - Future Work
  - Conclusion
Threat Model

BYOD devices can be compromised by malicious apps, but:

- Enterprise network is trustworthy
- OS kernel on BYOD device and context-collecting module are intact
System Design

Context Collector

Compiler

Switch (P4) Program

POISE Policy

Config File

App info
Sys info
Sensor info

Employee database

Access Point

Sales records

Enterprise Network

BYOD Device

Policy Language

Compiler Runtime
POISE Policy Language

• Based on Pyretic NetCore
  – Match-action based control flow

• Key additions
  – Added support for device contexts

Examples:
• Allow access only if admin is online
  
  ```
  def adminlst = [“Bob”, “Alice”]
  c = count(match(usr in adminlst))
  match(c>0) >>
  fwd(server)
  ```

• Distance-based access control
  
  ```
  if ((lat-x)*(lat-x)+(lon-y)*(lon-y) < D)
  then  fwd(server)
  else drop
  ```
Compiler

Input: POISE policy program
Output: Switch programs, Context collector configuration files

Monitor → Switch read/write register
Constant list → Switch match/action table
Context variable → Customized header

→ Config file entry
Context Collector Module

Two strategies:

- Add headers to all outgoing packets
- Send separate context packets
  - Periodically
  - When an app opens a new socket
  - When an dormant socket resumes activity
Outline

✓ Motivation
✓ Existing Solutions
✓ POISE: Programmable In-Network Security
✓ POISE Architecture
  ✓ System Design
  ✓ Policy Language
  ✓ Compiler
  ✓ Context Collector Module

- Initial Results
- Future Work
- Conclusion
Experimental Setup

• Compiler not implemented yet
  – P4 Programs written by hand
  – Context Collector hardcoded

• Mobile traces collected on Nexus 5 running client module

• Traces injected into Mininet topology with one switch and 6 end hosts
Evaluation Questions

• How much CPU overhead does POISE place on the client device?

• How much traffic overhead does POISE add?

• How effective is POISE at expressing sophisticated policies?
Results: Overhead*

- < 1% CPU overhead on client
- Additional traffic < 11.2 kbps

*Overhead measured when sending packets periodically, as opposed to tagging all outgoing packets
## Results: Expressiveness + Effectiveness

<table>
<thead>
<tr>
<th>Policy</th>
<th>Detected?</th>
</tr>
</thead>
<tbody>
<tr>
<td>Block services during working hours</td>
<td>✓</td>
</tr>
<tr>
<td>Direct traffic from guests through a middlebox</td>
<td>✓</td>
</tr>
<tr>
<td>Allow access only if user is close to a location</td>
<td>✓</td>
</tr>
<tr>
<td>Allow access only if admin is online</td>
<td>✓</td>
</tr>
<tr>
<td>Block requests without user interaction</td>
<td>✓</td>
</tr>
<tr>
<td>Scrub traffic if UIs are overlapping</td>
<td>✓</td>
</tr>
<tr>
<td>Conduct DPI if the camera or recorder is on</td>
<td>✓</td>
</tr>
</tbody>
</table>

We can support both existing and new policies!
Ongoing Work

• Build a full prototype, including a functional compiler

• Enhance threat model
  – Move information collector to kernel
  – Look for ways to address malicious/compromised kernels

• Perform larger-scale experiments on real-world BYOD traces
Conclusion

• Motivation: BYOD presents new security threats

• Existing approaches have pros and cons:
  – Client-side solutions
  – Server-side solutions

• Our approach: POISE (programmable in-network security)
  – Decouple context collection from policy enforcement
  – Leverage programmable data planes
  – Best of both worlds: context-aware and network-aware

• Preliminary results:
  – < 1% overhead
  – Can support a wide variety of context-aware policies
Questions and Issues

• Circumvention
  – What happens if a malicious app gets root access?

• Usability
  – How easy is POISE for network administrators to use?

• Testing
  – What kinds of testing/metrics would be important?

Thank you!