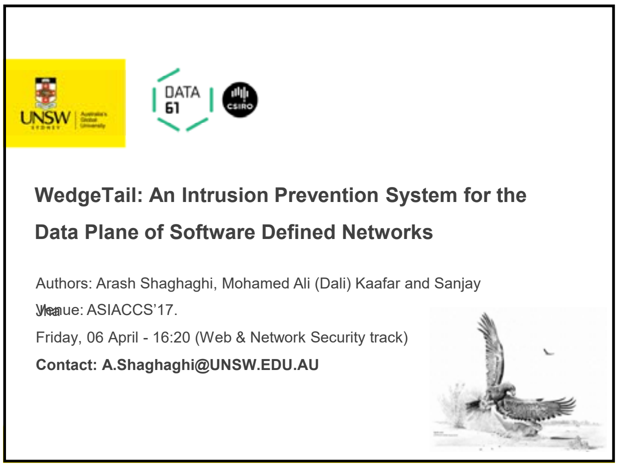


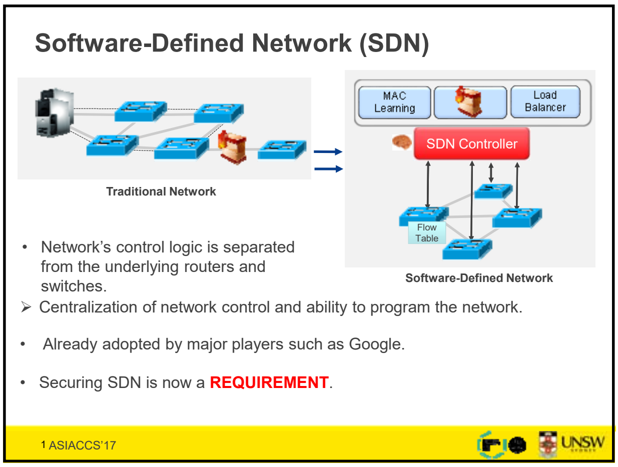
WedgeTail: An Intrusion Prevention System for the Data Plane of Software Defined Networks

Authors: Arash Shaghaghi, Mohamed Ali (Dali) Kaafar and Sanjay Jha

Venue: ASIACCS'17.

Friday, 06 April - 16:20 (Web & Network Security track)

Contact: A.Shaghaghi@UNSW.EDU.AU

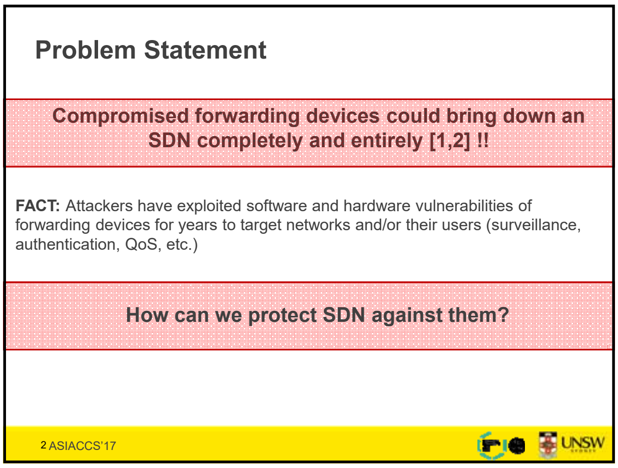


Software-Defined Network (SDN)

- Network's control logic is separated from the underlying routers and switches.

➢ Centralization of network control and ability to program the network.

- Already adopted by major players such as Google.
- Securing SDN is now a **REQUIREMENT**.

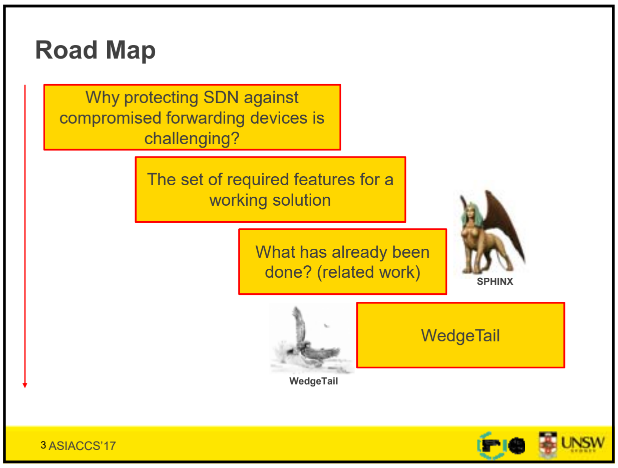


Problem Statement

Compromised forwarding devices could bring down an SDN completely and entirely [1,2] !!

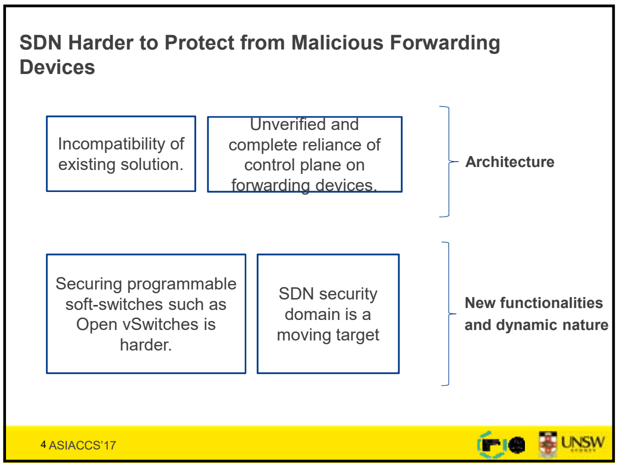
FACT: Attackers have exploited software and hardware vulnerabilities of forwarding devices for years to target networks and/or their users (surveillance, authentication, QoS, etc.)

How can we protect SDN against them?



Road Map

- Why protecting SDN against compromised forwarding devices is challenging?
- The set of required features for a working solution
- What has already been done? (related work)
- WedgeTail



SDN Harder to Protect from Malicious Forwarding Devices

Incompatibility of existing solution.

Unverified and complete reliance of control plane on forwarding devices.

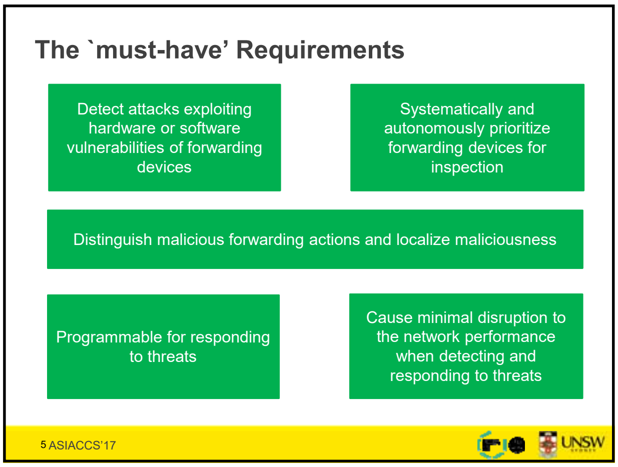
Architecture

Securing programmable soft-switches such as Open vSwitches is harder.

SDN security domain is a moving target

New functionalities and dynamic nature

4 ASIACCS'17



The `must-have' Requirements

Detect attacks exploiting hardware or software vulnerabilities of forwarding devices

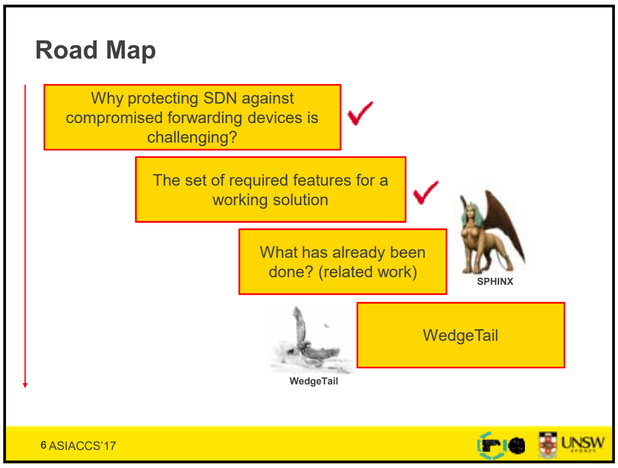
Systematically and autonomously prioritize forwarding devices for inspection

Distinguish malicious forwarding actions and localize maliciousness

Programmable for responding to threats

Cause minimal disruption to the network performance when detecting and responding to threats

5 ASIACCS'17



Road Map

Why protecting SDN against compromised forwarding devices is challenging? ✓

The set of required features for a working solution ✓

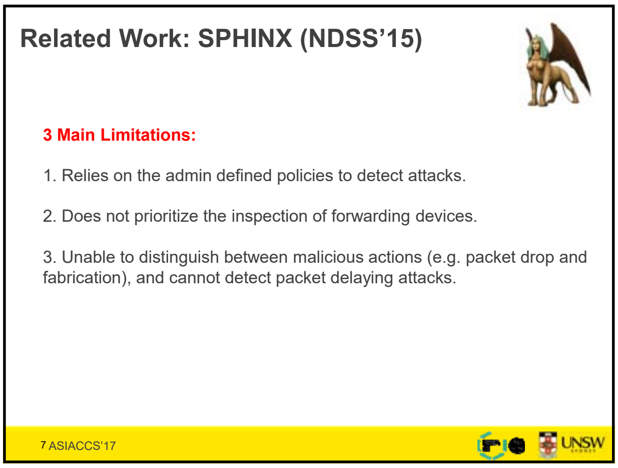
What has already been done? (related work)

SPHINX

WedgeTail

WedgeTail

6 ASIACCS'17



Related Work: SPHINX (NDSS'15)

3 Main Limitations:

1. Relies on the admin defined policies to detect attacks.

2. Does not prioritize the inspection of forwarding devices.

3. Unable to distinguish between malicious actions (e.g. packet drop and fabrication), and cannot detect packet delaying attacks.

7 ASIACCS'17

WedgeTail: Overview

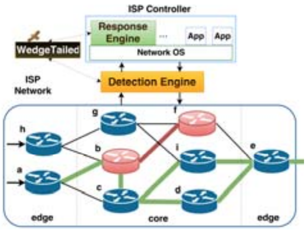
- Controller-agnostic Intrusion Prevention System (IPS) designed to ‘hunt’ for forwarding devices failing to process packets as expected.
- Addresses all the `must-have’ features.

How it works?

Target Identification

Attack Detection

Attack Mitigation



WedgeTail: Overview

Target Identification
1. Trajectory Creation
2. Scanning Zones

Attack Detection

Attack Mitigation

WedgeTail: Overview

Target Identification

1. Packets as `random-walkers’ → 2. Packet movements as trajectories in a geometric space → 3. Store all the trajectories in a trajectory database.

Cluster forwarding devices into scanning groups of varying priority depending on the cumulative frequency of occurrence in packet paths traversing the network.

Attack Detection

Attack Mitigation

WedgeTail: Overview

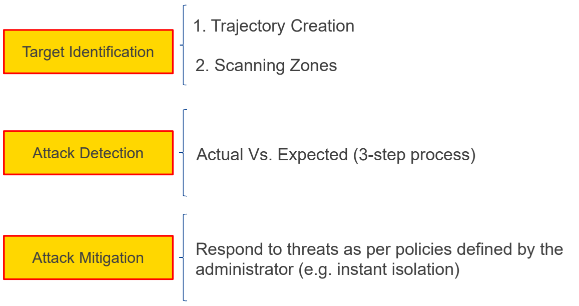
Target Identification
1. Trajectory Creation
2. Scanning Zones

Attack Detection

Compute the expected and actual trajectories to identify malicious actions (Packet **Drop, Misroute, Replay, Delay & Generation**) + Localize malicious forwarding devices.

Attack Mitigation

WedgeTail: Overview



Target Identification

1. Trajectory Creation

2. Scanning Zones

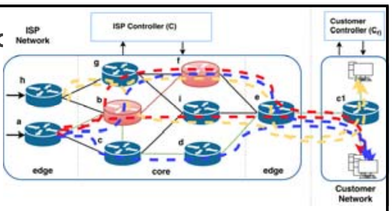
Attack Detection

Actual Vs. Expected (3-step process)

Attack Mitigation

Respond to threats as per policies defined by the administrator (e.g. instant isolation)

Trajectory Creatic

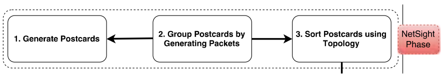


Packet Trajectory (TR): The route a uniquely identifiable packet takes while traversing the network from one forwarding device to another.

Actual Packet Trajectory:

1. **NetSight (NSDI'14):** Network troubleshooting solution that allows SDN applications to retrieve the complete packet history (preferred generic approach).
2. **Custom labelling:** Deterministic hash function over the packet header and use this hash to track packet as it traverses the network (for smaller networks).

NetSight - NSDI'14.



1. Generate Postcards

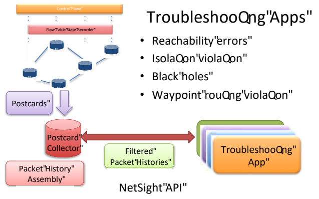
2. Group Postcards by Generating Packets

3. Sort Postcards using Topology

NetSight Phase

- More advanced, more systematic.
- Disadvantage? Not available everywhere.

Implemented an integration using their API for WedgeTailed.



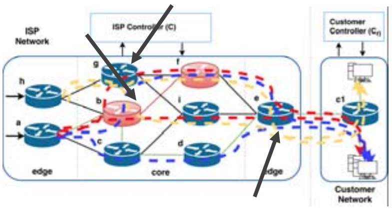
Control"Plane"
Flow"Table"State"Recorder"
Postcards"
Postcard" Collector"
Packet"History" Assembly"
Filtered" Packet"Histories"
TroubleshooQng" App"
NetSight"API"

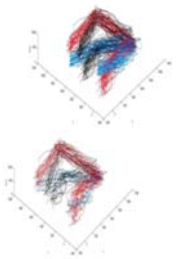
TroubleshooQng"Apps"

- Reachability"errors"
- IsolaQon"violaQon"
- Black"holes"
- Waypoint"rouQng"violaQon"

Scanning Zones

- Keep track of trajectories for all packets on all ports over time t.
- Using `Unsupervised Trajectory Sampling' [5] reduce the large set of trajectories into a representative sample that encapsulates the most commonly visited forwarding devices.





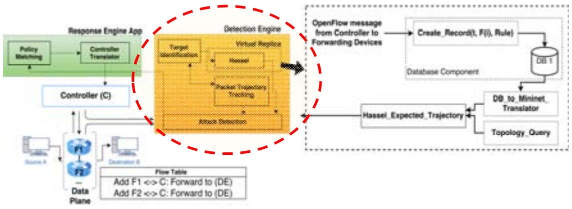
Scanning Zones

1. Model ALL trajectories of the TD in approximate way as vectors (preserves/lossless the mobility pattern of each trajectory, speeds up the computation)
2. SyTra: Represent each trajectory by a continuous function that implicitly describes the representativeness of each constituent part of it in respect to the whole TD. (relaxes time-based representation of the centroid trajectories, allows the modelling of the mobility pattern of each trajectory at a higher level of abstraction)
3. T-Sampling: takes into account not only the most (dense, frequent) but also the least representative.

Attack Detection (Step 1 of 3)

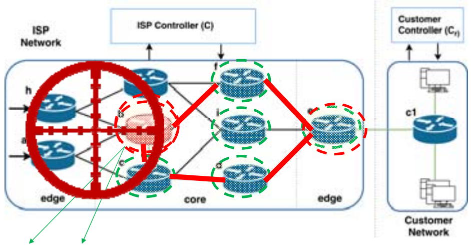
1. Compute Expected Trajectories

I. Intercept the relevant OpenFlow messages.
II. Maintain a virtual replica of the network.



Attack Detection (Step 1 of 3)

1. Compute the Expected Trajectories



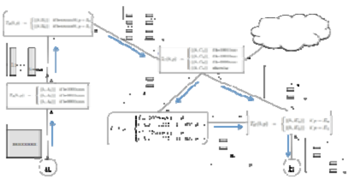
Scan FD(b) on Port(i)
Select FD(b) as source, → **Retrieve packets with matching trajectories**
All the others as destination.

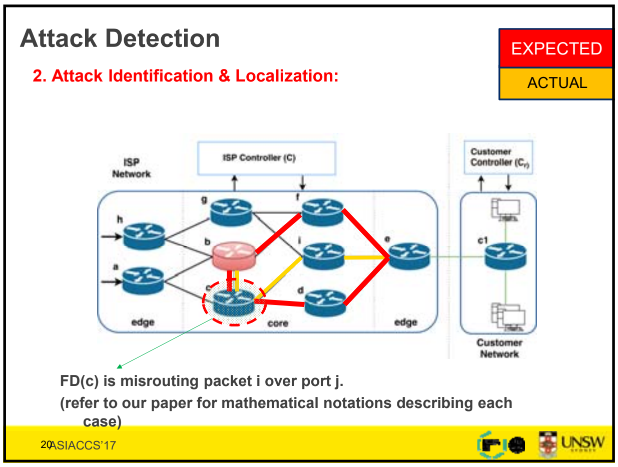
Header Space Analysis:

<u>Protocol independent abstraction for packets and forwarding functionality of networks.</u>

A L-bit packet header as L-dimensional space, and models all processes of routers and middleboxes as transfer functions, which transform subspaces of the L-dimensional space to other subspaces.

By analyzing forwarding rules of the network, HSA can calculate the path a packet traversing the network on a certain port will take.





Attack Detection

EXPECTED

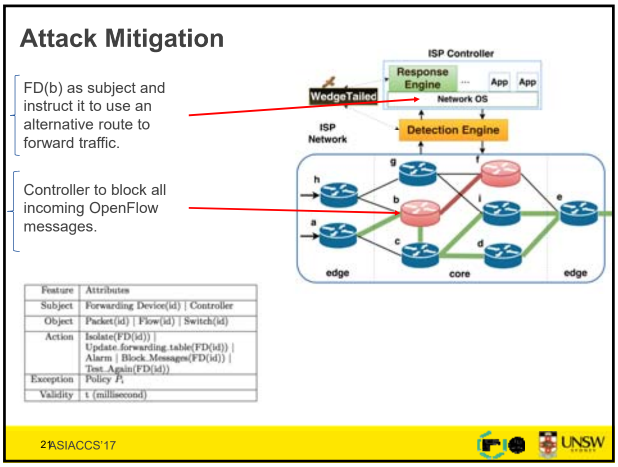
ACTUAL

2. Attack Identification & Localization:

FD(c) is misrouting packet i over port j.

(refer to our paper for mathematical notations describing each case)

20 ASIACCS'17



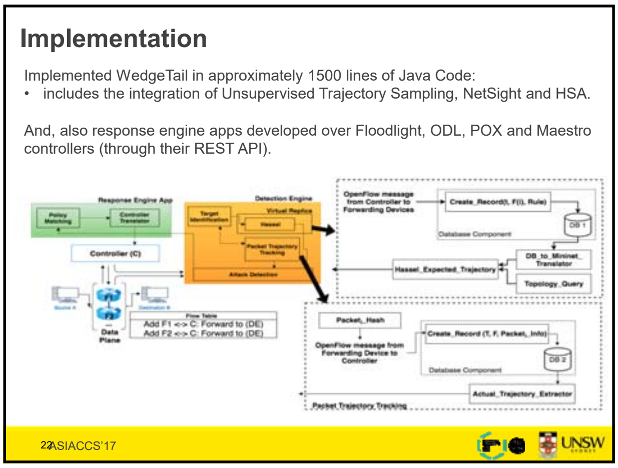
Attack Mitigation

FD(b) as subject and instruct it to use an alternative route to forward traffic.

Controller to block all incoming OpenFlow messages.

Feature	Attributes
Subject	Forwarding Device(id) | Controller
Object	Packet(id) | Flow(id) | Switch(id)
Action	Isolate(FD(id)) | Update_forwarding_table(FD(id)) | Alarm | Block_Messages(FD(id)) | Test_Again(FD(id))
Exception	Policy P_i
Validity	t (millisecond)

21 ASIACCS'17



Implementation

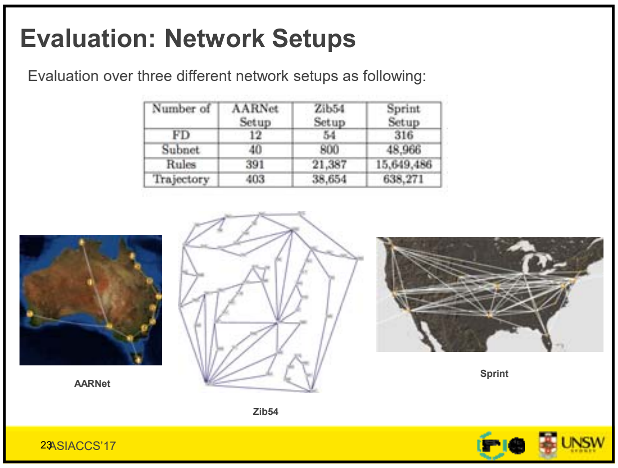
Implemented WedgeTail in approximately 1500 lines of Java Code:

- includes the integration of Unsupervised Trajectory Sampling, NetSight and HSA.

And, also response engine apps developed over Floodlight, ODL, POX and Maestro controllers (through their REST API).

Response Engine App
Policy Matching
Controller Translator
Detection Engine
Target Identification
Virtual Replica
Hassel
Packet Trajectory Tracking
Attack Detection
Controller (C)
Source A
Destination B
F1
F2
Data Plane
Flow Table
Add F1 <-> C: Forward to (DE)
Add F2 <-> C: Forward to (DE)
OpenFlow message from Controller to Forwarding Devices
Create_Record(t, F(i), Rule)
DB 1
Database Component
DB_to_Mininet_Translator
Hassel_Expected_Trajectory
Topology_Query
Packet_Hash
OpenFlow message from Forwarding Device to Controller
Create_Record (T, F, Packet_Info)
DB 2
Database Component
Actual_Trajectory_Extractor
Packet Trajectory Tracking

22 ASIACCS'17



Evaluation: Network Setups

Evaluation over three different network setups as following:

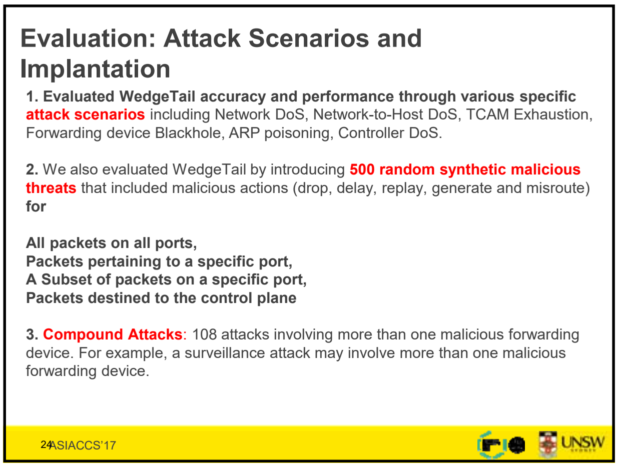
Number of	AARNet Setup	Zib54 Setup	Sprint Setup
FD	12	54	316
Subnet	40	800	48,966
Rules	391	21,387	15,649,486
Trajectory	403	38,654	638,271

AARNet

Zib54

Sprint

23 ASIACCS'17



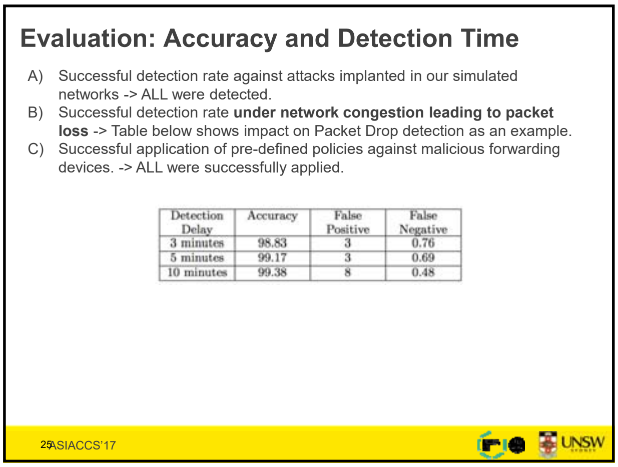
Evaluation: Attack Scenarios and Implantation

1. Evaluated WedgeTail accuracy and performance through various specific **attack scenarios** including Network DoS, Network-to-Host DoS, TCAM Exhaustion, Forwarding device Blackhole, ARP poisoning, Controller DoS.

2. We also evaluated WedgeTail by introducing **500 random synthetic malicious threats** that included malicious actions (drop, delay, replay, generate and misroute) **for**

All packets on all ports,
Packets pertaining to a specific port,
A Subset of packets on a specific port,
Packets destined to the control plane

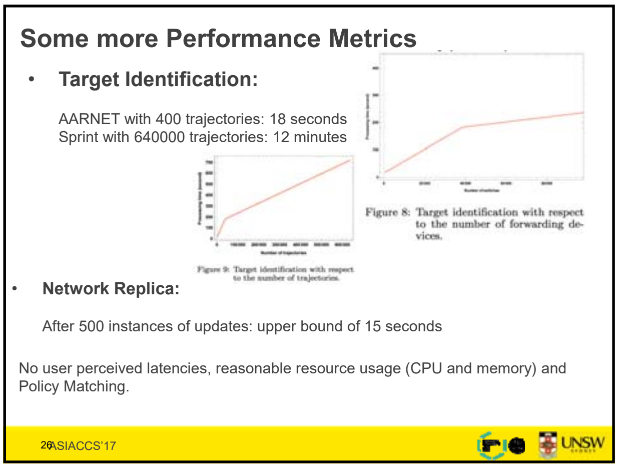
3. Compound Attacks: 108 attacks involving more than one malicious forwarding device. For example, a surveillance attack may involve more than one malicious forwarding device.



Evaluation: Accuracy and Detection Time

A) Successful detection rate against attacks implanted in our simulated networks -> ALL were detected.
B) Successful detection rate **under network congestion leading to packet loss** -> Table below shows impact on Packet Drop detection as an example.
C) Successful application of pre-defined policies against malicious forwarding devices. -> ALL were successfully applied.

Detection Delay	Accuracy	False Positive	False Negative
3 minutes	98.83	3	0.76
5 minutes	99.17	3	0.69
10 minutes	99.38	8	0.48



Some more Performance Metrics

- **Target Identification:**

  AARNET with 400 trajectories: 18 seconds
  Sprint with 640000 trajectories: 12 minutes

Figure 8: Target identification with respect to the number of forwarding devices.

Figure 9: Target identification with respect to the number of trajectories.

- **Network Replica:**

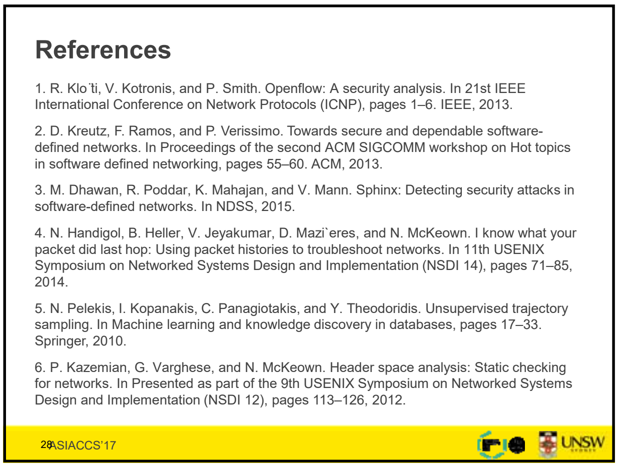
  After 500 instances of updates: upper bound of 15 seconds

No user perceived latencies, reasonable resource usage (CPU and memory) and Policy Matching.

Future Work

- Deploying WedgeTail over a real world network **focusing on scalability**.
- Evaluating WedgeTail's performance over **other attack scenarios** and use-cases such as **Virtualization, VM migration** and etc.
- WedgeTail currently analyzes **snapshots** and stability of these is challenging.
- WedgeTail compatibility with **distributed controllers** such as ONOS requires investigation.

Any Questions?



References

1. R. Klo`ti, V. Kotronis, and P. Smith. Openflow: A security analysis. In 21st IEEE International Conference on Network Protocols (ICNP), pages 1–6. IEEE, 2013.

2. D. Kreutz, F. Ramos, and P. Verissimo. Towards secure and dependable software-defined networks. In Proceedings of the second ACM SIGCOMM workshop on Hot topics in software defined networking, pages 55–60. ACM, 2013.

3. M. Dhawan, R. Poddar, K. Mahajan, and V. Mann. Sphinx: Detecting security attacks in software-defined networks. In NDSS, 2015.

4. N. Handigol, B. Heller, V. Jeyakumar, D. Mazi`eres, and N. McKeown. I know what your packet did last hop: Using packet histories to troubleshoot networks. In 11th USENIX Symposium on Networked Systems Design and Implementation (NSDI 14), pages 71–85, 2014.

5. N. Pelekis, I. Kopanakis, C. Panagiotakis, and Y. Theodoridis. Unsupervised trajectory sampling. In Machine learning and knowledge discovery in databases, pages 17–33. Springer, 2010.

6. P. Kazemian, G. Varghese, and N. McKeown. Header space analysis: Static checking for networks. In Presented as part of the 9th USENIX Symposium on Networked Systems Design and Implementation (NSDI 12), pages 113–126, 2012.