Automated Discovery of Cross-Plane Event-Based Vulnerabilities in Software-Defined Networking

Benjamin E. Ujcich[1], Samuel Jero[2], Richard Skowyra[2], Steven R. Gomez[2], Adam Bates[1], William H. Sanders[1], and Hamed Okhravi[2]

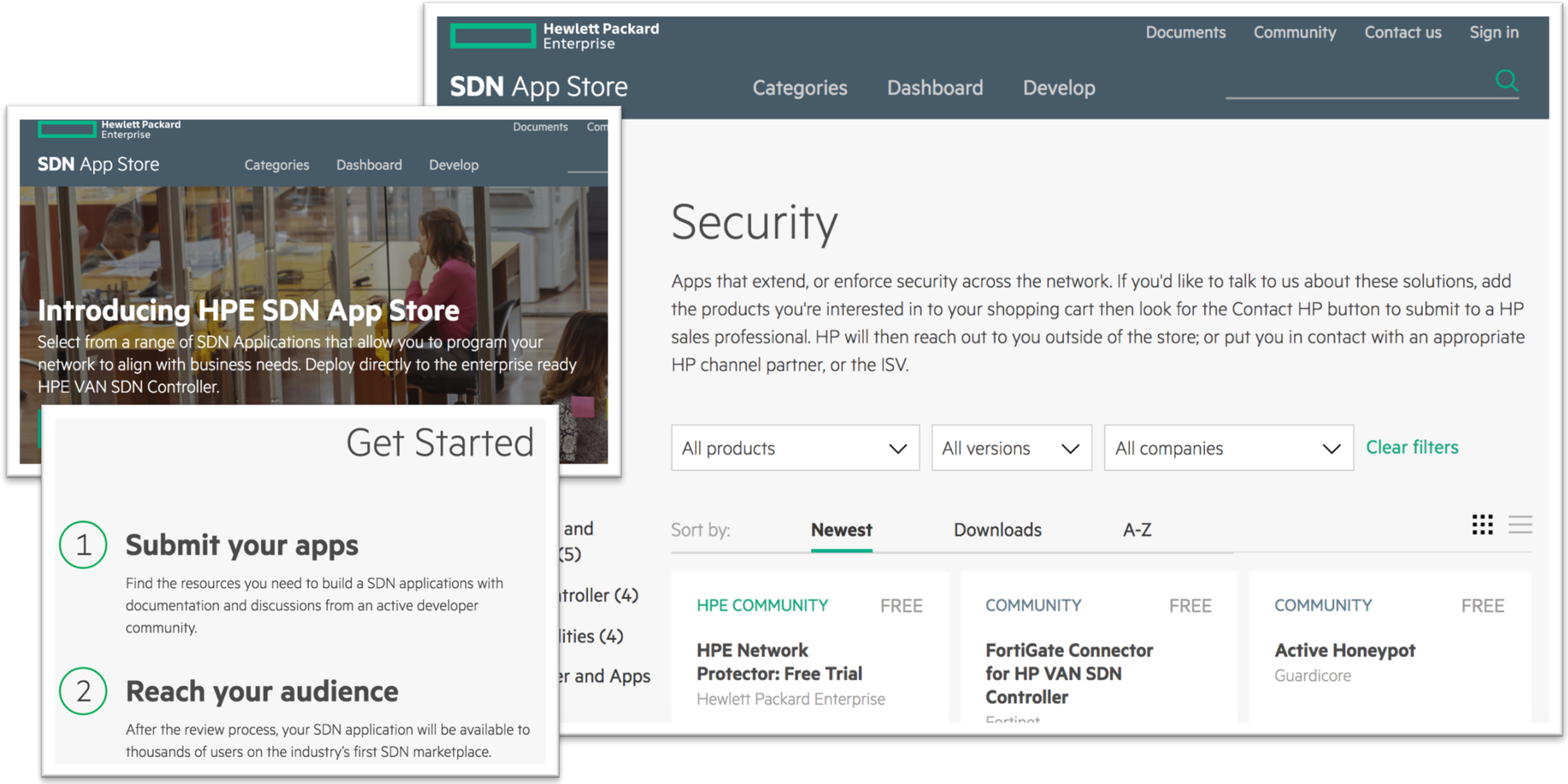
[1] University of Illinois at Urbana-Champaign, [2] MIT Lincoln Laboratory

2020 Network and Distributed System Security Symposium (NDSS)

February 25, 2020
San Diego, CA, USA

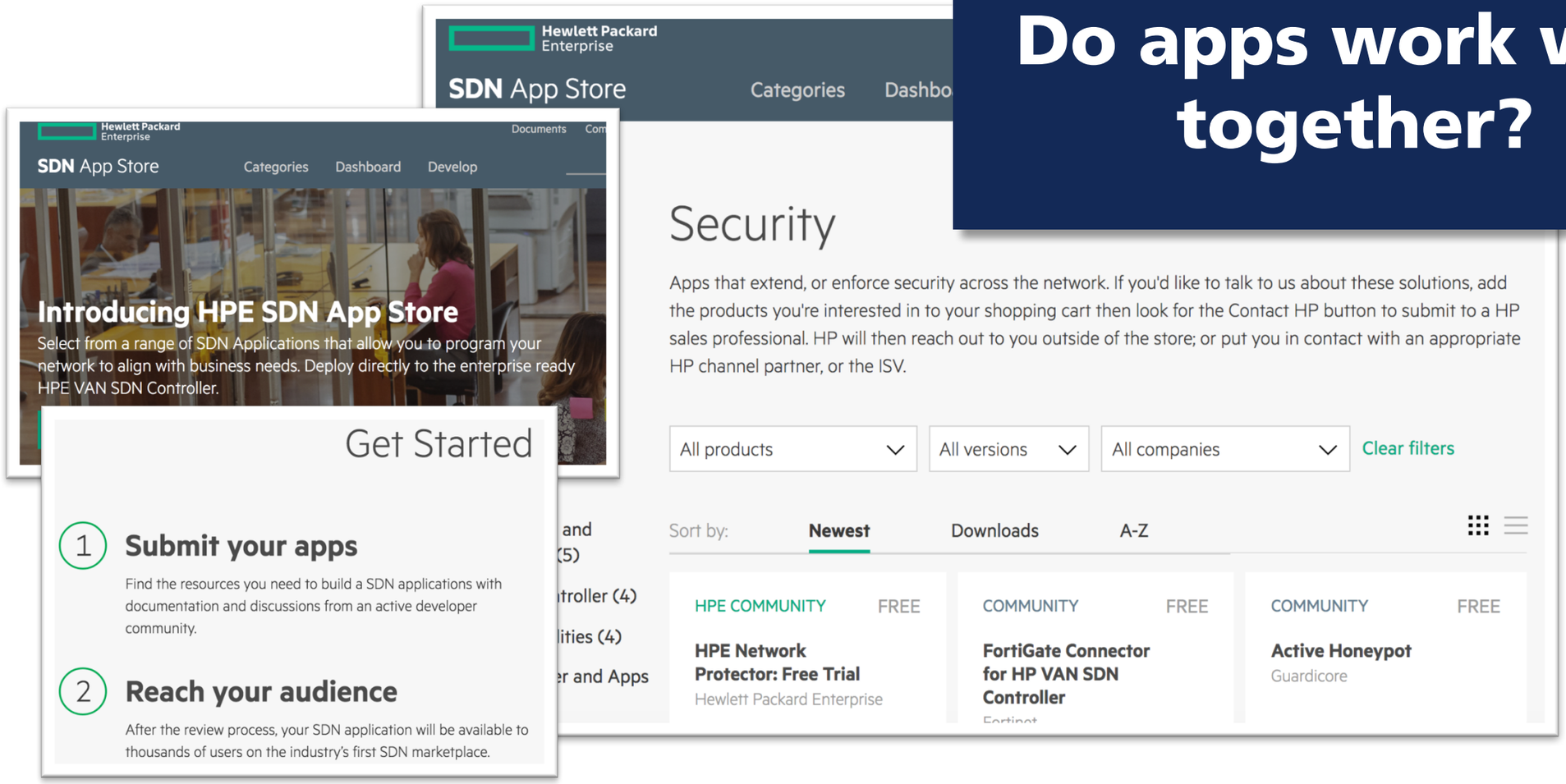
SDN is Everywhere!

Network “Appification”



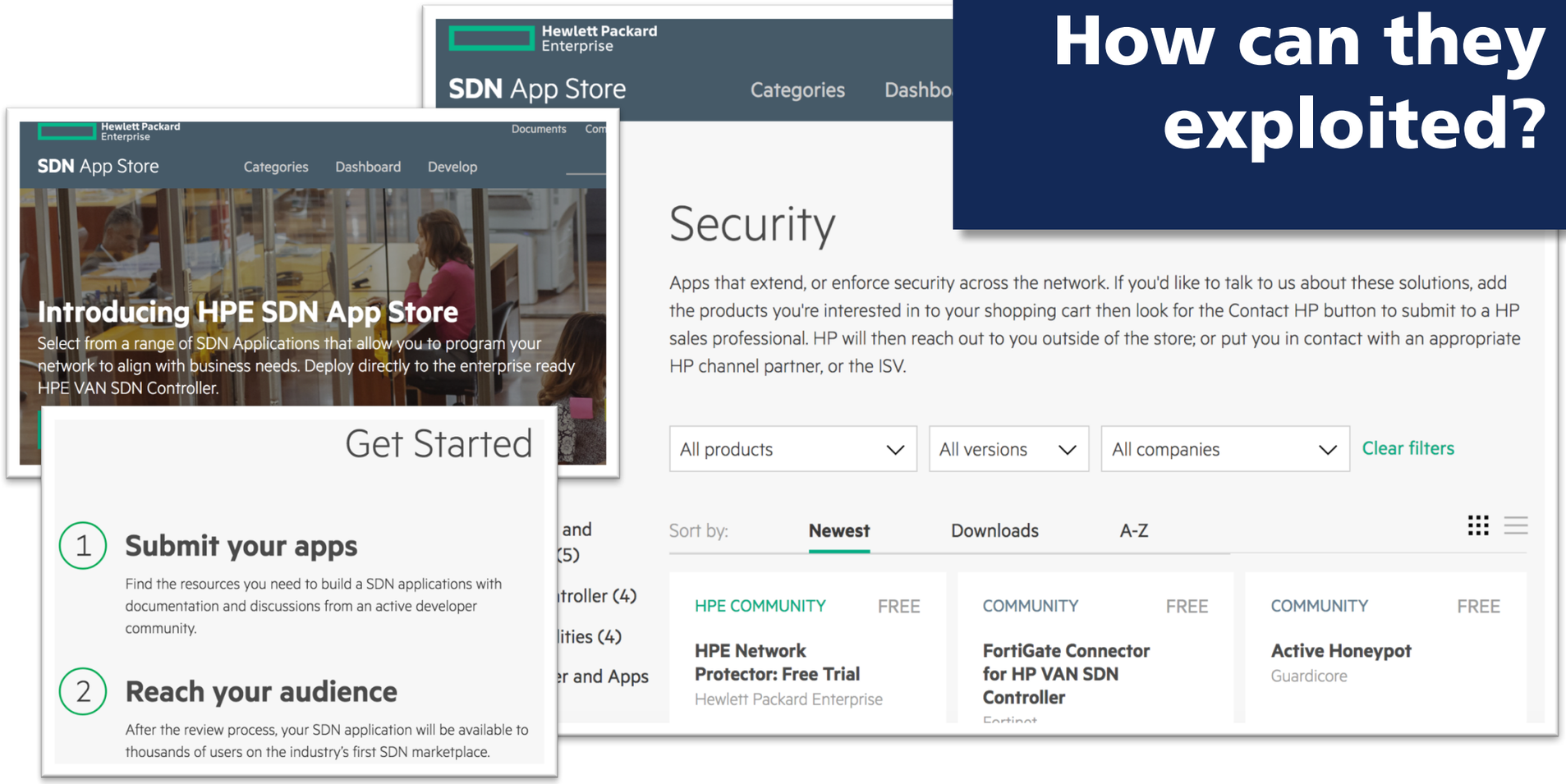
Network “Appification”

Do apps work well together?

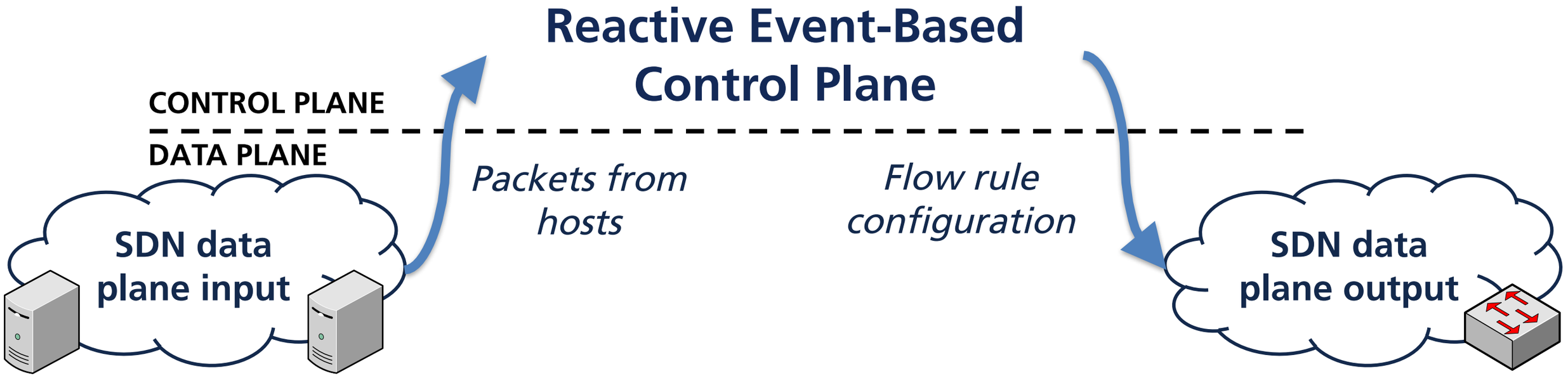


Network “Appification”

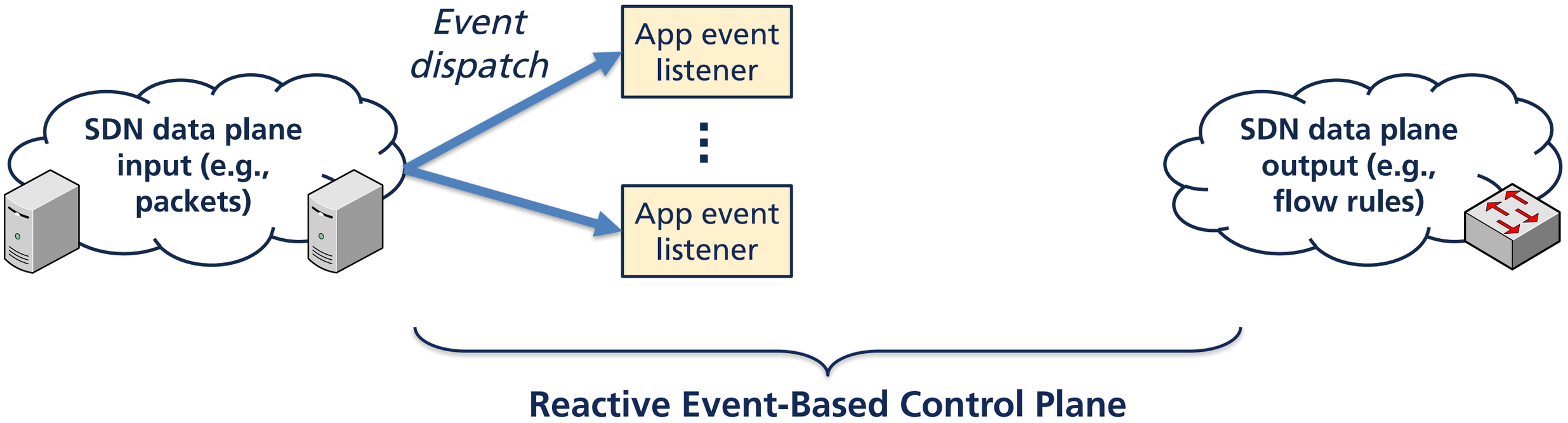
How can they be exploited?



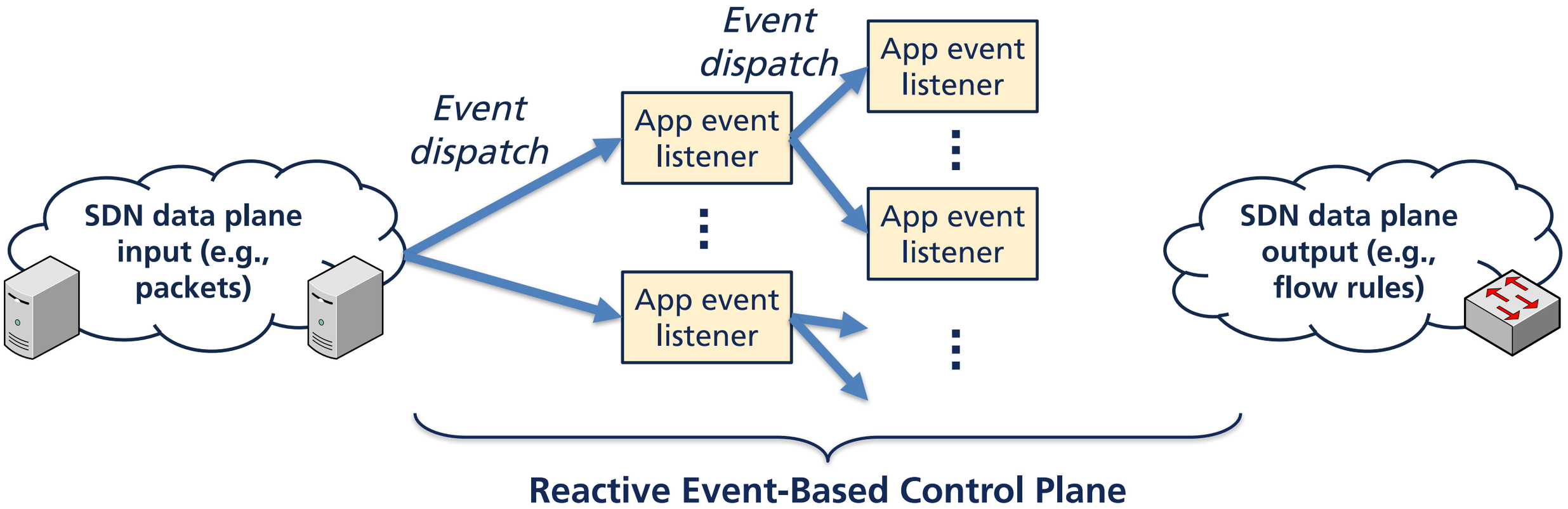
Cross-Plane Vulnerabilities



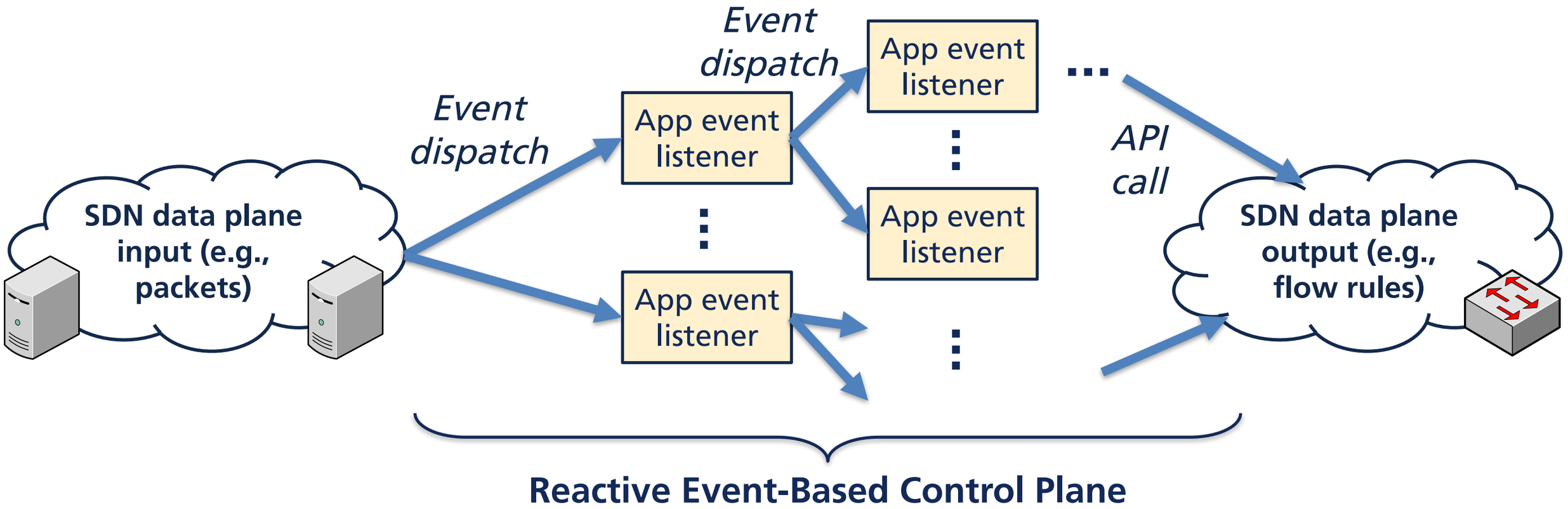
Cross-Plane Vulnerabilities



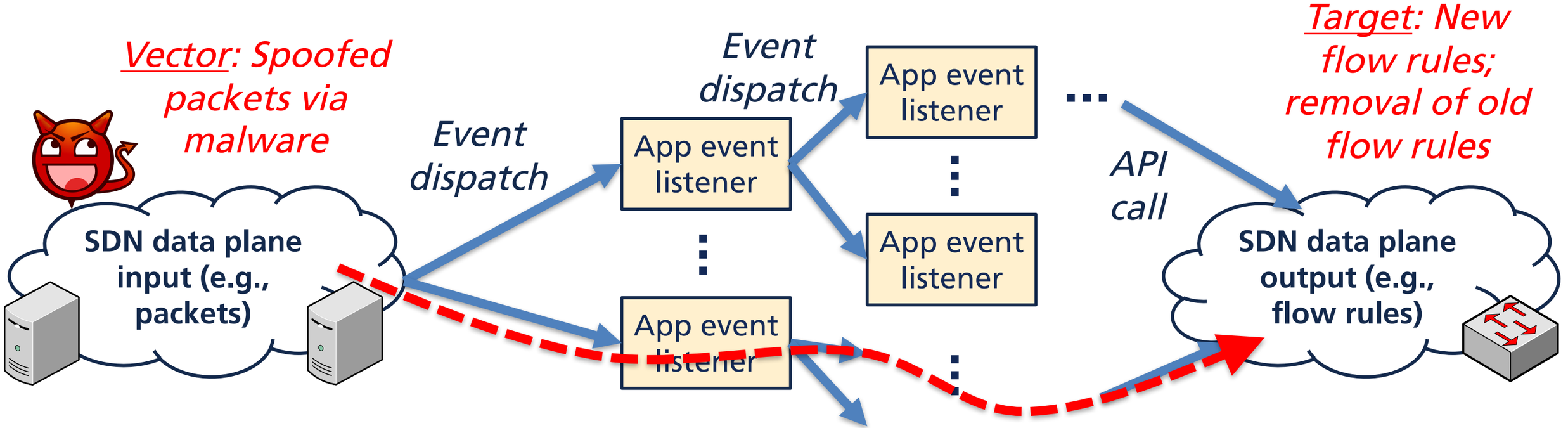
Cross-Plane Vulnerabilities



Cross-Plane Vulnerabilities



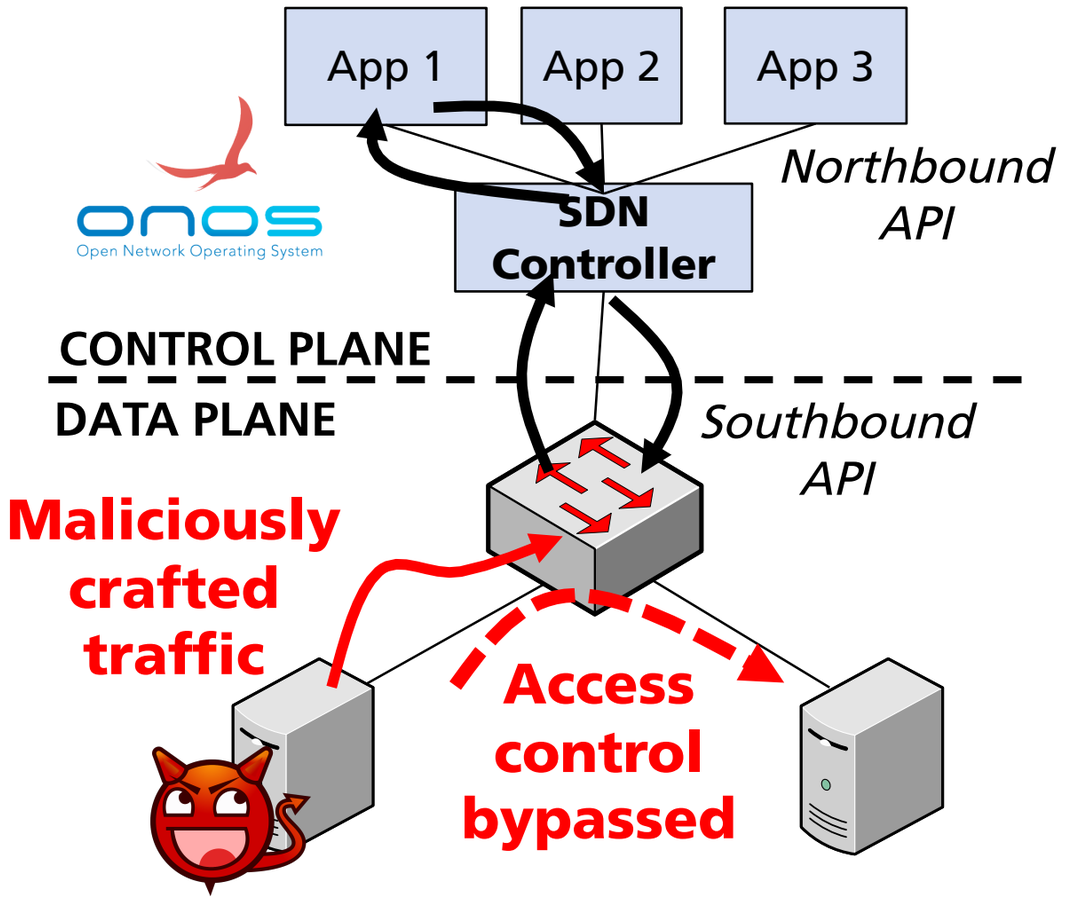
Cross-Plane Vulnerabilities: Exploitation



Insight: Missing event handling can be exploited

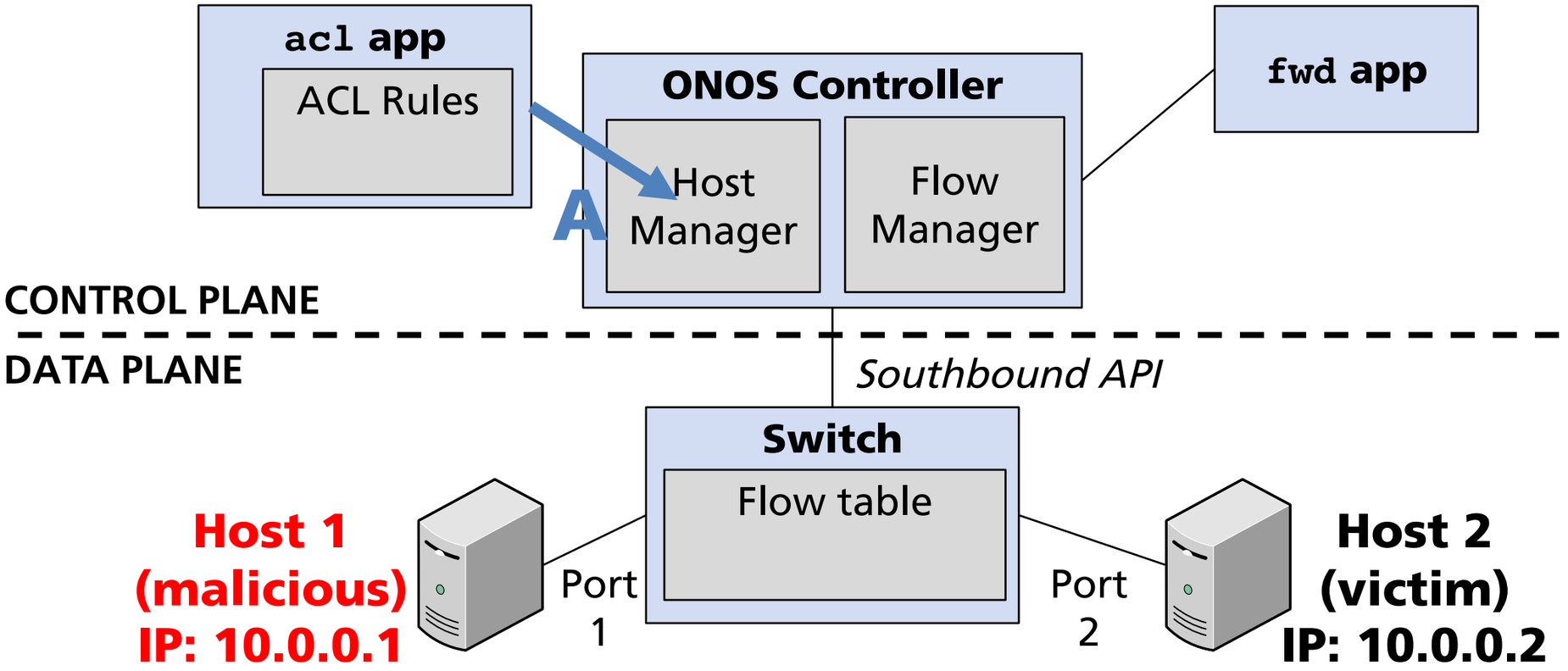
Data Plane Hosts as Attack Vectors

- Cross-app study led to explore hosts as attackers
- Discovered ONOS data plane firewall vulnerability → **arbitrary lateral movement**
- Reported to ONOS developers (CVE 2018-12691)



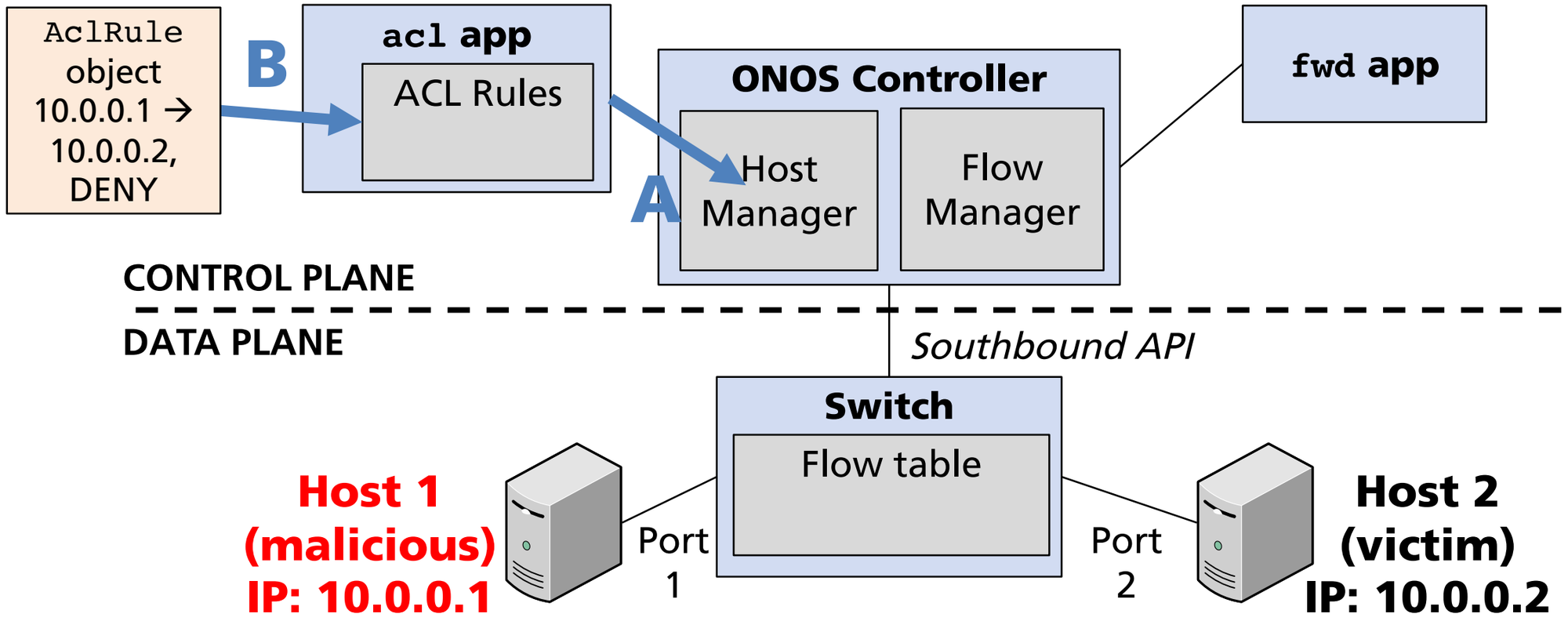
Anatomy of an Exploit

1. The access control app (`acl`) is activated and registers for any host events (A).



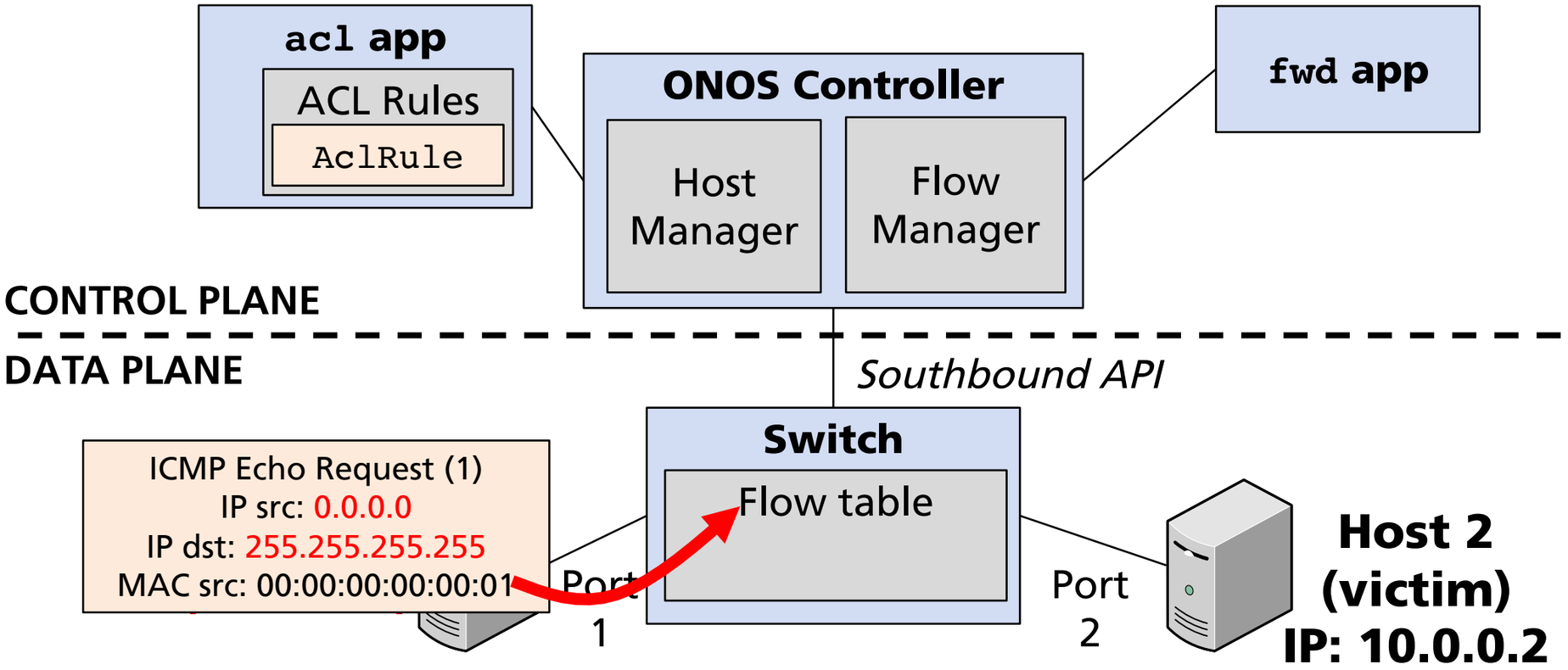
Anatomy of an Exploit

1. The access control app (`acl`) is activated and registers for any host events (A). The network operator adds access control policies (B).



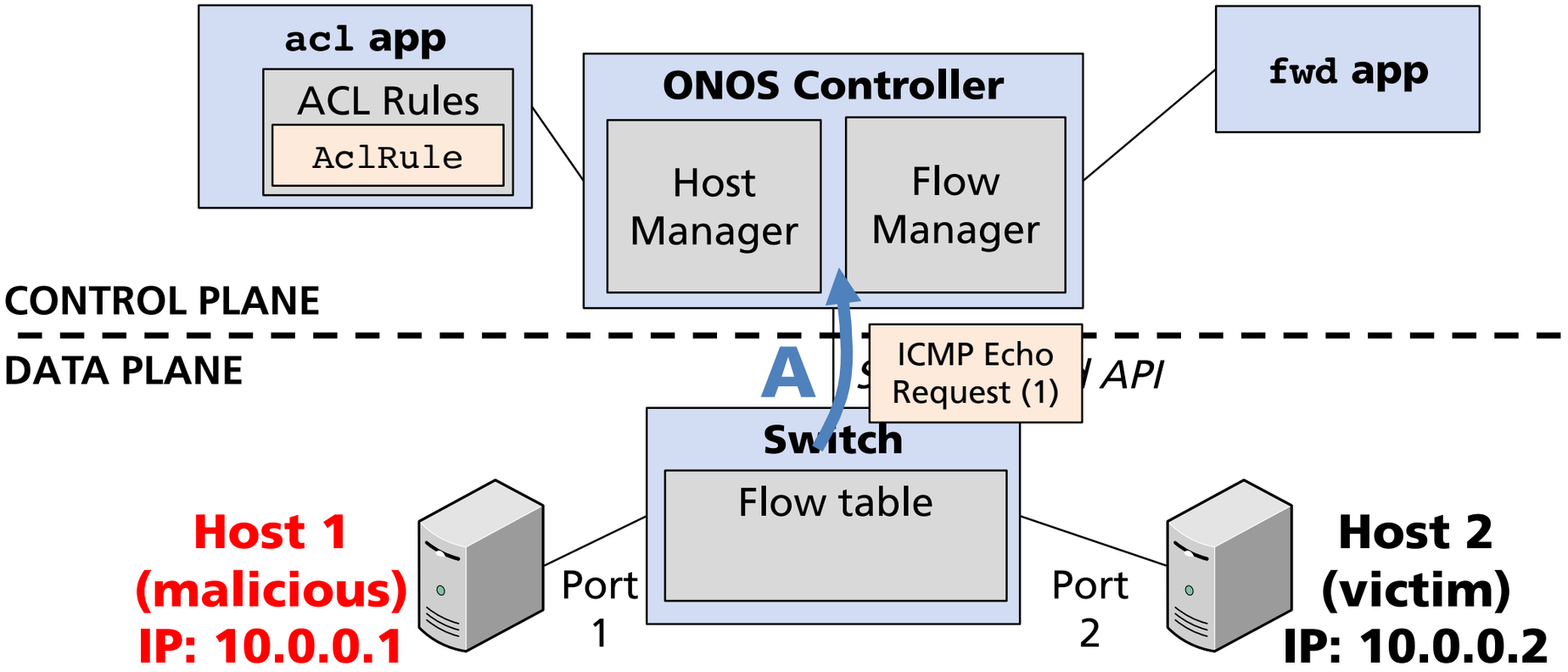
Anatomy of an Exploit

2. Host 1 sends a syntactically correct but semantically invalid ICMP packet with host 1's MAC address into the data plane.



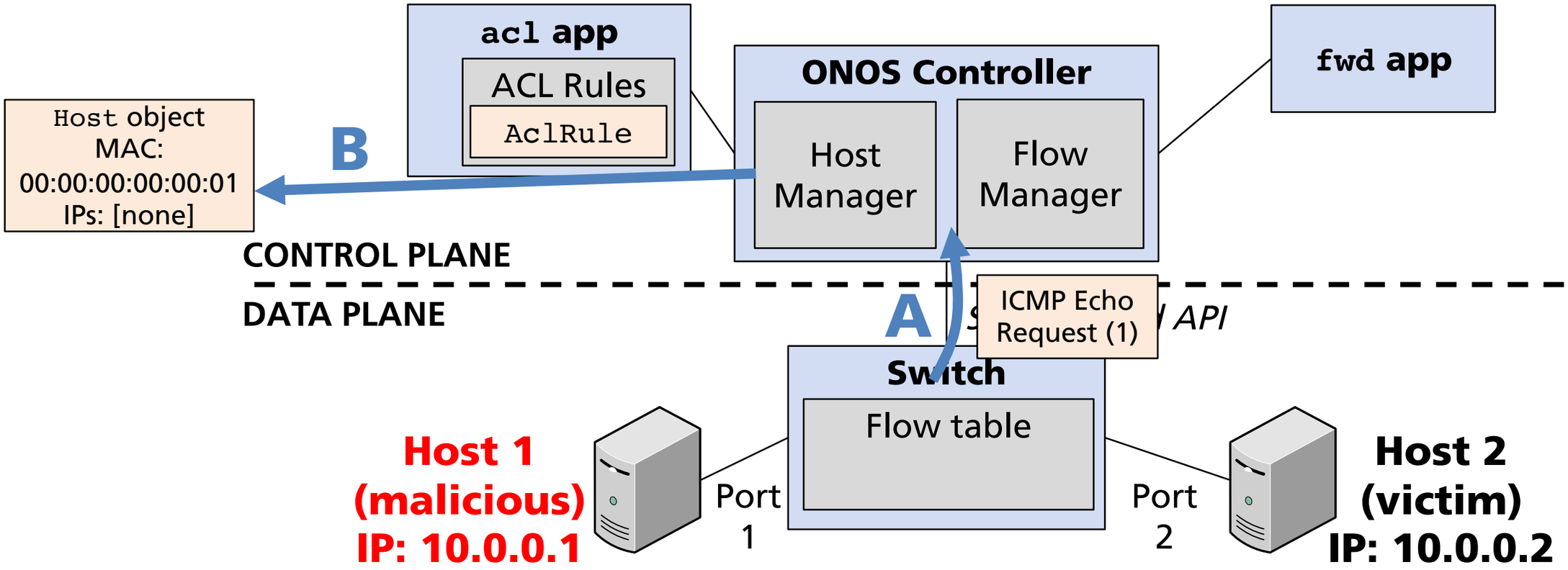
Anatomy of an Exploit

3. ONOS sees the packet (A)



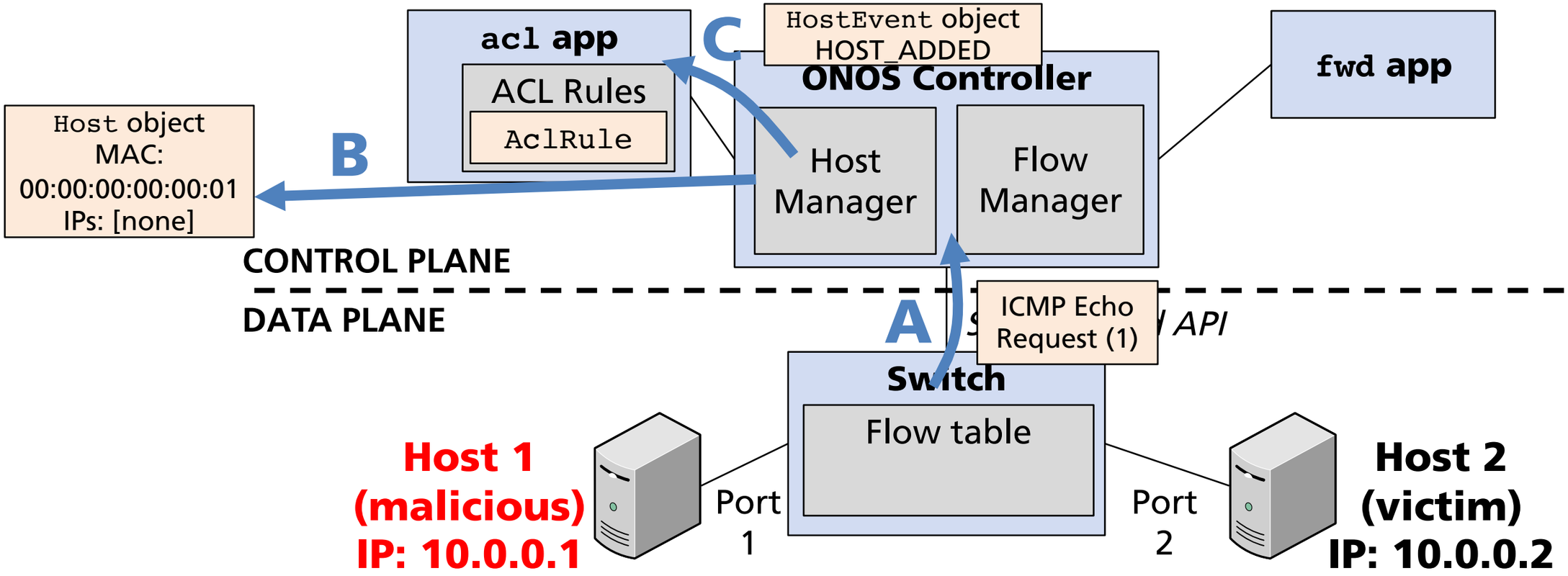
Anatomy of an Exploit

3. ONOS sees the packet (A) and registers a new host with its MAC address but not IP address (B).



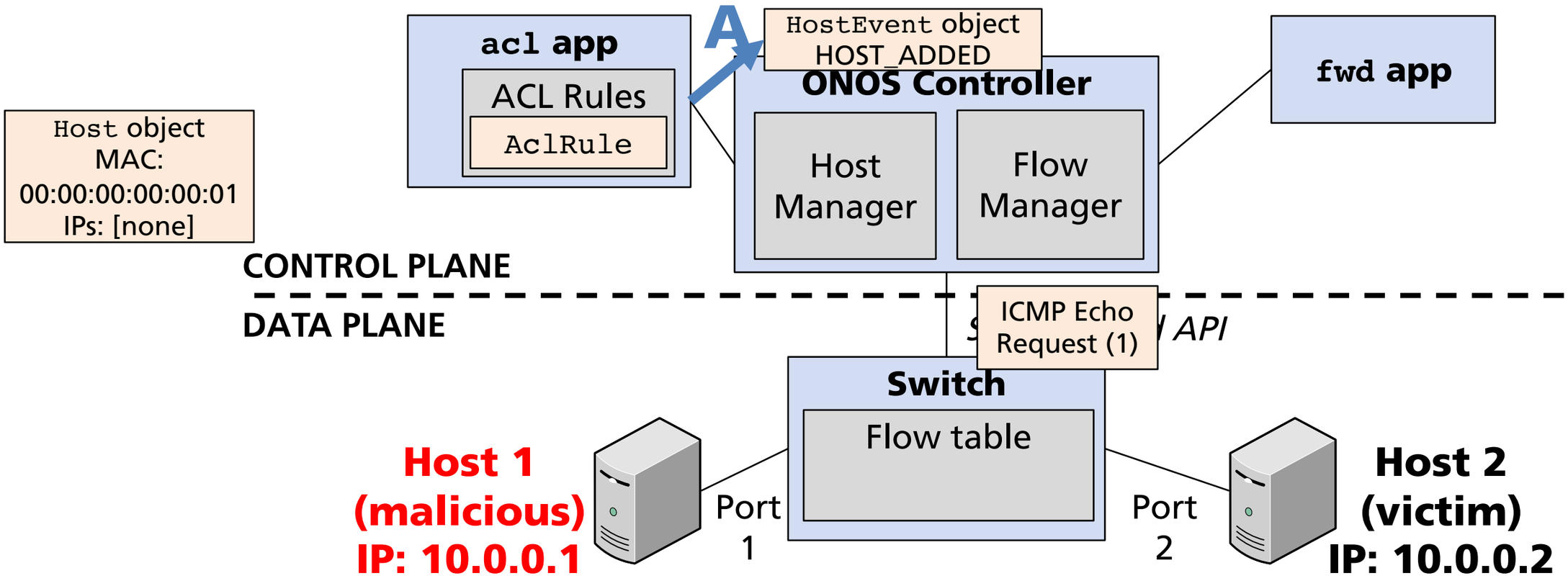
Anatomy of an Exploit

3. ONOS sees the packet (A) and registers a new host with its MAC address but not IP address (B). It generates a `HOST_ADDED` event (C).



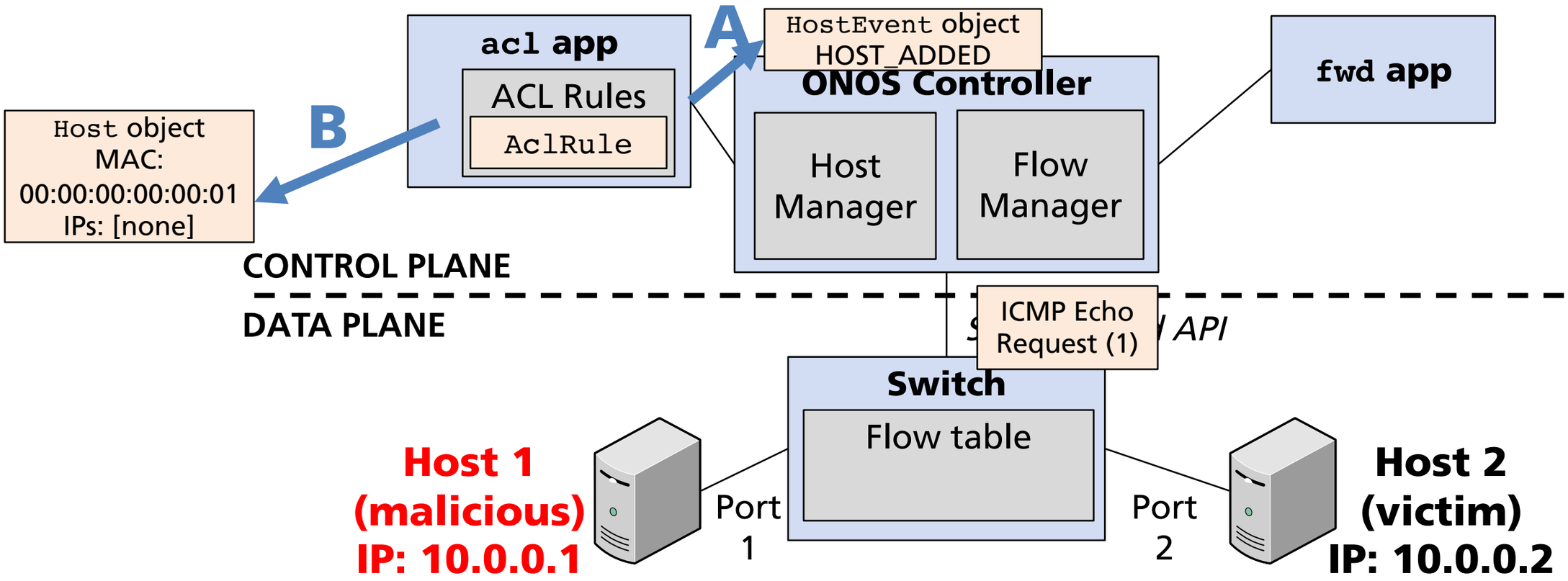
Anatomy of an Exploit

4. The `acl` app sees the `HOST_ADDED` event (A)



Anatomy of an Exploit

4. The `acl` app sees the `HOST_ADDED` event (A) and host (B), but since the host doesn't have an IP, the app does not insert flow deny rules.



Anatomy of an Exploit

4. The `acl` app sees the `HOST_ADDED` event (A) and host (B), but since the host doesn't have an IP, the app does not insert flow deny rules.

`AclManager.java`
handling new host events

```
@Override
public void event(HostEvent event) {
    // if a new host appears and an existing rule denies
    // its traffic, a new ACL flow rule is generated
    if (event.type() == HostEvent.Type.HOST_ADDED) {
        DeviceId deviceId = event.subject().location().device
        if (mastershipService.getLocalRole(deviceId) == Maste
            for (AclRule rule : aclStore.getAclRules()) {
                if (rule.action() != AclRule.Action.ALLOW) {
                    processHostAddedEvent(event, rule);
                }
            }
        }
    }
}
```

`AclManager.java` processing host events

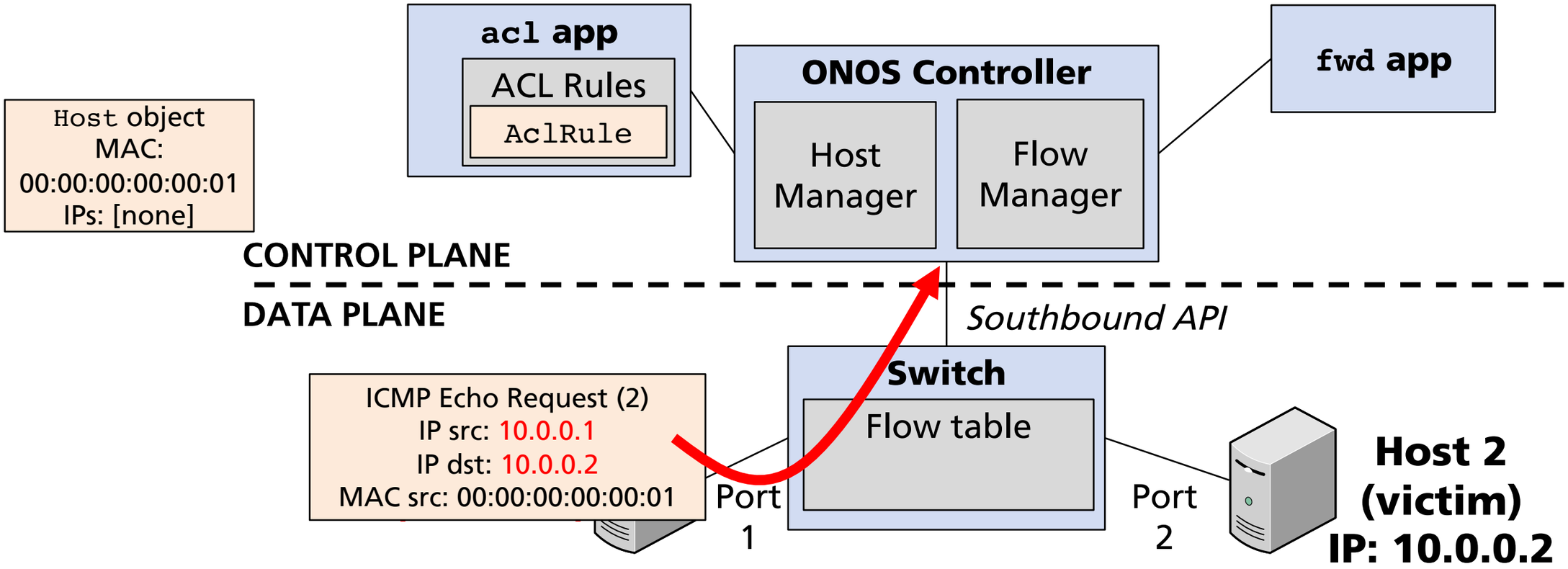
```
private void processHostAddedEvent(HostEvent event, AclRule rule) {
    DeviceId deviceId = event.subject().location().deviceId();
    for (IpAddress address : event.subject().ipAddresses()) {
        if ((rule.srcIp() != null) ?
                (checkIpInCidr(address.getIp4Address(), rule.srcIp()
                (checkIpInCidr(address.getIp4Address(), rule.dstIp()
            if (!aclStore.checkIfRuleWorksInDevice(rule.id(), devi
                List<RuleId> allowingRuleList = aclStore
                        .getAllowingRuleByDenyingRule(rule.id());
                if (allowingRuleList != null) {
                    for (RuleId allowingRuleId : allowingRuleList) {
                        generateAclFlow(aclStore.getAclRule(allowing
                    }
                }
                generateAclFlow(rule, deviceId);
            }
        }
    }
}
```

No IP addresses

Never gets called

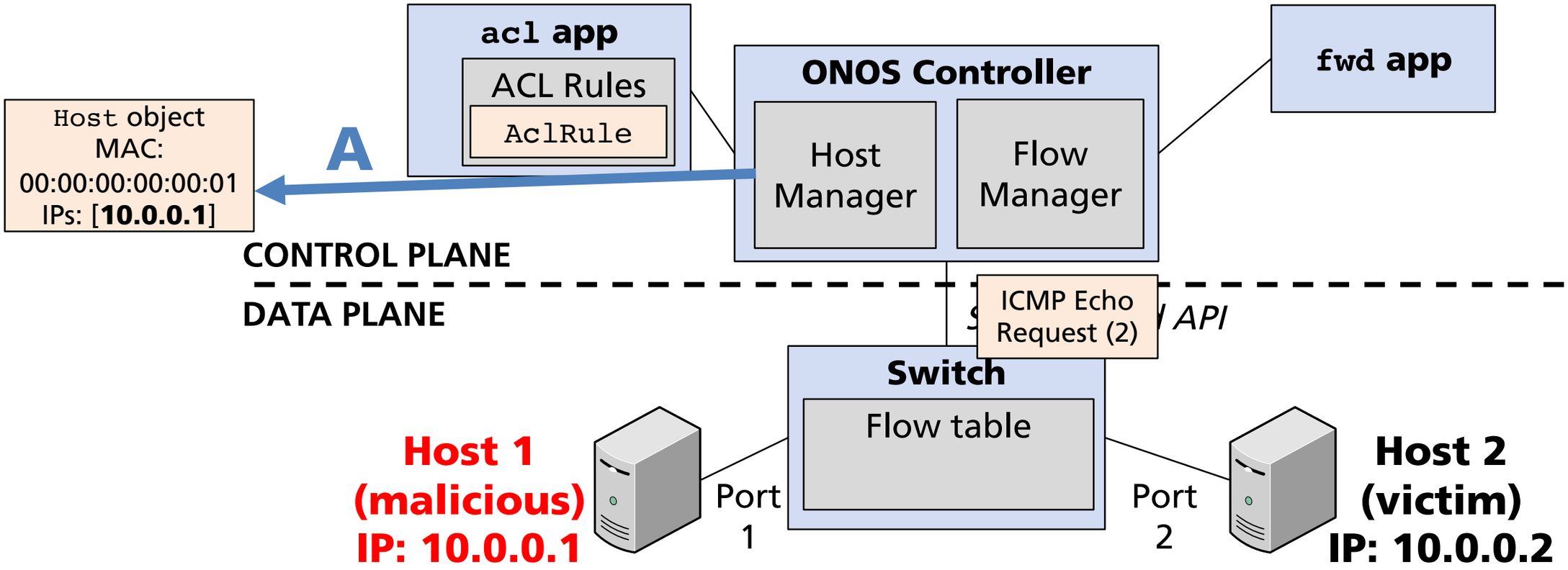
Anatomy of an Exploit

5. Host 1 attempts to send regular traffic to its desired victim destination (host 2). Since no matching flows exist, ONOS handles the packet.



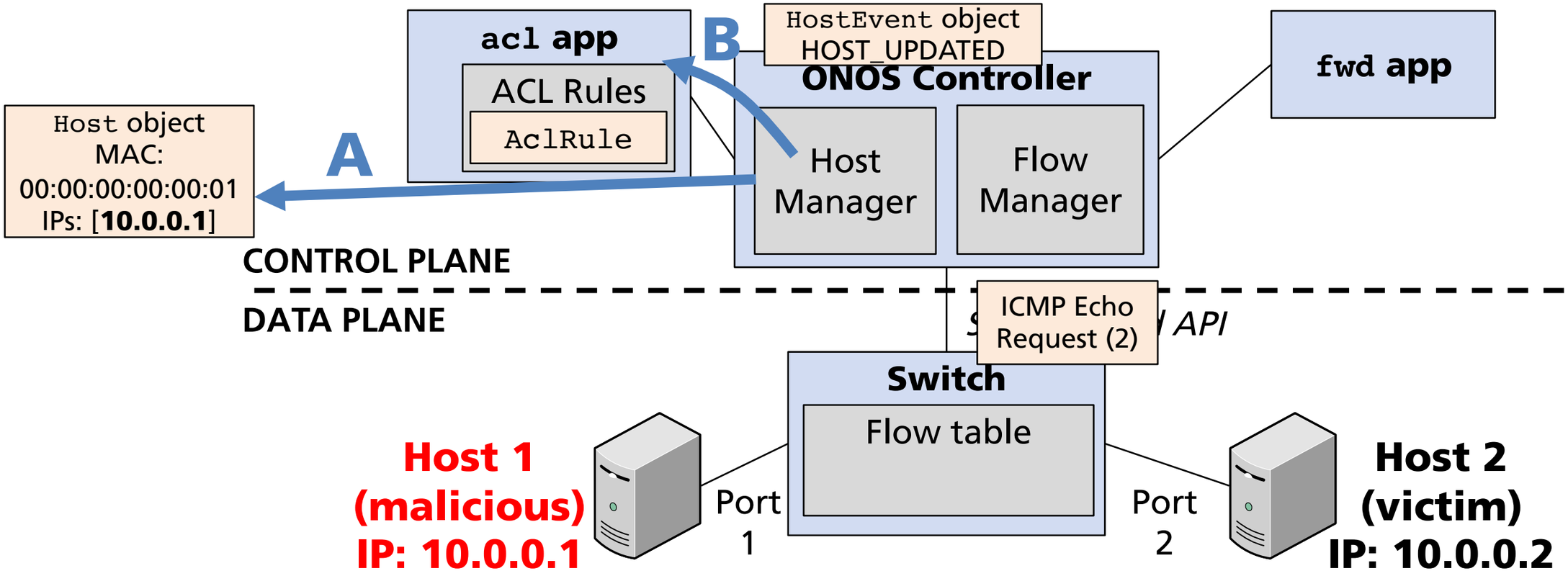
Anatomy of an Exploit

6. ONOS registers host 1's new IP address (A)



Anatomy of an Exploit

6. ONOS registers host 1's new IP address (A) as a `HOST_UPDATED` event (B). `acl` does not handle `HOST_UPDATED` events, so it does nothing.



Anatomy of an Exploit

6. ONOS registers host 1's new IP address (A) as a `HOST_UPDATED` event (B). `acl` does not handle `HOST_UPDATED` events, so it does nothing.

`AclManager.java`
handling new host events

```
@Override
public void event(HostEvent event) {
    // If host moves and violates an existing rule denies
    // its traffic, a new ACL flow rule is generated.
    if (event.type() == HostEvent.Type.HOST_ADDED) {
        DeviceId deviceId = event.subject().location().device
        if (mastershipService.getLocalRole(deviceId) == Maste
            for (AclRule rule : aclStore.getAclRules()) {
                if (rule.action() != AclRule.Action.ALLOW) {
                    processHostAddedEvent(event, rule);
                }
            }
        }
    }
}
```

`HOST_UPDATED` events not handled

Never gets called

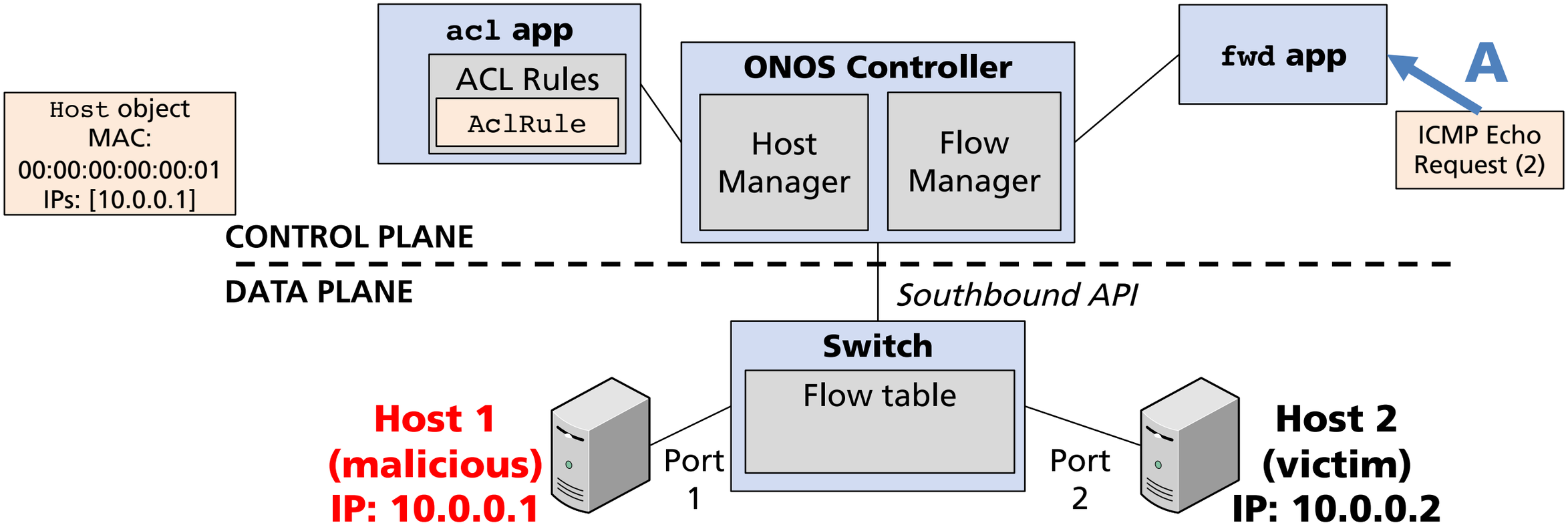
`AclManager.java` processing host events

```
private void processHostAddedEvent(HostEvent event, AclRule rule) {
    DeviceId deviceId = event.subject().location().deviceId();
    for (IpAddress address : event.subject().ipAddresses()) {
        if ((rule.srcIp() != null) ?
                (checkIpInCidr(address.getIp4Address(), rule.srcIp()
                (checkIpInCidr(address.getIp4Address(), rule.dstIp()
            if (!aclStore.checkIfRuleWorksInDevice(rule.id(), device
                List<RuleId> allowingRuleList = aclStore
                        .getAllowingRuleByDenyingRule(rule.id());
                if (allowingRuleList != null) {
                    for (RuleId allowingRuleId : allowingRuleList) {
                        generateAclFlow(aclStore.getAclRule(allowing
                    }
                }
                generateAclFlow(rule, deviceId);
            }
        }
    }
}
```

Never gets called

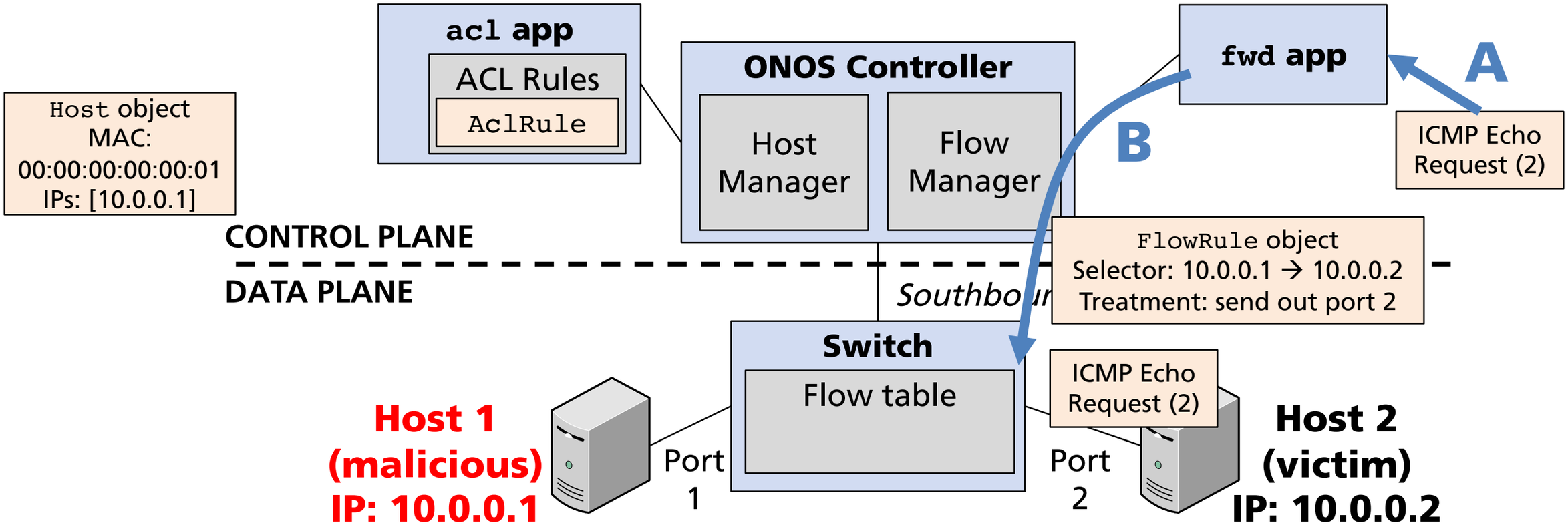
Anatomy of an Exploit

7. The packet gets sent to a second app (A)



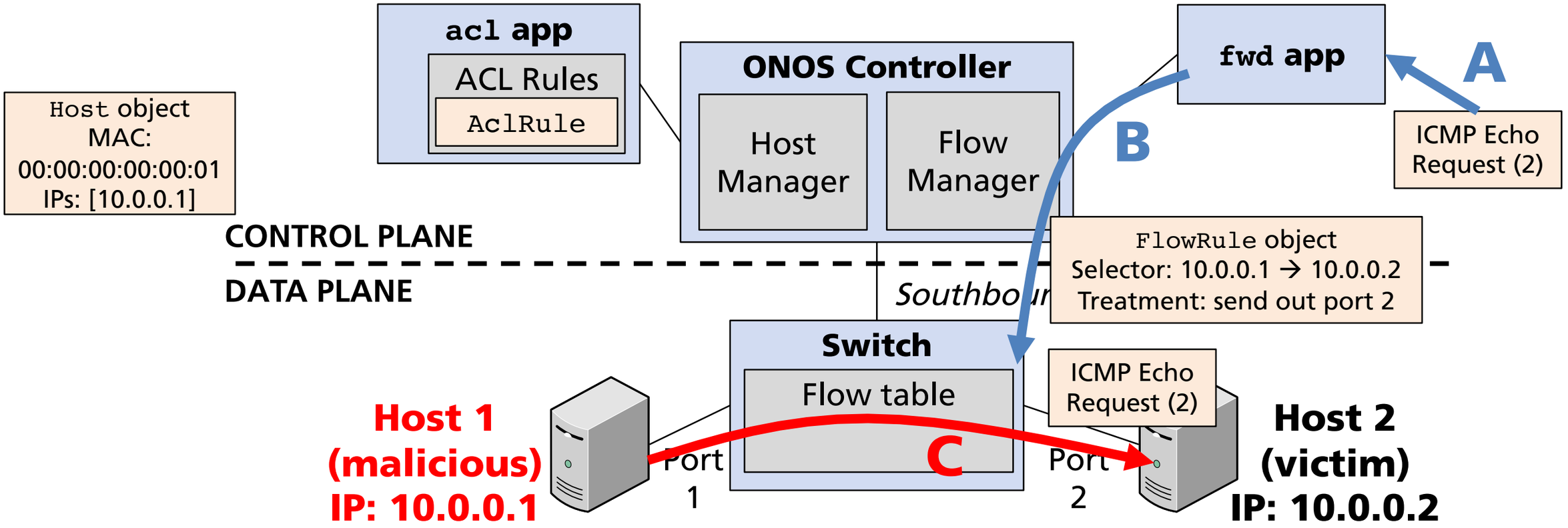
Anatomy of an Exploit

7. The packet gets sent to a second app (A), which instantiates the flow (allow) rule (B)



Anatomy of an Exploit

7. The packet gets sent to a second app (A), which instantiates the flow (allow) rule (B) and allows host 1 to communicate with host 2 (C).



What Makes This Challenging?

No ground truth about what events ought to be handled

Multiple entry points for code analysis

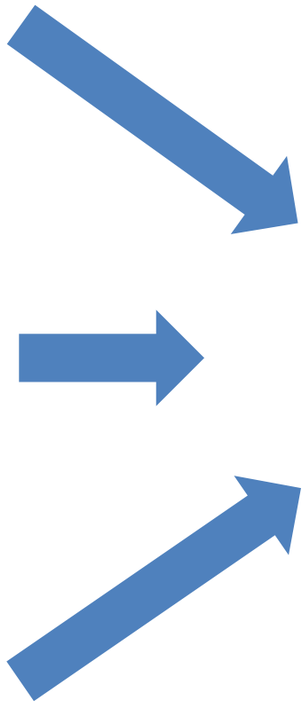
Not all event handling can affect the data plane

What Makes This Challenging?

No ground truth about what events ought to be handled

Multiple entry points for code analysis

Not all event handling can affect the data plane



EventScope

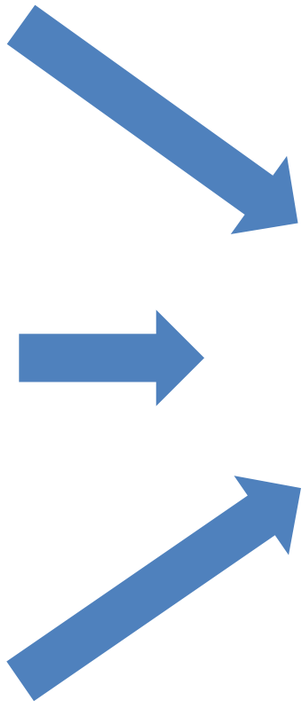
Automated Discovery of Cross-Plane Event-Based Vulnerabilities in SDN

What Makes This Challenging?

No ground truth about what events ought to be handled

Multiple entry points for code analysis

Not all event handling can affect the data plane



EVENTSCOPE

Automated Discovery of Cross-Plane Event-Based Vulnerabilities in SDN

Found 14 new vulnerabilities

EventScope Solution

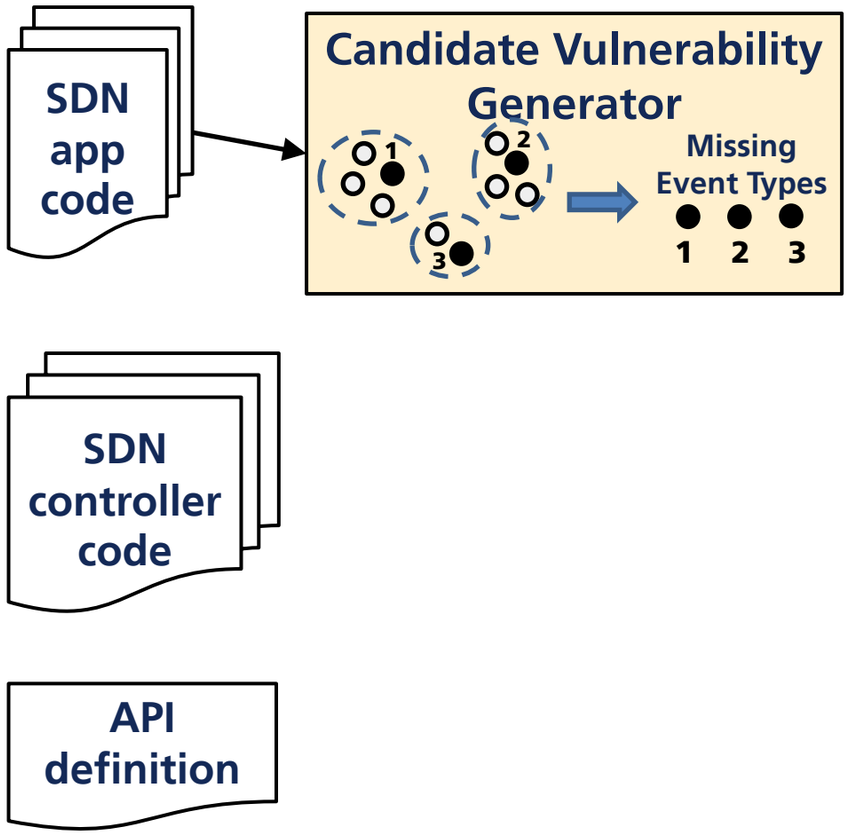
No ground truth about what events ought to be handled

Cluster apps according to similar functionality

Multiple entry points for code analysis

Not all event handling can affect the data plane

EVENTSCOPE App Event Use



Host event kind
- HOST_ADDED
- HOST_REMOVED
- HOST_UPDATED
- HOST_MOVED

Event types of **Host** event kind

Event types of **Link** event kind

Link event kind
- LINK_ADDED
- LINK_REMOVED
- LINK_UPDATED

Category 1

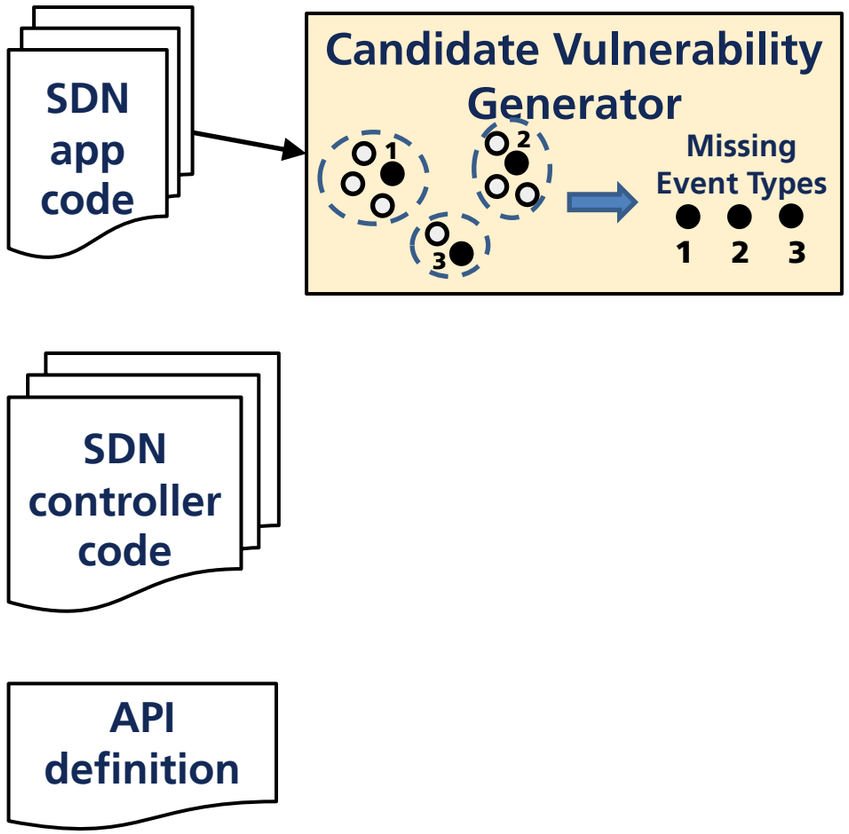
L2 forwarding app

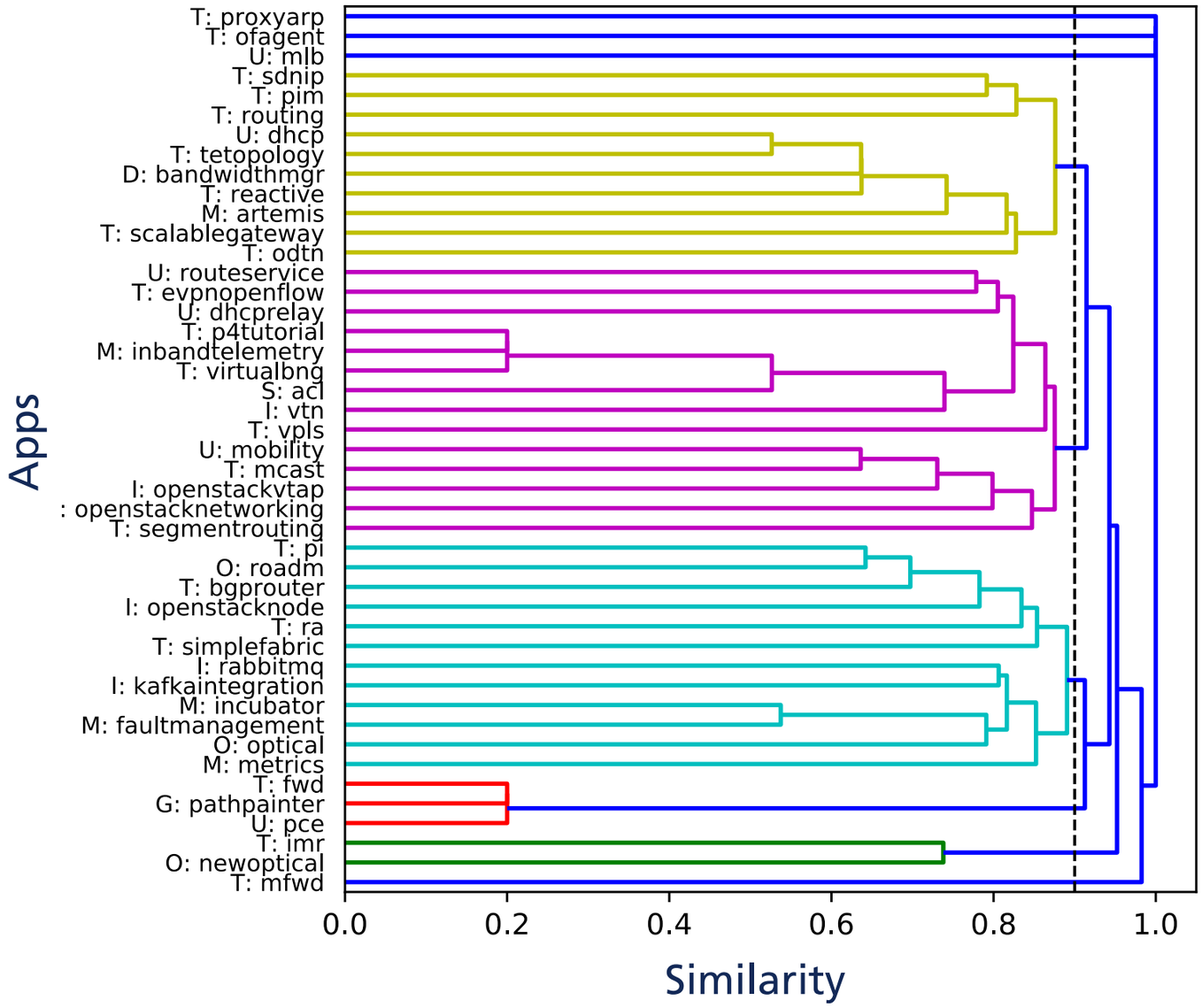
L3 routing app

Category 2

Firewall app

EventScope App Event Use





EventScope Solution

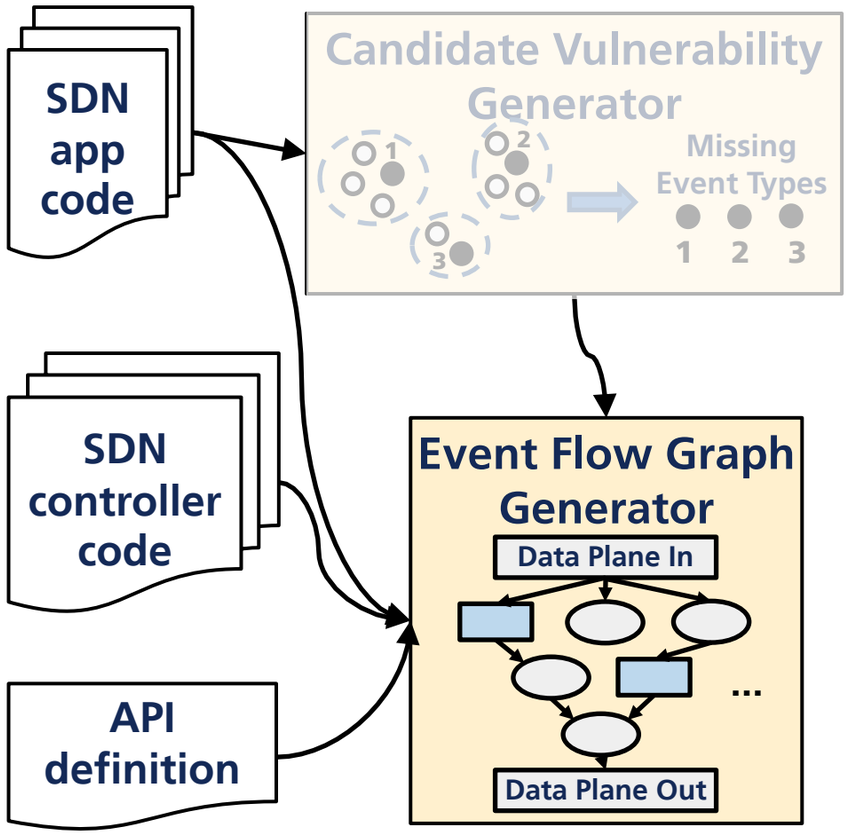
No ground truth about what events ought to be handled

Multiple entry points for code analysis

Abstract event flow with graphical model

Not all event handling can affect the data plane

EVENTSCOPE Event Flow Graph



Component 1 **Packet** event listener

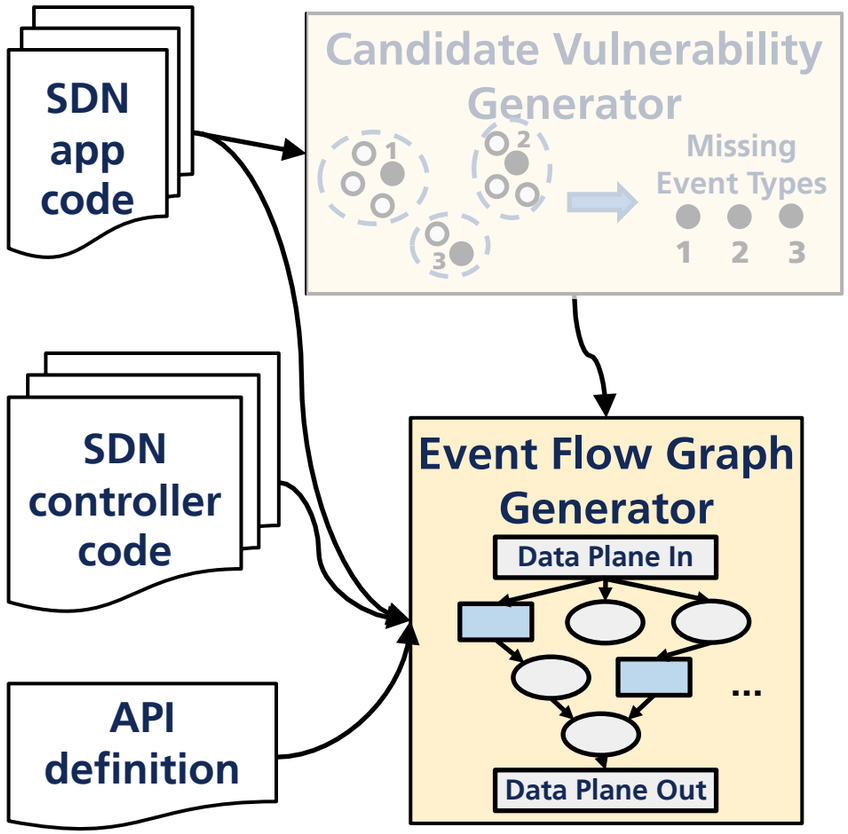
App 1 **Host** event listener

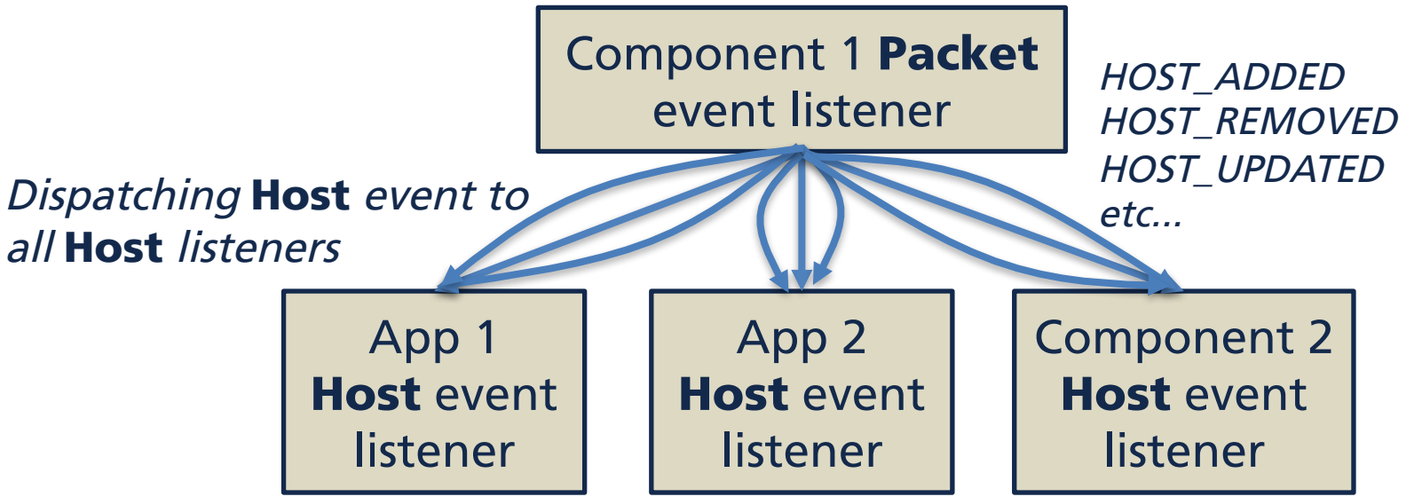
App 2 **Host** event listener

Component 2 **Host** event listener

Use event listeners of components and apps as **entry points**

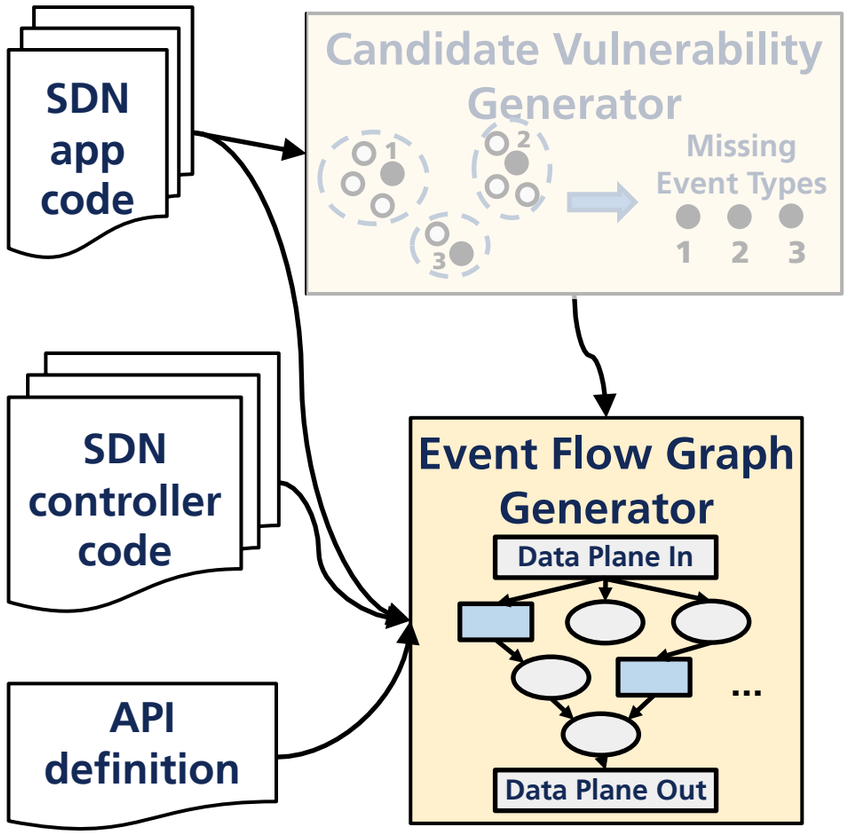
EVENTSCOPE Event Flow Graph

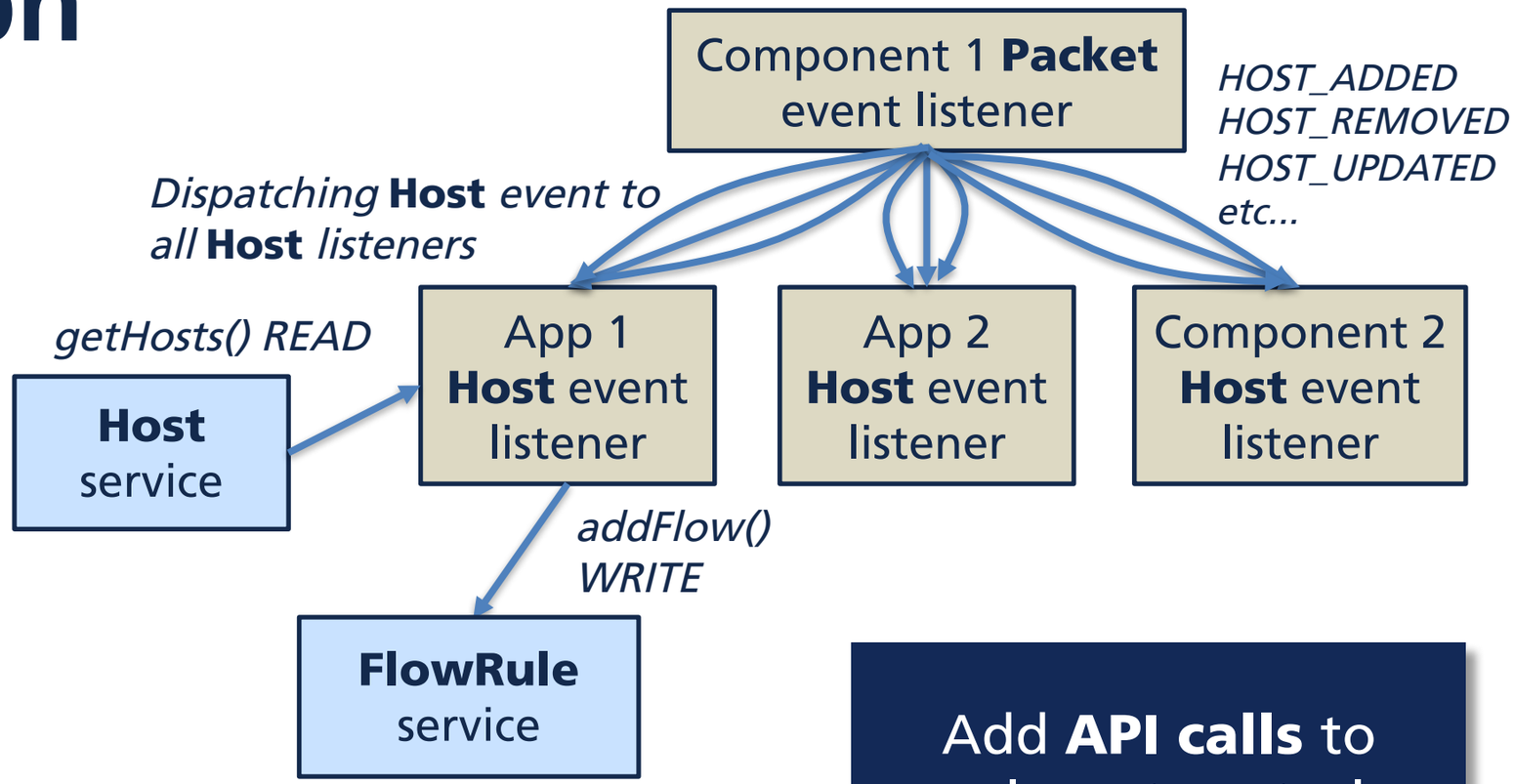




Link **event dispatchers** and **event listeners**

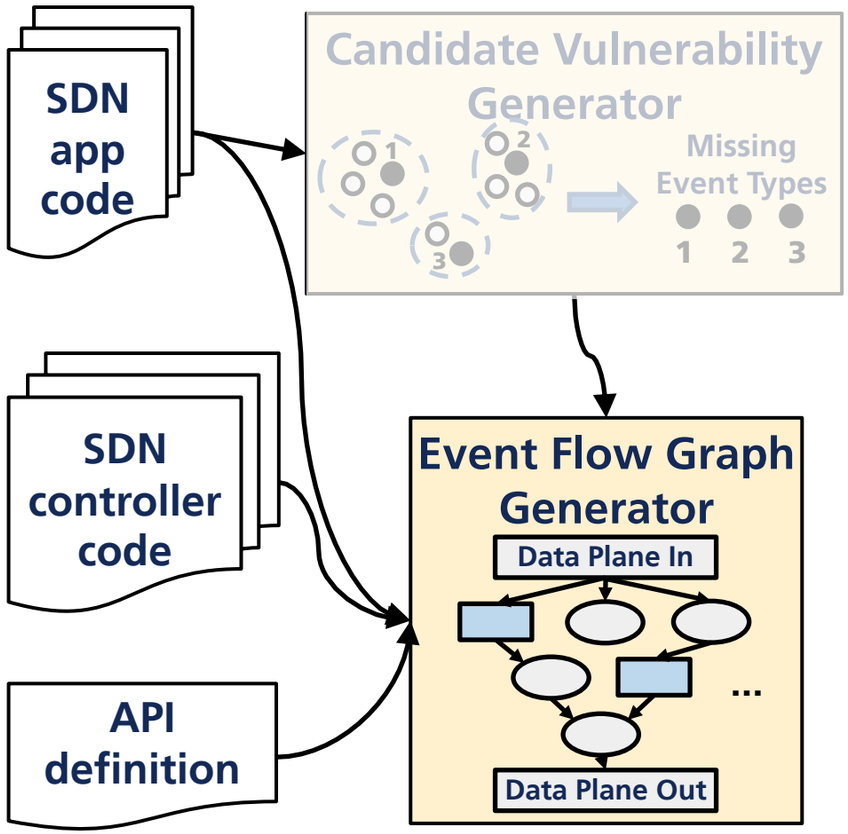
EventScope Event Flow Graph

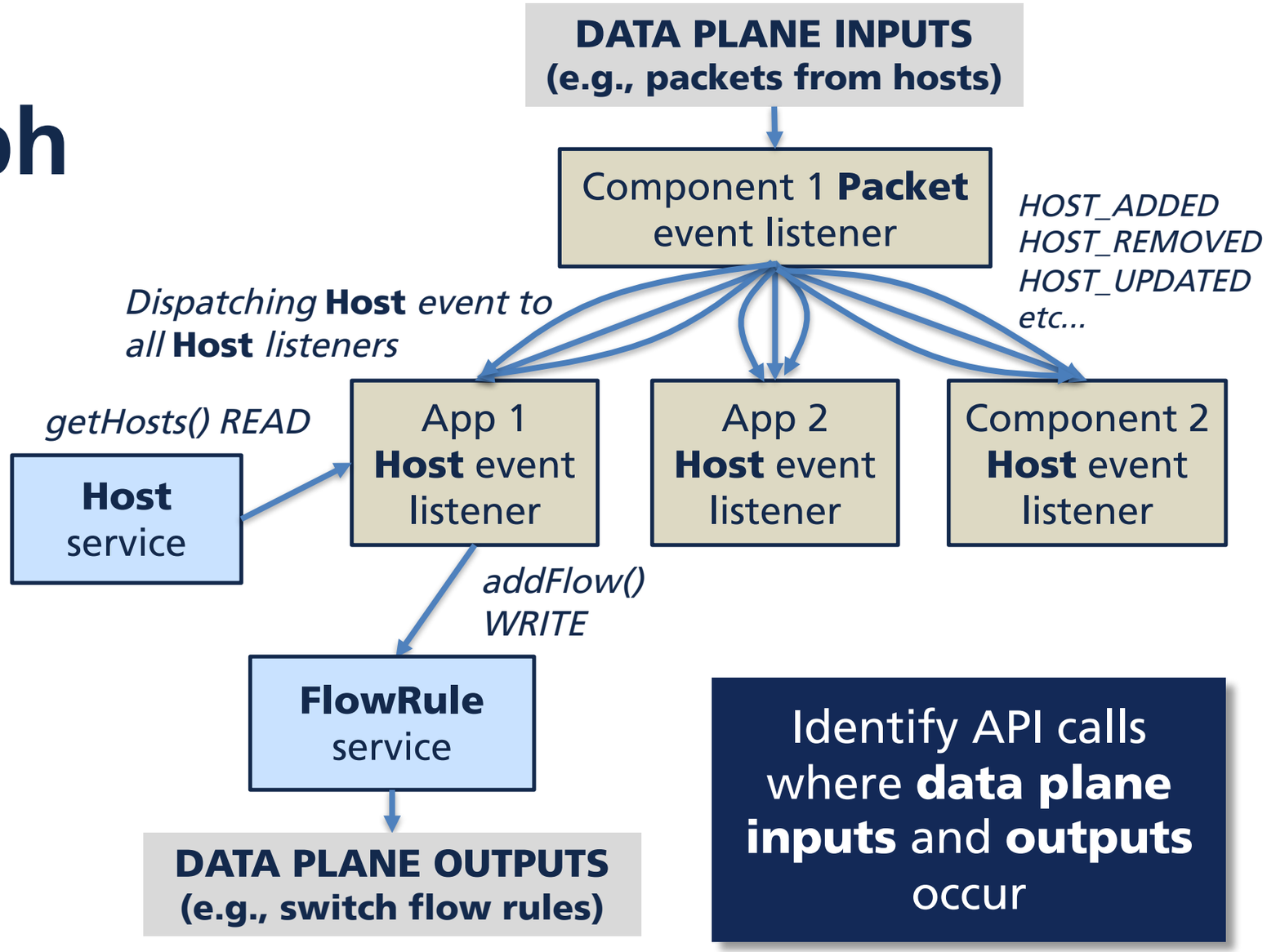




Add **API calls** to relevant control plane objects

EVENTSCOPE Event Flow Graph





Identify API calls where **data plane inputs** and **outputs** occur

EventScope Solution

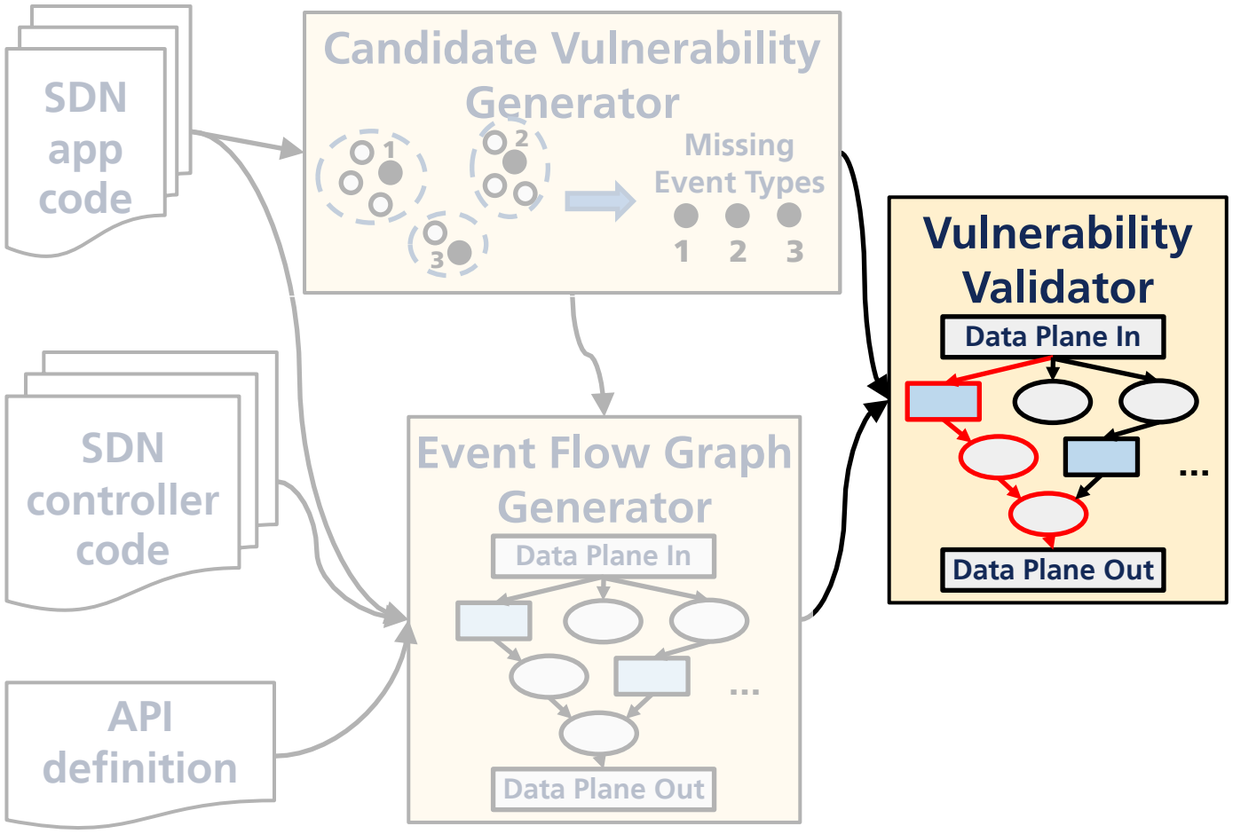
No ground truth about what events ought to be handled

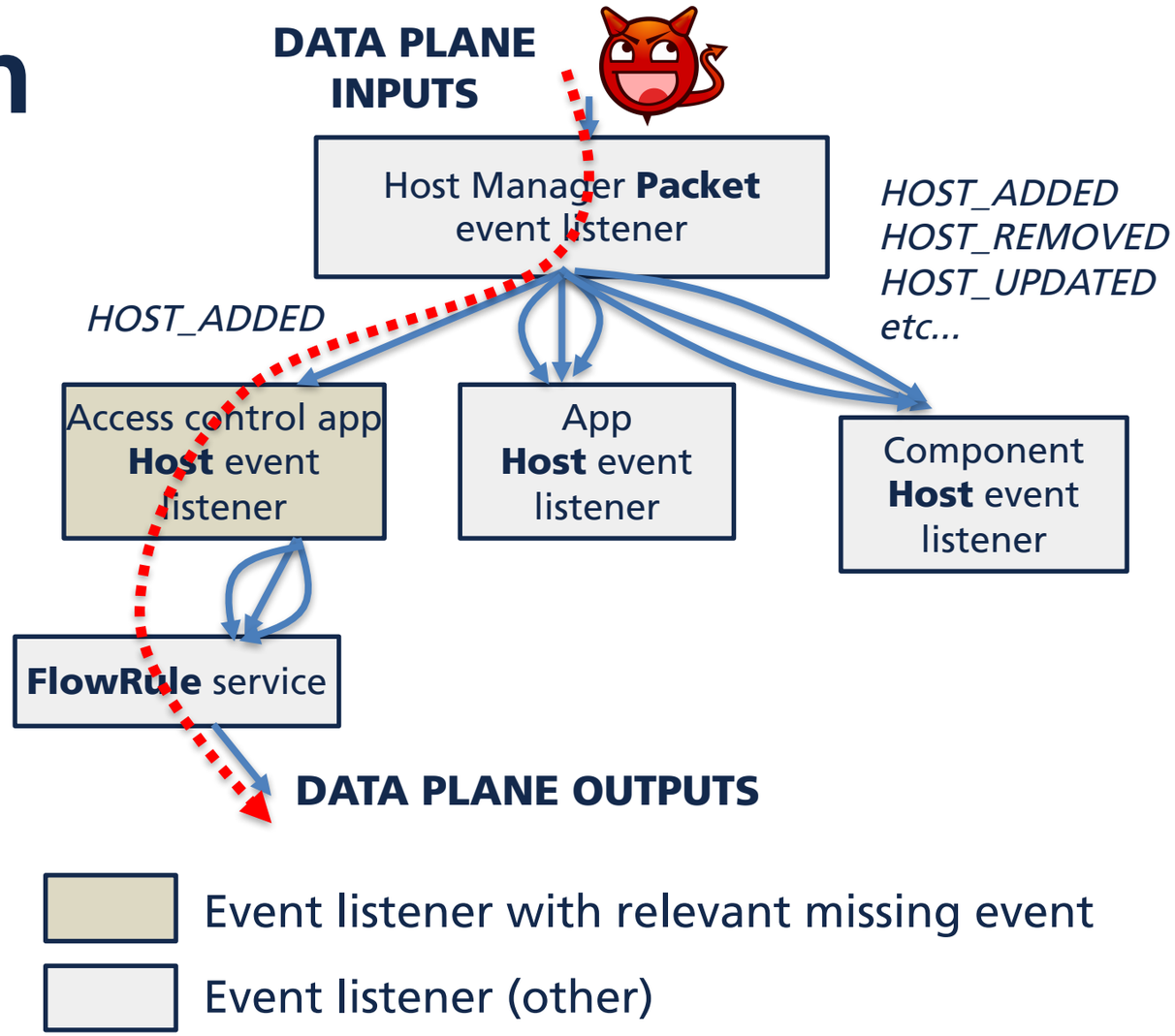
Multiple entry points for code analysis

Not all event handling can affect the data plane

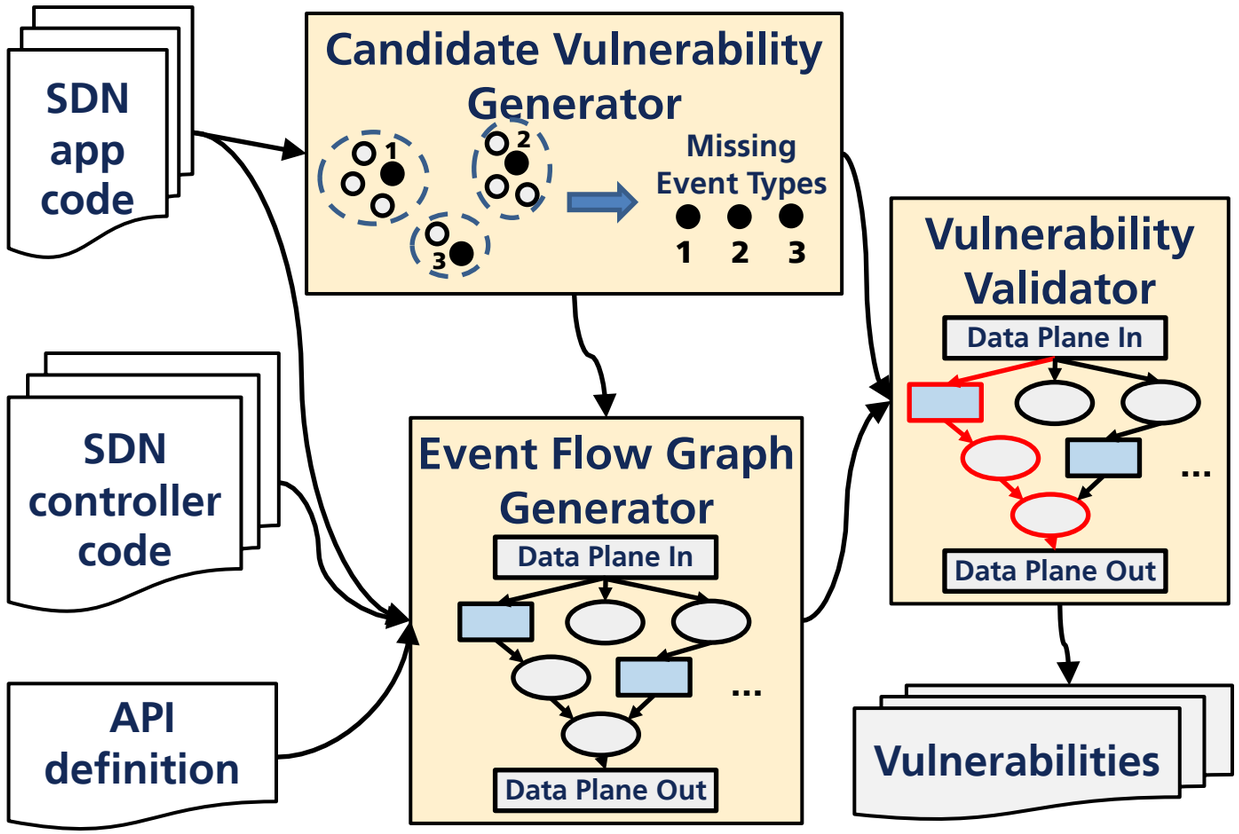
Trace viable control paths in event flow graph

EVENTSCOPE Vulnerability Validation





EventScope Evaluation

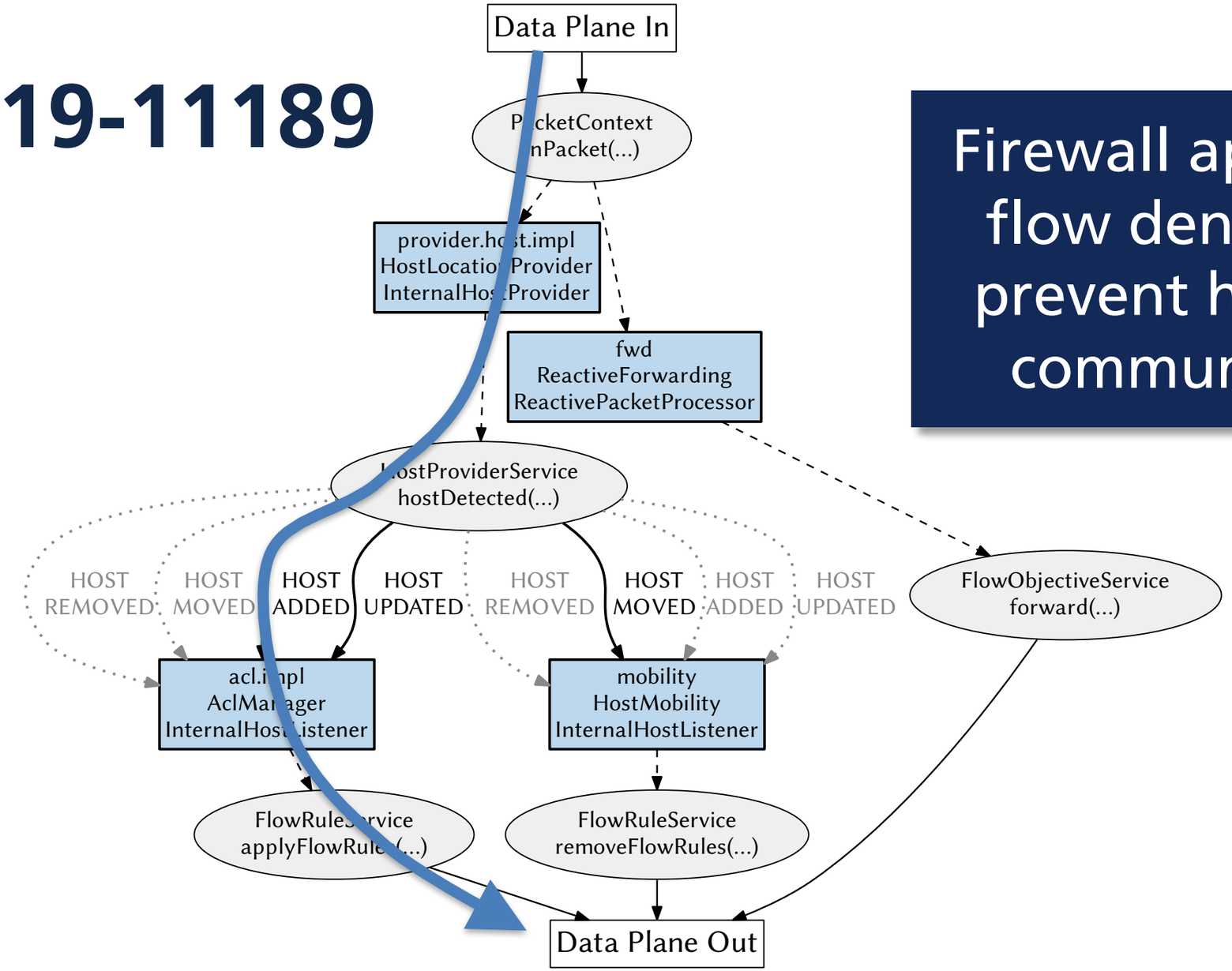


Reported 14 vulnerabilities to ONOS Security Team and requested CVE identifiers

TABLE X: Event Li...

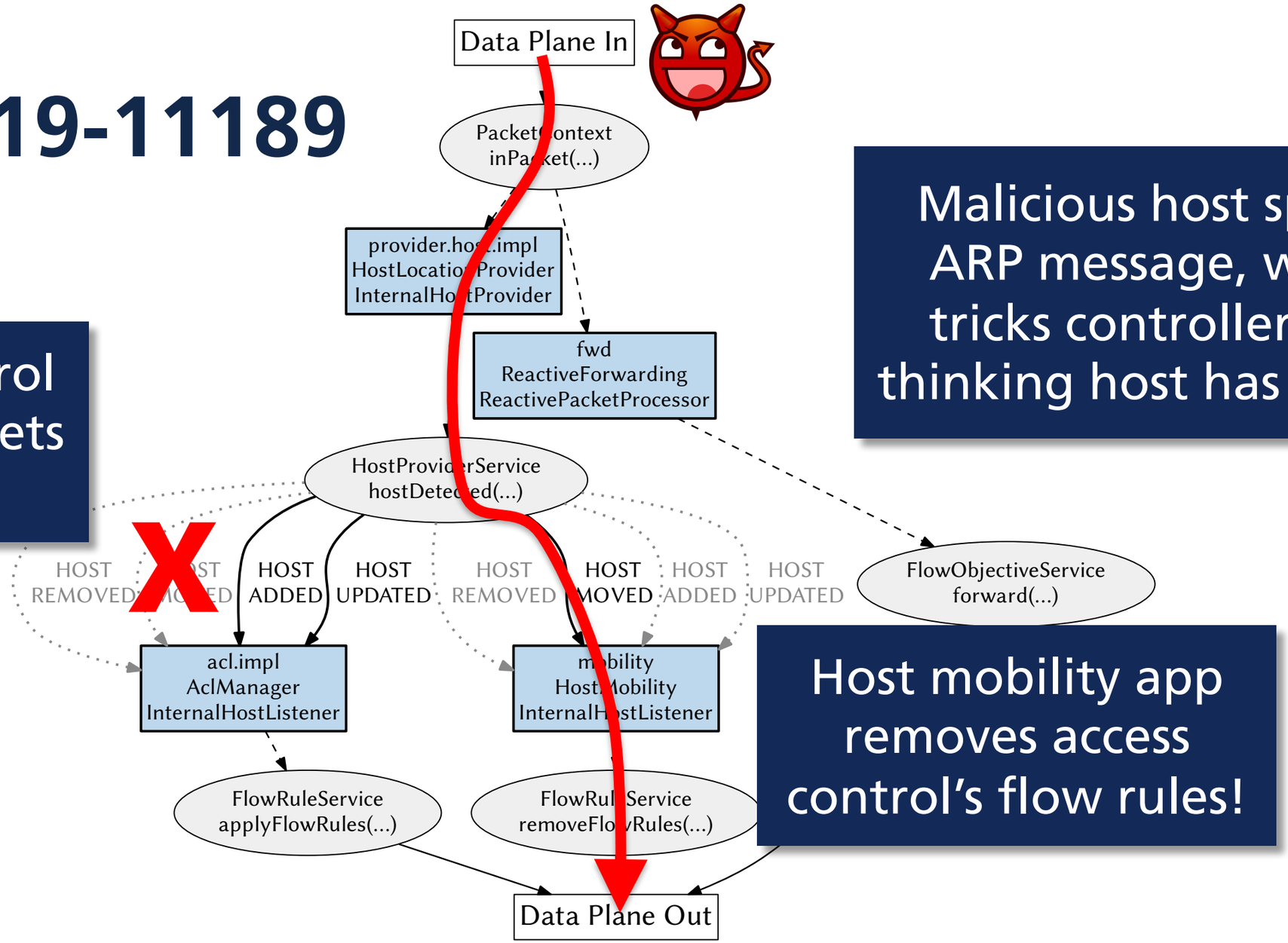
#	ID		Unhandled type	Example event flow graph path showing potential data plane input to data plane effect
1	CVE-201...6...	...l	HOST_UPDATED	See Figures 8 and 9 for event flow graph examples.
	...19-11189	ac...	HOST_MOVED	
2	CVE-2019-16300	acl	HOST_REMOVED	
3	CVE-2019-16298	virtualbng	HOST_MOVED	DPIn —DP_IN→ inPacket() —API_READ→ provider.host.InternalHostProvider —API_WRITE→ hostDetected() —HOST_ADDED→ virtualbng.InternalHostListener —API_WRITE→ startMonitoringIp() —DP_OUT→ DPOut
4	CVE-2019-16298	virtualbng	HOST_REMOVED	
5	CVE-2019-16298	virtualbng	HOST_UPDATED	
6	CVE-2019-16299	mobility	HOST_ADDED	DPIn —DP_IN→ inPacket() —API_READ→ provider.host.InternalHostProvider —API_WRITE→ hostDetected() —HOST_MOVED→ mobility.InternalHostListener —API_WRITE→ removeFlowRules() —DP_OUT→ DPOut
7	CVE-2019-16299	mobility	HOST_REMOVED	
8	CVE-2019-16299	mobility	HOST_UPDATED	
9	CVE-2019-16301	vtn	HOST_MOVED	DPIn —DP_IN→ inPacket() —API_READ→ provider.host.InternalHostProvider —API_WRITE→ hostDetected() —HOST_ADDED, HOST_UPDATED, or HOST_REMOVED→ vtn.InternalHostListener —API_WRITE→ forward() —DP_OUT→ DPOut
10	CVE-2019-16302	evpnopenflow	HOST_MOVED	DPIn —DP_IN→ inPacket() —API_READ→ provider.host.InternalHostProvider —API_WRITE→ hostDetected() —HOST_ADDED or HOST_REMOVED→ evpnopenflow.InternalHostListener —API_WRITE→ forward() —DP_OUT→ DPOut
11	CVE-2019-16302	evpnopenflow	HOST_UPDATED	
12	CVE-2019-16297	p4tutorial	HOST_MOVED	DPIn —DP_IN→ inPacket() —API_READ→ provider.host.InternalHostProvider —API_WRITE→ hostDetected() —HOST_ADDED→ p4tutorial.InternalHostListener —API_WRITE→ applyFlowRules() —DP_OUT→ DPOut
13	CVE-2019-16297	p4tutorial	HOST_REMOVED	
14	CVE-2019-16297	p4tutorial	HOST_UPDATED	

CVE-2019-11189



Firewall app installs flow deny rule to prevent host from communicating

CVE-2019-11189

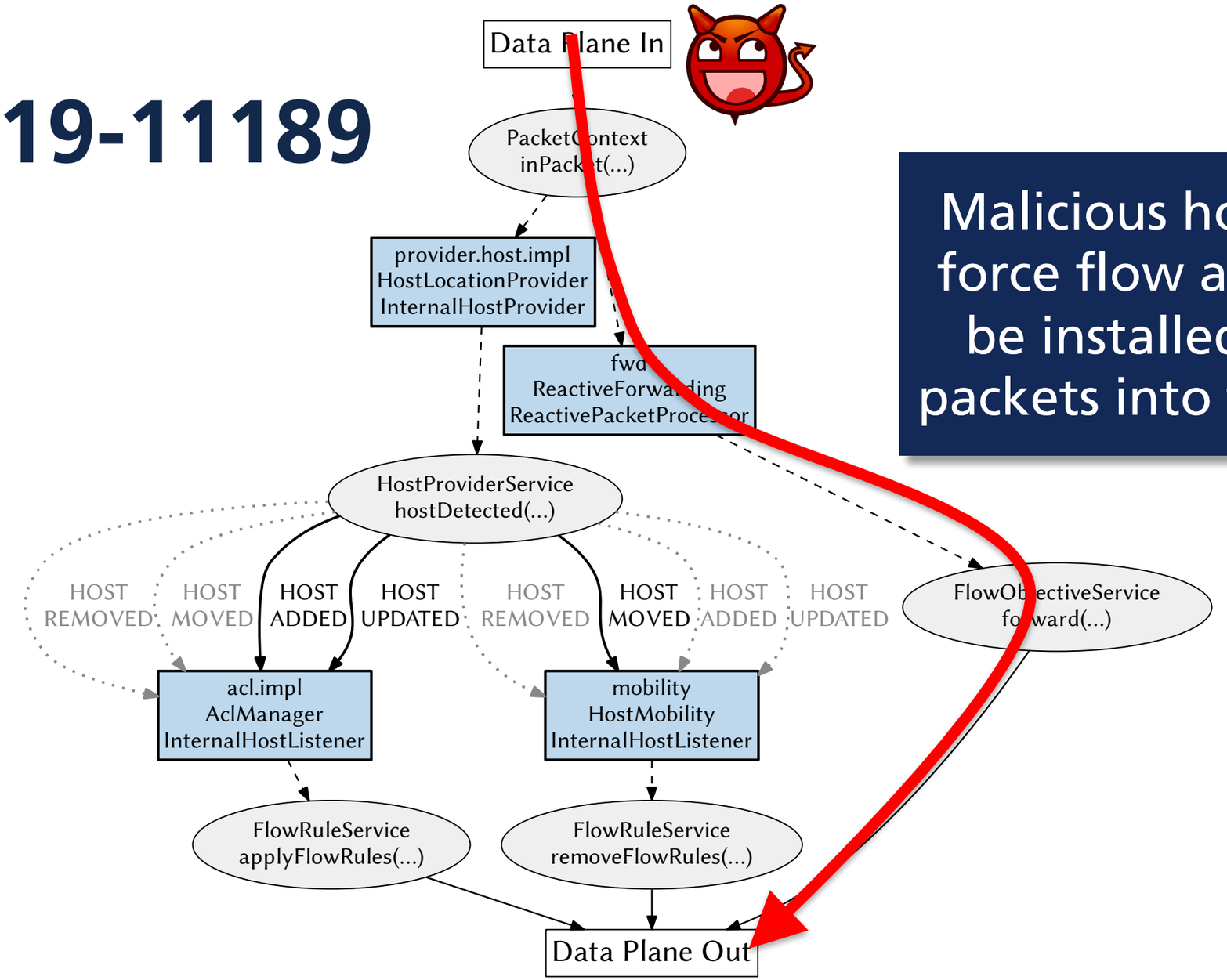


Access control app never gets called!

Malicious host spoofs ARP message, which tricks controller into thinking host has moved

Host mobility app removes access control's flow rules!

CVE-2019-11189



Malicious host can now force flow allow rule to be installed and send packets into the network

Conclusions

- Considered the **cross-plane event-based vulnerability** problem in SDN
- Design takeaways
  - **Hosts** have **outsized effect** on SDN operation
  - Security analysis must consider **all apps working together**
  - Developers must **design defensively**
- Discovered and validated **14 new vulnerabilities** in ONOS SDN controller

Questions?

Thank you for your time!

Benjamin E. Ujcich
E-mail: ujcich2@illinois.edu
Web: http://ujcich2.web.engr.illinois.edu

This work was supported in part by NSF Grant Nos. CNS-1657534 and CNS-1750024. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation.