

USENIX

THE ADVANCED COMPUTING SYSTEMS ASSOCIATION

When Address Learning Goes Wrong: Inducing Forwarding Loops and DoS Amplification in SDN

Dezhang Kong, *Zhejiang University;* Yilun Zhang, *Purdue University;* Zekun Xie, *Xi'an University of Posts and Telecommunications;* Ningpeng Zheng, *Yancheng Institute of Technology;* Shi Lin, *Zhejiang Gongshang University;* Zhenhua Xu, *Zhejiang University;* Minghao Li, *Chongqing University;* Zhebo Wang, Xiang Chen, and Changting Lin, *Zhejiang University;* Dong Zhang, *Fuzhou University;* Xuan Liu, *Yangzhou University;* Chunming Wu and Meng Han, *Zhejiang University*

https://www.usenix.org/conference/usenixsecurity26/presentation/kong

This artifact appendix is included in the Artifact Appendices to the Proceedings of the 35th USENIX Security Symposium and appends to the paper of the same name that appears in the Proceedings of the 35th USENIX Security Symposium.

August 12–14, 2026 • Baltimore, MD, USA

ISBN 978-1-939133-58-8

Open access to the Artifact Appendices to the Proceedings of the 35th USENIX Security Symposium is sponsored by



جامعة الملك عبدالله للعلوم والتقنية

King Abdullah University of Science and Technology

USENIX Security'26 Artifact Appendix: When Address Learning Goes Wrong: Inducing Forwarding Loops and DoS Amplification in SDN

Dezhang Kong[1], Yilun Zhang[2], Zekun Xie[3], Ningpeng Zheng[4], Shi Lin[5], Zhenhua Xu[1], Minghao Li[6]
Zhebo Wang[1], Xiang Chen[1], Changting Lin[1], Dong Zhang[7], Xuan Liu[8], Chunming Wu[1], Meng Han[1]
[1]Zhejiang University, [2]Purdue University, [3]Xi'an University of Posts & Telecommunications
[4]Yancheng Institute of Technology, [5]Zhejiang Gongshang University, [6]Chongqing University
[7]Fuzhou University, [8]Yangzhou University

A Artifact Appendix

A.1 Abstract

Our artifact contains the **source code, packet traces, raw data, and plotting material** used for the paper "When Address Learning Goes Wrong: Inducing Forwarding Loops and DoS Amplification in SDN." This appendix provides a walkthrough of the artifact, including the installation, execution, and expected results for each experiment. Note that for reviewers' convenience, we provide a **prebuilt VM image** that contains all dependencies and the artifact itself, so that the reviewers can run the experiments without manual installation. This artifact's main function is to instantiate the LoopGen attack and reproduce its (1) feasibility, (2) amplification effect, (3) cost, (4) stealthiness, and (5) defense evaluations.

A.2 Description & Requirements

A.2.1 Security, privacy, and ethical concerns

The artifact intentionally sends spoofed packets, induces forwarding loops, and creates congestion inside emulated SDN environments. Several workflows require `sudo` privileges. The artifact does not require human subjects and does not collect personal or sensitive user data.

A.2.2 How to access

The artifact is archived at https://doi.org/10.5281/zenodo.20307092. The link of our VM image is also in its `./README.md`.

A.2.3 Hardware dependencies

Please use Windows system to run the VM. We validated the artifact on a standard x86_64 Linux VM with at least 8 CPU cores, 16 GB RAM, and roughly 40 GB of free disk space. The VM runs on VirtualBox. We think a powerful laptop or PC is necessary to run the VirtualBox VM smoothly. Note that we also used Barefoot Tofino switch in a minor case study, but it is not required for the main evaluation workflow. We are applying for external access to the Tofino switch, and we will provide the details for reviewers if the access is granted.

A.2.4 Software dependencies

In the released VM image, all dependencies are pre-installed.

A.2.5 Benchmarks

We only use one public traffic dataset `202201011400.pcap` from the MAWI Working Group Traffic Archive (https://mawi.wide.ad.jp/mawi/samplepoint-F/2022/202201011400.html)

A.3 Set-up

A.3.1 Installation

1. Download and unpack the artifact.
2. Open the `Readme.md` in the root directory, install VirtualBox and download our VM image.
3. Use VirtualBox to boot the VM image, which already contains all dependencies and the artifact itself.
4. Choose the claims in Figure 1 and read the corresponding directory in the artifact. The `readme.md` in each directory contains the instructions[1].

A.3.2 Basic Test

Please see **E1.1** in Section A.4.2.

[1]We highly recommend readers to read the `Readme.md` files in the artifact, because they contain screenshots for step-by-step instructions while this appendix do not contain them due to space limitation.

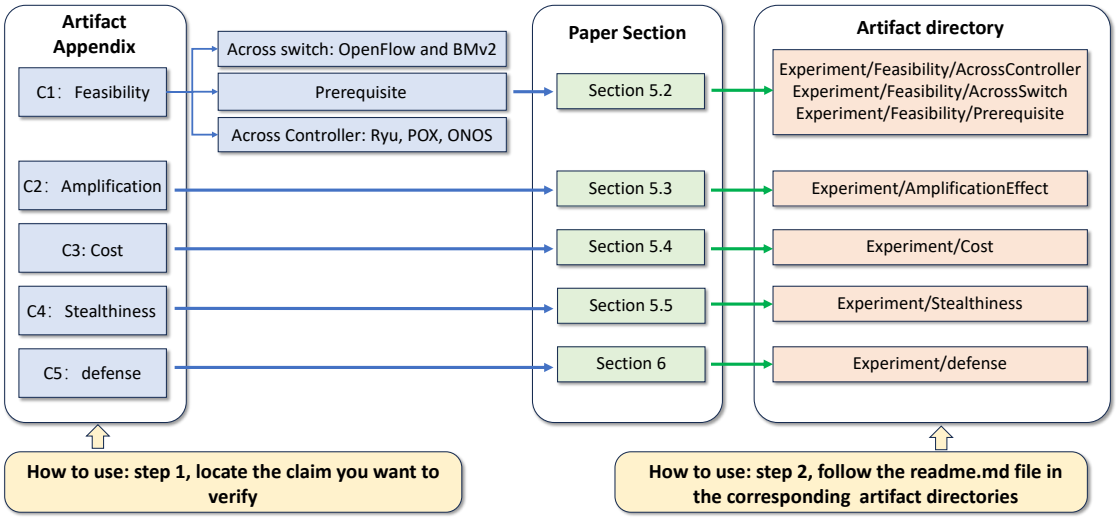

Figure 1: Organization of the artifact and its relation to the paper.

A.4 Evaluation workflow

A.4.1 Major Claims

Our major claims are summarized in Figure 1 and the following list. Each claim is verified by one or more experiments in the artifact.

(C1) *Feasibility: (1) its prerequisite in real-world topologies, (2) its feasibility on different controllers (ryu, pox, and onos), and (3) its feasibility on two types of switches (OpenFlow and programmable switch, i.e., BMv2).*

(C2) *Amplification Effect: LoopGen can significantly amplify traffic and degrade network performance (RTT and Bandwidth).*

(C3) *Low Cost: LoopGen achieves comparable or stronger impact than a non-looping DoS attack while requiring a much lower sending rate.*

(C4) *Stealthiness: LoopGen are not detected by existing detection mechanisms.*

(C5) *Our two defenses can detect LoopGen.*

A.4.2 Experiments

(E1.1) Prerequisite feasibility [5 human-minutes + 10 compute-minutes]. Verify Claim (C1). We recommend to read the `./Experiment/Feasibility/Prerequisite/readme.md` in the artifact.

Preparation. Please see Section A.3.1.

Execution. Open a terminal (entering the `p4` directory by default) and run

```
cd Prerequisite
```

For the probability that hosts form a loop structure, run

```
python3 Phost.py
```

For the probability in Leaf-Spine and Fat-Tree, run

```
python3 LSandFT.py
```

Results. The scripts print the probability of loop-feasible host placements in the evaluated topologies. These results are identical to the `./Experiment/Feasibility/Prerequisite/RawData&Figures/loopprobability.ipynb`. The `.ipynb` file can be opened in Visual Studio Code.

(E1.2) Feasibility on Ryu [5 human-minutes + 5 compute-minutes]. Verify Claim (C1). We recommend to read the `./Experiment/Feasibility/AcrossController/Ryu/readme.md` in the artifact.

Preparation. Please see Section A.3.1.

Execution. 1. Open a terminal (entering the `p4` directory by default) 2. Start the controller

```
ryu-manager ryu/ryu/app/simple_switch_13.py
```

3. Open a new terminal and start mininet topology

```
cd loopgenexp/ControllerFeasibility-ryu/
sudo python3 -E LoopGen_topo.py
```

4. Open the host terminals in mininet

```
mininet> xterm h1 h2 h3
```

5. Send packets in each host terminal. Note: Don't hit Enter right away. Type the command on all three first, then press Enter on all of them at once.

```
h1> python3 hack_h1_auto.py
h2> python3 hack_h2_auto.py
h3> python3 hack_h3_auto.py
```

Please **wait** until all hosts finshed the attack.

6. Open a new terminal and start wireshark. Then, monitor the **`s1-eth2`** port.

```
sudo wireshark
```

Results. Monitor `s1-eth2` with Wireshark. The expected outcome is a persistent stream of looping ARP packets.

(E1.3) Feasibility on POX [5 human-minutes + 5 compute-minutes]. Verify Claim (C1). We recommend to read the `./Experiment/Feasibility/AcrossController/POX/l2/README.md` in the artifact.

Preparation. Please see Section A.3.1.

Execution. 1. Open a terminal (entering the `p4` directory by default) 2. Start the controller

```
cd pox
./pox.py log.level -DEBUG openflow.of_01 -port=6633
forwarding.l2_learning
```

3. Open a new terminal and start Mininet topology

```
cd loopgenexp/ControllerFeasibility-pox/l2/
sudo python3 -E LoopGen_topo.py
```

4. Open the host terminals in Mininet

```
mininet> xterm h1 h2 h3
```

5. Send packets in each host terminal. Note: Don't hit Enter right away. Type the command on all three first, then press Enter on all of them at once.
In h1 xterm, run

```
python3 hack_h1_auto.py
```

In h2 xterm, run

```
python3 hack_h2_auto.py
```

In h3 xterm, run

```
python3 hack_h3_auto.py
```

Please **wait** until all hosts finish the attack.
6. Open a new terminal and start Wireshark. Then, monitor the **`s1-eth2`** port.

```
sudo wireshark
```

7. For convenience, filter only ARP packets. You can see that the attack packets are looped continuously. **Results.** Monitor `s1-eth2` with Wireshark. The expected outcome is a persistent stream of looping ARP packets.

(E1.4) Feasibility on ONOS [10 human-minutes + 10 compute-minutes]. Verify Claim (C1). We recommend to read the `./Experiment/Feasibility/AcrossController/ONOS/README.md` in the artifact.

Preparation. Please see Section A.3.1.

Execution. 1. Open a terminal (entering the `p4` directory by default) 2. Enter the `onos` directory and start ONOS

```
cd onos
bazel run onos-local
```

3. Open a new terminal, enter the `onos` directory, and connect to the ONOS CLI

```
cd onos
./tools/test/bin/onos localhost
```

4. Activate the required ONOS apps

```
app activate org.onosproject.openflow
app activate org.onosproject.drivers
app activate org.onosproject.fwd
wipe-out please; app deactivate
org.onosproject.fwd; app activate
org.onosproject.fwd
```

5. Open a new terminal, start the Mininet topology, and open the host terminals

```
cd loopgenexp/ControllerFeasibility-onos/
sudo python3 -E LoopGen_topo.py
xterm h1 h2 h3
```

6. Launch the attack in each host terminal. Note: Don't hit Enter right away. Type the command on all three first, then press Enter on all of them at once.
In h1 xterm, run

```
python3 attack_l2_timed.py h1-eth0
00:00:00:00:00:02
```

In h2 xterm, run

```
python3 attack_l2_timed.py h2-eth0
00:00:00:00:00:03
```

In h3 xterm, run

```
python3 attack_l2_timed.py h3-eth0
00:00:00:00:00:01
```

7. Then, in the Mininet terminal:

```
mininet> h1 python3 -c "from scapy.all import sendp, Ether,
IP, UDP;
sendp(Ether(src='00:00:00:00:00:01',
dst='00:00:00:00:00:08') /
IP(dst='10.0.0.8') / UDP(dport=9999), iface='h1-eth0',
count=1)"
```

8. Open a new terminal and start Wireshark

```
sudo wireshark
```

Results. Monitor the corresponding switch interface with Wireshark. The expected outcome is that the injected packet destined to `10.0.0.8` keeps looping after the trigger packet is sent.

(E1.5) BMv2 feasibility [5 human-minutes + 5 compute-minutes]. Verify Claim (C1). We recommend to read the `./Experiment/Feasibility/AcrossSwitch/readme.md` in the artifact.

Preparation. Please see Section A.3.1.

Execution. 1. Open a terminal (entering the `p4` directory by default) 2. Enter the experiment directory and start the topology

```
cd tutorials/exercises/LoopGen/
make
```

If any compiling errors occur, try

```
make clean
make
```

3. Open another terminal and start the controller

```
cd tutorials/exercises/LoopGen/
sudo python3 controller.py
```

4. In the Mininet terminal, open the host terminals. Note: Don't hit Enter right away. Type the command on all three first, then press Enter on all of them at once.

```
xterm h1 h2 h3
```

In h1 xterm, run

```
python3 hack_h1_auto.py
```

In h2 xterm, run

```
python3 hack_h2_auto.py
```

In h3 xterm, run

```
python3 hack_h3_auto.py
```

Please **wait** until they finish the attack.
5. Open a new terminal and start Wireshark. Then, monitor the **`s1-eth2`** port.

```
sudo wireshark
```

Results. Monitor `s1-eth2` with Wireshark. The expected outcome is again a continuous loop of attack packets on the monitored interface. This confirms that the same attack logic works on the programmable software-switch setting used by the rest of the BMv2-based experiments. The sibling `AcrossSwitch-Tofino/` note documents an additional hardware case study, but it is not the base of any other experiments; the same forwarding behavior is already mirrored by this BMv2 experiment.

(E2.1) Evaluate amplification factor [10 human-minutes + 10 compute-minutes + up to 40 GB disk for packet captures]. Verify Claim (C2). We recommend to read the `./Experiment/Feasibility/AmplificationEffect/readme.md` in the artifact.

Preparation. Please see Section A.3.1.

Execution. 1. Open a terminal (entering the `p4` directory by default). 2. Enter the experiment directory and start the topology.

```
cd tutorials/exercises/AmplificationEffect/
make
```

3. Open Wireshark **before** sending any packets. Then, monitor the **`s1-eth2`** port.

```
sudo wireshark
```

You can see that there is no packet at this point. In the Mininet terminal, open `h1` and send the test packet.

```
mininet> xterm h1
h1> python3 send.py
```

`send.py` sends **one** packet whose TTL is `255`.

Results. Then, you can see that `85` packets are captured by Wireshark, which means that this packet was looped `85` times.

(E2.2) Evaluate RTT [10 human-minutes + 10 compute-minutes + up to 40 GB disk for packet captures]. Verify Claim (C2). We recommend to read the `./Experiment/Feasibility/AmplificationEffect/readme.md` in the artifact.

1. First, open an `h1` xterm and use `tcpreplay` to inject attack packets.

```
mininet> xterm h1
h1> tcpreplay -i h1-eth0 -p 300 -l 0 LoopGen.pcap
```

2. Then, open a new `h1` xterm and ping `h2`.

```
h1> ping 10.0.0.2
```

Results. You can see RTT increases significantly.

(E2.3) Evaluate throughput [10 human-minutes + 10 compute-minutes + up to 40 GB disk for packet captures]. Verify Claim (C2). We recommend to read the `./Experiment/Feasibility/AmplificationEffect/readme.md` in the artifact.

1. First, open an `h1` xterm and use `tcpreplay` to inject attack packets.

```
mininet> xterm h1
h1> tcpreplay -i h1-eth0 -p 300 -l 0 LoopGen.pcap
```

2. Then, open a new `h1` xterm and an `h2` xterm.

```
mininet> xterm h1 h2
```

3. In the `h2` xterm, start an `iperf` TCP server.

```
h2> iperf -s
```

4. In the new `h1` xterm (do **not** change the other `h1` xterm running `tcpreplay`), run

```
h1> iperf -c 10.0.0.2
```

5. Then, on the `h2` xterm, you can see the evaluated bandwidth (the maximum is `10` Mbps, so we can calculate the degradation).

Results. Throughput is `8.33` Mbps corresponds to a `14.7`% degradation from the `9.76` Mbps normal link rate.

If you want to change topologies, edit the `Makefile`. The default topology is `network-BAP.py`. The other topology scripts are included in the same directory.

(E3) Cost comparison against normal DoS [20 human-minutes + 10 compute-minutes]. Verify Claim (C3). We also recommend reviewers to read the `./Experiment/Cost/readme.md` in the artifact.

Preparation. Please see Section A.3.1.

Execution. 1. Open a terminal (entering the `p4` directory by default). 2. Enter the experiment directory and start the topology.

```
cd tutorials/exercises/Cost/
make
```

• **Launch LoopGen.**

1. First, open an `h1` xterm and use `tcpreplay` to inject attack packets.

```
mininet> xterm h1
h1> tcpreplay -i h1-eth0 -p 300 -l 0 LoopGen.pcap
```

2. Then, open a new h1 xterm and ping h2.

```
h1> ping 10.0.0.2
```

• **Launch normal DoS.**

1. First, open an h1 xterm and use tcpreplay to inject attack packets. Use normalDoS.pcap instead of LoopGen.pcap, and increase the replay speed gradually (for example, 5000, 10000, 15000, 20000, ...) until the latency reaches the same level as under LoopGen.

```
mininet> xterm h1
h1> tcpreplay -i h1-eth0 -p 5000 -l 0
normalDoS.pcap
```

2. Then, open a new h1 xterm and ping h2.

```
h1> ping 10.0.0.2
```

Results. Compare the latency under LoopGen with the latency under normal DoS while increasing the replay rate. The expected outcome is that the non-looping DoS requires a much higher replay rate than LoopGen to reach a comparable impact, demonstrating LoopGen's cost-effectiveness.

(E4.1) Stealthiness against Identity-verification [5 human-minutes + 5 compute-minutes]. Verify Claim (C4). Read ./Experiment/Stealthiness/IdentityVerification/readme.md for detailed instructions.

Preparation. Please see Section A.3.1.

Execution. 1. Open a terminal (entering the p4 directory by default). 2. Start the identity-verification application.

```
ryu-manager ryu/ryu/app/switchfix.py
```

3. Open a second terminal and start the Mininet topology.

```
cd loopgenexp/identityverification/
sudo python3 topology.py
```

4. Open two sets of host terminals and start the responder scripts in one set.

```
mininet> xterm h1 h2 h3
mininet> xterm h1 h2 h3
```

In the first h1 xterm, run

```
sudo python3 responder1.py h1-eth0
```

In the first h2 xterm, run

```
sudo python3 responder2.py h2-eth0
```

In the first h3 xterm, run

```
sudo python3 responder3.py h3-eth0
```

5. In the second set of host terminals, start the LoopGen attack scripts. Note: Don't hit Enter right away. Type the command on all three first, then press Enter on all of them at once.

In the second h1 xterm, run

```
sudo python3 hack_h1_auto.py
```

In the second h2 xterm, run

```
sudo python3 hack_h2_auto.py
```

In the second h3 xterm, run

```
sudo python3 hack_h3_auto.py
```

6. Open Wireshark and monitor s1-eth2.

Results. Monitor s1-eth2 with Wireshark. The expected outcome is that packets still loop continuously, indicating that this identity-verification defense does not stop LoopGen in the tested setting.

(E4.2) Stealthiness against EventScope [10 human-minutes + 10 compute-minutes]. Verify Claim (C4). Read ./Experiment/Stealthiness/EventScope/README.md for detailed instructions.

Preparation. Please see Section A.3.1. The default JDK in the image is JDK 11, so first switch to JDK 8.

```
export JAVA_HOME=/usr/lib/jvm/java-8-openjdk-amd64
export PATH=$JAVA_HOME/bin:$PATH
java -version
```

Execution. Run the static analysis.

```
cd ~/EventScope-master
java -jar $ANALYZER_JAR
```

Open the generated event-flow-graph.pdf report.

Results. The expected outcome is that EventScope completes and generates the PDF analysis report for org.onosproject.fwd.ReactiveForwarding. It can be seen that the attack did not introduce new event listeners or data plane interaction paths in the graph. This is because it merely added static mappings to the existing event handling logic, thereby altering the forwarding semantics but not the event flow structure.

(E4.3) Stealthiness against SVHunter [5 human-minutes + 5 compute-minutes]. Verify Claim (C4). Read ./Experiment/Stealthiness/SVHunter/README.md for detailed instructions.

Preparation. Please see Section A.3.1. The default JDK in the image is JDK 11, so first switch to JDK 8.

```
export JAVA_HOME=/usr/lib/jvm/java-8-openjdk-amd64
export PATH=$JAVA_HOME/bin:$PATH
java -version
```

Execution. 1. Compile the analysis tool.

```
cd ~/SVHunter-master/tracer
mvn clean package
```

2. Run the analysis on the ONOS source tree.

```
java -jar target/SVHunter-Tracer-0.1-SNAPSHOT-jar-with-dep
endencies.jar \
~/onos \
config/onosEnv.cfg \
config/onosSensitiveMethodList.xml \
config/onosDFMethodList.xml
```

3. Inspect the result file under ~/SVHunter-master/tracer/backwardFlow.

Results. The expected outcome is that SVHunter completes successfully and the result file does not report an alert for `org.onosproject.fwd.ReactiveForwarding`, indicating that this workflow does not detect LoopGen through SVHunter.

(E4.4) Stealthiness against Lemon [5 human-minutes + 5 compute-minutes]. Verify Claim (C4). Read `./Experiment/Stealthiness/Lemon/readme.md` for detailed instructions.

Preparation. Please see Section A.3.1.

Execution. 1. Open a terminal (entering the `p4` directory by default). 2. Start the BMv2-based Mininet experimental environment.

```
cd tutorials/exercises/lemon/
sudo p4run
```

3. Start the attack. The file `output-1000B.pcap` is the attack traffic set, and `202201011400.pcap` is the background traffic.

```
mininet> xterm h1 h1
h1> tcpreplay -i h1-eth0 --duration=4
output-1000B.pcap
h1> tcpreplay -i h1-eth0 --pps=3200 --duration=4
--loop=0 202201011400.pcap
```

4. Open a new terminal and start the Lemon controller.

```
sudo python3 controller.py
```

Inspect the controller output. The destination `10.0.0.8` does not appear in the output.

5. Then open a new terminal and start Wireshark.

```
sudo wireshark
```

Verify the loop by sending only one attack packet.

```
h1> tcpreplay -i h1-eth0 -L 1 output-1000B.pcap
```

Results. The expected outcome is that Lemon does not report the LoopGen destination in the controller output, while Wireshark still shows that the attack packet is looped multiple times. This indicates that the attack remains stealthy to Lemon in this workflow.

(E4.5) Stealthiness against SISTAR [5 human-minutes + 5 compute-minutes]. Verify Claim (C4). Read `./Experiment/Stealthiness/SISTAR/readme.md` for detailed instructions.

Preparation. Please see Section A.3.1.

Execution. 1. Open a terminal (entering the `p4` directory by default). 2. Start the BMv2-based Mininet experimental environment.

```
cd tutorials/exercises/sistar/
make
```

3. Open a new terminal and install the flow rules for the `S1-S2-S3` triangle loop topology.

```
cd tutorials/exercises/sistar/
simple_switch_CLI --thrift-port 9090 <<< "table_add
MyIngress.ipv4_lpm MyIngress.ipv4_forward 10.0.99.99/32 =>
00:00:00:00:02:00 2"
simple_switch_CLI --thrift-port 9091 <<< "table_add
MyIngress.ipv4_lpm MyIngress.ipv4_forward 10.0.99.99/32 =>
00:00:00:00:03:00 3"
simple_switch_CLI --thrift-port 9092 <<< "table_add
MyIngress.ipv4_lpm MyIngress.ipv4_forward 10.0.99.99/32 =>
00:00:00:00:01:00 4"
```

4. Deploy the defense mechanism.

```
bash run_defense.sh
```

5. Open two xterms and send the background traffic and the attack traffic.

```
tcpreplay -i eth0 -p 15000 -L 10000
./202201011400.pcap &
tcpreplay -i eth0 -p 1000 -l 20000 attack.pcap
```

Then open Wireshark and monitor `s1-eth2`.

Results. The expected outcome is that, even after the SISTAR defense is deployed, Wireshark still shows looped attack packets whose destination is `10.0.99.99`. This indicates that the attack remains stealthy to SISTAR in this workflow.

(E5) Controller-side countermeasures [10 human-minutes + 10 compute-minutes]. Verify the second half of Claim (C5). We recommend to read the `./Experiment/defense/readme.md` in the artifact.

Preparation. Please see Section A.3.1.

Execution. 1. Open a terminal (entering the `p4` directory by default).

• **Loop-Detection Countermeasure.**

1. Start the Ryu controller with topology observation enabled.

```
ryu-manager --observe-links
ryu/ryu/app/loop_detect_13.py
```

2. Open a new terminal and start the Mininet topology.

```
cd loopgenexp/ControllerFeasibility-ryu/
sudo python3 -E LoopGen_topo.py
```

3. Open the host terminals in Mininet.

```
mininet> xterm h1 h2 h3
```

4. Send packets in each host terminal.

In `h1`, run

```
python3 hack_h1_auto.py
```

In `h2`, run

```
python3 hack_h2_auto.py
```

In `h3`, run

```
python3 hack_h3_auto.py
```

5. Inspect the controller output.

• **Time-Constraint Countermeasure.**

1. Start the Ryu controller with the time-constraint defense and topology observation enabled.

```
ryu-manager --observe-links
ryu/ryu/app/time_constraint_13.py
```

2. Open a new terminal and start the Mininet topology.

```
cd loopgenexp/ControllerFeasibility-ryu/
sudo python3 -E LoopGen_topo.py
```

3. Open the host terminals in Mininet.

```
mininet> xterm h1 h2 h3
```

4. Send packets in each host terminal.
In `h1`, run

```
python3 hack_h1_auto.py
```

In `h2`, run

```
python3 hack_h2_auto.py
```

In `h3`, run

```
python3 hack_h3_auto.py
```

5. Inspect the controller output.
Results. The expected outcome is that both controller-side countermeasures detect or suppress the LoopGen attack rather than allowing an endless loop to persist. The corresponding controller outputs should indicate successful detection or suppression for the two methods.

A.5 Version

Based on the LaTeX template for Artifact Evaluation V20231005. Submission, reviewing and badging methodology followed for the evaluation of this artifact can be found at https://secartifacts.github.io/usenixsec2026/.