
# USENIX Security '24 Artifact Appendix - SMARTCOOKIE: Blocking Large-Scale SYN Floods with a Split-Proxy Defense on Programmable Data Planes

Sophia Yoo, Xiaoqi Chen, Jennifer Rexford - Princeton University

## A Artifact Appendix

### A.1 Abstract

The artifact consists of two major pieces: 1) source code for the switch agent and server agent of SMARTCOOKIE's split-proxy SYN-flooding defense, related benchmark code, and measurement scripts (showing availability), and 2) a hardware testbed for running and evaluating SMARTCOOKIE under key attack scenarios (showing functionality and reproducibility).

### A.2 Description & Requirements

For the purposes of this artifact evaluation, our testbed consists of five servers and an Intel Tofino Wedge32X-BF programmable switch. Three machines act as attack machines. Two of the attack machines have 4-core Intel Core i5-6500 CPUs and Mellanox ConnectX-4 1x100Gbps NICs, generating attack traffic using DPDK 21.11.0 and pktgen-DPDK 21.11.0. The third attack machine has a 20-core Intel Xeon E5-2680 CPU and an Intel Xl710 2x40 Gbps NIC, generating attack traffic using DPDK 19.11.11 and pktgen-DPDK 19.12.0. Two other machines act as server and client, each with 8-core Intel Xeon D-1541 CPUs and Intel X552 2x10Gbps NICs. For simplicity of artifact evaluation, we are providing evaluators with access to our preconfigured testbed (access instructions below). Instructions for installations and dependencies are briefly included for completeness, but all installations and dependencies are already in place for the evaluation testbed.The remainder of this section describes how to access the source code and testbed, what hardware and software dependencies are required (these are preconfigured for the testbed), and what additional benchmarks can be run.

#### A.2.1 Security, privacy, and ethical concerns

There are no security, privacy, or ethical concerns or risks to evaluators or their machines. All experiments can be run on the authors' testbed, which is provisioned for the planned attack rates. For testbed access, please do not share or distribute the private key (discussed further below).

#### A.2.2 How to access

**Accessing the code:** Source code for the switch agent, server agent, benchmarks, and measurements are hosted on Github: https://github.com/Princeton-Cabernet/SmartCookie-Artifact/releases/tag/v1.0.1. Detailed instructions (encompassing the information in this appendix and much more) are also included in the README of the Github artifact.

**Accessing the testbed:**

- Save the SSH private access key (shared with you directly on the submission site) to your local machine under `~/.ssh/usenixsec24ae.priv.id_rsa`. Note your sudo password was also shared with you on the submission site. Update the permissions with `chmod 600 ~/.ssh/usenixsec24ae.priv.id_rsa`. Start the ssh-agent and load the key: `eval $(ssh-agent -s)` and `ssh-add ~/.ssh/usenixsec24ae.priv.id_rsa`.

#### A.2.3 Hardware dependencies

The switch agent requires an Intel Tofino Wedge32X-BF programmable switch. In order to compare the defense performance of the switch agent to that of other benchmark defenses, the adversarial machines must be capable of generating at least 60 Mpps of combined adversarial traffic. This can, for example, be accomplished with either two attack machines with 20 cores and 2x100Gbps links, or with three or more machines with fewer cores. (Note: to reach the full defense capacity without any packet loss on the SMARTCOOKIE switch agent, at least *150 Mpps* of combined adversarial traffic should be generated, but this can be difficult to accomplish within a limited hardware setting.)

#### A.2.4 Software dependencies

**Switch Agent Prerequisite**[1]**:** please use `bf-sde` version 9.7.1 or newer to compile the P4 program.

**Server Agent Prerequisite:** please use kernel 5.10 or newer and the latest version of the `bcc` toolkit.

**Adversary Machine Prerequisite:** please use DPDK 19.12.0 or newer and a matching pktgen-DPDK version.

[1] For evaluation simplicity, all software dependencies are pre-installed and configured on the artifact testbed.

### A.2.5 Benchmarks

**Hashing:** We compare the cookie hashing performance of SMARTCOOKIE's switch-based HalfSipHash to that of AES (on the switch) and XDP (on the server). Source code for running the benchmarks are under `/p4src/benchmark` and `/ebpf/benchmark/` respectively.

## A.3 Set-up

This section includes basic installation and configuration steps to launch SMARTCOOKIE and prepare the testbed environment for evaluation. It also walks through a simple functionality test of the switch agent and server agent, with an end-to-end connection test between a client and server.

### A.3.1 Installation

All hardware and software dependencies have been preconfigured on the artifact testbed, so there are no additional installation steps for the purposes of this evaluation. However, for completeness, basic installation steps will be included in the artifact repo: https://github.com/Princeton-Cabernet/SmartCookie-Artifact/releases/tag/v1.0.1.
The source code can cloned from the repo with "`git clone git@github.com:Princeton-Cabernet/SmartCookie-Artifact.git`".

### A.3.2 Basic Test

**Terminal 1 - Compile and Launch the Switch Agent:** First, open a new terminal window and SSH into the switch. Clone the SMARTCOOKIE artifact repo and `cd SmartCookie-Artifact/p4src`. Run the `./switchagent_compile.sh` script to compile the program. This may take a few seconds, and you will see some warnings, but these can safely be ignored. Once the compilation is complete, run `./switchagent_load.sh` to load the `SMARTCOOKIE-HalfSipHash.p4` program onto the switch. A successful load should output "bfruntime gRPC server started" as the last log line and land on the switch driver shell starting with `bfshell>`. Keep this terminal open while running experiments, and open other terminals for other operations. If, for any reason you need to restart the switch driver, run `sudo killall bf_switchd` first, then run `./switchagent_load.sh` to reload the program again. Next, to configure the switch interfaces, copy and paste the below in the `bfshell>` to manually bring up ports:

```
ucli
pm
port-add 1/1 10G NONE
an-set 1/1 2
port-enb 1/1
an-set 1/1 1
port-add 1/3 10G NONE
an-set 1/3 2
port-enb 1/3
an-set 1/3 1
port-add 3/0 100G RS
port-enb 3/0
port-add 4/0 100G RS
port-enb 4/0
port-add 5/0 40G NONE
an-set 5/0 2
port-enb 5/0
an-set 5/0 1
port-add 6/0 40G NONE
an-set 6/0 2
port-enb 6/0
an-set 6/0 1
show
rate-period 1
rate-show
```

The last three commands will list packet counts and throughput rates for each of the switch interfaces.

**Terminal 2, 3, & 4 - Launch the Server Agent:** Open three other terminal windows and access the server agent in each window. Clone the artifact repo if you haven't already, and `cd SmartCookie-Artifact/ebpf`. Run `./configure/configure_server.sh` once to configure static IP addresses and ARP entries. Next, use the provided python scripts in the separate terminals to compile and load the eBPF programs to the interface connected to the switch:

1) `sudo python3 xdp_load.py enp3s0f1` for ingress
2) `sudo python3 tc_load.py enp3s0f1` for egress

You should see output that the programs have been loaded. Finally, run the following python script to sync timestamps between the server agent and switch agent, which is necessary for cookie checks: `sudo python3 send_ts.py`.

**Terminal 5 & 6 - A Quick Functionality Test:** To test a simple end-to-end connection between the client and server (protected by the intermediate switch agent and server agent), open two more terminals. SSH into the client and SSH once more into the server. On the server, start up a `netcat` server with `nc -l -p 2020`. On the client, connect to the `nc` server with `nc 131.0.0.6 2020`. The client will seamlessly connect to the server after verification at the switch agent, and you can send messages between the client and server, with the messages popping up on the receiving side. If you are curious, you can use `tcpdump -evvvnX -i enp3s0f1` on both client and server to view the full packet sequence during connection setup, and map it to that of Figure 4 in the paper[2].

[2]Note that tcpdump is positioned after XDP on the *ingress* pipeline, and after TC on the *egress* pipeline (XDP–>tcpdump–> network stack on ingress, and network stack–>TC->tcpdump on egress).

## A.4 Evaluation workflow

There are three main experiments that showcase the key results and major claims of our work. These are described next.

### A.4.1 Major Claims

**(C1):** SMARTCOOKIE *defends against attacks without packet loss until high rates (up to 136 Mpps), significantly outperforming the benchmarks of other defenses, which become exhausted at attack rates starting at only ~1.3 Mpps up to ~52 Mpps. This is proven by experiment (E1), and described in Section 8.2 of the paper.*

**(C2):** *During attacks,* SMARTCOOKIE *protects benign clients from performance penalties and protects servers from additional CPU usage. It adds little to no latency overhead to benign connections during attacks, and any latency is comparable to the baseline latency with no ongoing attack. Additionally, it protects the server's CPU during attacks, fully keeping the CPU resources for other applications. This is proven by experiments (E2) and (E3), and shown in Section 8.3 and 8.4 of the paper.*

### A.4.2 Experiments

Please note that due to space constraints, only the minimal instructions for experiments are listed here. We strongly recommend you follow the experiment workflow with all the detailed steps using the README of our artifact: https://github.com/Princeton-Cabernet/SmartCookie-Artifact.

**(E1):** [Hashing Throughput] [1 human-hour]: Compare the hashing throughput SMARTCOOKIE can achieve *without packet loss* to the the maximum hashing throughput of the benchmarks: SMARTCOOKIE-AES (SC-AES) and XDP-HalfSipHash (XDP-HSH). The Tx (response) rates should exactly match Rx (received) rates for as long as SMARTCOOKIE or the benchmark is handling the attack without any packet loss. Once a defense begins to reach its capacity, the Tx rate will begin to dip below Rx rates.
**Preparation:** There are different preparation steps for each of the benchmarks. Due to space limitations, we refer evaluators to the README for the individual benchmarks. For just the SMARTCOOKIE system, launch the switch agent, as described in §A.3.2. In three additional terminals, SSH into the attack machines where DPDK and pktgen are already configured for you. For each attack terminal, `cd /home/shared/pktgen-dpdk` and launch pktgen with `sudo -E tools/run.py testbed`.
**Execution:** From within the `Pktgen:/>` console of each of the attack machines, launch the SYN flood against the server. Use the commands specified in the artifact README (Section 4.1).
**Results:** To verify the attack rates that SMARTCOOKIE can handle before any packet loss, increase the sending attack rates on the attack servers. As long as the Rx/Tx rates match in the switch agent, the switch agent is successfully defending against the SYN flood attack packets without any packet loss.

**(E2):** [Latency Overhead] [1 human-hour]: Measure the end-to-end setup latency for benign client connections during an attack, to show that SMARTCOOKIE adds *little to no latency overhead* to the baseline without any attack.
**Preparation:** Bring up the SMARTCOOKIE switch agent and server agent as described previously and in the README. Start up an HTTP server on port 8090 on the server (see README for details).
**Execution:** Launch the attack, following the instructions from E1 (refer to the README for details). On the client machine, measure the connection latency across multiple clients by running the script `./experiments/measurements/collect_latency.sh <attack_rps> <local_port>`, specifying the current attack rate and desired local source port.
**Results:** The latency collection script will print out connection latency. You can turn off the attack and verify what the average end-to-end connection latency is for the baseline without the attack, and compare it to the latency during an attack, which should be relatively close.

**(E3):** [Server CPU Usage] [1 human-hour]: Measure the server's CPU usage during attack, to show that SMARTCOOKIE fully protects server CPU usage during attacks.
**Preparation:** The preparation is identical to that of E2. Bring up the SMARTCOOKIE switch agent and server agent as described previously and in the README. Start up an HTTP server on port 8090 on the server (see README for details).
**Execution:** Launch the attack, following the instructions from E1 (refer to the README for details). The simplest way to verify and visualize the CPU usage is with 'htop' on the server machine. You can keep a terminal open to continually track CPU usage during the experiment. The more robust way of measuring the CPU usage is to run the script `./experiments/measurements/collect_cpu_instr.sh <attack_rps>`, while specifying the current attack rate and where the collected data should be stored.
**Results:** The CPU usage collection script will store CPU usage rates in the specified directory. You can turn off the attack and verify what the CPU usage is for the baseline without the attack, and it should directly match the CPU usage during an attack (effectively none).

## 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/usenixsec2024/.