CISPA
HELMHOLTZ CENTER FOR
INFORMATION SECURITY

Efficient and Scalable Fuzzing of Complex Software Stacks

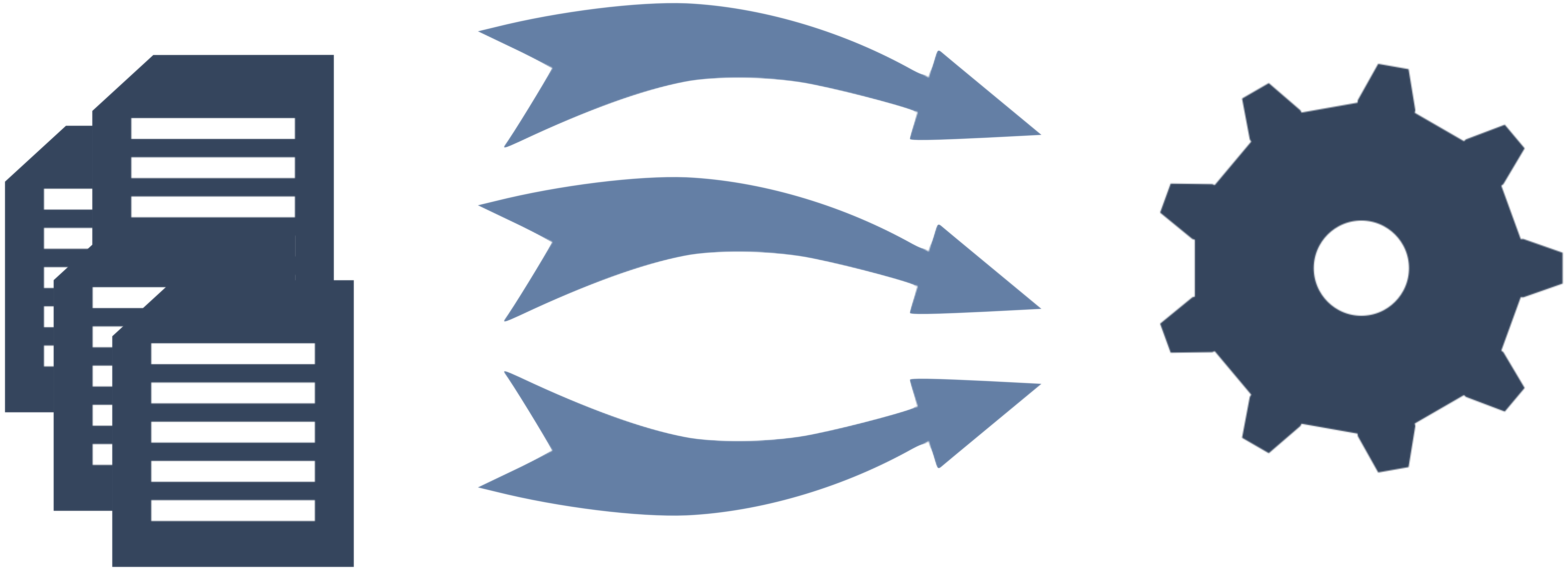
Prof. Thorsten Holz

ASSURE

Recent Research

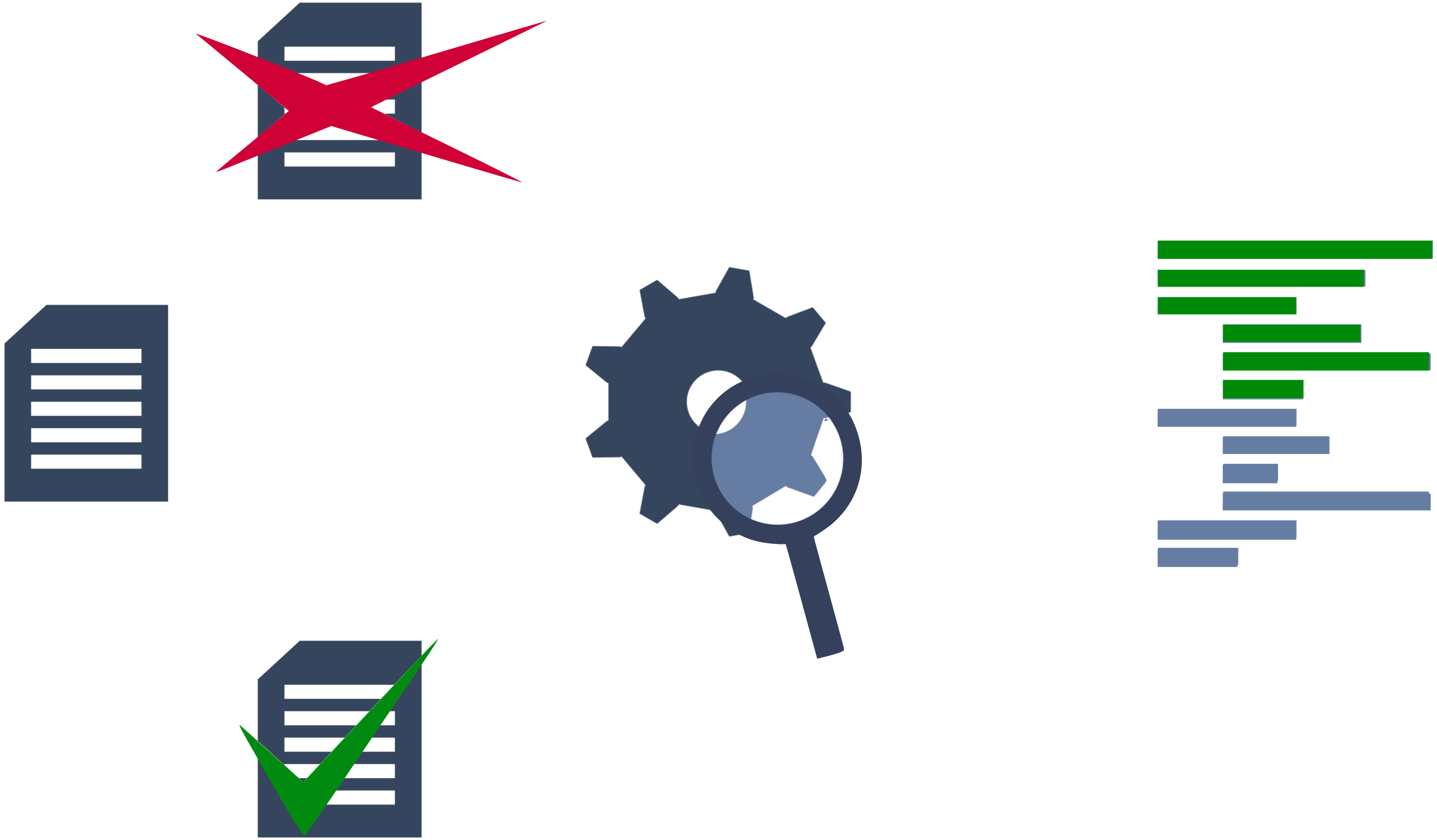
Fuzzing Overview

Fuzzing Overview

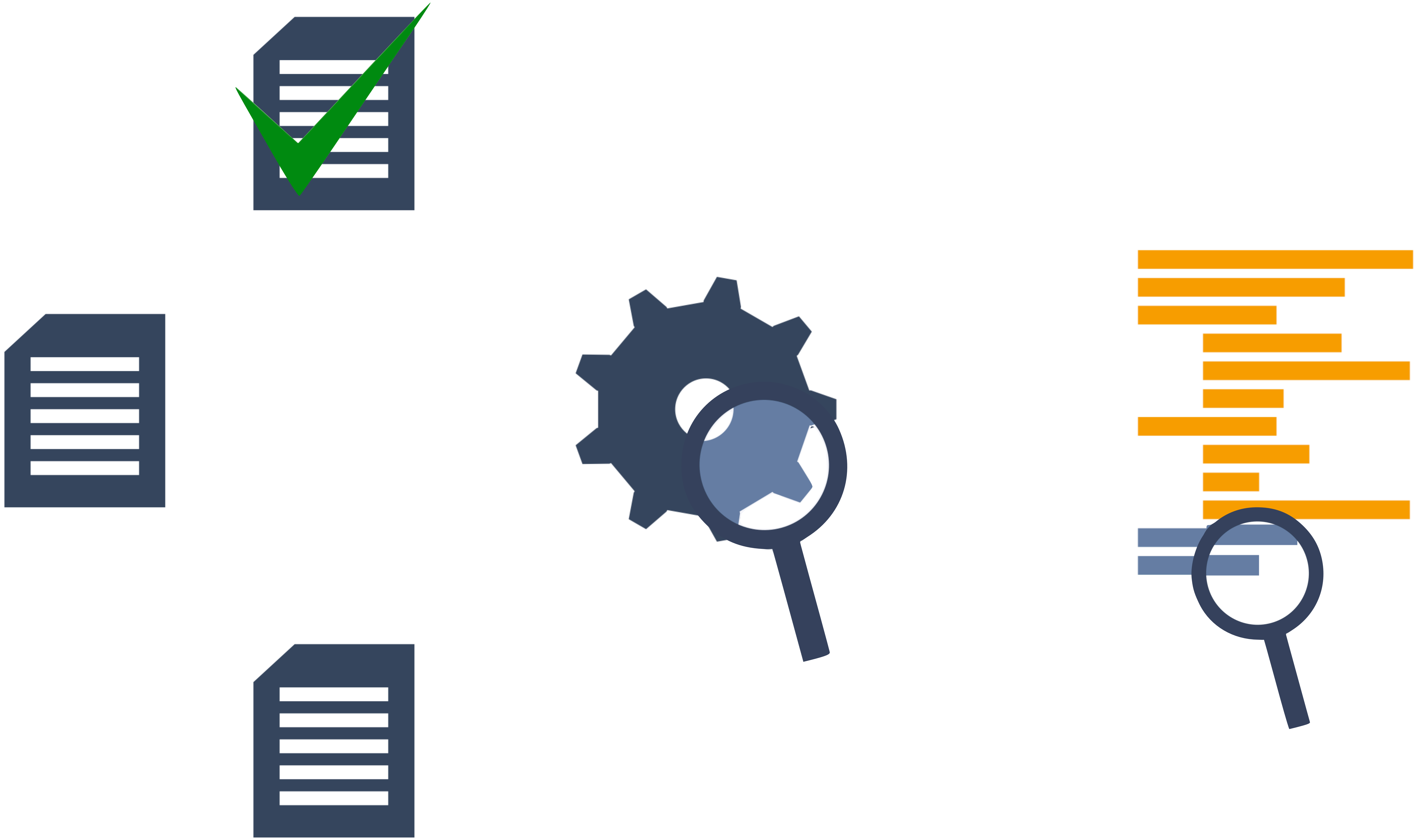


Miller: http://pages.cs.wisc.edu/~bart/fuzz/ (1988)

Coverage-Guided Fuzzing



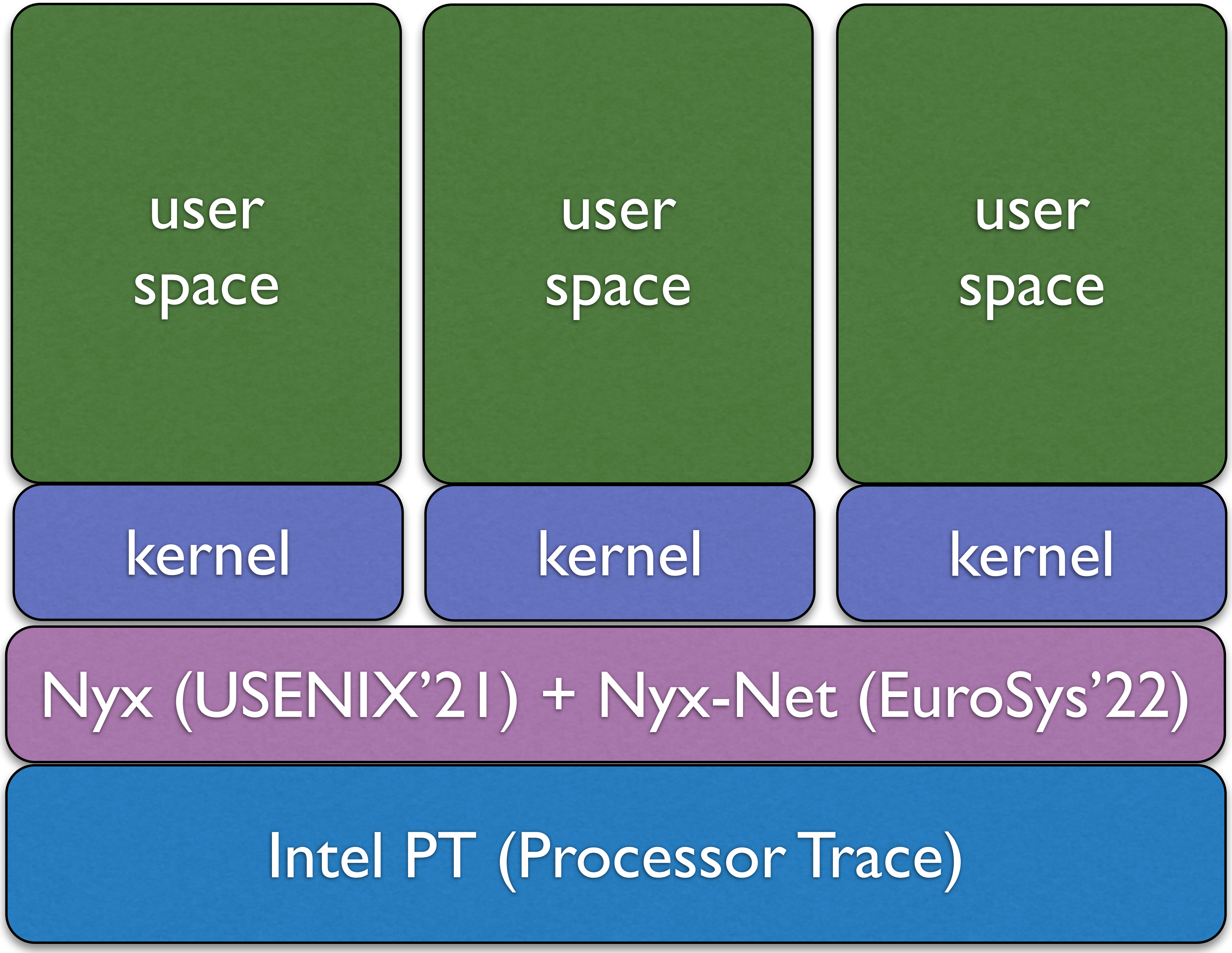
Coverage-Guided Fuzzing



NYX / NYX-Net

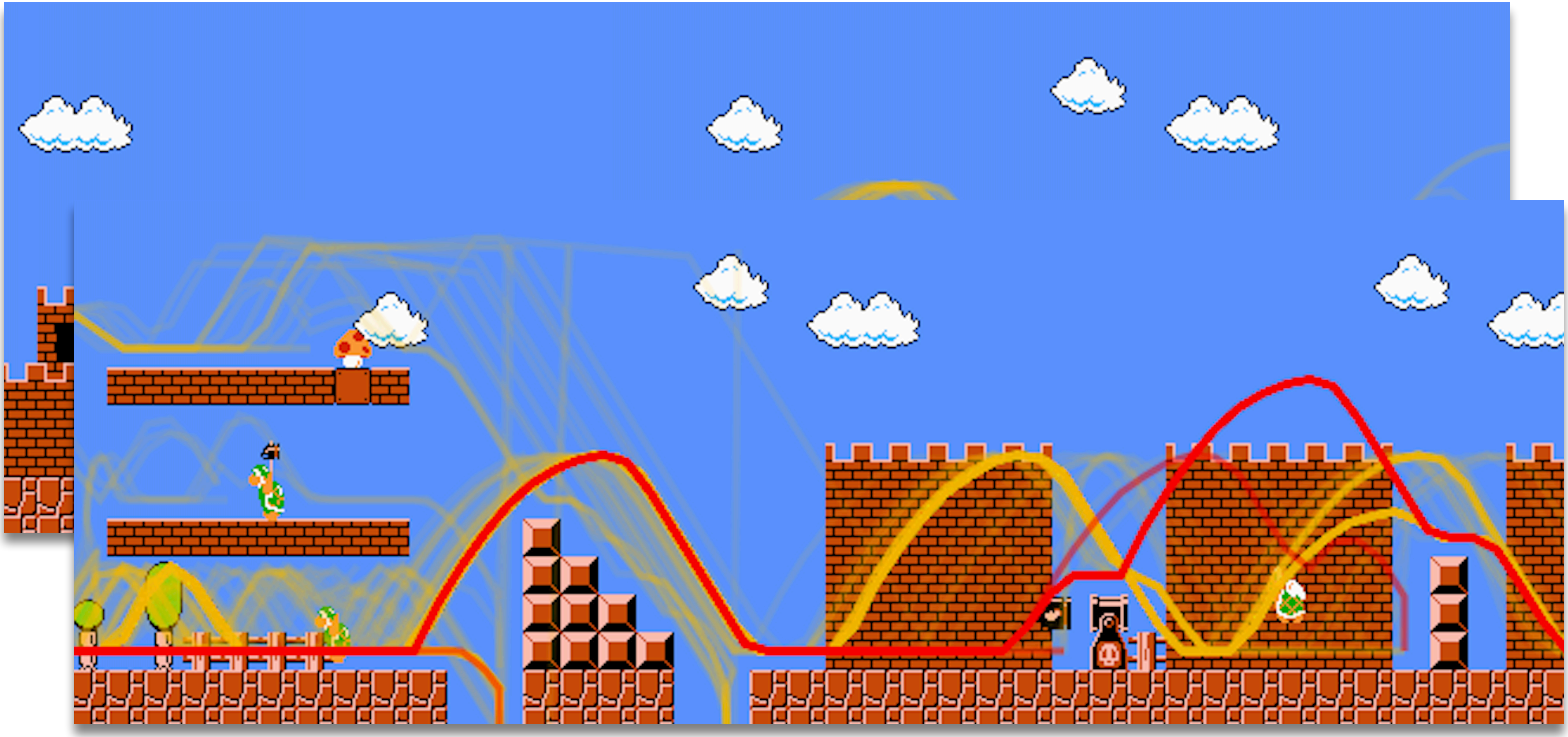
Fuzzing Overview

https://github.com/RUB-SysSec/

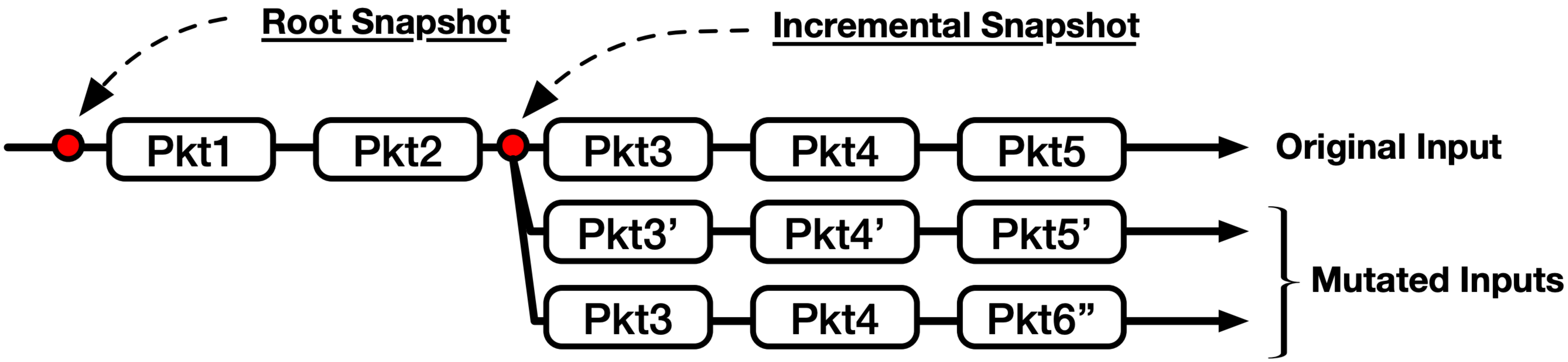

user space
user space
user space
kernel
kernel
kernel
Nyx (USENIX'21) + Nyx-Net (EuroSys'22)
Intel PT (Processor Trace)

Efficient State-Machine Exploration

Efficient State-Machine Exploration

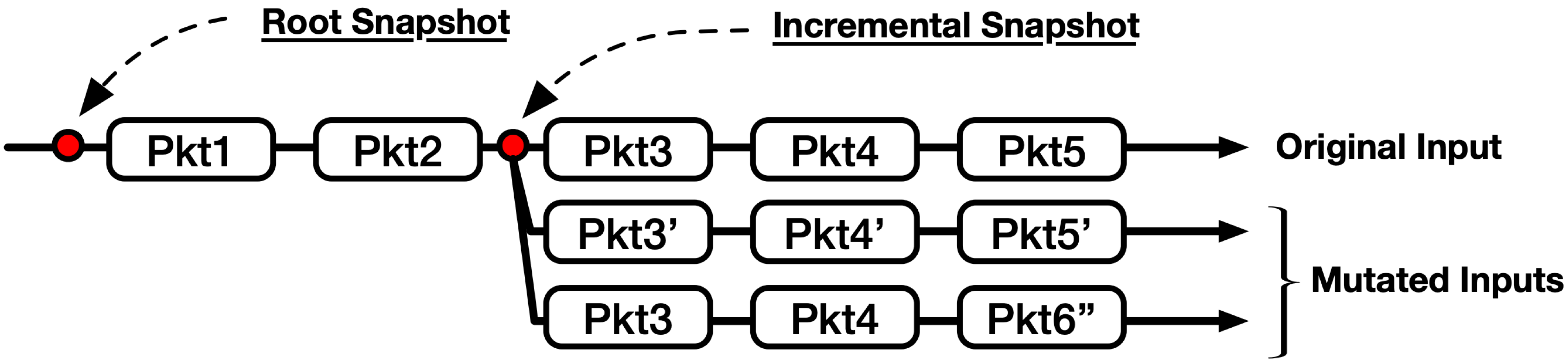


Snapshot-based Fuzzing



Schumilo et al.: “Nyx-Net: Network Fuzzing with Incremental Snapshots”, EuroSys’22

Snapshot-based Fuzzing



Root Snapshot

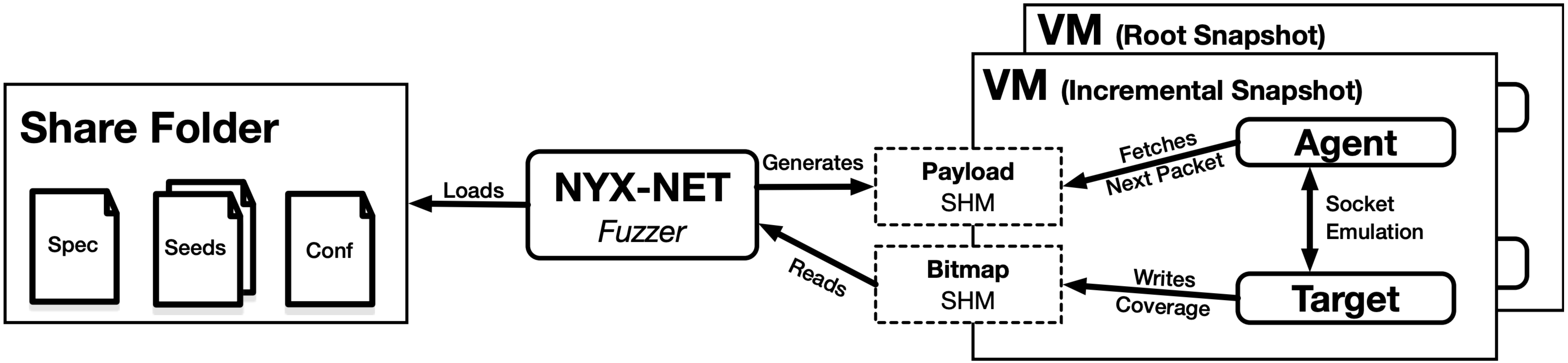
Incremental Snapshot

Pkt1 Pkt2 Pkt3 Pkt4 Pkt5 → Original Input

Pkt3' Pkt4' Pkt5' →

Pkt3 Pkt4 Pkt6" →

Mutated Inputs



VM (Root Snapshot)

VM (Incremental Snapshot)

Share Folder

Spec

Seeds

Conf

Loads

NYX-NET

Fuzzer

Generates

Reads

Payload SHM

Bitmap SHM

Fetches Next Packet

Agent

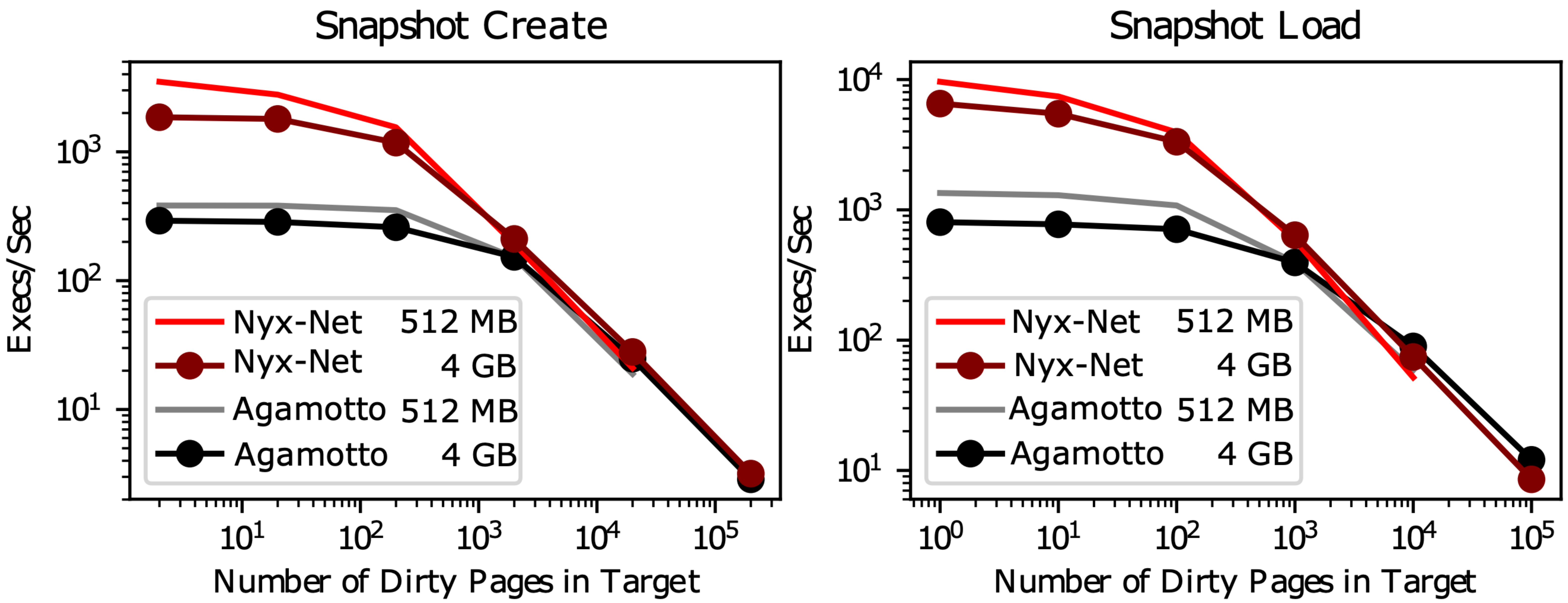
Socket Emulation

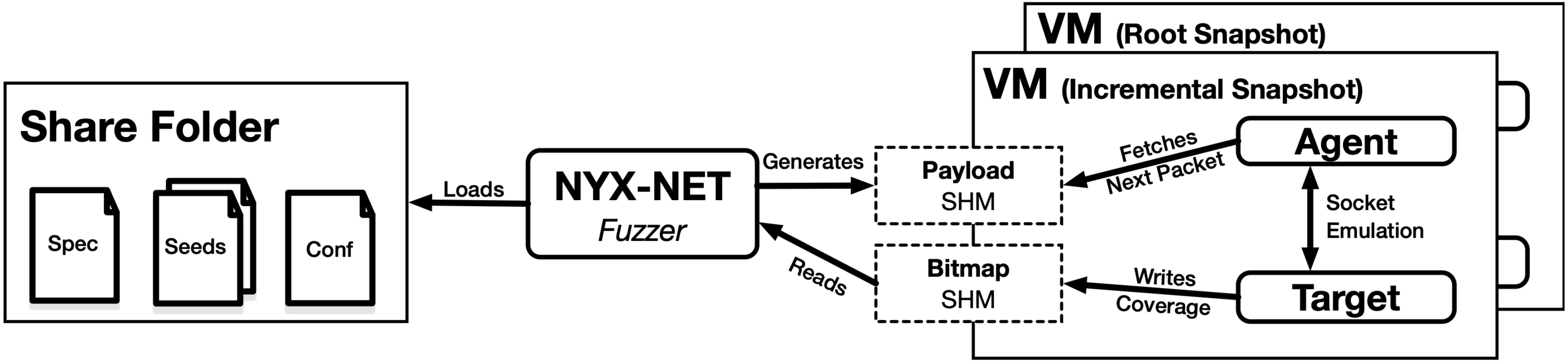
Target

Writes Coverage

Schumilo et al.: "Nyx-Net: Network Fuzzing with Incremental Snapshots", EuroSys'22

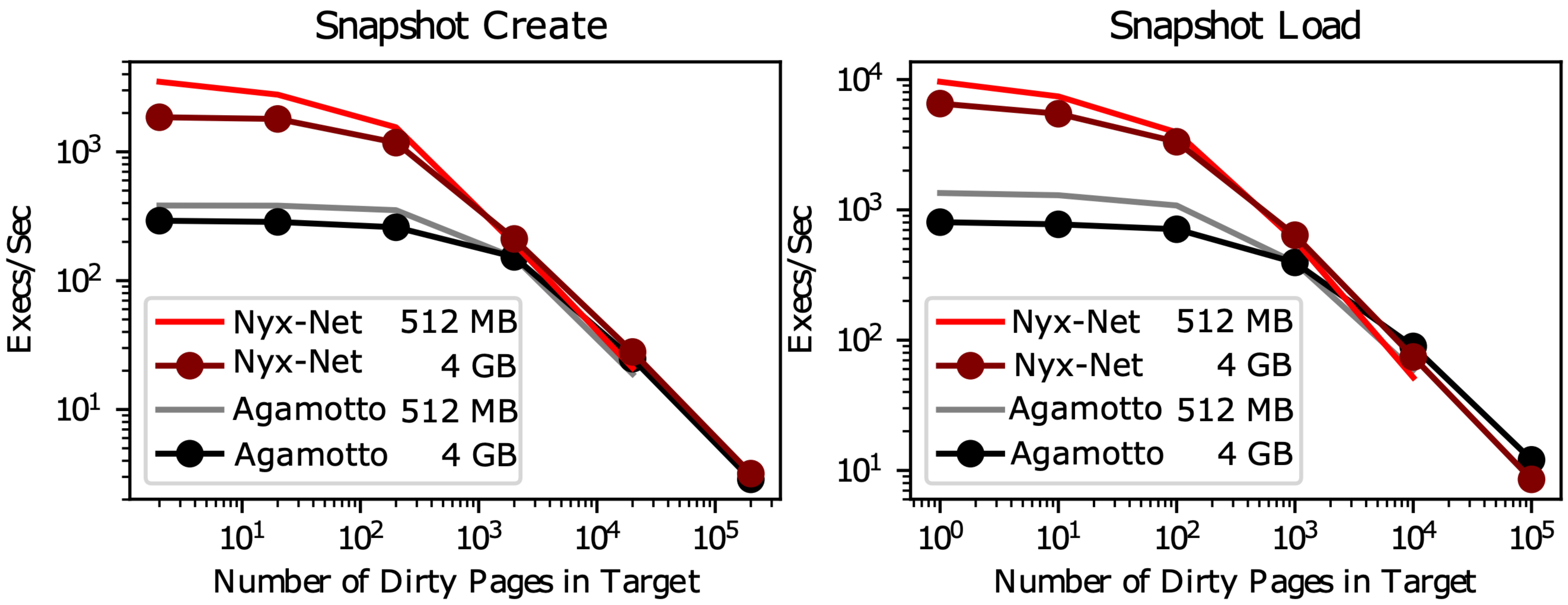
Snapshot-based Fuzzing

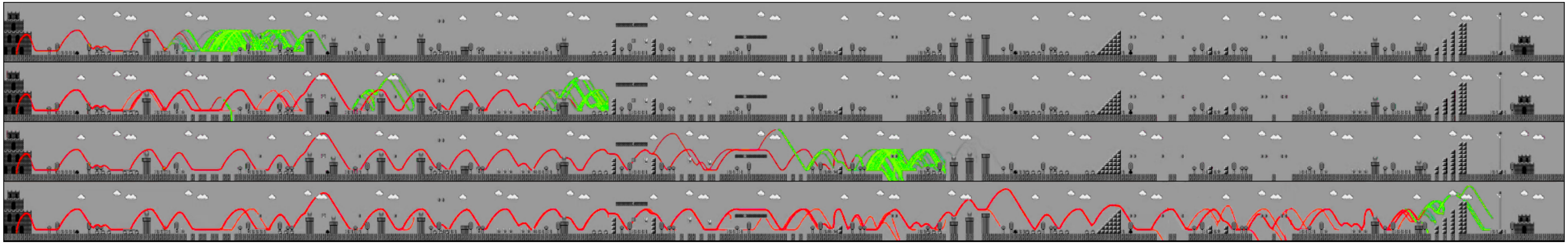




Schumilo et al.: “Nyx-Net: Network Fuzzing with Incremental Snapshots”, EuroSys’22

Snapshot-based Fuzzing

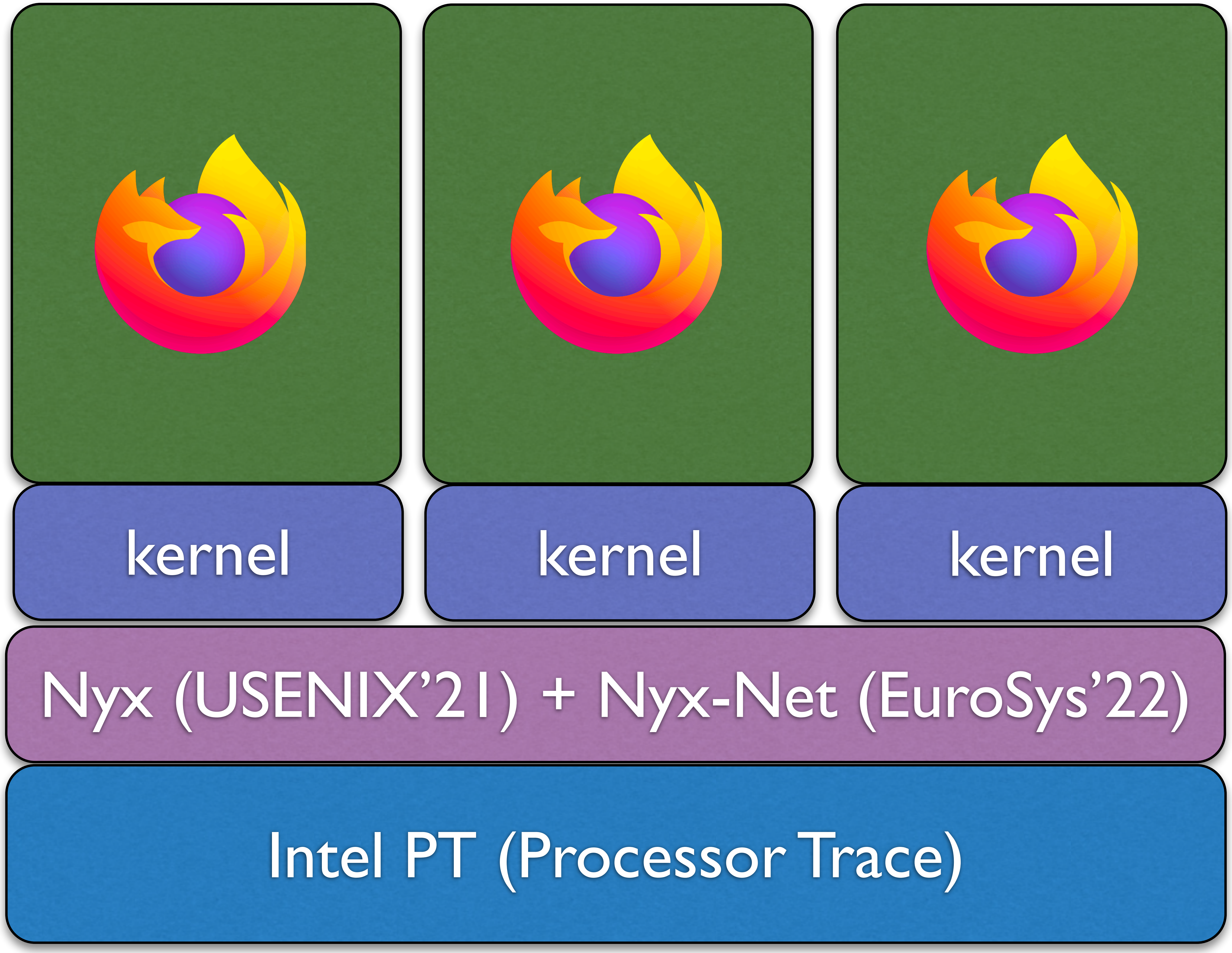




Schumilo et al.: "Nyx-Net: Network Fuzzing with Incremental Snapshots", EuroSys'22

Fuzzing Firefox with Nyx-Net

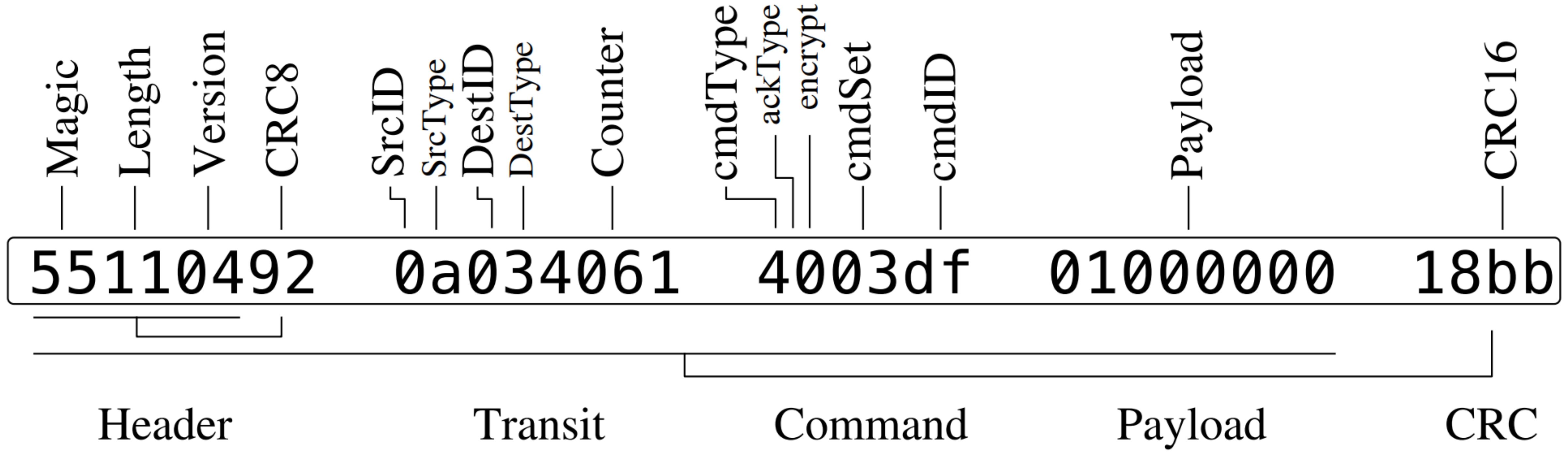
https://nyx-fuzz.com/


kernel
kernel
kernel
Nyx (USENIX'21) + Nyx-Net (EuroSys'22)
Intel PT (Processor Trace)

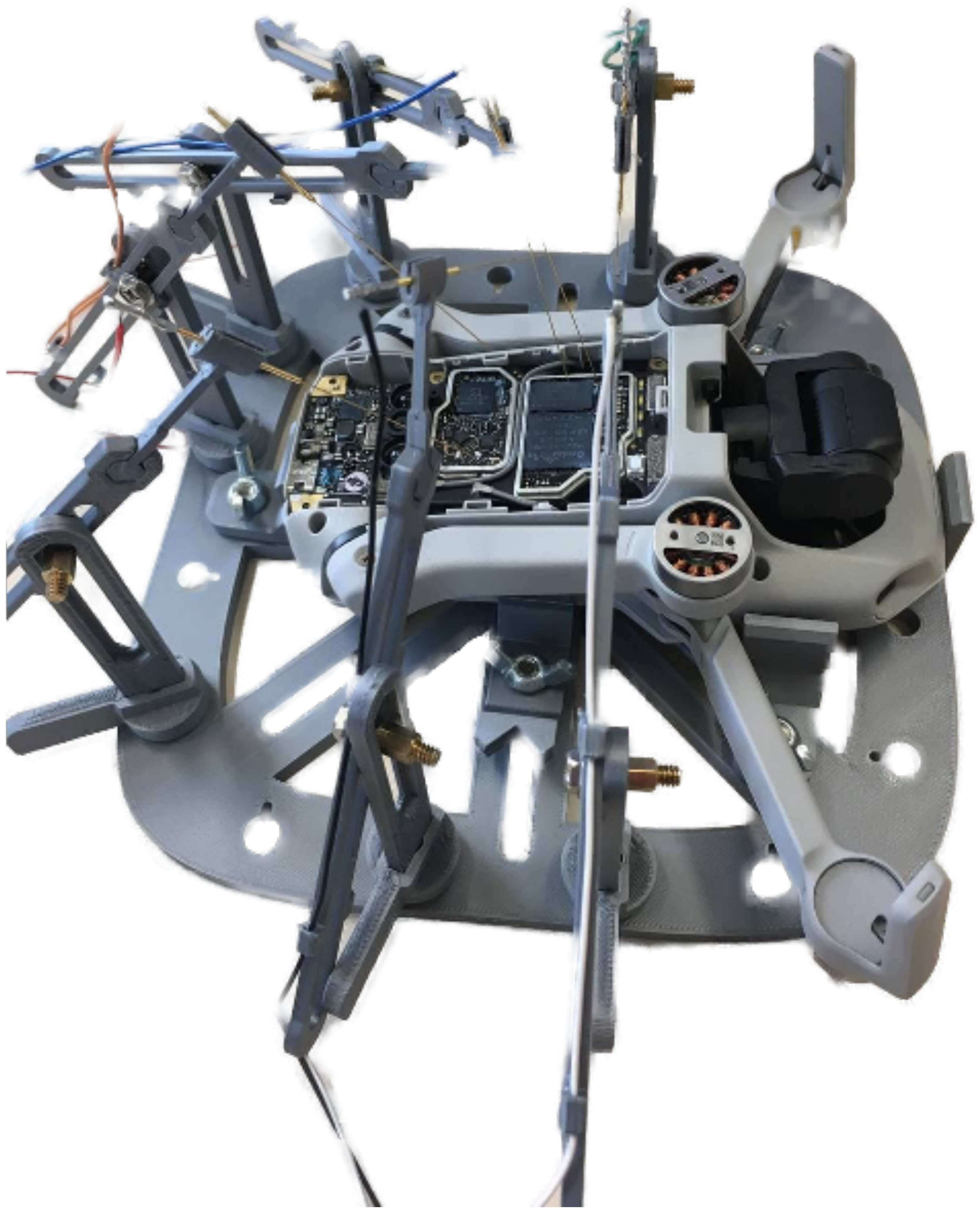
Drone Security

DJI Drones

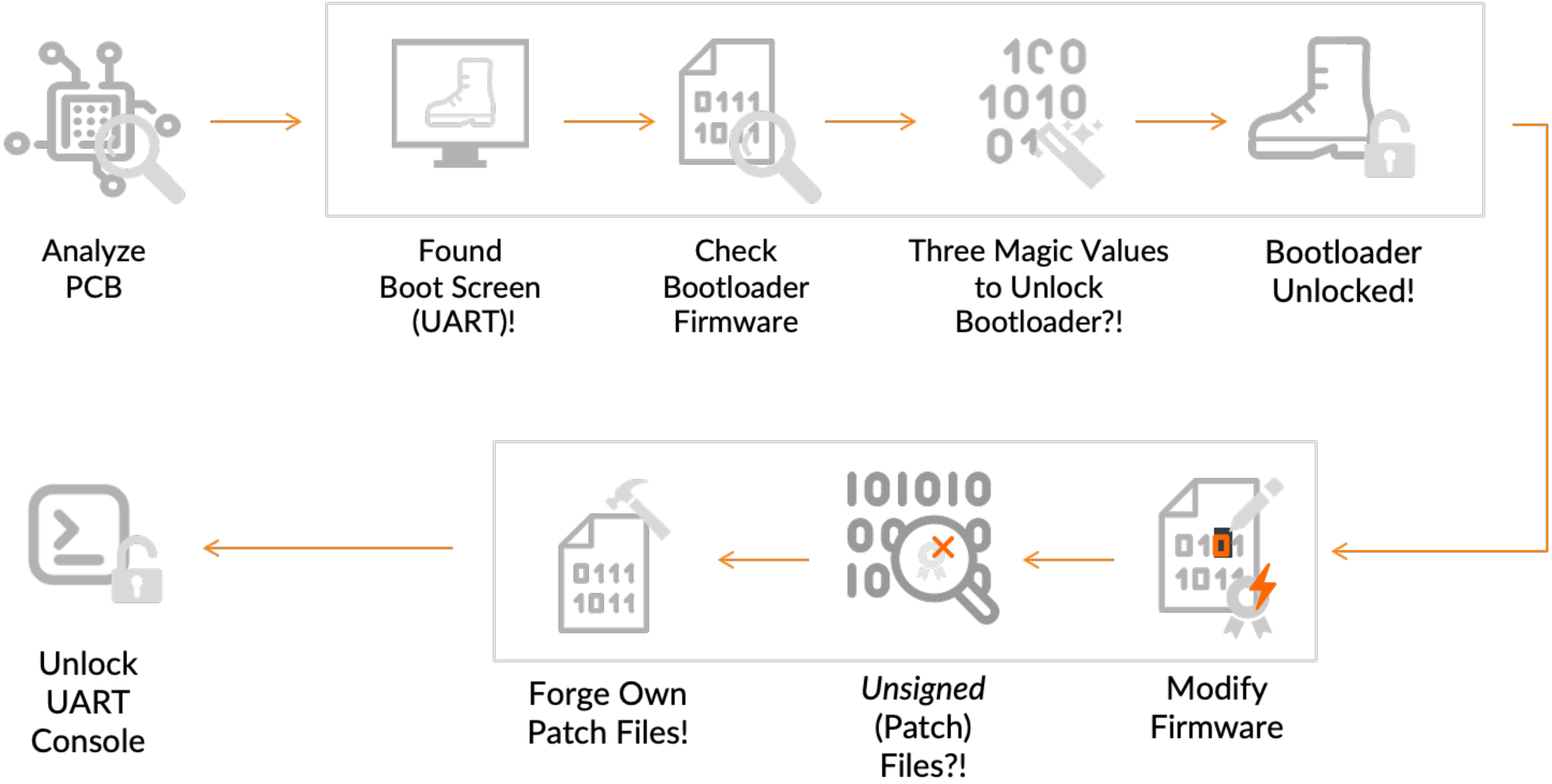
DJI Universal Markup Language (DUML)



```
55110492  0a034061  4003df  01000000  18bb
```



Security Assessment



Analyze PCB

Found Boot Screen (UART)!

Check Bootloader Firmware

Three Magic Values to Unlock Bootloader?!

Bootloader Unlocked!

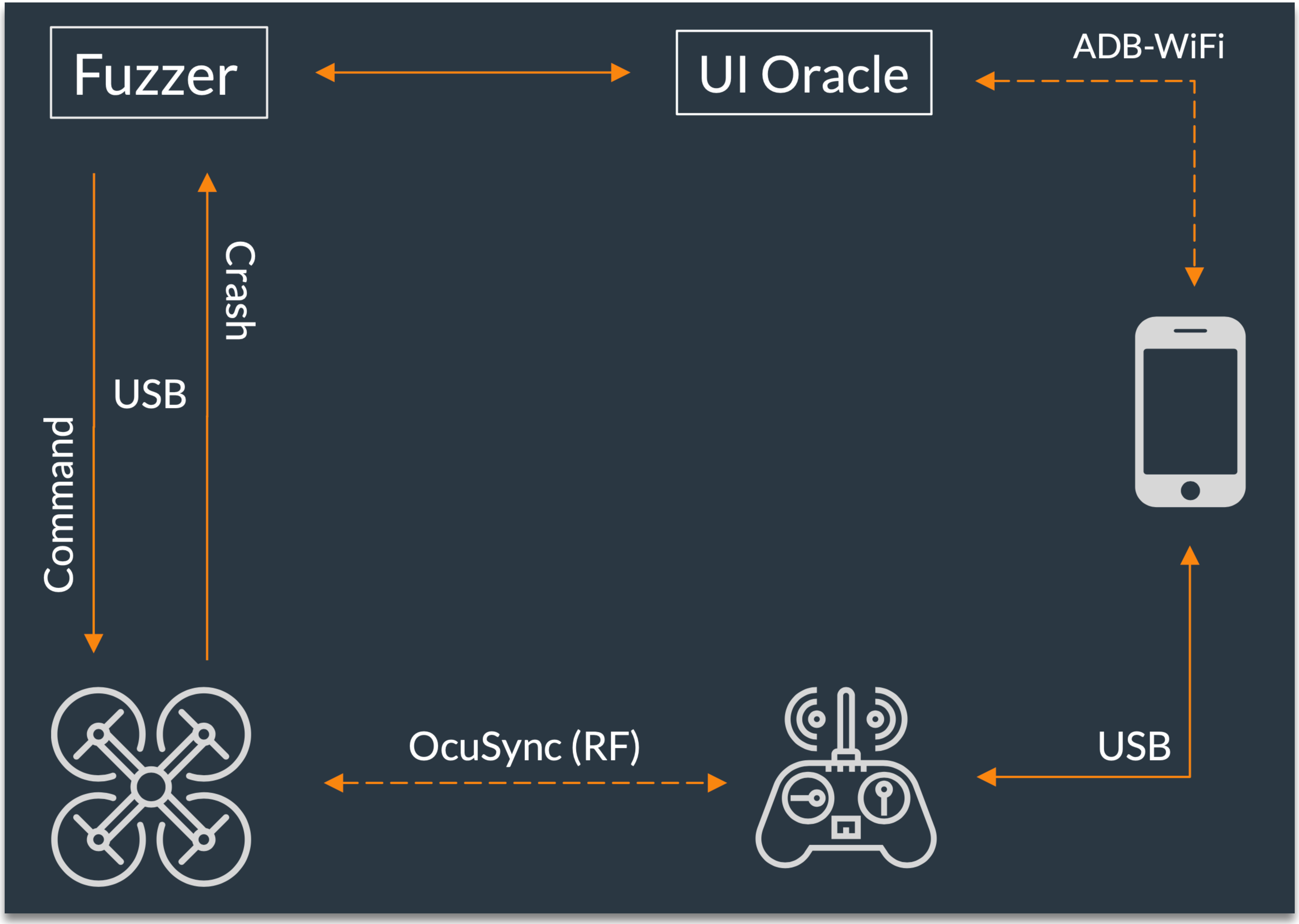
Modify Firmware

Unsigned (Patch) Files?!

Forge Own Patch Files!

Unlock UART Console

Fuzzing Drones



Fuzztruction

Motivation

CISPA HELMHOLTZ CENTER FOR INFORMATION SECURITY

- Pairs of programs encode domain knowledge about given protocol
  - ***Generator*** generates content (e.g., generate PDF file or encrypted message)
  - ***Consumer*** processes content (e.g., display PDF file or decrypt encrypted message)

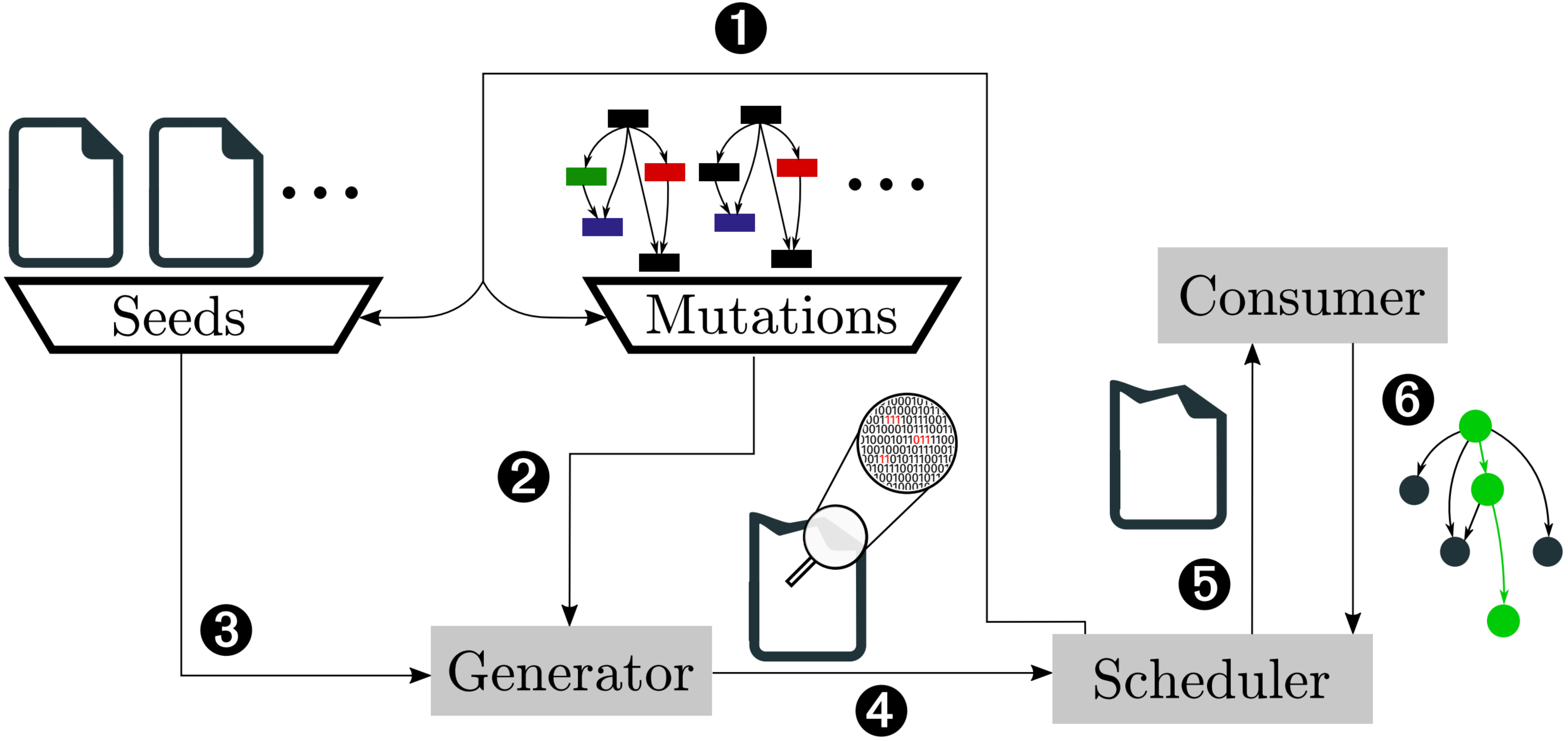
Motivation

- Pairs of programs encode domain knowledge about given protocol
  - ***Generator*** generates content (e.g., generate PDF file or encrypted message)
  - ***Consumer*** processes content (e.g., display PDF file or decrypt encrypted message)
- How can we efficiently test such programs without domain knowledge?
- Basic insight: we can use generator for *input generation*

Intuition

- Randomly flipping instruction bits in generator would not affect output and—even worse—lead to crashes
- Compile-time analysis to identify operations on data and filter out crashing operations
  - Analyze data-flow dependencies to avoid redundant mutations
- Instrument generator and just-in-time (JIT)-compile both tracing and mutation mechanisms

Overview

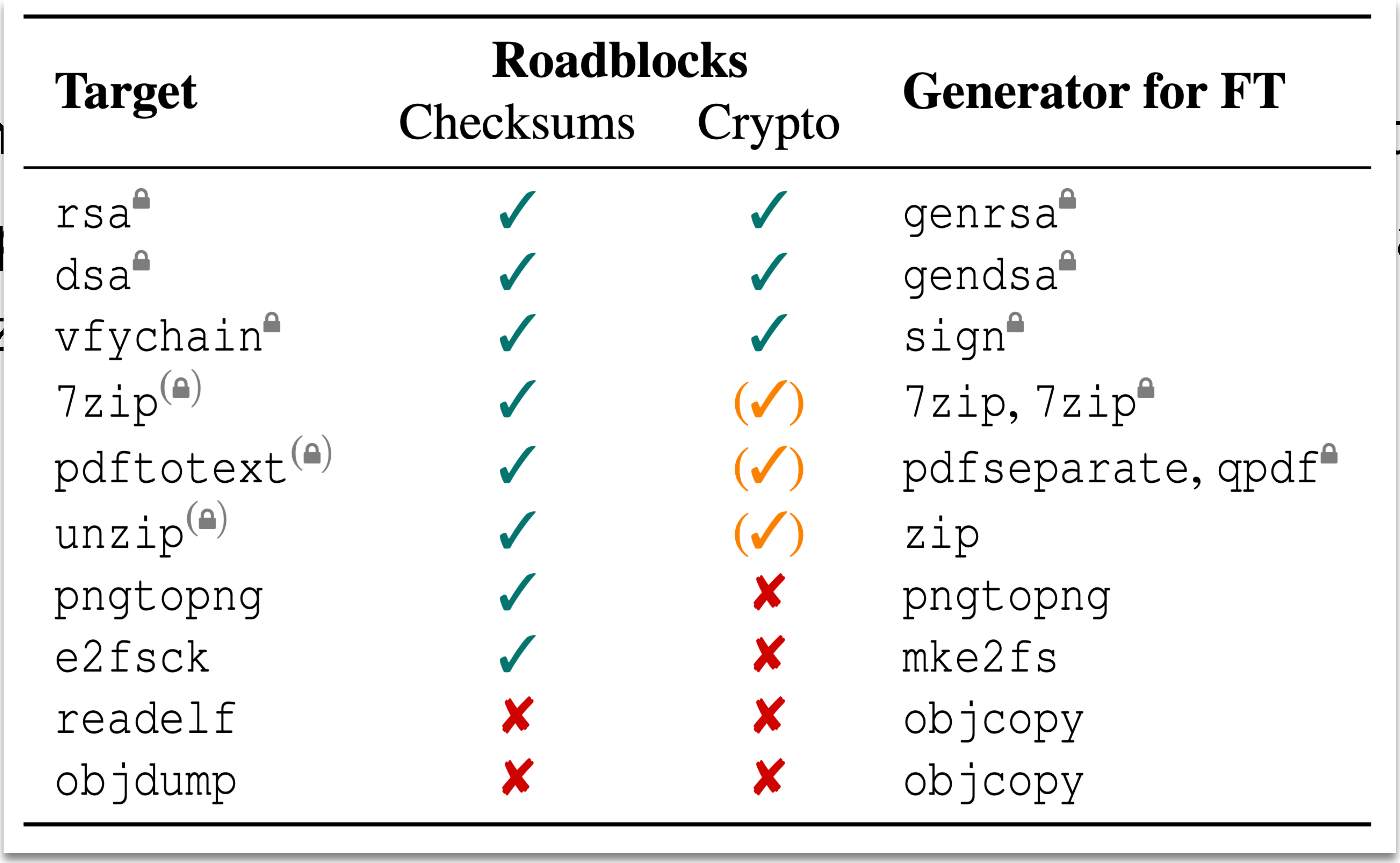

❶
Seeds
Mutations
. . .
❷
❸
Generator
❹
Scheduler
❺
Consumer
❻
10010001011
001111011110011
00100010111001
10001011011100
10010001011001
01110101110011
10111001100001
100010001011

Results

- Loosely Structured Formats (objdump, readelf)
- Complex Formats (pngtopng, unzip, 7zip, and pdftotext)
- Cryptographic Formats (OpenSSL’s dsa and rsa, and Mozilla NSS’ vfychain)

Results

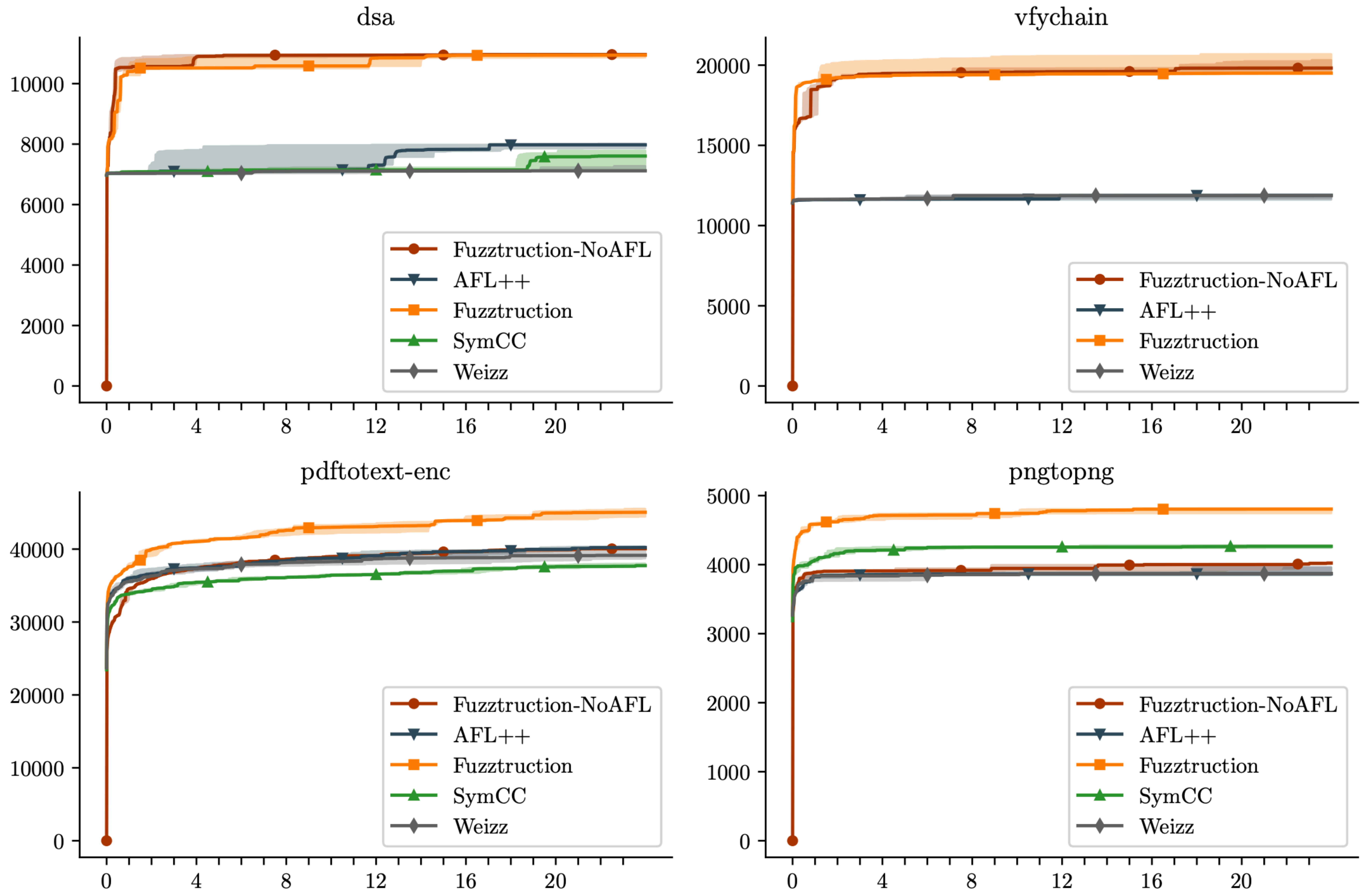
- Loo
- Con dftotext)
- Cry and Moz



Target	Roadblocks		Generator for FT
	Checksums	Crypto	
rsa🔒	✓	✓	genrsa🔒
dsa🔒	✓	✓	gendsa🔒
vfychain🔒	✓	✓	sign🔒
7zip(🔒)	✓	(✓)	7zip, 7zip🔒
pdftotext(🔒)	✓	(✓)	pdfseparate, qpdf🔒
unzip(🔒)	✓	(✓)	zip
pngtopng	✓	✗	pngtopng
e2fsck	✓	✗	mke2fs
readelf	✗	✗	objcopy
objdump	✗	✗	objcopy

Results

- Loc
- Cor dftotext)
- Cry and Mo:



Summary & Outlook

Trophy Cases

macOS High Sierra
TigerVNC
WINE
Parallels Desktop
php
binutils
Perl
CONTIKI
NEXT GENERATION
Qt
Qtjs
Zephyr®
ImageMagick
vmware Fusion
intel.
ACRN
GOODiX
OpenLDAP
curl://
QEMU
Nextcloud
mruby
nasm
the netwide assembler
Libxml2
GNOME
TCC
ORACLE
VirtualBox
Lua
FreeBSD®
bhyve
7ZIP
dji
TIFF
libtiff
TCPDUMP
Chakra Core
Gnuplot
BASH
THE BOURNE-AGAIN SHELL
Fraunhofer FDK for Android™
synaptics®
Windows
COUNTER STRIKE SOURCE™

Towards Secure Systems

- Efficiently fuzz deeper parts of the compute stack
  - UEFI? SMM / SMI handler? MSRs? ISA? Pre-silicon?

Towards Secure Systems

- Efficiently fuzz deeper parts of the compute stack
  - UEFI? SMM / SMI handler? MSRs? ISA? Pre-silicon?
- Machine learning to the rescue?
  - Can we reuse knowledge from previous fuzzing campaigns?
  - Can we use LLMs to generate interesting inputs?

Towards Secure Systems

- Efficiently fuzz deeper parts of the compute stack
  - UEFI? SMM / SMI handler? MSRs? ISA? Pre-silicon?
- Machine learning to the rescue?
  - Can we reuse knowledge from previous fuzzing campaigns?
  - Can we use LLMs to generate interesting inputs?
- How to handle all the bugs founds?
  - Automated root cause analysis
  - Automated patching of found vulnerabilities

References

- Bars et al.: “Fuzztruction: Using Fault Injection-based Fuzzing to Leverage Implicit Domain Knowledge”, USENIX Security’23
- Schiller et al.: “Drone Security and the Mysterious Case of DJI's DroneID“, NDSS’23
- Schumilo et al.: “Nyx: Greybox Hypervisor Fuzzing using Fast Snapshots and Affine Types”, USENIX Security’21
- Schumilo et al.: “Nyx-Net: Network Fuzzing with Incremental Snapshots“, EuroSys’22
- Aschermann et al.: “IJON: Exploring Deep State Spaces via Fuzzing”, IEEE S&P’20
- Aschermann et al.: “Nautilus: Fishing for Deep Bugs with Grammars”, NDSS’19
- Blazytko et al.: “Grimoire: Synthesizing Structure while Fuzzing”, USENIX Security’19
- Aschermann et al.: “Redqueen: Fuzzing with Input-to-State Correspondence”, NDSS’19

Abstract

CISPA
HELMHOLTZ CENTER FOR INFORMATION SECURITY

In this talk, I will give an overview of our recent progress in randomized testing (“fuzzing”) and present some of the methods we have developed in the last few years. These include fuzzing of operating system kernels and hypervisors, ~~grammar-based fuzzing of complex interpreters~~, and fuzz testing of embedded systems. The talk will focus on our recent work on Fuzztruction, a novel perspective on generating inputs in highly complex formats without relying on heavyweight program analysis techniques, coarse-grained grammar approximation, or a human domain expert. I will conclude the talk with an outlook on challenges yet to be solved.