Chair of Network Architectures and Services
School of Computation, Information, and Technology
Technical University of Munich

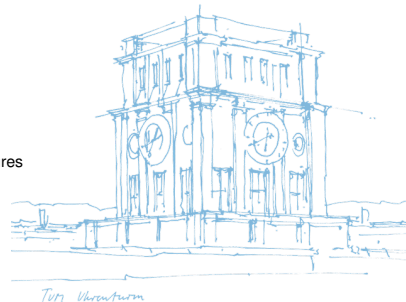
Network Profiles for Detecting Application-Characteristic Behavior Using Linux eBPF

Lars Wüstrich, Markus Schacherbauer, Markus Budeus, Dominik Freiherr von Künßberg,
Sebastian Gallenmüller, Marc-Oliver Pahl*, Georg Carle

Sunday 10th September, 2023

Chair of Network Architectures and Services
School of Computation, Information, and Technology
Technical University of Munich

*Chaire Cybersecurity for Critical Networked Infrastructures
Department SRCD
IMT Atlantique



Motivation

A practical problem . . .

- Do you notice that something is off with the picture on the right?

Complex patterns

Motivation

A practical problem . . .

- Do you notice that something is off with the picture on the right?
- **Hint:** look for a smartphone

Complex patterns

Motivation

A practical problem . . .

- Do you notice that something is off with the picture on the right?
- **Hint:** look for a smartphone
- No, the smartphone is not under the table

Complex patterns

Motivation

A practical problem . . .

- Do you notice that something is off with the picture on the right?
- **Hint:** look for a smartphone
- No, the smartphone is not under the table

Complex patterns

Motivation

A practical problem . . .

- Do you notice that something is off with the picture on the right?
- **Hint:** look for a smartphone
- No, the smartphone is not under the table

What has that to do with computer systems?

- Something can be hard to find even if we know what to look for

Complex patterns
cf. https://mymodernmet.com/rayana-lira-hidden-cell-phone/

Motivation

A practical problem . . .

- Do you notice that something is off with the picture on the right?
- **Hint:** look for a smartphone
- No, the smartphone is not under the table

What has that to do with computer systems?

- Something can be hard to find even if we know what to look for
- Complex patterns are widely spread across computer systems (cf. kernels, network stacks)
- Relevant information may be hidden among complex patterns
- **However,** with the right tools, we can detect this information

Complex patterns

cf. https://mymodernmet.com/rayana-lira-hidden-cell-phone/

Motivation

In this talk, we use eBPF . . .

to enable the detection of application-characteristic network behavior

Our work addresses two main questions:

1. How can we characterize an application's network behavior?
2. How can we efficiently associate packets and processes?

Related eBPF Work

Related Work	Application
Cilium Hubble[1] and Tetragon[2]	Traffic monitoring and policy enforcement
Falco[3]	Intrusion detection
Opensnitch[4]	Packet filtering

[1] Cilium Hubble. https://github.com/cilium/hubble
[2] Cilium Tetragon. https://github.com/cilium/tetragon
[3] Falco. https://falco.org/
[4] Opensnitch. https://github.com/evilsocket/opensnitch

Contributions

- Network application profiles
- We show that eBPF is suitable to reliably associate packets and processes
- Our evaluation shows:
  - Network profiles can identify unexpected process behavior
  - eBPF allows to efficiently and reliably collect the necessary data

Network Profiles

Characterizing Application Behavior

There are various aspects defining characteristic behavior

- Associated processes on end-hosts
- Sequences & dependencies
- Periodicity
- Flow characteristics

$\Rightarrow$ There is no single method to capture all facets

Network Profiles

Methods

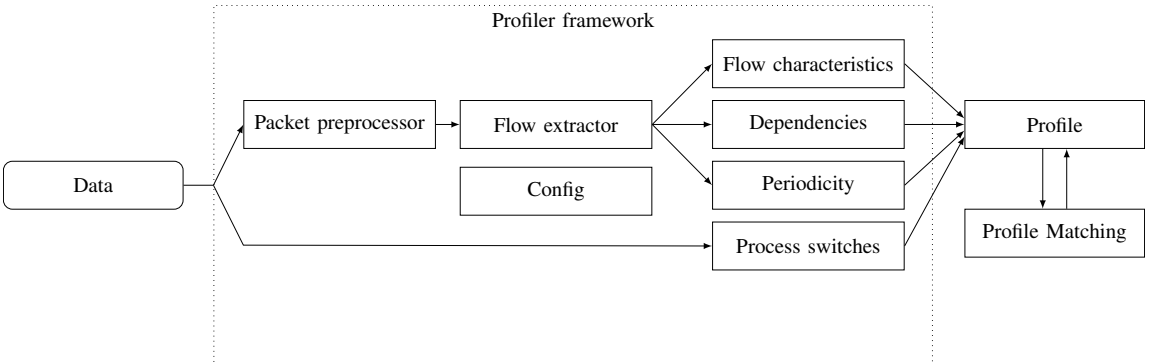
Framework to combine methods to enable network application profiles

Characteristic	Method
Associated processes	**Heuristic measurements**, code analysis
Sequences & Dependencies	**Episode-**, Sequential- Association **Rule Mining**, **Markov model**, config file parsing
Periodicity	**Periodogram**, pattern mining
Flow characteristics	**Energy-based Flow Classifier**[5], frequency analysis

[5] C. F. Pontes et al. A new method for flow-based network intrusion detection using the inverse potts model. IEEE TNSM, 2021

Network Profiles

Process



Network Profiles

To build **accurate** network profiles, we need

- a solid data foundation
- an association between packets and processes
  - complete, i. e., every packet to a process
  - efficient, i. e., little overhead/additional load

⇒ The next slides will focus on the data collection

```yaml
human_name: some_application
process_profiles:  # list of process profiles
- human_name: name_of_process
  dependencies:  # Singular dependencies
  - confidence: 1.0
    ip: some_ip
    protocol: some_protocol
    service: some_service
    support: 10
  dependency_rules: [] # Timing Rules between dependencies
  flow_classifier: # Average energy values of EFC-Classifier
    duration: 8.969111066022899
    # more_keys
    size_out: 15.809292887877858
  periodicity: # Periodicity information
    confidence: 0.8
    max_period: 22.3
    min_period: 26.1
  ports:
  - occurred_port
  process_names:
  - occurred_matcher_name
  protocols:
  - occurred_protocol
  services:
  - occurred_service
process_switches: null # Switches between processes (null as a single process)
```

Correlating Packets and Processes

Method	Implementation	Completeness	Overhead	Ground truth	Related Work
Heuristics					
Polling					
Logging					
eBPF					

[6] H. Asai, et al. "Network application profiling with traffic causality graphs." International Journal of Network Management (2014)
[7] S. Haas et al. Zeek-osquery: Host-network correlation for advanced monitoring and intrusion detection., IFIP TC 11, 2020
[8] L. Popa et al. Macroscope: End-point approach to networked application dependency discovery. CoNEXT, 2009
[9] T. Karagiannis et al. BLINC: multilevel traffic classification in the dark., ACM SIGCOMM, 2005
[10] S. Ma et al. Protracer: Towards practical provenance tracing by alternating between logging and tainting. NDSS, 2016
[11] Cilium Hubble. https://github.com/cilium/hubble
[12] Cilium Tetragon. https://github.com/cilium/tetragon
[13] Falco. https://falco.org/
[14] Opensnitch. https://github.com/evilsocket/opensnitch

Correlating Packets and Processes

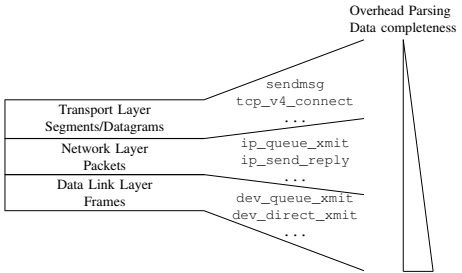
Method	Implementation	Completeness	Overhead	Ground truth	Related Work
Heuristics	++	++	++	−−	Traffic Causality Graphs[6]
Polling	++	−−	−−	+	Zeek-osquery[7], Macroscope[8], BLINC[9]
Logging	−	++	−	++	Protracer[10]
eBPF	−	++	+	++	Hubble[11], Tetragon[12], Falco[13], Opensnitch[14]

[6] H. Asai, et al. "Network application profiling with traffic causality graphs." International Journal of Network Management (2014)
[7] S. Haas et al. Zeek-osquery: Host-network correlation for advanced monitoring and intrusion detection., IFIP TC 11, 2020
[8] L. Popa et al. Macroscope: End-point approach to networked application dependency discovery. CoNEXT, 2009
[9] T. Karagiannis et al. BLINC: multilevel traffic classification in the dark., ACM SIGCOMM, 2005
[10] S. Ma et al. Protracer: Towards practical provenance tracing by alternating between logging and tainting. NDSS, 2016
[11] Cilium Hubble. https://github.com/cilium/hubble
[12] Cilium Tetragon. https://github.com/cilium/tetragon
[13] Falco. https://falco.org/
[14] Opensnitch. https://github.com/evilsocket/opensnitch

Matching via eBPF

Egress Matching

- eBPF allows to execute additional code upon executing syscalls
  - gather information from the `skb`
- Variety syscalls to `kprobe`
- Tradeoff between parsing overhead and data completeness
- Our goal is data completeness



⇒ Our egress data collection focuses on the stated Data Link Layer syscalls

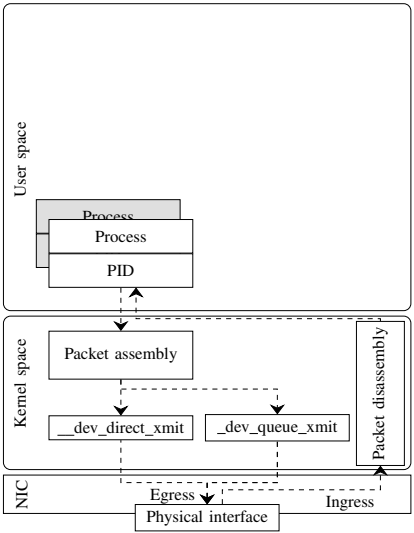
Matching via eBPF

Ingress Matching

- Matching ingress is more difficult than egress
- On reception, it is unclear which process will read it
- We use a set of heuristics to match ingress traffic
  - flow based,
  - via ICMP ID, and
  - via payload in ICMP error messages.

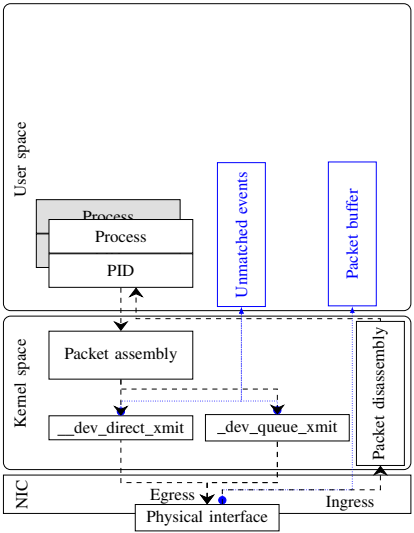
Matching via eBPF

Matcher Architecture



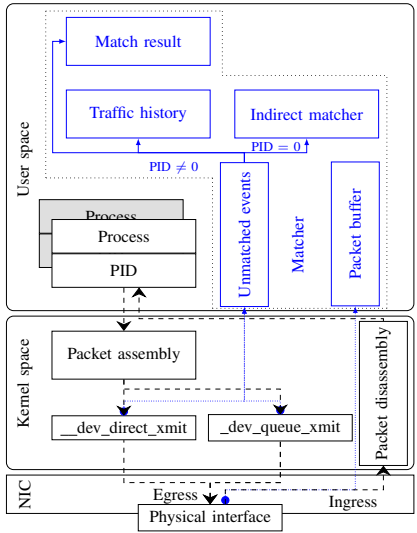
Matching via eBPF

Matcher Architecture



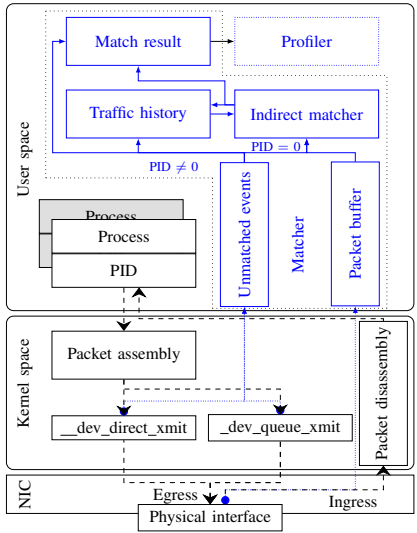
Matching via eBPF

Matcher Architecture



Matching via eBPF

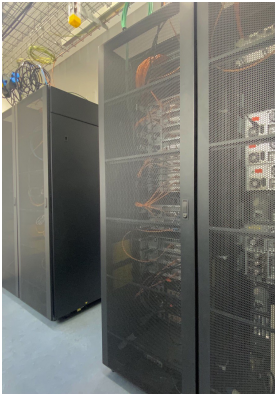
Matcher Architecture



Evaluation

- Our evaluation consists of two parts
  1. profiler framework
  2. matcher
- All experiments in local testbed
- All experiments automated with pos[15]
- Setup: Two directly connected hosts

OS	Debian Bullseye (5.10.0-8-amd64)
CPU	Intel Xeon CPU D-1518 (4×2.2 GHz)
RAM	32 GB



Evaluation testbed

[15] S. Gallenmüller et al. The pos framework: A methodology and toolchain for reproducible network experiments. CoNEXT, 2021

Evaluation

Profiler Evaluation

Goal: Identifying Command and Control (C2) botnet traffic mimicking `ntpd`
Ground truth:

- 10×5 min traces of `ntpd`
- Detected periodicity: 62 s to 75 s

[16] https://www.extrahop.com/resources/attacks/c-c-beaconing/
[17] https://attack.mitre.org/techniques/T1036/004/
[18] https://attack.mitre.org/techniques/T1036/004/

Evaluation

Profiler Evaluation

Goal: Identifying Command and Control (C2) botnet traffic mimicking `ntpd`
Ground truth:

- 10×5 min traces of `ntpd`
- Detected periodicity: 62 s to 75 s

Emulated attack:

- beacon to C2 server every 64 s[16]
- C2 server responds with random string
- process appears as `/usr/sbin/ntpd`[17] [18]
- more details in our paper

- 10×5 min traces during botnet activity

[16] https://www.extrahop.com/resources/attacks/c-c-beaconing/
[17] https://attack.mitre.org/techniques/T1036/004/
[18] https://attack.mitre.org/techniques/T1036/004/

Evaluation

Profiler Evaluation

Goal: Identifying Command and Control (C2) botnet traffic mimicking `ntpd`
Ground truth:

- 10×5 min traces of `ntpd`
- Detected periodicity: 62 s to 75 s

Emulated attack:

- beacon to C2 server every 64 s[16]
- C2 server responds with random string
- process appears as `/usr/sbin/ntpd`[17] [18]
- more details in our paper

- 10×5 min traces during botnet activity

```
/usr/sbin/ntpd:
  dependencies:
    dependency_match:
      missing_known_dependencies: []
      unknown_dependencies:
      - confidence: 1.0
        dst_port: 123
        ip: 192.168.1.2
        protocol: UDP
    dependency_rules: null
  flows:
    match: 0.5669291338582677
  periodicity:
    found_periodicity: null
    match: false
    profile_periodicity:
      confidence: 1.0
      max_period: 75.26808510638298
      min_period: 62.7157400156617
...
```

[16] https://www.extrahop.com/resources/attacks/c-c-beaconing/
[17] https://attack.mitre.org/techniques/T1036/004/
[18] https://attack.mitre.org/techniques/T1036/004/

Evaluation

Matcher Evaluation

- **Goal:** Identify limits of packet-process correlation
- Setup:
  - trafgen for traffic generation
  - generate 10 000 packets/s (64 B and 1500 B)
  - matching rate, memory consumption, CPU load

Evaluation

Matcher Evaluation

- **Goal:** Identify limits of packet-process correlation
- Setup:
  - trafgen for traffic generation
  - generate 10 000 packets/s (64 B and 1500 B)
  - matching rate, memory consumption, CPU load

Matching Rate

- 99.9% at 60 Mbit/s
- 96.3% at 120 Mbit/s

Evaluation

Matcher Evaluation

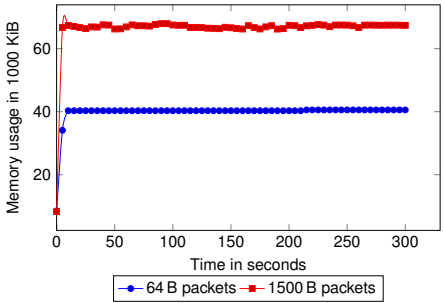
- **Goal:** Identify limits of packet-process correlation
- Setup:
  - trafgen for traffic generation
  - generate 10 000 packets/s (64 B and 1500 B)
  - matching rate, memory consumption, CPU load

Matching Rate

- 99.9% at 60 Mbit/s
- 96.3% at 120 Mbit/s

Memory Consumption

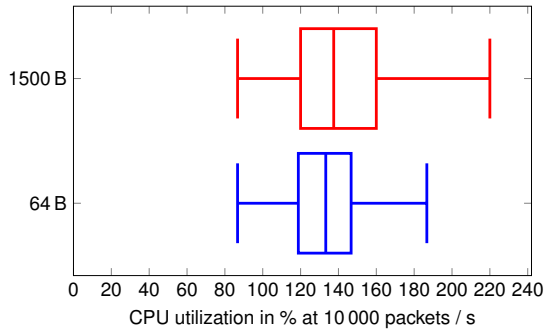
- 40 MiB at 10 000 packets/s (64 B packets)
- 68 MiB at 10 000 packets/s (1500 B packets)



Evaluation

CPU load

- Depends on the number of packets
- Independent of packet size



Contributions

- Network application profiles
- We show that eBPF is suitable to collect the necessary data
- Our evaluation shows:
  - Network profiles can identify unexpected process behavior
  - Our eBPF matcher is efficient and reliable

Future work

- Enhance network profiles
- Ingress matching via eBPF

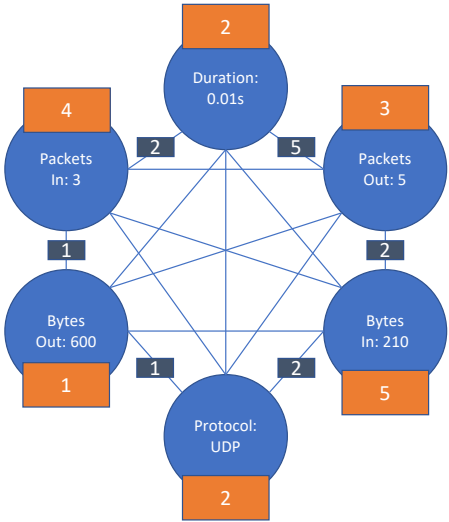
Read our paper:

Code of eBPF matcher:

Contributions

Energy Based Flow Classifier

- Threshold-based binary semi-supervised classifier
- Proposed by Pontes et al.[19]
- Flow modelled as fully connected graph $G = (V, E)$
- Vertices V are features (e. g. protocol)
- Edges E combination of V
- Energy of flow: Sum of all node and edge energies



[19] C. F. Pontes et al. A new method for flow-based network intrusion detection using the inverse potts model. IEEE TNSM, 2021