eBPF: The Double-Edged Sword of Linux Security and Malware

An exploration of how eBPF enhances Linux security while also introducing potential vulnerabilities for malware exploitation.

About Me

Background

Worked in tech for 25 years, cyber security for 10 and a cyber security researcher and red teamer for the past 7.

EDUCATION

- Pursuing Masters @ NYU

EXPERIENCE

- Red Teamer & Cyber Security Researcher @ Wells Fargo
- Spoken at a few conferences and delivered workshops at DefCon

PASSIONS

- Muay Thai
- Camping (Scouting)

Why should I care about this talk?


eBPF
DOCUMENTARY FILM
eBPF: UNLOCKING THE KERNEL
BY SPEAKEASY

Full Documentary

https://youtu.be/Wb_vD3XZYOA

What is eBPF?

eBPF originally stood for extended Berkeley Packet Filter

It is a revolutionary technology that allows running sand-boxed programs in the Linux kernel.

Powerful and flexible virtual machine

eBPF provides a safe and efficient way to execute custom code within the kernel, enabling complex data processing and monitoring tasks.

Versatile use cases

eBPF can be used for networking, tracing, security, and performance profiling, making it a valuable tool for system administrators and developers.

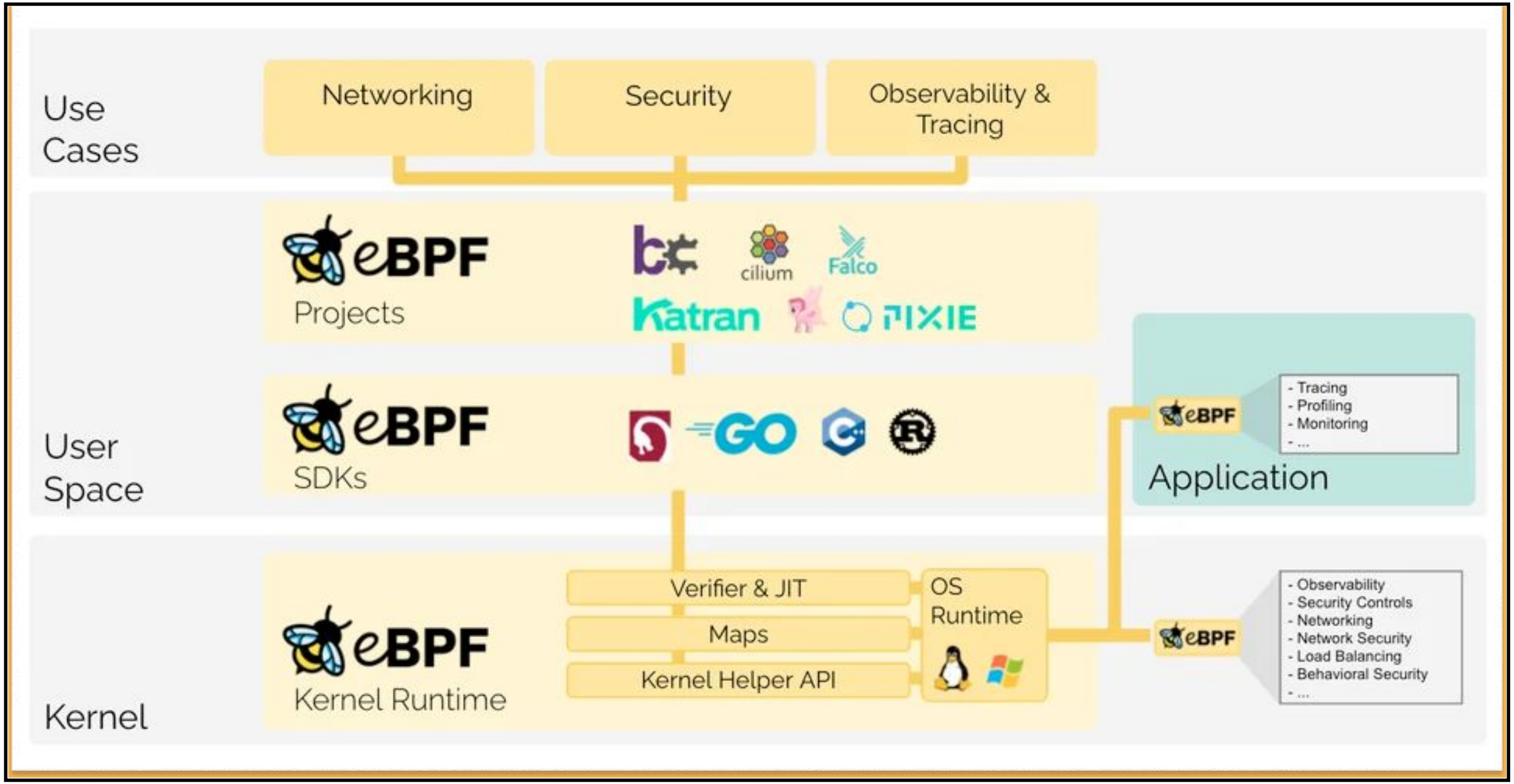
eBPF is a powerful and versatile technology that enhances Linux functionality.

"eBPF brings **super powers** to Linux."

- Brendan Greggs, Netflix

"eBPF does for the Linux kernel what JavaScript **did for the web**."

- Everyone who talks about eBPF (unattributed)


Use Cases
Networking
Security
Observability & Tracing
eBPF
Projects
cilium
Falco
Katran
PIXIE
User Space
eBPF
SDKs
GO
Application
eBPF
- Tracing
- Profiling
- Monitoring
- ...
Kernel
eBPF
Kernel Runtime
Verifier & JIT
Maps
Kernel Helper API
OS Runtime
eBPF
- Observability
- Security Controls
- Networking
- Network Security
- Load Balancing
- Behavioral Security
- ...

Legitimate Uses of eBPF

Network Packet Filtering

eBPF can create efficient network filters that inspect and process packets inline, improving performance over traditional user-space solutions.

Application Tracing and Profiling

eBPF enables comprehensive tracing and profiling of applications, providing deep insights into their behavior and performance characteristics.

Kernel-Level Security Policies

eBPF can be used to implement various security policies at the kernel level, such as system call filtering, process monitoring, and sandboxing.

eBPF offers powerful capabilities for network filtering, application analysis, and security enforcement, making it a versatile tool for system administrators and developers.

Real-World eBPF Examples

Cilium Network Monitoring

Cilium leverages eBPF for network visibility, enforcing security policies, and load balancing.

Sysdig Falco

Falco uses eBPF to detect anomalous activity and threats in applications and containers.

bcc Tools

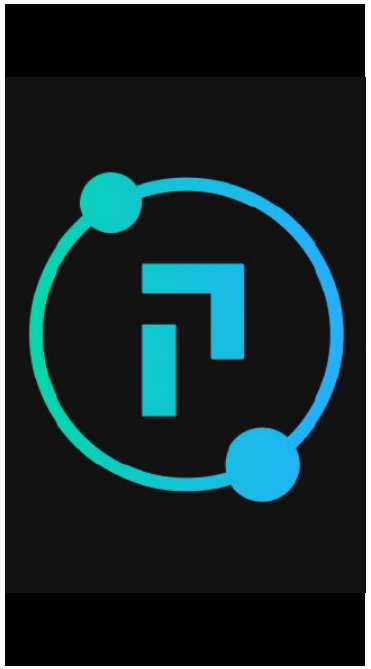
The bcc toolkit provides various eBPF-based tools for system-level monitoring and troubleshooting.

Project Calico

Provides network security & policy for applications in container, virtual machine, and bare metal environments.

Tracee

Tracee uses eBPF for runtime security and forensics by tracing system events and activities.



Pixie

Pixie leverages eBPF for instant, seamless, and live visibility into Kubernetes environments.

Other Use Cases

- ## Detection products (EDR)

  Attach malicious eBPF programs to authorized kernel hooks or system events, enabling privilege escalation and code execution.

- ## Debugging and Tracing

  Capture low level activity in the kernel for troubleshooting and capturing precise behavior of applications with minimal overhead or hooking into the application. Can be used for reverse engineering of applications as well.

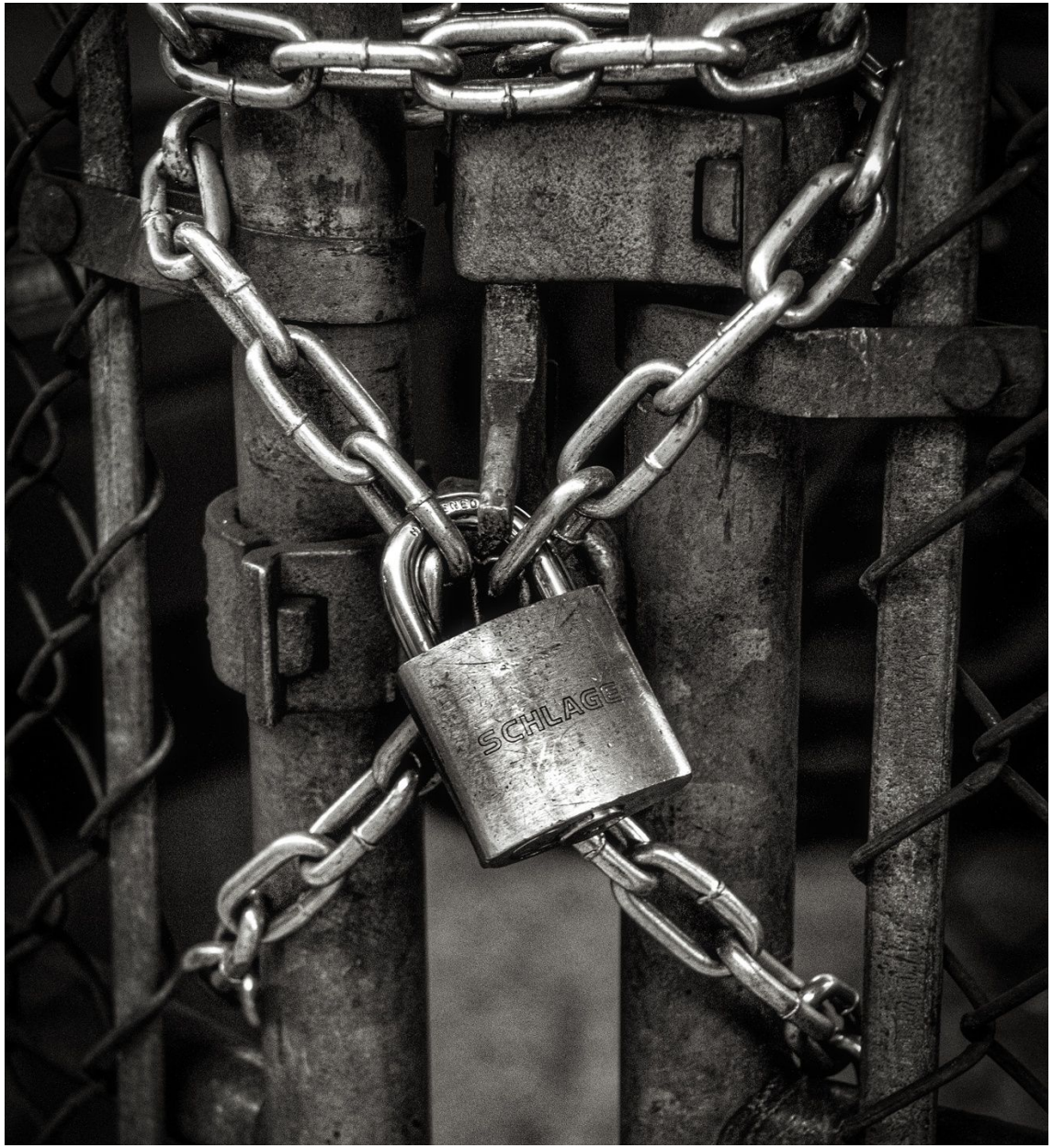
- ## Kernel Module replacement

  Easier to develop features on eBPF for use cases like Security, Observability and Networking. Still need kernel modules for device drivers and file systems.

- ## Live Kernel Patching

  Cloudflare demonstrated a way to life patch a vulnerability in a kernel using eBPF and Linux Security Modules.

“The power of eBPF is both a blessing and a curse; a potent tool capable of enhancing system security, yet equally adept at subverting it.”



Practical Examples and Live Demonstrations

Offensive and Defensive

Monitoring and Detection

```
# bpftrace -e 'tracepoint:syscalls:sys_enter_openat { printf("%s %s\n", comm, str(args.filename)); }'
Attaching 1 probe...
snmp-pass /proc/cpuinfo
snmp-pass /proc/stat
snmpd /proc/net/dev
snmpd /proc/net/if_inet6
^C
```

Detection (cont)

```
func (sig *K8SServiceAccountToken) GetMetadata() (detect.SignatureMetadata, error) {
        return detect.SignatureMetadata{
                ID:          "TRC-108",
                Version:     "1",
                Name:        "K8s service account token file read",
                EventName:   "k8s_service_account_token",
                Description: "The Kubernetes service account token file was read on your container. This
                Properties: map[string]interface{}{
                        "Severity":             0,
                        "Category":             "credential-access",
                        "Technique":            "Exploitation for Credential Access",
                        "Kubernetes_Technique": "Container service account",
                        "id":                   "attack-pattern--9c306d8d-cde7-4b4c-b6e8-d0bb16caca36",
                        "external_id":          "T1212",
                },
        }, nil
}

func (sig *K8SServiceAccountToken) GetSelectedEvents() ([]detect.SignatureEventSelector, error) {
        return []detect.SignatureEventSelector{
                {Source: "tracee", Name: "security_file_open", Origin: "container"},
        }, nil
}

func (sig *K8SServiceAccountToken) OnEvent(event protocol.Event) error {
        eventObj, ok := event.Payload.(trace.Event)
        if !ok {
                return fmt.Errorf("invalid event")
        }
```

Credit: https://github.com/aquasecurity/tracee/

Offensive Use Cases

- ## Root Kit

  Load a malicious kernel module containing eBPF bytecode to hijack system calls or install rootkits.

- ## Abusing eBPF Attach Points

  Attach malicious eBPF programs intercept or manipulate network traffic, system logging, hide malware from other processes. Capture user input such as keylogging.

- ## Exploiting Kernel Vulnerabilities

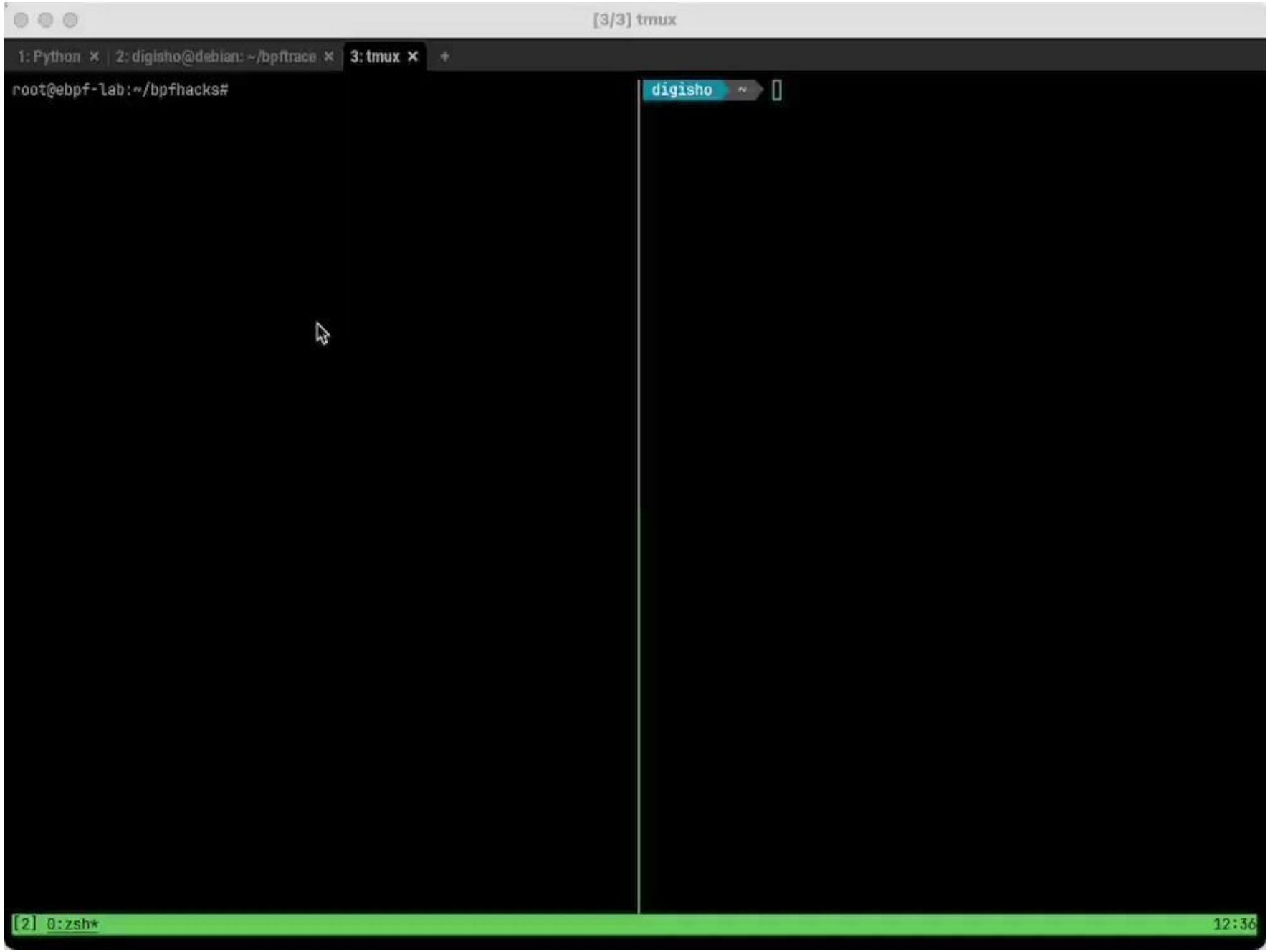
  Leverage kernel vulnerabilities or memory corruption bugs to inject eBPF code into the kernel space and gain elevated privileges.

- ## Userspace eBPF Program Injection

  Inject eBPF programs into userspace processes or containers, potentially bypassing security controls or monitoring mechanisms.

- ## Persistence Mechanisms

  Establish persistence by modifying system startup scripts, kernel modules, or other mechanisms to ensure malicious eBPF code is loaded during system boot.


[3/3] tmux
1: Python
2: digisho@debian: ~/bpftrace
3: tmux
root@ebpf-lab:~/bpfhacks#
digisho ~
[2] 0:zsh*
12:36

Finding eBPF programs

Explain what eBPF (extended Berkeley Packet Filter) is and how it allows attaching custom programs to various kernel hooks.

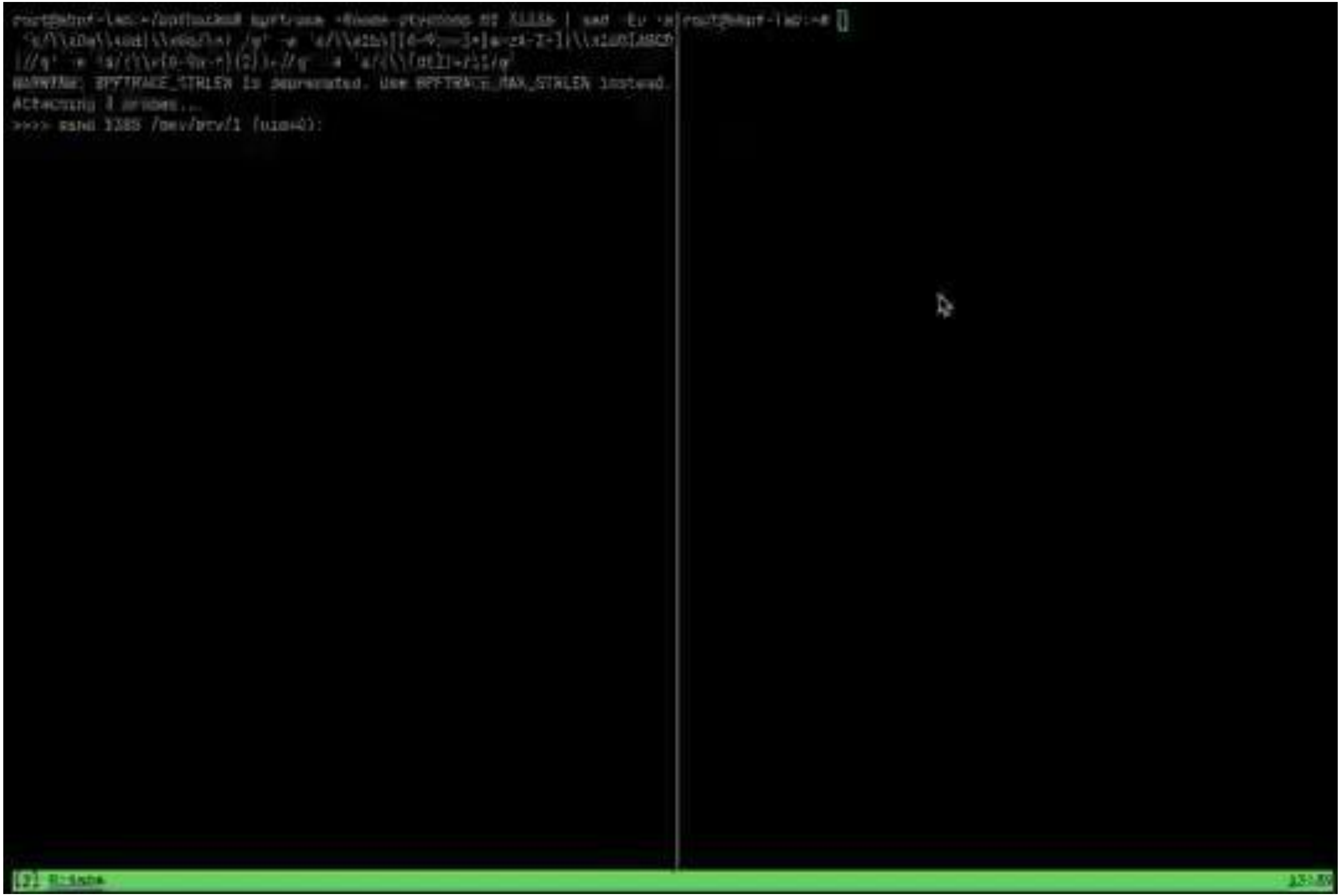
Highlight how eBPF programs can interact with the Linux kernel at a deep level, enabling the creation of powerful rootkits.

Describe how eBPF rootkits can execute malicious code stealthily, evading traditional detection methods.

Explain how eBPF rootkits can hook into kernel functions and intercept system calls, allowing them to hide processes, files, and network connections.

Discuss techniques used by eBPF rootkits to persist across reboots and maintain their presence on the compromised system.

Outline the challenges in detecting and removing eBPF rootkits, and present potential mitigation strategies.


WARNING: BPFTRACE_STRLEN is deprecated. Use BPFTRACE_MAX_STRLEN instead.
Attaching 1 probes...

Advanced Detection Mechanisms

Monitoring eBPF programs

Due to the powerful nature of eBPF, it's crucial to monitor the eBPF programs loaded into the kernel for potential security threats.

Detecting malicious eBPF code

Advanced detection mechanisms should be in place to identify and mitigate any malicious eBPF code that could compromise system security.

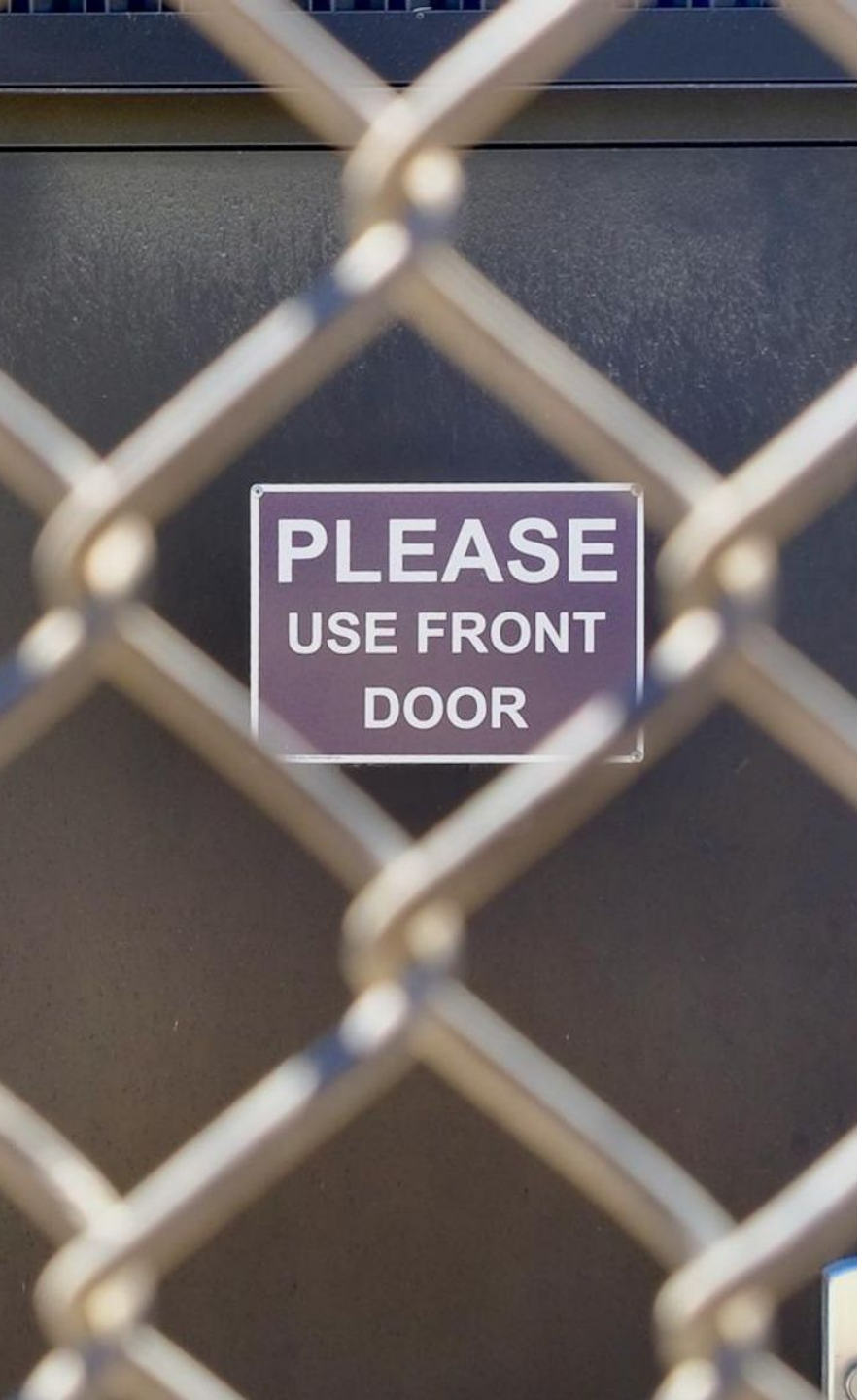
Real-time analysis

Real-time analysis of eBPF programs is essential to quickly detect and respond to potential threats as they emerge.

Disabling eBPF functionality

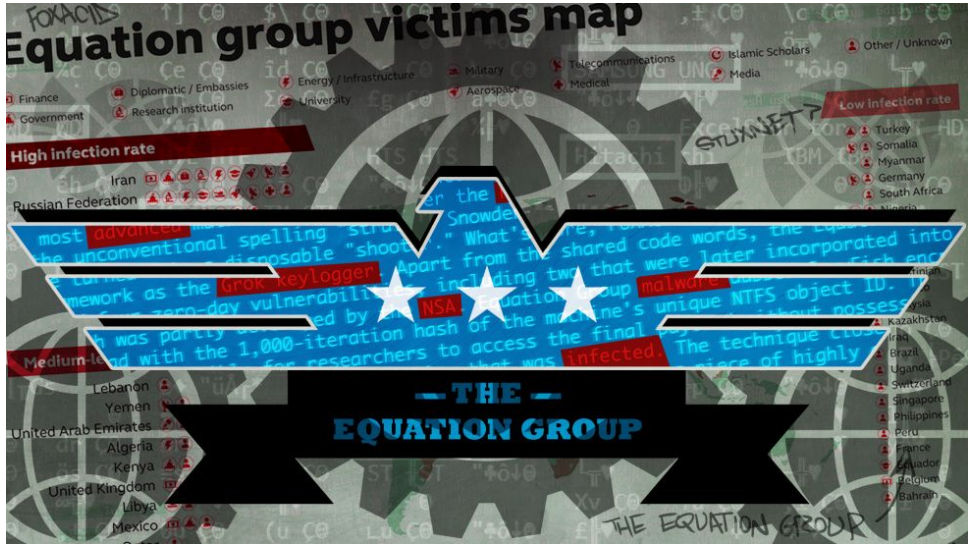
Disable or restrict where eBPF programs can be run. Unprivileged users should be disabled from using eBPF programs.

Advanced detection mechanisms are crucial for ensuring the secure and responsible use of eBPF, enabling organizations to harness its power while mitigating potential risks.



Bvp47

Bvp47 is a highly sophisticated backdoor malware attributed to the Equation Group (likely NSA). This backdoor was discovered by Pangu Lab during a forensic investigation in 2013 and submitted to VirusTotal. Remained undetected for nearly a decade. (>287 Targets, 45 Nations)



Bvp47

SYN Knock
(port 1357)

SYN Response
(port 2468)

Linux rejects/ignores SYN packet because port not open or contains non TCP spec fields, treats as bad packet, nothing is logged

eBPF program

Monitors:

SYN packets to port 1357

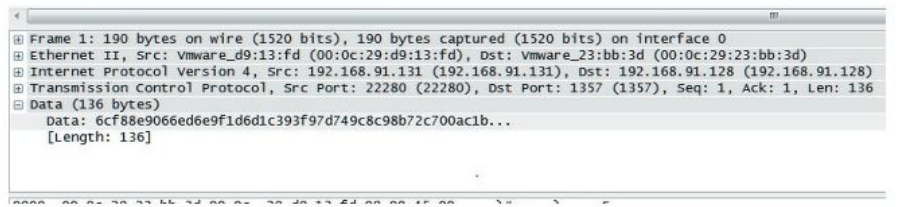
Reads:

Non-TCP spec data field in SYN packet

Decrypt data

Execute Instruction

Send Response



Details

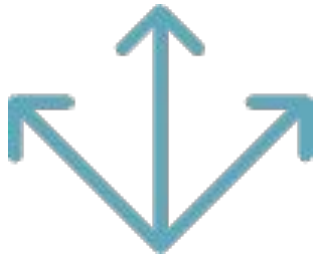
More Examples

PoC Rootkits

Boopkit

Developed to work similarly to Bvp47

https://github.com/krisnova/boopkit

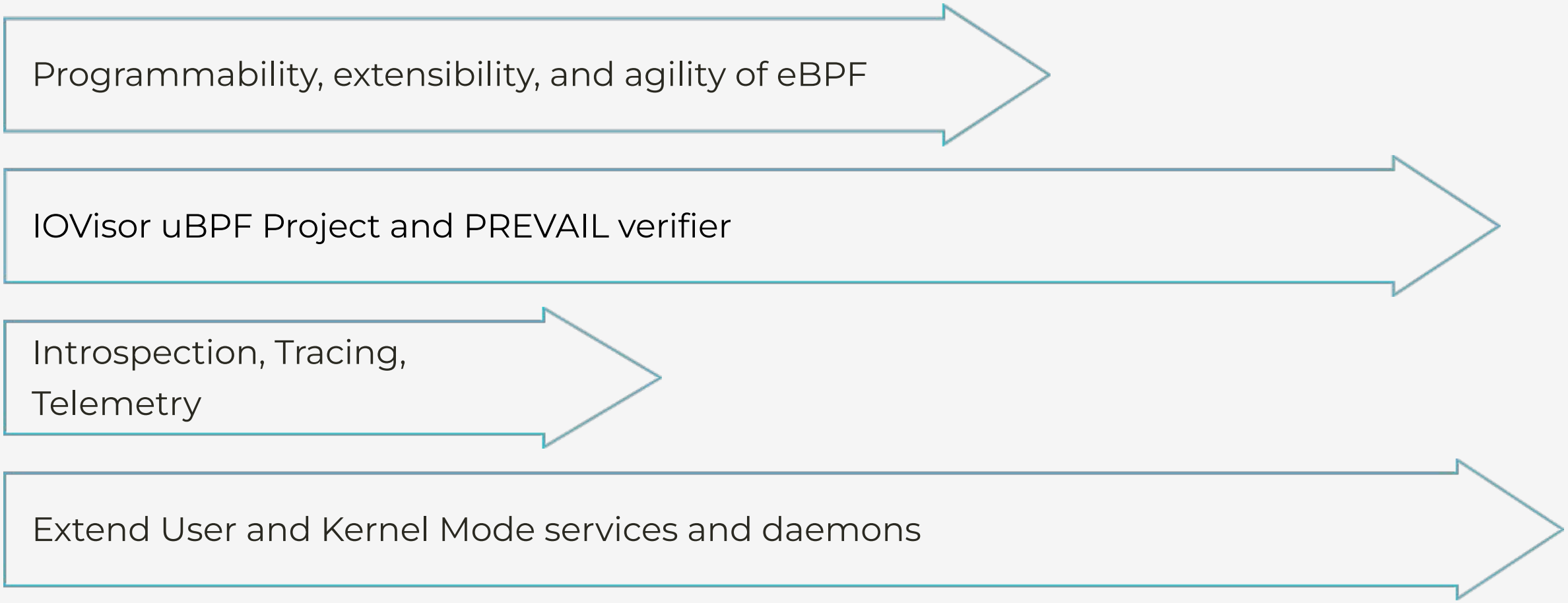


TripleCross

An amazing undergraduate thesis project.

https://github.com/h3xduck/TripleCross

eBPF for Windows



Conclusion

eBPF's versatility as a security tool

eBPF can be leveraged to implement powerful security monitoring and prevention mechanisms.

Potential for abuse by malware authors

The same capabilities that make eBPF useful for security can be exploited by malware for persistence and stealth.

Ongoing evolution and research

The eBPF ecosystem continues to evolve, with new use cases and potential risks being discovered.

Understanding eBPF's dual nature as a security tool and potential malware vector is crucial for staying ahead of emerging threats and leveraging its capabilities responsibly.

Thank You

eBPF Resources

https://digital-shokunin.net/page/ebpf-resources/
or
https://shorturl.at/Fu5Bq