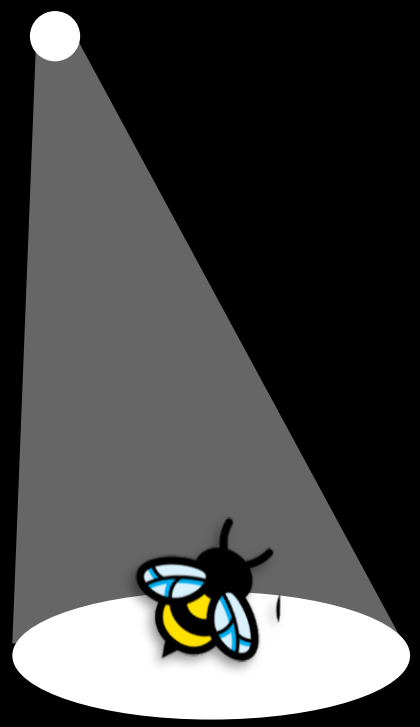
eBPFShield

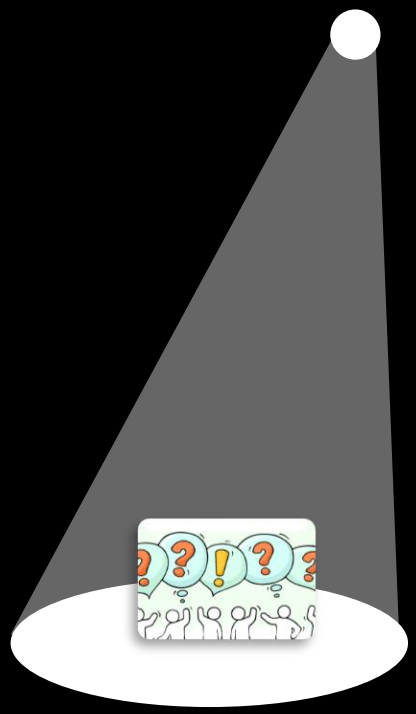
Advanced IP-Intelligence and DNS Monitoring using eBPF

Hack in Paris 2023



Overview eBPF

Coding eBPFShield



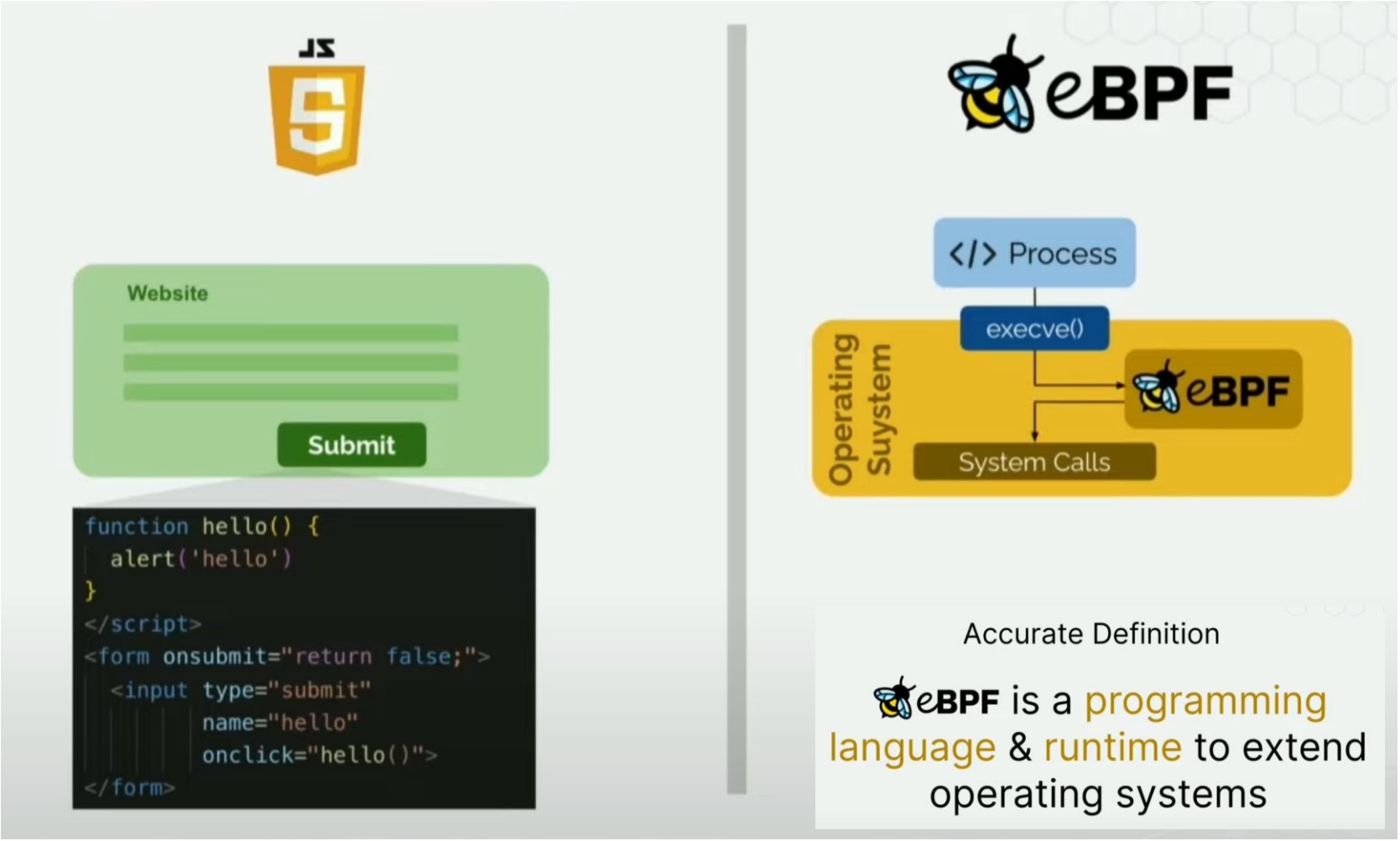
Q&A

$whoami

SAGAR BHURE

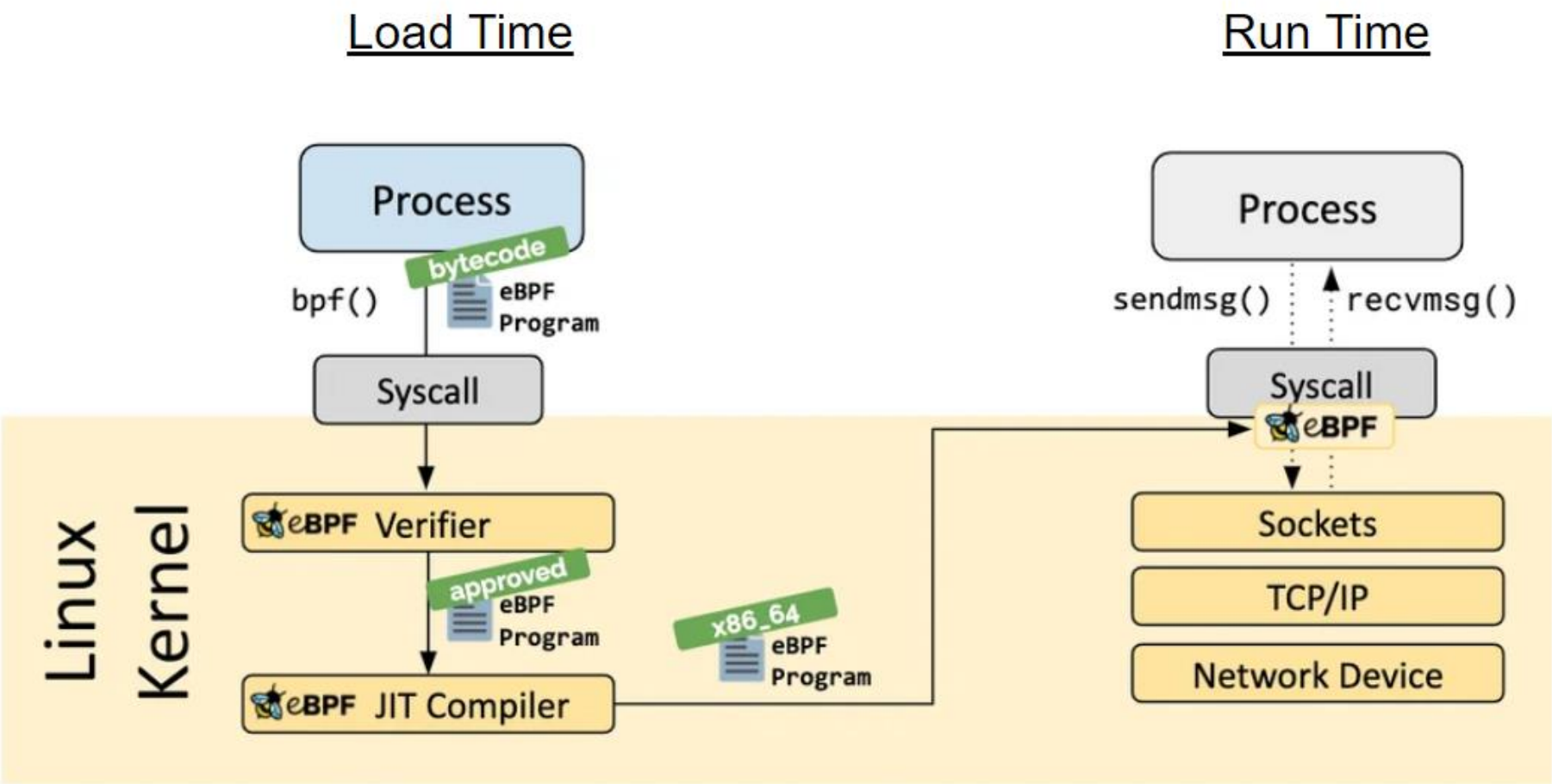
Software Engineer | OWASP Project Lead | Blackhat Speaker

Overview & Inspiration



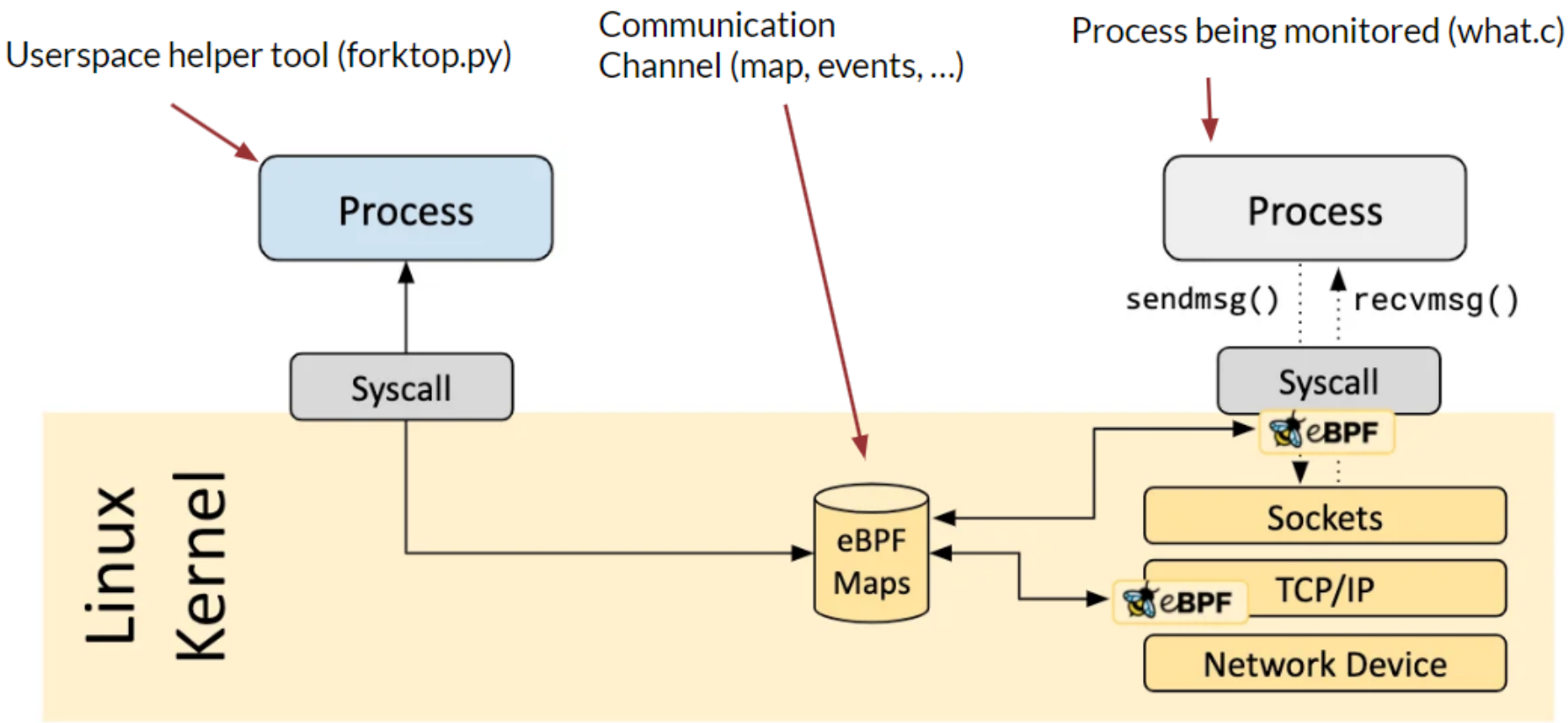
eBPF does to Linux, what JavaScript does to HTML

Overview - Architecture



ebpf.io/what-is-ebpf/

Overview - Architecture

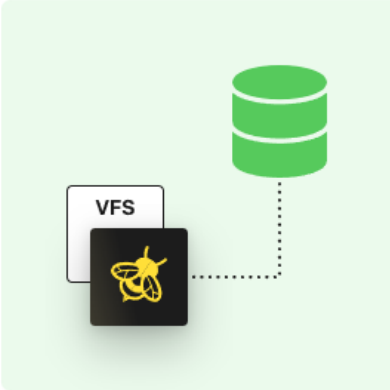


ebpf.io/what-is-ebpf/

What’s possible with eBPF

Networking

Speed packet processing without leaving kernel space. Add additional protocol parsers and easily program any forwarding logic to meet changing requirements.



Observability

Collection and in-kernel aggregation of custom metrics with generation of visibility events and data structures from a wide range of possible sources without having to export samples.

Tracing & Profiling

Attach eBPF programs to trace points as well as kernel and user application probe points giving powerful introspection abilities and unique insights to troubleshoot system performance problems.

Security

Combine seeing and understanding all system calls with a packet and socket-level view of all networking to create security systems operating on more context with a better level of control.

Workshop Overview

This tutorial focuses on developing BCC tools and programs using Python.

Part 1: Observability

- Understand system observability.
- Develop BCC tools for observability.

Part 2: Networking

- Explore networking capabilities.
- Create BCC networking tools.

BPF Programming Essentials
Lesson 1

eBPF: ‘Hello World’

https://github.com/sagarbhure/HIP-eBPF/blob/main/lesson1.py

BPF Programming Essentials
Lesson 2

Tracing ‘sys_sync()’ System Call with BPF

https://github.com/sagarbhure/HIP-eBPF/blob/main/lesson2.py

BPF Programming Essentials
Lesson 3

Tracing ‘sys_clone()’ System Call with BPF

https://github.com/sagarbhure/HIP-eBPF/blob/main/lesson3.py

BPF Programming Essentials
Lesson 4

Monitoring Multiple Synchronization Events via sys_sync

https://github.com/sagarbhure/HIP-eBPF/blob/main/lesson4.py

BPF Programming Essentials
Lesson 5

Tracing System Calls with BPF_PERF_OUTPUT in Python

hello_perf_output

https://github.com/sagarbhure/HIP-eBPF/blob/main/lesson5.py

DNS Monitoring

Decoding DNS Monitoring

- Monitors DNS queries in the system
- Helps detect and block DNS tunneling attempts in real-time
- Provides proactive defense against potential DNS-based attacks
- Prevents damage before it occurs by identifying and blocking malicious DNS queries

Enabling DNS Monitoring in eBPFShield

- Start eBPFShield in DNS monitoring mode with the following command:
  `python3 main.py --feature ebpf_monitor`

- Test DNS queries from a client by using the dig command with the following syntax:
  `dig @1.1.1.1 geekwire.com +tcp`

- eBPFShield will capture and display all DNS queries made from the client, providing visibility into DNS traffic for effective monitoring and detection of potential DNS tunneling attempts.

DNS Monitoring

```
root@host-virtualfair:~# dig @1.1.1.1 google.com +tcp +short
142.250.183.174
root@host-virtualfair:~# dig @1.1.1.1 geekwire.com +tcp

; <<>> DiG 9.16.1-Ubuntu <<>> @1.1.1.1 geekwire.com +tcp
; (1 server found)
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 59677
;; flags: qr rd ra ad; QUERY: 1, ANSWER: 3, AUTHORITY: 0, ADDITIONAL: 1

;; OPT PSEUDOSECTION:
; EDNS: version: 0, flags:; udp: 1232
;; QUESTION SECTION:
;geekwire.com.                  IN      A

;; ANSWER SECTION:
geekwire.com.           251     IN      A       104.26.14.176
geekwire.com.           251     IN      A       172.67.69.185
geekwire.com.           251     IN      A       104.26.15.176

;; Query time: 32 msec
;; SERVER: 1.1.1.1#53(1.1.1.1)
;; WHEN: Sun Jan 15 17:53:24 UTC 2023
;; MSG SIZE  rcvd: 89

root@host-virtualfair:~#
```

```
root@host-virtualfair:~/arsenal/eBPFShield# python3 main.py --feature ebpf_monitor
The program is running. Press Ctrl-C to abort.
COMM=dig PID=140898 TGID=140899 DEV=ens3 PROTO=TCP SRC=10.218.20.37 DST=1.1.1.1 SPT=44695 DPT=53 UID=0 GID=0 DNS_QR=0 DNS_NAME=google.com. D
NS_TYPE=A
COMM=dig PID=140898 TGID=140899 DEV=ens3 PROTO=TCP SRC=1.1.1.1 DST=10.218.20.37 SPT=53 DPT=44695 UID=0 GID=0 DNS_QR=1 DNS_NAME=google.com. D
NS_TYPE=A DNS_DATA=142.250.183.174
COMM=dig PID=140902 TGID=140903 DEV=ens3 PROTO=TCP SRC=10.218.20.37 DST=1.1.1.1 SPT=41777 DPT=53 UID=0 GID=0 DNS_QR=0 DNS_NAME=geekwire.com.
 DNS_TYPE=A
COMM=dig PID=140902 TGID=140903 DEV=ens3 PROTO=TCP SRC=1.1.1.1 DST=10.218.20.37 SPT=53 DPT=41777 UID=0 GID=0 DNS_QR=1 DNS_NAME=geekwire.com.
 DNS_TYPE=A DNS_DATA=104.26.14.176
COMM=dig PID=140902 TGID=140903 DEV=ens3 PROTO=TCP SRC=1.1.1.1 DST=10.218.20.37 SPT=53 DPT=41777 UID=0 GID=0 DNS_QR=1 DNS_NAME=geekwire.com.
 DNS_TYPE=A DNS_DATA=172.67.69.185
COMM=dig PID=140902 TGID=140903 DEV=ens3 PROTO=TCP SRC=1.1.1.1 DST=10.218.20.37 SPT=53 DPT=41777 UID=0 GID=0 DNS_QR=1 DNS_NAME=geekwire.com.
 DNS_TYPE=A DNS_DATA=104.26.15.176
```

IP Intelligence

Decoding IP Intelligence

- IP Intelligence module in eBPFshield blocks outgoing connections to blacklisted IPs.
- eBPFshield updates its threat feed list from sources like Talos Intelligence.
- Regular updates of the threat feed list can be scheduled with the update_feed_list script.
- Real-world use cases demonstrate the effectiveness of IP intelligence in preventing access to malicious destinations.

Enabling IP-Intelligence in eBPFShield

- Update the threat feed list with the command: **`./update_feeds.sh`**
- Start the process with IP Intelligence feature and blocking action: **`python3 main.py --feature ebpf_ipintelligence --block kill`**
- Supported actions with the "`--block`" flag: print (**`default`**), **`suspend`**, **`kill`**, **`dump`**
- Actions like suspend, kill, and dump can be used to immediately stop potentially malicious behavior.

IP Intelligence

```
$ python3.7 main.py  --feature ebpf_ipintelligence --block kill
Namespace(block='kill', buffer_size=10, bytes_to_capture=100, cport=9000, delay=100,
feature='ebpf_ipintelligence', n_runs=10, qdisc='fq', rate=8, run_scenario='just_one_
flow', store_pcaps=False, time=10, verbose=False)

The program is running. Press Ctrl-C to abort.

Client:b'curl' (pid:390539) was killed by eBPFShield (ip-blacklist:31.3.230.31)

Client:b'curl' (pid:390540) was killed by eBPFShield (ip-blacklist:185.242.113.224)
```

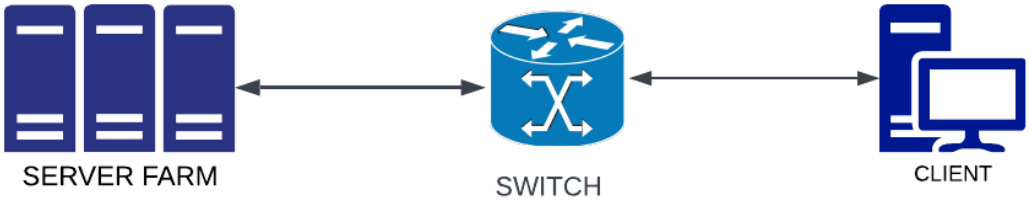
```
$ curl -v 31.3.230.31
*   Trying 31.3.230.31:80...
* TCP_NODELAY set
Killed
$
$ curl -v 185.242.113.224
*   Trying 185.242.113.224:80...
* TCP_NODELAY set
Killed
$
$ curl google.com
<HTML><HEAD><meta http-equiv="content-type" content="text/html;charset=u
tf-8">
<TITLE>301 Moved</TITLE></HEAD><BODY>
<H1>301 Moved</H1>
The document has moved
<A HREF="http://www.google.com/">here</A>.
</BODY></HTML>
$
```

Machine Learning

Decoding and Usage eBPFShield ML

- **Code Compilation**: Wrapper code compiled for userspace and kernelspace with alternative data structures.
- **Decision Engine**: Python module evaluates packets to detect malicious activity in real-time.
- **Experimental Test**: Emulated network with Linux namespaces, showing 10% improvement in analyzed packets.
- **Future Directions**: Explore performance of other machine learning models in eBPF implementation.

```
Run in userspace
- g++ -DUSERSPACE -fpermissive -I/usr/include/bcc ebpf_wrapper.cc -lbcc -o ebpf_wrapper
Run in kernel space
- g++ -fpermissive -I/usr/include/bcc ebpf_wrapper.cc -lbcc -o ebpf_wrapper
```



eBPF CTF