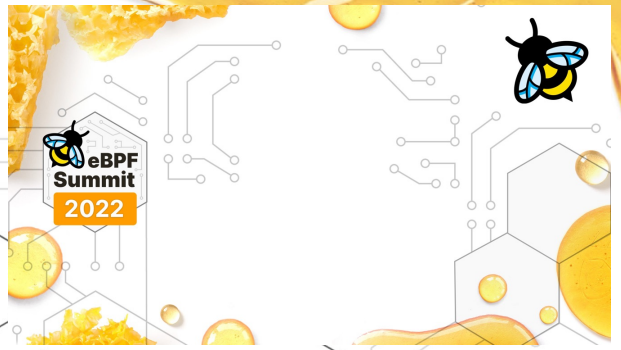
ANALYSIS OF OFFENSIVE CAPABILITIES OF eBPF AND IMPLEMENTATION OF A ROOTKIT

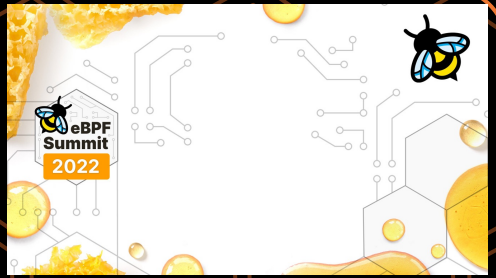


github.com/h3xduck/TripleCross

Marcos Bajo – Juan Tapiador

@h3xduck
@0xjet

TABLE OF CONTENTS



01

INTRODUCTION

- Rootkits
- Ebpf rootkits
- Project objectives

02

OFFENSIVE eBPF

- Tracing programs
- Memory corruption
- Network programs

03

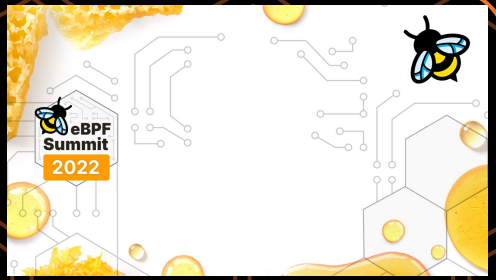
ROOTKIT DESIGN

- Library injection
- Privilege escalation
- Execution hijacking
- Backdoor and C2
- Rootkit persistence
- Rootkit stealth

04

DEFENCE AGAINST THE DARK ARTS

- Defence techniques
- Final remarks

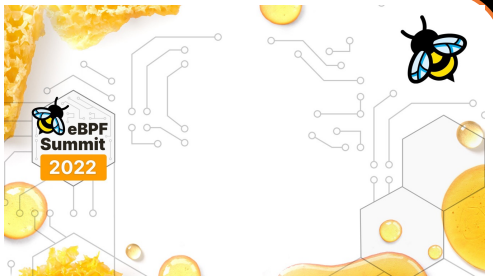


01

INTRODUCTION

- Rootkits
- Ebpf
- Previous research
- Project objectives
- Project benefits

PROJECT GOALS



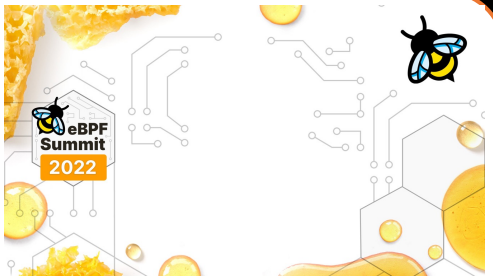
LINUX KERNEL MODULES (LKMs)

- Usually forbidden at critical systems
- **NOT** secure
- Can be used for building rootkits

eBPF

- Usually available by default
- *Is it secure?*
- *Could it be used for building rootkits?*

ROOTKITS



STEALTH

Hide files and activity from user

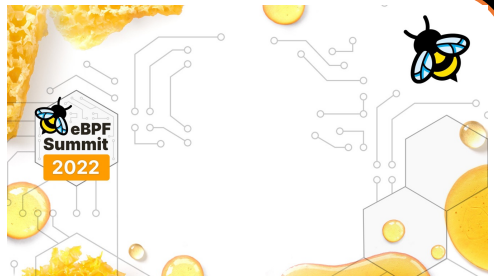
Avoid monitoring software

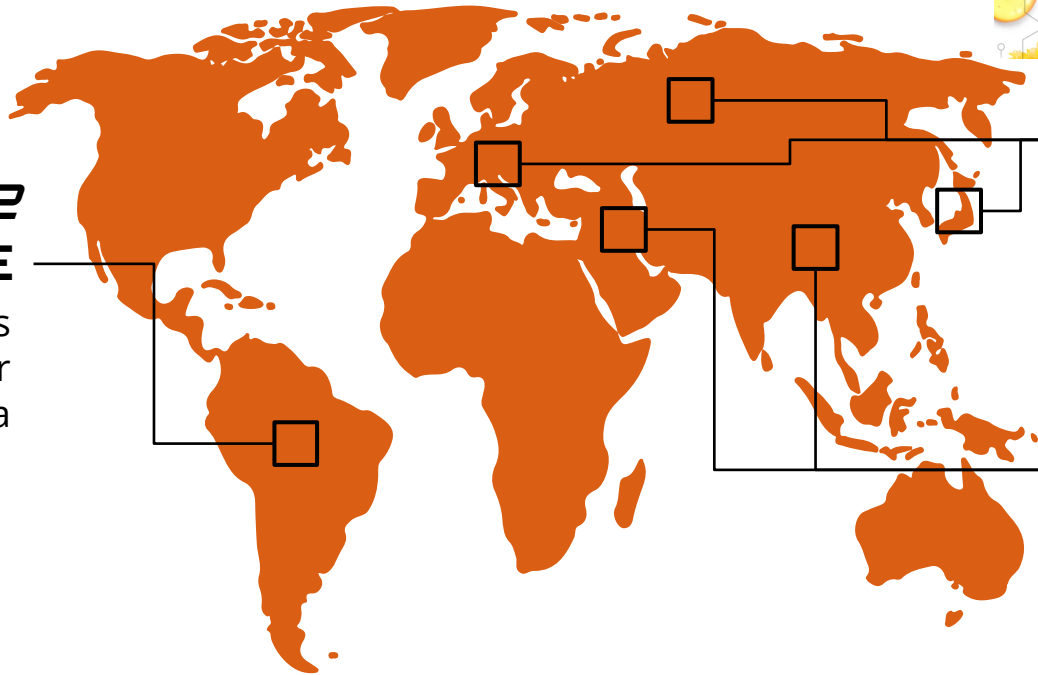
BACKDOOR

Access through the network

Remote Command and Control (C2)

BPF/eBPF ROOTKITS ARE ALREADY HERE

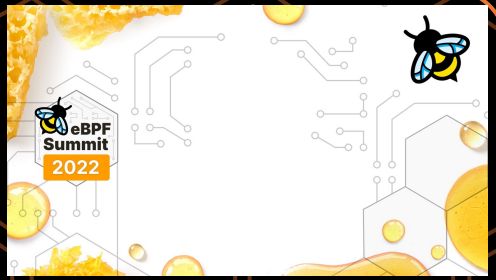




2022
SYMBIOTE
Implant targets financial sector in Latin America

2022
BVP47
NSA backdoor targets military and telecommunication systems

2021
BPFDoor
China-based actor targets telecommunication systems

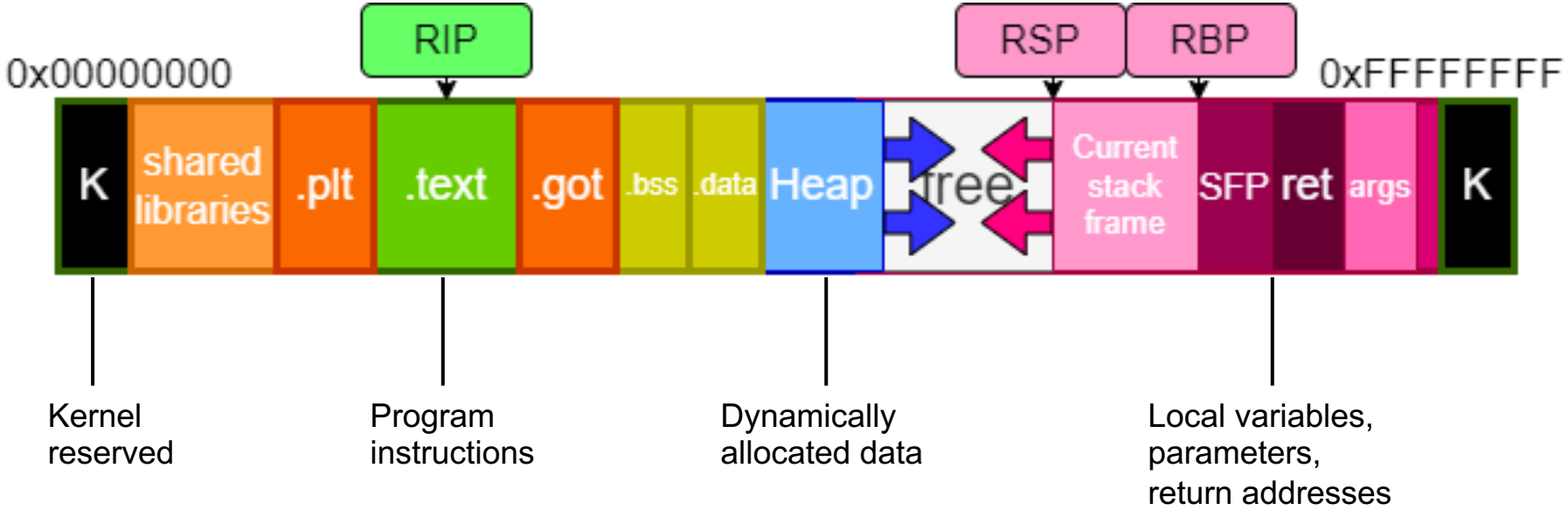


02

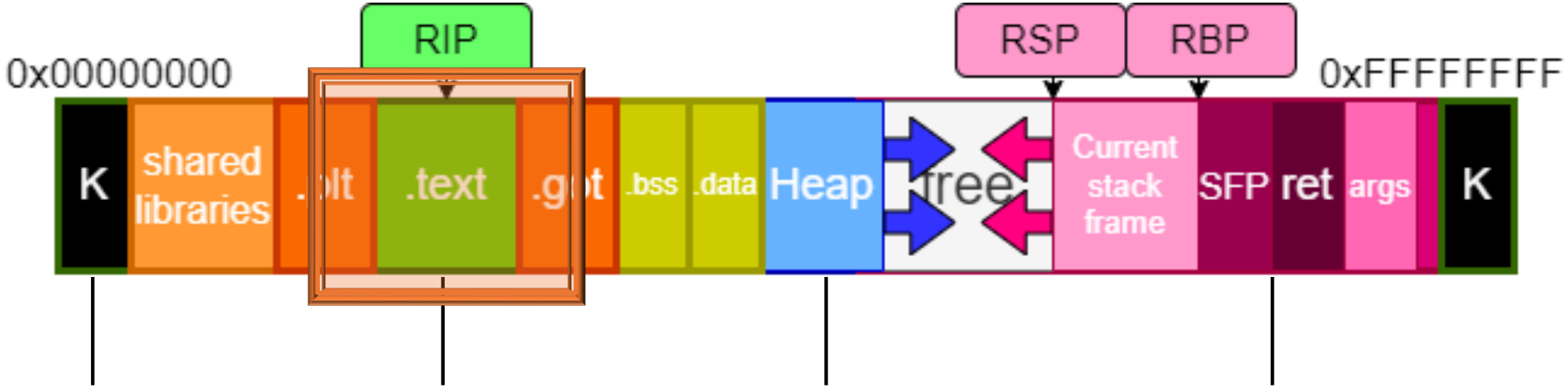
OFFENSIVE eBPF

- Tracing programs
- Memory corruption
- Network programs

VIRTUAL MEMORY



VIRTUAL MEMORY



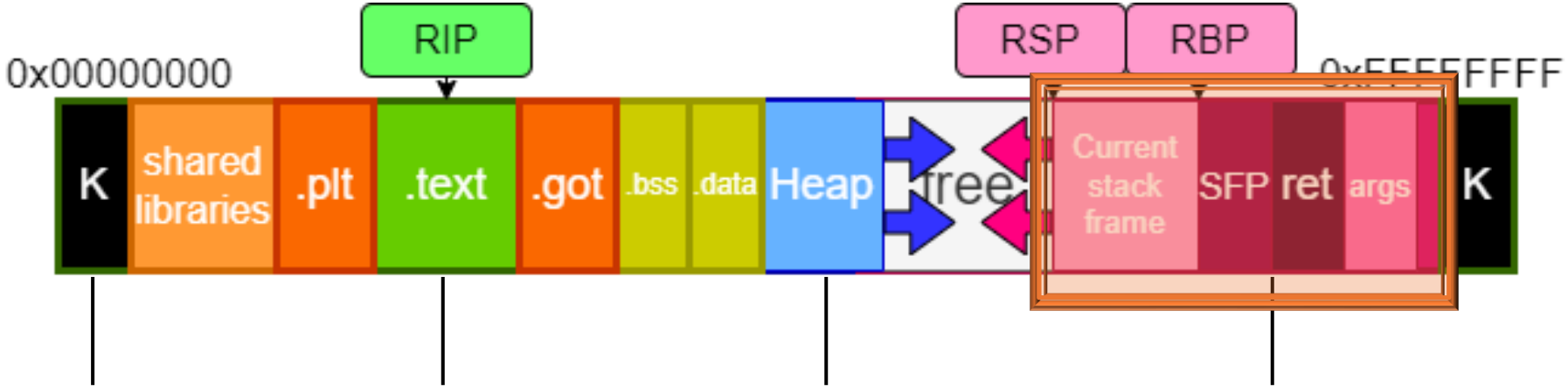
Kernel reserved

Program instructions

Dynamically allocated data

Local variables, parameters, return addresses

VIRTUAL MEMORY



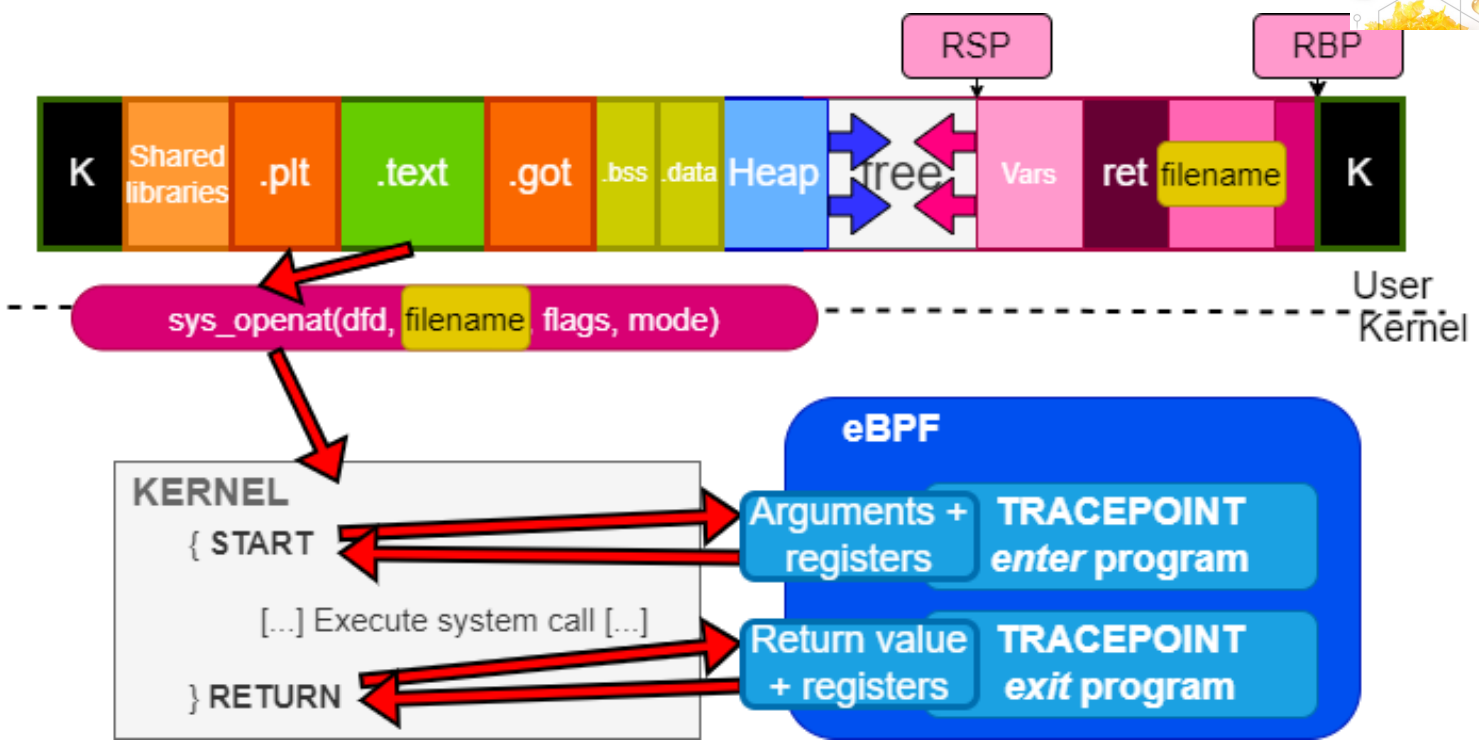
Kernel reserved

Program instructions

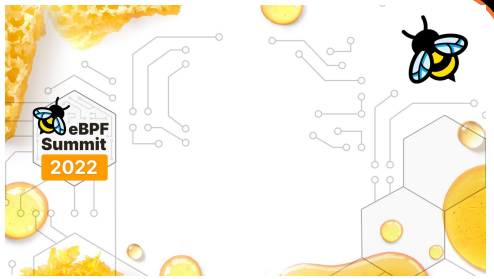
Dynamically allocated data

Local variables, parameters, return addresses

eBPF TRACING



READ ONLY ACCESS



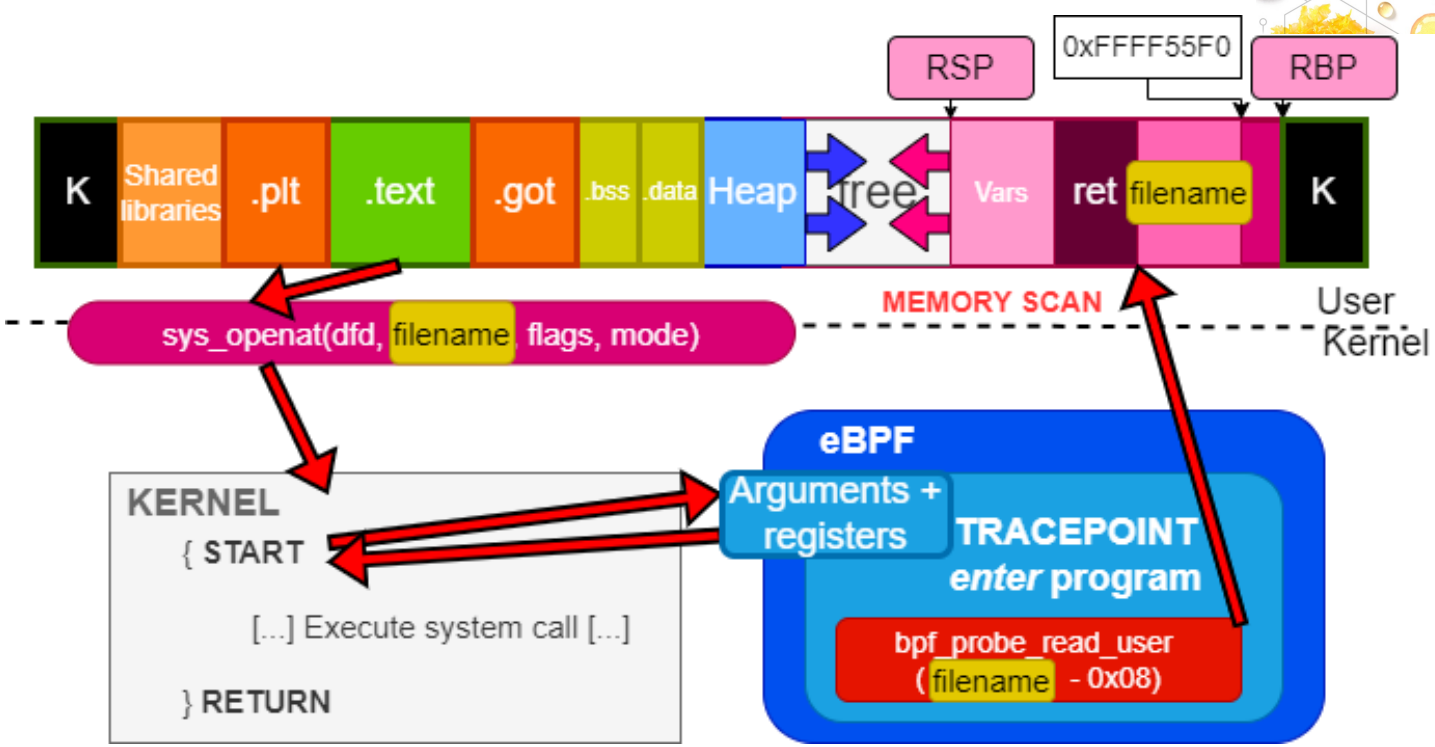
bpf_probe_read_user()

Read data at the user space

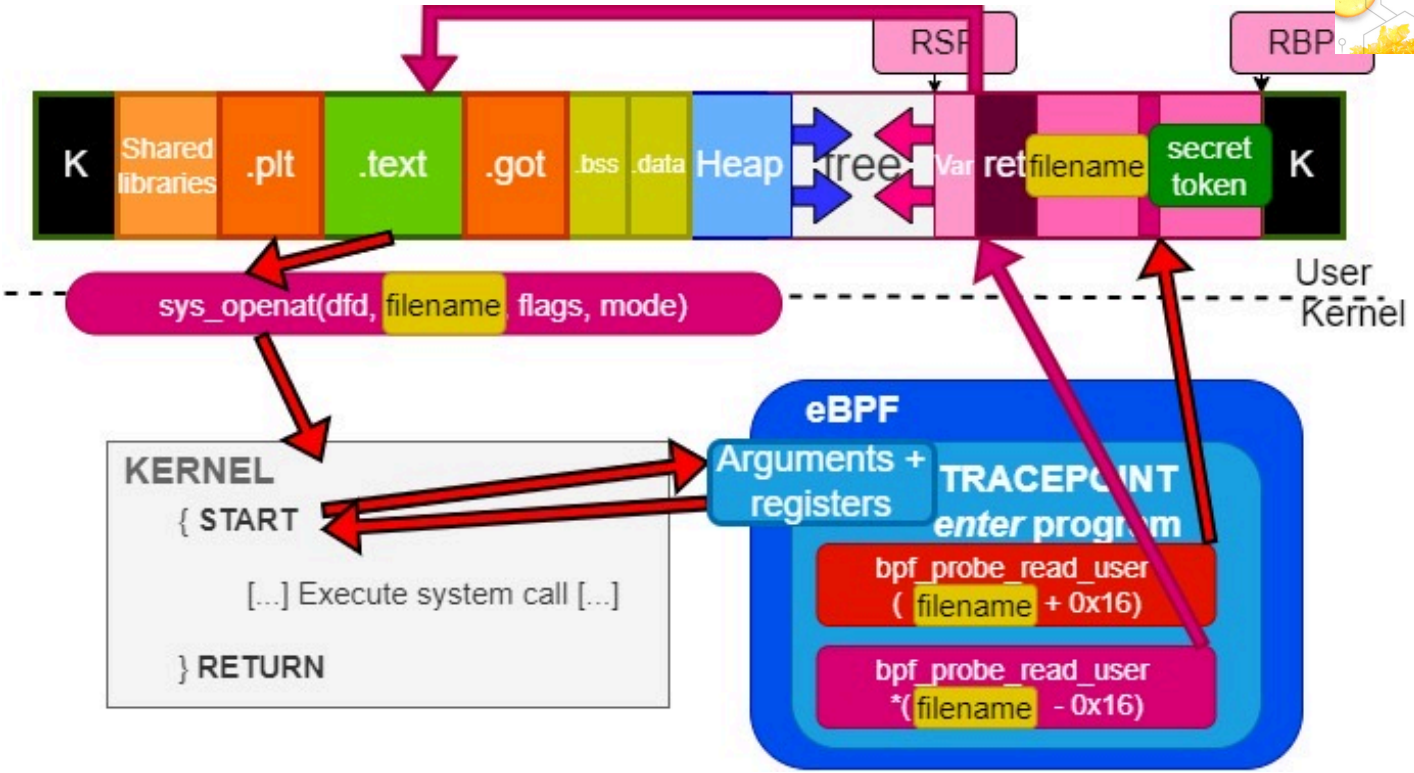
bpf_probe_read_kernel()

Read data at the kernel space

OUT OF BOUNDS READ



OUT OF BOUNDS READ

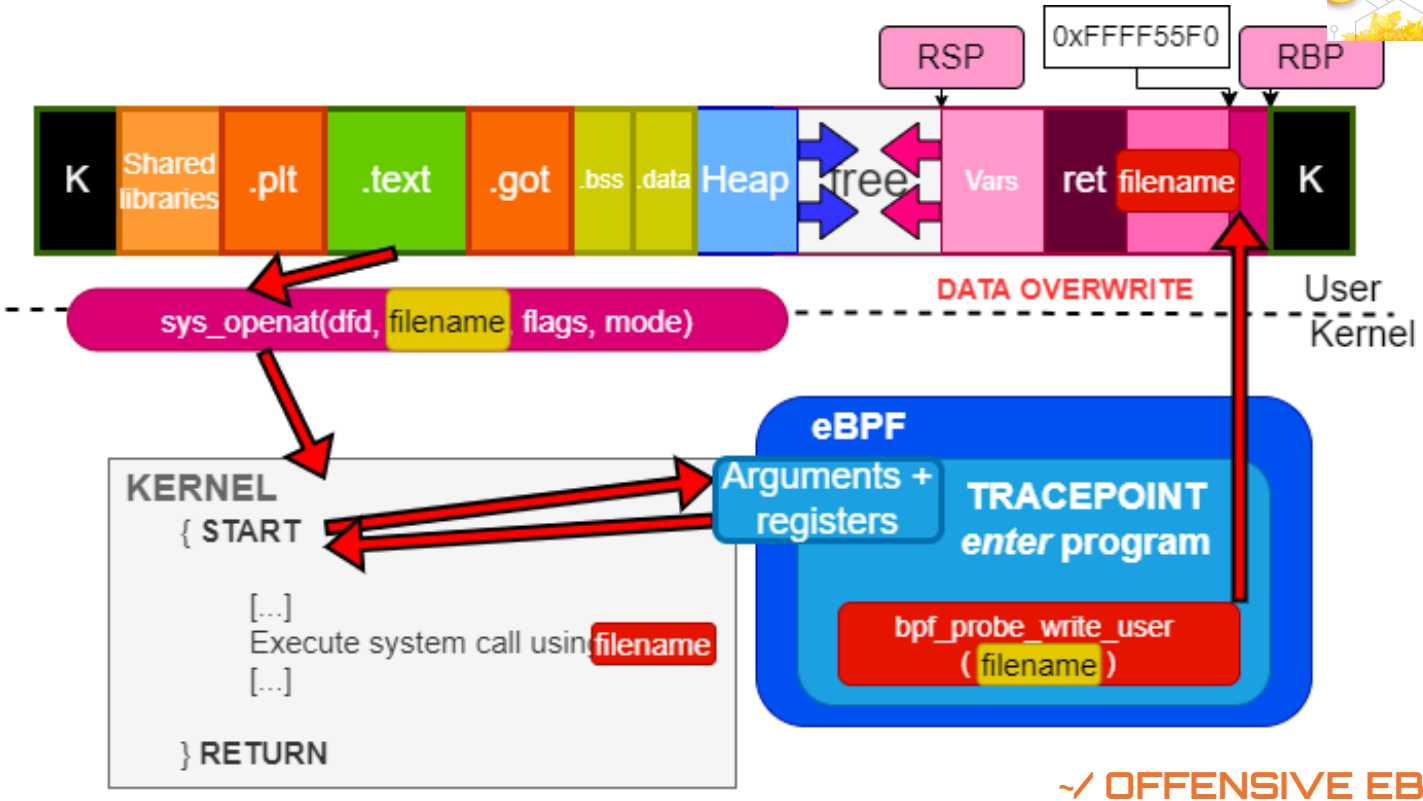


USER SPACE WRITING

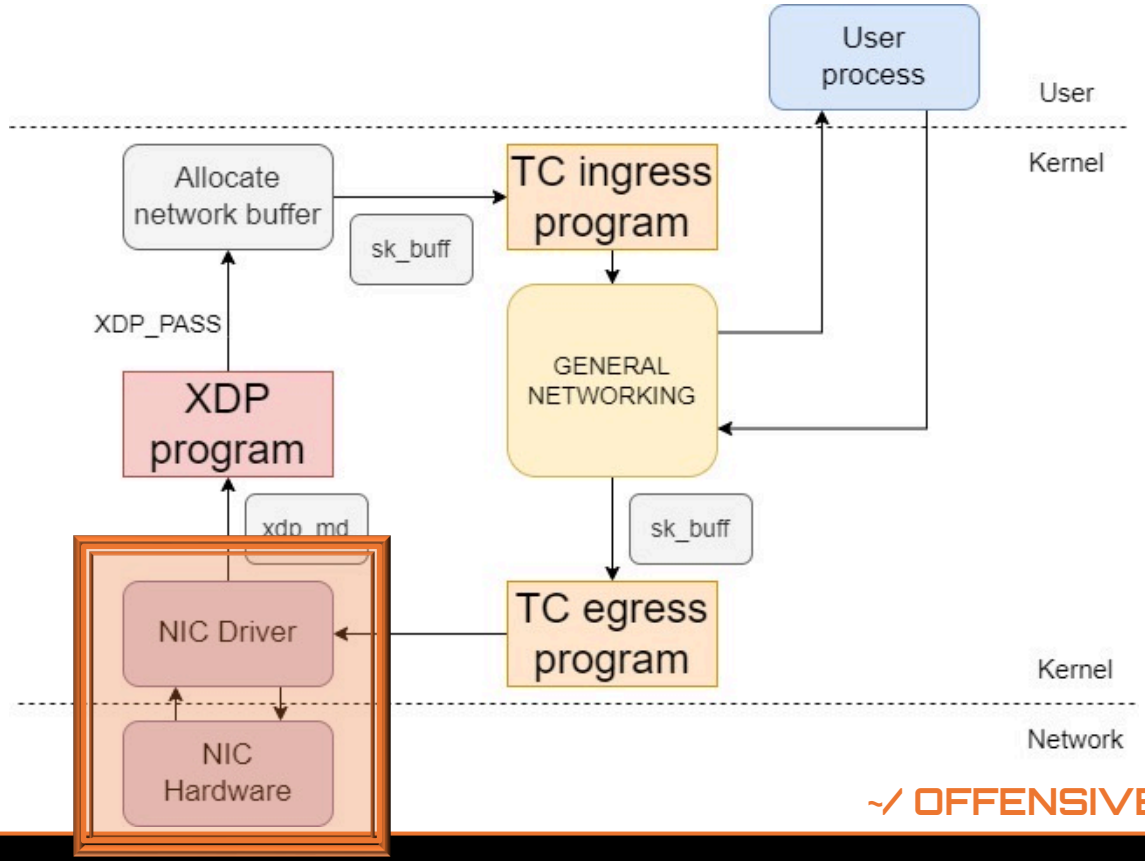
bpf_probe_write_user()

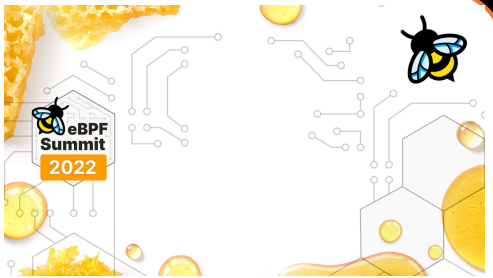
Write data at the user space
Memory must be writeable

MEMORY CORRUPTION



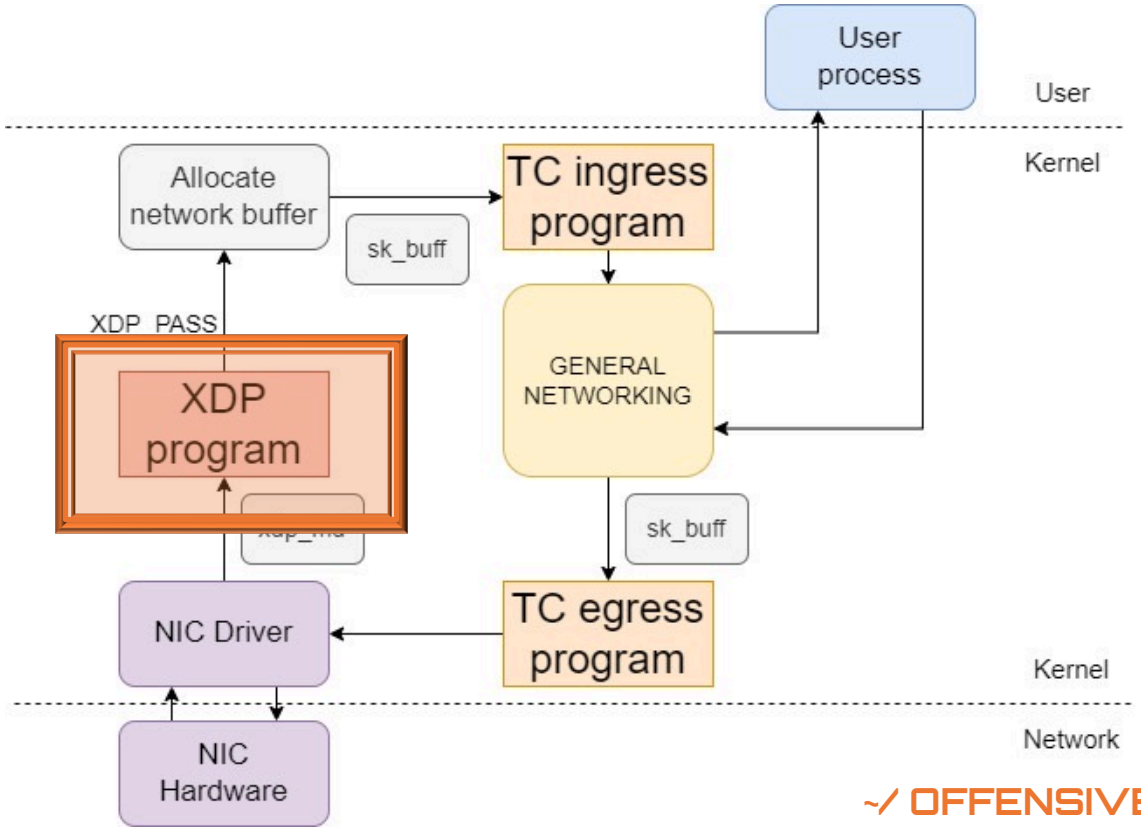
eBPF NETWORKING

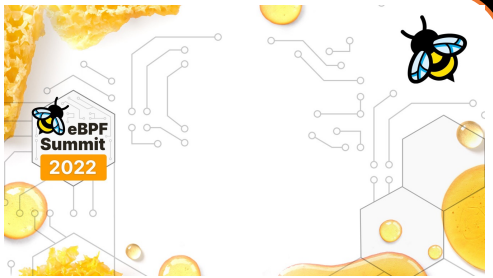




- **Dropping** packets.
- **Modifying** packets.
- **Cannot generate** new packets.

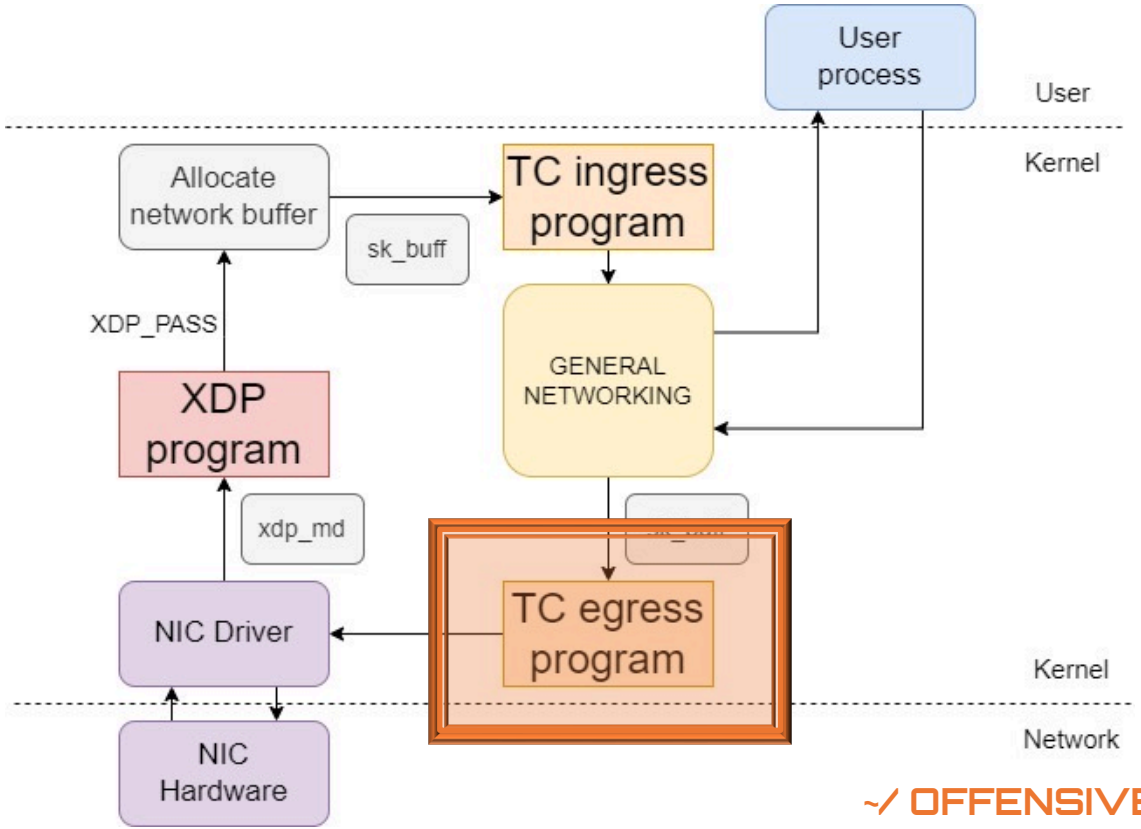
eBPF NETWORKING

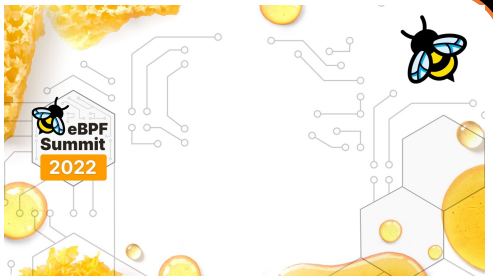




- **Dropping** packets.
- **Modifying** packets.
- **Cannot generate** new packets.

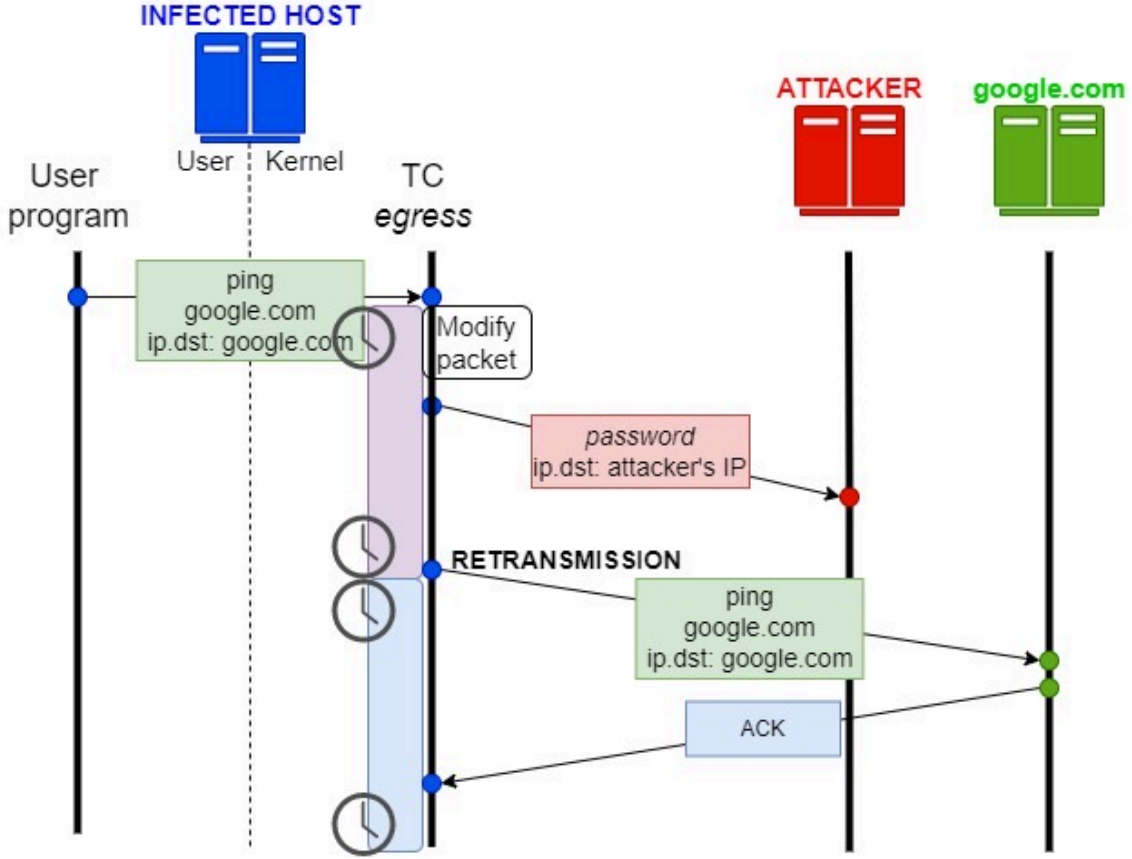
eBPF NETWORKING

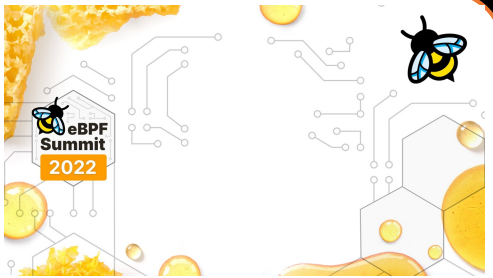




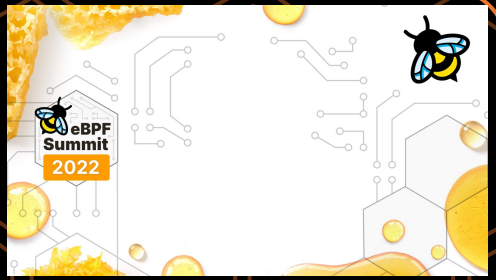
- **Dropping** packets.
- **Modifying** packets.
- **Cannot generate** new packets.

eBPF NETWORKING





- We can create **arbitrary new packets**

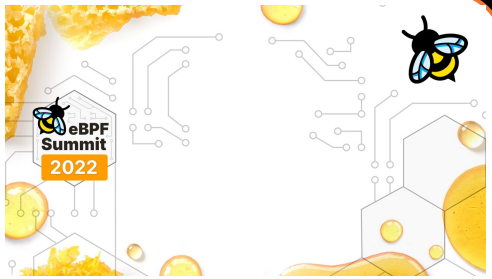


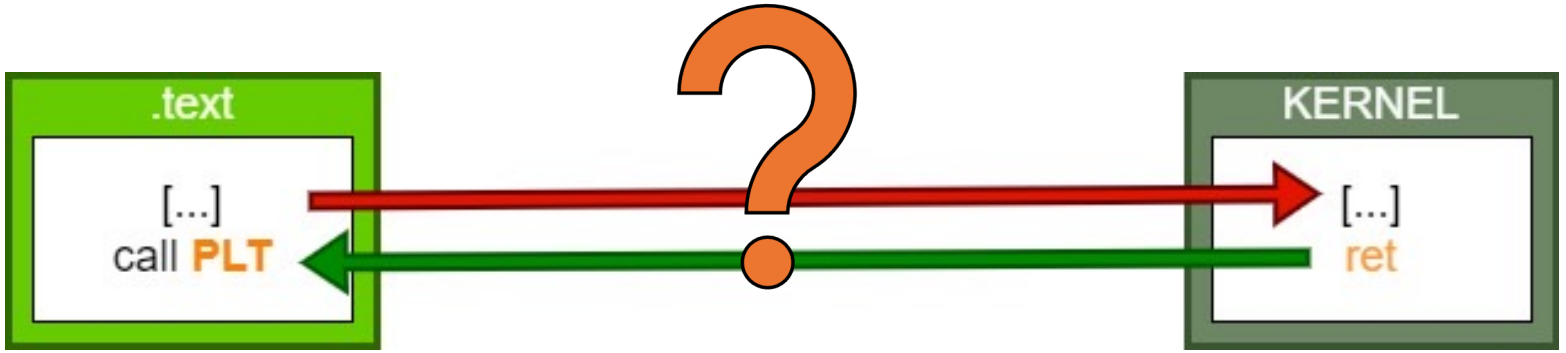
03

ROOTKIT DESIGN

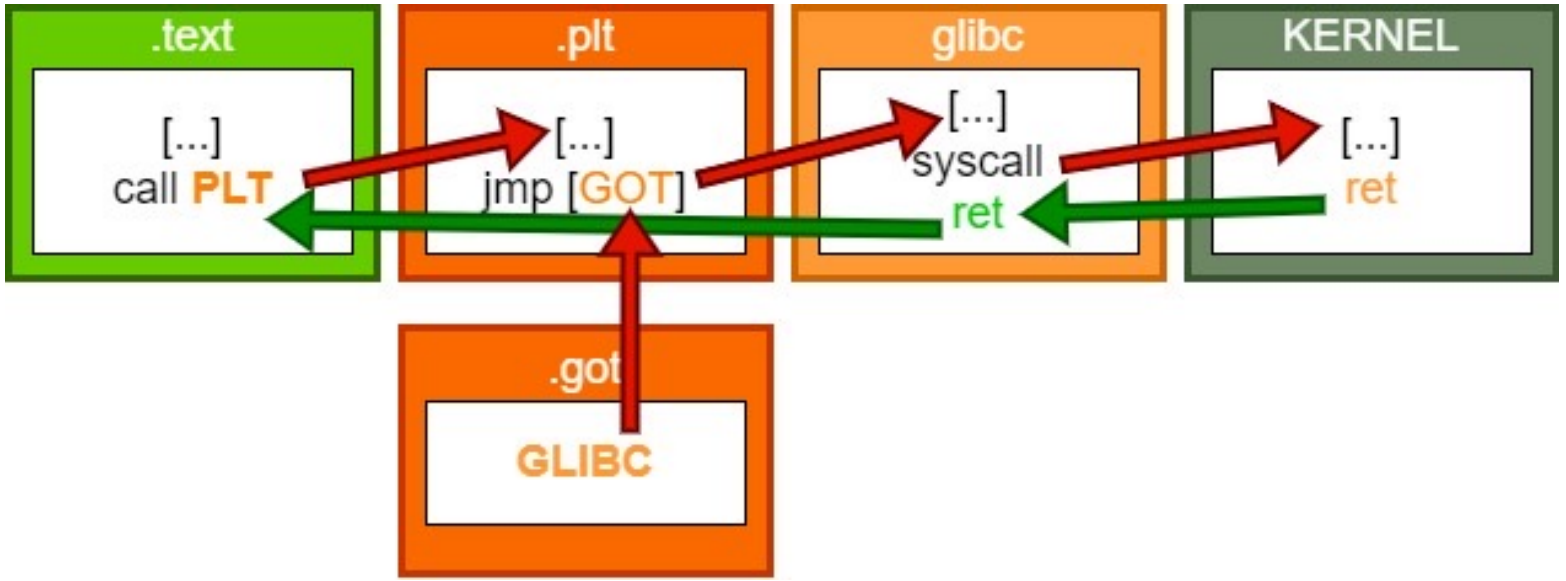
- Library injection
- Privilege escalation
- Execution hijacking
- Backdoor and C2
- Rootkit persistence
- Rootkit stealth

LIBRARY INJECTION

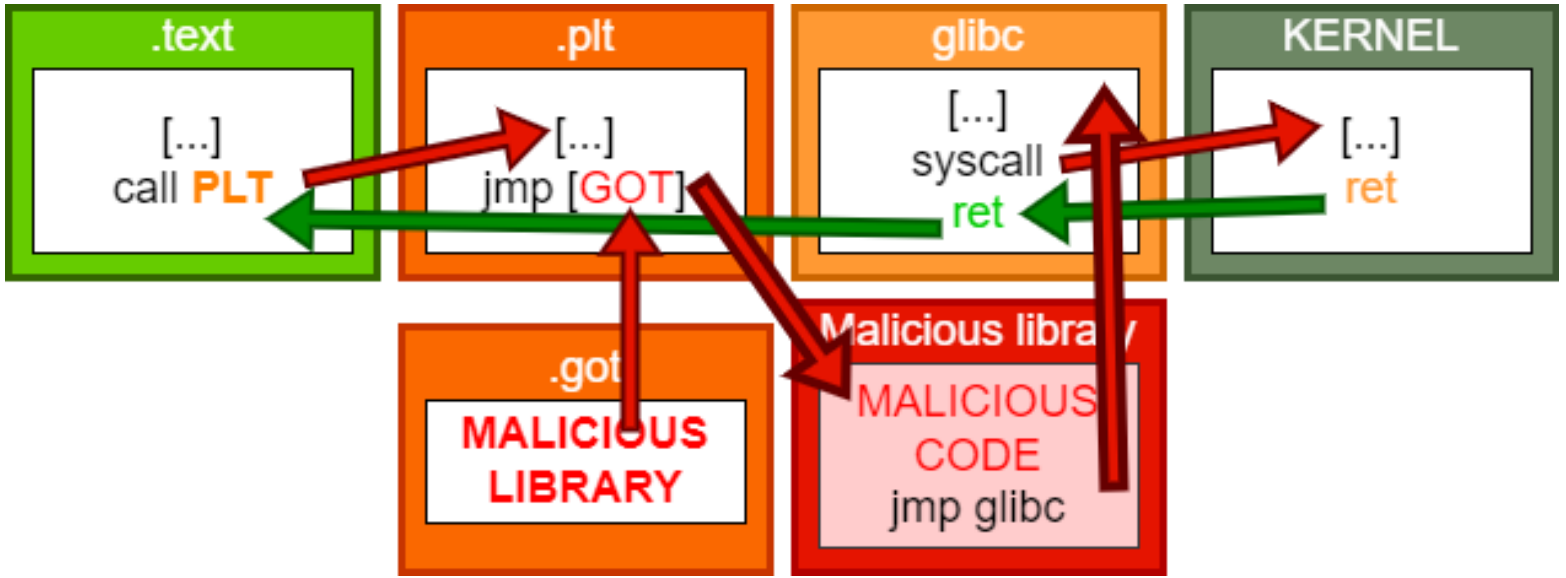




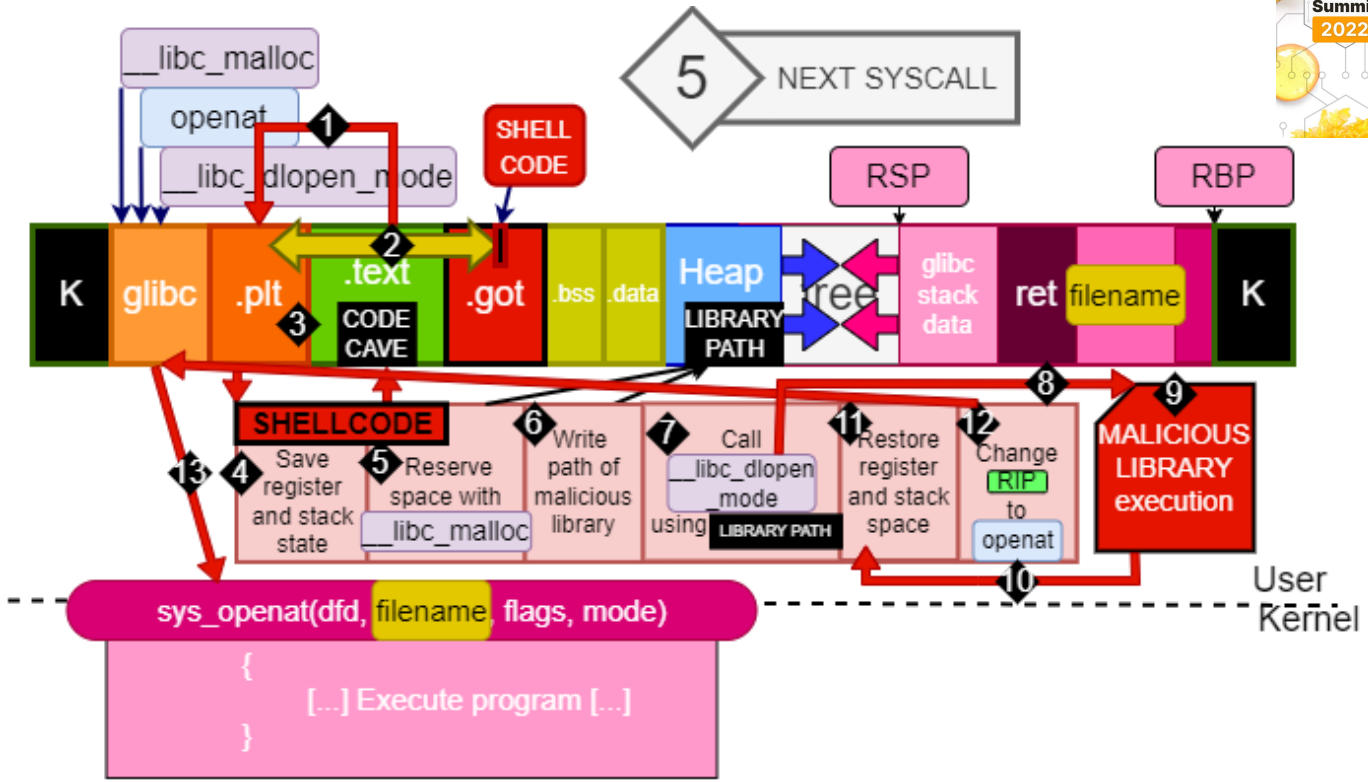
NORMAL EXECUTION



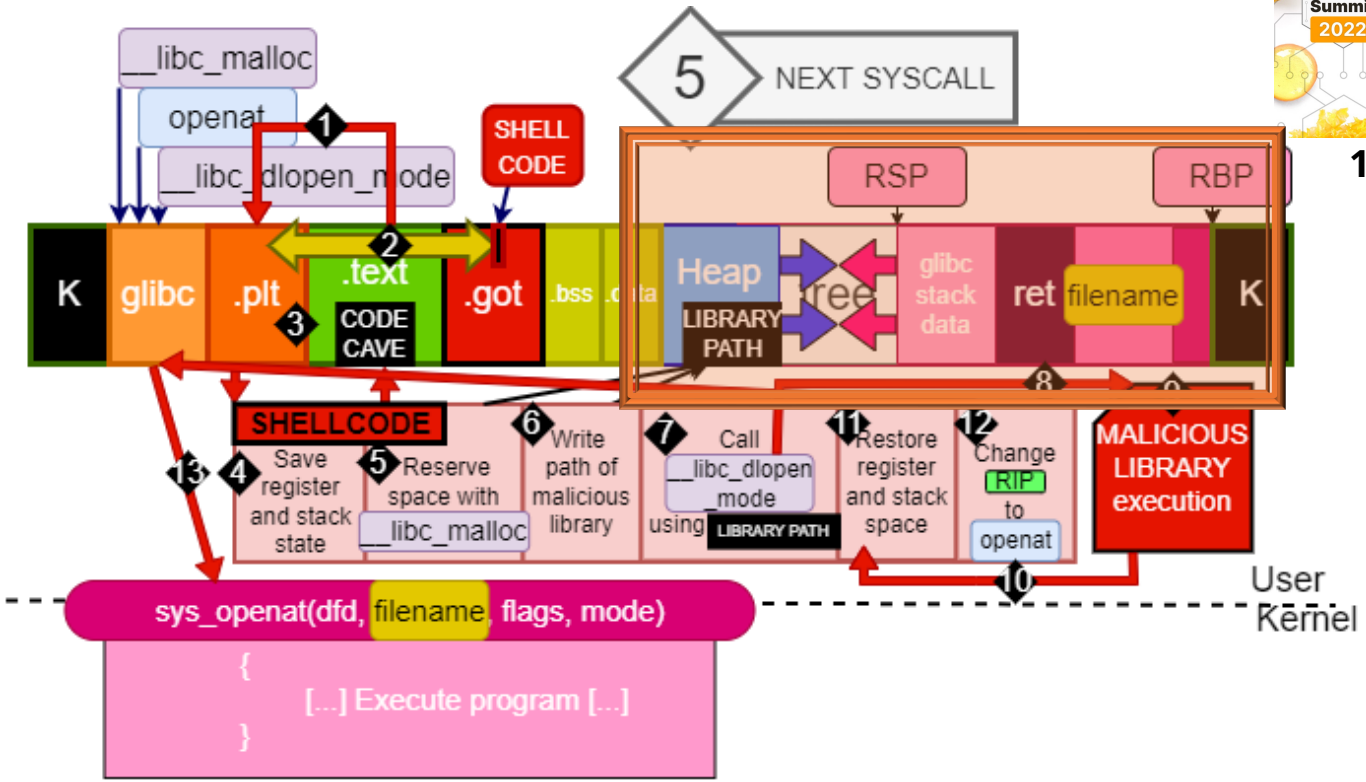
GOT HIJACKING

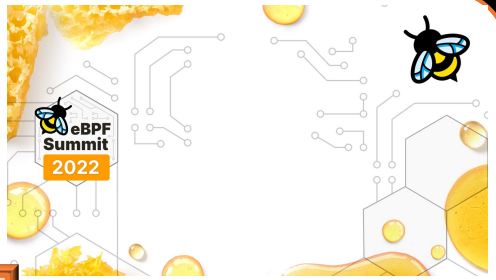


5 STAGES TECHNIQUE



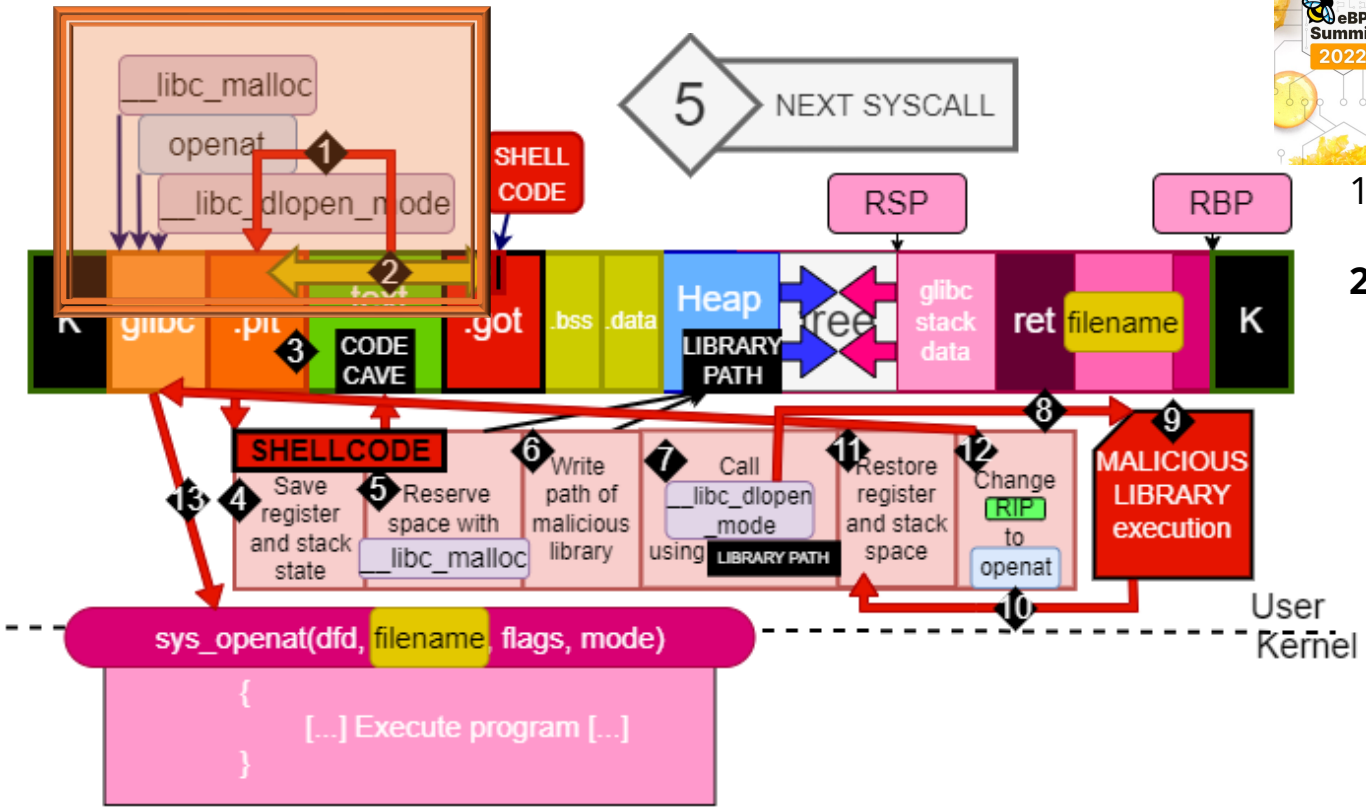
5 STAGES TECHNIQUE

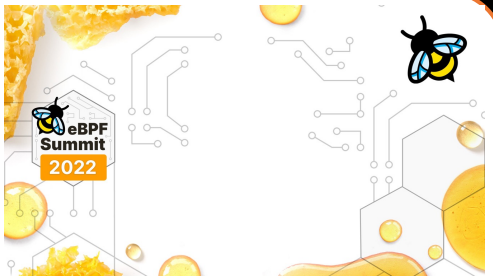




1. **Stack scanning**

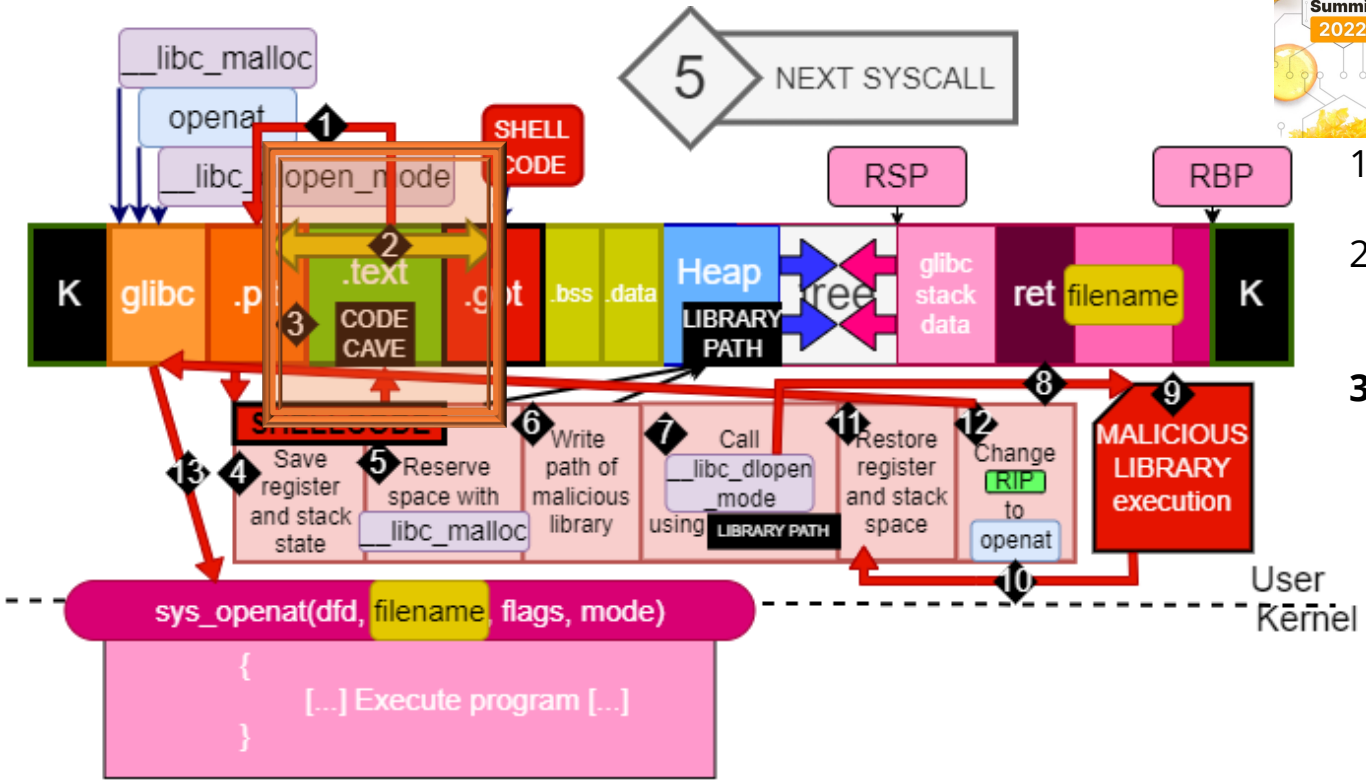
5 STAGES TECHNIQUE

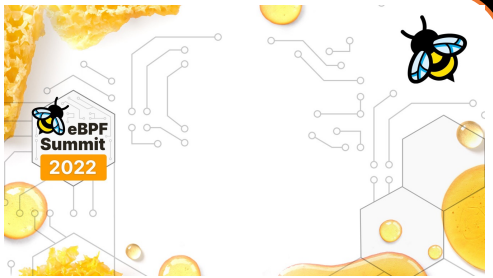




1. Stack scanning
2. **Locate key functions at glibc**

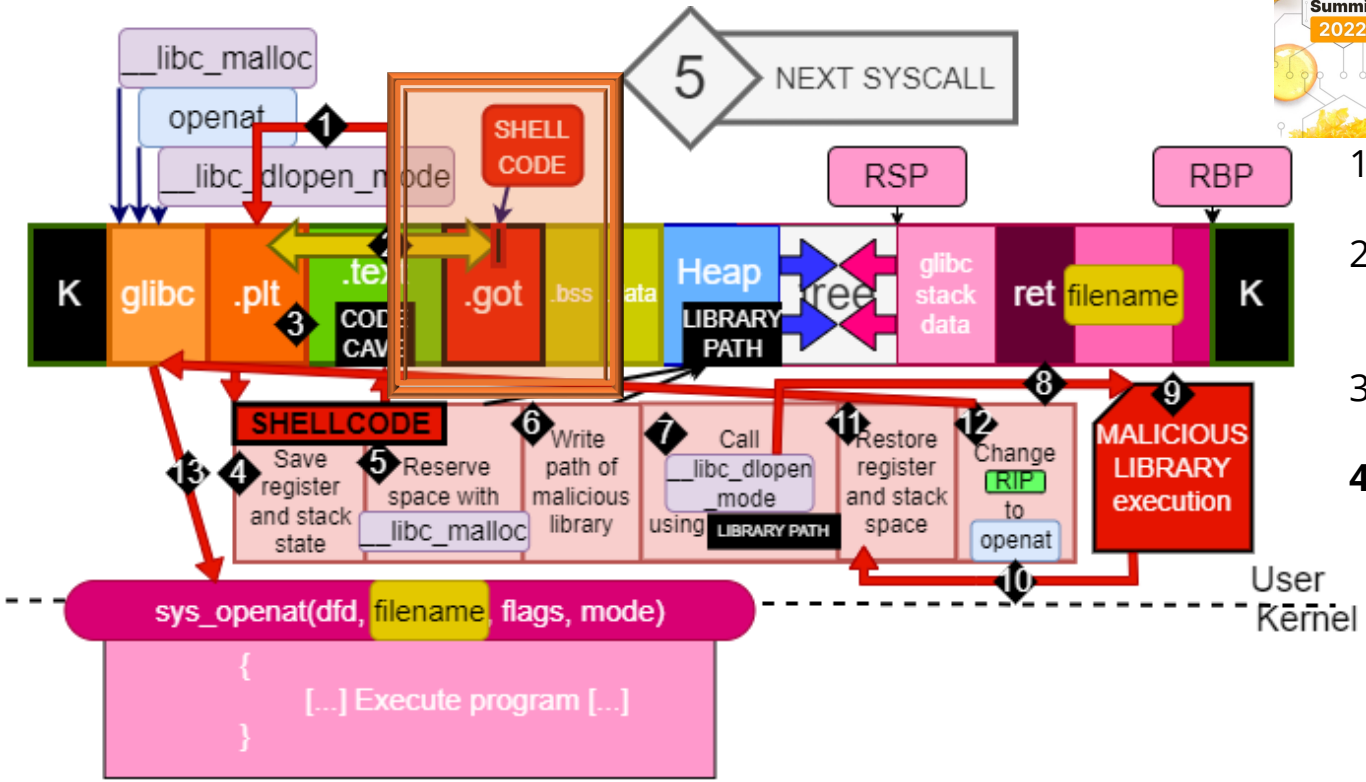
5 STAGES TECHNIQUE





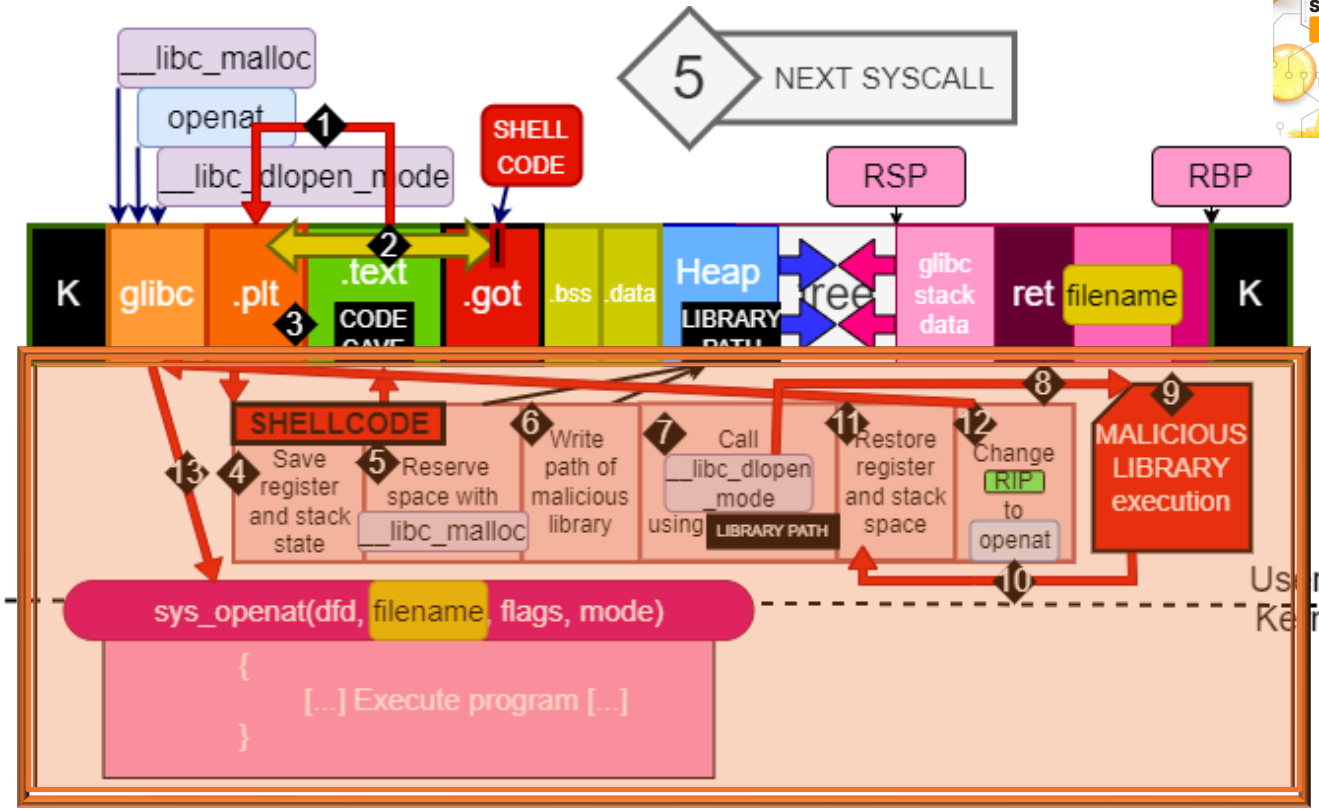
1. Stack scanning
2. Locate key functions at glibc
3. **Write shellcode**

5 STAGES TECHNIQUE



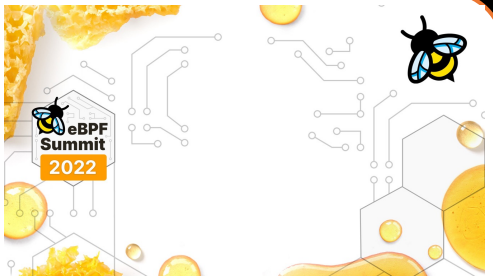
1. Stack scanning
2. Locate key functions at glibc
3. Write shellcode
4. **Overwrite GOT**

5 STAGES TECHNIQUE



1. Stack scanning
2. Locate key functions at glibc
3. Write shellcode
4. Overwrite GOT
5. **Exploitation**

GOT HIJACKING



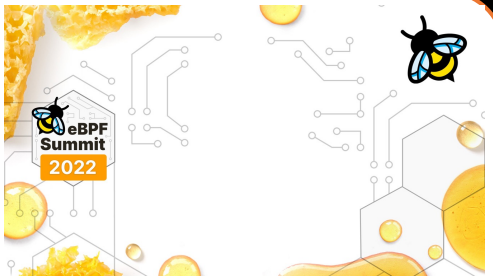
Are there protections against this attack?

- ASLR, PIE, Full RELRO, DEP/NX
- **All bypassed**

How novel is this attack?

- **Never done before** with eBPF
- Tested in **Ubuntu 21.04**
- Can be updated to any Linux version with relatively low effort

PRIVILEGE ESCALATION



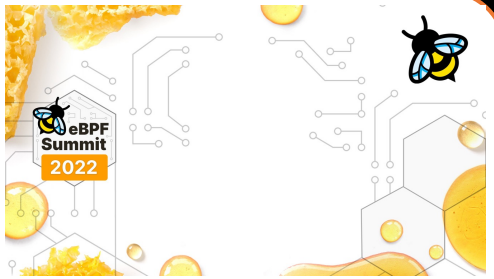
SUDO: Access control in Linux

- Configuration in **/etc/sudoers**

Malicious /etc/sudoers

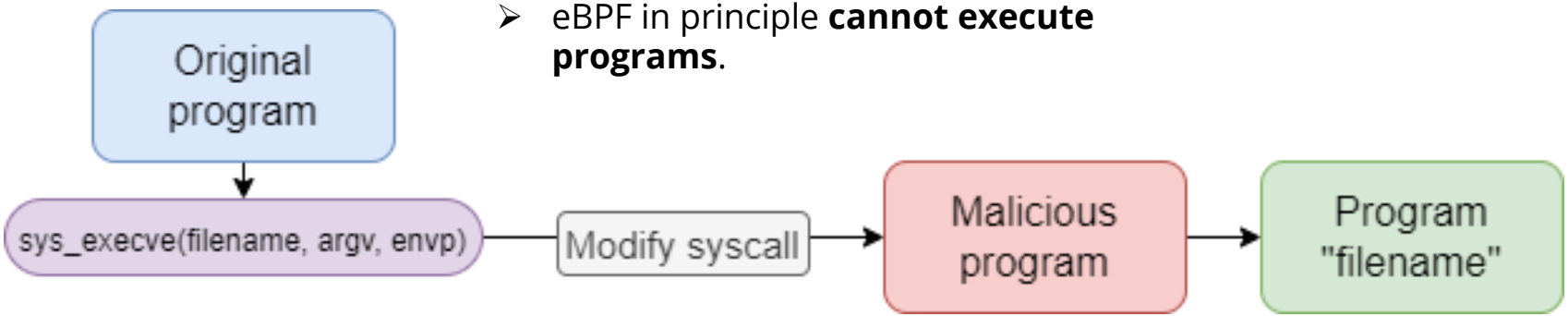
- Malicious kprobe/tracepoint program
- Any rootkit program can be run with privileged access

EXECUTION HIJACKING

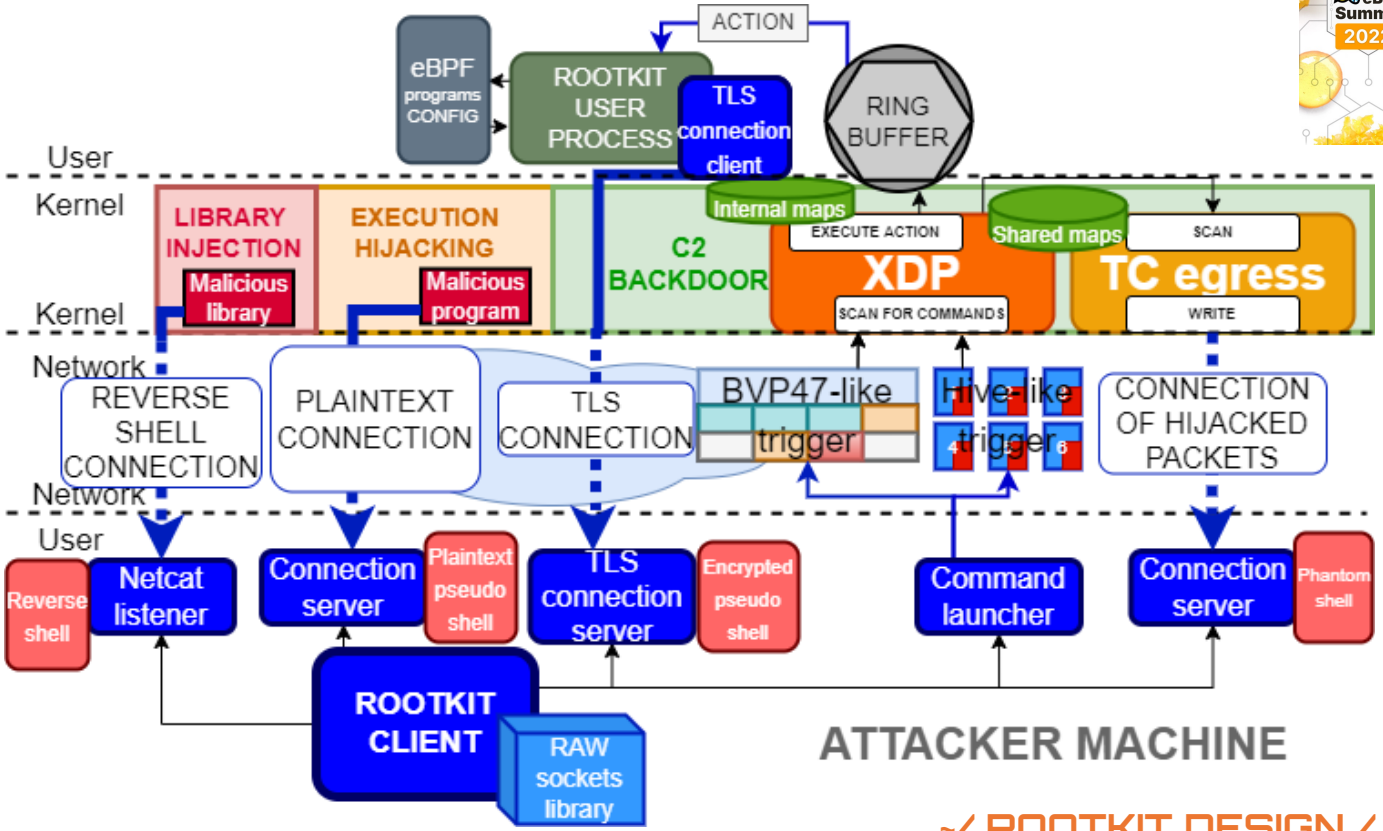


Attack implications:

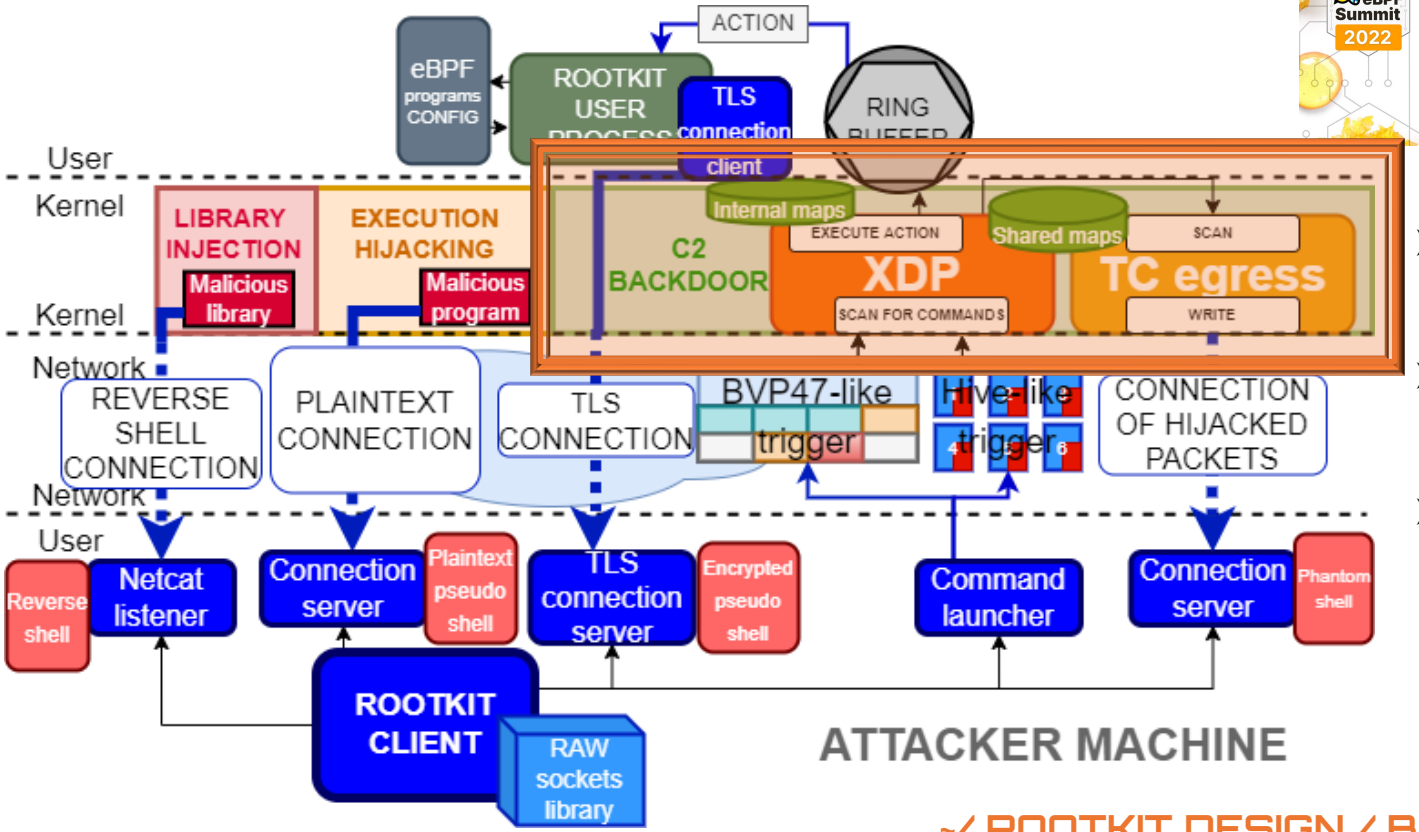
- We can **run malware secretly** every time a program is executed.
- eBPF in principle **cannot execute programs**.



BACKDOOR & C2 SYSTEM

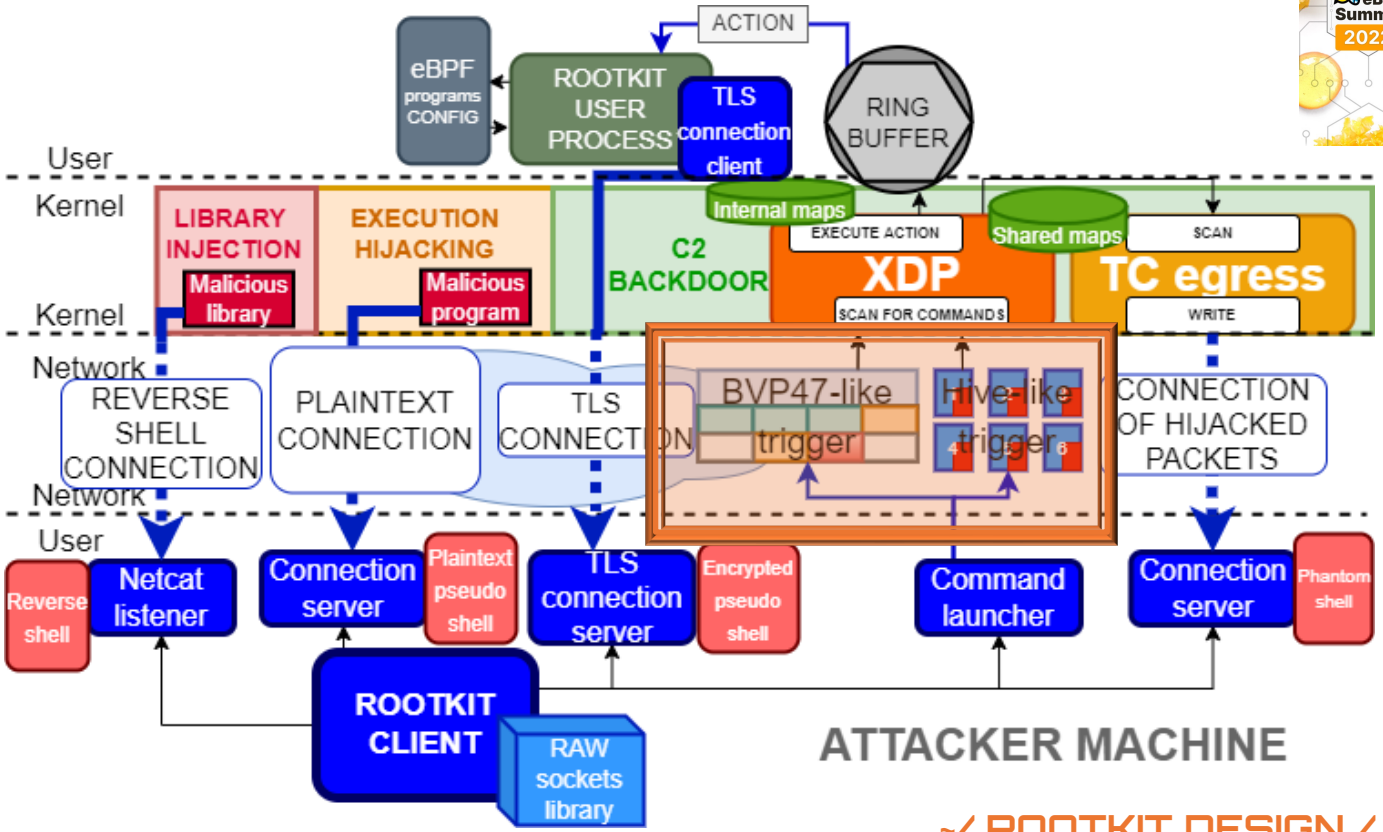


BACKDOOR & C2 SYSTEM



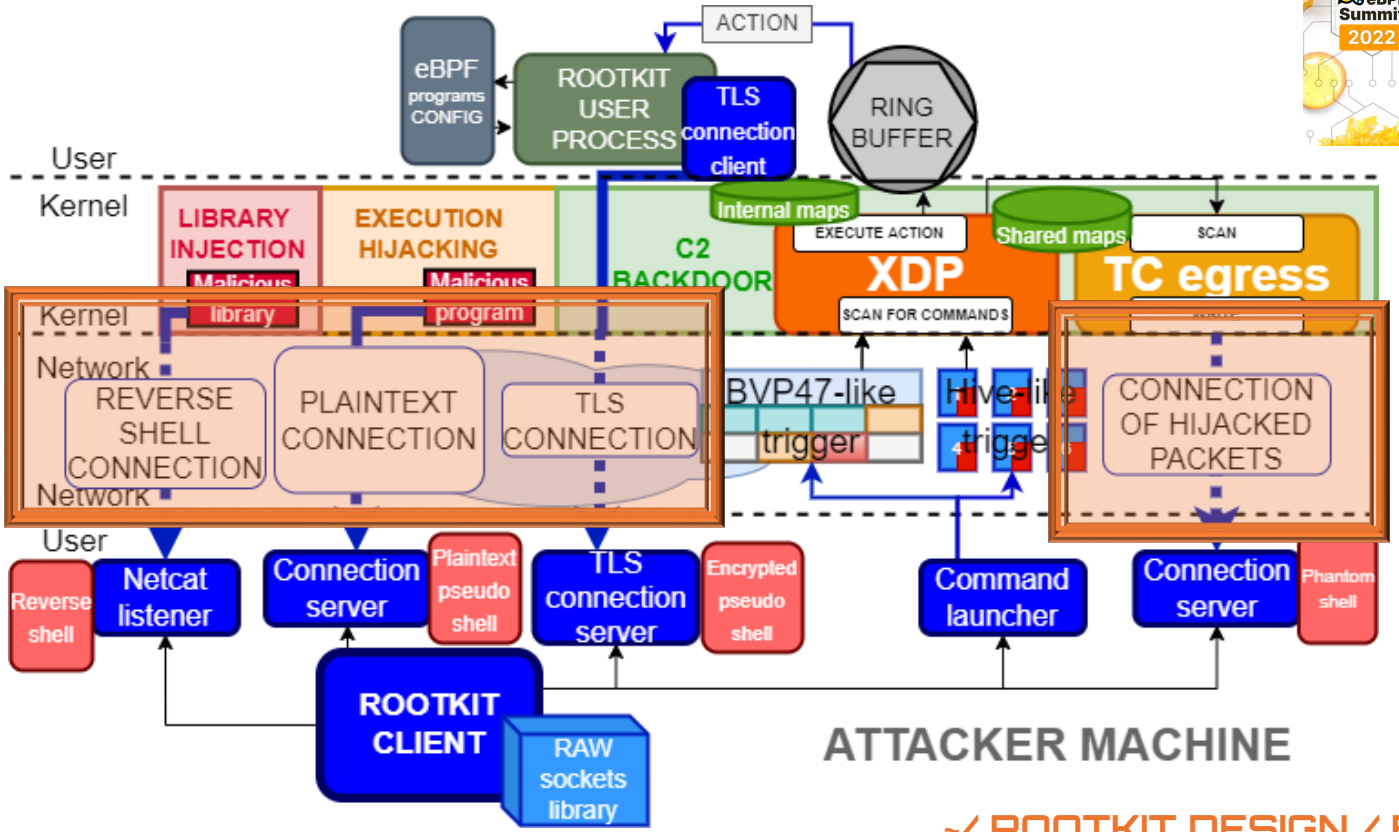
- **Backdoor triggers**
- Bypass **firewall** protections
- Indicate **actions** to execute

BACKDOOR & C2 SYSTEM



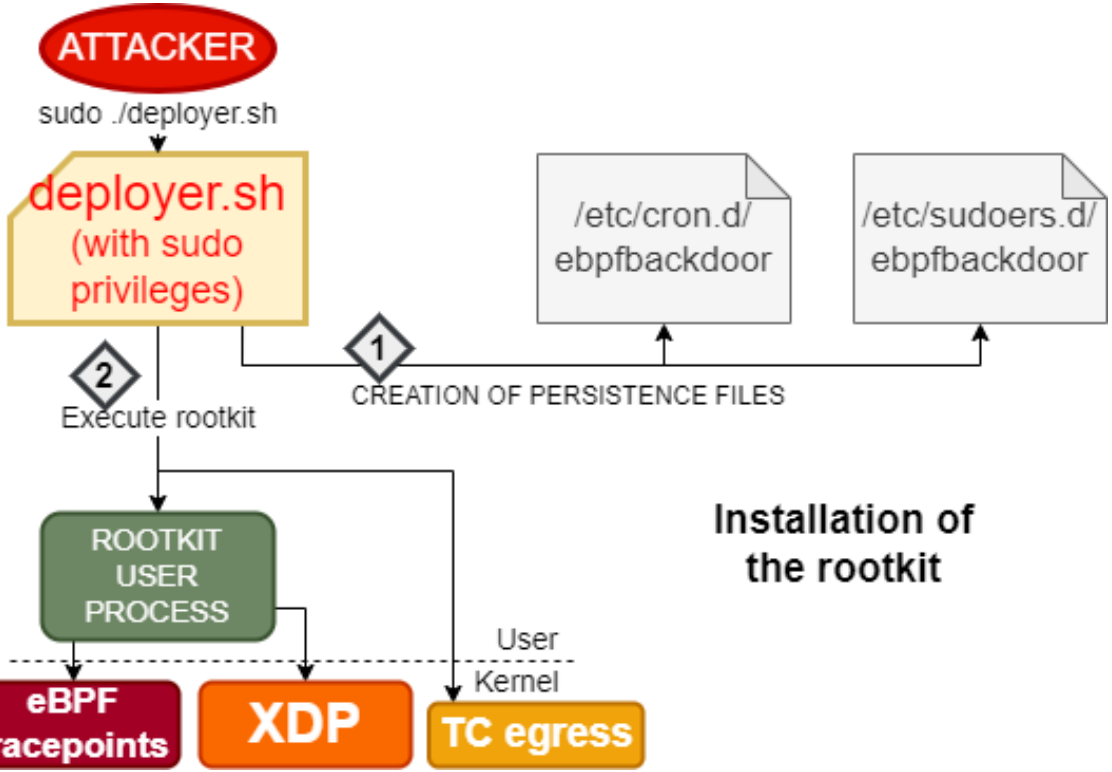
- **Backdoor triggers**
- Bypass **firewall** protections
- Indicate **actions** to execute

BACKDOOR & C2 SYSTEM



- **Reverse** shell
- **Plaintext** pseudo-shell
- **Encrypted** pseudo-shell
- **Phantom** shell

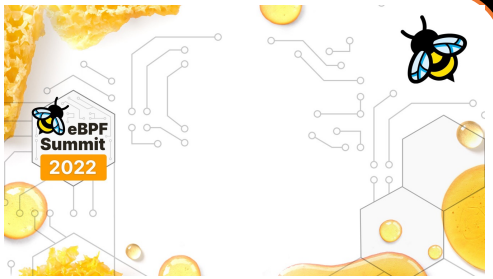
ROOTKIT PERSISTENCE



Installation of the rootkit

- **Cron** malicious file for **installation** persistence
- **Sudo** malicious file for **privileges** persistence

HIDING ROOTKIT FILES

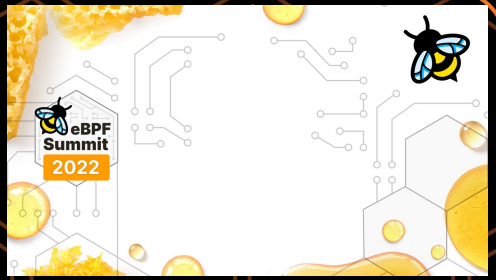


Listing files and directories

- Use of ***ls*** command
- Calls sys_getdents internally

Malicious sys_getdents

- All rootkit files and directories are invisible
- The persistence files are invisible

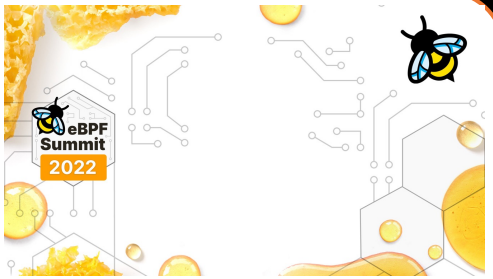


04

DEFENCE AGAINST THE DARK ARTS

- Defence techniques
- Final remarks

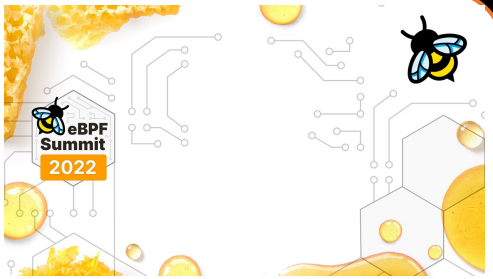
DEFENCE TECHNIQUES



Network monitoring

- Detect suspicious communications
- Firewalls at the endpoint can be deceived

DEFENCE TECHNIQUES



Network monitoring

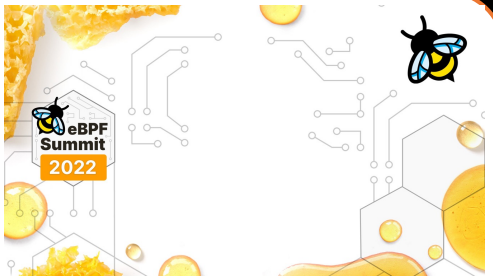
- Detect suspicious communications
- Firewalls at the endpoint can be deceived

Monitor eBPF

- Publicly-available eBPF tools monitor **bpf() activity**

https://github.com/libbpf/bpftool
https://github.com/Gui774ume/ebpfkit-monitor

DEFENCE TECHNIQUES



Network monitoring

- Detect suspicious communications
- Firewalls at the endpoint can be deceived

Monitor eBPF

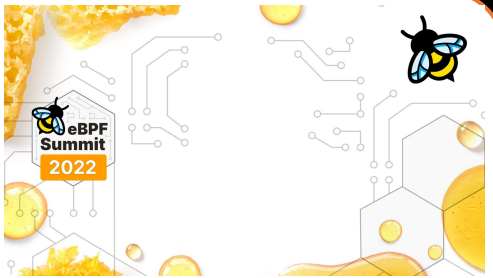
- Publicly-available eBPF tools monitor **bpf() activity**

https://github.com/libbpf/bpftool
https://github.com/Gui774ume/ebpfkit-monitor

Lowest-privilege eBPF

- Rootkits require **privileged** eBPF
- Solution: eBPF capabilities
  - CAP_BPF
  - CAP_NET_ADMIN
  - CAP_PERFMON
  - CAP_SYS_ADMIN

DEFENCE TECHNIQUES



Signed eBPF

- Only trusted and signed eBPF programs can be run in the kernel.

User: Password: Log in | Subscribe |

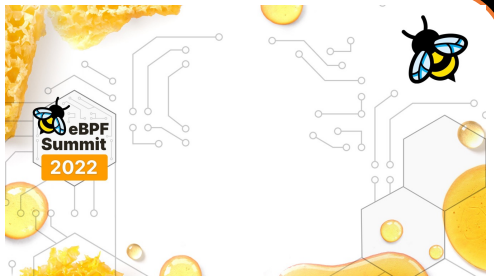
Toward signed BPF programs

By **Jonathan Corbet**
April 22, 2021

The kernel's BPF virtual machine is versatile; it is possible to load BPF programs into the kernel to carry out a large (and growing) set of tasks. The growing body of BPF code can reasonably be thought of as kernel code in its own right. But, while the kernel can check signatures on loadable modules and prevent the loading of modules that are not properly signed, there is no such

https://lwn.net/Articles/853489/

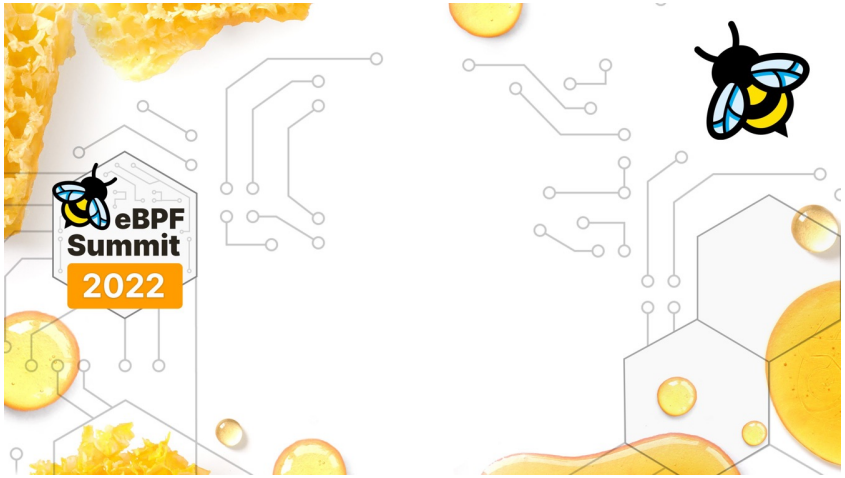
DON'T LET YOUR GUARD DOWN!



- eBPF restricts userland and kernel capabilities, but malware can find ways around them.
- Once the rootkit is installed, it is posible to avoid any further monitoring or detection efforts.
- **eBPF malware is a reality**.

Q&A on Slack!

github.com/h3xduck/TripleCross



Marcos Bajo (@h3xduck)
Juan Tapiador (@0xjet)

eBPF Summit
August 28-29, 2022