TripleCross
A Linux eBPF rootkit

Marcos Bajo
@h3xduck

Juan Tapiador
@0xjet

/RootedCON

LINUX ROOTKITS 101

USER-LEVEL ROOKITS

LD_PRELOAD technique

- User-level capabilities
- Limited system access
- Easily detectable

LINUX ROOTKITS 101

USER-LEVEL ROOKITS

LD_PRELOAD technique

- User-level capabilities
- Limited system access
- Easily detectable

KERNEL-LEVEL ROOTKITS

Linux Kernel Modules (LKMs)

- Kernel-level capabilities
- Advanced system control
- Extremely noisy

NEW GENERATION ROOTKITS

- Not noisy when being loaded
- Hard to remove once installed
- System-wide capabilities
- Relatively unknown ;)

NEW GENERATION ROOTKITS

- Not noisy when being loaded
- Hard to remove once installed
- System-wide capabilities
- Relatively unknown ;)

WHO WE ARE

Marcos Bajo
(aka ***h3xduck*****)**

@h3xduck

- Cybersecurity researcher at COSEC (UC3M)
- MSc student in cybersecurity
- Interested in low-level stuff
- Linux internals, malware, pwn

Juan Tapiador
(aka ***0xjet*****)**

@0xjet

- Currently Professor of Computer Security at UC3M and Head of the COSEC Lab
- Research in systems security, Internet measurements, and privacy

WHAT WE WILL LEARN

1. The eBPF technology

What is eBPF? How to use it? What can I do with it?

WHAT WE WILL LEARN

1. The eBPF technology

What is eBPF? How to use it? What can I do with it?

2. Offensive eBPF

Could an eBPF program be malicious?

WHAT WE WILL LEARN

1. The eBPF technology

What is eBPF? How to use it? What can I do with it?

2. Offensive eBPF

Could an eBPF program be malicious?

3. TripleCross: Building an eBPF rootkit

Can we build a rootkit using eBPF? How far can we get?

WHAT WE WILL LEARN

1. The eBPF technology

What is eBPF? How to use it? What can I do with it?

2. Offensive eBPF

Could an eBPF program be malicious?

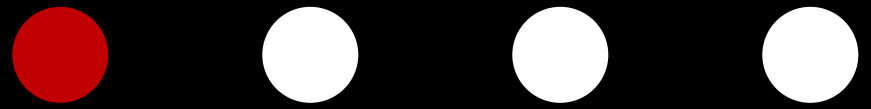
3. TripleCross: Building an eBPF rootkit

Can we build a rootkit using eBPF? How far can we get?

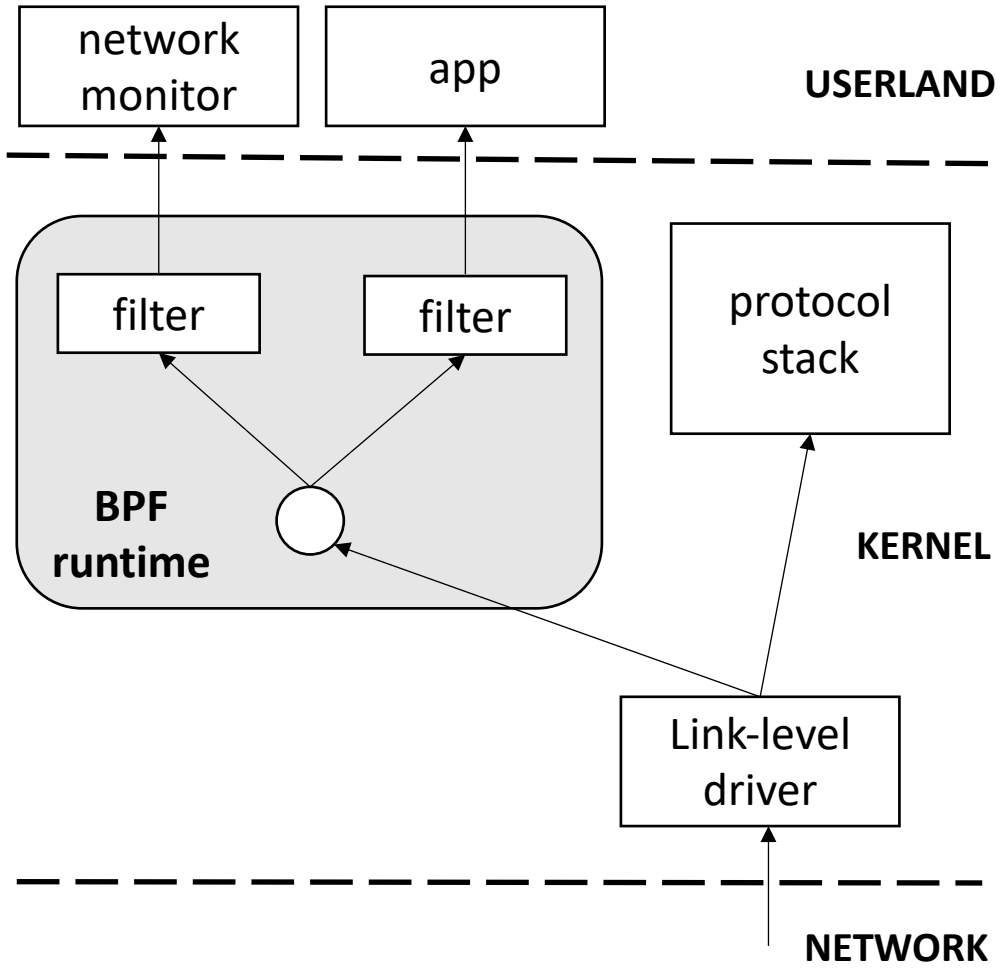
4. Defenses and conclusions

Are we defenseless against eBPF rootkits?

1 The eBPF technology



WHAT IS eBPF ?



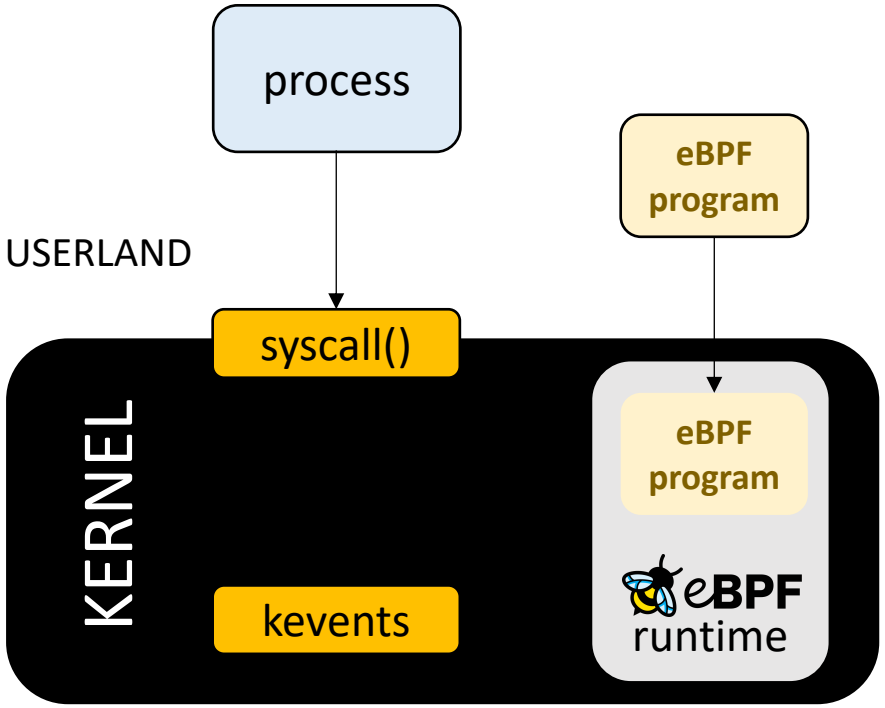
BPF (1992)

- Attach (hook) to network events
- Filter and measure/analyze network traffic

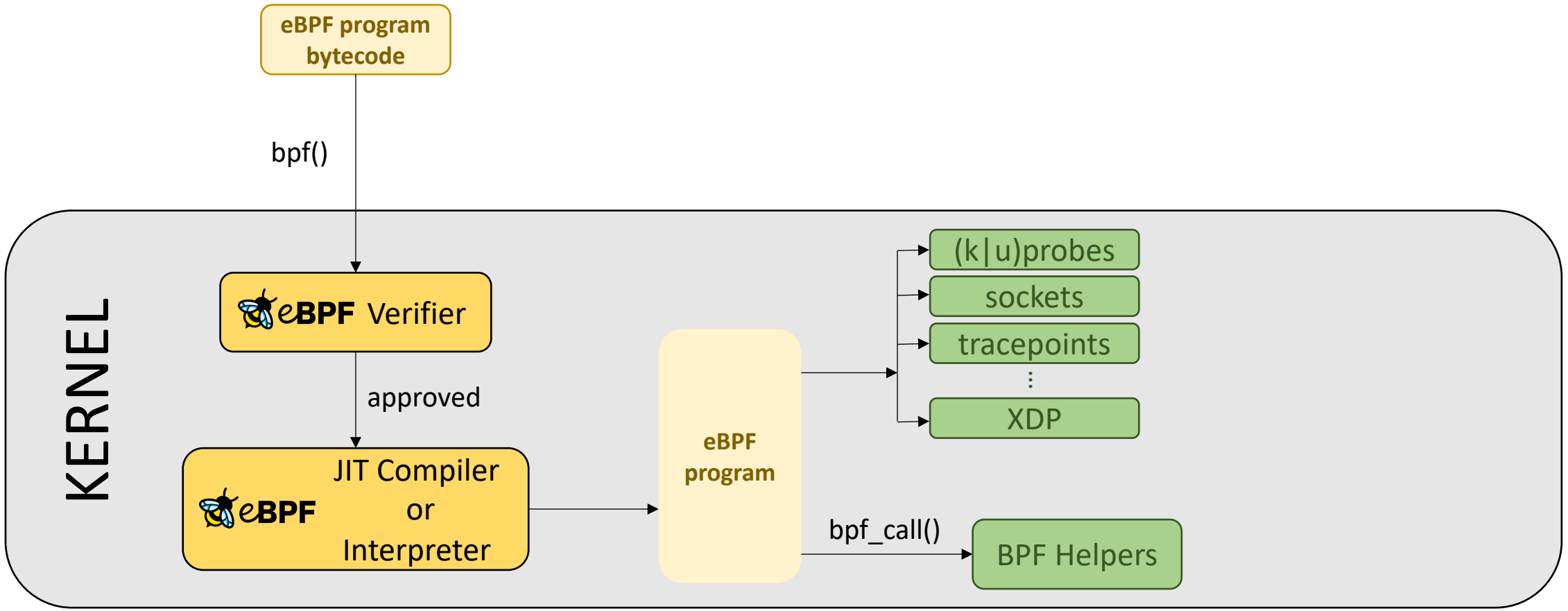
WHAT IS eBPF ?

eBPF (2014, still evolving)

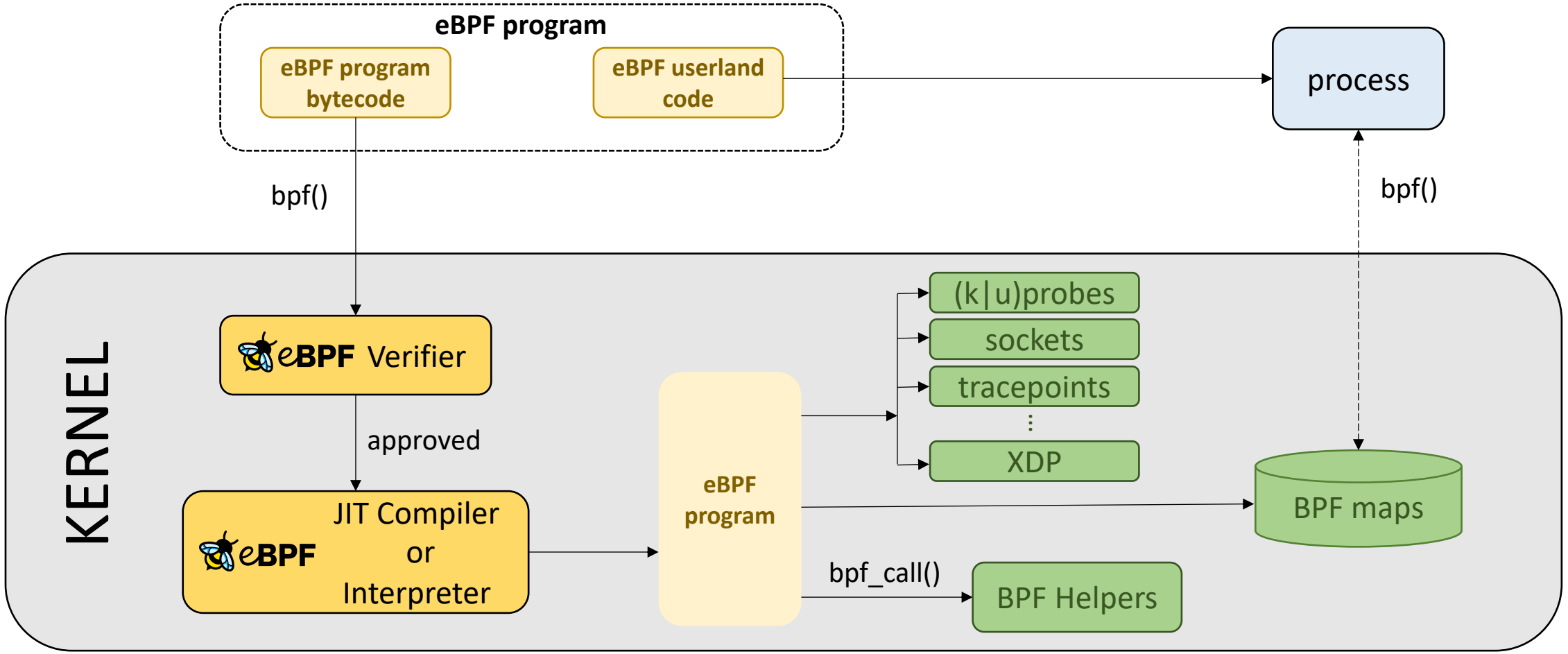
- Extends kernel observability and programmability
- Access to essentially all syscalls and kernel events (more than LKMs)
- Data structures to store and exchange data with userland programs



eBPF = PL + RUNTIME



eBPF = PL + RUNTIME



TRACEPOINTS

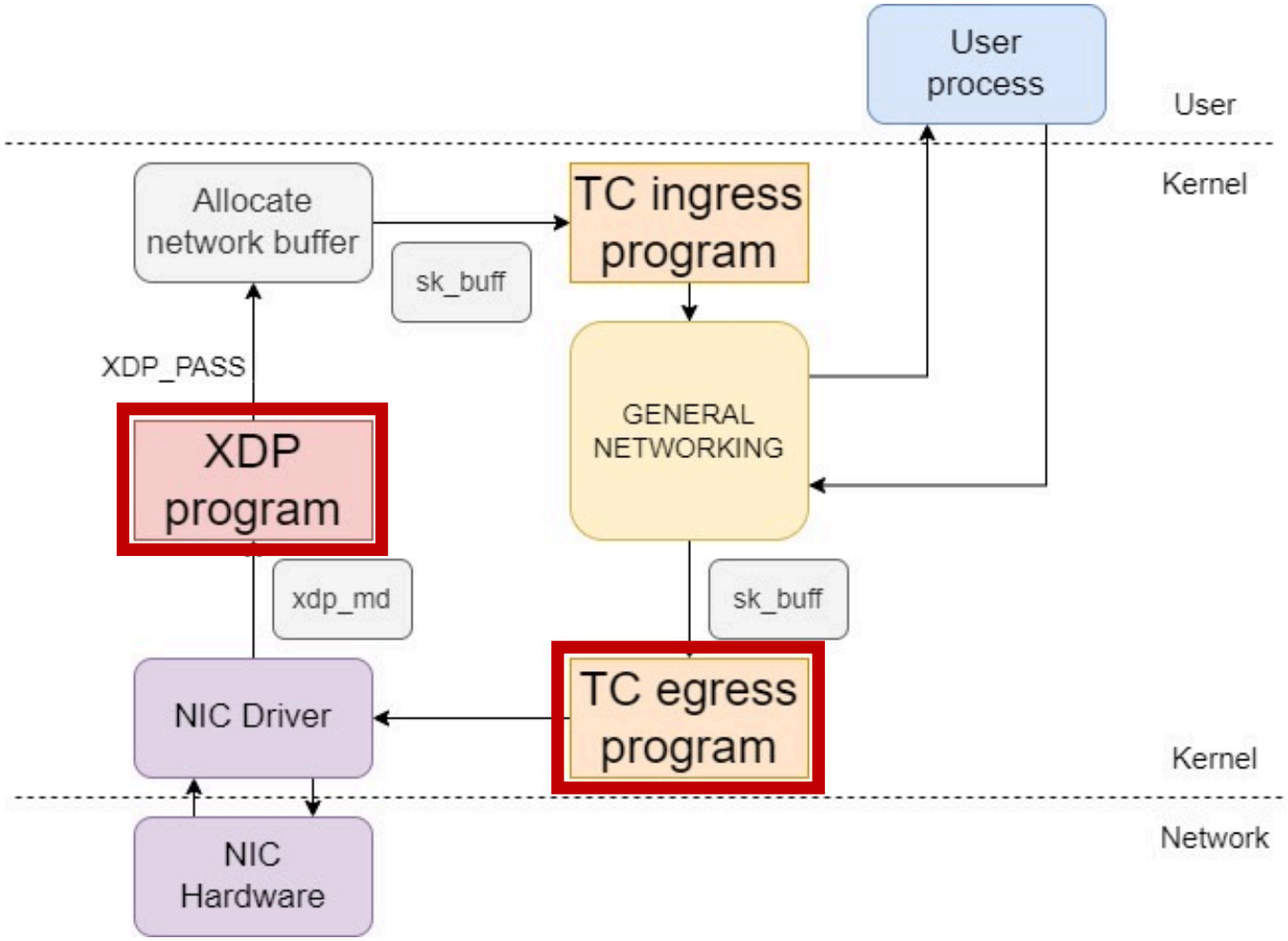
```c
int main(){
    char filename[BUFLEN];
    ...
    int map_fd = bpf_obj_get("/sys/fs/bpf/my_map");
    bpf_map_lookup_elem(map_fd, &key, &filename);
    ...
}
```

USERLAND

```c
struct format {
    int dfd;
    char* filename;
    uint flags;
    umode_t umode;
}

SEC(tp/syscalls/sys_enter_openat)
int hook(struct format *ctx){
    char filename[BUFLEN] = {0};
    bpf_probe_read_user(&filename, BUFLEN, ctx->filename);
    bpf_map_update_elem(&my_map, &key, filename, 0);
}
```

KERNEL

eBPF NETWORKING



- ✓ Read packets
- ✓ Modify packets
- ✓ Drop packets
- X Generate new packets

XDP PROGRAMS

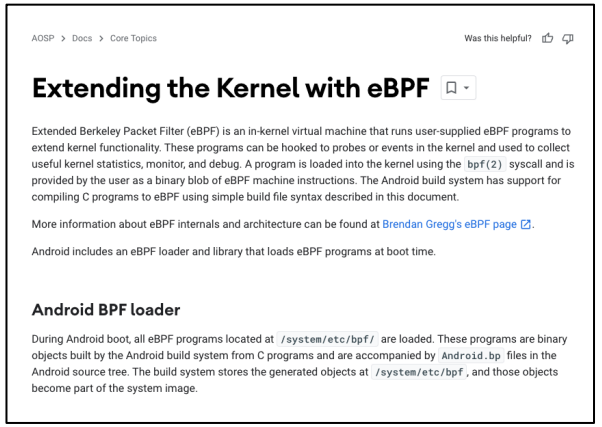
```
SEC("xdp")
int xdp_receive(struct xdp_md *ctx){
  void *data_end = (void *)(long)ctx->data_end; //buffer start
  void *data = (void *)(long)ctx->data;         //buffer end

  // Parse packet headers
  struct ethhdr *eth = data;
  struct iphdr *ip = data + sizeof(*eth);
  struct tcphdr *tcp = (void *)ip + sizeof(*ip);

  // Ensure header correctness
  ...

  // Access headers and payload
  payload = (void *)tcp + tcp->doff*4;

  ...
}
```

WHERE ELSE IS eBPF ?

All major cloud providers have adopted it for observability and programmability applications:

- Network
- Security
- Profiling

android


AOSP > Docs > Core Topics

Was this helpful?

Extending the Kernel with eBPF

Extended Berkeley Packet Filter (eBPF) is an in-kernel virtual machine that runs user-supplied eBPF programs to extend kernel functionality. These programs can be hooked to probes or events in the kernel and used to collect useful kernel statistics, monitor, and debug. A program is loaded into the kernel using the `bpf(2)` syscall and is provided by the user as a binary blob of eBPF machine instructions. The Android build system has support for compiling C programs to eBPF using simple build file syntax described in this document.

More information about eBPF internals and architecture can be found at Brendan Gregg's eBPF page.

Android includes an eBPF loader and library that loads eBPF programs at boot time.

Android BPF loader

During Android boot, all eBPF programs located at `/system/etc/bpf/` are loaded. These programs are binary objects built by the Android build system from C programs and are accompanied by `Android.bp` files in the Android source tree. The build system stores the generated objects at `/system/etc/bpf`, and those objects become part of the system image.


Microsoft Microsoft Security Azure Dynamics 365 Microsoft 365 Microsoft Teams Windows 365

Microsoft Open Source Blog

Making eBPF work on Windows

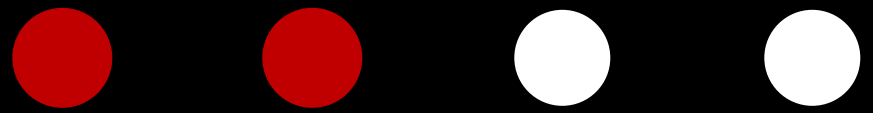
May 10, 2021 • 3 min read Share

Dave Thaler
Partner Software Engineer, Microsoft

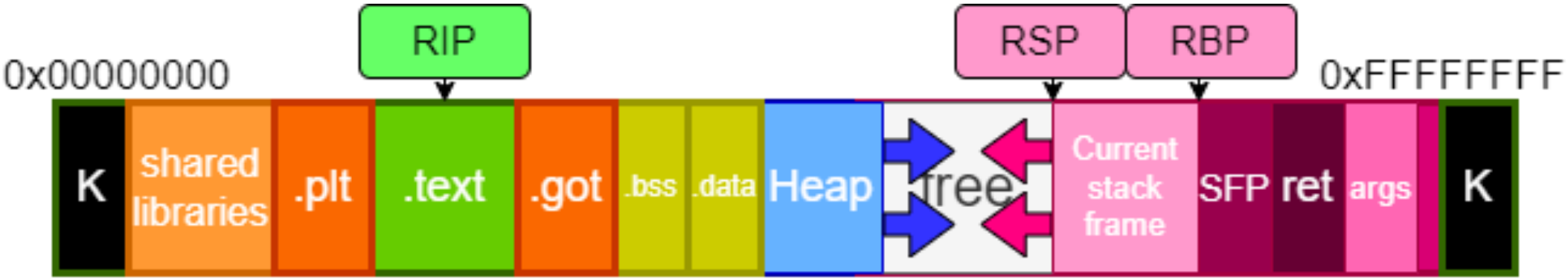
Poorna Gaddehosur
Principal Software Engineer Lead, Microsoft

eBPF is a well-known but revolutionary technology—providing programmability, extensibility, and agility. eBPF has been applied to use cases such as denial-of-service protection and observability. Over time, a significant ecosystem of tools, products, and experience has been built up around eBPF. Although support for eBPF was first implemented in the Linux kernel, there has been increasing interest in allowing eBPF to be used on other operating systems and also to extend user-mode services and daemons in addition to just the kernel.

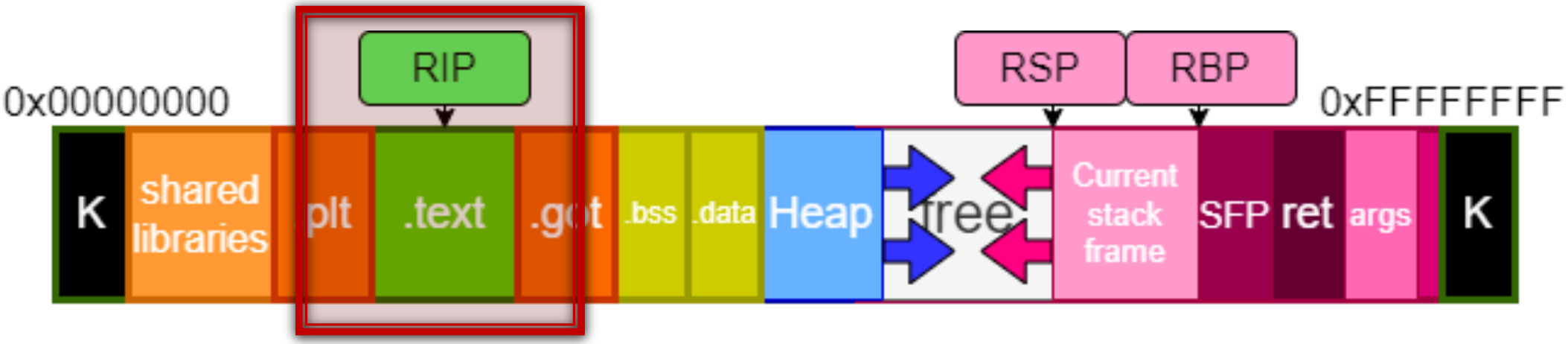
2 Offensive eBPF



VIRTUAL MEMORY

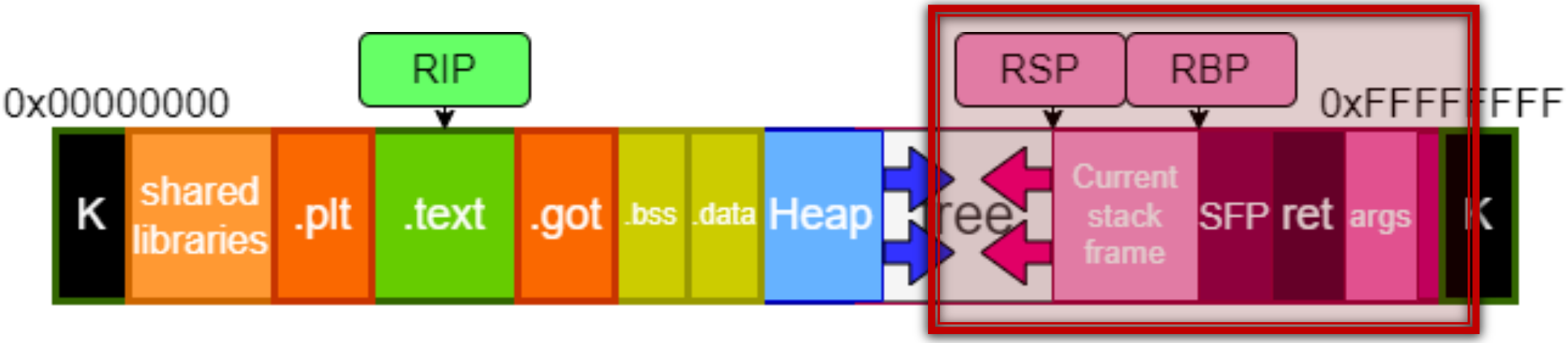


VIRTUAL MEMORY



Program instructions

VIRTUAL MEMORY



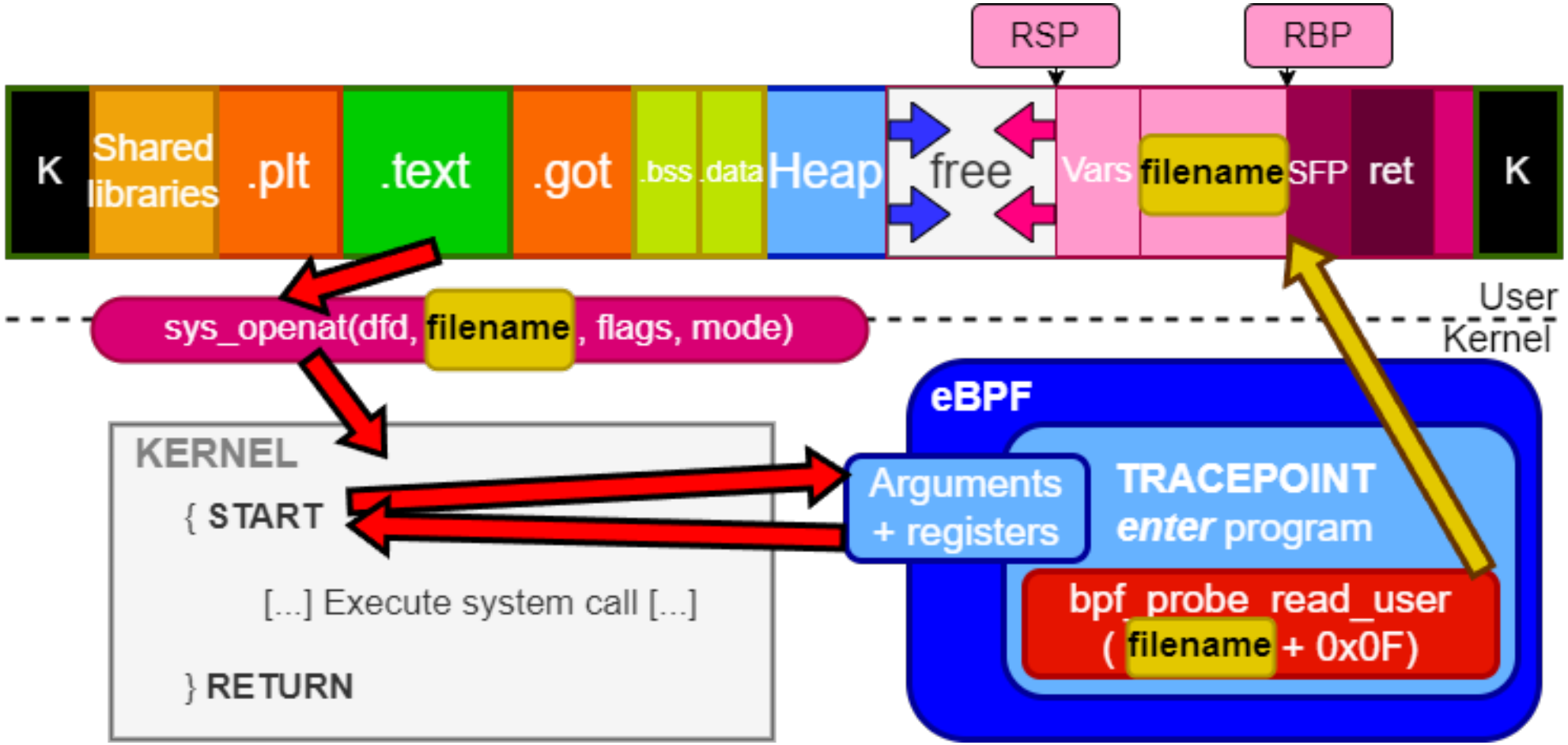
Program instructions

Local variables, parameters, return addresses

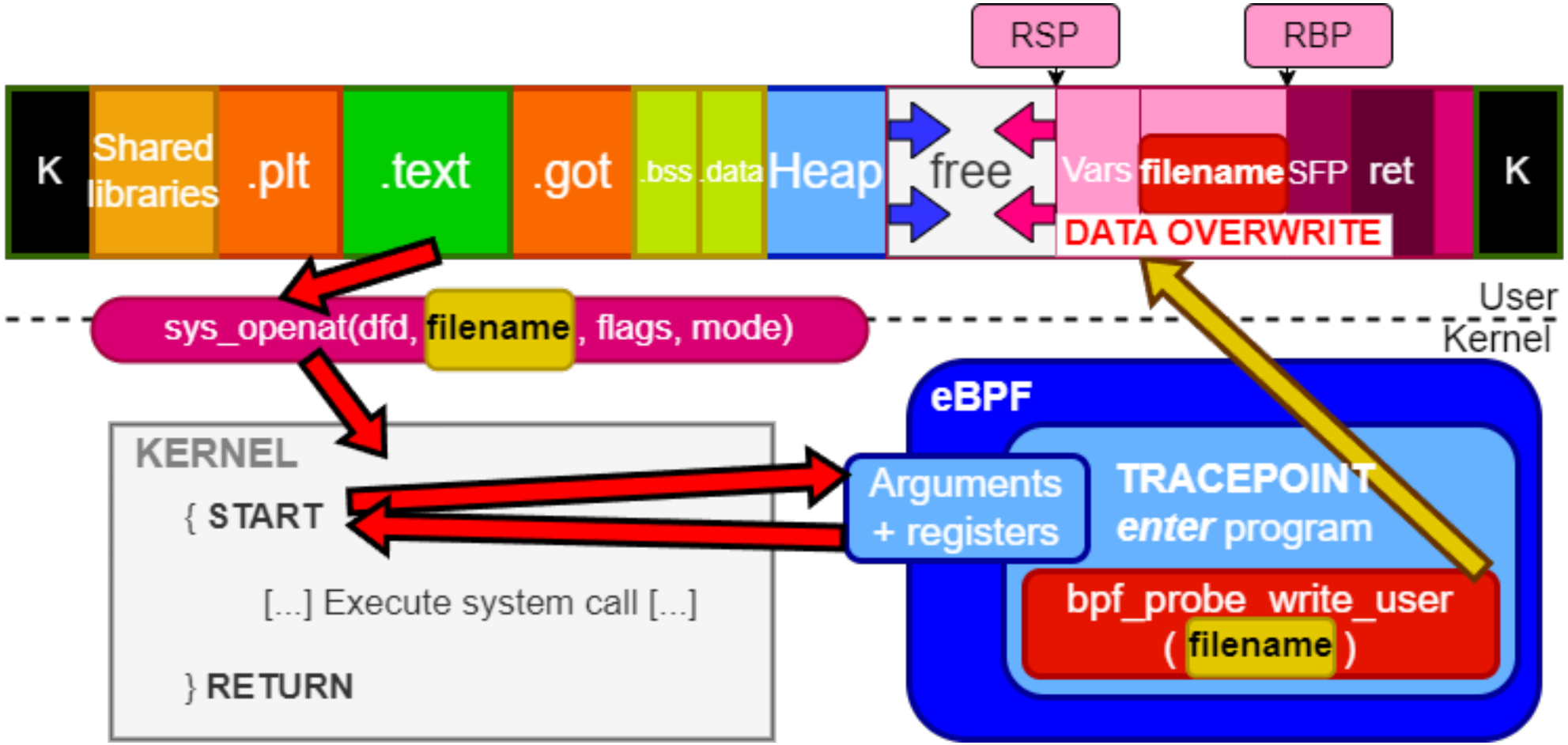
OUT-OF-BOUNDS USERLAND R+W

```
struct sys_openat_enter_ctx{
  […]
  int dfd;              // file descriptor
  char* filename;       // name of the file to open
  unsigned int flags;   // access mode
  umode_t umode;        // permissions for new file
}

SEC(tp/syscalls/sys_enter_openat)
int tp_sys_enter_openat(struct sys_openat_enter_ctx *ctx){
  char filename[BUFLEN] = {0};
  ...
  bpf_probe_read_user(&filename, BUFLEN, (char*) ctx->filename + 0x0F);
  ...
  bpf_probe_write_user((void*)(ctx->filename + 0x0F), (void*)xyzzy, 1);
  ...
}
```

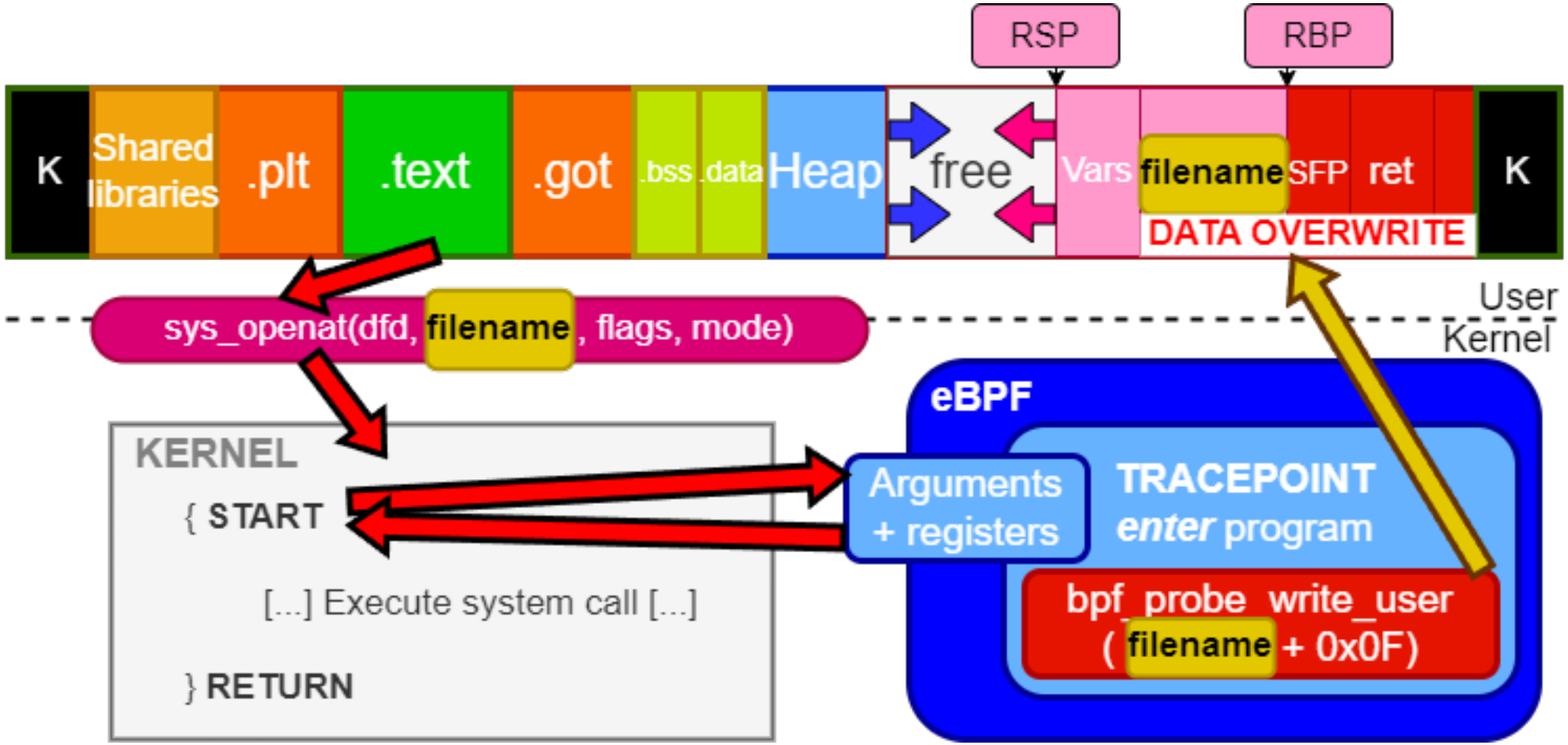
OUT-OF-BOUNDS USERLAND R+W



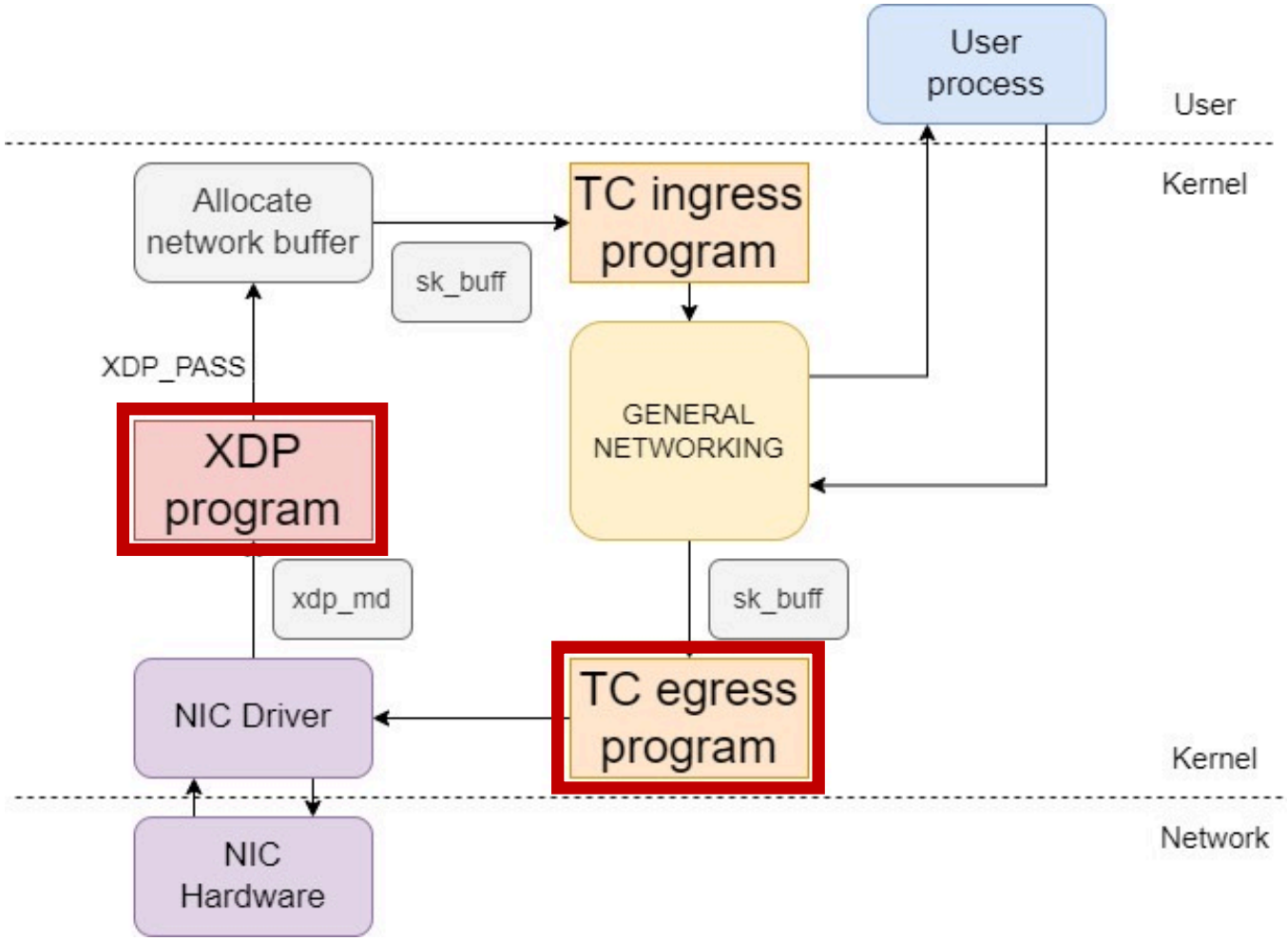
OUT-OF-BOUNDS USERLAND R+W



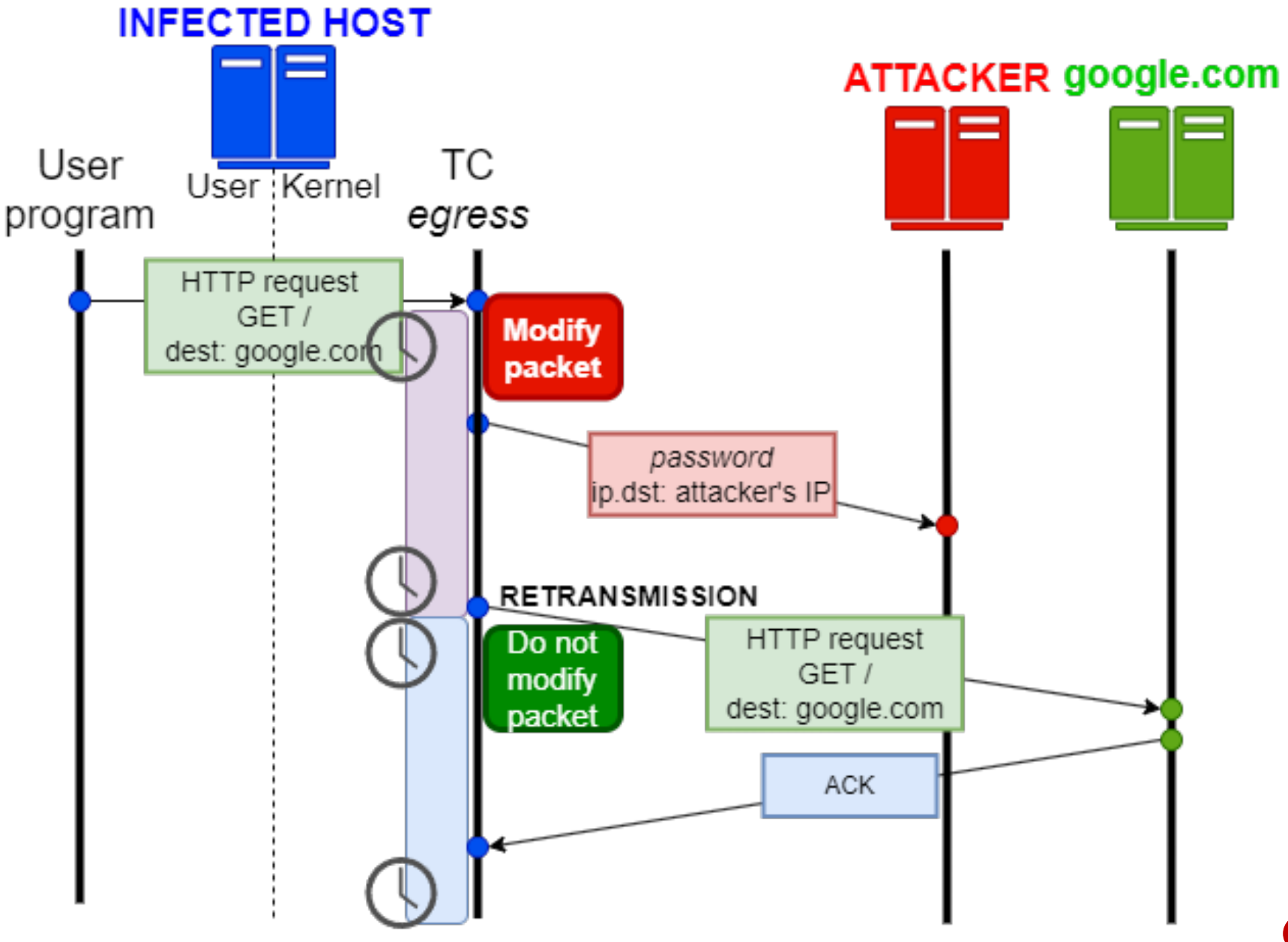
OUT-OF-BOUNDS USERLAND R+W



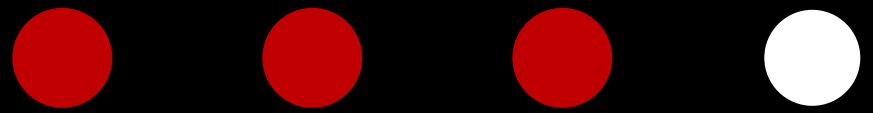
eBPF NETWORKING

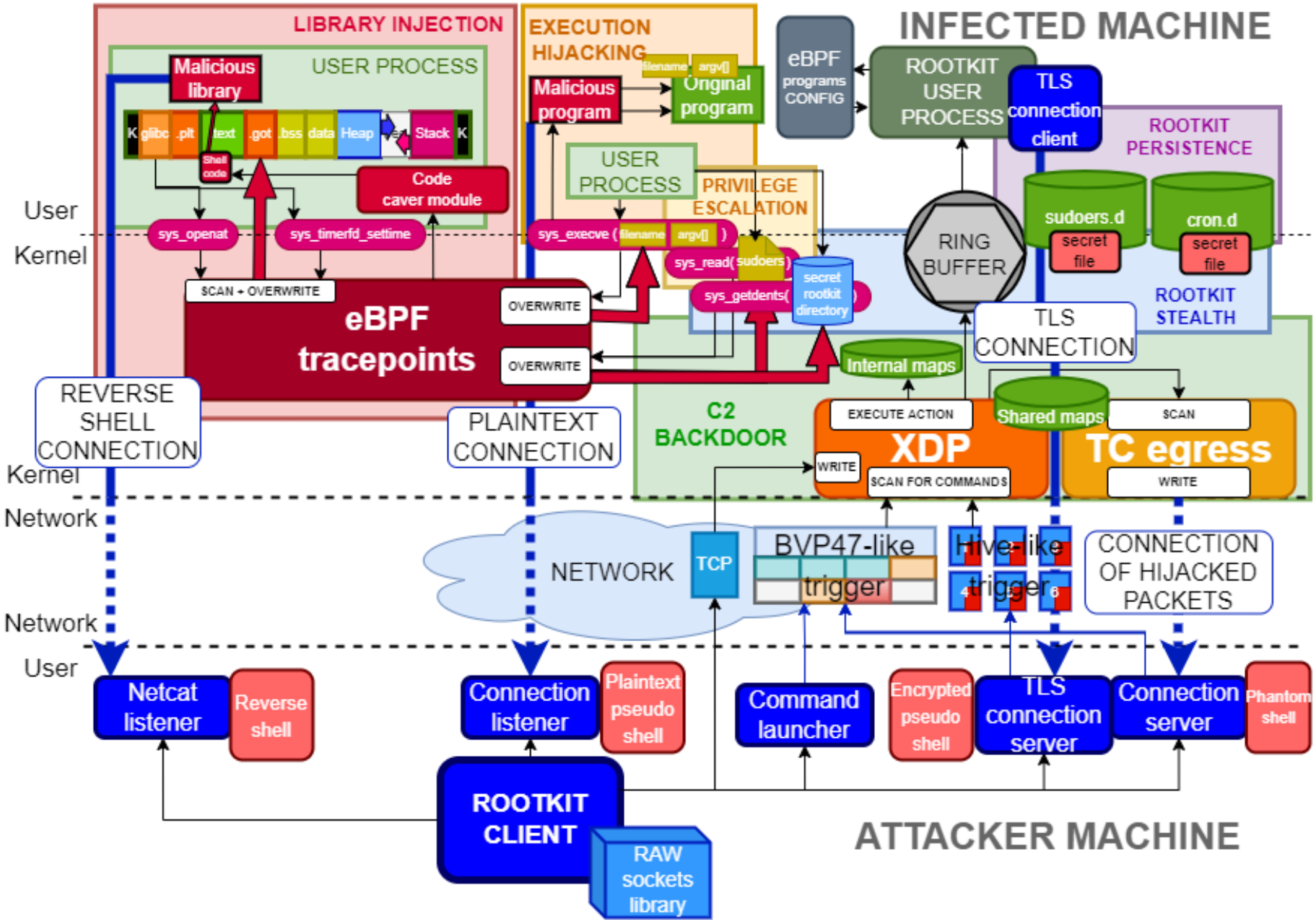


- ✓ Read packets
- ✓ Modify packets
- ✓ Drop packets
- X Generate new packets

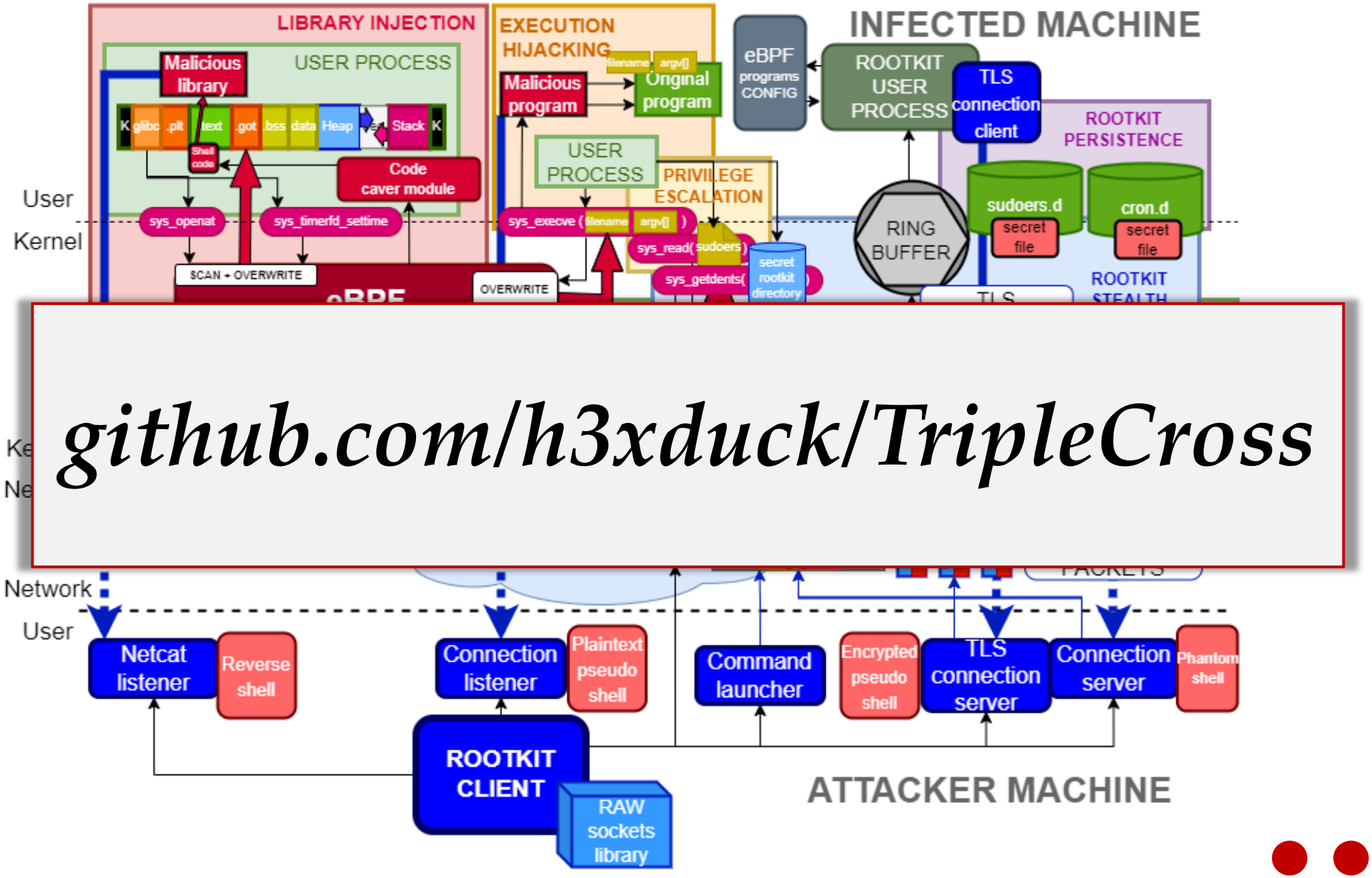

INFECTED HOST
ATTACKER
google.com
User program
User
Kernel
TC
egress
HTTP request
GET /
dest: google.com
Modify packet
password
ip.dst: attacker's IP
RETRANSMISSION
Do not modify packet
HTTP request
GET /
dest: google.com
ACK

3 TripleCross: eBPF rootkit



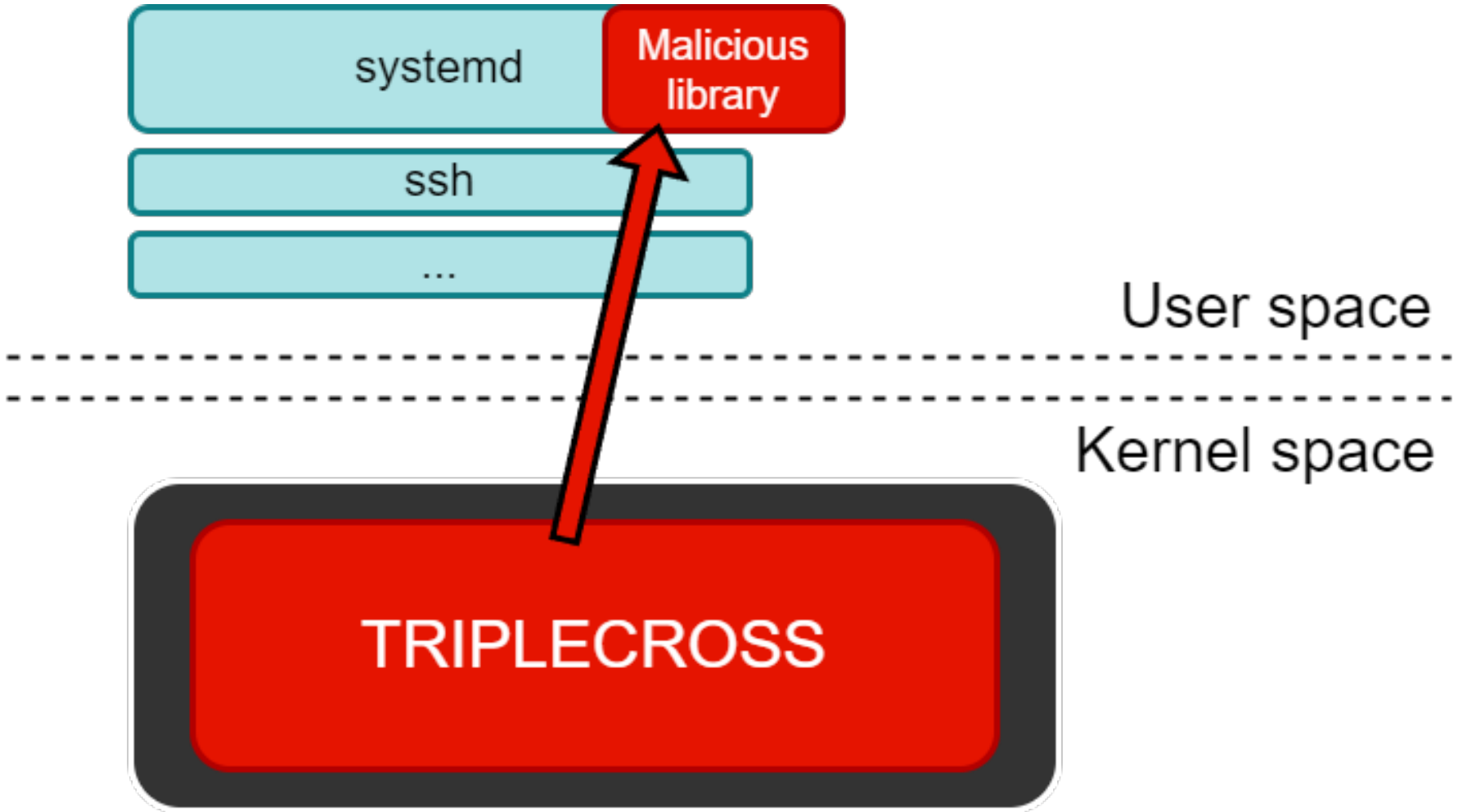


TripleCross rootkit



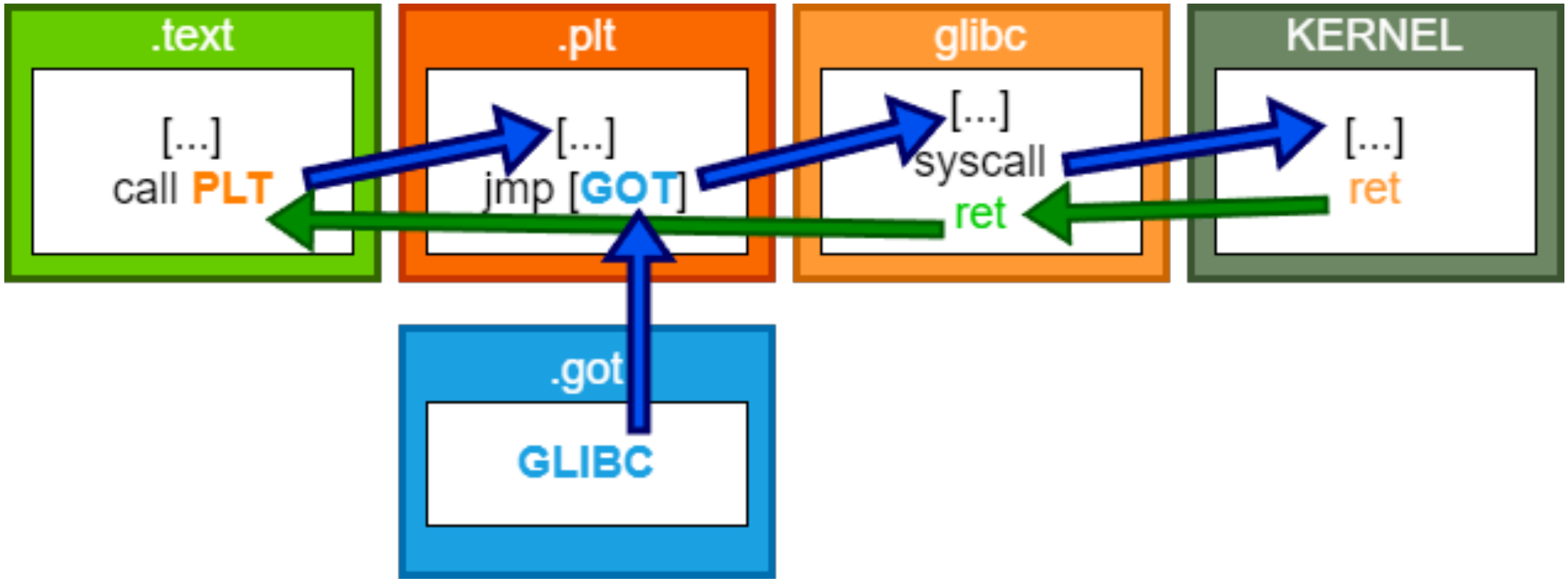
github.com/h3xduck/TripleCross

LIBRARY INJECTION

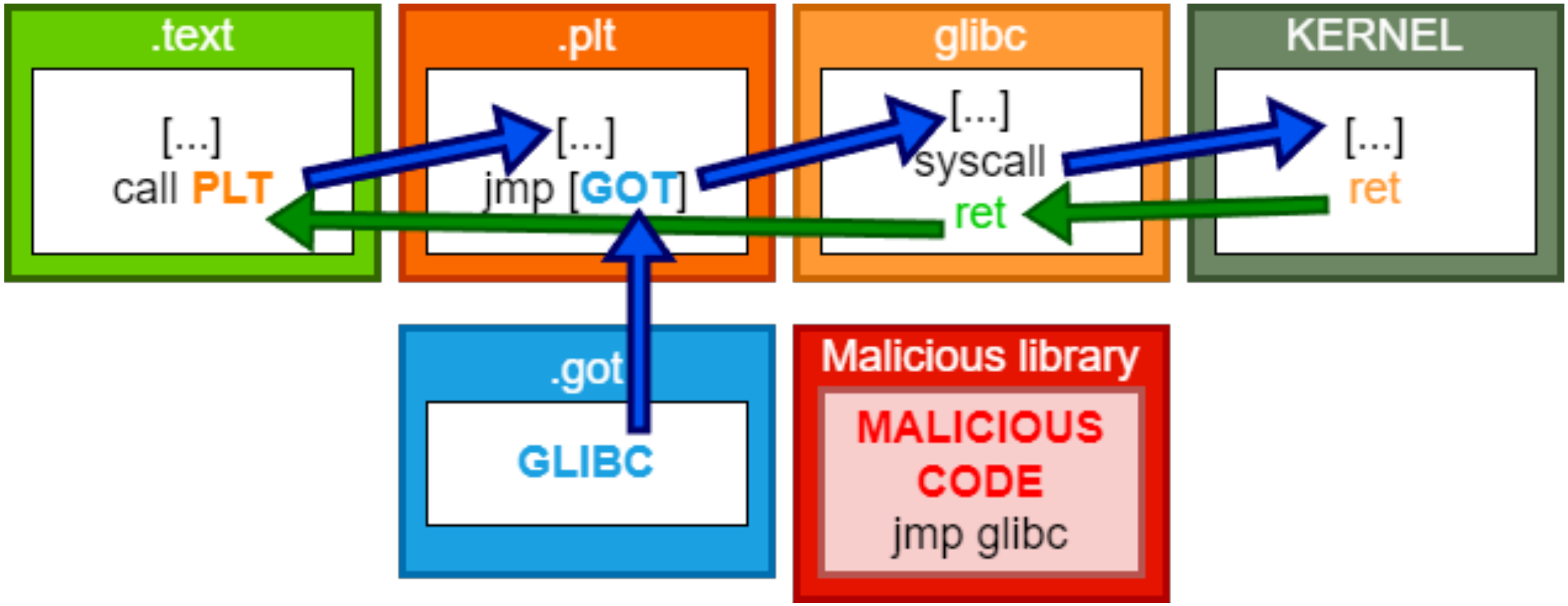


LAZY BINDING

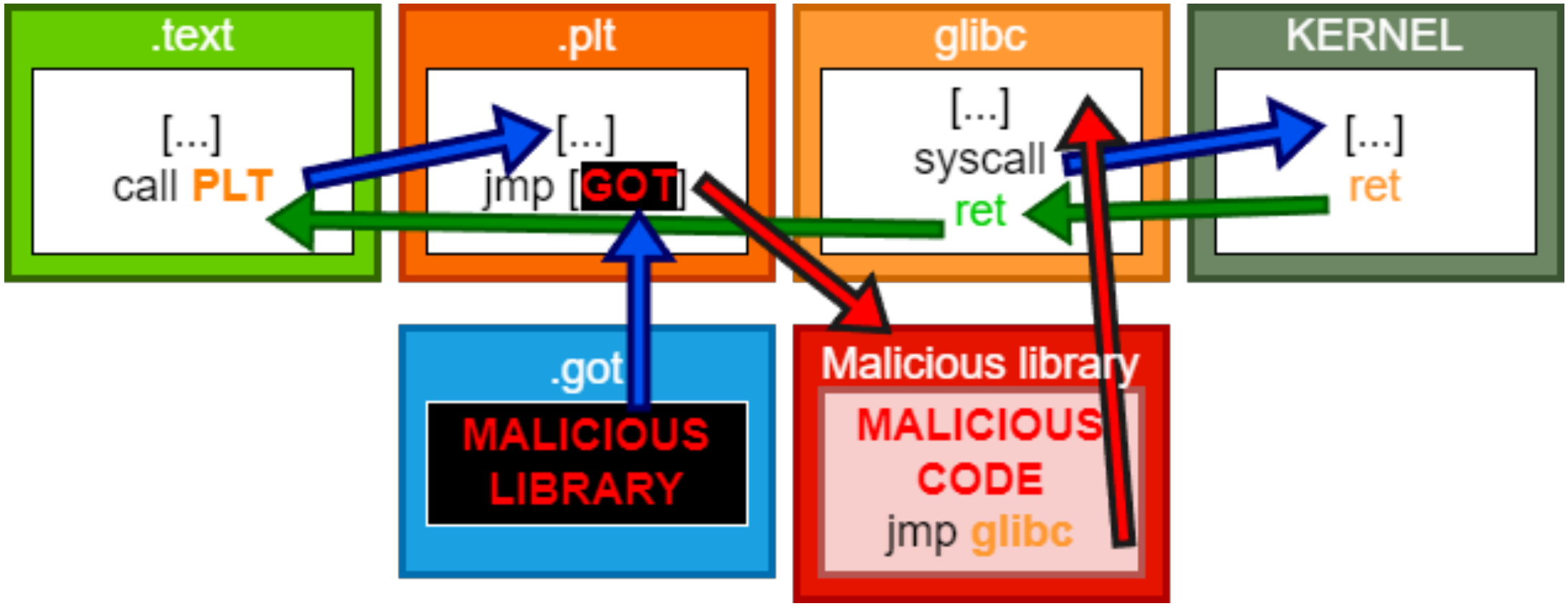
LAZY BINDING



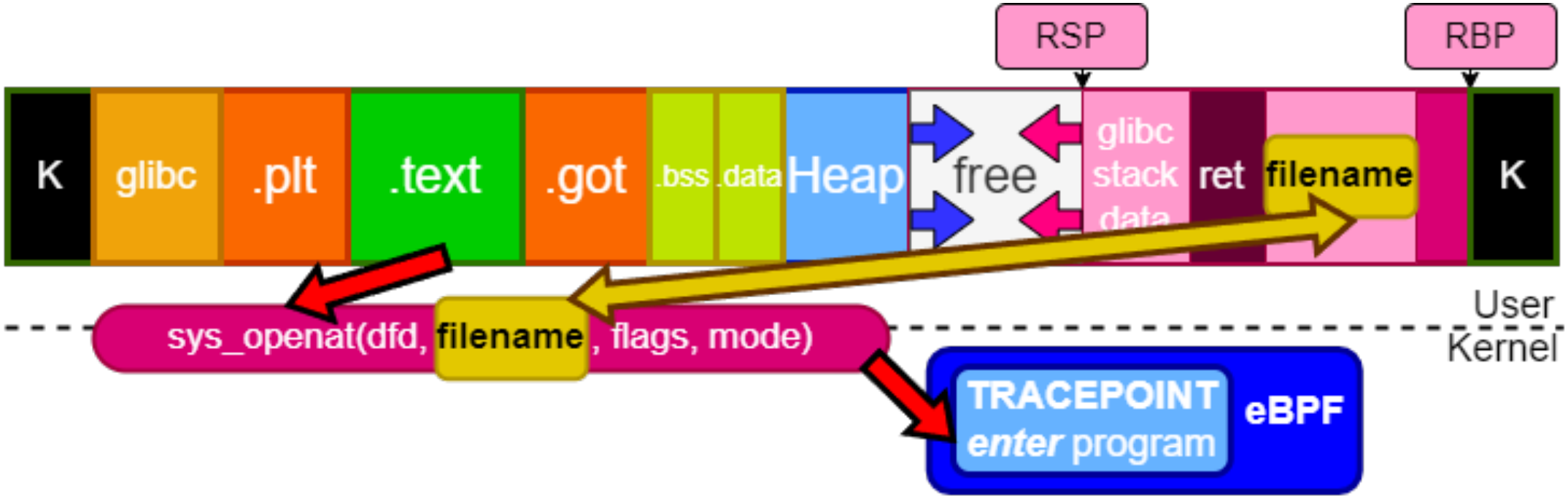
GOT HIJACKING



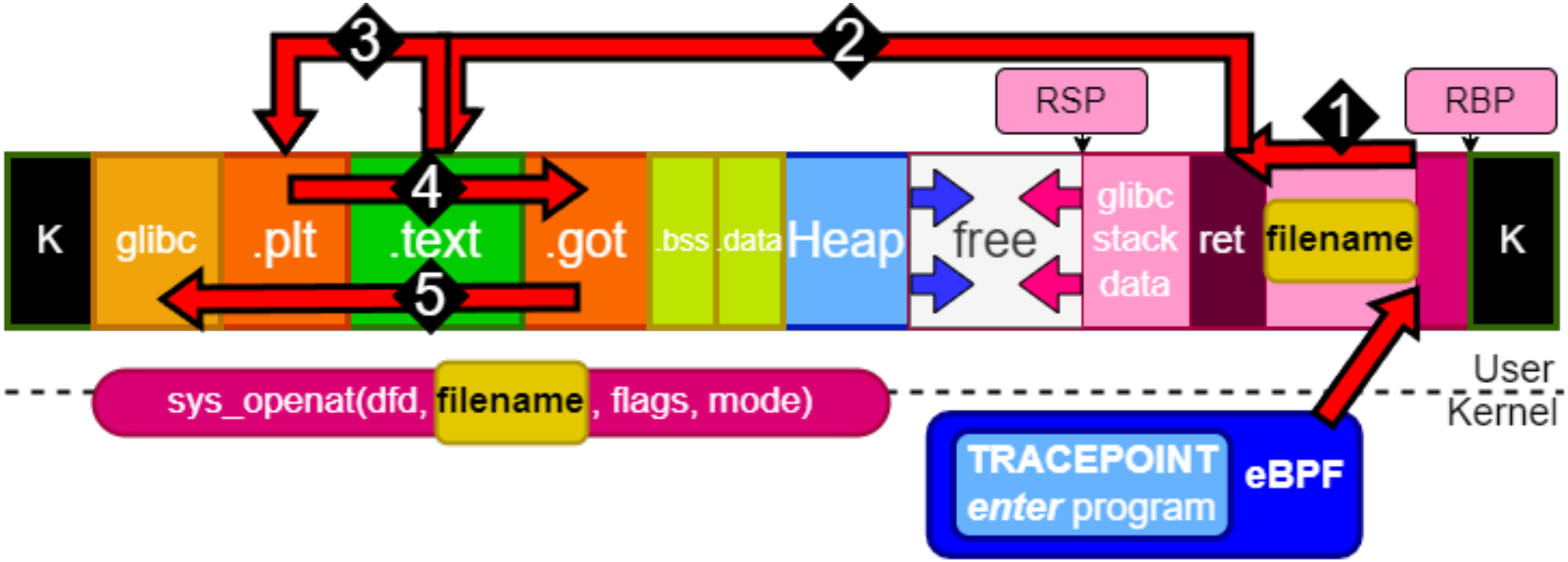
GOT HIJACKING



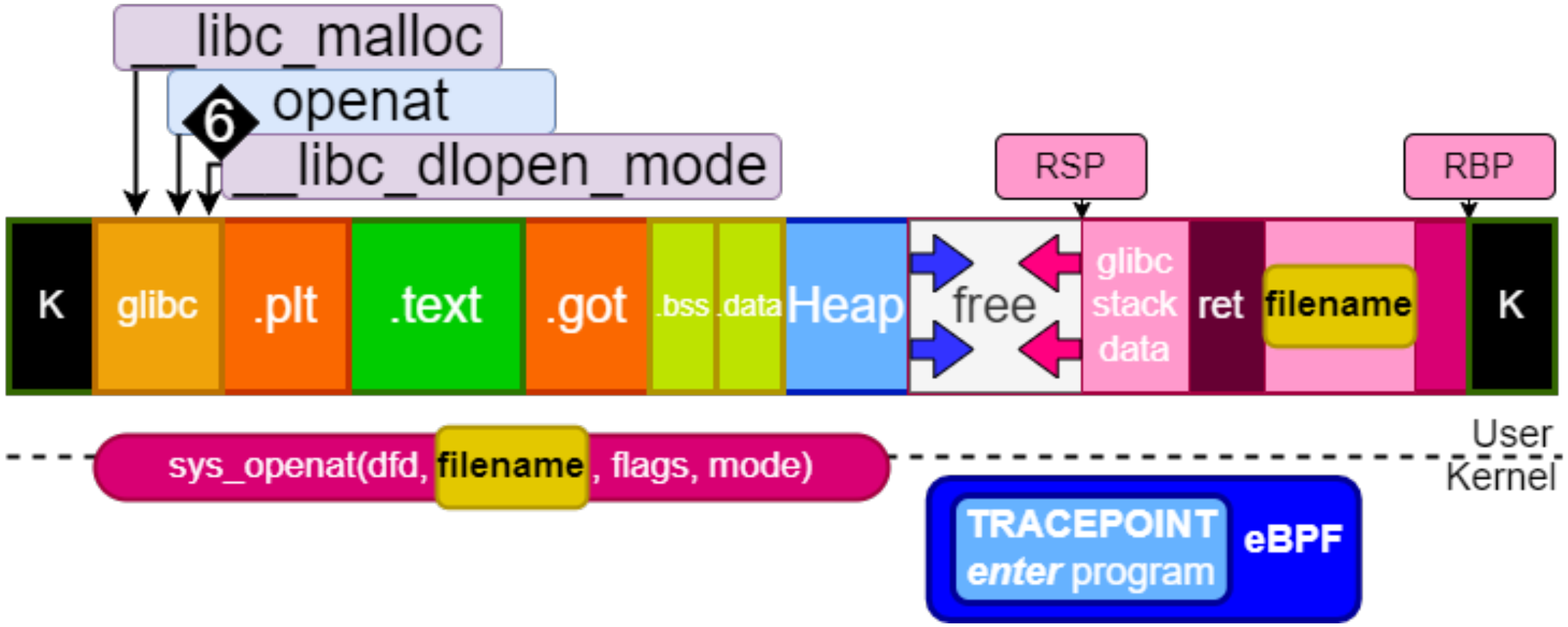
LIBRARY INJECTION



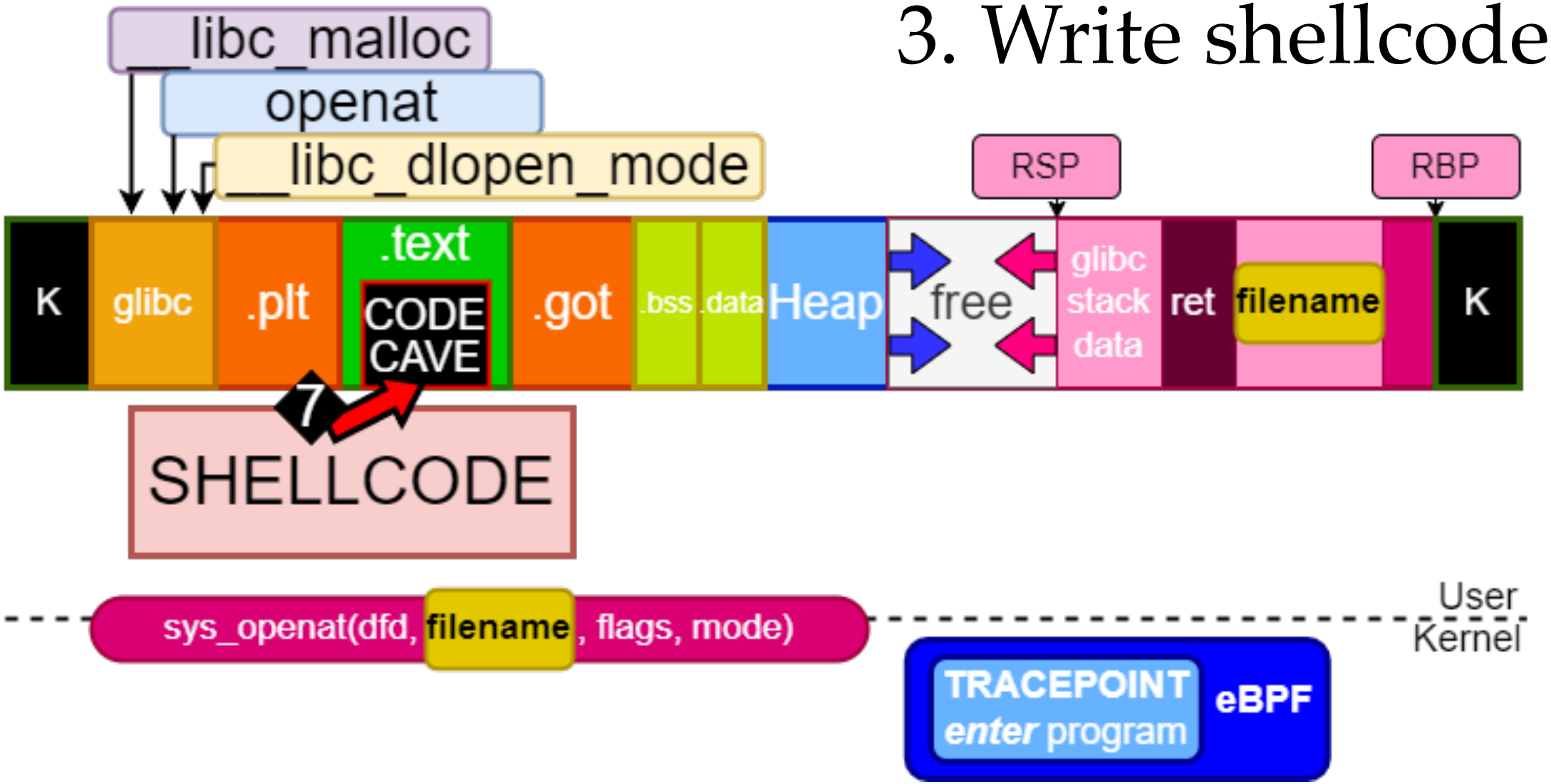
1. Memory scanning



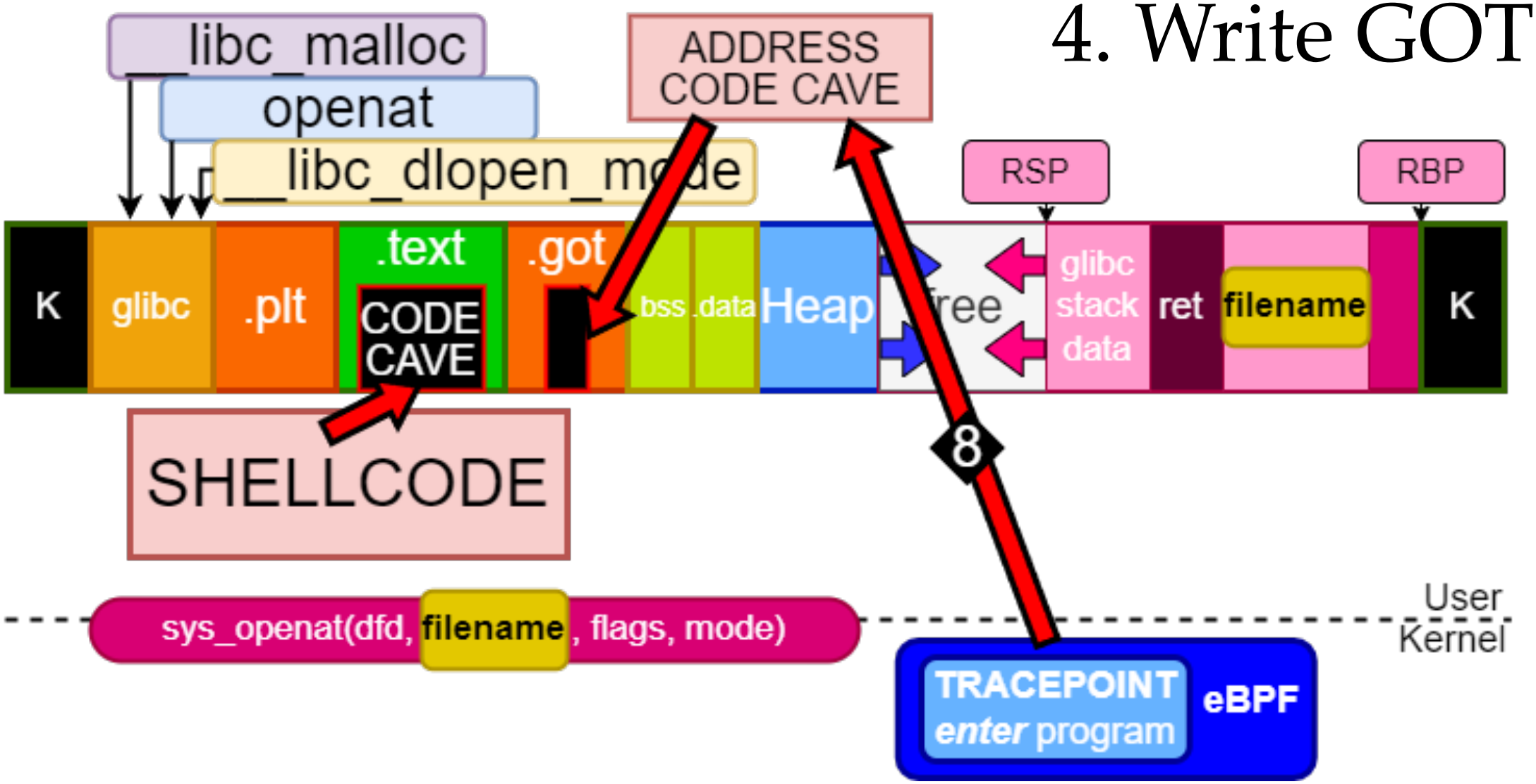
2. Locate key libc functions



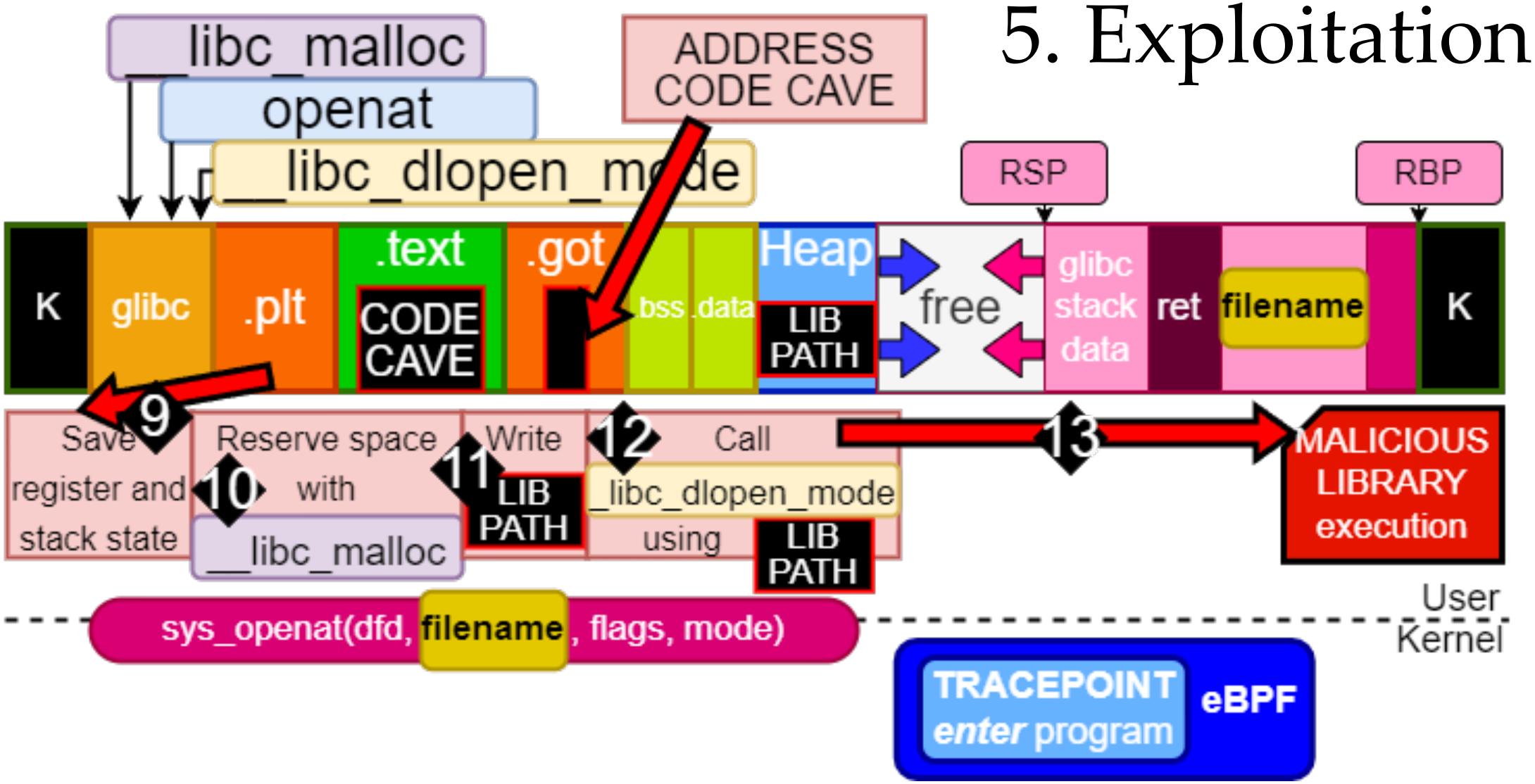
3. Write shellcode



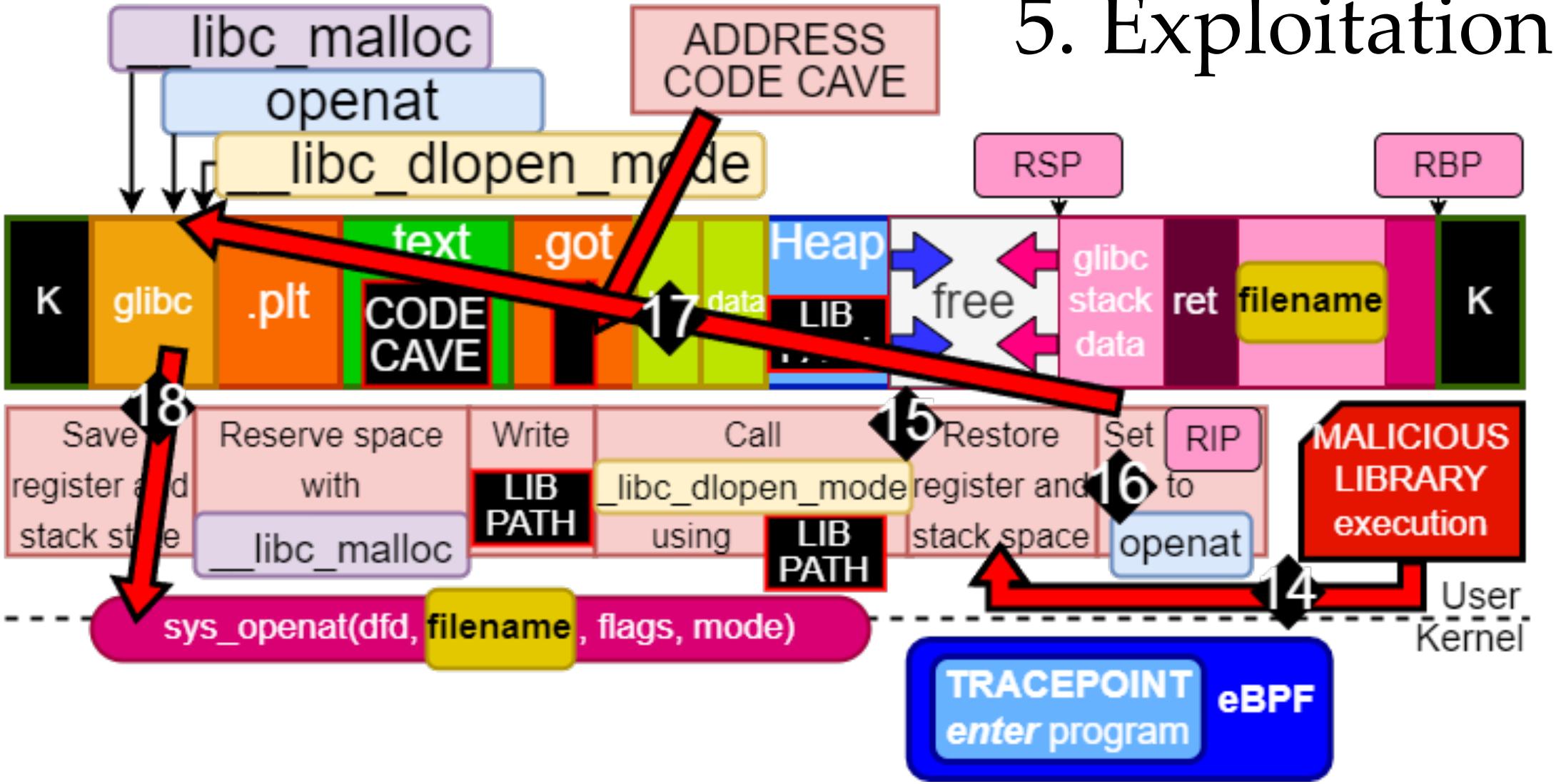
4. Write GOT


__libc_malloc
openat
__libc_dlopen_mode
ADDRESS
CODE CAVE
RSP
RBP
K
glibc
.plt
.text
CODE
CAVE
.got
bss
.data
Heap
free
glibc
stack
data
ret
filename
K
SHELLCODE
8
sys_openat(dfd, filename, flags, mode)
User
Kernel
TRACEPOINT
enter program
eBPF

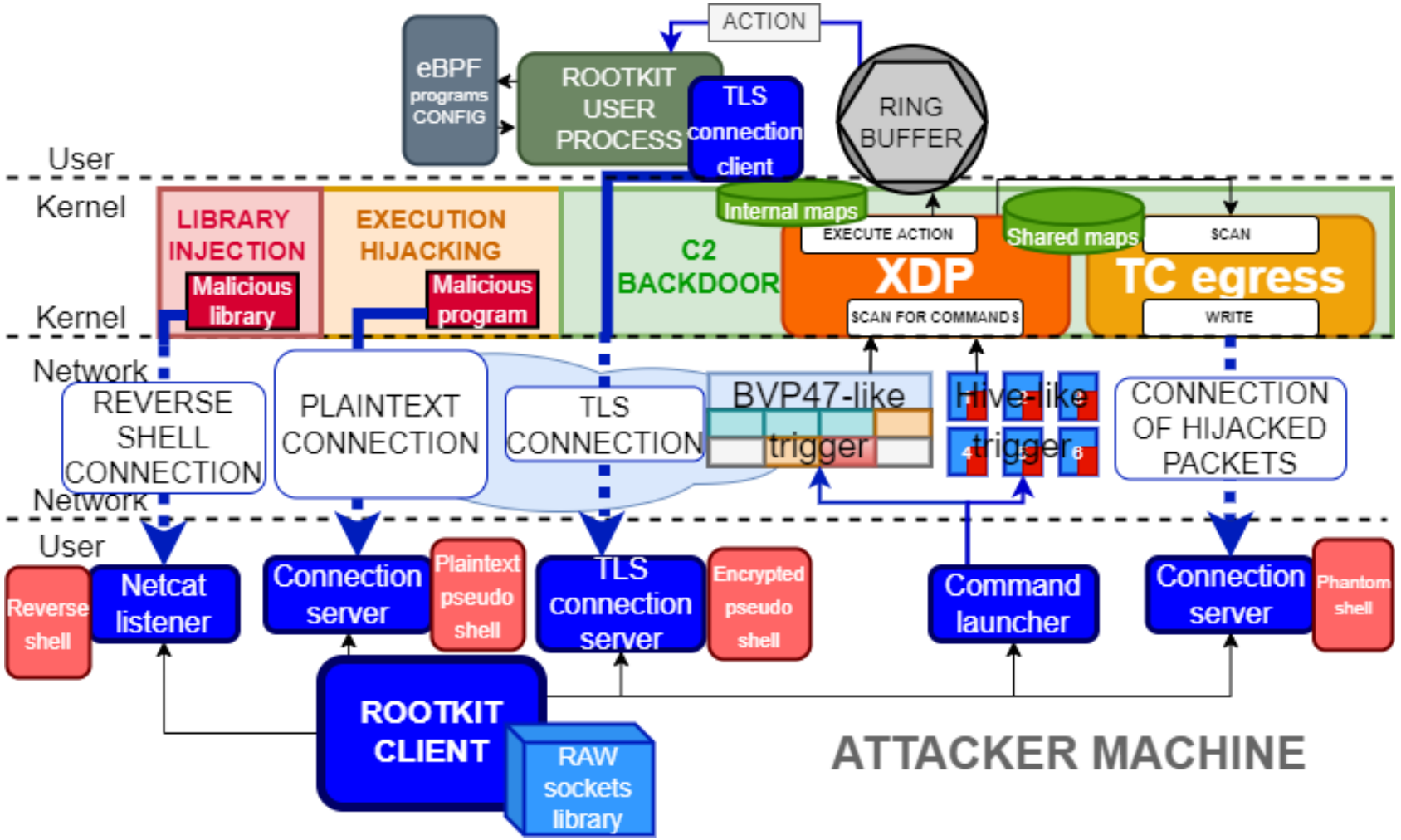
5. Exploitation



5. Exploitation



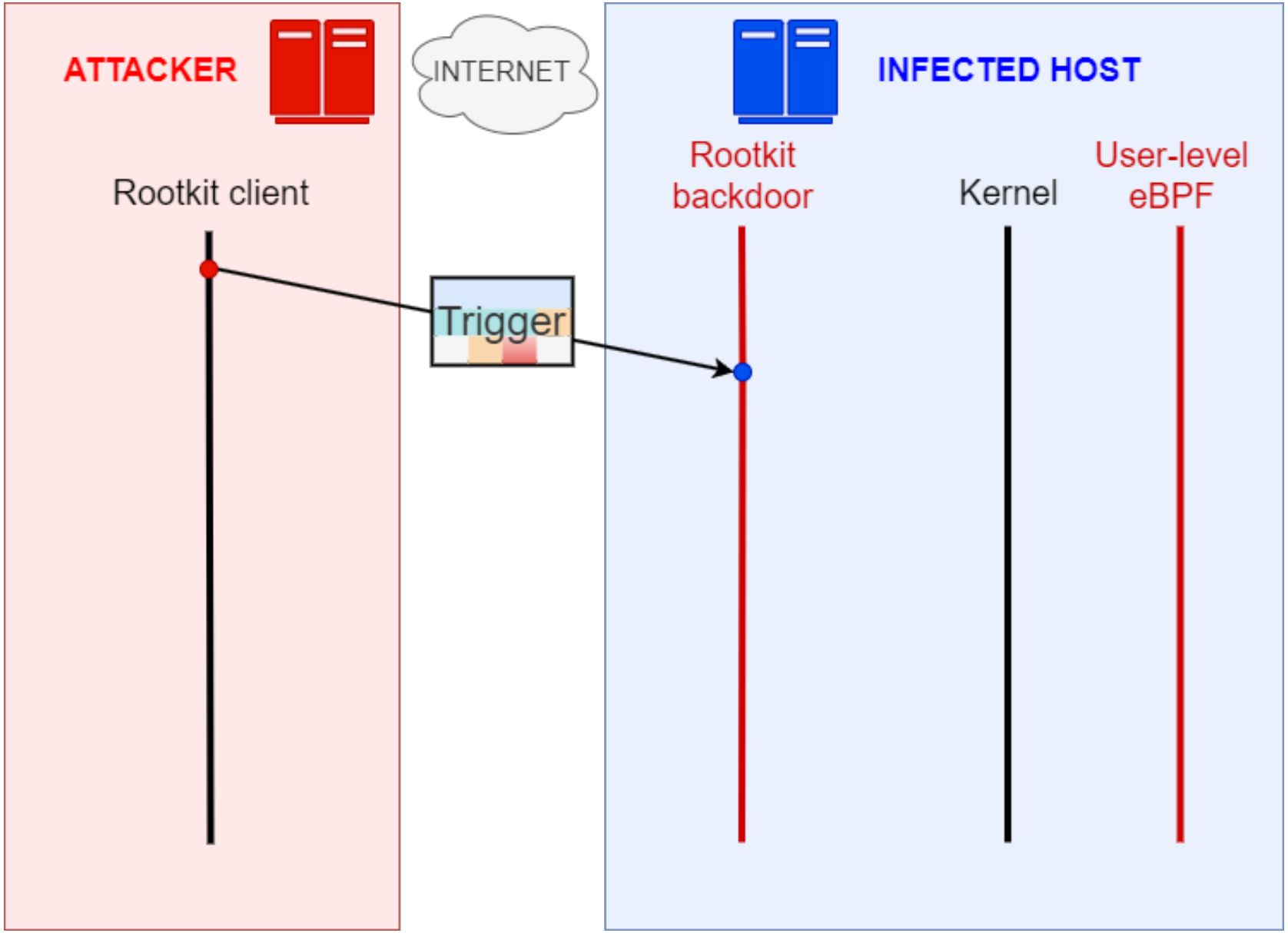
BACKDOOR & C2 SYSTEM



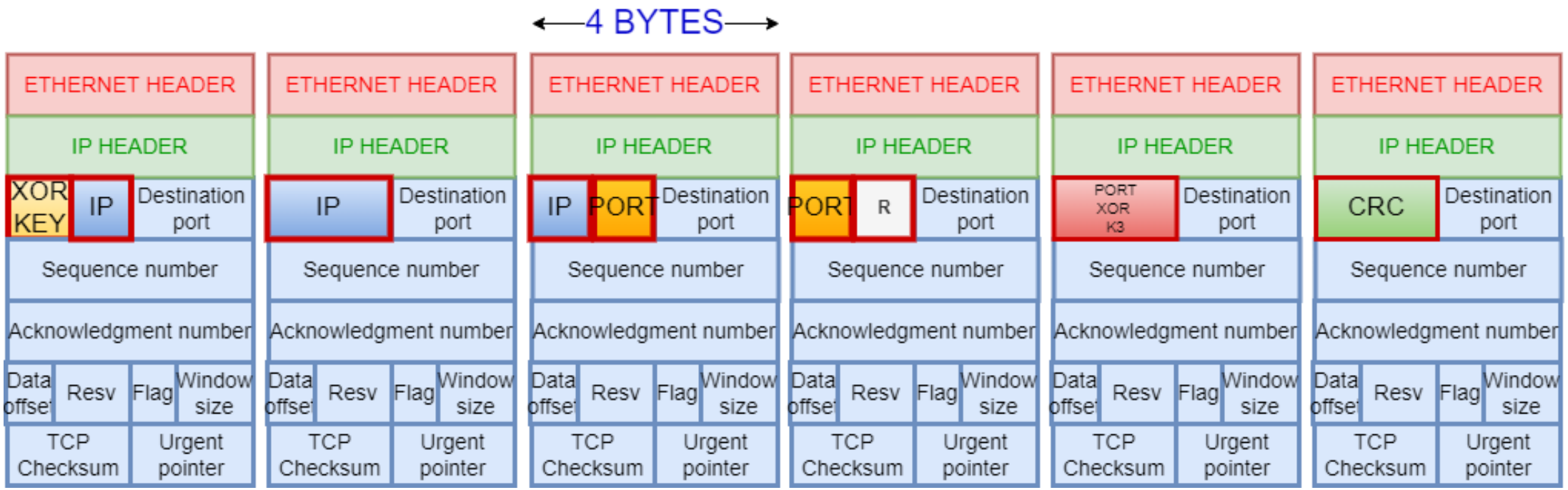
TripleCross rootkit

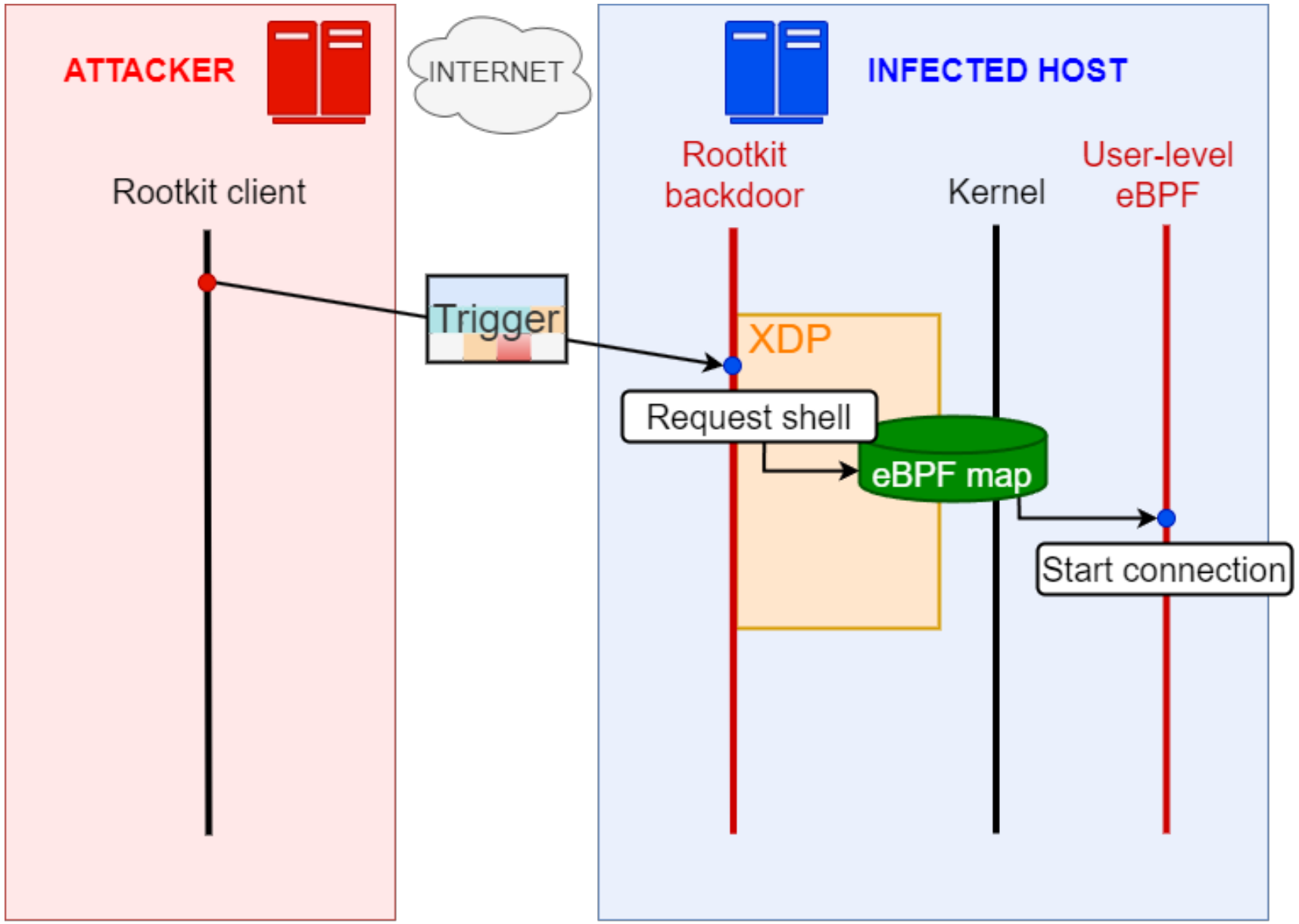
BACKDOOR & C2 SYSTEM

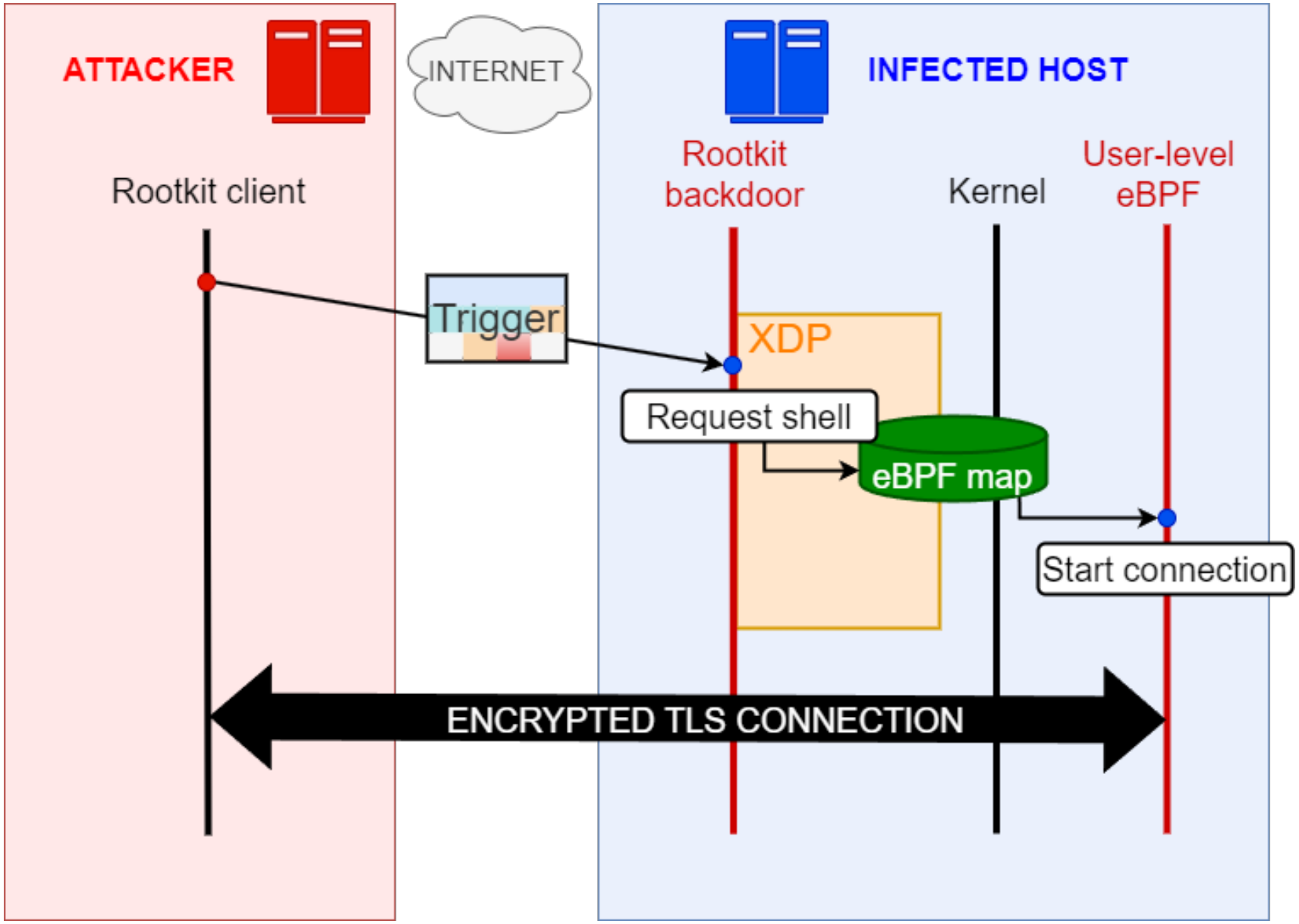
```
h3xduck@UbuntuTFG:~/TripleCross/src/client$ ./injector -h

Usage: ./injector OPTION victim_IP

Program OPTIONs
        -S IP                                   Send a secret message to IP (PoC)

        -c IP                                   Spawn plaintext pseudo-shell with IP - using execution hijacking

        -e IP                                   Spawn encrypted pseudo-shell with IP - with pattern-based trigger

        -s IP                                   Spawn encrypted pseudo-shell with IP - with multi-packet trigger

        -p IP                                   Spawn a phantom shell - with pattern-based trigger

        -a IP                                   Activate all of rootkit's hooks

        -u IP                                   Deactivate all of rootkit's hooks

        -h                                      Print this help
```

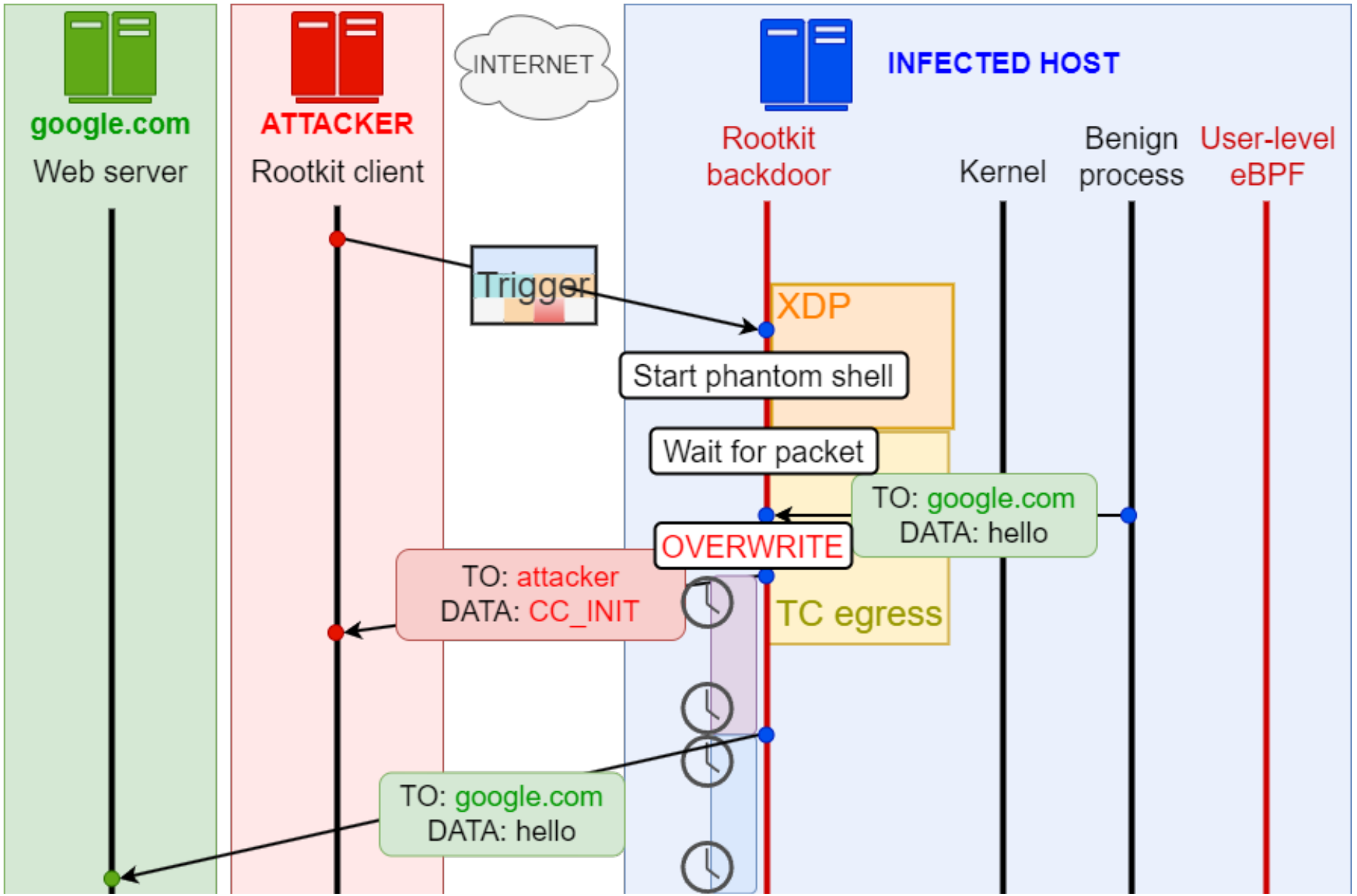

ATTACKER
INTERNET
INFECTED HOST
Rootkit client
Rootkit backdoor
Kernel
User-level eBPF
Trigger

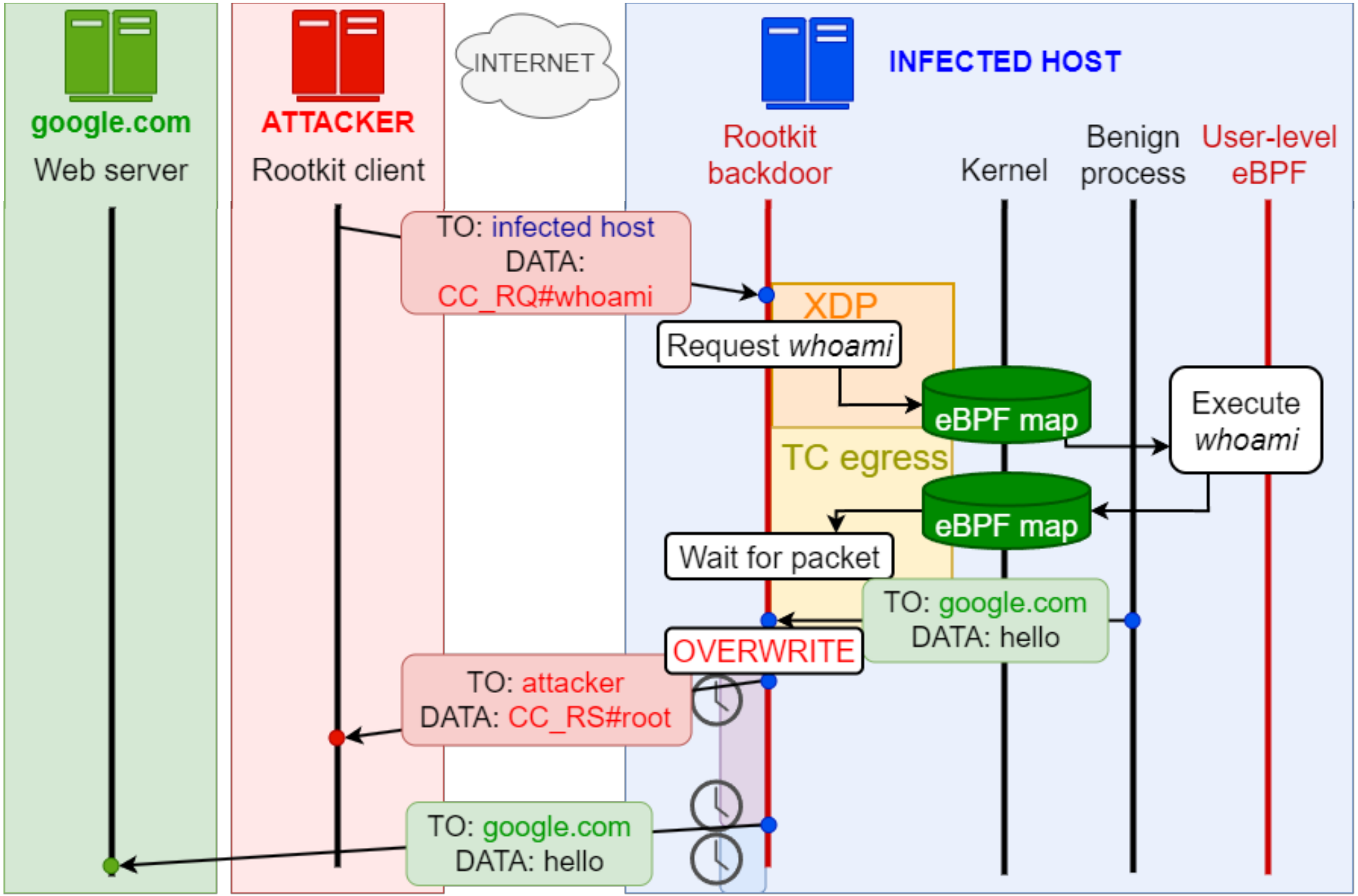
C2: TRIGGERS

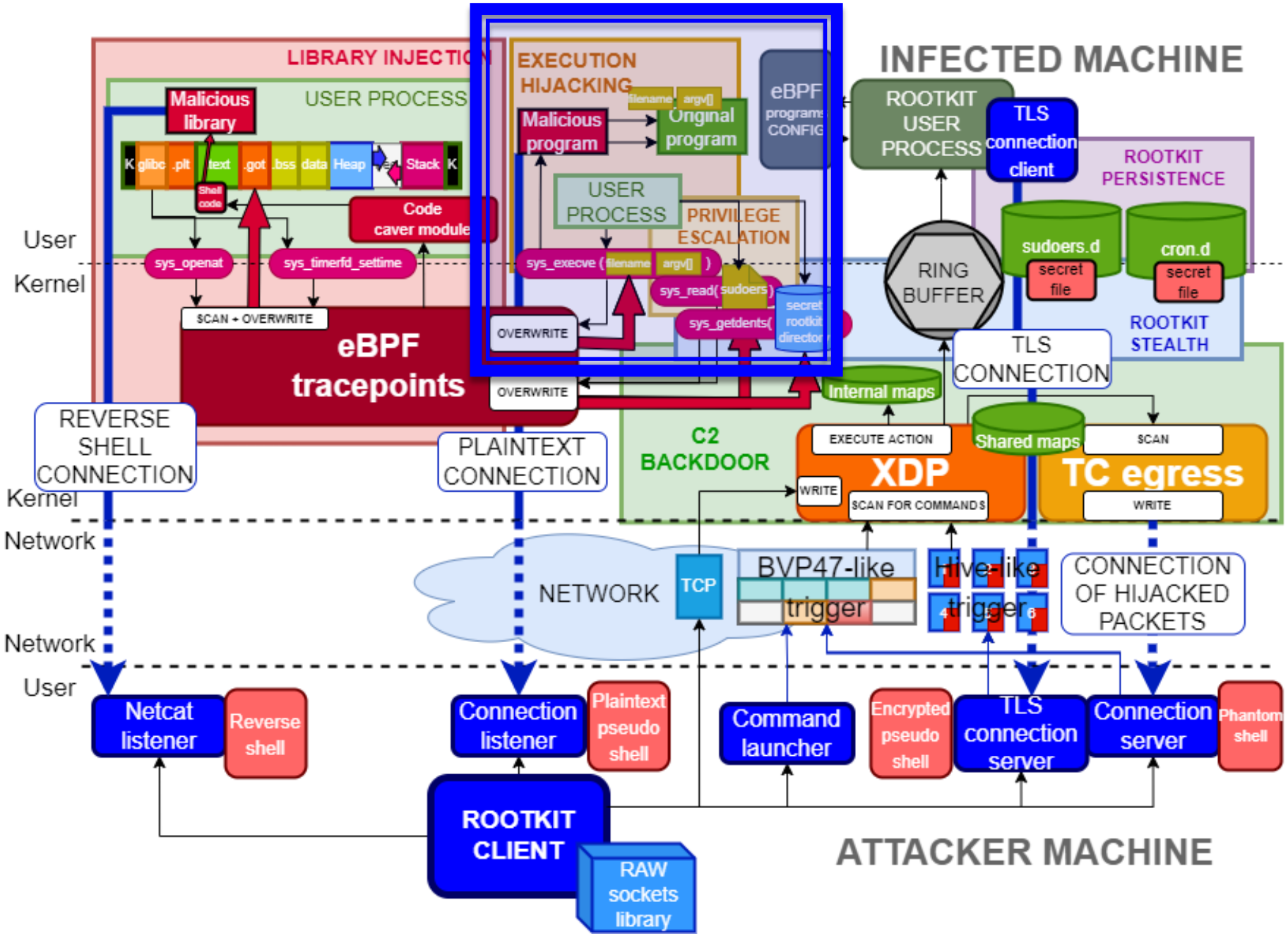

4 BYTES
ETHERNET HEADER
IP HEADER
XOR KEY
IP
Destination port
Sequence number
Acknowledgment number
Data offset
Resv
Flag
Window size
TCP Checksum
Urgent pointer
ETHERNET HEADER
IP HEADER
IP
Destination port
Sequence number
Acknowledgment number
Data offset
Resv
Flag
Window size
TCP Checksum
Urgent pointer
ETHERNET HEADER
IP HEADER
IP
PORT
Destination port
Sequence number
Acknowledgment number
Data offset
Resv
Flag
Window size
TCP Checksum
Urgent pointer
ETHERNET HEADER
IP HEADER
PORT
R
Destination port
Sequence number
Acknowledgment number
Data offset
Resv
Flag
Window size
TCP Checksum
Urgent pointer
ETHERNET HEADER
IP HEADER
PORT XOR K3
Destination port
Sequence number
Acknowledgment number
Data offset
Resv
Flag
Window size
TCP Checksum
Urgent pointer
ETHERNET HEADER
IP HEADER
CRC
Destination port
Sequence number
Acknowledgment number
Data offset
Resv
Flag
Window size
TCP Checksum
Urgent pointer


ATTACKER
INTERNET
INFECTED HOST
Rootkit client
Rootkit backdoor
Kernel
User-level eBPF
Trigger
XDP
Request shell
eBPF map
Start connection


ATTACKER
INTERNET
INFECTED HOST
Rootkit client
Rootkit backdoor
Kernel
User-level eBPF
Trigger
XDP
Request shell
eBPF map
Start connection
ENCRYPTED TLS CONNECTION

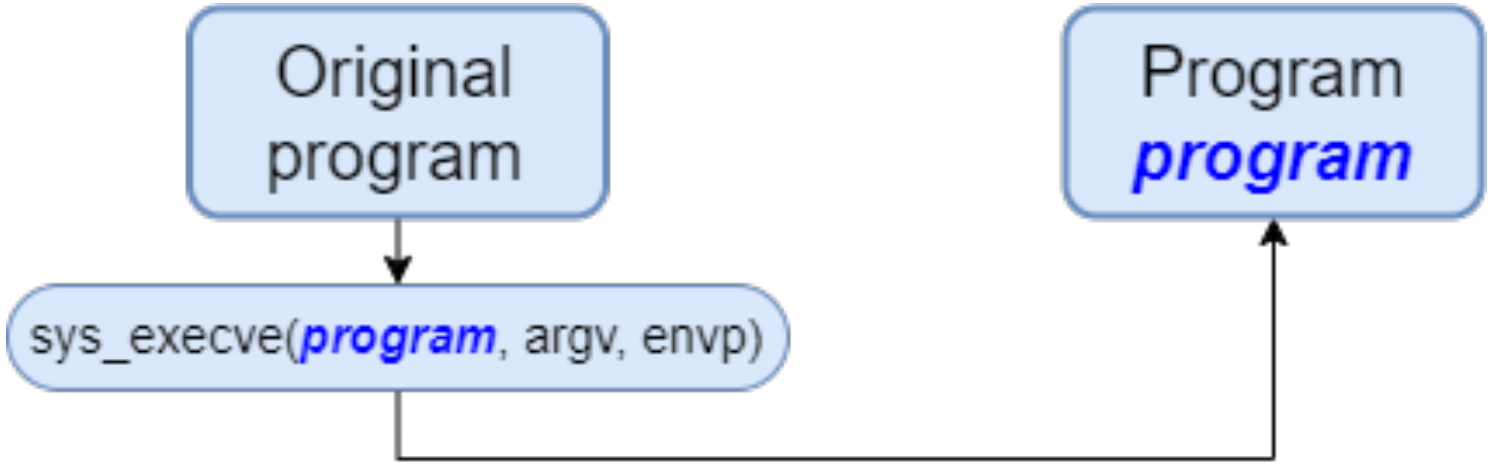

google.com
Web server
ATTACKER
Rootkit client
INTERNET
INFECTED HOST
Rootkit backdoor
Kernel
Benign process
User-level eBPF
Trigger
XDP
Start phantom shell
Wait for packet
TO: google.com
DATA: hello
OVERWRITE
TO: attacker
DATA: CC_INIT
TC egress
TO: google.com
DATA: hello


google.com
Web server
ATTACKER
Rootkit client
INTERNET
INFECTED HOST
Rootkit backdoor
Kernel
Benign process
User-level eBPF
TO: infected host
DATA:
CC_RQ#whoami
XDP
Request whoami
eBPF map
Execute whoami
TC egress
eBPF map
Wait for packet
TO: google.com
DATA: hello
OVERWRITE
TO: attacker
DATA: CC_RS#root
TO: google.com
DATA: hello



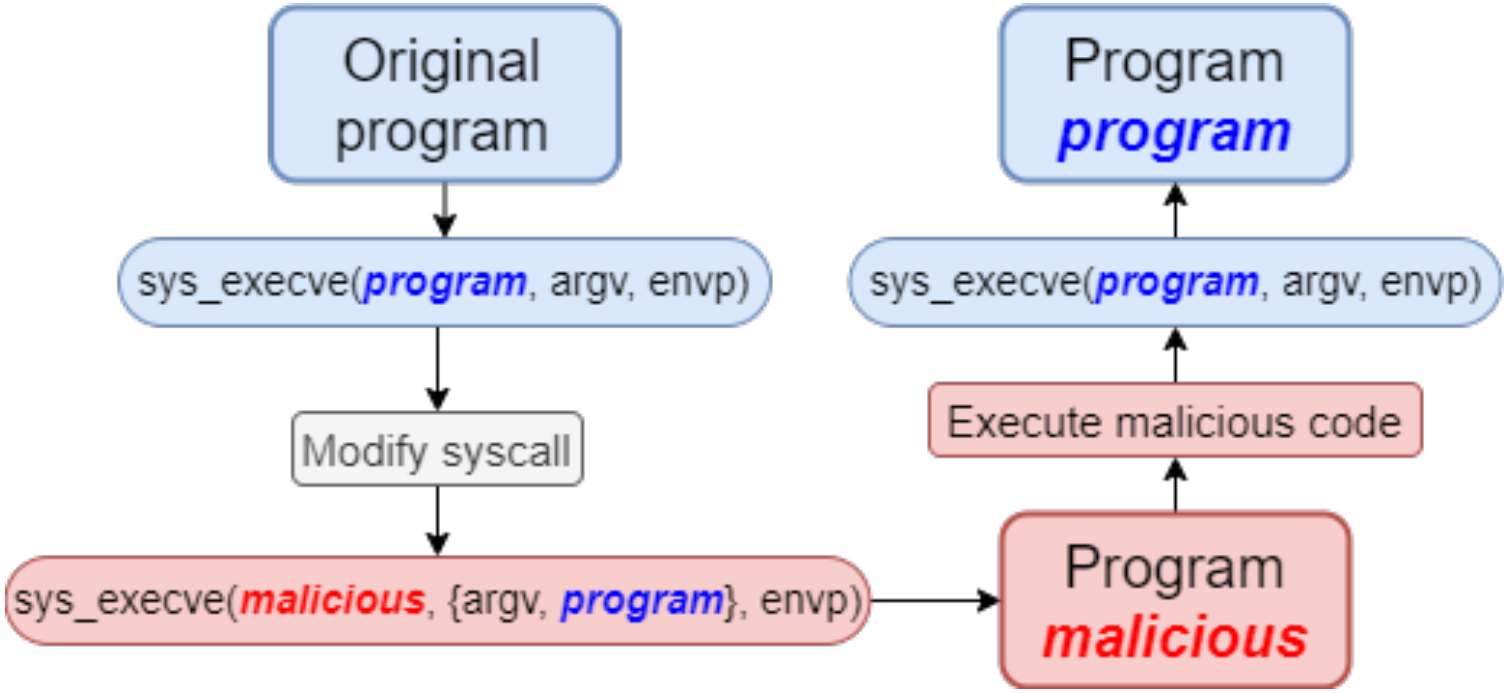
TripleCross rootkit

EXECUTION HIJACKING



We cannot execute new programs from the kernel using eBPF…

EXECUTION HIJACKING



We cannot execute new programs from the kernel using eBPF…

But we can hijack any program execution.

PRIVILEGE ESCALATION

```
Defaults        env_reset
Defaults        mail_badpass
Defaults        secure_path="/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/snap/bin"

# Host alias specification

# User alias specification

# Cmnd alias specification

# User privilege specification
root    ALL=(ALL:ALL) ALL

# Members of the admin group may gain root privileges
%admin ALL=(ALL) ALL

# Allow members of group sudo to execute any command
%sudo   ALL=(ALL:ALL) ALL

# See sudoers(5) for more information on "@include" directives:

@includedir /etc/sudoers.d
```

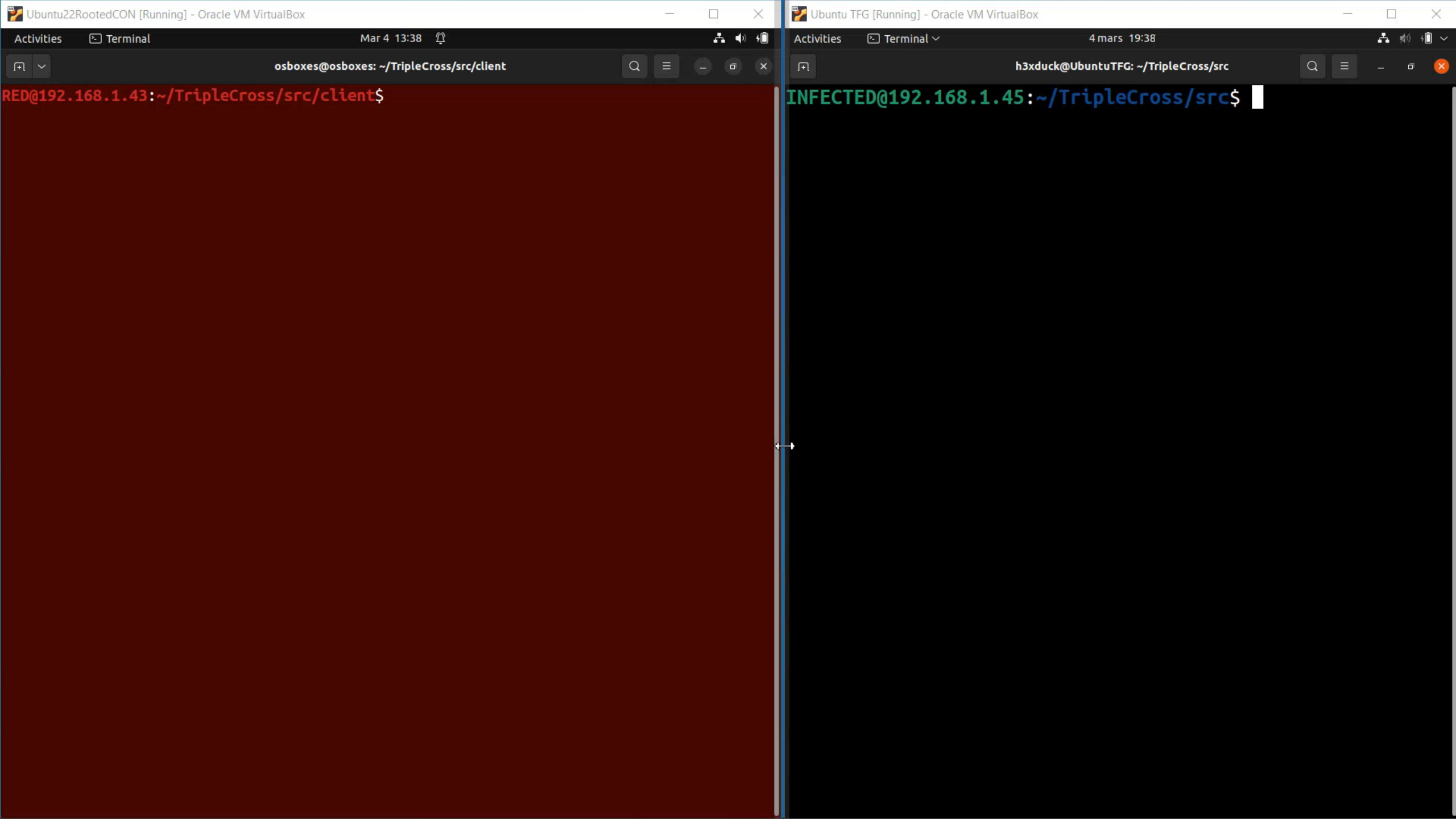
TripleCross rootkit

PRIVILEGE ESCALATION

```
<USER> ALL=(ALL:ALL) NOPASSWD:ALL ##################################################
##################################################################################
##################################################################################
##################################################################################
##################################################################################
##################################################################################
##################################################################################
##################################################################################
##################################################################################
##################################################################################
##################################################################################
##################################################################################
##################################################################################
##################################################################################
##################################################################################
##################################################################################
```

TripleCross rootkit

Demo


Ubuntu22RootedCON [Running] - Oracle VM VirtualBox
Activities
Terminal
Mar 4 13:38
osboxes@osboxes: ~/TripleCross/src/client
RED@192.168.1.43:~/TripleCross/src/client$
Ubuntu TFG [Running] - Oracle VM VirtualBox
Activities
Terminal
4 mars 19:38
h3xduck@UbuntuTFG: ~/TripleCross/src
INFECTED@192.168.1.45:~/TripleCross/src$

Ubuntu22RootedCON [Running] - Oracle VM VirtualBox

Activities Terminal Mar 4 20:16

osboxes@osboxes: ~/TripleCross/src/client

```
RED@192.168.1.43:~/TripleCross/src/client$
```

Ubuntu TFG [Running] - Oracle VM VirtualBox

Activities Visual Studio Code 5 mars 02:16

constants.h - TripleCross - Visual Studio Code

File Edit Selection View Go Run Terminal Help

constants.h M | simple_open.c M | kit.c M

src > common > constants.h > CODE_CAVE_ADDRESS_STATIC

```
20  #define SECRET_DIRECTORY_NAME_HIDE "SECRETDIR"
21  #define SECRET_FILE_PERSISTENCE_NAME "ebpfbackdoor"
22
23  // EXECUTION HIJACKING
24  #define PATH_EXECUTION_HIJACK_PROGRAM "/home/osboxes/TFG/src/helpers/execve
25  #define EXEC_HIJACK_ACTIVE 0 // 0 Deactivated, 1 active
26  #define TASK_COMM_NAME_RESTRICT_HIJACK "bash"
27  #define TASK_COMM_RESTRICT_HIJACK_ACTIVE 1
28
29  // LIBRARY INJECTION
30  #define TASK_COMM_NAME_INJECTION_TARGET_TIMERFD_SETTIME ""
31  #define TASK_COMM_FILTER 1 // 0 do not filter by task. 1 filter by task.
32  #define TASK_COMM_NAME_INJECTION_TARGET_OPEN "simple_open"
33
34  #define CODE_CAVE_ADDRESS_STATIC 0x00000000004012c4
35  #define CODE_CAVE_SHELLCODE_ASSEMBLE_1 \
36      "\x55\x50\x51\x52\x53\x57\x56\
37  \xbf\x00\x20\x00\x00\x48\xbb"
38  #define CODE_CAVE_SHELLCODE_ASSEMBLE_1_LEN 14
39
40  #define CODE_CAVE_SHELLCODE_ASSEMBLE_2 \
41      "\xff\xd3\x48\x89\xc3\xc7\x00\x2f\x68\x6f\x6d\
42  \xc7\x40\x04\x65\x2f\x6f\x73\xc7\x40\x08\x62\x6f\x78\
43  \x65\xc7\x40\x0c\x73\x2f\x54\x46\xc7\x40\x10\x47\x2f\
44  \x73\x72\xc7\x40\x14\x63\x2f\x68\x65\xc7\x40\x18\x6c\
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL

```
make[2]: 'simple_execve' is up to date.
INFECTED@192.168.1.45:~/TripleCross/src$
```

```
printk: Phantom shell NOT active right now
```

bash src
sudo src
bash

More demos?

github.com/h3xduck/TripleCross

OTHER ROOTKIT MODULES

1. Persistence

Keep the rootkit installed across reboots

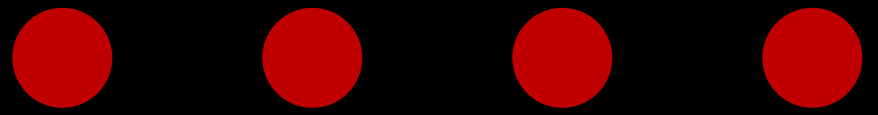
Using cron jobs

2. Stealth

Hide rootkit files and activity from the user

Modifying of sys_getdents() syscall

4 Defenses & conclusions



DEFENSES

General

- EDR / network appliances
- May still be deceived

Monitor eBPF

- Available tools to monitor bpf() activity

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

Least-privilege eBPF

- Rootkits require privileged eBPF
- Use eBPF capabilities

Kernel lockdown

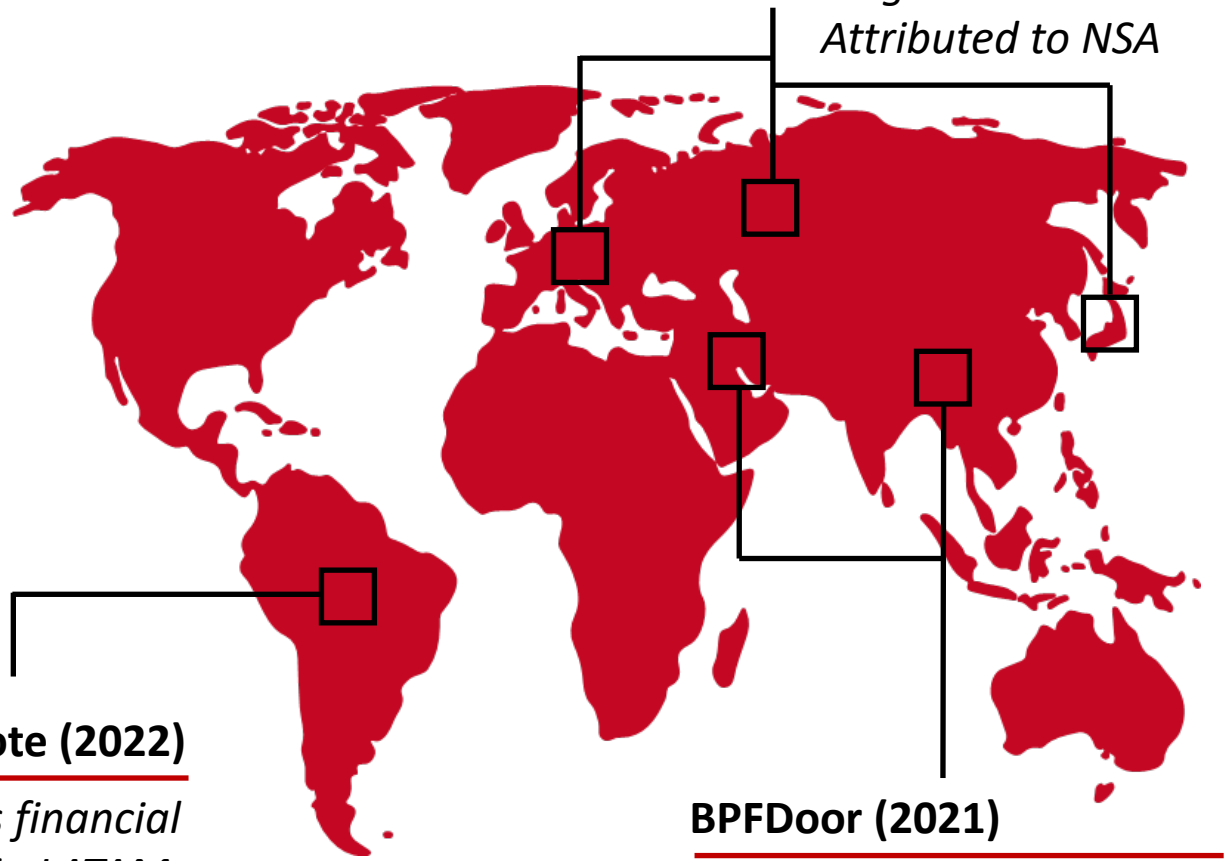
- Prevents altering the kernel
- Default *integrity* mode if secure boot active: backdoors still possible

OFFENSIVE eBPF

DEFCON 29 (2021): Warping reality
New offensive eBPF techniques

DEFCON 27 (2019): Evil eBPF
First discussion of offensive capabilities

DEFCON 29 (2021): ebpfKit
First public eBPF rootkit



BVP47 (2022)

Targets mil and telco
Attributed to NSA

Symbiote (2022)

Targets financial sector in LATAM

BPFDoor (2021)

Targets telco
Attributed to Chinese actor

TripleCross
A Linux eBPF rootkit

Marcos Bajo
@h3xduck

Juan Tapiador
@0xjet

github.com/h3xduck/TripleCross

/RootedCON