

BOOTKITTY: Multi-OS Trust Chain compromise from Bootkit to Rootkit

Junho Lee | HyunA Seo

About Speakers



Junho Lee (BlackCat)

- B.S. at Mokpo National University
- Low-level systems research (from hardware to kernel)
- x.com @B1ack3at



HyunA Seo (yuseong)

- B.S. at Sungshin Women's University
- Interested in Reverse Engineering, System Hacking, and Hardware Research
- yuseong@catn1p.com (yuseong@gmail.com)

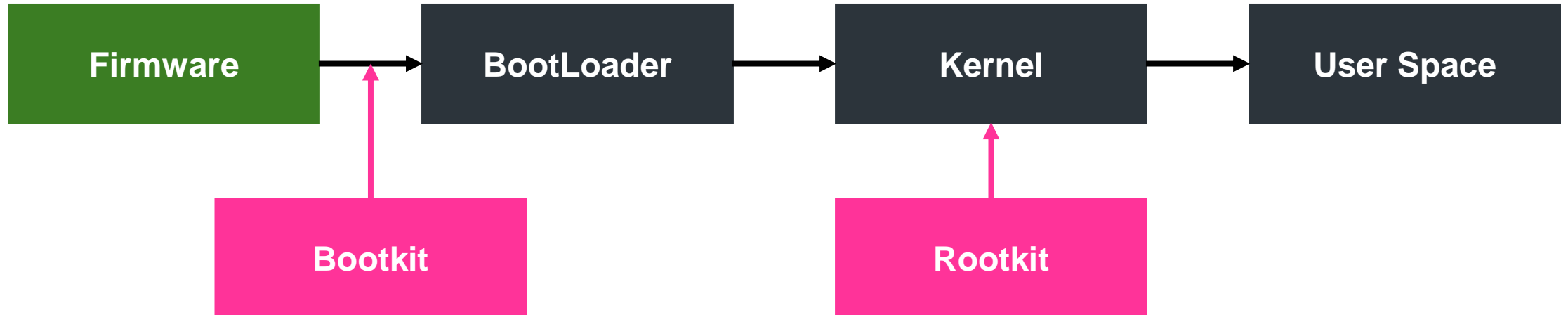
Our Team

- **Jihoon Kwon**
 - B.S. at Korea University
- **Myeongyeol Lee**
 - B.S. at Chosun University
- **Hyungyu Seo**
 - B.S. at Keimyung University
- **Jinho Jung, Youngjin Sim, Sumin Hwang**
 - Advisors

Introduction to Bootkits & Rootkits

Bootkits & Rootkits

- An advanced malware that hijacks a boot process
 - Takes control of the system before the operating system launches
- **Bootkit vs. Rootkit**



Bootkits as Persistent Threats

- Bootkits have been actively studied with annual reports

ESET Research

BlackLotus UEFI bootkit: Myth confirmed

The first in-the-wild UEFI bootkit bypassing UEFI Secure Boot on fully updated UEFI systems is now a reality

Bootkitty and Linux Bootkits: We've Got You Covered

By: Paul Asadoorian | December 4, 2024

More elusive and more persistent: the third known firmware bootkit shows major advancement

January 21, 2022

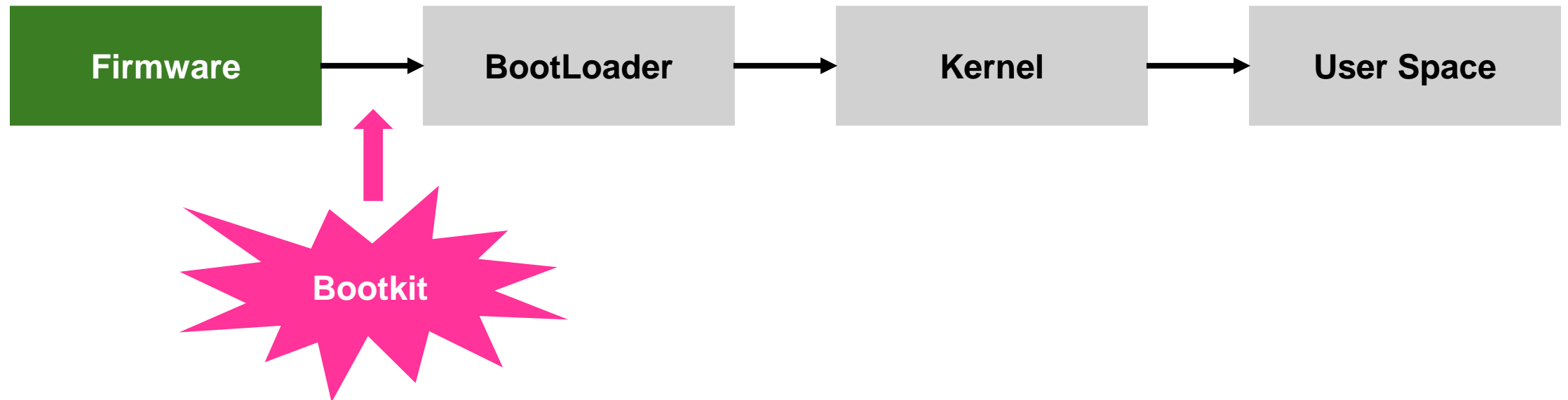
Kaspersky's researchers have uncovered the third case of a firmware bootkit in the wild. Dubbed MoonBounce, this malicious implant is hidden within a computer's Unified Extensible Firmware Interface (UEFI) firmware, an essential part of computers, in the SPI flash, a storage component external to the hard drive. Such implants are notoriously difficult to remove and are of limited visibility to security products. Having first appeared in the wild in the spring of 2021, MoonBounce demonstrates a sophisticated attack flow, with evident advancement in comparison to formerly reported UEFI firmware bootkits. Kaspersky researchers have attributed the attack with considerable confidence to the well-known advanced persistent threat (APT) actor APT41.

LoJax UEFI Rootkit Used in Cyberespionage

October 01, 2018

Why is Dangerous?

- **Stealth:** Attacks during pre-booting, bypassing anti-viruses or EDRs
- **Persistence:** Implants into a boot partition or firmware
- **Compromise:** Controls the system before kernel initialization



Background

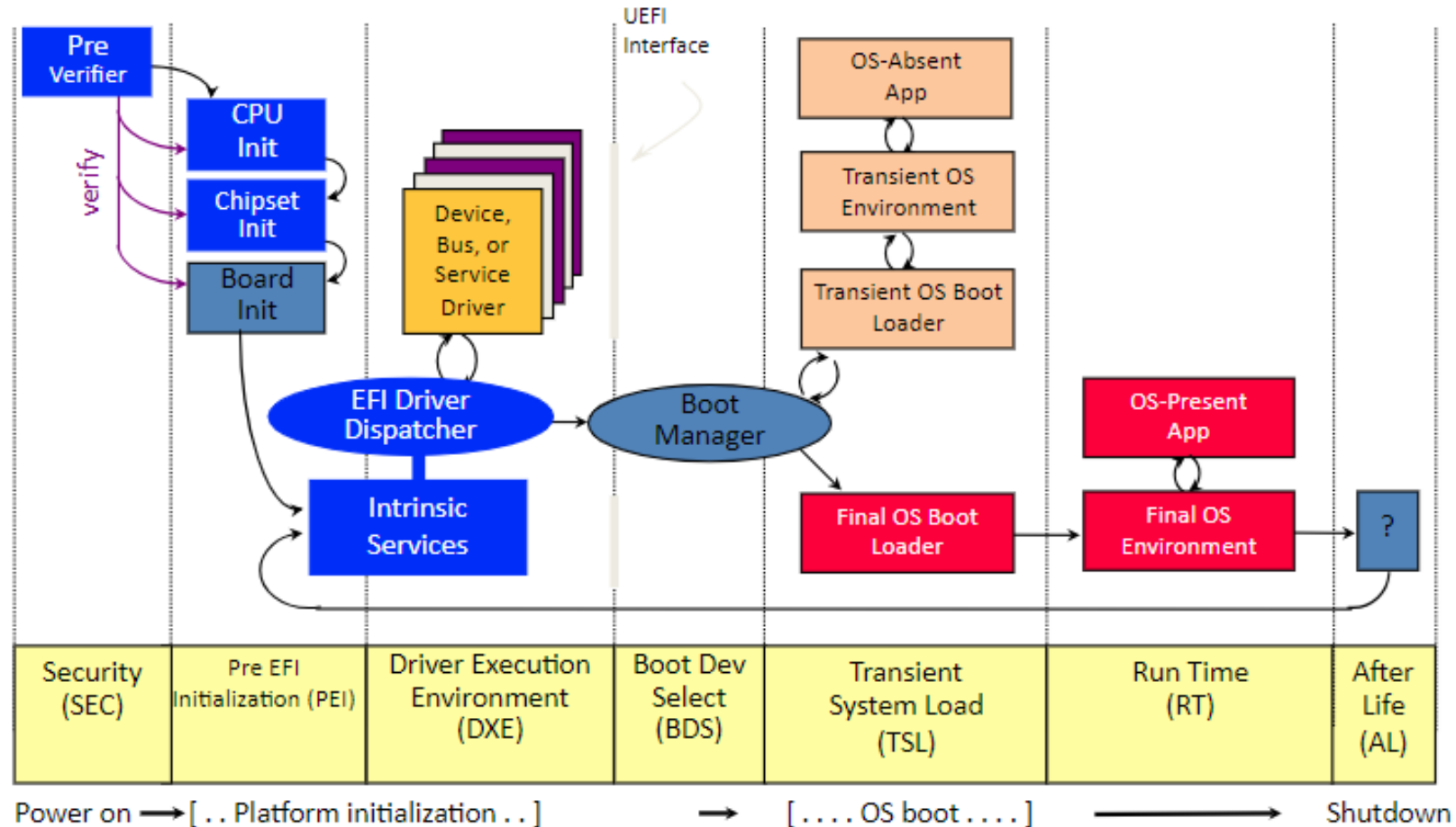
What is Unified Extensible Firmware Interface (UEFI)?

- UEFI replaces legacy BIOS, handling hardware initialization and the OS boot process.
- Its modular, extensible design enables faster, more reliable startup and supports 64-bit systems and GPT drives.



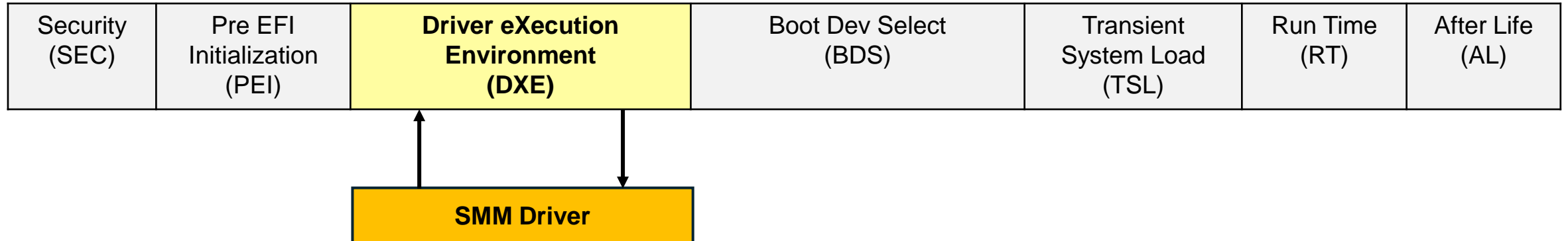
Basic boot process on UEFI

- Architecture Execution Flow



UEFI components: DXE Phase

- DXE loads PE/COFF drivers to initialize the platform and provide boot and runtime services.
- DXE includes System Management Mode, a highest-privilege firmware mode backed by SMRAM and SMI handlers.
- If DXE drivers are vulnerable or SMM is exploited, attackers can install stealthy, persistent Bootkits.



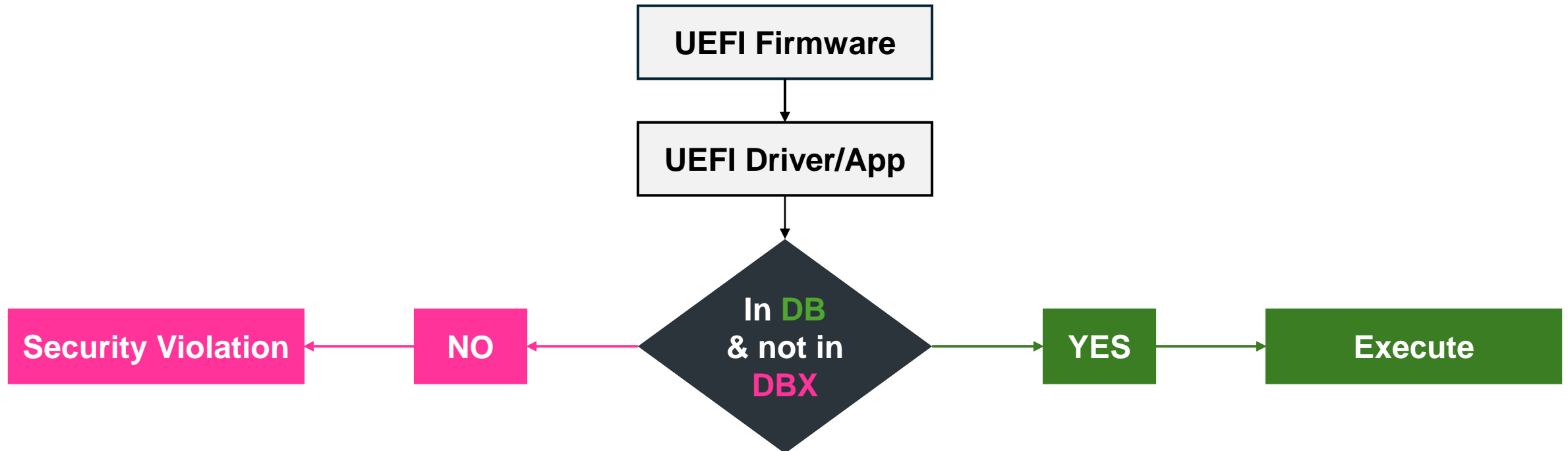
UEFI components: BDS Phase

- BDS selects and launches the EFI application (.efi) from the ESP per BootOrder.
- Boot loaders run from the ESP, and the operating system can access and modify its contents.
- Installing a Bootkit via ESP file replacement or NVRAM changes is relatively easy but leaves artifacts, reducing stealth.

Security (SEC)	Pre EFI Initialization (PEI)	Driver eXecution Environment (DXE)	Boot Dev Select (BDS)	Transient System Load (TSL)	Run Time (RT)	After Life (AL)
-------------------	------------------------------------	--	----------------------------------	-----------------------------------	------------------	--------------------

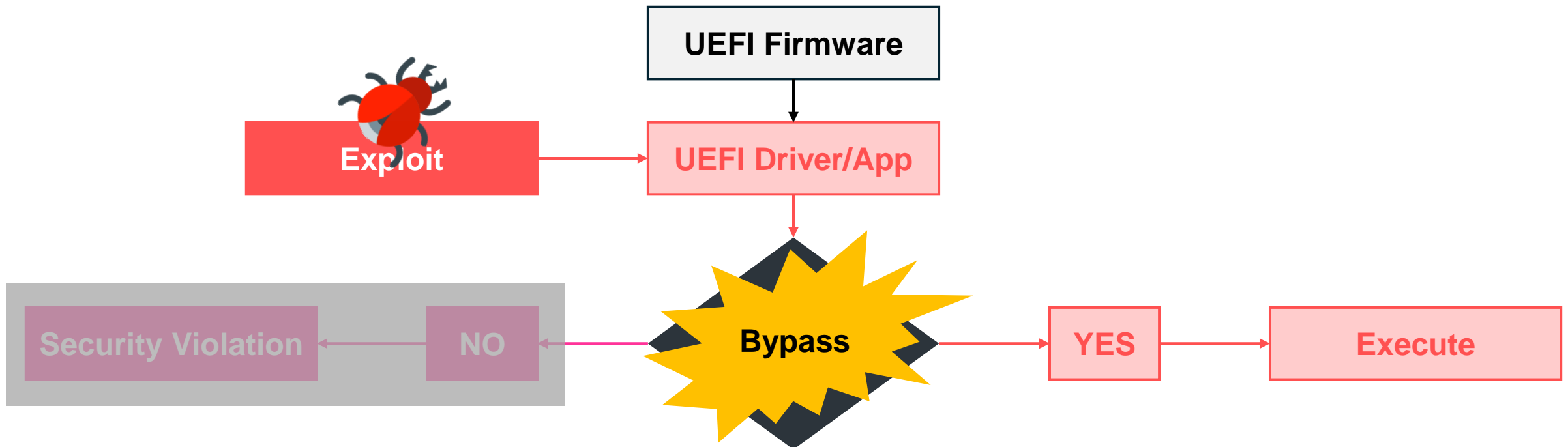
Existing Defense: Secure Boot

- Load & Verify against DB & DBX
 - DB : list of Allowed certificates or (PE authenticode) hashes
 - DBX : list of Forbidden certificates or (PE authenticode) hashes



Limitations of Secure Boot: Not Fully Secure

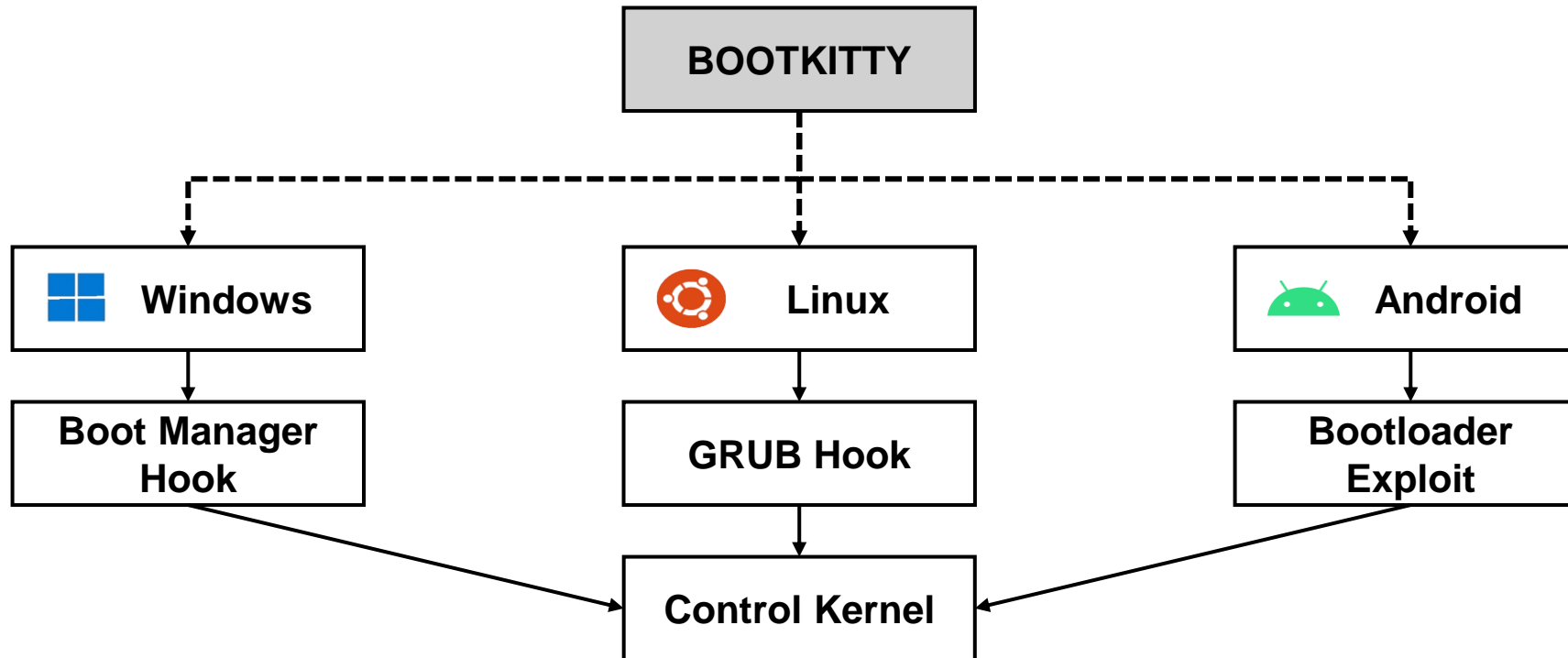
- Secure boot has vulnerabilities in firmware and certificate management
- These vulnerabilities can be leveraged to bypass Secure Boot



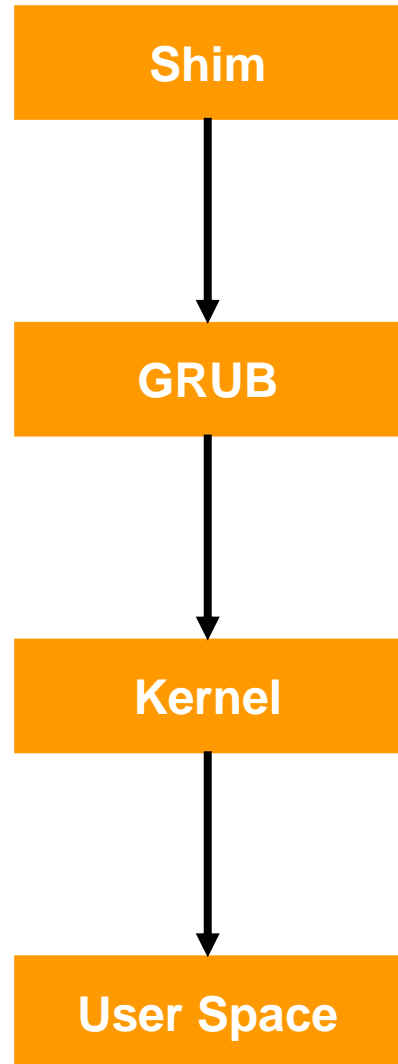
Design of BOOTKITTY for Multi-OS (Windows, Linux, Android)

BOOTKITTY: Hybrid Bootkit-Rootkit

- BOOTKITTY hijacks the boot process and injects a stealth rootkit

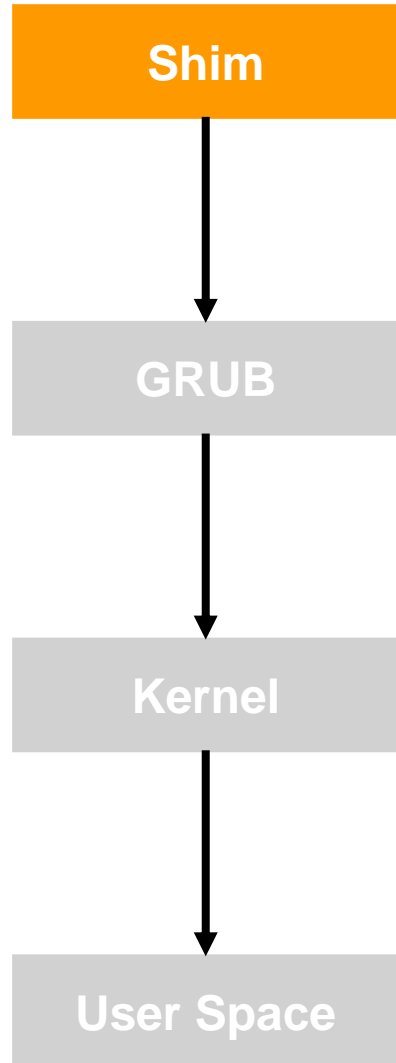


Linux Boot Process



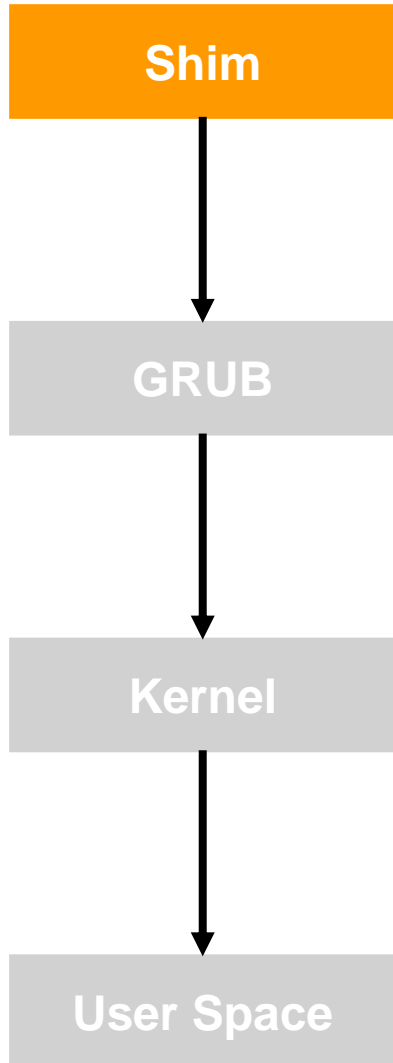
1. UEFI loads Shim
2. Shim loads GRUB
3. GRUB loads the Kernel
4. Kernel initializes the user space

Attack Surface: Linux Shim



- Verifies a bootloader signature
- Loads GRUB

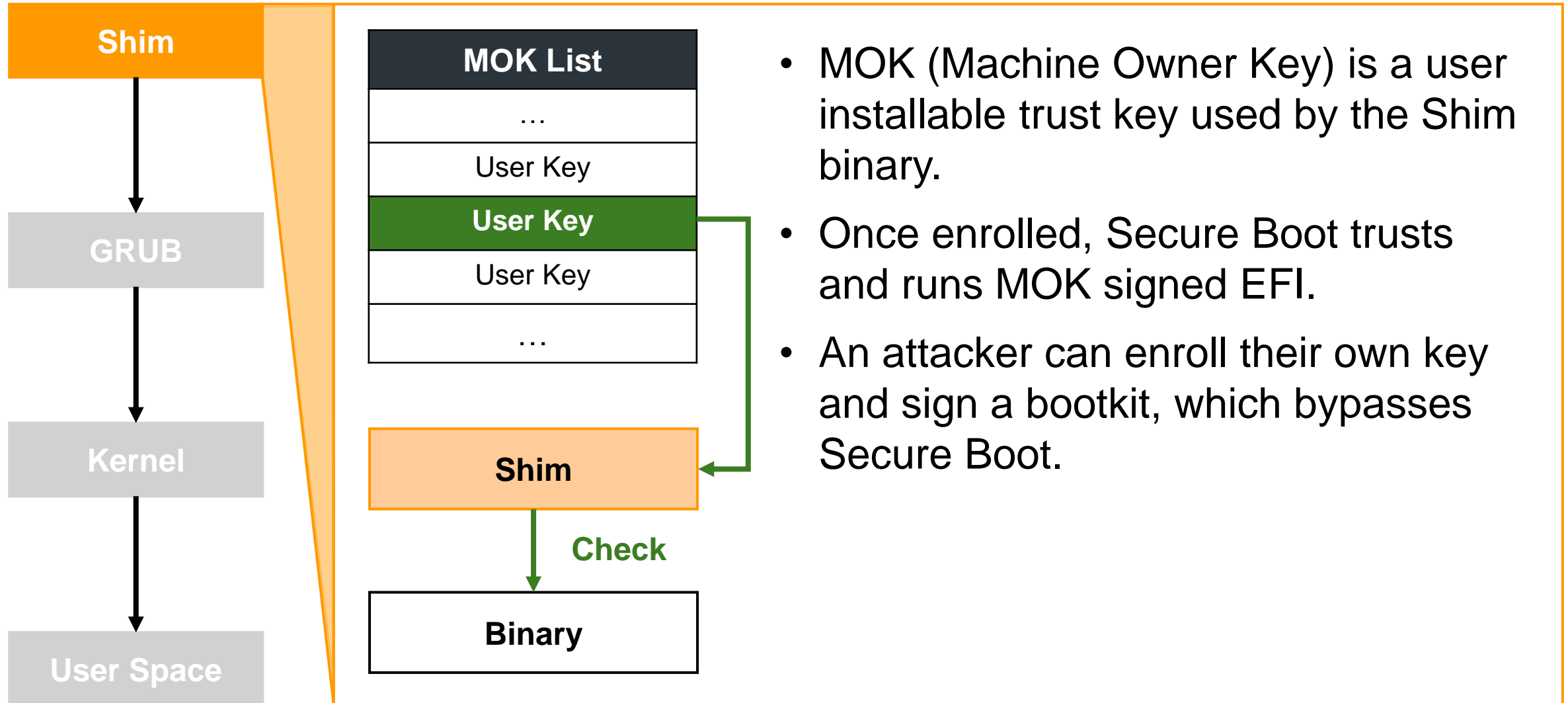
Attack Surface: Linux Shim



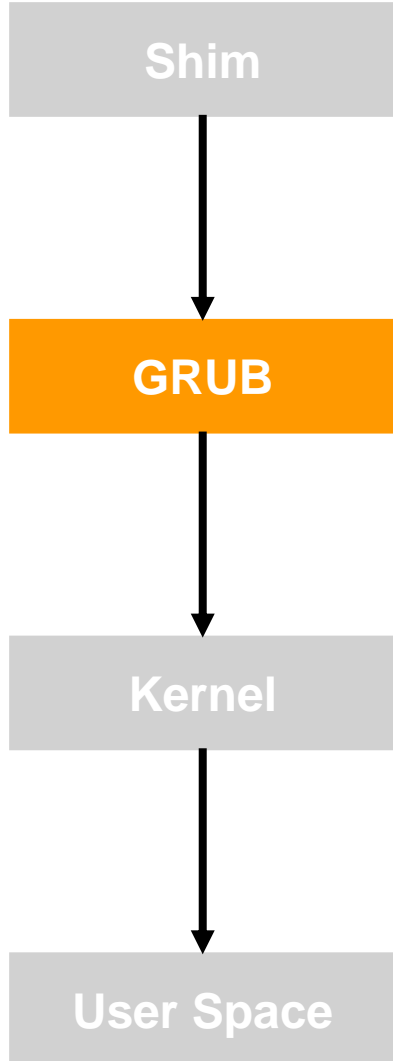
Attack Methods

- Shim binary tampering
- Runtime hooking
- **Machine Owner Key (MOK)**

Attack Surface: Linux Shim (MOK)

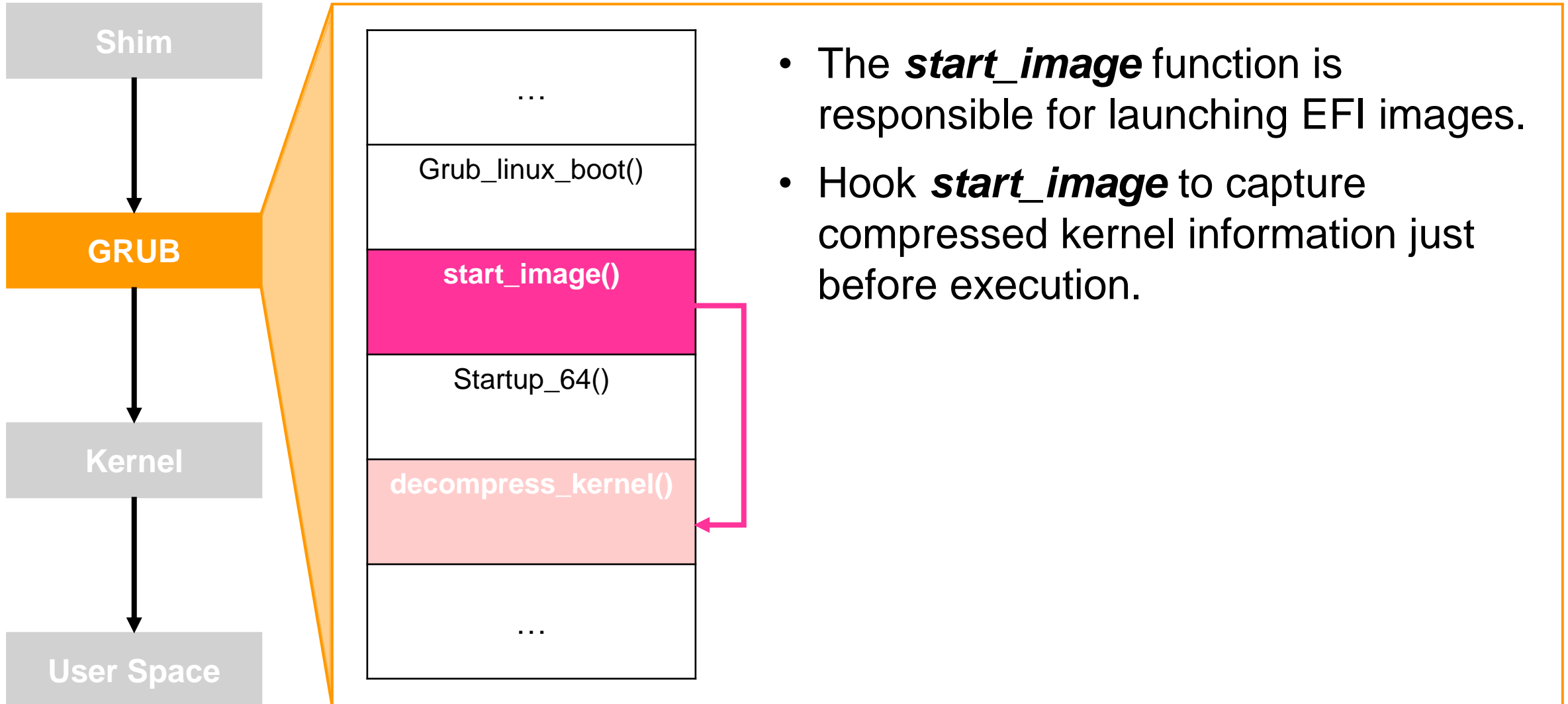


Attack Surface: Linux GRUB

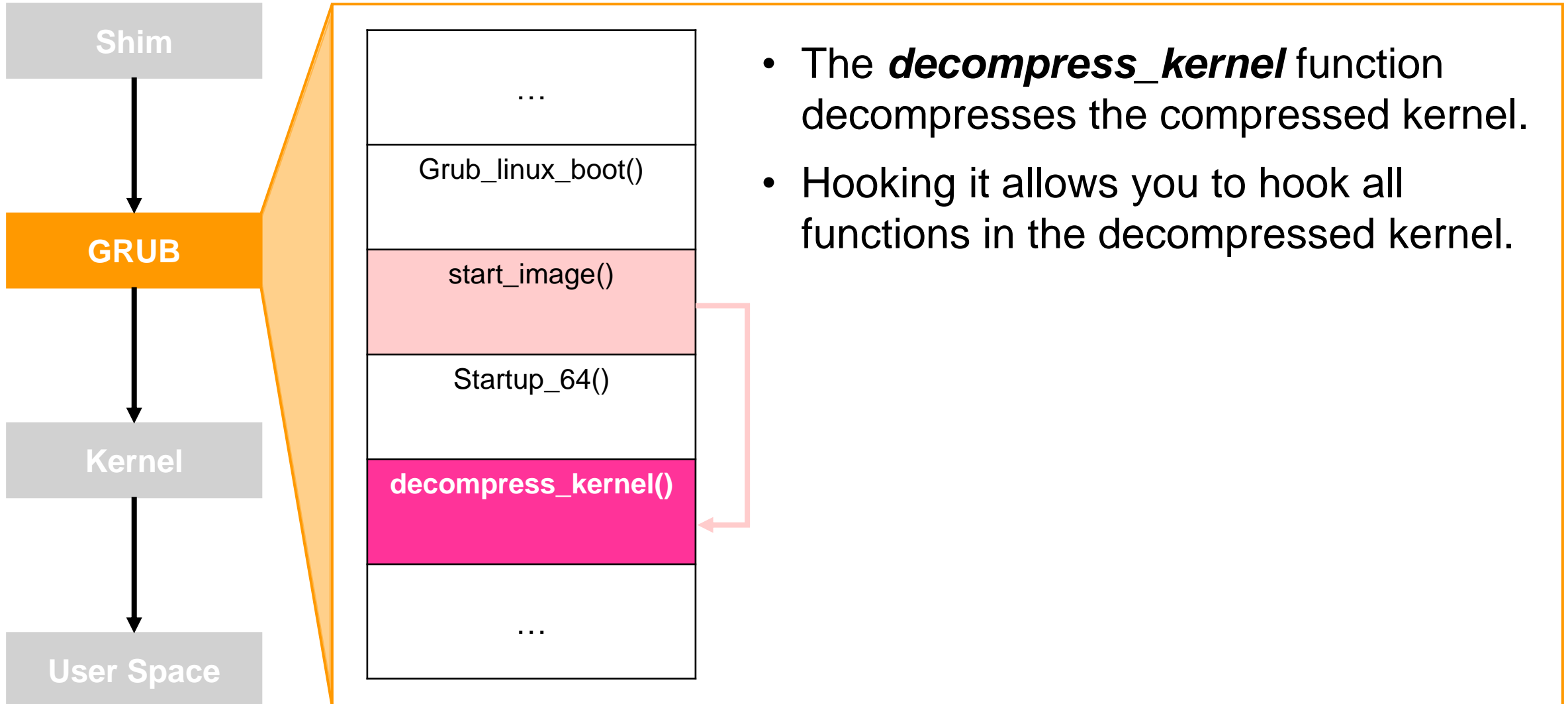


- Loads the kernel
- Transfers control to the kernel

Attack Surface: Linux GRUB (Hook)

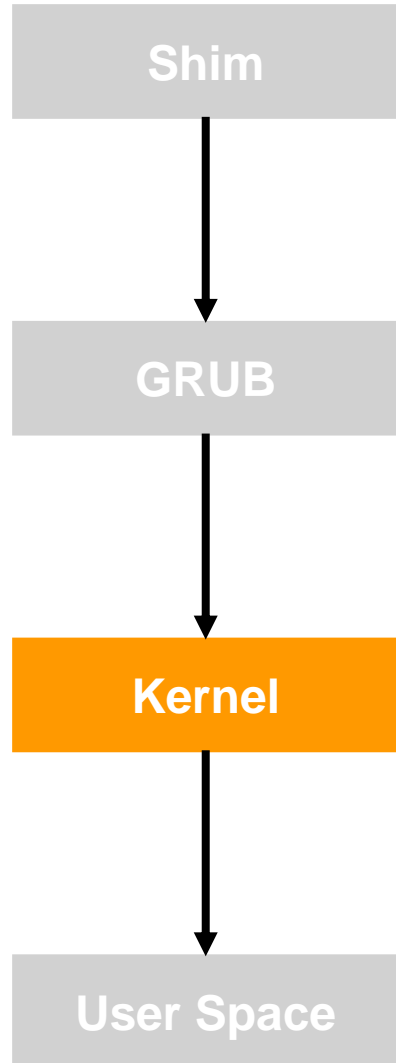


Attack Surface: Linux GRUB (Hook)



- The ***decompress_kernel*** function decompresses the compressed kernel.
- Hooking it allows you to hook all functions in the decompressed kernel.

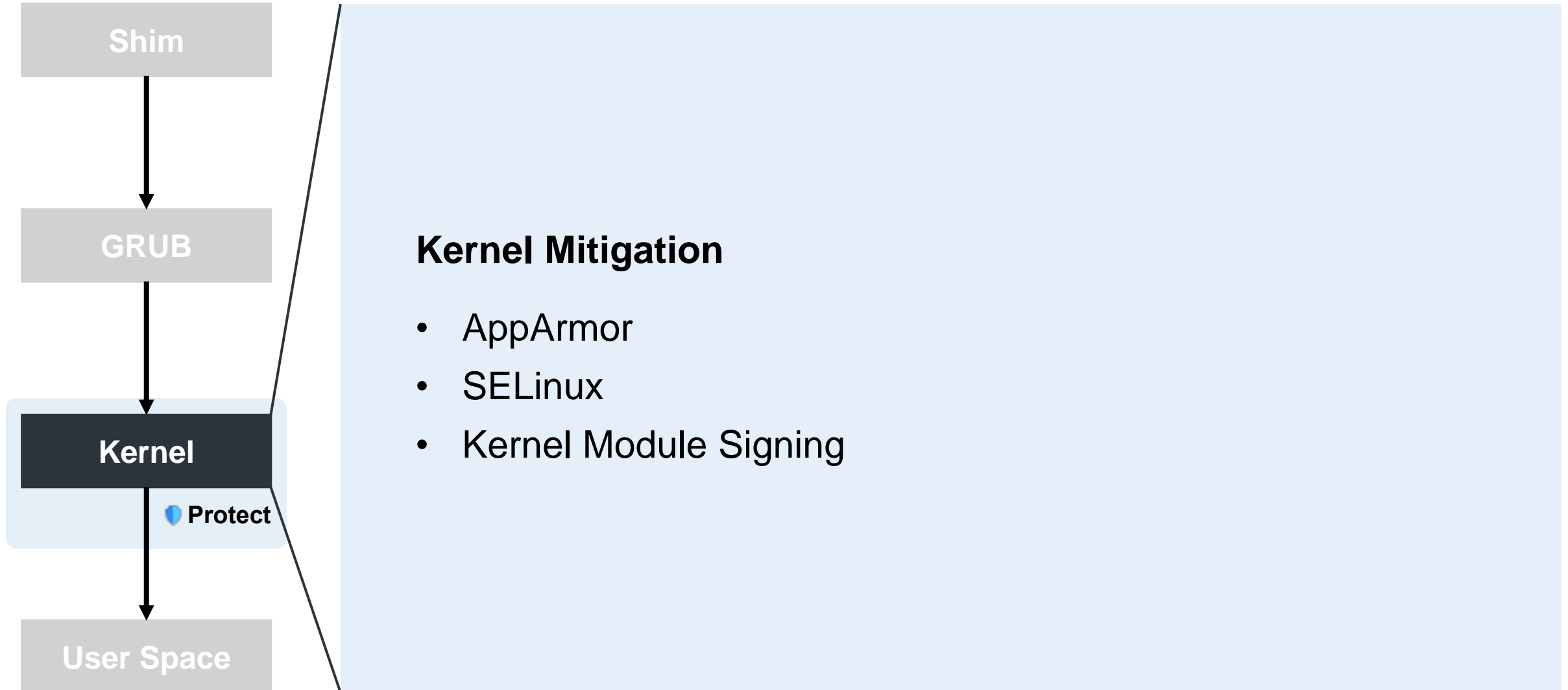
Attack Surface: Linux Kernel



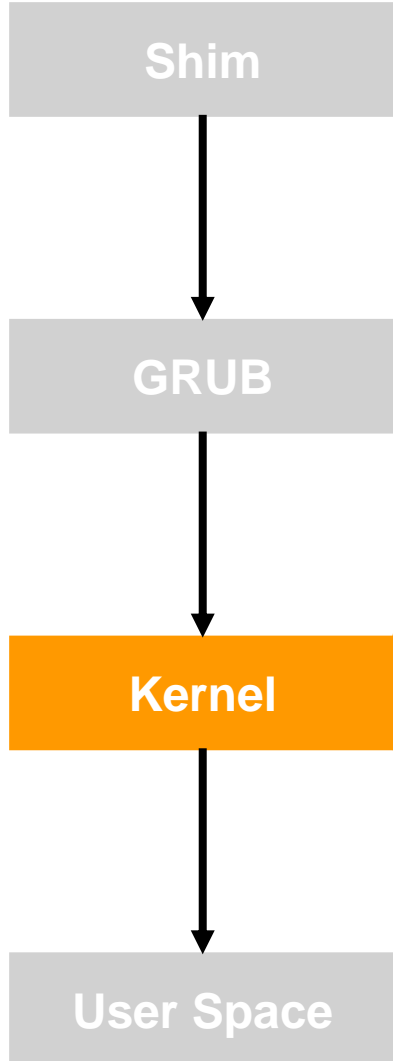
Kernel loading

- Compresses the kernel
- Decompresses the kernel

Attack Surface: Linux Kernel



Attack Surface: Linux Kernel (Hook)

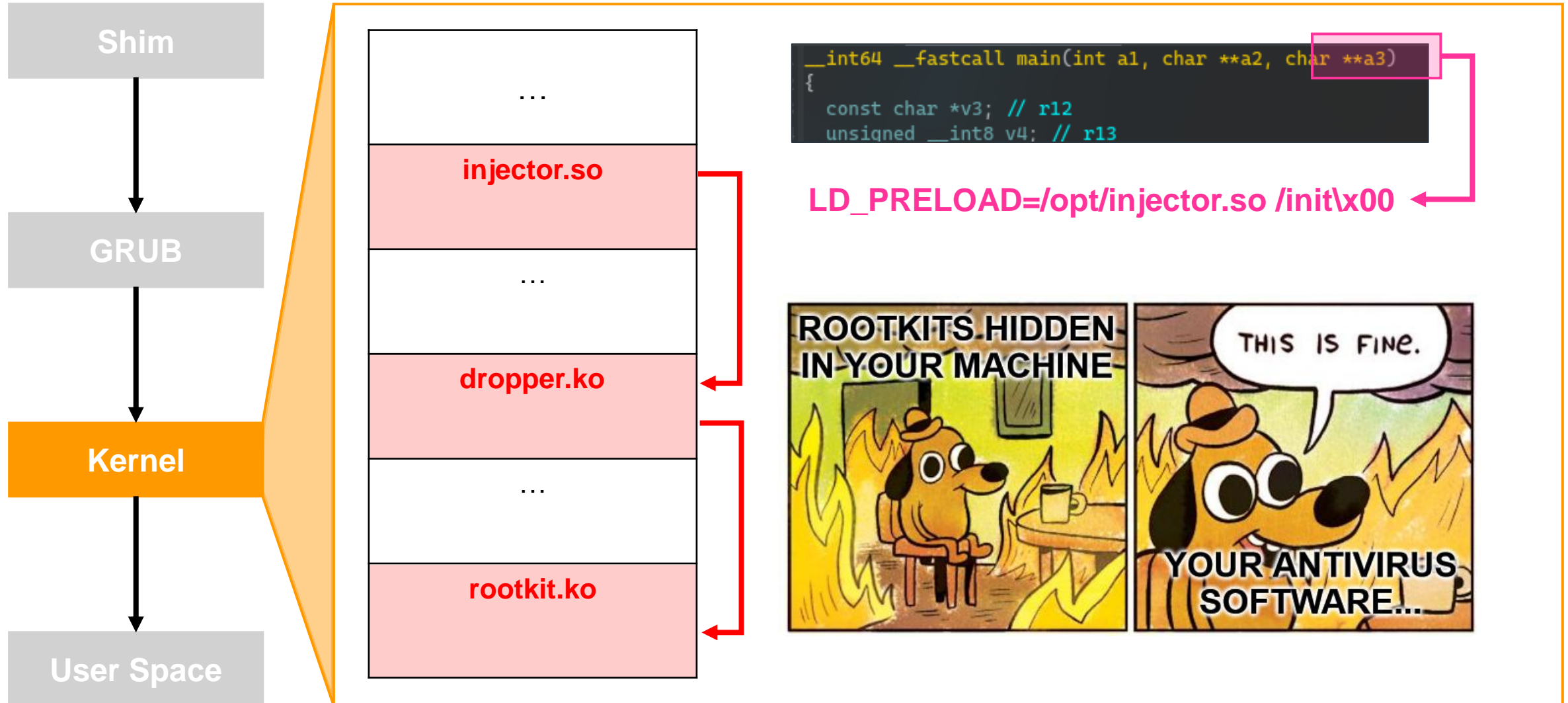


- Disable Kernel Module Signing

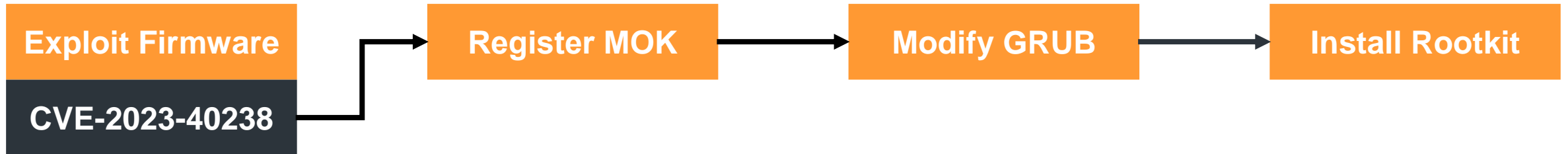
```
70 int module_sig_check(struct load_info *info, int flags)
71 {
72     int err = -ENODATA;
73     const unsigned long markerlen = sizeof(MODULE_SIG_STRING) - 1;
74     const char *reason;
75     const void *mod = info->hdr;
76     bool mangled_module = flags & (MODULE_INIT_IGNORE_MODVERSIONS |
77                                     MODULE_INIT_IGNORE_VERMAGIC);
78     /*
79      * Do not allow mangled modules as a module with version information
80      * removed is no longer the module that was signed.
81      */
82     if (!mangled_module &&
83         info->len > markerlen &&
84         memcmp(mod + info->len - markerlen, MODULE_SIG_STRING, markerlen) == 0) {
85         /* We truncate the module to discard the signature */
86         info->len -= markerlen;
87         err = mod_verify_sig(mod, info);
88         if (!err) {
89             info->sig_ok = true;
90             return 0;
91         }
92     }
93 }
```

Always
return 0 🕶️

Attack Surface: Linux Kernel (Hook)

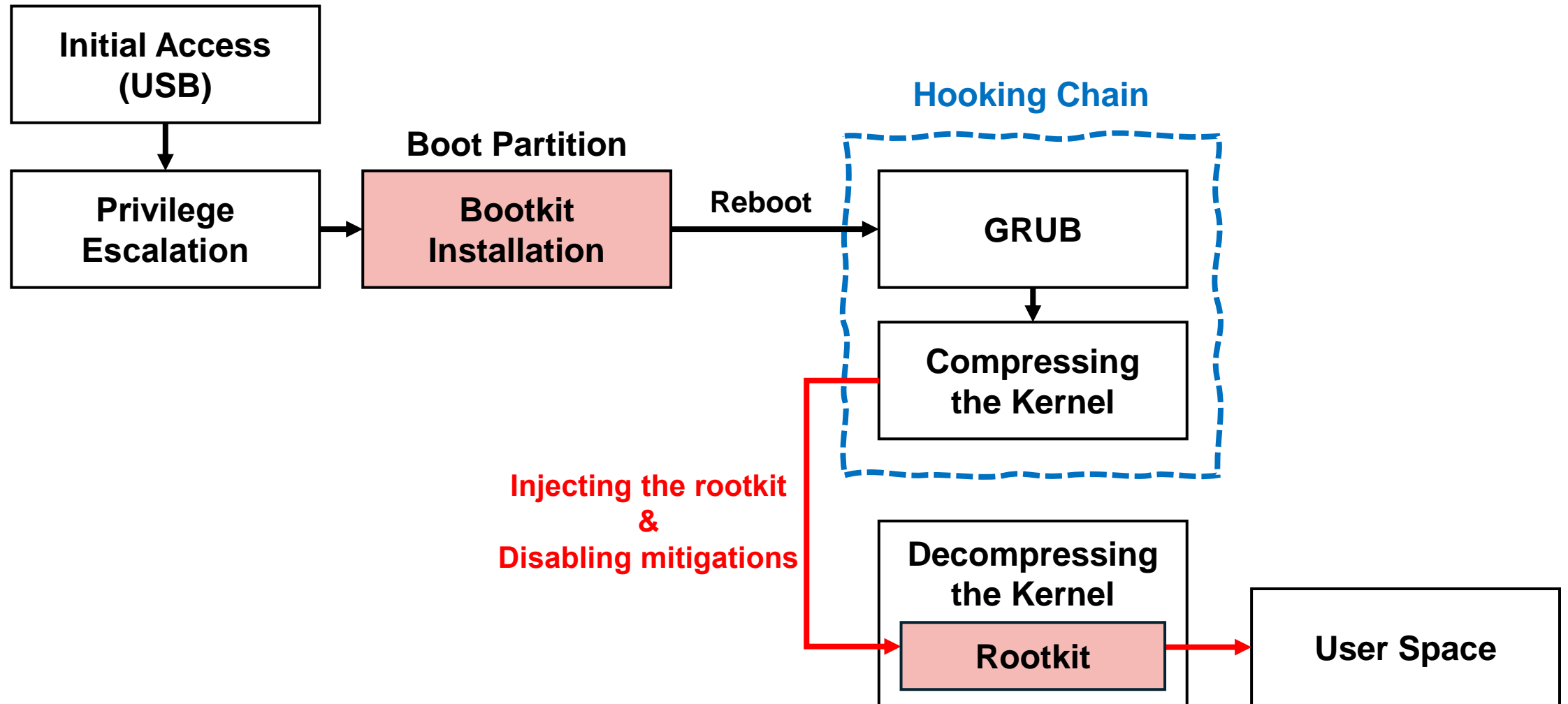


How BOOTKITTY Compromised the Boot Process on Linux



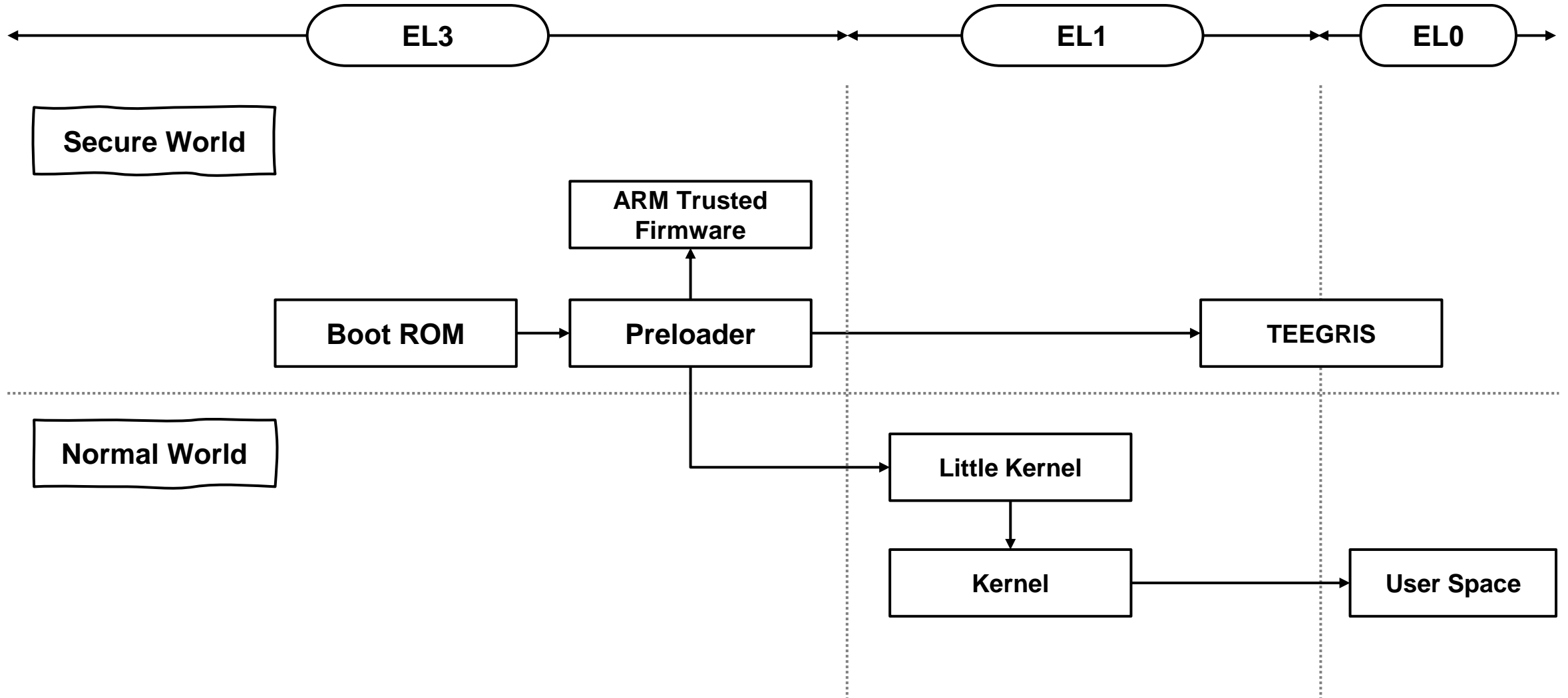
- LogoFAIL (*CVE-2023-40238*) → Enrolled MOK
- GRUB modification via the firmware bug
- Chains hijacking: GRUB → kernel

BOOTKITTY on Linux: USB Attack Scenario

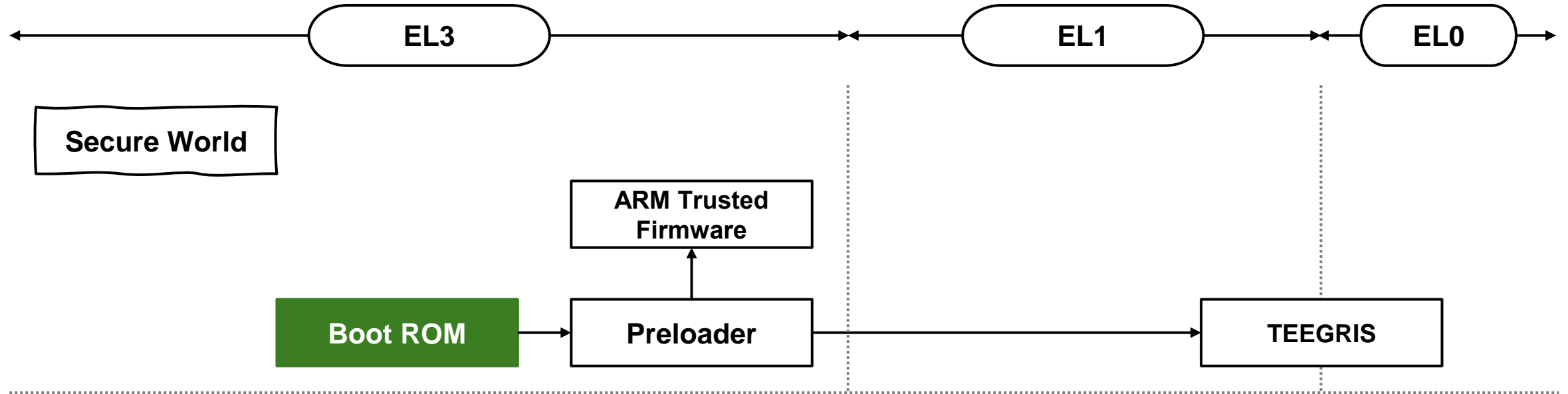


Linux Demo: Bootkit-Only

Android Boot Process: Exception Levels



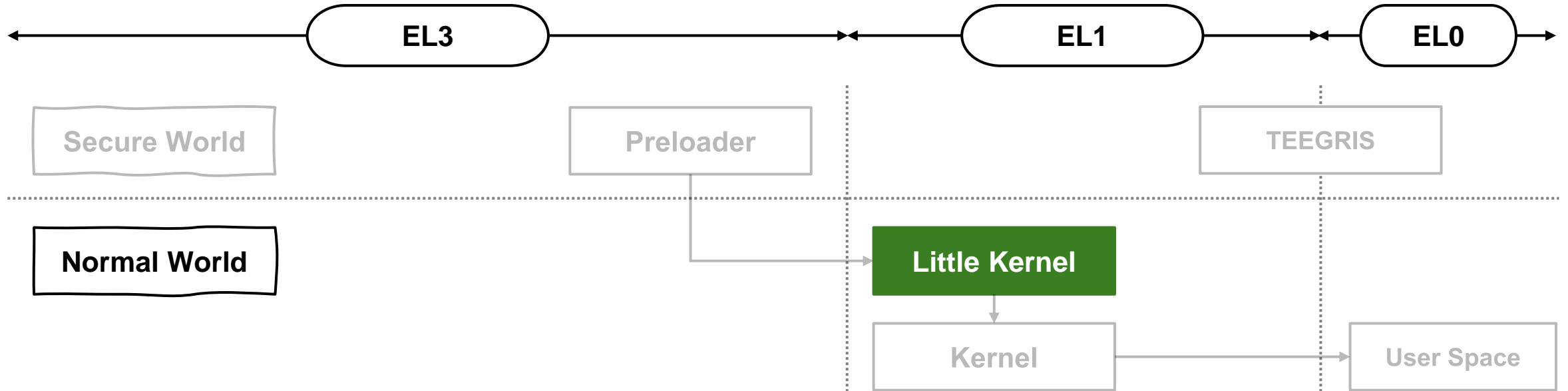
Attack Surface: Boot ROM (EL3)



Limitations

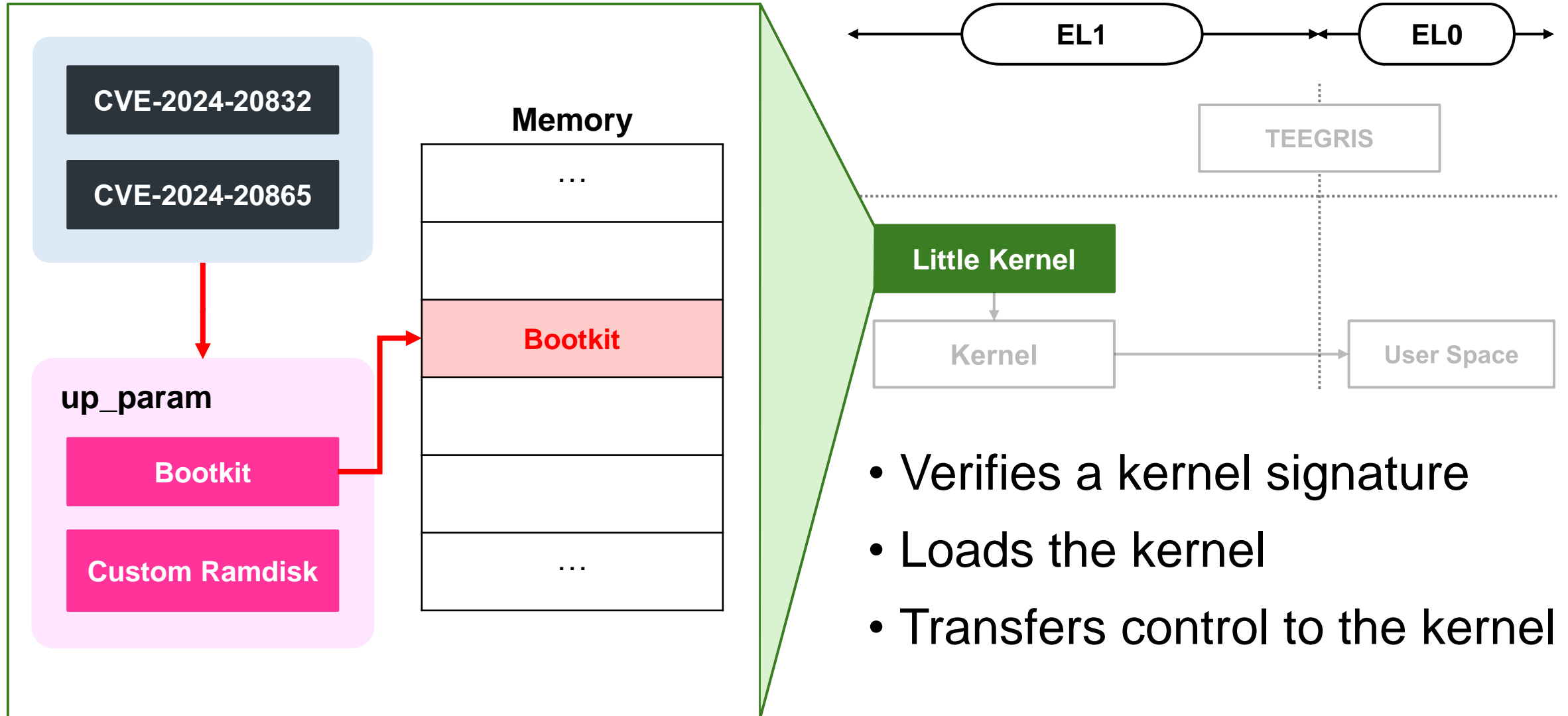
- Difficult to hook or infect
- Hard to access with software attacks

Attack Surface: Little Kernel (EL1)



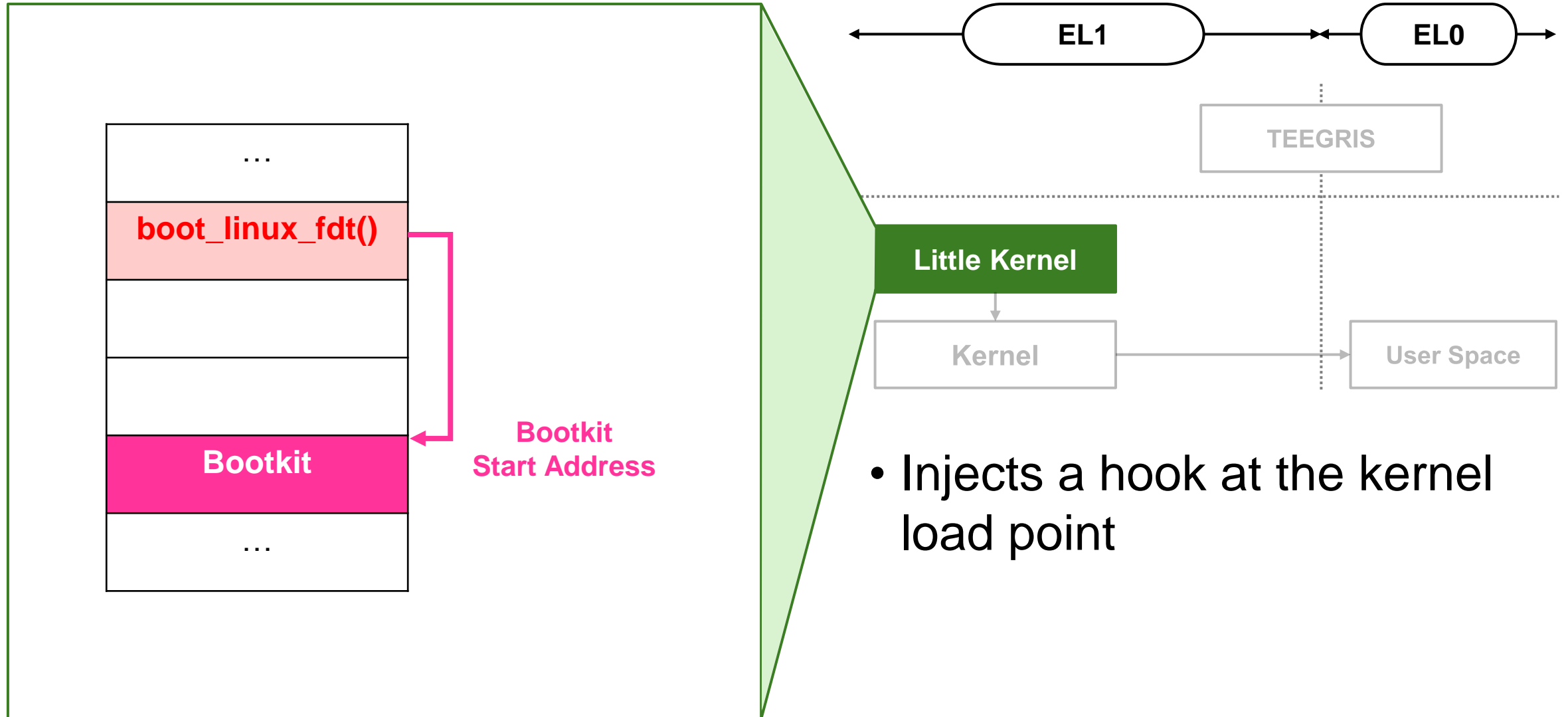
- Verifies a kernel signature
- Loads the kernel
- Transfers control to the kernel

Attack Surface: Little Kernel (EL1 - Hook)



- Verifies a kernel signature
- Loads the kernel
- Transfers control to the kernel

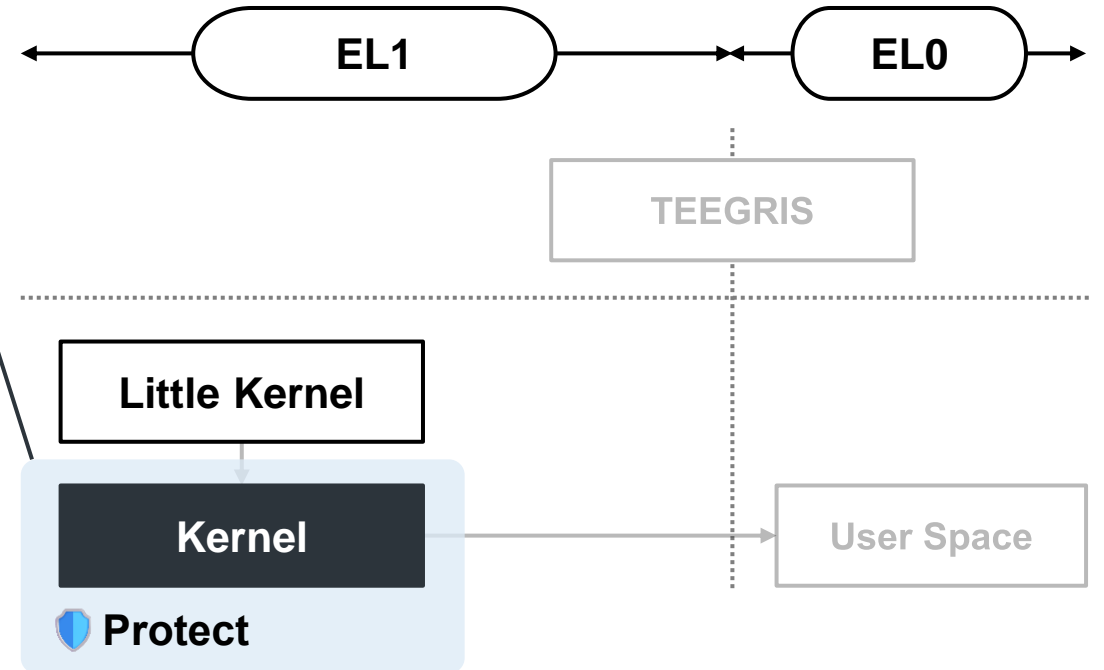
Attack Surface: Little Kernel (EL1 - Hook)



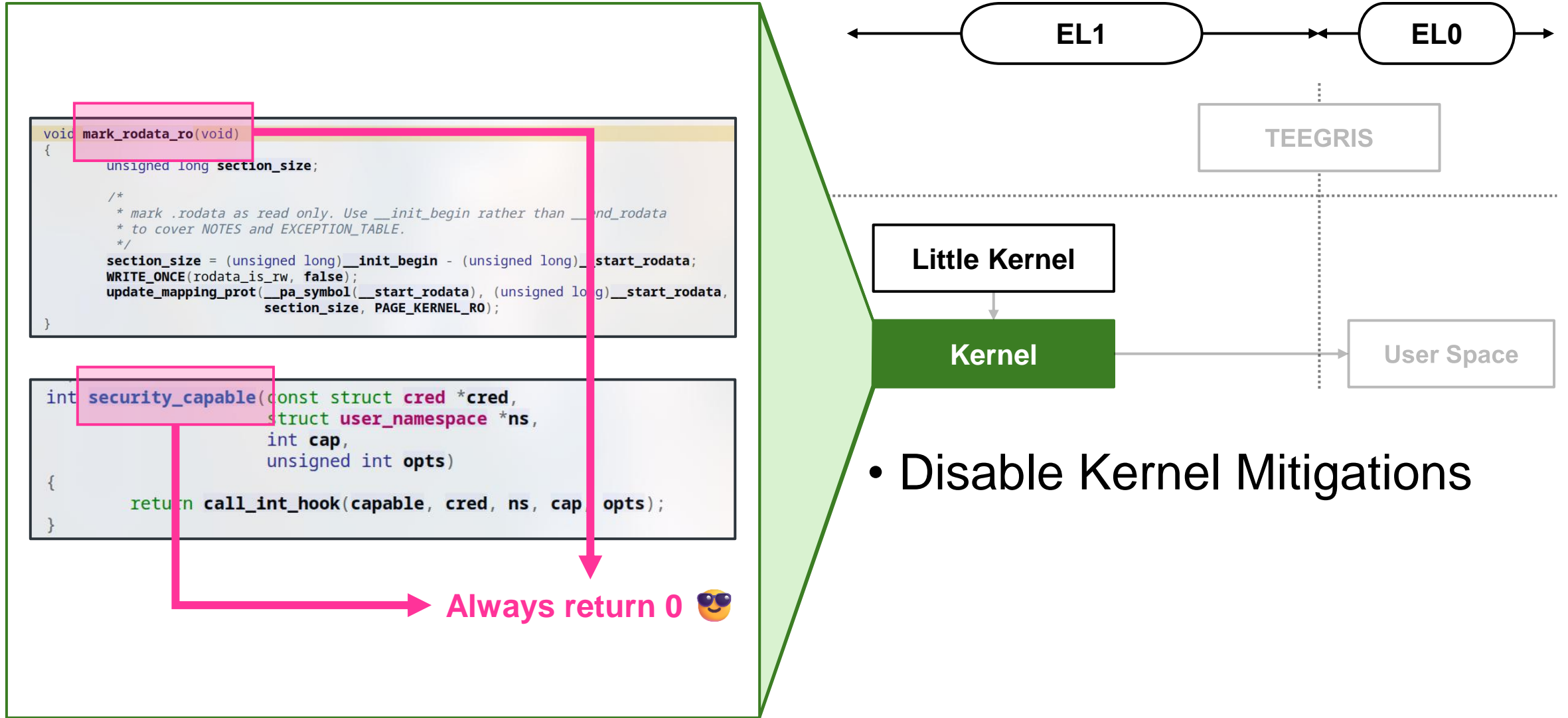
Attack Surface: Kernel (EL1 ~ EL0)

Kernel Mitigation

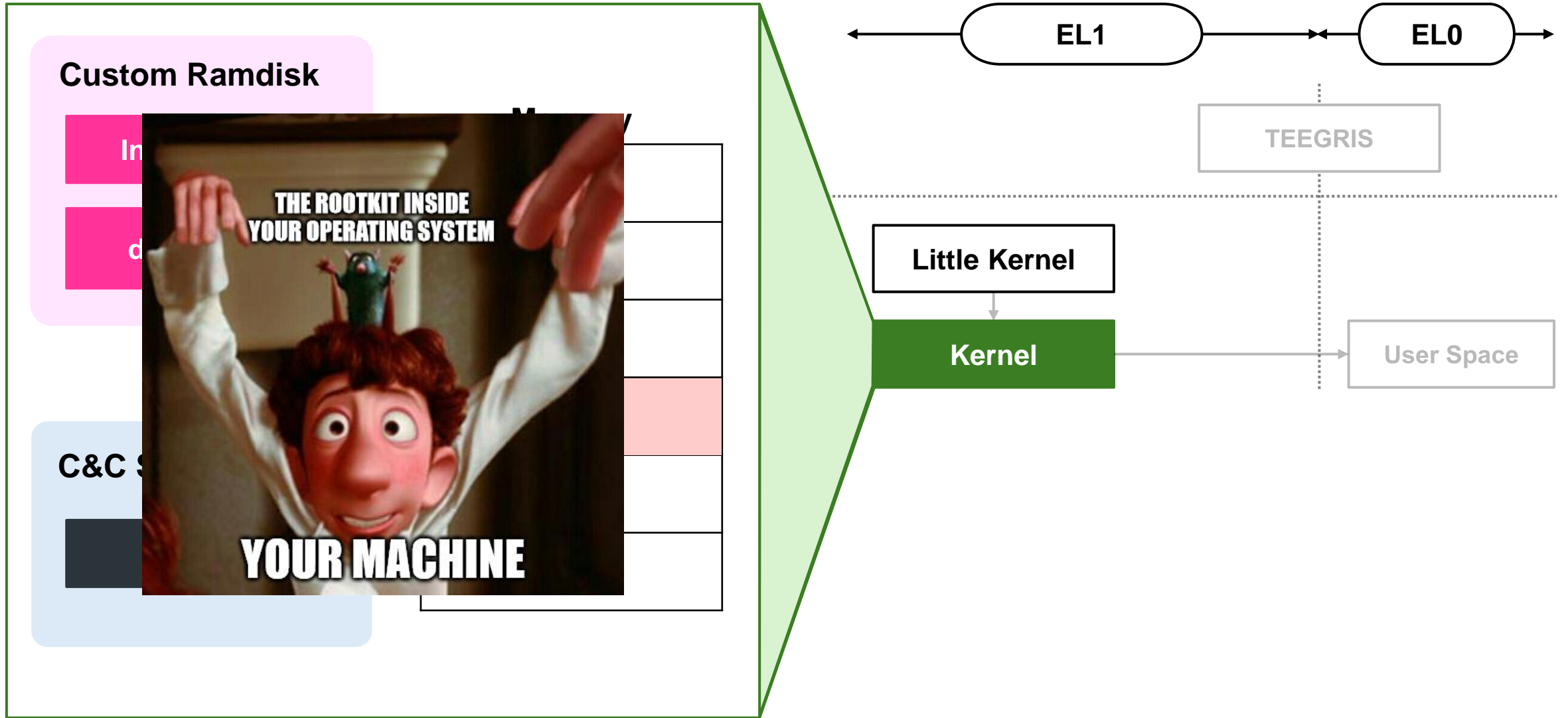
- Real-time Kernel Protection
- SELinux
- `mark_rodata_ro()`
- `security_capable()`



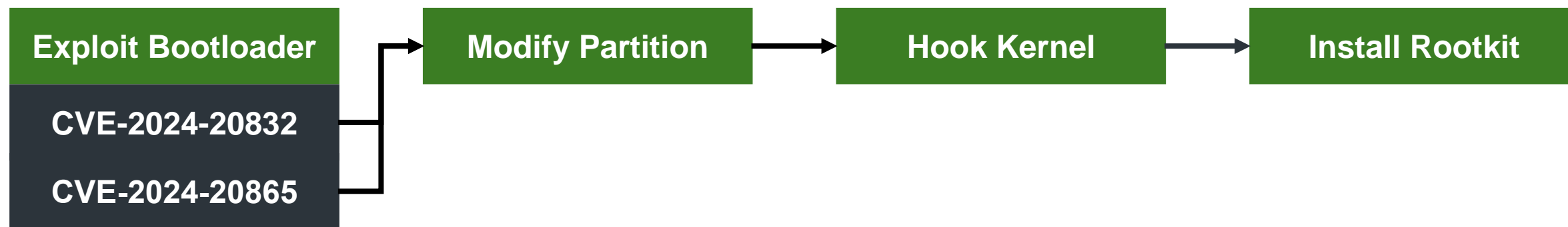
Attack Surface: Kernel (EL1 ~ EL0 - Hook)



Attack Surface: Kernel (EL1 ~ EL0 - Hook)

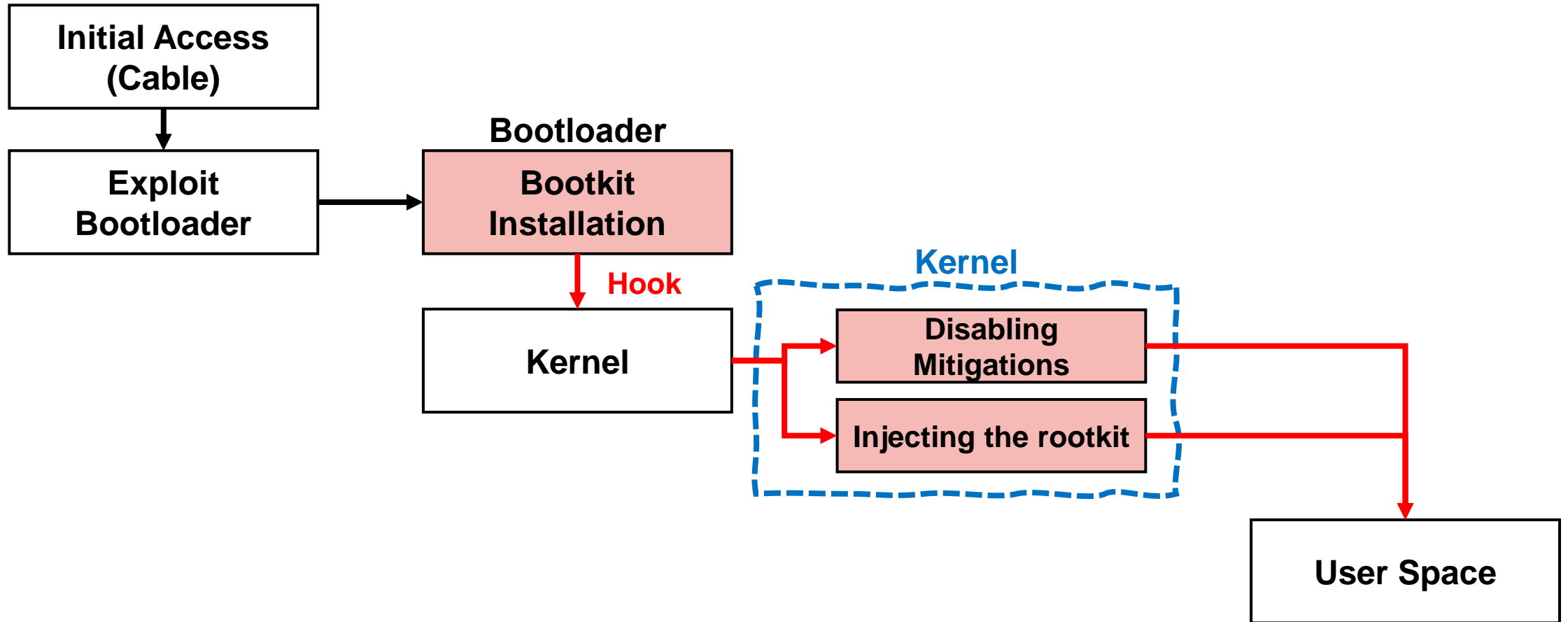


How BOOTKITTY Compromised the Boot Process on Android



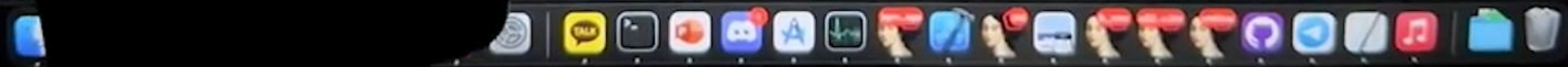
- Bootloader vulnerabilities (*CVE-2024-20832*, *CVE-2024-20865*)
→ Boot partition compromise
- Bootkit installation via bootloader bugs
- Full-chain hijacking: Bootloader → Kernel

BOOTKITTY on Android: Bootloader Attack Scenario



Android Demo: Bootkit-Only

```
seo@seoui-noteubug poc %
```



Key Takeaways

- Pre-kernel hooking → full system compromise
- Hardening the boot process and boot-chain integrity verification are essential

What We Have Not Talked About

- BOOTKITTY on windows
 - Windows boot process structure
 - Attack surfaces within the Windows boot process
 - Bootkit implementation strategy on Windows
- Rootkit implementation and evasion mechanisms

Please read our paper!

Artifacts

- VM link



(<https://zenodo.org/records/15501870>)

- QEMU link



(<https://zenodo.org/records/15582744>)

Thank you

**BOOTKITTY:
Multi-OS Trust Chain compromise
from Bootkit to Rootkit**