KShot: Live Kernel Patching with SMM and SGX

IEEE/IFIP DSN 2020 (*Runner-up Best Paper Award*)

Lei Zhou[1 2 *], Fengwei Zhang [1 **], Jinghui Liao [3], Zhenyu Ning[1], Jidong Xiao[4], Kevin Leach[5], Westley Weimer[5], Guojun Wang[6]

[1] SUSTech, China

[2] Central South University, China

[3] Wayne State University, USA

[4] Boise State University, USA

[5] University of Michigan, USA

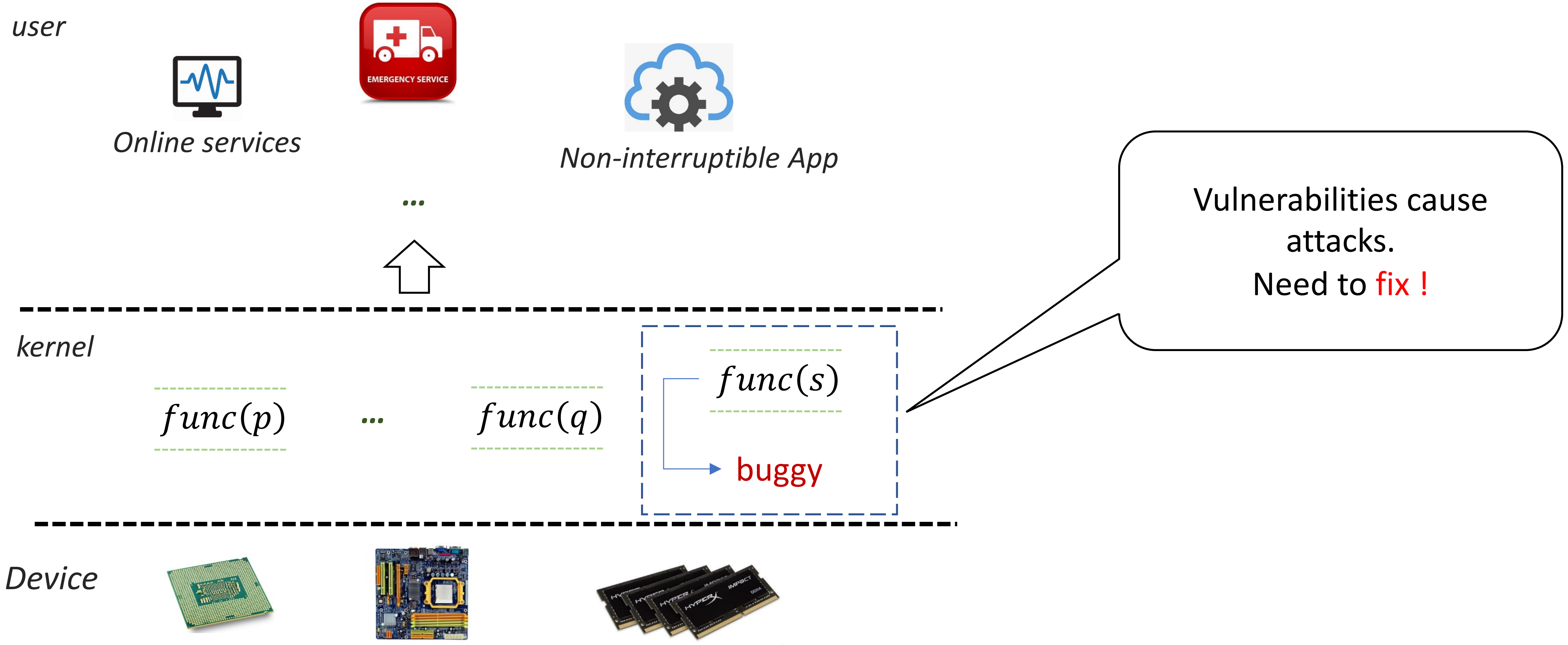
[6] Guangzhou University, China

* Work was done while visiting COMPASS lab at SUSTech; ** The corresponding author

Outline

- **Introduction and Background**
- Architecture of KShot
- Design and Implementation
- Evaluation: Effectiveness and Performance
- Conclusion

Why Need Patch the Kernel



Patching Mechanism

Traditional Update

WINDOWS UPDATE

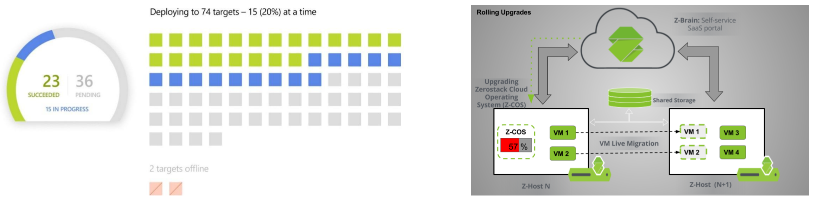
Ubuntu Updater

RED HAT RARE CRITICAL UPDATE

yum yellowdog updater modified

...

Rolling Update



Live Patching

ORACLE Linux | Ksplice

Livepatch

Canonical Livepatch

debian kpatch

kGraft SUSE

...

Users may unwilling to stop the runtime system, even need to patch kernel.

So, they choose kernel-based live patching.

But what if the kernel is compromised?

All Resources from Internet

Challenges: Security

1. To patch the kernel, need to trust the kernel first!

That's a trap if the compromised kernel is against the patching!

Challenges: Resource Overhead

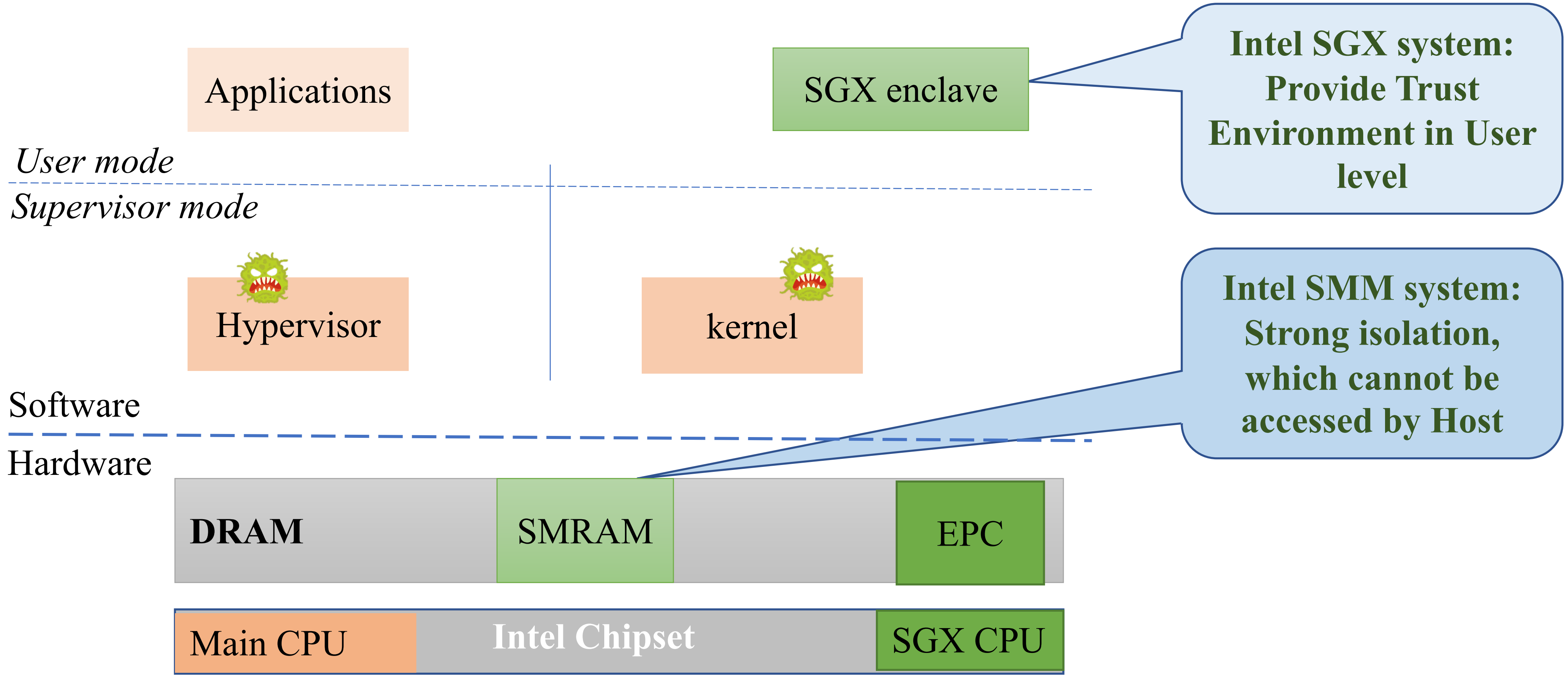
2. Overhead on Live patching may be larger than Restart

Kernel-based Live Patching needs to store and restore the current system state

Reliable Solution

Using Trusted and Isolated Execution Environment live patches the kernel without interrupting the target system!

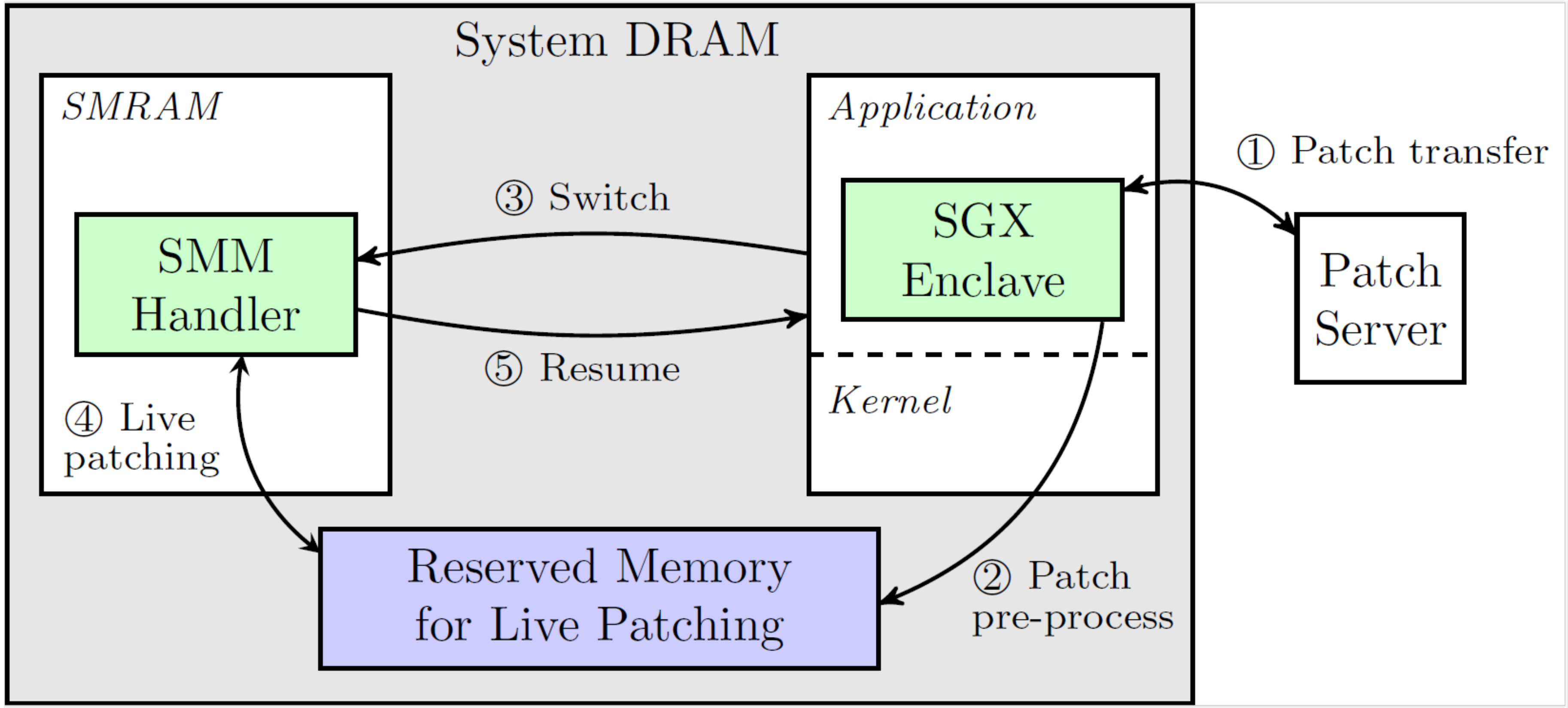
TEE Background: SGX and SMM



Outline

- Introduction and Background
- **Architecture of KShot**
- Design and Implementation
- Evaluation: Effectiveness and Performance
- Conclusion

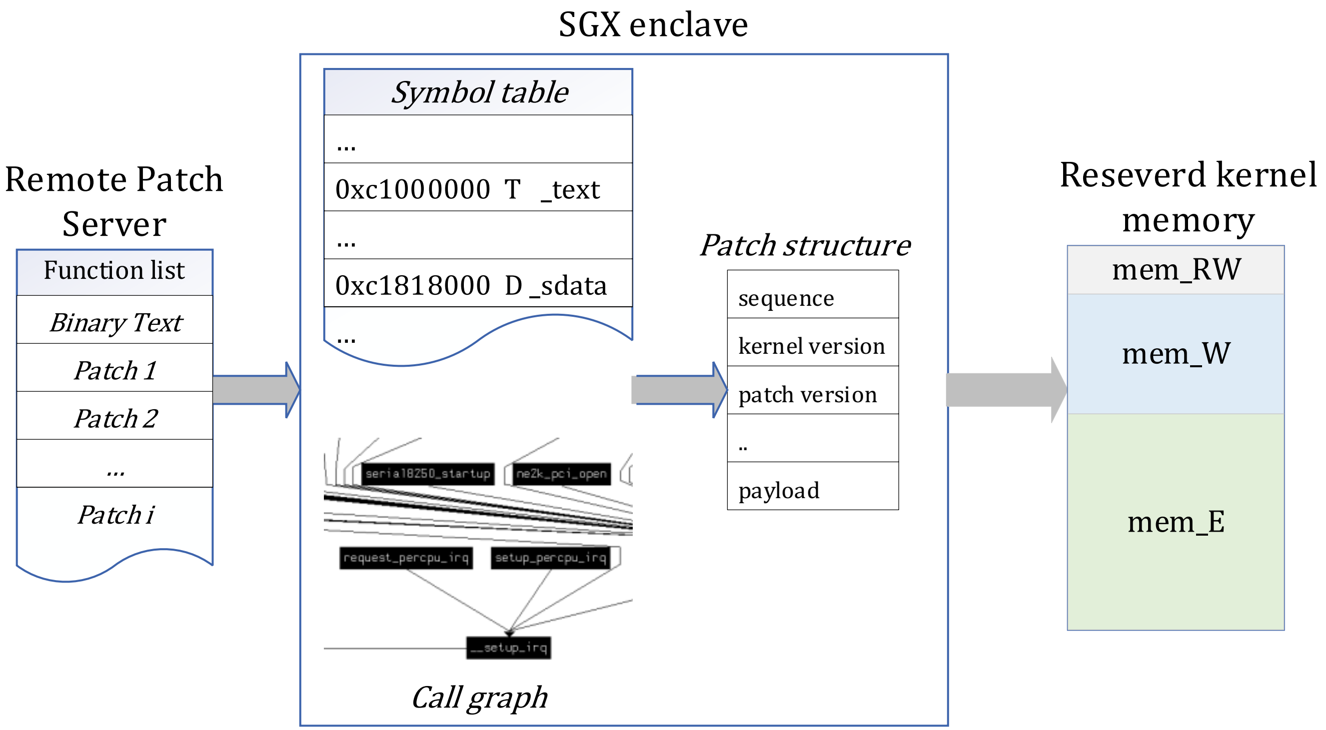
High-level Architecture of KShot



Outline

- Introduction and Background
- Architecture of KShot
- **Design and Implementation**
- Evaluation: Effectiveness and Performance
- Conclusion

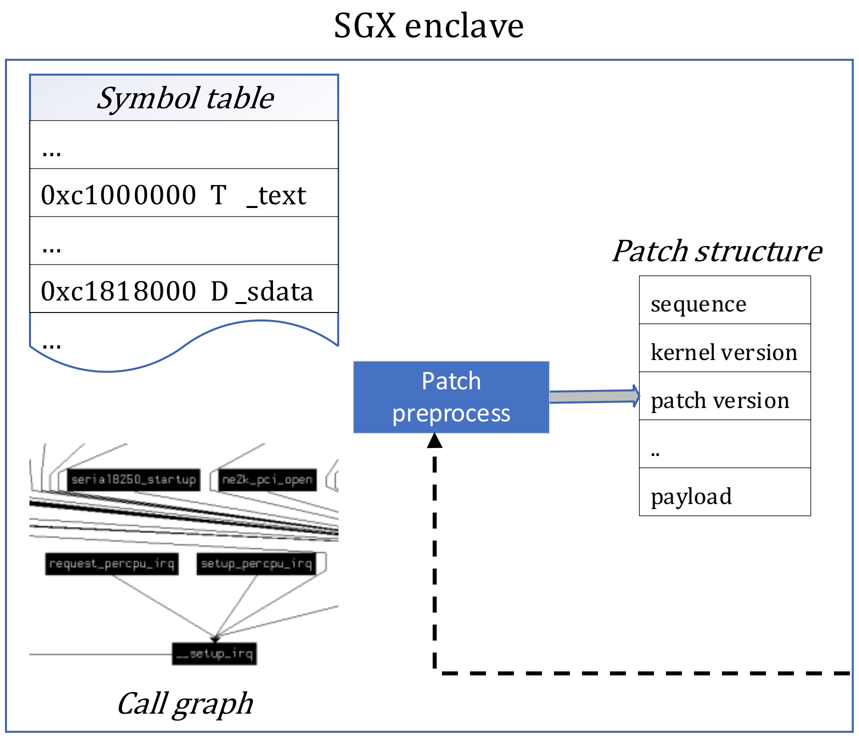
SGX-based Patch Preparation



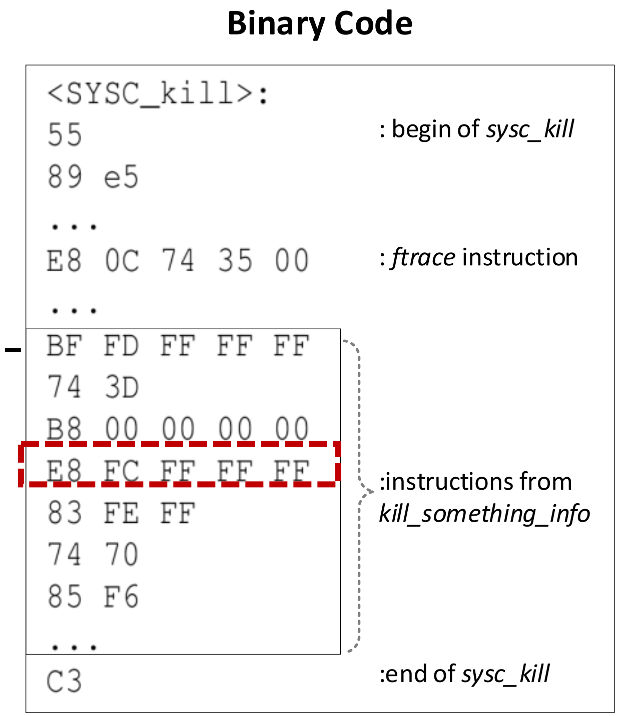
1 Reserve an isolated memory space.

2 Design a pre-preparing module
In SGX enclave.

SGX-based Patch Preparation



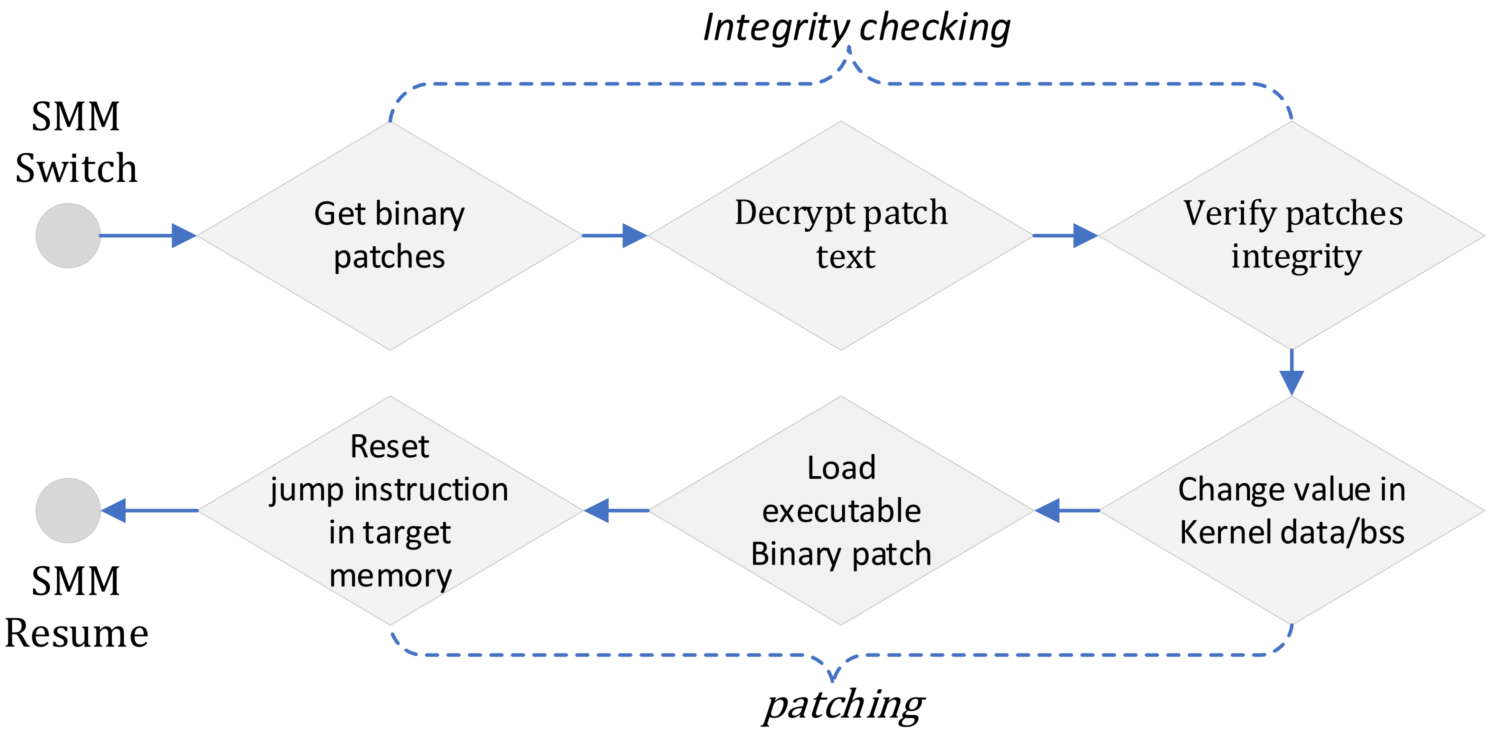
3 *Check the input binary patch.*



4 *Modify the effected instruction: like branch.*

5 *Final patch was encrypted and sent to reserved share memory.*

SMM-based Live Patching



The workflow of patching in SMM handler.

Also, it is easy to rollback and update the patch with the similar operations.

Outline

- Introduction and Background
- Architecture of KShot
- Design and Implementation
- **Evaluation: Effectiveness and Performance**
- Conclusion

Evaluation

The test environment platform:

- ✓ real-world patches from the Common Vulnerabilities and Exposures (CVE) Database.
- ✓ analyzed 267 such vulnerabilities for Linux kernels 3.14 and 4.4.
- ✓ Intel Core i7 CPU (supporting SGX and SMM) with 16GB memory.
- ✓ a combination of Coreboot with a SeaBIOS payload as the system BIOS.
- ✓ Ubuntu 14.04 and 16.04 using kernel versions 3.14 and 4.4.

Evaluation: Effectiveness and Performance

While deploying KShot, we consider three research questions:

- RQ1. Can KShot correctly apply kernel patches?
- RQ2. What is KShot’s performance overhead?
- RQ3. How does KShot compare to existing approaches?

RQ1. Can KShot correctly apply kernel patches?

CVE Number	Affected Functions	Size	Type*
CVE-2014-0196[1]	n_tty_write	86	1
CVE-2014-3687[1]	sctp_chunk_pending, ctp_assoc_lookup_asconf_ack	16	1,2
CVE-2014-3690[1]	vmx_vcpu_run, vmcs_host_cr4, vmx_set_constant_host_state	247	3
CVE-2014-4157[1]	current_thread_info	5	2
CVE-2014-5077[1]	sctp_assoc_update	98	1
CVE-2014-5206[1]	do_remount	34	2
CVE-2014-7842[1]	handle_emulation_failure	16	1
CVE-2014-8133[1]	set_tls_desc, regset_tls_set	81	1,2
…	…	…	
CVE-2016-5829[2]	hiddev_ioctl_usage	119	1
CVE-2016-7914[2]	assoc_array_insert-_into_terminal_node	330	1
CVE-2016-7916[2]	environ_read	63	1
CVE-2017-6347[1,2]	ip_cmsg_recv_checksum	15	2
CVE-2017-8925[1,2]	omninet_open	9	2
CVE-2017-16994[2]	walk_page_range	27	1
CVE-2017-17053[2]	init_new_context	13	2
CVE-2017-17806[1,2]	shash_no_setkey, hmac_create, crypto_shash_alg_has_setkey	91	1,2
CVE-2017-18270[1,2]	key_alloc, install_user_keyrings, join_session_keyring	273	1,2
CVE-2018-10124[1,2]	kill_something_info, sys_kill	51	1,2

[1] affects Linux 3.14. [2] affects Linux 4.4. * indicates patch type

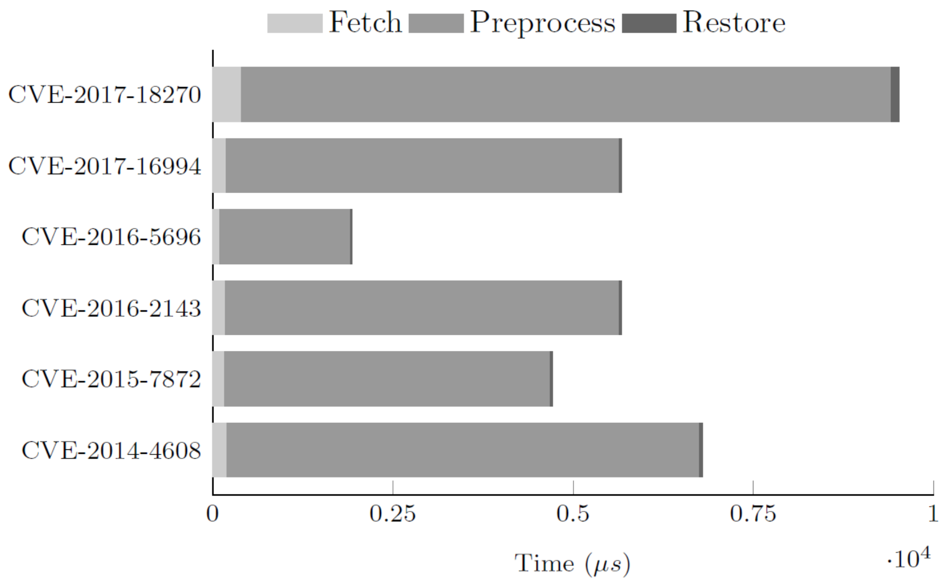
We randomly selected 30 of those 214 patches.

Part of experimental results shown in above table.

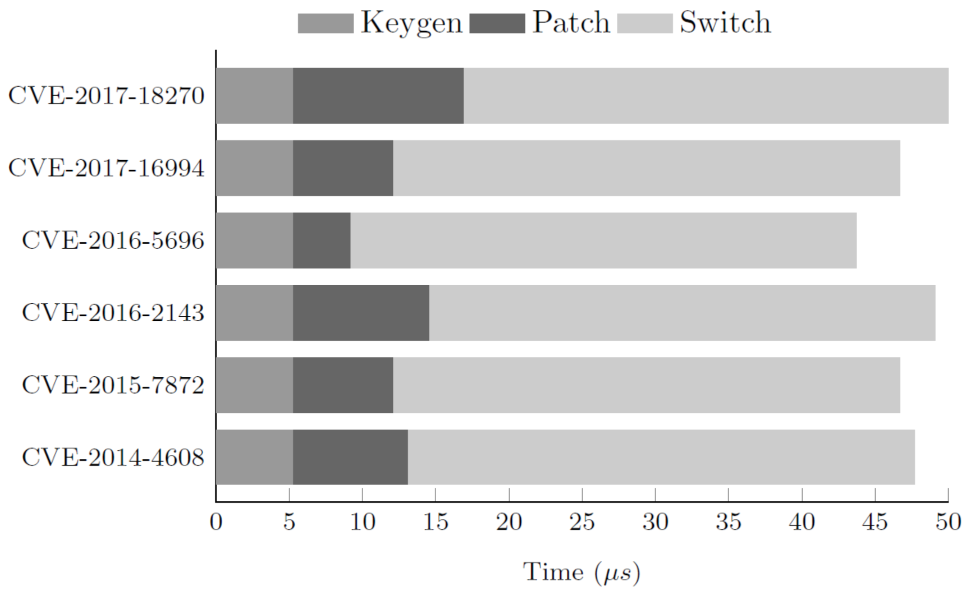
KShot can correctly apply kernel patches.

RQ2. What is KShot's performance overhead?

- SGX-based pre-preparation introduces extra overhead, but does not interrupt the normal system.
- SMM-based patching causes a very short pause, and the normal system state stays the same.



SGX-based patch preparation time.



SMM-based live patching time.

Time overhead in each step of real CVE case live patching

RQ3. How does KShot compare to existing approaches?

Comparison with non-kernel binary patching.

	Kernel Dependency	Untrusted OS	Applicability
Dyninst [24]	✓	✗	userspace
EEL [10]	✓	✗	userspace
Libcare [25]	✓	✗	userspace
Kitsune [59]	✓	✗	userspace
PROTEOS [26]	✓	✗	kernel
KSHOT	✗	✓	kernel

We can find that only KShot is kernel independent and useable in Untrusted OS

RQ3. How does KShot compare to existing approaches?

Comparison with kernel patching systems.

	Type	Downtime	Untrusted OS	Memory
KUP [8]	kernel	3s/kernel	✗	>30G
KARMA [9]	instruction	$5\mu s$/patch[1]	✗	lua engine
kpatch [10]	function	45.6ms/patch[1]	✗	16G
KSHOT	function	$50\mu s$/patch[1]	✓	18M

[1] for an average-sized patch of less than 1KB

The performance of KShot is better

Outline

- Introduction and Background
- Architecture of KShot
- Design and Implementation
- Evaluation: Effectiveness and Performance
- **Conclusion**

Conclusion

- **KShot -secure and efficient framework for kernel patching**
  — Leveraging Intel SMM.
  — Leveraging Intel SGX.
  — Against indicative kernel vulnerabilities.

- **Application scenarios**
  — Compromised Hypervisor, OS kernels.
  — Without external checkpoint-and-restore resources.

- **Introducing low overhead and a small trusted code base**

Thank You for Your Attention!
Questions?

zhangfw@sustech.edu.cn

http://cse.sustech.edu.cn/faculty/~zhangfw/

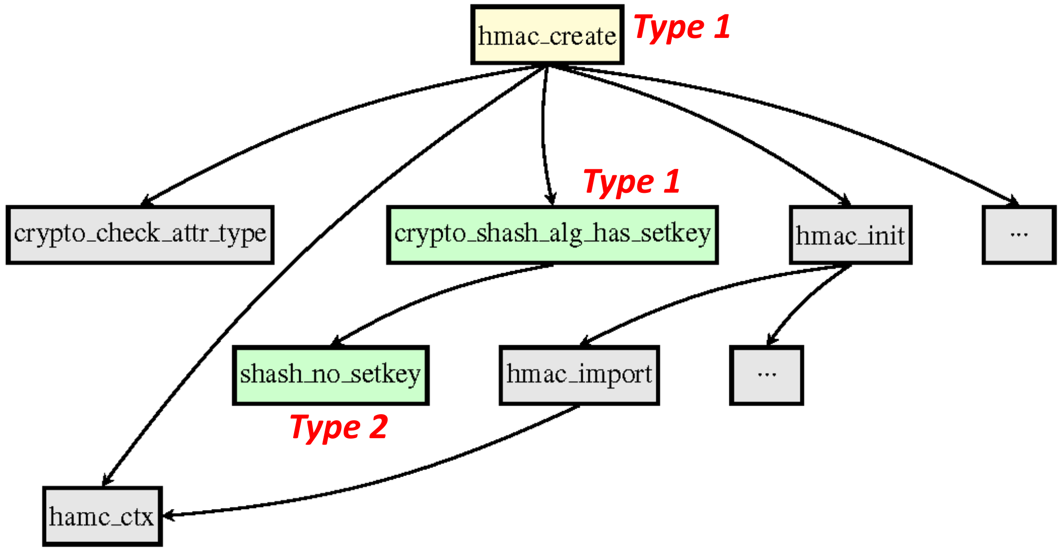
Backup Slides

KShot Design & Implementation

- *Binary Patch Preparation*
- SGX-based Patch Preparation
- SMM-based Live Patching
- Patching Protection

Identify the Patch Function

The patch case for CVE-2017-17806



Type1 and Type 2 are shown in such case

We assume we can get the trusted patch source code.

Vulnerable functions are defined with three types:

Type 1: *non-inline function,*

Type 2: *inline function,*

Type 3: special case: *data structure changed function.*

Finding the final target function for patching is different in each type.

Target Function Analysis

With knowing a vulnerable function, need to find the patching function:

- 1 get the binary kernel code through compiling the kernel source.
- 2 locate the vulnerable instruction segments.
- 3 identify the patching-needed function.

KShot Design & Implementation

- Binary Patch Preparation
- SGX-based Patch Preparation
- SMM-based Live Patching
- *Patching Protection*

Patching Protection

Malicious Patch Reversion

- SMM-based kernel protection.
- Introspect regions of memory overwritten with trampoline instructions.

Denial-of-service attacks

- Generally difficult to defend.
- Identify the memory written events with SMM and remote server.