A Framework for Software Diversification with ISA Heterogeneity

Xiaoguang Wang
Virginia Tech

SengMing Yeoh
Virginia Tech

Robert Lyerly
Virginia Tech

Pierre Olivier
The University of Manchester

Sang-Hoon Kim
Ajou University

Binoy Ravindran
Virginia Tech

Abstract

Software diversification is one of the most effective ways to defeat memory corruption based attacks. Traditional software diversification such as code randomization techniques diversifies program memory layout and makes it difficult for attackers to pinpoint the precise location of a target vulnerability. Some recent work in the architecture community use diverse ISA configurations to defeat code injection or code reuse attacks, showing that dynamically switching the ISA on which a program executes is a promising direction for future security systems. However, most of these work either remain in a simulation stage or require extra efforts to write program.

In this paper, we propose HeterSec, a framework to secure applications *utilizing a heterogeneous ISA setup composed of real world machines*. HeterSec runs on top of commodity x86_64 and ARM64 machines and gives the process the illusion that it runs on a multi-ISA chip multiprocessor (CMP) machine. With HeterSec, a process can dynamically select its underlying ISA environment. Therefore, a protected process would be capable of hiding the instruction set on which it executed or detecting abnormal program behavior by comparing execution results step-by-step from multiple ISA-diversified instances. To demonstrate the effectiveness of such a software framework, we implemented HeterSec on Linux and showcased its deployability by running it on a pair of x86_64 and ARM64 servers, connected over InfiniBand. We then conducted two case studies with HeterSec. In the first case, we implemented a multi-ISA moving target defense (MTD) system, which introduces uncertainty at the instruction set level. In the second case, we implemented a multi-ISA-based multi-version execution (MVX) system. The evaluation results show that HeterSec brings security benefits through ISA diversification with a reasonable performance overhead.

1 Introduction

Software diversification has proven to be a very effective way to defeat software memory corruption attacks [42]. By diversifying the target application memory layout, these diversification techniques are capable of randomizing vulnerable code locations [3, 8, 35, 36, 40, 63, 78, 81], detecting abnormal program behaviors (i.e. attacks) [15, 38, 51, 57, 58, 72, 73, 83], or hiding the secret data [39, 41]. The uncertainty brought about by a diversified program effectively raises the bar for launching a successful attack.

The "end of Moore's Law" [21, 25] has forced chip vendors to advance performance and energy efficiency boundaries elsewhere, in particular by designing radically different hardware: multicore and manycore chips [11, 56, 61]; CPUs with heterogeneous micro-architectural properties [34, 53], partially overlapping ISAs [32], and various forms of accelerators and programmable hardware [22] that exploit heterogeneity. CPUs with heterogeneous-ISA cores – studied by the academic research community [4, 44, 52, 69, 71] – are another point in the architectural design space that are now available as commodity hardware – e.g., Intel Skylake processor with in-package FPGA [28, 29] enables synthesizing RISC-V and x86 soft cores; AMD's new generation x86 processor integrates ARM cores; commodity smart NICs integrate ARM [24, 49], MIPS64 [64], or Tile cores [48].

Recently, some research efforts explored using multiple, heterogeneous-ISA CPUs to secure the application execution. For example, architecture researchers have proposed systems that implement heterogeneous ISAs over one single chip to achieve inter-ISA program state randomization with higher entropy [70, 71, 76]. Another recent work leverages the distributed, heterogeneous-ISA machines to detect program vulnerability exploits [73]. Specifically, it simultaneously runs multiple instances of the same application on heterogeneous-ISA machines to detect execution divergence caused by eventual security attacks (a.k.a., multi-variant execution [15]). However, programming on such a distributed, multi-ISA environment is not easy, as it requires tremendous efforts to synchronize program states over differential OS kernels and instruction sets.

In this paper, we make the first step towards applying software-based diversification concepts to a real multi-ISA en-

vironment, aiming to secure software execution with ISA heterogeneity. An ISA-diversified program can have additional randomness in its code and data memory layout, register usage, instruction orders, and micro-architecture behaviors. Furthermore, the diversified program variants could potentially leverage some architecture-dependent security extensions, making it even harder for attackers to bypass a single layer of protection. To achieve this goal, we propose HeterSec, a software framework that facilitates securing applications with multiple ISAs. Unlike existing simulation-based approaches, HeterSec bridges real heterogeneous-ISA machines. HeterSec works at the operating system and process runtime level, giving processes an illusion of running on a CMP machine while possessing the ability to dynamically select the underlying instruction set or cross check program state between two ISA-diversified program instances.

To demonstrate its effectiveness, we have built two security applications on top of HeterSec. The first security application enables the target program to randomly execute between machines with different ISAs, implementing a moving target defense (MTD) system [31]. The second security application implements a multi-variant execution (MVX) system [15, 38]. A traditional MVX system runs multiple variants of an application with non-overlapping address space [15, 83]. On detecting abnormal runtime behaviors from the variants (e.g., unmatched system call return values, segfault), a MVX monitor could deduce there is likely an ongoing exploit. Variants generated from the ISA heterogeneity can automatically obtain an additional level of diversity, making attackers even harder to successfully launch an attack. Overall, we explored the research space in securing software execution with diversified instruction sets. To that aim, we made the following contributions:

- We built a software framework that can manage the process execution over coupled multi-ISA machine nodes for security purposes.
- We implemented two security applications on top of such a framework, namely multi-ISA based MTD and multi-ISA based MVX. The multi-ISA MTD randomly changes the execution ISA, hiding the precise target hardware features from attackers. The multi-ISA MVX uses ISA diversity as an additional dimension to differentiate program instances so that it is even harder for attackers to bypass the violation check.
- We demonstrated the potential of such multi-ISA based security systems with real-world evaluation; the results show that the additional layer of ISA diversity increases the cost for attackers, adding about 15% overhead for Nginx and Redis server applications in real-world scenarios.

The rest of this paper is organized as follows: Section 2 provides some background information of multi-ISA systems. We then describe the design, implementation and case studies of HeterSec in Section 3. The evaluation is presented in Section 4. Afterwards, we summarize the related works in Section 5 and conclude the paper in Section 6.

2 Background and Threat Model

In this section, we briefly introduce the background on moving target defense and multi-variant execution; next we describe our motivation by summarizing recent multi-ISA systems and the security implications; we then proceed to define the threat model at the end of this section.

2.1 MTD and MVX

Moving Target Defense (MTD) Most information systems are built on relatively static platforms. Many defense techniques also involve static integrity checks and introspection. The static nature of such defense mechanisms gives attackers the time to thoroughly study the target system and launch the exploit [31]. The goal of MTD is to break the static nature of the target systems, with deviation of existing defense mechanisms and adaptations over time. MTD is an abstract concept, leaving options open with regards to how it is implemented. Thus realization of its design philosophy can be demonstrated in many ways. For example, dynamic systems or network configuration [30, 31], dynamic application code and data [13, 81], etc. In this paper, we demonstrate the security benefit and performance cost of running processes with dynamic execution on multiple ISAs using HeterSec.

Multi-Variant eXecution (MVX) Another interesting way to secure applications with multiple ISAs is through multi-variant execution systems [15]. MVX is a software security technique that runs multiple functionally equivalent programs (variants) with differing memory layouts. Some examples of such memory layout differences and deltas include non-overlapping memory maps [38, 57, 83], reverse stack growth [58], etc. By executing the diversified variants with the same inputs, the MVX engine is capable of detecting when an attack happens if one of the variants fails. That being said, existing MVX techniques might not be met with as much success when attempting to detect attacks based on relative addresses [23, 27, 73] or architecture level vulnerabilities [37, 46, 82]. With HeterSec, we built a prototype to use multiple ISAs as the source of variation between variants and prove a multi-ISA MVX system is still capable of obtaining reasonable performance despite the overheads involved.

2.2 Multi-ISA Systems and Security

Heterogeneous CPUs have been widely adopted in both data centers and end devices. On mobile platforms, ARM big.LITTLE technology uses two types of processor to

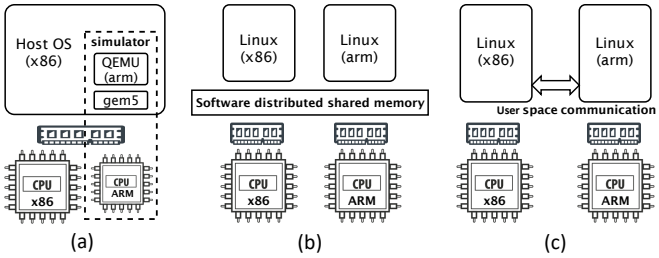

Figure 1: Comparison of multi-ISA security systems: (a) HIPStR with simulated multi-ISA chip [70, 71]; (b) HeterSec; (c) DMON on completely decoupled machines [73].

achieve a dynamic balance between maximum power efficiency and maximum compute performance [1]. On data center servers, heterogeneous ISA processors are being used in different scenarios. For example, GPUs and TPUs are often equipped to accelerate machine learning workloads [33]. ARM based PCIe-pluggable SmartNIC cards are used to offload network applications for improved throughput and security [67, 68]. In academia, there have been several works exploring the benefit and cost of building single-chip multi-ISA systems [20, 70, 71]. DeVuyst et al. [20] first demonstrated the possibility of building a multi-ISA chip on a simulator, showing the ability to migrate processes between different ISAs. Venkat et al. [71] further expanded on the idea by proposing that applications running on multiple ISA could have benefits in reducing power consumption and accelerating computation speed, which they called *ISA affinity*. Their findings proved that an application can have lower power consumption (or better performance) by being split into code phases. Based on the ISA affinity of each code phase, the application code can be selectively executed across a heterogeneous ISA chip. In terms of security, HIPStR explores using multiple ISAs to increase code entropy, which makes return oriented programming (ROP) attacks difficult to launch [70]. The difference between these works and our system is that they are all built on top of CPU simulators (gem5 [9] and QEMU [55] as shown in Figure 1 (a)). The simulation-based approach makes it hard for security researchers to investigate the security benefits of using a multi-ISA architecture.

A recent concurrent work, DMON [73], uses distributed heterogeneous-ISA machines to generate and host program variants (a distributed version of N-Variant Execution [15]). The variants run on completely separate machine nodes and each variant communicates with the counterpart variant through a lightweight UDP-based network protocol (Figure 1 (c)). Although lightweight network protocols can provide low latency communication cost to exchange data between distributed nodes, the use of `ptrace` interface to intercept system calls brings extra context switches. For example, running Lighttpd web server on DMON will have 5.43x performance overhead [73]. Furthermore, DMON does not provide generally sufficient abstraction to secure applications with multi-ISA machines. Therefore, it may provide less extensibility for developing multi-ISA based security applications which require timely execution of ISA switches. HeterSec instead focuses on building a generic framework to secure applications with the multi-ISA architecture. To this end, HeterSec adopts a hybrid approach – it runs on top of the real ISA-heterogeneous hardware; but the protected process has a unified view of system resources as if it runs on a multi-ISA CMP platform (Figure 1 (b)).

3 Design and Implementation

3.1 System overview

HeterSec aims to secure process execution by utilizing ISA heterogeneity to, for example, randomize the process execution environment over heterogeneous machines. To achieve this, HeterSec provides a per-process HeterSec execution environment. Specifically, it allows the protected process to be executed on machines running with different ISAs as if it were running on a single machine.

Figure 2 shows an overview of HeterSec with its new components added to an existing computer system stack. The components introduced by HeterSec include both the kernel and the user-space runtime as shown in blue. Figure 2 also illustrates two security application scenarios on top of HeterSec. In the first scenario, HeterSec switches the underlying ISA out from under the protected application, increasing the entropy of possible program states by masking the ISA switch and preventing attackers from divining underlying hardware details. In the second scenario, HeterSec launches multiple variants of the program, monitors the variants' execution states (e.g., return values of system calls, or segfault), and raises an alert on any execution divergence caused by a potential attack. The HeterSec kernel provides additional functionality to control the target process at runtime, such as the process interception, per process shared memory and fast inter-kernel messaging. For example, the HeterSec distributed operating system kernel maintains a synchronized page table for each protected process. The page tables are synchronized during each ISA switch, giving the HeterSec process a unique view of the memory. Secure application scenarios can be implemented as loadable kernel modules that interact with the target process execution. Since HeterSec only intercepts and interacts with the target process, it introduces nearly zero performance overhead to other processes running on HeterSec [1].

HeterSec has a concept of master OS. The master OS is the OS where the HeterSec process is initialized and launched. Correspondingly, the OS that works as the counterpart to the master OS is called the follower OS. The master HeterSec OS exports the view of system resources to the HeterSec process

[1] Except for a few in-kernel checks to verify the process status, HeterSec kernel does not bring extra code paths for non-HeterSec processes.

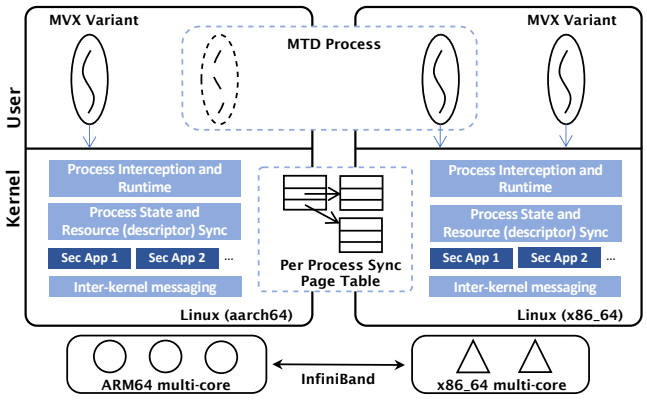

Figure 2: The architecture overview of HeterSec with two security application scenarios. The components in blue indicate modifications over existing software stack.

running on follower OS. Such system resources are often unique for each process, for example, open file descriptors, sockets, or event poll descriptors. For consistency reasons, HeterSec has to ensure that only one copy of such resources is maintained across OSes, maintaining a single source of truth for in-kernel state. When necessary, the master OS also helps to initialize and build the virtual address space for the follower OS' process. Virtual memory areas (VMAs) and pages are synchronized between the two OSes. All the inter-OS communication requests are registered with an inter-kernel messaging API so that messages can be less expensive as they avoid going through the complicated network stack. All the software components mentioned above are running on multi-ISA machines, connected over InfiniBand. In the following section, we describe the details of each component.

3.2 HeterSec distributed kernel

The HeterSec distributed kernel can be considered as a special implementation of multikernel systems [7]. Instead of running on a multi-core NUMA machine, HeterSec runs on a *heterogeneous ISA multi-domain "machine"*, with each computing domain connected with a high-speed network connection. By using this approach we could avoid to use simulation or dynamic code translation, so that code can be executed at nearly native speed. However, there are two problems with such a heterogeneous ISA multi-domain "machine": first, there is no memory coherence guaranteed between multi-ISA nodes, which raises programmability issues if we intend to leverage the heterogeneous ISA, multi-domain capabilities to implement security applications. Second, it is hard to manage the distributed resources (e.g., opened descriptors, network connections) on top of the heterogeneous instruction sets.

To solve those issues, HeterSec does not maintain global state for all OSes, but instead chooses only to maintain some HeterSec process specific states, synchronizing them on demand. To be compatible with existing software stacks, the HeterSec distributed operating system is designed as several kernel extensions and is built based on the Linux kernel. There are three major components that facilitate HeterSec processes running on heterogeneous ISA machines: the per-process page table synchronization, secure applications, and the system resource sharing service.

The first component is a per-process page table synchronization handler. HeterSec provides a synchronized page table for each HeterSec process. The state is then synchronized across the x86_64 and ARM64 machines on demand. Before the process is started as a HeterSec protected process, the secure application (a kernel module) has to be loaded and subsequently pass the defined security policy to the process runtime. Such security policies include how frequently to switch the instruction sets, which system calls are used to synchronize and check the program states in the multi-variant execution. The runtime then executes the protected process accordingly - for example, randomly running processes across multi-ISA nodes or concurrently executing variants with cross-ISA lockstep state checking. In short, based on the secure application scenario, the HeterSec kernels maintain the synchronized memory views across the multi-ISA nodes. In the current design, HeterSec leverages a dedicated kernel thread to synchronize the pages in the background. It maintains a simple *read-duplicate write-invalidate* protocol for the shared memory pages [80].

Another essential component for HeterSec is the system resource sharing service. HeterSec maintains a single view of the system resource from the HeterSec process perspective. That means for each HeterSec process, there should be only one set of the network sockets, opened file descriptors, etc. Unfortunately, system resources such as file descriptors, sockets and event descriptors, are difficult to be shared across machine boundaries due to the difficulty in splitting the in-kernel state. One potential solution could be using a Network File System (NFS) to share and synchronize the file systems across the OSes. However, this will introduce potential issues for those pseudo-files located in `/dev/tty`, or `/proc`. Architecture dependent shared libraries also use different instruction formats and EFL binary contents. Naively synchronizing those files will cause runtime errors and crashes in these programs. To address this problem, HeterSec combines an implementation of system resource remote procedure call (RPC) and a virtual descriptor table (VDT).

Before starting the process, the secure target application can be specified with a white list of files that should be loaded locally. By default, we put the standard output (i.e., `stdout` and `stderr`), the shared libraries and configuration files in the white list. During the protected process's runtime, the follower OS will build up a VDT. For each table entry, it indicates whether a descriptor should be accessed locally or remotely on the master node. For instance, we do not want to create two sockets for a single connection request on HeterSec

MVX. Therefore, the HeterSec kernel running on follower OS have to simulate the socket creation by placing a fake socket descriptor in the virtual descriptor table and mark that entry as virtual (V). On the contrary, descriptors of the opened library files and a `stdout` are marked as real (R) as they should be accessed locally. For system resource requests that have to be handled on the master OS, a system call RPC mechanism is provided. A system call server on the master OS handles the remote system call request, sets up the buffer value on the virtually shared pages, and returns the result to the caller on the follower node. The virtually shared pages are synchronized between nodes by the HeterSec kernel, as mentioned above.

We also support some termination signals (e.g., `SIGINT`) on the master node. On receiving a termination signal during the remote system call context, the master side HeterSec kernel replies a negative system call return value (i.e., `-ERESTARTSYS`) back to the follower kernel. The follower kernel then stops the HeterSec processes on the follower node. When the termination signal comes within the master kernel context, the master forwards the signal to the follower. Correspondingly, the follower terminates the execution loop. The master then stops itself by calling `do_exit()`.

3.3 Handling the cross-ISA code execution

Executing code on the multi-ISA "machine" as if on a single machine is challenging, since it requires several architecture-dependent code generation and state exchanges. HeterSec requires the architecture-dependent binaries generated from the same source code (e.g., same application source code and library code). This can unify most of the cross-ISA code execution behaviors, such as system call sequences. The generated binaries contains all the necessary information to run a protected process across ISA-different nodes. This information consists of instructions and data emitted by the compiler for each ISA. It may also carry some additional information such as the program state transformation routines. The types of information are decided by each individual security scenario. For example, cross-ISA randomized MTD execution would require information to transform the execution state from one architecture to another. This is because fine-grained program state (e.g., the variables on stack) must be synchronized accordingly as each architecture has its own specification for stack layout and register usage (Section 3.4.1). Security applications such as multi-ISA MVX require less information in metadata as each program instance is mostly self-contained on a single machine. The system call parameters (i.e., userspace buffers) and the opened descriptors are synchronized by the distributed operating system kernels mentioned above. It simplifies the system call simulation which is commonly used in existing MVX techniques (Section 3.4.2).

HeterSec introduces a new system call (i.e., `sys_hscall`) to identify the protected process and enable it to run on multi-ISA nodes. That system call sets up a bit in the process descriptor (`task_struct` in Linux); after that, the HeterSec code path in the distributed kernels will be triggered to support cross node process execution. For example, when defending the protected process in the MVX mode, we can initiate that system call to launch two process variants on both nodes. The in-kernel MVX engine checks the system call sequences and return values and raises an alert on any execution divergence. More details are discussed in Section 3.4.2.

3.4 Case Studies

We have built two security application scenarios on top of HeterSec, which can fully utilize the instruction set diversity.

3.4.1 Multi-ISA MTD

The first security application is a heterogeneous-ISA based MTD system. Unlike most existing MTD or code randomization techniques [19, 30, 31], HeterSec randomizes the code execution path by switching ISAs at runtime. From the protected process's perspective, it runs on top of a dynamic hardware environment with ISA diversity. Therefore, it would be hard for an attacker to prepare the exploit payload, for example, finding the correct ROP gadget chain or accurately measuring the hardware timing for side-channel attacks. When the process execution encounters a potential ISA switching point, the runtime will randomly decide which ISA the process will execute on in next step. Those ISA switching points are similar to the randomization points in existing code re-randomization works [8, 81], except existing randomization techniques update the code pointer references while HeterSec updates the architecture related states (e.g., stack slots, register set). Although the implementation sounds straightforward, there are some subtle issues when implementing such a system on multi-ISA architecture.

Pointer and architecture specific structure handling is one such case. Some system calls return with data updated to the userspace. Linux handles block data copying between userspace and kernel with pointers and helper functions such as `copy_to_user()`. When calling a system call across nodes, the follower OS context has to make sure any userspace memory updates are synchronized to the local node. In our implementation, most of the userspace memory updates caused by remote system calls will be synchronized correctly with the help of on-demand page synchronization. However, we noticed that Linux maintains slightly different format of some data structures on ARM64 and x86_64. The `struct stat` and `struct epoll_event` are two such cases. The `struct epoll_event` on x86_64 Linux is enforced to have the same alignment as that structure in 32-bit Linux (with `packed` attribute) in order to make 32-bit simulation easier. On the other hand, the ARM64 kernel does not enforce such alignment.

To solve this issue, we converted the structure formats in

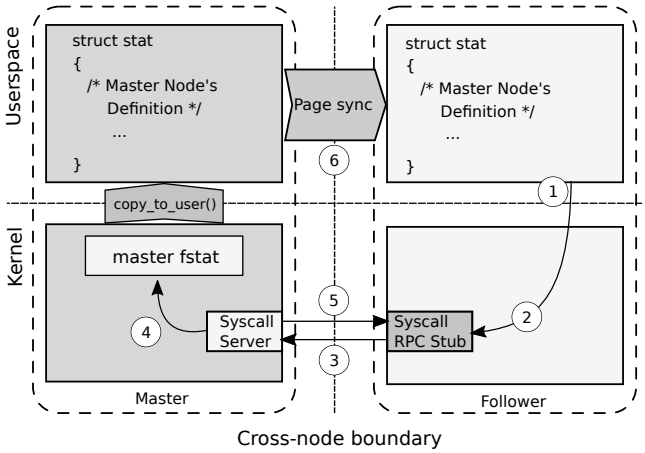

Figure 3: Program flow for an example `fstat` system call executed on the follower node.

musl-libc headers on the follower node to mimic the layout of the master's format. When the follower OS issues a system call RPC, the master OS handles the request and updates the memory references in its own address. The page synchronization handler forwards the change to the follower OS. Figure 3 represents the different stages a system call like `fstat` goes through when called on the follower node. First, the syscall enters the kernel on the follower and calls the RPC stub in Step 2. This RPC stub then communicates to the syscall server on the master node over the messaging API in Step 3. Next, in Step 4 the syscall server calls `fstat` on behalf of the follower, which subsequently completes and returns to the stub as shown in Step 5. It also copies the data to the master's userspace memory. This userspace data is synchronized to the userspace of the follower through page synchronization shown in Step 6 to maintain the illusion to the user program that its own kernel performed the syscall operation.

Randomization and transformation library: As a working proof of concept for the Multi-ISA MTD idea, we implemented an MTD randomization library. It makes decisions on whether a process should execute on a particular node based on random numbers generated from `/dev/urandom`. The probability threshold is also read in from a configuration file (in root mode) at runtime. This enables us to modify MTD switch probability without needing to recompile the application. We leverage the transformation library in Popcorn compiler framework to transform the code execution states between multi-ISA nodes [4].

ISA-switching point insertion, modular compilation, and MTD region activation: When compiling these production level applications the main goal was to make the process as unobtrusive as possible, only generating required metadata in relevant or vulnerable functions and files. The HeterSec framework allows the user to compile specific source files and generate stack transformation metadata for only those sources, effectively enabling MTD functionality for certain parts of the program. While this functionality only works on a file-based granularity, one step of the compilation process includes an LLVM pass to add in calls to the randomization library mentioned in the previous section. These calls are added to the instrumented function prologues and epilogues and can be individually activated on each architecture through configuration files specifying the individual call-sites to activate. Once activated, whenever the program enters or exits these functions it checks with the MTD randomization library if it should switch ISAs, giving us granularity at a function level. We select the source code that contains the critical path for most of the work (e.g., the event loops), and compile them with ISA-switching points instrumented. For example, on Nginx we selected function call paths like `ngx_process_events_and_timers()` in `event/ngx_event.c`. For Redis, we selected similar functions in event loop path, for example `processTimeEvents()` located in `ae.c` that calls the `serverCron()`. These functions are called at a frequency of `server.hz` which defaults to 10 hz. By targeting where we place these checks across the program through this modular compilation, we avoid unnecessary calls to the MTD randomization library, reducing the overhead of the HeterSec framework.

3.4.2 Multi-ISA MVX

The second security application is a heterogeneous-ISA based multi-variant execution system. Similar to a traditional MVX system, the HeterSec MVX also has one leader variant and one follower variant. The leader runs on the master OS with full access to system resources, while the follower is only allowed to execute computational and memory-related code. Since there is only one valid copy of system resources (i.e., opened file, socket and event descriptors), the MVX engine should have the ability to guarantee two program variants can execute simultaneously over a single set of system resources. The HeterSec MVX engine uses system call simulation to synchronize the state across the variants. Specifically, an MVX monitor intercepts the selected system calls from the leader variant's execution and forwards the system effects (e.g., memory update) to the follower. The MVX monitor verifies the system call return values between the running variants and also captures any memory fault to detect divergent (and potentially malicious) behaviors.

In HeterSec, the MVX engines are located inside each distributed kernel as shown in Figure 2. At runtime, the MVX engine on the follower OS verifies whether a system call should be simulated or directly passed through to the local kernel. For system calls tagged for pass-through, the follower OS serves the HeterSec process as usual. For system calls that access the per process descriptors, the HeterSec runtime will verify the descriptor against the virtual descriptor table (described in Section 3.2). Currently, sockets and event poll descriptors are marked as virtual descriptors, meaning that

those descriptors accesses will be simulated on the follower variant by replaying the system call effects from the master OS. For example, the MVX engine will simulate the system call `sys_recvfrom(int sockfd, void *ubuf, ...)` for the follower variant by coping the `ubuf` data from the leader variant to the `ubuf` address in the follower variant. Unless specified, file descriptors by default are marked as real descriptors and they are accessed locally. Similar to the files handled in MTD scenario, variants executed on heterogeneous-ISA nodes have to load shared libraries in different ELF formats. We require the user to manually copy all the necessary files before starting the application as a MVX process. This procedure can also be made automatic by using a NFS to synchronize these files across nodes.

There are other subtle issues when implementing MVX on multi-ISA nodes. One issue is the default libc libraries on two nodes could potentially cause differing system call sequences. To prevent false positives, we compile the application source code and link the object files with the musl libc library generated from the same source. As a result, the system call sequences are almost the same across the binaries on different architecture, with the exception of a few thread initialization functions such as `set_tid_address(int *tidptr)`. In this case, we just ignore performing comparisons on such system call executions. In addition, the system call numbers (and some names) are different in ARM64 and x86_64 architectures and therefore cannot be directly mapped across multi-ISA nodes. For instance, the ARM64 Linux kernel has replaced the `open` system calls with `openat`. This is also the case for several other system calls with *"at"* suffix. Consequently, we cannot forward a system call directly across machines using its number on any particular architecture. Instead, we maintain a system call number translation table, so that any system call (number) being sent to the counterpart node for simulation will be translated to the corresponding system call number first. In our current MVX engine implementation, we do not handle multi-threaded applications. However, we believe a deterministic multi-threading library [47] can be used in HeterSec to solve that issue.

We implemented two types of multi-ISA MVX where the monitor resides (1) in a separate process using `ptrace` to check the application and (2) in the kernel as a Linux kernel module. The `ptrace`-version MVX monitor is used as the developing mode, as it is easier to debug the monitor code without rebooting the operating systems. Similar to some existing MVX works, the `ptrace` version MVX uses the `ptrace` parent process as the MVX monitor, leveraging `ptrace` primitives to intercept and simulate the system calls. It implements a shared ring buffer to pass events (e.g., the syscall return values, or the modifications of data structures) between nodes using a FIFO queue policy in order to maintain sequential consistency. The kernel module version of MVX can be used for deployment because of the better performance. The MVX engine in the kernel intercepts the system calls by wrapping and instrumenting the system call handlers. It also registers the MVX engine code with the HeterSec message layer for fast cross-node messaging. As a result of moving the implementation inside the kernel scope, the system call interception cost and communication latency are both lower than the `ptrace`-based prototype (in Section 4.2).

3.5 Implementation

We implemented a prototype of HeterSec on a x86_64 and ARM64 machine pair, connected using a *Mellanox ConnectX-4* InfiniBand network. The synchronized address space was implemented by placing hooks in the page table handler in the kernel virtual memory subsystem (e.g., hooking the *vma* and *pte* operations [17]). When the protected process is executed on the follower OS, the follower OS kernel handles the page fault by fetching pages from the master OS. The master OS kernel maintains a *vma server* and *page server* which work together to serve the missing pages for the follower OS and invalidate dirty pages (those replicated pages being written). Thus any updates on HeterSec protected process space are synchronized across machine boundaries. HeterSec uses a fast in-kernel message handling layer to send messages across nodes. Since it directly involves the kernel network drivers (e.g., RDMA over Gigabit Infiniband), the cost of switching between user-space and kernel-space is eliminated. Sending messages back and forth are relatively cheap between nodes. For example, the round-trip latency averages 17μs on RDMA in our micro-benchmark test, as described in Section 4.2.

We also implemented the system call RPC server to communicate over a fast message handling layer which similarly rides on RDMA over Infiniband. Similar to cross-node page and VMA handling, the master OS kernel registers a *system call server* in the message handling layer. The first time a process issues the `sys_hscall` system call (either an MTD or MVX), both master and follower kernels will mark that process as a *HeterSec process* (we introduce a flag in `task_struct`). For system calls that manipulate cross-node state (e.g., file, socket and event poll), the follower OS kernel verifies the file descriptor against the VDT to decide whether to invoke the remote system call handler in the master OS or execute them locally. Note that the follower kernel will only check system calls of HeterSec processes, any other processes will be free from this inspection.

To enable cross-ISA program state transformation, we leverage the open source Popcorn compiler [4–6] to embed all the ISA related metadata into the executable. Such information includes the ISA specific instructions, the state relocation mapping, as well as the ISA-switching points. The state relocation mapping is used at each ISA-switch point, with which a translation library transfers the currently running process state (e.g., register states, stack slots, etc.) from one ISA to another. The compiler was built on LLVM, and all the ISA specific code instrumentation was implemented as several

middle-end and backend passes. When compile applications into ISA-specific binaries, we use the same LLVM IR to generate the assembly code for each architecture. Therefore, the stack variables in different architecture can be mapped based on the same origin in IR.

4 Evaluation

In this section, we evaluate the HeterSec prototype as well as its two applications in terms of the security benefits and the performance overhead. All the experiments were evaluated on an x86_64 and ARM64 machine pair. The x86_64 server contains an Intel Xeon E5-2620v4 CPU with the clock speed of 2.1GHz. The ARM64 server contains a Cavium ThunderX CPU (ARMv8.1) with clock speed of 2.0GHz. The two machines are equipped with 32GB and 128GB of DRAM respectively, and they are connected using Mellanox ConnectX-4 InfiniBand with a bandwidth of up to 56 Gbps.

4.1 Security Analysis

Similar to existing diversification-based defense systems, HeterSec also leverages randomized and unknown target process address information (a.k.a ASLR) for the baseline security. However, heterogeneous-ISA based approaches could bring an additional layer of ISA diversity for the process, making it harder for attackers to generate payloads that fit both architectures. Similar to most of the existing diversification systems, we assume the attackers have remote access to the target process with a known interface (e.g., connection sockets). However, HeterSec provides a black box of ISA diversified instances to attackers. With HeterSec, we can leverage the ISA divergent hardware and compilation toolchain to generate program instances with differing instruction sets. The generated application instances also possess different calling conventions, variable register usages, and differential stack layouts. For example, ARM64 allows at most 8 general-purpose registers (`x0` - `x7`) to be used for passing function call parameters; while x86_64 only has at most 6 general-purpose registers for passing parameters. In terms of the system call, ARM64 uses `x8` register for system call number and `x0` for system call return value; while x86_64 uses `rax` for both system call number and return value. Furthermore, most security essential system calls have different system call numbers in the two architectures (e.g., the system call number of `execve` is 59 on x86_64 and 221 on ARM64). This altogether brings extra difficulties for attackers to launch an attack by, for example, return oriented programming.

One observation is that stack operations behave differently on ARMv8 and x86_64. ARMv8 stores the frame pointer (FP) and the link register (LR) both on the lowest address of the stack frame. Whereas x86_64 pushes the instruction pointer (RIP) and the stack base pointer (RBP) into the highest address of the stack frame. The slight difference in control pointer location will make it hard for most of the stack based control flow hijacks to work on both instances. To further prove that the ISA diversified instances will have differing memory layouts, we wrote a tool utilizing ptrace and capstone [65] to dump the code and data pointers of a running process. We examined the potential pointers in `.data`, `.stack`, and `.heap`, and found 7846 pointers in the x86_64 version of lighttpd while there were 10385 pointers in a lighttpd web server running on ARM64. Despite the large number of pointers found in each binary we only found 3 pointers which had overlapping addresses between the two lighttpd processes running on these different ISAs. In the above mentioned experiment, we only examined the pointers with their relative addresses from the base of code segment. That means in reality, there will be almost zero chance of overlapping pointers, since ASLR disturbs the base code addresses of those program instances [2]. In addition, we also examined some real-world exploits on HeterSec environment. One example is CVE-2013-2028, in which an integer overflow and a buffer overflow in the Nginx `ngx_http_read_discarded_request_body()` function are used to gain control over the execution flow and carry out ROP attacks [10]. To trigger the vulnerability an attacker first sends a HTTP chunked request with a large chunked length, resulting in a negative integer to be casted to an unsigned `size_t` type. Subsequently this causes a `recv` call to read in a value larger than the buffer size from the client, leading to a buffer overflow. We ran a ROP attack script leveraging the CVE-2013-2028 buffer overflow [74] and while it was able to execute and trigger the vulnerability on an x86_64 machine, the script failed on the ARM64 machine and caused the Nginx process to crash and restart. The stack layouts between architectures differ therefore the address at which the overflow gains control over the program control flow are not the same.

Another interesting benefit of a multi-ISA security system is that it could potentially raise the bar for micro-architecture attacks [37, 46]. This is due to the fact multiple attack primitives have to be implemented differently on different architectures. For example, cache timing measurement and cache flush have different implementation details. In terms of cache timing measurement, attackers could use an unprivileged `rdtsc` instruction on x86_64 hardware. However, the similar performance counter is only accessible in kernel space on both ARMv7 and ARMv8 processors. Similarly, attackers can directly flush the cache line with `clflush` on x86_64, but have to carefully construct a memory access footprint that defeats the cache replacement policy in order to flush the cache line on ARM processors [45]. The run-time cache layout and timing diversities increases the cost to launch such attacks. Besides the diversified instance memory layout and the micro-architecture behaviors, multi-ISA software diversification could also allows the protected process to hybridize the architecture specific features for increased security. For example, the protected process running on x86_64 can potentially

switch to ARMv8 and validate whether the pointers were modified by attackers with ARMv8 pointer authentication, while still making use of the Intel MPK or MPX hardware features to secure memory page accesses and check boundaries [43, 50, 54, 77].

4.2 Performance Evaluation

To evaluate the performance impact on multi-ISA security applications, we first report the costs involved in cross-node operations such as remote system calls and ISA-switches. Next, we evaluated HeterSec with real-world applications.

Micro-benchmark To get a breakdown of these costs, we implemented a simple micro-benchmark to execute code remotely on the follower node, which triggers 100,000 ISA-switches. We measured the network latency on an ARM64 machine node using a x86_64 node as the follower. As shown in Table 1, a remote system call like `getpid()` imposes an additional ∼17.6 μs overhead when being called compared with the native execution of the `getpid()` system call. The primary reason for this overhead is the unavoidable communication cost brought by the dual-node architecture. The result matches the raw network ping-pong micro-benchmark (∼17.62 μs), in which we wrote a simple kernel module sending 100,000 short messages back and forth between the two machines. Interestingly, this cross-node network latency is much smaller than the network latency observed using Linux `ping` command (∼112 μs). This is because the HeterSec messaging APIs are implemented in the kernel, thus it avoids the complicated TCP/IP network stack and the user/kernel context switch cost. We also observed the ISA switching cost (∼ 504.85 μs) in our micro-benchmark is higher than a remote system call, this is mainly caused by the cross-ISA program state transformation and the page synchronization.

Table 1: The cost (in μs) of remote system call and ISA-switch compared to local getpid() system call on x86_64.

Operations	Latency (in μs)
getpid()	0.47 ± 0.01
remote getpid()	18.09 ± 0.36
raw network ping-pong	17.62 ± 0.33
ping latency	112 ± 15
ISA-switch	504.85 ± 4.70

Application Benchmarks We selected the *nbench* benchmark suite [12], two web server applications – *Nginx* and *Lighttpd*, an in-memory database *Redis* server and a file compression utility GNU *gzip*. We used nbench to measure the performance of HeterSec on CPU and memory intensive workloads. Nbench is a compute, FPU and memory intensive benchmark suite containing some common computation

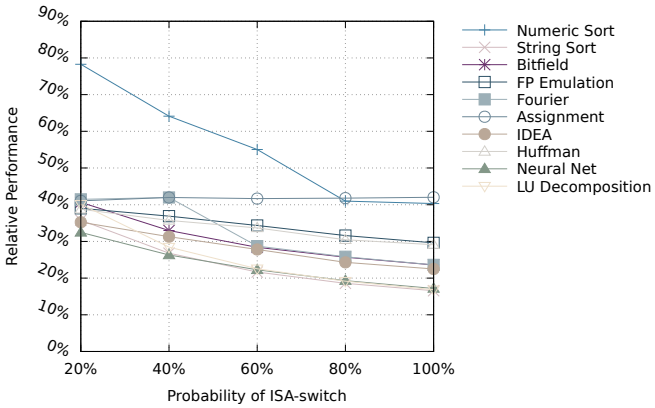

Figure 4: Performance of `nbench` with the probability of 20%, 40%, 60%, 80% and 100% to switch to the counterpart ISA respectively. The numbers are normalized with zero ISA switch, execution on the x86_64 node.

workloads, such as string sorting and neural network back propagation. We measured the performance overhead of using multi-ISA hardware under the security scenarios mentioned in Section 3.4 and reported the performance overhead incurred.

We first measured the performance impact of running nbench under variable probabilities to switch to the counterpart node with different ISA (lthe MTD scenario). In the experiment, we started the nbench program on the x86_64 node; the code will be executed randomly on each node afterwards. We measured the execution time of each test case and normalized to zero probability of ISA-switch (all code executed on x86_64 node). We show the normalized performance overhead numbers in y-axis of Figure 4. As expected, the performance decreases as the probability to perform an ISA-switch increases. This is because the ISA-switch is a relatively expensive operation; the higher chance of ISA-switches during application execution, the more overall performance overhead each application could have. When the first time program execution switches to the counter part node, HeterSec kernels have to load the code and setup the kernel data structure. This explains the reason that some benchmarks lose 50% performance even under 20% chance of ISA-switch. Overall, execution transfer across nodes contributes mainly for the performance degradation. Since the nbench is CPU and memory intensive, any latency incurred during the execution will have significant impact on relative performance.

To prove the feasibility of HeterSec on real-world applications, we evaluated the performance impact of executing Nginx and Redis in HeterSec MTD mode. Nginx and Redis are applications which are used in various commodity systems and best reflect the types of overheads which would be seen when deploying HeterSec in the field. We used ApacheBench to generate the HTTP requests to the web servers and queried for a web page of 4 KB size for 1 million times. We first run ApacheBench on a laptop located in the same LAN of the

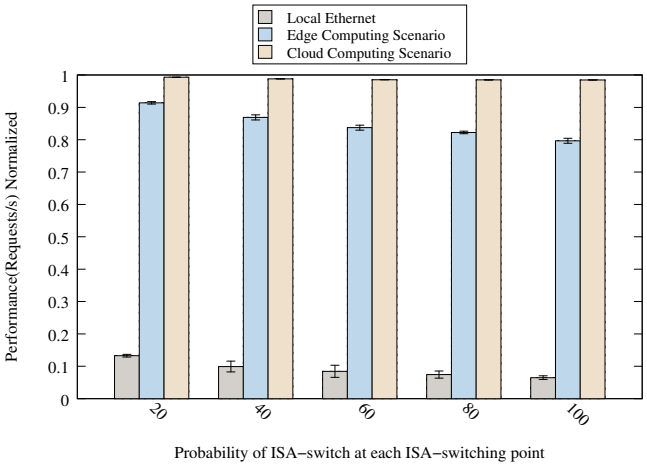

Figure 5: Performance of Nginx (requests/s) with variable probabilities to switch ISAs at every ISA-switching point.

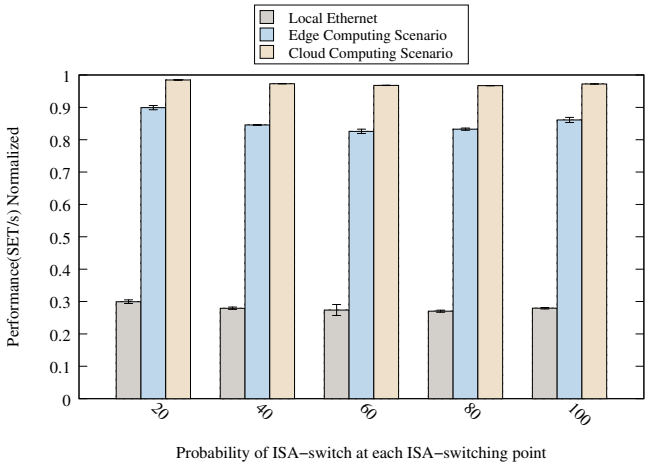

Figure 6: Performance of Redis (SET instructions/s) with variable probabilities to switch ISAs at every ISA-switching point.

target HeterSec machine pair. The laptop and the target machine pair are connected using a 10Gbps Ethernet with about 0.4 ms latency. We also run our test with artificial network latency of 10 ms and 40 ms respectively. The 10 ms latency is to emulate the typical latency seen in Edge Computing scenarios [14], while the 40 ms network latency could be seen as the minimal network overhead between two availability zones of the same region in the Amazon Web Services (AWS) cloud [66]. We manually configured the randomization probability at each ISA-switching point, and ran each test case 5 times. The average value and the standard deviations are reported in Figure 5. With a local network connection, Nginx on HeterSec performs at only about 11% throughput compared to the baseline. This is because internally an ISA-switch brings some additional costs of cross machine communication. Although the inter machine communication has been optimized by using fast in-kernel message API, the time spent on inter machine communication dominates the total request handing time. However, if we consider a real network scenario such as edge or cloud, HeterSec incurs a reasonable overhead. For example, we only observe a 10%-20% performance overhead depending on the frequency we trigger the migrations under 10 ms network latency (the edge computing scenario in Figure 5). At 100% ISA switch probability this equates to 5 switches per request, or about 3800 ISA switches per second. When testing on network designed to emulate the cloud (40 ms latency), the throughput of Nginx shows a very small drop in performance even with a 100% probability to switch ISAs (the cloud scenario in Figure 5). Note that HeterSec kernel brings minimal performance overhead to non-HeterSec processes. For example, the vanilla Nginx performs 22357.5 req/s on vanilla Linux kernel, whereas it performs 22273.8 req/s on HeterSec kernel (∼0.37% overhead).

We observed similar results when running redis-benchmark to measure the throughput of Redis SET instructions. As shown in Figure 6, Redis performs about 30% throughput when running on HeterSec compared with the native execution. However, the overhead drops to 15% and 2% when running the benchmark over edge and cloud computing cases respectively. We set the ISA switch points in a periodic job for the Redis evaluation which resulted in about 20 ISA switches per second, pegged to the `server.hz` value. Interestingly enough, we saw a slight throughput improvement when we increased the ISA-switching probability threshold from 80% to 100%. This is likely due to the deterministic execution flow transfer avoiding destroying the branch prediction. The results show that although the frequent ISA-switch is expensive, it is feasible to use for server applications in real-world scenarios.

Next, we report *the performance of two heterogeneous ISA multi-version execution prototypes*. As mentioned in Section 3.4.2, the `ptrace` version multi-ISA MVX prototype is used to find out all the necessary system calls for simulation, as it is easier to debug with an userspace MVX engine. The MVX engine running in HeterSec kernels can achieve better performance. In our experiment, both MVX prototypes use the ARM64 node to launch the master variant, and offload the follower variant to the x86_64 node. The cost of MVX are mostly from the program state synchronization in between the two variants. For example, the master variant has to wait the system call simulation to be finished on the follower side in order to continue the execution (a.k.a., lock-step check).

We evaluated the two MVX prototypes with *nbench*, *gzip* and *Lighttpd* web server. *Gzip* and *Lighttpd* are two I/O intensive applications. In *gzip* test case, we randomly generate files in different size from `/dev/urandom`. We also used ApacheBench to generate workloads for *Lighttpd* web server. We run all the benchmarks with both kernel-based MVX and ptrace-based MVX prototypes. Figure 7 shows the normalized performance evaluation results using the vanilla application

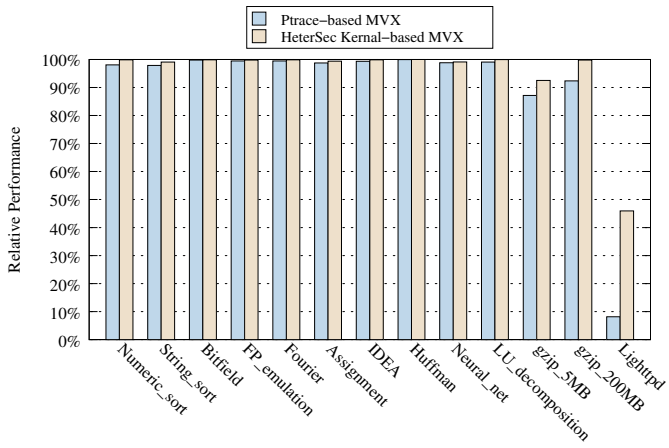

Figure 7: Relative performance of nbench, gzip, and Lighttpd running on the HeterSec kernel-based MVX and the ptrace-based MVX.

running on the ARM64 node as the baseline. For most of the CPU and memory-intensive workloads, kernel-based MVX and ptrace-based MVX have similar performance overheads. This is because most of the system calls in computation-intensive applications do not need to be simulated in the MVX engine. For I/O-intensive applications, both MVX engines process and check on descriptor related system calls such as `read/write(v)`. Overall, both multi-ISA MVX engines introduce about 10% overhead for the *gzip* benchmark. Since we duplicated the files on both nodes, there is no need to transfer data between nodes. For the web server application, the MVX engines have to simulate a number of network I/O related system calls, including `accept4`, `socket`, `sendfile` and `recvfrom`, etc. In general, the HeterSec kernel-based MVX engine pulls down the *Lighttpd* throughput to about 50% of its native performance. However, that performance is still better than the ptrace-based MVX engine (∼10x) and the MVX engine in DMON (∼5.43x) [73].

5 Related Works

The first category of related work is the *various techniques for software diversity* [42]. An important assumption for a software attack is the attacker could have the information of the target system [16, 59, 60], or at least by chance to obtain such information by, for example, brute forcing [10,59]. It makes attacks easier if the code itself and the defense mechanisms are static. Software diversity provides uncertainty for the target system, which breaks the static nature of the target and thus increases the cost of an attack. For example, one of the notable software diversification techniques is ASLR (for most cases, in the form of code randomization) [3, 8, 13, 26, 36, 63, 78, 81]. Previous research demonstrated the effectiveness of code randomization at program module level [63], page level [3], function level [36], basic block level [13, 78], or even instruction level [26]. Some latest research further show the feasibility of ASLR at runtime, making the code layout re-randomized for a given period of time [8, 13, 81]. HeterSec extends this research line by exploring the feasibility of using heterogeneous instruction set to diversify the program.

Multi-version execution is another concrete technique of software diversity. Instead of randomizing a single code instance, MVX engines run multiple variants of program instances simultaneously [15, 38, 51, 57, 58, 72, 73, 83]. Those variants are different in memory layout, so that a malicious input might trigger the vulnerable code in one variant but likely to fail on other variants. Such memory layout differences could be non-overlapping memory map [38, 57, 83], reverse stack growth [58], etc. Recently, researchers also proposed to apply MVX inside Linux kernel, to detect kernel bug exploits [83]. DMON is a very recent and concurrent work using distributed heterogeneous-ISA machines for multi-version execution [73]. DMON shows that MVX with heterogeneous ISA setting can achieve better effectiveness for advanced code reuse attacks, such as the position-independent ROP [23, 73]. As we have compared in Section 2.2, DMON focuses on a heterogeneous-ISA MVX engine only, whereas HeterSec is proposed as a general framework. The multi-ISA MVX engine is a showcase of the HeterSec application scenarios.

Another category of the related work includes the split-interface systems [18, 19, 62, 75] and the multikernel OSes [4, 6, 7, 79]. The split-interface systems normally leverage two compartments to separate and isolate program code execution or secret data access. For example, proxos [62] splits the application execution into trusted and untrusted parts. The trusted part of the execution is isolated in a separate private VM, while the untrusted code can only communicate with the trusted code through a proxy OS. Nested kernel [18] and SecPod [75] split the OS kernels into isolated components for enhanced kernel security. Isomeron [19] on the other hand splits the code execution between two diversified variants. By randomly "flip-coin" selecting the next function to be executed, Isomeron randomizes the execution path to mitigate conventional code reuse attacks [19]. HeterSec shares the same idea of splitting interface to secure application execution, but HeterSec further enhances the execution security by split-executing code on two ISA-diversified nodes.

The multikernel OS treats a multi-core machine as a distributed network of independent cores. A number of systems leverage multiple OS kernels to manage the heterogeneous and multi-core machines in a divide and conquer way [4, 6, 7, 79]. For example, Barrelfish [7] runs multiple OS kernels on top of a multi-core machine in order to make multi-thread application performance scalable. Similarly, fos tackles the scalability issues by factoring the OS into micro-kernel components [79]. Popcorn Linux is a most similar work that runs multikernel on heterogeneous hardware [4, 6]. Popcorn Linux focuses on single-threaded HPC applications migra-

tion; on the other hand, HeterSec targets a not well explored research area – the feasibility of securing an application execution with ISA diversity. Furthermore, server applications and multi-threaded applications are supported with HeterSec.

6 Conclusion

In this paper, we explored the missed research space of securing application execution with ISA diversity. We described the design and implementation of HeterSec, a framework to improve application security with ISA heterogeneity. HeterSec enables HeterSec processes to leverage the diversified ISAs as an additional layer of dynamic defense. HeterSec was built with several compiler and kernel extensions to facilitate processes running on heterogeneous hardware in a security enhanced manner. The two security applications built on HeterSec show that it is feasible to leverage the existing heterogeneous hardware to improve application security.

The source code of HeterSec is publicly available as part of the Popcorn Linux project at http://popcornlinux.org.

Acknowledgments

We would like to thank the anonymous reviewers for their insightful comments. This work is supported in part by grants received by Virginia Tech including that from the US Office of Naval Research (ONR) under grants N00014-18-1-2022, N00014-16-1-2104, and N00014-16-1-2711, and from NAVSEA/NEEC under grant N00174-16-C-0018. Kim's work at Virginia Tech (former affiliation) was supported by ONR under grants N00014-16-1-2711 and N00014-18-1-2022. Olivier's work at Virginia Tech (former affiliation) was supported by ONR under grants N00014-16-1-2104 and N00014-18-1-2022. Lyerly's work at Virginia Tech (former affiliation) was supported in part by NAVSEA/NEEC under grant N00174-16-C-0018.

This work is also supported by Electronics and Telecommunications Research Institute (ETRI) grant funded by the Korean government (20ZS1310).

References

[1] ARM Limited (or its affiliates). ARM BIG.LITTLE. https://www.arm.com/why-arm/technologies/big-little, Accessed: 2020-07-08.

[2] Linux Kernel Address Space Layout Randomization. http://lwn.net/Articles/569635/.

[3] Michael Backes and Stefan Nürnberger. Oxymoron: Making Fine-grained Memory Randomization Practical by Allowing Code Sharing. *Proc. 23rd Usenix Security Sym*, pages 433–447, 2014.

[4] Antonio Barbalace, Robert Lyerly, Christopher Jelesnianski, Anthony Carno, Ho-Ren Chuang, Vincent Legout, and Binoy Ravindran. Breaking the boundaries in heterogeneous-ISA datacenters. In *ACM SIGPLAN Notices*, volume 52, pages 645–659. ACM, 2017.

[5] Antonio Barbalace, Binoy Ravindran, and David Katz. Popcorn: a replicated-kernel os based on linux. In *Proceedings of the Linux Symposium, Ottawa, Canada*, 2014.

[6] Antonio Barbalace, Marina Sadini, Saif Ansary, Christopher Jelesnianski, Akshay Ravichandran, Cagil Kendir, Alastair Murray, and Binoy Ravindran. Popcorn: Bridging the programmability gap in heterogeneous-isa platforms. In *Proceedings of the Tenth European Conference on Computer Systems*, EuroSys '15, New York, NY, USA, 2015. Association for Computing Machinery.

[7] Baumann, Andrew and Barham, Paul and Dagand, Pierre-Evariste and Harris, Tim and Isaacs, Rebecca and Peter, Simon and Roscoe, Timothy and Schüpbach, Adrian and Singhania, Akhilesh. The multikernel: a new OS architecture for scalable multicore systems. In *Proceedings of the ACM SIGOPS 22nd symposium on Operating systems principles*, pages 29–44. ACM, 2009.

[8] David Bigelow, Thomas Hobson, Robert Rudd, William Streilein, and Hamed Okhravi. Timely rerandomization for mitigating memory disclosures. In *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*, pages 268–279. ACM, 2015.

[9] Nathan Binkert, Bradford Beckmann, Gabriel Black, Steven K Reinhardt, Ali Saidi, Arkaprava Basu, Joel Hestness, Derek R Hower, Tushar Krishna, Somayeh Sardashti, et al. The gem5 simulator. *ACM SIGARCH Computer Architecture News*, 39(2):1–7, 2011.

[10] Andrea Bittau, Adam Belay, Ali Mashtizadeh, David Mazieres, and Dan Boneh. Hacking Blind. In *Security and Privacy (SP), 2014 IEEE Symposium on*, pages 227–242. IEEE, 2014.

[11] Shekhar Borkar. Thousand core chips: a technology perspective. In *Proceedings of the 44th annual Design Automation Conference*, pages 746–749. ACM, 2007.

[12] BYTEmark benchmark. Linux/Unix nbench. http://www.math.utah.edu/~mayer/linux/bmark.html, Accessed: 2020-07-08.

[13] Yue Chen, Zhi Wang, David Whalley, and Long Lu. Remix: On-demand live randomization. In *Proceedings of the Sixth ACM Conference on Data and Application Security and Privacy*, pages 50–61. ACM, 2016.

[14] Sharon Choy, Bernard Wong, Gwendal Simon, and Catherine Rosenberg. The brewing storm in cloud gaming: A measurement study on cloud to end-user latency. In *Proceedings of the 11th annual workshop on network and systems support for games*, page 2. IEEE Press, 2012.

[15] Benjamin Cox, David Evans, Adrian Filipi, Jonathan Rowanhill, Wei Hu, Jack Davidson, John Knight, Anh Nguyen-Tuong, and Jason Hiser. N-variant systems: A secretless framework for security through diversity. In *USENIX Security Symposium*, pages 105–120, 2006.

[16] Stephen Crane, Christopher Liebchen, Andrei Homescu, Lucas Davi, Per Larsen, Ahmad-Reza Sadeghi, Stefan Brunthaler, and Michael Franz. Readactor: Practical Code Randomization Resilient to Memory Disclosure. In *36th IEEE Symposium on Security and Privacy (Oakland)*, May 2015.

[17] Daniel, P and Marco, Cesati and others. Understanding the Linux kernel, 2007.

[18] Nathan Dautenhahn, Theodoros Kasampalis, Will Dietz, John Criswell, and Vikram Adve. Nested kernel: An operating system architecture for intra-kernel privilege separation. In *Proceedings of the Twentieth International Conference on Architectural Support for Programming Languages and Operating Systems*, pages 191–206. ACM, 2015.

[19] Lucas Davi, Christopher Liebchen, Ahmad-Reza Sadeghi, Kevin Z Snow, and Fabian Monrose. Isomeron: Code Randomization Resilient to (just-in-time) Return-oriented Programming. *Proc. 22nd Network and Distributed Systems Security Sym.(NDSS)*, 2015.

[20] Matthew DeVuyst, Ashish Venkat, and Dean M Tullsen. Execution migration in a heterogeneous-isa chip multiprocessor. In *ACM SIGARCH Computer Architecture News*, volume 40, pages 261–272. ACM, 2012.

[21] Hadi Esmaeilzadeh, Emily Blem, Renee St Amant, Karthikeyan Sankaralingam, and Doug Burger. Dark silicon and the end of multicore scaling. In *Computer Architecture (ISCA), 2011 38th Annual International Symposium on*, pages 365–376. IEEE, 2011.

[22] Peter N Glaskowsky. NVIDIA's Fermi: the first complete GPU computing architecture. *White paper*, 18, 2009.

[23] Enes Göktas, Benjamin Kollenda, Philipp Koppe, Erik Bosman, Georgios Portokalidis, Thorsten Holz, Herbert Bos, and Cristiano Giuffrida. Position-independent code reuse: On the effectiveness of aslr in the absence of information disclosure. In *2018 IEEE European Symposium on Security and Privacy (EuroS&P)*, pages 227–242. IEEE, 2018.

[24] Boncheol Gu, Andre S. Yoon, Duck-Ho Bae, Insoon Jo, Jinyoung Lee, Jonghyun Yoon, Jeong-Uk Kang, Moonsang Kwon, Chanho Yoon, Sangyeun Cho, Jaeheon Jeong, and Duckhyun Chang. Biscuit: A framework for near-data processing of big data workloads. In *Proceedings of the 43rd International Symposium on Computer Architecture*, ISCA '16, pages 153–165, Piscataway, NJ, USA, 2016. IEEE Press.

[25] Nikos Hardavellas, Michael Ferdman, Babak Falsafi, and Anastasia Ailamaki. Toward dark silicon in servers. *IEEE Micro*, 31(4):6–15, 2011.

[26] Jason Hiser, Anh Nguyen-Tuong, Michele Co, Matthew Hall, and Jack W Davidson. ILR: Where'd My Gadgets Go? In *Security and Privacy (SP), 2012 IEEE Symposium on*, pages 571–585. IEEE, 2012.

[27] Hong Hu, Shweta Shinde, Sendroiu Adrian, Zheng Leong Chua, Prateek Saxena, and Zhenkai Liang. Data-oriented programming: On the expressiveness of non-control data attacks. In *2016 IEEE Symposium on Security and Privacy (SP)*, pages 969–986. IEEE, 2016.

[28] Intel. *Intel Acceleration Stack for Intel Xeon CPU with FPGAs Core Cache Interface (CCI-P) Reference Manual*, Nov 2019. https://www.intel.com/content/dam/www/programmable/us/en/pdfs/literature/manual/mnl-ias-ccip.pdf.

[29] Intel. Intel Xeon processor scalable family. https://www.intel.com/content/www/us/en/products/processors/xeon/scalable.html, Accessed: 2020-07-08.

[30] Jafar Haadi Jafarian, Ehab Al-Shaer, and Qi Duan. Openflow random host mutation: transparent moving target defense using software defined networking. In *Proceedings of the first workshop on Hot topics in software defined networks*, pages 127–132. ACM, 2012.

[31] Jajodia, Sushil and Ghosh, Anup K and Swarup, Vipin and Wang, Cliff and Wang, X Sean. *Moving Target Defense: Creating Asymmetric Uncertainty for Cyber Threats*, volume 54. Springer Science & Business Media, 2011.

[32] James Jeffers and James Reinders. *Intel Xeon Phi Coprocessor High Performance Programming*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1st edition, 2013.

[33] Norman P Jouppi, Cliff Young, Nishant Patil, David Patterson, Gaurav Agrawal, Raminder Bajwa, Sarah Bates,

Suresh Bhatia, Nan Boden, Al Borchers, et al. In-datacenter performance analysis of a tensor processing unit. In *2017 ACM/IEEE 44th Annual International Symposium on Computer Architecture (ISCA)*, pages 1–12. IEEE, 2017.

[34] Shubham Kamdar and Neha Kamdar. big.LITTLE architecture: Heterogeneous multicore processing. *International Journal of Computer Applications*, 119(1), 2015.

[35] Gaurav S Kc, Angelos D Keromytis, and Vassilis Prevelakis. Countering code-injection attacks with instruction-set randomization. In *Proceedings of the 10th ACM conference on Computer and communications security*, pages 272–280. ACM, 2003.

[36] Chongkyung Kil, Jinsuk Jim, Christopher Bookholt, Jun Xu, and Peng Ning. Address Space Layout Permutation (ASLP): Towards Fine-grained Randomization of Commodity Software. In *Computer Security Applications Conference, 2006. ACSAC'06. 22nd Annual*, pages 339–348. IEEE, 2006.

[37] Paul Kocher, Jann Horn, Anders Fogh, , Daniel Genkin, Daniel Gruss, Werner Haas, Mike Hamburg, Moritz Lipp, Stefan Mangard, Thomas Prescher, Michael Schwarz, and Yuval Yarom. Spectre Attacks: Exploiting Speculative Execution. In *40th IEEE Symposium on Security and Privacy (S&P'19)*, 2019.

[38] Koen Koning, Herbert Bos, and Cristiano Giuffrida. Secure and Efficient Multi-Variant Execution using Hardware-Assisted Process Virtualization. In *2016 46th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, pages 431–442. IEEE, 2016.

[39] Koen Koning, Xi Chen, Herbert Bos, Cristiano Giuffrida, and Elias Athanasopoulos. No need to hide: Protecting safe regions on commodity hardware. In *Proceedings of the Twelfth European Conference on Computer Systems*, pages 437–452. ACM, 2017.

[40] H. Koo, Y. Chen, L. Lu, V. P. Kemerlis, and M. Polychronakis. Compiler-assisted code randomization. In *2018 IEEE Symposium on Security and Privacy (SP)*, pages 461–477, 2018.

[41] Volodymyr Kuznetsov, László Szekeres, Mathias Payer, George Candea, R Sekar, and Dawn Song. Code Pointer Integrity. In *USENIX Symposium on Operating Systems Design and Implementation (OSDI)*, 2014.

[42] Per Larsen, Andrei Homescu, Stefan Brunthaler, and Michael Franz. SoK: Automated Software Diversity. In *Proceedings of the 2014 IEEE Symposium on Security and Privacy*, SP '14, 2014.

[43] Hans Liljestrand, Thomas Nyman, Kui Wang, Carlos Chinea Perez, J Ekberg, and N Asokan. Pac it up: Towards pointer integrity using arm pointer authentication. *arXiv preprint arXiv:1811.09189*, 2018.

[44] Felix Xiaozhu Lin, Zhen Wang, and Lin Zhong. K2: A mobile operating system for heterogeneous coherence domains. In *Proceedings of the 19th International Conference on Architectural Support for Programming Languages and Operating Systems*, ASPLOS '14, pages 285–300, New York, NY, USA, 2014. ACM.

[45] Moritz Lipp. *Cache attacks on arm*. PhD thesis.

[46] Moritz Lipp, Michael Schwarz, Daniel Gruss, Thomas Prescher, Werner Haas, Anders Fogh, Jann Horn, Stefan Mangard, Paul Kocher, Daniel Genkin, et al. Meltdown: Reading kernel memory from user space. In *27th USENIX Security Symposium (USENIX Security 18)*, pages 973–990, 2018.

[47] Tongping Liu, Charlie Curtsinger, and Emery D. Berger. Dthreads: Efficient deterministic multithreading. In *Proceedings of the Twenty-Third ACM Symposium on Operating Systems Principles*, SOSP '11, page 327–336, New York, NY, USA, 2011. Association for Computing Machinery.

[48] Timothy P. Morgan. Tilera rescues CPU cycles with network coprocessors, 2013. https://bit.ly/2DfM53R.

[49] Netronome. Agilio SmartNICs, 2019. https://www.netronome.com/products/smartnic/overview/.

[50] Oleksii Oleksenko, Dmitrii Kuvaiskii, Pramod Bhatotia, Pascal Felber, and Christof Fetzer. Intel mpx explained: A cross-layer analysis of the intel mpx system stack. *Proceedings of the ACM on Measurement and Analysis of Computing Systems*, 2(2):28, 2018.

[51] Pierre Olivier, Antonio Barbalace, and Binoy Ravindran. Multi-variant execution atop a decomposed hypervisor on emerging heterogeneous-isa multicore. 2016. EuroSys'16 (Poster).

[52] Pierre Olivier, Sang-Hoon Kim, and Binoy Ravindran. OS Support for Thread Migration and Distribution in the Fully Heterogeneous Datacenter. In *Proceedings of the 16th Workshop on Hot Topics in Operating Systems*, pages 174–179. ACM, 2017.

[53] Shruti Padmanabha, Andrew Lukefahr, Reetuparna Das, and Scott Mahlke. Dynamos: dynamic schedule migration for heterogeneous cores. In *Proceedings of the 48th International Symposium on Microarchitecture*, pages 322–333. ACM, 2015.

[54] Soyeon Park, Sangho Lee, Wen Xu, Hyungon Moon, and Taesoo Kim. libmpk: Software Abstraction for Intel Memory Protection Keys (Intel MPK). In *2019 USENIX Annual Technical Conference (USENIX ATC 19)*, pages 241–254, 2019.

[55] QEMU. http://www.qemu.org.

[56] Stefan Rusu, Simon Tam, Harry Muljono, Jason Stinson, David Ayers, Jonathan Chang, Raj Varada, Matt Ratta, Sailesh Kottapalli, and Sujal Vora. A 45 nm 8-core enterprise Xeon processor. *IEEE Journal of Solid-State Circuits*, 45(1):7–14, 2010.

[57] Babak Salamat, Todd Jackson, Andreas Gal, and Michael Franz. Orchestra: intrusion detection using parallel execution and monitoring of program variants in user-space. In *Proceedings of the 4th ACM European conference on Computer systems*, pages 33–46. ACM, 2009.

[58] Salamat, Babak and Gal, Andreas and Franz, Michael. Reverse stack execution in a multi-variant execution environment. In *Workshop on Compiler and Architectural Techniques for Application Reliability and Security*, pages 1–7, 2008.

[59] Hovav Shacham, Matthew Page, Ben Pfaff, Eu-Jin Goh, Nagendra Modadugu, and Dan Boneh. On the Effectiveness of Address-space Randomization. In *Proceedings of the 11th ACM Conference on Computer and Communications Security*, CCS '04, pages 298–307, 2004.

[60] Kevin Z Snow, Fabian Monrose, Lucas Davi, Alexandra Dmitrienko, Christopher Liebchen, and Ahmad-Reza Sadeghi. Just-in-time Code Reuse: On the Effectiveness of Fine-grained Address Space Layout Randomization. In *Security and Privacy (SP), 2013 IEEE Symposium on*, pages 574–588. IEEE, 2013.

[61] Avinash Sodani, Roger Gramunt, Jesus Corbal, Ho-Seop Kim, Krishna Vinod, Sundaram Chinthamani, Steven Hutsell, Rajat Agarwal, and Yen-Chen Liu. Knights landing: Second-generation Intel Xeon Phi. *IEEE micro*, 36(2):34–46, 2016.

[62] Richard Ta-Min, Lionel Litty, and David Lie. Splitting interfaces: Making trust between applications and operating systems configurable. In *Proceedings of the 7th Symposium on Operating Systems Design and Implementation*, OSDI '06, pages 279–292, Berkeley, CA, USA, 2006. USENIX Association.

[63] PaX Team. PaX Address Space Layout Randomization (ASLR), 2003.

[64] Marvell Technology. Liquidio ii 10/25gbe Adapter family, 2019. https://bit.ly/2H7NWLk.

[65] The Ultimate Dissassembly Framework – Capstone. http://www.capstone-engine.org/.

[66] AWS Inter-Region Latency. https://www.cloudping.co/.

[67] Reduce TCO with Arm Based SmartNICs. https://community.arm.com/developer/ip-products/processors/b/processors-ip-blog/posts/reduce-tco-with-arm-based-smartnics.

[68] High-Performance Programmable SmartNICs. https://www.mellanox.com/products/smartnic/.

[69] Ashish Venkat, Harsha Basavaraj, and Dean Tullsen. Composite-isa cores: Enabling multi-isa heterogeneity using a single isa. In *25th IEEE International Symposium on High Performance Computer Architecture*. IEEE, February 2019.

[70] Ashish Venkat, Sriskanda Shamasunder, Hovav Shacham, and Dean M Tullsen. Hipstr: Heterogeneous-isa program state relocation. In *ACM SIGARCH Computer Architecture News*, volume 44, pages 727–741. ACM, 2016.

[71] Ashish Venkat and Dean M Tullsen. Harnessing isa diversity: Design of a heterogeneous-isa chip multiprocessor. *ACM SIGARCH Computer Architecture News*, 42(3):121–132, 2014.

[72] Stijn Volckaert, Bart Coppens, and Bjorn De Sutter. Cloning your gadgets: Complete rop attack immunity with multi-variant execution. *IEEE Transactions on Dependable and Secure Computing*, 13(4):437–450, 2016.

[73] Alexios Voulimeneas, Dokyung Song, Fabian Parzefall, Yeoul Na, Per Larsen, Michael Franz, and Stijn Volckaert. Dmon: A distributed heterogeneous n-variant system. *arXiv preprint arXiv:1903.03643*, 2019.

[74] w00d. Analysis of nginx 1.3.9/1.4.0 stack buffer overflow and x64 exploitation (CVE-2013-2028)). https://www.vnsecurity.net/research/2013/05/21/analysis-of-nginx-cve-2013-2028.html.

[75] Xiaoguang Wang, Yong Qi, Zhi Wang, Yue Chen, and Yajin Zhou. Design and Implementation of SecPod, A Framework for Virtualization-Based Security Systems. *IEEE Transactions on Dependable and Secure Computing*, 16(1):44–57, 2019.

[76] Xiaoguang Wang, SengMing Yeoh, Robert Lyerly, Sang-Hoon Kim, and Binoy Ravindran. A Framework to Secure Applications with ISA Heterogeneity. In *The 9th Workshop on Systems for Multi-core and Heterogeneous Architectures (SFMA)*, 2019.

[77] Xiaoguang Wang, SengMing Yeoh, Pierre Olivier, and Binoy Ravindran. Secure and efficient in-process monitor (and library) protection with intel mpk. In *Proceedings of the 13th European Workshop on Systems Security*, EuroSec '20, page 7–12, New York, NY, USA, 2020. Association for Computing Machinery.

[78] Richard Wartell, Vishwath Mohan, Kevin W. Hamlen, and Zhiqiang Lin. Binary stirring: Self-randomizing instruction addresses of legacy x86 binary code. In *Proceedings of the 2012 ACM Conference on Computer and Communications Security*, CCS '12, 2012.

[79] David Wentzlaff and Anant Agarwal. Factored operating systems (fos): The case for a scalable operating system for multicores. *SIGOPS Oper. Syst. Rev.*, 43(2):76–85, April 2009.

[80] Wikipedia. MSI Protocol. https://en.wikipedia.org/wiki/MSI_protocol, Accessed: 2020-07-08.

[81] David Williams-King, Graham Gobieski, Kent Williams-King, James P Blake, Xinhao Yuan, Patrick Colp, Michelle Zheng, Vasileios P Kemerlis, Junfeng Yang, and William Aiello. Shuffler: Fast and Deployable Continuous Code Re-Randomization. In *OSDI*, pages 367–382, 2016.

[82] Yuval Yarom and Katrina Falkner. FLUSH+ RELOAD: A High Resolution, Low Noise, L3 Cache Side-Channel Attack. In *USENIX Security Symposium*, volume 1, pages 22–25, 2014.

[83] Sebastian Österlund, Koen Koning, Pierre Olivier, Antonio Barbalace, Herbert Bos, and Cristiano Giuffrida. kMVX: Detecting Kernel Information Leaks with Multi-variant Execution. In *ASPLOS*, April 2019.