A Framework to Secure Applications with ISA Heterogeneity

Xiaoguang Wang, SengMing Yeoh, Robert Lyerly, Sang-Hoon Kim*, Binoy Ravindran
Virginia Tech, Blacksburg, USA
{xiaoguang,sengming,rlyerly,binoy}@vt.edu
*Ajou University, South Korea
sanghoonkim@ajou.ac.kr

ABSTRACT

Software security attacks are evolving from exploiting common code vulnerabilities to exploiting micro architecture side-channels. Traditional software diversity or code randomization techniques diversify the code memory layout and make it difficult for potential attackers to pinpoint the precise location of the target vulnerability. However, those approaches may not be sufficient enough for the new micro architecture attacks (e.g., Spectre). While some architecture researchers have proposed using diverse ISA configurations to defeat code injection or code reuse attacks, most of these works remain in the simulation stage due to legal, licensing, and verification costs involved in bringing a heterogeneous chip design into physical hardware [39].

In this paper, we report our on-going work of HeterSec, a framework to secure applications *utilizing real world heterogeneous ISA machines*. HeterSec runs on top of the commodity x86_64 and ARM64 machines. It gives the process the ability to dynamically select its underlying ISA environment. Therefore, the protected process would hide the vulnerable targets with the diversified instruction set, or would detect the abnormal behavior by comparing the execution results step-by-step from multiple ISA-diversified instances. To demonstrate the effectiveness of such software framework, we implemented HeterSec on Linux and showed its deployability by running it on a x86_64 and ARM64 machine pair, connected using InfiniBand. We then conduct two case studies with HeterSec. In the first case, we timely randomize the process execution path across the ISA, which achieves similar security guarantees as the existing architecture based solutions. In the second case, we implement a multi-ISA based multi-version execution (MVX) system, providing a stronger security guarantee than current homogeneous-ISA MVX designs.

CCS CONCEPTS

• **Computer systems organization → Heterogeneous (hybrid) systems**; • **Security and privacy → Systems security**; *Software and application security*.

KEYWORDS

Heterogeneous ISA, secure application, operating system, runtime

1 INTRODUCTION

The battles on software security are rapidly evolving. The classic buffer overflow and stack smashing [2] uncovered many of the early software exploits. By leveraging a smashed return address on the stack, attackers can redirect a program's control flow to execute a remote shell. Later techniques such as data execution prevention (DEP) prevented directly executing the injected code [45]. However, the code reuse attacks could still leverage existing code to perform arbitrary memory operations [30, 34]. Researchers have proposed several defense mechanisms to defeat such code reuse attacks. For example, control flow integrity (CFI) checks the program control flow to enforce the runtime control flow follows the control flow obtained from the static program analysis [1, 49]. However, the latest research on function reuse attacks show that even if the control flow is enforced, it is still possible to reuse the existing functions in the target program to launch an attack [15, 38].

Fundamentally, this is caused by the static nature of such defense mechanisms which gives the attackers a time advantage to study the target system and launch the exploit [20]. Address space layout randomization (ASLR) as well as it's advanced variant runtime code re-randomization, try to break the static nature of the process by randomizing the code layout at the initial time or on the fly, to prevent attackers from knowing the vulnerable code locations [9, 12, 44, 47]. Although it has raised the bar and made it more difficult for attackers to discern the precise location of a vulnerability, some research on full function reuse attacks show that it is very hard to re-randomize some immutable function pointers and those unchanged pointers could still be used to launch an attack [15, 38]. Even if the full memory address space is re-randomized, recent researches on micro architecture attacks (such as Spectre and Meltdown [24, 27]) show that it is still possible to steal the secret data from memory by exploiting the processor prefetching vulnerabilities [17].

The attacks mentioned above are hard to detect or prevent. This is because the static nature of the targeting platform is by default not easy to change. This gives attackers time to study the target platform features (e.g., LLC access time [24, 27, 48]), retrieve the leaked information [36] and make the exploit [10]. Runtime ASLR itself may not be sufficient enough to defeat this new breed of attacks, such as micro architecture attacks on CPU pipeline prefetching [24, 27], or DRAM bit flipping [33]. This is caused by most existing randomization techniques work on the same hardware platform, giving the advanced attackers the possibility to study the target hardware features and launch a architecture related attack. Architecture researchers proposed a few systems that implement heterogeneous ISA over one single chip to achieve inter-ISA program state randomization with higher entropy [22, 40]. However, due to a lack of real heterogeneous ISA platforms, it is extremely hard for security researchers to leverage ISA heterogeneity to implement security systems.

In this paper, we propose HeterSec, a framework that facilitates the design and implementation of security systems over heterogeneous ISA. HeterSec works at the operating system and runtime level, giving processes the ability to migrate or cross check between

SFMA workshop'19, March 2019, Dresden, Germany

two machines running on different ISA. To demonstrate the effectiveness, we have built two security applications on top of HeterSec. The first security application enables the random execution between ISA different machines, achieving the similar security guarantee as HIPStR [40] does. The second security application implements a multi-version execution (MVX) [1] system [25]. The traditional MVX runs multiple variants of an application with different memory layouts and checks the running behavior. Since all variants run on the same hardware, the static nature of the hardware platform will remain the same if the attacker launches a micro architecture attack. Our heterogeneous ISA based MVX system could detect such attacks. To the best of our knowledge, HeterSec is the first practical system enables researchers to build security applications on top of the real commodity heterogeneous ISA machines. Overall, we made the following contributions:

- We proposed a software system that can manage the process execution over heterogeneous ISA for security purpose.
- We implemented two security applications on top of such system, namely execution randomization over multi-ISA machines and heterogeneous-ISA based MVX.
- We showed the potential of such security system that could defeat/detect the traditional memory vulnerability exploits as well as the latest micro architecture attacks.

2 RELATED WORKS

In this section, we summarize the related works, describe the motivation, and show the design space of our work.

Software diversity: The first category of related work is about software diversity [26]. An important assumption for a software attack is the attacker could have the information of the target system [14, 35, 36, 43], or at least by chance to obtain such information by, for example, brute forcing [10, 35]. It makes attacks easier if the code itself and the defense mechanisms are static. Software diversity provides uncertainty for the target system. It 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) [5, 9, 12, 18, 23, 37, 44, 47]. Previous researches demonstrated the effectiveness of code randomization at program module level [37], page level [5], function level [23], basic block level [12, 44], or even instruction level [18]. And some latest researches further show the feasibility of ASLR at runtime, making the code layout re-randomized for a given period of time [9, 12, 47].

Multi-version execution is another concrete technique of software diversity. Instead of randomizing a single code instance, MVX creates and runs multiple variants of code in memory simultaneously [13, 25, 29, 31, 32, 41, 42, 50]. 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 [25, 31, 50], reverse stack growth [32], etc. Recently, researchers also propose to apply MVX inside Linux kernel, to detect kernel bug exploits [50]. However, most of the existing works focus on defeating traditional software exploits, which are caused by the vulnerable program logic [42]. After the acceptance of this paper, by a private conversation, the authors of [42] let us know about their work under review. Very few of the MVX system is able to detect the inconsistent behavior caused by architecture level differences. Thus there is an emerging need to detect the new coming architecture related attacks.

Hardware related attacks: Recently, several attacks have surfaced targeting the microarchitecture of modern processors, leveraging race conditions brought about by advanced optimization features such as out-of-order execution [27], and speculative execution [24]. Since these attacks are reliant of hardware microarchitecture, they are capable of breaking security guarantees provided by paravirtualization and containerization. One key piece these attacks hinge on is the usage of cache side-channel timing to leak secrets by using them as an index into an oracle array. By observing the latency differences between hot and cold cache lines over multiple executions an attacker can then discern the values and exfiltrate secret data.

This *Evict+Time* method relies on architecture specific intrinsics such as `_mm_clflush()` on Intel x86_64 to perform a cache eviction [19, 48]. While ARMv8 does have instructions to evict cache lines such as `DC IVAC`, these can be trapped or disabled at Execution Level 0 [3]. Cache timing differences between systems would also make the attack more difficult in a heterogeneous system as microarchitectural state is not migrated between systems.

In addition to these microarchitecture attacks, hardware memory attacks such as Rowhammer have successfully been exploited to gain kernel access [33]. By accessing Dynamic Random Access Memory (DRAM) rows repeatedly attackers successfully cause bit flips in adjacent rows, enabling sandbox security guarantees to be bypassed. However, in order to access specific DRAM rows repeatedly, a cache miss needs to first occur. This is done via the same Intel intrinsic function mentioned previously, `_mm_clflush()`. The same thought process of adding entropy via heterogeneous migration also applies here as this attack relies on consistent accesses to the same DRAM rows in order to trigger the bit flip.

ISA Heterogeneity: Another category of the related work includes a variety of applications on heterogeneous ISA. The ISA heterogeneity has been well studied and broadly used in many parallel programming scenarios [11, 21, 28]. For example, GPUs are generally used to accelerate HPC workloads [28], machine learning [11] and even AI inference [21]. On mobile devices, heterogeneous processors are used to provide a good balance between performance and power consumption (e.g., ARM big.LITTLE [4]). In addition, recent researches also show some interesting use cases of ISA heterogeneity helps to improve energy efficiency and security [6, 8, 40]. For example, one of the most related work of our paper is HIPStR, which implements a heterogeneous-ISA multi-core processor based on the gem5 simulator [40]. The authors showed that the ISA diversity could significantly increase the entropy to defeat a branch of code reuse attacks, such as ROP, JIT-ROP and several evasive variants [30, 34, 36, 40]. Different from the architecture simulation, we built HeterSec on the real world commodity software/hardware stack, and we also show the possibility of broader use cases which benefit from the ISA heterogeneity.

[1] Some literature would also name it multi-variant execution, so the word *version* and *variant* could be used interchangeably.

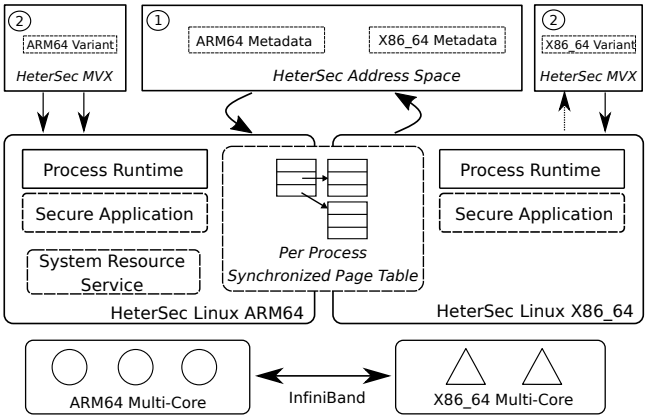

Figure 1: The overview of HeterSec with two security application scenarios. The dotted line indicates the modification over existing software stack.

3 DESIGN

3.1 System overview

HeterSec aims at securing the process execution by utilizing the ISA heterogeneity, for example, randomizing the process execution environment over heterogeneous machines. To achieve that, HeterSec provides *a HeterSec execution environment*. Specifically, it allows the protected process to be executed on machines running with different ISAs as if it runs on a single machine. Figure 1 shows an overview of HeterSec with our multiple modifications on an existing computer system stack. The modifications include both the kernel and the userspace process as shown in dotted line. Figure 1 also shows two secure application scenarios on top of HeterSec. *In the first scenario*, HeterSec timely switches the underlying ISA for the protected application, increases the entropy of the possible program states, and bewitches the attackers from knowing the underlying hardware details. *In the second scenario*, HeterSec launches multiple variants of the program, monitors the variants' execution of system calls, and raises an alert on any execution differences caused by a potential attack.

To support ISA-switching, HeterSec retrofits the HeterSec address space to embed metadata of both ISAs. The metadata contains the current program state and the resource mapping of each ISAs. It runs on top of the HeterSec distributed operating system. 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 underlying memory. The secure application will be loaded by the HeterSec loader. It controls the HeterSec runtime which activates the corresponding security purpose.

The HeterSec has a concept of dominating OS, also referred to as the master OS. The master OS is the OS where we launch the HeterSec process. Correspondingly, we call the OS that works as the counterpart a follower OS. The master HeterSec OS exports the system resources to the follower OS. Such system resources are often unique for each process, for example, the opened file descriptors, sockets, or event poll descriptors. For safety reasons, the HeterSec has to make sure that only one copy of such resources is maintained across OSes. All the software systems above mentioned are running on the ISA different machines, with fast network connection. In the following of this section, we will break down each component and describe the design details.

3.2 HeterSec address space

HeterSec address space contains all the necessary information to run a process across ISA. For example, the ISA specific instructions and the ISA independent data. It might also carry some additional information, such as the program state for execution relocation. The types of information are decided by each individual security application. For example, the cross-ISA randomized execution would require most of the information embedded as metadata. Because it needs the program state (e.g., the variables on stack) to be synchronized across ISAs during each execution relocation. Security application, such as multi-ISA MVX, requires less information in metadata as it only has to execute the ISA specific instructions. The data and the opened descriptors will be synchronized by the operating system runtime. The HeterSec address space is virtually spread across the ISAs. Depend on the security application, the update on one node would be reflected on the counterpart node immediately, postponed or discarded. The synchronization is performed with the help of the underlying HeterSec distributed kernel.

3.3 HeterSec distributed kernel

The HeterSec distributed kernel could be considered as a special implementation of the multikernel system [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 fast network connection. HeterSec does not maintain the global state for all OSes, instead it maintains some HeterSec process specific states and synchronizes 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 process running on heterogeneous ISA machines: the per process page table handler, 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 would be 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) have to be loaded and pass the security policy to the process runtime. The runtime then decides the process execution behavior, for example, random execution across ISA or concurrent execution with cross-ISA lockstep state checking. In short, based on the runtime security policy, the page table synchronization handler would selectively synchronize the data pages, stack pages, or only several shared pages for lockstep checking. In current design, HeterSec leverages a dedicated kernel thread to handle the synchronization requests. It maintains a simple *read-duplicate write-invalidate* protocol for the shared pages [46].

Another important component for HeterSec is the system resource sharing service. HeterSec maintains a single view of the

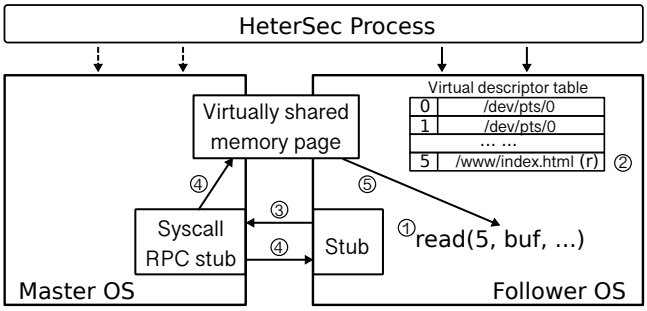

Figure 2: HeterSec virtual descriptor table. This figure shows when the HeterSec process is accessing a remote descriptor on follower OS side.

system resource from HeterSec process's perspective. That means for each HeterSec process, there will be only one set of the network sockets, opened file descriptors, etc. Unfortunately, system resources such as file descriptors, sockets and event descriptors, are by default not allowed to share across the machine boundary. One partial solution could be using a Network File System (NFS) to share and synchronize the file systems across the OS. However, there might be some files that are OS related, e.g., a shared library is often in different ELF format between ISA different OSes. To address this problem, HeterSec combines an implementation of system resource Remote Procedure Call (RPC) and a virtual descriptor table (VDT).

Figure 2 shows the how the system resource RPC works with the virtual descriptor table. Before starting the process, the security application will specify a white list of files that should be loaded locally. During the HeterSec process's running time, the follower OS would build up a virtual descriptor table. For each table entry, it specifies whether the descriptor is in local node, is in remote master node or has to be simulated. The simulation happens when executing multiple versions of code, therefore, some descriptors might only have to be created once. For example, we do not want to create two sockets for a same connection request on HeterSec MVX. For those system resource requests that have to be handled on master OS, a RPC alike mechanism is provided. The RPC stub on master OS handles the RPC request, sets up the buffer value on a virtually shared page, and returns the result back. Note that the virtually shared page is synchronized by the HeterSec kernel, as mentioned above.

3.4 An early prototype

We describe our early implementation of HeterSec prototype. Based on that, we implement two security applications which exploit the ISA heterogeneity.

We implement a prototype of HeterSec on a x86_64 and an ARM64 machine pair, connected using a *Mallanox ConnectX-3* InfiniBand. To enable the HeterSec address space, we leverage an open source popcorn compiler [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 migration/cross-checking points. The state relocation mapping is used at each migration point, which translates the currently running states (e.g., register states, stack slots, etc.) from one ISA to another. The popcorn compiler was built on LLVM, and all the ISA specific code instrumentation was implemented as several backend passes [6]. The synchronized page table handler is implemented in kernel virtual memory subsystem. The updates on HeterSec protected process space would be synchronized across machine boundary, by hooking the *vma* and *pte* operations [16]. Based on the above mentioned prototype, we implement two security applications:

Code randomization with ISA switching: The first security application is a heterogeneous-ISA based code randomization system. Unlike most of the existing runtime code randomization techniques, HeterSec randomizes the code execution with runtime switchable ISA. From the process's perspective, it runs on top of a dynamic hardware environment, with explicit ISA diversity. Therefore, it would be very hard for an attacker to prepare the exploit workload, for example, finding the correct ROP gadget chain, or accurately measuring the side-channel.

When the process executes at 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 randomizing points in existing code re-randomization works [9, 47], except that existing randomization techniques update the code pointer references while HeterSec updates the ISA related states (e.g., stack slots, register set). A per process virtual descriptor table is used to give the process a unique view of the system resources, even if the process is executed on the follower OS. We have supported several compute and memory bound workloads as well as an I/O bound workload like *Nginx* web server.

MVX with ISA diversity: The second security application is a heterogeneous-ISA based multi-version execution system. Similar to a traditional MVX system, the HeterSec MVX also has one leading variant and one (or several) follower variant(s).[2] The leader runs with full access to system resources, while the follower can only conduct computational and memory related operations. Since all the system resource accesses are from the system call interface, most MVX systems do the state comparison at each system call entry and return. HeterSec MVX follows the same principle.

Specifically, the runtime on follower OS will verify whether the resource access should be simulated or passed through. For system calls tagged for passthrough, the follower OS serves the HeterSec process as usual. For those system calls that access the per process resources (e.g., opened file descriptors, sockets, etc.), the HeterSec runtime would simulate those accesses by synchronizing the system call effects from the master OS. HeterSec MVX also implements a ring buffer (located in the virtually shared page) to pass those events in a FIFO queue based manner (e.g., the syscall return values, or the modifications of data structures).

To prevent false positive introduced by the different libraries across machines, the HeterSec compiler compiles the application source code and links the object files with the *musl library* generated from the same source. Therefore, the system call sequences are almost the same among the ISA-different binaries, except for a few thread initialization functions like `set_tid_address(int`

[2]For the sake of simplicity, we use one leader and one follower in this paper description.

`*tidptr`). In this case, we just avoid the comparison for such system call executions. We have successfully supported an unmodified *lighttpd web server* running on top of HeterSec MVX system.

4 DISCUSSION

The approach of diversifying the underlying ISA environment during the process execution is quite challenging. It is not only caused by the lack of such commodity hardware, but also due to very few software system supported to exploit ISA difference in security purpose. HeterSec was proposed and designed under such scenario.

Address space across node: HeterSec implements a shared address space that spreads across the nodes. Such address space synchronization is maintained by the software, especially the kernels. It would be much convenient if the memory could be synchronized across the ISA (or machine node) by hardware. We have seen a branch of the new coming new hardware techniques, such as RDMA over InfiniBand. So it might be possible to utilize those features to speed up some cross-node synchronization or comparison operations. Descriptors and their handlers are another issues that we realized in implementing such system. Currently, we are using a virtual descriptor table to synchronize and simulate the distributed system resource. And it might be easier for HeterSec if the descriptors could be managed in a global and unified form.

Possible attacks: HeterSec hides the underlying architecture environment from attackers by dynamically switching the ISA. However, there might be some possible attack surfaces. For example, the current HeterSec implementation does not support runtime code re-randomization. Currently, HeterSec leverages the ISA switching to disturb some attack presumptions, such as the known gadget chain or timing information. While it is possible to re-randomize the code by using dynamic binary instrumentation (DBI) or page table based re-randomization during the program state relocation.

5 CONCLUSION

We reported the design and implementation of our on-going work of HeterSec, a framework to improve application security with ISA heterogeneity. HeterSec provides an environment to enable HeterSec process to select its underlying ISA, thus it adds an additional layer of dynamic and software diversity. HeterSec was built with several compiler and kernel extensions to facilitate process 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.

ACKNOWLEDGMENTS

We would like to thank the anonymous reviewers for their insightful comments. This work is supported in part by ONR under grant N00014-18-1-2022.

REFERENCES

[1] Martín Abadi, Mihai Budiu, Úlfar Erlingsson, and Jay Ligatti. 2009. Control-Flow Integrity Principles, Implementations, and Applications. *ACM Transactions on Information and System Security (TISSEC)* 13, 1 (2009), 4.

[2] One Aleph. 1996. Smashing the stack for fun and profit. *http://www.shmoo.com/phrack/Phrack49/p49-14* (1996).

[3] ARM 2018. *The ARMv8-A Architecture Reference Manual.* ARM.

[4] ARM Limited (or its affiliates). Accessed: 2019-01-14. ARM BIG.LITTLE. https://www.arm.com/why-arm/technologies/big-little.

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

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

[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. 2009. The multikernel: a new OS architecture for scalable multicore systems. In *Proceedings of the ACM SIGOPS 22nd symposium on Operating systems principles*. ACM, 29–44.

[8] Bhat, Sharath K and Saya, Ajithchandra and Rawat, Hemedra K and Barbalace, Antonio and Ravindran, Binoy. 2016. Harnessing energy efficiency of heterogeneous-ISA platforms. *ACM SIGOPS Operating Systems Review* 49, 2 (2016), 65–69.

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

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

[11] Tianqi Chen, Mu Li, Yutian Li, Min Lin, Naiyan Wang, Minjie Wang, Tianjun Xiao, Bing Xu, Chiyuan Zhang, and Zheng Zhang. 2015. Mxnet: A flexible and efficient machine learning library for heterogeneous distributed systems. *arXiv preprint arXiv:1512.01274* (2015).

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

[13] Benjamin Cox, David Evans, Adrian Filipi, Jonathan Rowanhill, Wei Hu, Jack Davidson, John Knight, Anh Nguyen-Tuong, and Jason Hiser. 2006. N-Variant Systems: A Secretless Framework for Security through Diversity.. In *USENIX Security Symposium*. 105–120.

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

[15] Stephen J Crane, Stijn Volckaert, Felix Schuster, Christopher Liebchen, Per Larsen, Lucas Davi, Ahmad-Reza Sadeghi, Thorsten Holz, Bjorn De Sutter, and Michael Franz. 2015. It's a TRaP: Table randomization and protection against function-reuse attacks. In *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*. ACM, 243–255.

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

[17] John Demme, Robert Martin, Adam Waksman, and Simha Sethumadhavan. 2012. Side-channel vulnerability factor: A metric for measuring information leakage. In *Computer Architecture (ISCA), 2012 39th Annual International Symposium on*. IEEE, 106–117.

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

[19] Intel 2018. *Intel 64 and IA-32 Architectures Software Developerś Manual.* Intel.

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

[21] Hyeran Jeon, Yinglong Xia, and Viktor K Prasanna. 2010. Parallel exact inference on a CPU-GPGPU heterogenous system. In *Parallel Processing (ICPP), 2010 39th International Conference on*. IEEE, 61–70.

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

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

[24] 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. 2019. Spectre Attacks: Exploiting Speculative Execution. In *40th IEEE Symposium on Security and Privacy (S&P'19)*.

[25] Koen Koning, Herbert Bos, and Cristiano Giuffrida. 2016. 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)*. IEEE, 431–442.

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

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

[28] Mittal, Sparsh and Vetter, Jeffrey S. 2015. A survey of CPU-GPU heterogeneous computing techniques. *ACM Computing Surveys (CSUR)* 47, 4 (2015), 69.

[29] Pierre Olivier, Antonio Barbalace, and Binoy Ravindran. 2016. Multi-Variant Execution atop a Decomposed Hypervisor on Emerging Heterogeneous-ISA Multicore. EuroSys'16 (Poster).

[30] Ryan Roemer, Erik Buchanan, Hovav Shacham, and Stefan Savage. 2012. Return-oriented Programming: Systems, Languages, and Applications. *ACM Transactions on Information and System Security (TISSEC)* 15, 1 (2012), 2.

[31] Babak Salamat, Todd Jackson, Andreas Gal, and Michael Franz. 2009. 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*. ACM, 33–46.

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

[33] Mark Seaborn and Thomas Dullien. 2015. Exploiting the DRAM rowhammer bug to gain kernel privileges. *Black Hat* 15 (2015).

[34] Hovav Shacham. 2007. The Geometry of Innocent Flesh on the Bone: Return-Into-Libc without Function Calls (on the x86). In *Proceedings of the 14th ACM Conference on Computer and Communications Security*.

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

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

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

[38] Victor van der Veen, Dennis Andriesse, Manolis Stamatogiannakis, Xi Chen, Herbert Bos, and Cristiano Giuffrdia. 2017. The dynamics of innocent flesh on the bone: Code reuse ten years later. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*. ACM, 1675–1689.

[39] Ashish Venkat, Harsha Basavaraj, and Dean Tullsen. 2019. Composite-ISA Cores: Enabling Multi-ISA Heterogeneity Using a Single ISA. In *25th IEEE International Symposium on High Performance Computer Architecture*. IEEE.

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

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

[42] Alexios Voulimeneas, Dokyung Song, Fabian Parzefall, Yeoul Na, Per Larsen, Michael Franz, and Stijn Volckaert. 2019. DMON: A Distributed Heterogeneous N-Variant System. In Submission.

[43] Xiaoguang Wang and Yong Qi. 2017. Secure the commodity applications against address exposure attacks. In *Computers and Communications (ISCC), 2017 IEEE Symposium on*. IEEE, 450–456.

[44] Richard Wartell, Vishwath Mohan, Kevin W. Hamlen, and Zhiqiang Lin. 2012. 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)*.

[45] Wikipedia. Accessed: 2019-01-14. Data Execution Prevention. http://en.wikipedia.org/wiki/Data_Execution_Prevention.

[46] Wikipedia. Accessed: 2019-02-14. MSI Protocol. https://en.wikipedia.org/wiki/MSI_protocol.

[47] 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. 2016. Shuffler: Fast and Deployable Continuous Code Re-Randomization.. In *OSDI*. 367–382.

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

[49] Mingwei Zhang and R. Sekar. 2013. Control Flow Integrity for COTS Binaries. In *Proceedings of the 22Nd USENIX Conference on Security (SEC'13)*.

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