Transparent, Cross-ISA Enclave Offloading

Xiaoguang Wang
Virginia Tech
Blacksburg, USA

Carlos Bilbao
Virginia Tech
Blacksburg, USA

Binoy Ravindran
Virginia Tech
Blacksburg, USA

ABSTRACT

Hardware vendors constantly design new architecture extensions to improve software security. Due to intellectual property protection and architecture design issues, these hardware extensions are often CPU-specific. This creates an availability gap between different CPU types (security extensions) and software code that runs on top. This is particularly an issue for low-end, embedded devices as they often utilize wimpy CPU cores. This paper aims to bridge this availability gap for security-related CPU extensions, focusing on low-end embedded devices' access to hardware trusted execution environments (TEEs). We present PopSGX, a framework to transparently offload security-sensitive workloads to a remote enclave of a centralized edge server. The low-end embedded devices act as clients, taking advantage of hardware security features provided by the edge server. We have built an early prototype of PopSGX on top of an open-sourced hardware-agnostic Open Enclave SDK [21]. Any application written using the Open Enclave SDK can leverage the framework without modifying source code. We evaluate our prototype on a simulated IoT/edge computing environment (a Raspberry Pi and an SGX-enabled laptop). The result shows that PopSGX has reasonable performance overhead but secures confidential computations running on the Raspberry Pi.

CCS CONCEPTS

• **Security and privacy → Software and application security**; **Systems security**.

KEYWORDS

Memory Protection, Software Security, Enclave Offloading

ACM Reference Format:
Xiaoguang Wang, Carlos Bilbao, and Binoy Ravindran. 2022. Transparent, Cross-ISA Enclave Offloading. In *Proceedings of Proceedings of the 5th Workshop on System Software for Trusted Execution (SysTEX '22 Workshop).* ACM, New York, NY, USA, 6 pages.

1 INTRODUCTION

The hybrid-CPU architecture environment has gained more attention due to the growing popularity of mobile computing, the internet of things (IoT), machine learning, and data centers. Many new computing patterns have been developed. For example, an edge server can be a hub for smart home devices to process information [28]. An ARM-based smart NIC can process distributed transactions [24] or run network server applications [26]. The workloads on these new computing environments are often split across the hybrid-CPU architecture boundary. As a result, very few existing software security models can apply to such workloads.

Some barriers hinder securing a program on a hybrid-CPU architecture. For example, several existing software protection techniques require architecture-specific hardware features [3, 12, 35], but lacks an effective way to utilize them to protect software running on a hybrid-CPU architecture. Furthermore, code splitting and data movement are difficult for a hybrid-CPU architecture. This is partly because many heterogeneous architecture nodes are cache-incoherent; moving sensitive data closer to a CPU core with a particular security extension requires code instrumentation and data synchronization. Thus, few works have considered software security enhancement for executing code across heterogeneous cores.

One way to solve this is to build a universal API for different CPU types and security extensions [8, 18, 21]. *Open Enclave* is such a project aiming to build a universal programming interface by abstracting different enclave types of various architectures [21]. Although the goal is ambitious, supporting multiple enclave types with a single API seems hard as, e.g., *Open Enclave* developers recently confirmed that ARM TEE support is still under development [7]. Other research efforts, such as FlexOS [18], build an abstraction for each memory isolation extension so that users can switch between different protection primitives at deployment time. However, FlexOS's approach only works on a single machine node (one architecture) with different isolation primitives. It cannot be easily applied to a smart NIC or an IoT/edge environment of varying CPU types.

This paper presents our early work on bridging the availability gap of security-related CPU extensions in a hybrid-architecture environment. We use the hardware enclave as an example and present the design and implementation of PopSGX, a framework to securely offload security-sensitive computations from low-end CPU cores to remote CPU cores with hardware enclave support. PopSGX mainly consists of two parts: an extended version of the Open Enclave SDK and a user-space PopSGX code monitor. Applications written using the Open Enclave SDK can directly run on low-end CPU cores and transparently offload their confidential computations to a remote node with SGX support.

The PopSGX monitor is implemented as a user-space program and runs in a separate address space from the target process. The PopSGX monitor transparently maintains a synchronized shared memory region between cache in-coherent computing nodes. A page update on one node will be transparently synchronized with the other node. The PopSGX monitor leverages a recent kernel feature named `userfaultfd` [23] to delegate page faults to the user-space monitor; thus, PopSGX does not require any kernel modifications. We have supported several applications from SPEC CPU 2017, a web server, and sample programs from the Open Enclave SDK. We ran the experiments on a simulated IoT/edge computing environment (a Raspberry Pi and an SGX-enabled laptop) using PopSGX.

SysTEX '22 Workshop, @ASPLOS, Lausanne, Switzerland
2022.

In summary, we make the following contributions:

- We explore the research space and design requirement of utilizing hardware security extensions on cache-incoherent, heterogeneous architectures to secure program code and data;
- We use the hardware enclave as an example to demonstrate the feasibility of such a vision and present PopSGX for securely offloading confidential computations to a remote enclave across different architectures;
- We extend a popular hardware-agnostic open-source Open Enclave SDK to ease the integration of PopSGX with security-sensitive applications;
- We evaluate PopSGX on a Raspberry Pi and an SGX-enabled laptop using SPEC CPU 2017 and a web server; the results show that PopSGX can protect encryption keys on low-end embedded devices from Heartbleed-alike attacks (CVE-2014-0160) with a reasonable performance overhead.

The rest of the paper is organized as follows. Section 2 describes the background and motivation. We describe the design and implementation of PopSGX in Section 3. In Section 4, a number of experiments are carried out to analyze security and performance aspects. We summarize related work and discuss open questions and future work in Section 5. Finally, we conclude in Section 6.

2 BACKGROUND AND MOTIVATION

The Trusted Execution Environment (TEE) is a hardware sandbox to safeguard an application's security-sensitive code and data [34]. In the TEE model, application code and data is split into a trusted part (enclave) and an untrusted part (host). Functions that handle confidential data are written as enclave code. At runtime, the hardware TEE (e.g., Intel SGX) enforces memory and computation safety for the trusted component. Should the host need to access enclave data, it will have to require explicit authorization from the enclave, which exposes a list of functions that can be called from the outside, known as `ecall`s. Similarly, the host side of the application includes a list of functions that can be called from the enclave, referred to as `ocall`s. The `ecall` and `ocall` interfaces are defined by an enclave definition language (EDL) file. Both Intel SGX SDK and the Open Enclave SDK provide tools (e.g., `oeedger8r`) to convert a `.edl` file into interface files for enclaves and host code to communicate.

Although the general idea of TEE has gained traction, different hardware vendors have their own hardware implementation of enclaves (e.g., Intel SGX [14], ARM TrustZone [2], AMD SEV [1], RISC-V Keystone [17]). The different API specifications increases the challenge in developing trusted applications. This is the primary motivation behind projects such as the Open Enclave project [21] and the Asylo project [8], which aim to make enclave deployment flexible. For example, Open Enclave is an open-source and hardware agnostic API interface (SDK) for enclave development. Open Enclave was initially designed to support Intel SGX and ARM TrustZone as the enclave targets. However, due to the different hardware interfaces, at the time of writing this paper, the ARM TrustZone is not fully supported yet [7]. Besides the various hardware interfaces, hardware enclaves are not ubiquitous. There is a lack of enclave support for low-end embedded devices. Furthermore, Intel plans only to support SGX on server CPUs while deprecating SGX support on its 11th and 12th-gen desktop CPUs [27]. Thus, there is a need to support existing confidential computation applications and their use scenarios on devices without hardware TEE support.

DSM and userfaultfd: Distributed shared memory (DSM) is a form of memory architecture where physically separated memories can be addressed as a single shared address space [22]. DSM can be implemented in either hardware or software. Examples of hardware-based DSM include the cache-coherent hardware for uniform memory access (cc-NUMA) and the network interface controller card. Software-based DSM can be implemented in different ways. For example, a modified kernel handles page fault caused by memory modifications on remote nodes, following an invalidation-based MSI protocol [16]. We argue that existing designs are too heavy-weight and cannot be directly applied for low-end CPU cores. In our design, we leverage the existing Linux kernel interfaces, `userfaultfd` and `ptrace`, for lightweight shared memory between nodes.

Userfaultfd is a Linux kernel facility that allows code to handle page faults in userspace. It allows an application to register regions of memory with a file descriptor. When a page fault happens (e.g., when some memory has not been loaded into memory), the application is notified of it and is able to serve the fault. Once a `userfaultfd` file descriptor is opened, it can also be passed to a manager process using UNIX domain sockets so that the same manager process can handle the page faults of a multitude of different processes [15].

3 SYSTEM DESIGN

PopSGX aims to transparently offload confidential computations to a computing node with SGX support. To achieve this, PopSGX needs to transparently split the code and data into different nodes and later synchronize the results back. Figure 1 depicts an overview of PopSGX running on cache-incoherent heterogeneous cores. PopSGX is mainly composed of two components: a PopSGX monitor for execution monitoring and synchronization, and an extended Open Enclave SDK for `enclave`/`host` code split across heterogeneous-ISA nodes.

A security-sensitive application running on low-end CPU cores behaves like a client, while the actual enclave code runs on SGX-enabled cores. A PopSGX monitor intercepts the `ecall`/`ocall` execution and redirects the control flow across the nodes. The PopSGX monitor handles the page faults from the target process's specific memory regions and maintains a distributed shared memory region to synchronize the data. Thus, the target application is transparently aware of any memory updates caused by the remote enclave execution. The PopSGX monitor is designed as a userspace program that runs on commodity software/hardware stacks to ease deployment.

Cross-architecture enclave code generation. The Open Enclave SDK can only generate the enclave/host code for *x86_64* architectures at the time of writing this paper. However, application code must be compiled and split on different architectures. To solve this issue, we extended the Open Enclave SDK and automatically generated the cross-architecture host/enclave code.

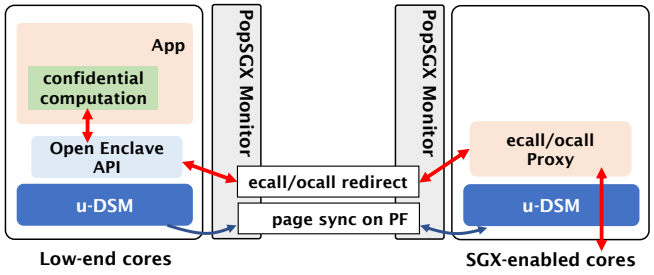

Figure 1: Overview of PopSGX design.

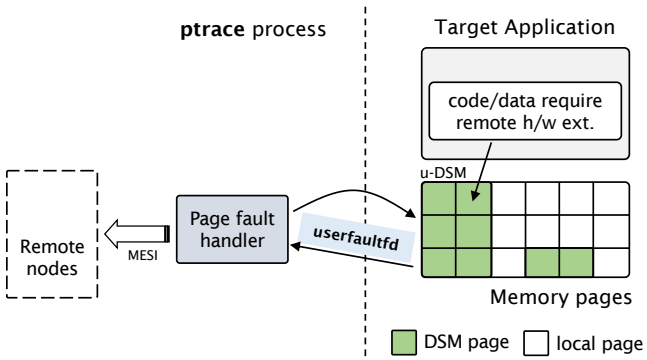

Figure 2: PopSGX monitor for userspace DSM support.

SGX SDKs allow users to define the enclave interfaces and provide tools to transform the enclave interfaces into auxiliary interface files (i.e., `*_u.c, *_t.c, *_args.h` when using the Open Enclave SDK) [21]. These auxiliary files contain the trampoline code to the enclave functions, the function table for enclaves to call the host (including system calls), data structures of operating system resources, and code for parameter marshaling. PopSGX extends this SDK framework.

As mentioned earlier, the enclave/host code generation mainly consists of two steps: generating the auxiliary interface files from the *.edl* file, and host/enclave code compilation and binary generation. PopSGX modifies this process and instruments the code for running on heterogeneous architectures. Specifically, PopSGX provides a helper library for host code generation. The helper library replaces the ecall functions with a dispatcher function. On each ecall to the enclave, the dispatcher function looks up the calling ID, marshals the function parameters, and forwards the ecall request to the PopSGX monitor. This is achieved by raising a user-defined signal, which is captured by the PopSGX monitor. Note that `ecall` replacement is realized by modifying the `oeedger8r` tool that generates the SGX auxiliary interface files. This process is transparent to the application source code.

PopSGX does not modify enclave code generation. Instead, it creates a proxy on the server node that loads the enclave into memory. The proxy also handles ecall requests from a remote node and forwards ocalls back. Although the overall software stack is complex, PopSGX does not burden code development and deployment. This is achieved by the lightweight userspace PopSGX monitor.

PopSGX monitor. The PopSGX monitor runs as a separate process from the target application but maintains a distributed shared memory for the target program. PopSGX uses a kernel interface, `ptrace` [33], to monitor code execution and synchronize the memory content of the target. `ptrace` is widely used by debuggers to inspect and control the execution of *tracees*. `ptrace` allows a tracer (i.e., the PopSGX monitor) to intercept signals from the tracee, read tracee's memory contents, and modify its program states (e.g., register values) [33].

Figure 2 demonstrates PopSGX monitor's design. Users launch the target program as a child process of PopSGX monitor. Next, the PopSGX monitor reads the process information from the `/proc` filesystem and then enters an event-loop. The PopSGX monitor receives a signal when the host program issues an ecall. For the `ecalls` that only contain values, the PopSGX monitor directly forwards the `ecall` request to the remote server. For `ecalls` that contain both value and memory pointer as arguments, the `ecall` forwarding becomes tricky because the other node does not have access to the same virtual memory regions. Consequently, PopSGX needs to synchronize the memory pages used for the `ecall/ocall` transaction on both sides of the connection.

To solve this issue, the PopSGX monitor creates a thread for handling page faults caused by remote enclave execution. The PopSGX monitor registers the memory region used by the ecall and then marks it as read-only. Whenever one node modifies the pages, the page fault handler captures the event and broadcasts the modification to the other node. The counterpart node invalidates the pages. The PopSGX monitor uses this invalidation-based approach to transparently synchronize the memory contents of the target application. Currently, we have implemented a simpler version of a userspace DSM (u-DSM in Figure 2) that copies the pages directly for `ecall/ocalls` without considering the read/write details to memory.

Note that the `userfaultfd` file descriptor can only be created within the target process's address space. To solve this issue, currently, we leverage the instrumented auxiliary interface files generated by our modified `oeedger8r` tool to create the `userfaultfd` descriptor and then use a Unix Domain Socket (UDS) to pass the file descriptor to the PopSGX monitor process. Future improvements could use the *compel library* from the CRIU project to implant parasite code into applications directly from the PopSGX monitor's address space [5, 20]. In this way, `userfaultfd` descriptor creation will be fully transparent to the target process.

4 PRELIMINARY EVALUATION

We now present our preliminary evaluation of PopSGX from security and performance aspects. We simulated an IoT/edge computing scenario using a Raspberry Pi 4 Model B as the client and a Lenovo ThinkPad as the server, both running Ubuntu 20.04 as the operating system. Table 1 shows the experimental setup.

We first examined the CVE list regarding the sensitive data leakage on embedded and IoT devices, and found at least 7 CVEs in the past five years [6]. Next, we reproduced the heartbleed attack on a Raspberry Pi and used PopSGX to prevent it. Heartbleed is a

Table 1: Experimental setup

Description	Lenovo ThinkPad	Raspberry Pi 4
CPU	Core i9-9880H	Cortex-A72 (ARMv7/v8)
Cores	8 (16 HT)	4
Clock (GHz)	2.3	1.5
RAM	16 GB	8 GB
Intel SGX	Yes	No
Interconnect	Wireless network (IPv4/IPv6)	

vulnerability in OpenSSL 1.0.1 through 1.0.1f (CVE-2014-0160). In particular, it is a buffer overflow that allows an attacker to overread the server's memory. The flaw is triggered when a client tells the server to send a message of a certain amount of bytes, but then sends a shorter message. An unpatched OpenSSL library will not check the length of the data but reads the data required by the client from memory and sends it back. We replicated the attack with a vulnerable version of the OpenSSL (version 1.0.1) on an SSL server running on the Raspberry Pi. Lastly, we repeated the experiment and protected the SSL keys using PopSGX. In our experiment, we offload the SSL key's access/store to the SGX node, preventing Heartbleed.

Process isolation mechanisms guarantee security of the PopSGX monitor. The PopSGX monitor runs in a separate address space from the target program so that an attacker cannot directly access the monitor's code to hijack the monitor. The target process is unaware of the underlying memory page synchronization mechanisms. When it passes the parameters to a remote enclave through ecalls, the pages are automatically shared between two nodes. This is achieved by the `userfaultfd` running inside the monitor process that tracks the page faults. An attacker may potentially raise a malicious signal as a fake ecall. In that case, the PopSGX monitor can maintain an allowlist of legal ecall locations and raise an alert when receiving a `SIGUSR` other than from these locations.

To understand the performance impact of PopSGX, we ported two sample programs from the Open Enclave SDK (*hello world* and *file encryptor*), one open-sourced computer vision application (openFABMAP [13]), and six C/C++ applications from the SPEC CPU 2017 benchmark suite using PopSGX. There are about 8% performance overhead for *hello world*, *file encryptor*, and openFABMAP applications. This is likely because we only insert a single `ecall/ocall` pair during whole program execution. The single `ecall/ocall` execution does not significantly affect the overall performance. We also evaluated the execution time of running each SPEC CPU application natively on the Raspberry Pi, and the time for offloading five `ecall/ocalls`, and 50 `ecall/ocalls` to a remote SGX machine node, respectively. Figure 3 shows the normalized results. We found that more `ecall/ocalls` incur larger performance overhead. In general, applications with longer execution time have relatively less performance overhead. `Ecalls` with more parameters or a larger memory footprint will have more considerable performance overhead (possibly because of more memory pages to be synchronized). However, from our previous research effort on DSM, the overall performance can be significantly improved if the network connection is faster [16]. We expect that deploying PopSGX on closely-coupled devices (e.g., a SmartNIC) will have better performance.

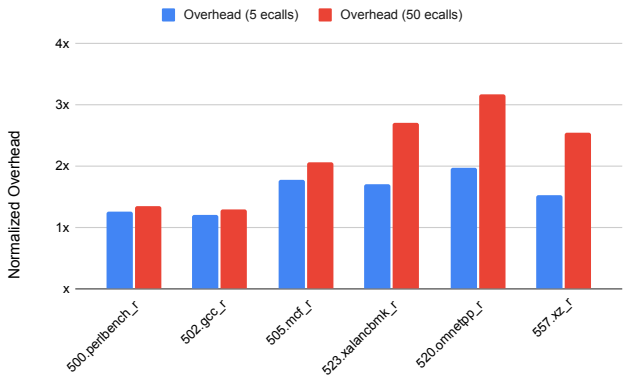

Figure 3: Normalized performance overhead running SPEC CPU 2017 on PopSGX.

5 RELATED WORK AND DISCUSSION

Several existing works aim to enable applications running on CPUs of heterogeneous-ISAs by dynamic binary translation (DBT) [10], code offloading [30], and code migration [4, 9]. In particular, dynamic binary translation takes short code sequences (i.e., basic blocks) and translates sequences of instructions from a source instruction set to the target instruction set [10]. In [30], Wang et al. propose cross-ISA binary offloading, which leverages a DBT engine to offload code to a server to reduce energy consumption of mobile devices. Popcorn Linux bridges the programmability gap for applications running on heterogeneous-ISA CPUs using a compiler that generates "migratable" binaries [4]. A kernel-space DSM is also implemented to allow the migrated code to synchronize stack, heap, or other global data [4, 16]. HeterSec is another related work that diversifies the program state using heterogeneous CPUs [31]. However, HeterSec cannot handle architecture-specific code. Unlike these works, PopSGX is designed to offload confidential computations in heterogeneous CPUs. PopSGX is also implemented as a lightweight approach, thus applications written using the Open Enclave SDK can be directly ported without any modification.

The second category of related work focuses on improving the usability of hardware enclaves. As mentioned earlier, Open Enclave [21] and Asylo [8] are two open-source projects aiming to build portable enclave applications. Asylo provides application developers with choices of security backends (e.g., TEEs or VMs). Asylo also uses a secure enclave gRPC channel to transit data [8]. In contrast, PopSGX implements a userspace DSM layer, which enables transparent offloading of enclaves. Rust-SGX [29] and EGo [25] are two projects that extend the SGX programming interface to other high-level memory-safe languages. Rust-SGX creates a secure binding for Rust applications to utilize Intel SGX APIs and libraries written in C/C++, and formally verifies the memory model [29]. PopSGX, on the other hand, transparently binds low-end devices with remote hardware enclaves. Although currently PopSGX only supports C/C++ applications, we believe that porting SGX SDK and the monitor to Rust and Go is feasible.

SGXJail is another related work that isolates the enclave within a sandbox process [32]. SGXJail also forwards ecall/ocall requests through a dispatcher; thus, SGXJail can prevent a malicious enclave from compromising the host code. In contrast, PopSGX has a different goal: to bridge the availability gap of hardware security extensions from different architectures.

There are several other open questions on borrowing a remote hardware extension for local confidential computations. For example, *what hardware extensions can be remotely shared?* Our work explores how a shared memory system can be utilized for offloading confidential computations. Besides hardware enclaves, there exists a number of other potentially "offloadable" hardware security extensions, including hardware-based control flow enhancement [11], pointer authentication, and memory tagging [19]. Determining the offloadable security workload is an interesting future direction.

6 CONCLUSION

We presented the design and implementation of PopSGX, a system to offload confidential computations from low-end embedded devices to a remote centralized edge server. PopSGX consists of two parts: an extended Open Enclave SGX SDK for cross-architecture enclave code compilation and generation, and a PopSGX monitor for transparent ecall/ocall interception and memory synchronization. The user-space PopSGX monitor is implemented using `ptrace` and `userfaultfd` mechanisms. We have built a prototype of PopSGX and evaluated the prototype using applications from the SPEC CPU 2017, two sample programs from the Open Enclave SDK, and a server application simulating the Heartbleed attack. The evaluation results show that PopSGX secures confidential computations for IoT devices with an acceptable performance overhead.

Acknowledgments

We thank the anonymous reviewers for their insightful comments. This work is supported in part by US Office of Naval Research (ONR) under grants N00014-18-1-2022 and N00014-19-1-2493, and National Science Foundation (NSF) under grant CNS 2127491. Any opinions, findings, and conclusions expressed in this material are those of the authors and do not necessarily reflect the views of these agencies.

REFERENCES

[1] Advanced Micro Devices, Inc. 2021. AMD Secure Encrypted Virtualization (SEV). https://developer.amd.com/sev/.
[2] Arm Limited. 2021. Arm TrustZone Technology. https://developer.arm.com/ip-products/security-ip/trustzone.
[3] Sergei Arnautov, Bohdan Trach, Franz Gregor, Thomas Knauth, Andre Martin, Christian Priebe, Joshua Lind, Divya Muthukumaran, Dan O'Keeffe, Mark L. Stillwell, David Goltzsche, Dave Eyers, Rüdiger Kapitza, Peter Pietzuch, and Christof Fetzer. 2016. SCONE: Secure Linux Containers with Intel SGX. In *12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16)*. USENIX Association, Savannah, GA, 689–703. https://www.usenix.org/conference/osdi16/technical-sessions/presentation/arnautov
[4] 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.
[5] CRIU Dev. 2021. Compel. https://criu.org/Compel.
[6] CVE Database. [n. d.]. Common Vulnerabilities and Exposures Database. http://www.cvedetails.com/.
[7] OpenEnclave Dev. Accessed: 2022-01-12. Open Enclave SDK. https://github.com/openenclave/openenclave/issues/4255.
[8] Asylo Developers. 2021. Asylo: An open and flexible framework for enclave applications. https://asylo.dev/.
[9] Matthew DeVuyst, Ashish Venkat, and Dean M. Tullsen. 2012. Execution migration in a heterogeneous-ISA chip multiprocessor. In *Proceedings of the 17th International Conference on Architectural Support for Programming Languages and Operating Systems, ASPLOS 2012, London, UK, March 3-7, 2012*, Tim Harris and Michael L. Scott (Eds.). ACM, 261–272. https://doi.org/10.1145/2150976.2151004
[10] Kemal Ebcioglu, Erik R. Altman, Michael Gschwind, and Sumedh W. Sathaye. 2001. Dynamic Binary Translation and Optimization. *IEEE Trans. Computers* 50, 6 (2001), 529–548. https://doi.org/10.1109/12.931892
[11] Tom Garrison. 2022. Intel CET Answers Call to Protect Against Common Malware Threats. *https://newsroom.intel.com/editorials/intel-cet-answers-call-protect-common-malware-threats/* (2022).
[12] Xinyang Ge, Weidong Cui, and Trent Jaeger. 2017. GRIFFIN: Guarding Control Flows Using Intel Processor Trace. In *Proceedings of the Twenty-Second International Conference on Architectural Support for Programming Languages and Operating Systems, ASPLOS 2017, Xi'an, China, April 8-12, 2017*, Yunji Chen, Olivier Temam, and John Carter (Eds.). ACM, 585–598. https://doi.org/10.1145/3037697.3037716
[13] Arren Glover. Accessed: 2022-01-12. openFABMAP: Open Source C++ Code for the FAB-MAP Algorithm. *https://github.com/arrenglover/openfabmap* (Accessed: 2022-01-12).
[14] Intel 2013. *Software Guard Extensions Programming Reference*. Intel.
[15] The kernel development community. 2021. The Linux kernel user's and administrator's guide – Userfaultfd. https://www.kernel.org/doc/html/latest/admin-guide/mm/userfaultfd.html.
[16] Sang-Hoon Kim, Ho-Ren Chuang, Robert Lyerly, Pierre Olivier, Changwoo Min, and Binoy Ravindran. 2020. DeX: Scaling Applications Beyond Machine Boundaries. In *40th IEEE International Conference on Distributed Computing Systems, ICDCS 2020, Singapore, November 29 - December 1, 2020*. IEEE, 864–876. https://doi.org/10.1109/ICDCS47774.2020.00021
[17] Dayeol Lee, David Kohlbrenner, Shweta Shinde, Krste Asanovic, and Dawn Song. 2020. Keystone: an open framework for architecting trusted execution environments. In *EuroSys '20: Fifteenth EuroSys Conference 2020, Heraklion, Greece, April 27-30, 2020*, Angelos Bilas, Kostas Magoutis, Evangelos P. Markatos, Dejan Kostic, and Margo I. Seltzer (Eds.). ACM, 38:1–38:16. https://doi.org/10.1145/3342195.3387532
[18] Hugo Lefeuvre, Vlad-Andrei Badoiu, Alexander Jung, Stefan Teodorescu, Sebastian Rauch, Felipe Huici, Costin Raiciu, and Pierre Olivier. 2022. FlexOS: Towards Flexible OS Isolation. *ASPLOS* (2022). arXiv:2112.06566 https://arxiv.org/abs/2112.06566
[19] ARM Limited. 2022. Armv8.5-A Memory Tagging Extension – White Paper. (2022).
[20] Robert Lyerly, Xiaoguang Wang, and Binoy Ravindran. 2020. Dynamic and Secure Memory Transformation in Userspace. In *Computer Security - ESORICS 2020 - 25th European Symposium on Research in Computer Security, ESORICS 2020, Guildford, UK, September 14-18, 2020, Proceedings, Part I (Lecture Notes in Computer Science, Vol. 12308)*, Liqun Chen, Ninghui Li, Kaitai Liang, and Steve A. Schneider (Eds.). Springer, 237–256. https://doi.org/10.1007/978-3-030-58951-6_12
[21] Microsoft. Accessed: 2022-01-12. Open Enclave SDK. https://openenclave.io/sdk/.
[22] Bill Nitzberg and Virginia Mary Lo. 1991. Distributed Shared Memory: A Survey of Issues and Algorithms. *Computer* 24, 8 (1991), 52–60. https://doi.org/10.1109/2.84877
[23] Mike Rapoport. 2021. userfaultfd(2) — Linux manual page. https://man7.org/linux/man-pages/man2/userfaultfd.2.html.
[24] Henry N. Schuh, Weihao Liang, Ming Liu, Jacob Nelson, and Arvind Krishnamurthy. 2021. Xenic: SmartNIC-Accelerated Distributed Transactions. In *SOSP '21: ACM SIGOPS 28th Symposium on Operating Systems Principles, Virtual Event / Koblenz, Germany, October 26-29, 2021*, Robbert van Renesse and Nickolai Zeldovich (Eds.). ACM, 740–755. https://doi.org/10.1145/3477132.3483555
[25] Edgeless Systems. 2022. Build Confidential Go Apps with Ease. *https://www.ego.dev/* (2022).
[26] Maroun Tork, Lina Maudlej, and Mark Silberstein. 2020. Lynx: A SmartNIC-driven Accelerator-centric Architecture for Network Servers. In *ASPLOS '20: Architectural Support for Programming Languages and Operating Systems, Lausanne, Switzerland, March 16-20, 2020*, James R. Larus, Luis Ceze, and Karin Strauss (Eds.). ACM, 117–131. https://doi.org/10.1145/3373376.3378528
[27] Bill Toulas. 2022. New Intel chips won't play Blu-ray disks due to SGX deprecation. https://www.bleepingcomputer.com/news/security/new-intel-chips-wont-play-blu-ray-disks-due-to-sgx-deprecation/
[28] Rahmadi Trimananda, Ali Younis, Bojun Wang, Bin Xu, Brian Demsky, and Guoqing Harry Xu. 2018. Vigilia: Securing Smart Home Edge Computing. In *2018 IEEE/ACM Symposium on Edge Computing, SEC 2018, Seattle, WA, USA, October 25-27, 2018*. IEEE, 74–89. https://doi.org/10.1109/SEC.2018.00013
[29] Huibo Wang, Pei Wang, Yu Ding, Mingshen Sun, Yiming Jing, Ran Duan, Long Li, Yulong Zhang, Tao Wei, and Zhiqiang Lin. 2019. Towards Memory Safe Enclave Programming with Rust-SGX. In *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security, CCS 2019, London, UK, November*

11-15, 2019, Lorenzo Cavallaro, Johannes Kinder, XiaoFeng Wang, and Jonathan Katz (Eds.). ACM, 2333–2350. https://doi.org/10.1145/3319535.3354241

[30] Wenwen Wang, Pen-Chung Yew, Antonia Zhai, Stephen McCamant, Youfeng Wu, and Jayaram Bobba. 2017. Enabling Cross-ISA Offloading for COTS Binaries. In *Proceedings of the 15th Annual International Conference on Mobile Systems, Applications, and Services, MobiSys'17, Niagara Falls, NY, USA, June 19-23, 2017*, Tanzeem Choudhury, Steven Y. Ko, Andrew Campbell, and Deepak Ganesan (Eds.). ACM, 319–331. https://doi.org/10.1145/3081333.3081337

[31] Xiaoguang Wang, SengMing Yeoh, Robert Lyerly, Pierre Olivier, Sang-Hoon Kim, and Binoy Ravindran. 2020. A Framework for Software Diversification with ISA Heterogeneity. In *23rd International Symposium on Research in Attacks, Intrusions and Defenses, RAID 2020, San Sebastian, Spain, October 14-15, 2020*, Manuel Egele and Leyla Bilge (Eds.). USENIX Association, 427–442. https://www.usenix.org/conference/raid2020/presentation/wang-xiaoguang

[32] Samuel Weiser, Luca Mayr, Michael Schwarz, and Daniel Gruss. 2019. SGXJail: Defeating Enclave Malware via Confinement. In *22nd International Symposium on Research in Attacks, Intrusions and Defenses, RAID 2019, Chaoyang District, Beijing, China, September 23-25, 2019*. USENIX Association, 353–366. https://www.usenix.org/conference/raid2019/presentation/weiser

[33] Wikipedia. 2021. Ptrace. http://en.wikipedia.org/wiki/Ptrace.

[34] Wikipedia. 2022. Trusted execution environment. *https://en.wikipedia.org/wiki/Trusted_execution_environment* (2022).

[35] Yajin Zhou, Xiaoguang Wang, Yue Chen, and Zhi Wang. 2014. ARMlock: Hardware-Based Fault Isolation for ARM. In *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security* (Scottsdale, Arizona, USA) *(CCS '14)*. Association for Computing Machinery, New York, NY, USA, 558–569.