# UniGuard: Protecting Unikernels using Intel SGX

Ioannis Sfyrakis and Thomas Groß
School of Computing Science, Newcastle University, UK
{ioannis.sfyrakis, thomas.gross}@newcastle.ac.uk

***Abstract*—Computations executed in lightweight virtual machines called unikernels have a minimal attack surface and improved performance. However, unikernels are still prone to leaking information to the operating system or to the hypervisor that hosts them. This is attributed to vulnerabilities in privileged software and to malicious insiders operating in cloud infrastructures. Indeed, the deployment of unikernels requires a protection mechanism to ensure that information does not leak from unikernels. In this paper, we present our initial experiments into the use of an approach to creating a Trusted Execution Environment (TEE) in unikernels. We present UniGuard: a security architecture that leverages Intel Software Guard Extensions (SGX) to protect security-sensitive computations inside unikernels. We believe that unikernels are an excellent match for Intel SGX to create a TEE. We implemented our solution on top of the KVM hypervisor and its Intel SGX support. Results show that UniGuard has a comparable 20% overhead when starting an enclave inside a unikernel and 10% when executing ocalls.**

## I. Introduction

Cloud computing relies on a cloud infrastructure to deliver the required services to cloud tenants. A prominent concern among tenants is that a cloud provider is trusted to access a tenant's data. Privileged users such as system administrators of a cloud infrastructure can use privileged software to launch attacks to other software components that tenants use [1]. In addition, vulnerabilities in hypervisors provide an attack vector for malicious insiders [2] to gain access to sensitive information. Hence, there is a need to shield access to a tenant's application and data.

Unikernels are a disruptive technology based on the library OS concept for developing minimal cloud computing applications and services. Unikernels are specialized single address systems with a minimal attack surface, near-instant boot times, and improved system optimization. According to where they are deployed, unikernels are either lightweight virtual machines (VMs), or minimal processes inside an operating system.

Even though unikernels provide a minimal attack surface without the need for a full OS, there is still the concern for the protection of a unikernel's security-sensitive information from a privileged user. Existing approaches have focused mainly on the OS to provide the platform for securing a user's applications using containers [3], library OS [4], and micro-containers [5]. All of the above approaches leverage Intel SGX for achieving protection against software and hardware attacks. By using Intel SGX, these approaches need to sanitize the system calls that originate from an enclave, from privileged software and vice versa. Currently, there has been little research on how to leverage Intel SGX with virtualization technology that also incorporates a minimal attack surface to protect a tenant's application and data from an untrusted cloud.

In this paper, we propose a novel security architecture that integrates unikernels with Intel SGX called *UniGuard*. Our approach is to create a Trusted Execution Environment (TEE) for security-sensitive computations inside a unikernel. We believe that the adoption of TEEs for unikernels will increase the trustworthiness of applications deployed in the cloud and ensure that tenants' data are protected from cloud providers.

Currently, *UniGuard* is able to create TEEs for MirageOS unikernels for the KVM hypervisor. A similar approach can be taken to create TEEs for unikernels deployed on the Xen hypervisor. In this paper, we mainly focus on the KVM hypervisor, while leaving support for the Xen hypervisor as future work.

**Contributions:** In summary, we make the following contributions: 1) A novel security architecture that uses enclaves in unikernels to protect security-sensitive computations; 2) We design and develop a shim library for enclaves that sanitize calls from and to an enclave; 3) We evaluate the overhead of our architecture in relation to the executing the same computation in a regular enclave.

The remainder of this paper is organized as follows: Section II presents background information related to unikernels and Intel SGX. Section III discusses the threat model. Section IV outlines the UniGuard architecture, and Section V discusses our implementation. In Section VI, we provide a security evaluation of UniGuard, and in Section VII we conduct a performance evaluation. Finally, Section VIII outlines related work and Section IX concludes with a discussion of our future plans.

## II. Background

### A. Unikernels

*Unikernels* [6] are specialized single-address VMs where system libraries, language runtime, and application and configuration files are compiled into a minimal VM that boots on a commodity hypervisor, such as Xen or KVM as seen in Figure 1.

Previous research has demonstrated a number of different methods to restructure the components of an operating system. One such way is to change the architecture of a general purpose operating system, and another is to construct a new operating system that uses libraries of the operating system and applications in the same address space [7], [8]. More precisely, the application is linked together with a set of libraries that provide the high-level set of abstractions. These

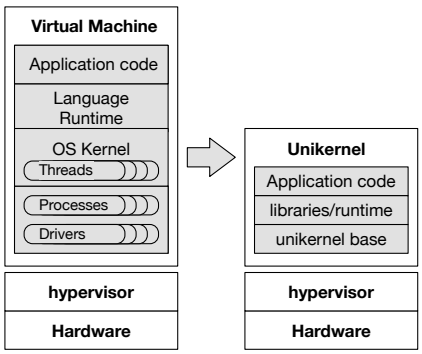


**Fig. 1: Unikernel architecture**

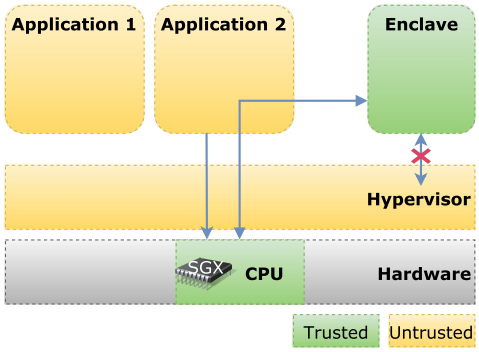


**Fig. 2: Intel SGX architecture**

abstractions take the place of operating system functionality such as processes, files, IPC, network, and threads.

Using such an approach for restructuring an operating system, we have a more flexible method of building applications with only the required libraries. This method eliminates extraneous code that could increase the attack surface of the application.

The main premise of unikernels is that they have a smaller file size, smaller attack surface, minimal footprint, and near-instant booting. Unikernels can be created with a small file size such as 1MB and in many situations unikernels accomplish their task with only 32MB of memory depending on the memory requirements of the application. The size of the allocated memory can be adjusted via a configuration file. Therefore, it is possible to have thousands of unikernels in one single physical host.

One of the existing projects that create unikernels is MirageOS. MirageOS constructs unikernels authored in OCaml and targets multiple platforms such as Xen, KVM, Linux, and macOS. For the first two target platforms, the unikernel is deployed as a regular VM while for the last two targets a user application is deployed. In this paper, we mainly focus on MirageOS unikernels that are deployed to the KVM hypervisor.

### *B. Intel SGX*

Intel Software Guard Extensions (SGX) [9] is a mechanism that enables the creation of Trusted Execution Environments (TEEs). A Trusted Execution Environment is an environment that executes trusted applications in isolation inside an operating system. Intel SGX includes a set of hardware instructions and mechanisms for memory accesses that are used for securing sensitive applications and data. A user application uses SGX to create a protected address area referred to as an enclave. The enclave provides confidentiality and integrity even when privileged software such as an OS or a hypervisor is compromised. The memory area of the enclave cannot be accessed by any software that is not resident in the enclave. The main components of Intel SGX are illustrated in Figure 2

The SGX architecture contains three software components to facilitate enclave developers in using enclaves in their projects. The first component is the SGX driver, which manages the interactions between the OS kernel and the SGX hardware. The second component is the SGX SDK, which enables developers to use the SGX API and the third component is the SGX PSW which includes the platform API.

The SGX SDK provides two function call mechanisms for enclaves depending on where the direction of the call originates from. The *Enclave Call (ECALL)* is a trusted function call, where a user application invokes code inside the enclave and the function call returns a result to the application. The other function call is called *Outer Call (OCALL)* and originates from inside the enclave. This function is untrusted as the execution happens outside the enclave and the result is sent back to the enclave.

### *C. KVM-SGX*

We have chosen the KVM hypervisor over other hypervisors to develop UniGuard mainly because of the KVM's support for unikernels that use hardware-assisted virtualization rather than paravirtualization. The MirageOS unikernels that support Xen only support paravirtualization which cannot be easily integrated with the SGX architecture. SGX requires the supervisor instructions to be executed in protection ring 0, which is not supported in when using paravirtualization which places the guest OS kernel in protection ring 1. If an SGX supervisor instruction is executed in such a configuration, then a general exception is triggered to the hypervisor. Although, it would be possible to catch the exception in the hypervisor and perform the instruction on behalf of the paravirtualized guest, it would require to reengineer the Xen hypervisor to support all the supervisor instructions. That is the main reason that the Xen hypervisor only supports hardware-assisted virtualization for SGX.

Another way this could be solved is to add support for hardware-assisted virtualization in unikernels that are deployed in the Xen hypervisor. This can be achieved by extending the mini-os unikernel base to support hardware-assisted virtualization. Recently, a proposal was presented for the *Unicore* [10] project that aims to include support for hardware-assisted virtualization and provide a holistic unikernel base.

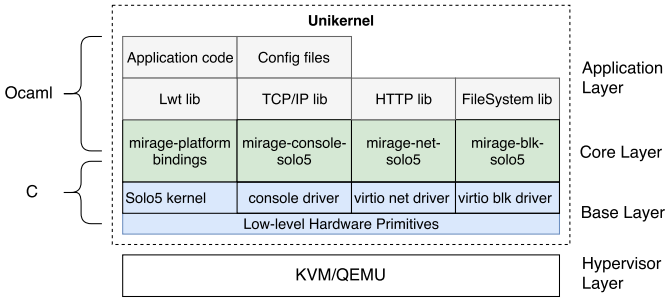


**Fig. 3: Unikernel support for the KVM hypervisor**

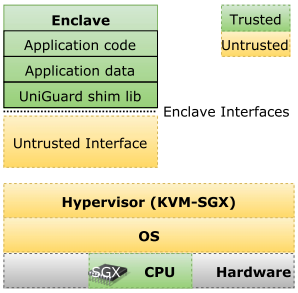


**Fig. 4: UniGuard architecture**

The security mechanism introduced with UniGuard is based on a version of KVM that supports SGX. We will refer to this version of KVM as KVM-SGX for the remainder of this paper.

The main objective of KVM-SGX is to call the SGX driver's API to manage the SGX information such as allocating or freeing EPC pages. KVM-SGX follows a unified model where the SGX driver manages all the EPC pages. The hypervisor is not able to call the driver's API directly, since that hosts without SGX-enabled CPUs will not load an SGX driver in the OS. Hence, if the APIs are called directly, KVM-SGX will not be loaded. This is achieved by using the *symbol_get* function to get the SGX driver's APIs during runtime. KVM-SGX uses the SGX driver to delegate SGX features and the EPC resource detection to the driver itself.

### D. KVM Support for MirageOS Unikernels

MirageOS unikernels provide support for the KVM hypervisor by using the Solo5 unikernel base. Figure 3 illustrates the changes in each software layer needed for the unikernels to support the KVM hypervisor.

The development of the Solo5 unikernel base for KVM requires three elements. First, it requires to setup the kernel hardware initialization that loads data into specified memory regions and starts the processor in the correct mode. Second, the integration with virtio devices which are supported in KVM hypervisor. Virtio provides a paravirtualized standard for the KVM hypervisor and other hypervisors. Third, the *mirage-platform* package includes the functionality for the bindings between the unikernel base and the higher level code used inside the unikernel. The Solo5 unikernel base first supports the KVM hypervisor and QEMU and then it includes support for a specialized unikernel monitor called *ukvm*. This monitor is a user space application that monitors the execution of unikernels and creates a minimal interface with only the required capabilities.

## III. Threat Model

UniGuard adopts a typical threat model for software applications that use SGX. We only trust the CPU package with SGX support and any code that is executing inside the enclave. All other hardware components and software components are not trusted.

The only other component that we trust is the *aesmd* enclave that is provided by the SGX SDK. The goal of this enclave is to verify attributes in the enclave signature and allow the creation of the enclave. This enclave needs to be trusted when remote attestation is used.

UniGuard does not protect from side-channel attacks, denial-of-service (DoS) attacks and controlled-channel attacks [11]. These are vulnerabilities that are shared by all software that is built on top of SGX.

## IV. System Architecture

In this section, we first focus on the design choices for UniGuard. Then we discuss the architecture and the components that comprise UniGuard. Figure 4 depicts a high-level view of the UniGuard architecture.

### A. System Design Choices

Designing a security architecture based on SGX needs to account for three design choices and their trade-offs. First, the amount of functionality to include inside the enclave. On one hand, there are works such as Haven [12] which they include a large percentage of the application code and supporting OS functionality inside the enclave. On the other hand, works such as SCONE [3] wrap a thin layer of the functionality inside the enclave. Even though including more code inside the enclave increases the TCB it minimizes the attack surface of the interface which sits between the enclave and the OS or the hypervisor and its complexity.

Second, the complexity of the shielding support for the SGX. Although SGX can protect an application from malicious privileged software, it cannot protect an application that requires functionality from the OS or the hypervisor. SGX does not protect from Iago attacks that can compromise an application using an unchecked system call. Therefore, a sanitizing mechanism is required to verify or reject inputs from the OS or the hypervisor. The complexity of the enclave API directly affects the complexity of the shielding mechanism.

Third, the complexity of applications that support SGX. Applications range from simple services that execute a cryptographic library inside the enclave which includes a minimal shim library [13] to more complex applications which require a large number of system calls.

Fourth, how the application is partitioned. There are various ways of partitioning applications. One way is for an application to have multiple enclaves where they can communicate

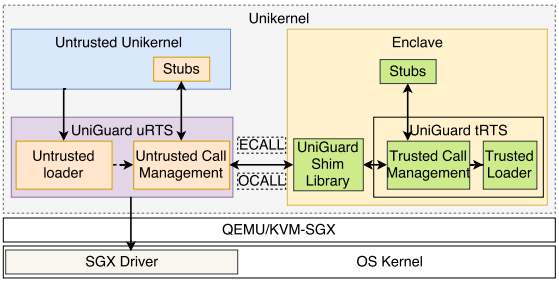


**Fig. 5: UniGuard components**

with each other. Another is to include more functionality outside of the enclave. For UniGuard we focus on having a one-to-one relation between a unikernel and an enclave. This is facilitated by the fact that unikernels include minimal functionality and in most situations, it would require only one enclave. We consider support for multiple enclaves as future work.

### B. UniGuard Architecture

The architecture of UniGuard includes the following components: the KVM-SGX hypervisor, the ukvm, the unikernel layers, the secure enclave, and the shim library that provides an interface to the untrusted part of the unikernel.

Figure 5 depicts the components of the UniGuard architecture. All the software components inside the enclave are considered trusted. We have ported the trusted Run Time Service (tRTS) so that it can be used in UniGuard by using only the code necessary for the unikernels. For instance, we did not port the code for the Intel VTune library that provides profiling information and any code related to the simulation mode of the SGX. The tRTS contains the trusted loader and the Trusted Call Management inside the enclave. The Trusted Call Management provides an interface where ECALLs can be received, and OCALLs sent to the untrusted hypervisor to execute a hypercall. The stubs provide an interface inside the enclave for the security-sensitive application code executed inside the enclave.

On the untrusted part of the unikernel includes the untrusted stubs code which interface with the untrusted Call Management which delegates any OCALLS to the untrusted hypervisor and sends an ECALL to the trusted part of the unikernel. Even though, porting libraries from the SGX SDK and PSW to unikernels it still adds more functionality in the unikernel which could be avoided if these libraries were implemented in OCaml. This is the same approach that the Rust SGX SDK [14] project has taken. We consider this as potential future work for integrating unikernels with SGX.

## V. Implementation

### A. UniGuard Library

UniGuard creates an interface in unikernels where it can accept an enclave as an extra library and load the enclave during the execution of the unikernel. This is achieved by using all the required items needed by SGX to correctly load the enclave such as the enclave signature, the enclave library, and the enclave token. During the creation of the enclave, if the enclave token is not present, then it is created by calling the *aesmd* service that is included with the Intel SDK.

We have created a number of function calls that follow the required partition for SGX. There are untrusted function calls for executing outside the enclave and trusted functions that can be executed inside the enclave. The enclave includes a shim library that it is also responsible for sanitizing each call's parameters. Hence, we are creating a shield for the enclave untrusted calls. In addition, the virtualization mechanism also ensures that the unikernel is isolated from other processes or unikernels in the host. The aim of UniGuard is not only to create enclaves, but to also shield the calls to the OS and the hypervisor.

### B. Shim Library

The UniGuard shim library sits in between the untrusted part of the unikernel and the enclave. Essentially, this library provides an interface that verifies and sanitizes input or output information. The aim of this library is to maintain the confidentiality and integrity guarantees of the enclave. The shim library provides three different mechanisms that sanitize parameters, check pointers, and verify the order of ecalls.

The first mechanism checks the parameters for each ecall according to certain criteria such as size of a string, or length of an array. If the criteria do not match the parameters, then the parameters are sanitized or if they cannot be sanitized the ecall is rejected by the shim library. The second mechanism validates that the pointer should access only trusted memory inside the enclave, or only untrusted memory inside the unikernel. Therefore, mixing up pointers is not allowed by the shim library. If the request cannot be accepted by the shim library, it is rejected. The third mechanism focuses on keeping the correct order of ecalls and providing a freshness counter to mitigate replay attacks.

### C. Changes in ukvm

UniGuard uses the ukvm monitor to provide a specialized hypervisor for unikernels. The ukvm monitor is modified to provide experimental support for SGX in a unikernel. The main modifications are used for providing access to an SGX driver API from inside the unikernel to the enclave. Hence, the unikernel part provides the untrusted part of an enclave application. Even though the unikernel part is untrusted, it is isolated using hardware virtualization and protecting the unikernel from colluding processes if it was constructed as regular OS process. The unikernel includes the port of the SGX related functionality needed to interface with the host's kernel SGX driver. This is architected in this way because the enclave needs to be executed in protection ring 3. Therefore, we also need to create a user mode inside the unikernel and then create the enclave inside the unikernel.

The enclave requires a subset of memory from the memory that the unikernel is allocated. If the enclave memory is more than the available unikernel memory, then an exception is

triggered, and the enclave stops its execution and displays an error message to the console. Subsequently, the unikernel stops execution and the hypervisor destroys the unikernel. If the memory is enough, then the enclave is started and the execution starts inside the enclave, performs the computations assigned to the enclave with confidentiality and integrity and returns control to the unikernel's untrusted part to continue execution. The enclave cannot execute any OS calls from inside the enclave but can delegate the call to the untrusted part of the unikernel which in turn executes the call for the enclave. The unikernel then returns the results to the enclave after first verifying and sanitizing the result. The main idea here is that the unikernel part of the application can perform its computations in kernel mode and then return the result to the enclave.

One advantage of using the ukvm monitor is that we have a specialized monitor that supports SGX. Even though, the ukvm monitor is a user space process it could be secured by using an enclave to hold the main interactions with the unikernel. Using this approach a ukvm monitor is protected from any tampering attempts.

### D. Changes in Solo5 unikernel base

We have introduced a number of changes for the Solo5 unikernel base and related libraries that provide support for the KVM hypervisor in MirageOS. Our first change in the *solo5* package that creates the *solo5-kernel-virtio* library is to introduce another module in the package that provides support for SGX.

## VI. Security Evaluation

We evaluate the security of UniGuard and argue that privileged software cannot compromise the confidentiality and integrity of security-sensitive computations executing inside an enclave. UniGuard security architecture focuses on implementing applications using unikernels with only the functionality needed for the application and the enclave, while also using hardware virtualization to isolate unikernels.

### A. Security Analysis

The security analysis considers that UniGuard is a security architecture that enables unikernels to execute secure-sensitive computations in enclaves using SGX with an untrusted part that has a minimal attack surface.

**Minimal attack surface.** One of our requirements for UniGuard is to develop to a method to execute enclaves in a manner that it does not require an untrusted part with a large code base to realize the application. It has been shown that the number of software bugs increases proportionally to the size of the code [15]. By eliminating code that is not needed, it reduces the probability of a malicious user to exploit a vulnerability in the software. In UniGuard, this is achieved by leveraging Solo5 unikernels and the ukvm monitor. We have added SGX support for both of these components. Instead of using QEMU, which has a large attack surface, we use the ukvm monitor. This software component has a specialized interface for the unikernel it monitors.

**Application-specific API.** A unikernel includes the untrusted part of the application and the enclave corresponds to the trusted part of the application. By using UniGuard, we can create a specialized API according to the requirements of the application that the untrusted part can use to communicate with the enclave interface. Since the untrusted part can only accommodate a specific set of functions, we can eliminate any attacks that involve calls that are not needed by the application.

**Application-specific shim library.** The shim library provided in the enclave matches the functions that the untrusted part of the unikernel supports. Of course, UniGuard does not protect from code inside the enclave that is not well developed. However, UniGuard gives the ability to ensure that the enclave includes a shim library that interfaces with the untrusted part of the unikernel in a minimal way.

**Application partitioning.** Currently, UniGuard does not support automatic partitioning of applications. Instead, we rely on partitioning the applications using a manually way. For instance, if we want to support a legacy application, we would first port the application as a unikernel and then partition the application in an untrusted part and a trusted part that would create the enclave. Therefore, UniGuard is more suitable when we have to create a new application as a unikernel. An application could be separated into multiple unikernels if it is required. Each one of the unikernels can have one enclave associated with it.

**Mitigate internal attacks exploiting untrusted code.** By using unikernels, we minimize the probability that a malicious user can exploit a vulnerability on the application's code base or at the libraries it depends. This is an advantage over creating an enclave in a regular OS that includes a large attack surface. Another layer of defense in UniGuard is the shielding mechanisms that the shim library provides when untrusted function calls are received. This layered defense architecture enables UniGuard to minimize attacks that exploit untrusted code in an application that uses enclaves.

**Mitigate external attacks exploiting untrusted code.** UniGuard helps reduce the likelihood that an external attacker to the unikernel is able to attack the trusted part of the unikernel. This is achieved due to the isolation that the hypervisor provides. We consider the isolation provided by hardware isolation mechanisms achieve a higher degree of isolation than containers or processes in an OS. Containers are considered to have weaker isolation than VMs and there are instances that containers are executed inside a VM in order to achieve the required degree of isolation [16].

## VII. Performance Evaluation

We conducted our performance evaluation of UniGuard using an Intel NUC kit as a virtualization server for our test experiments. Our testbed uses an Intel Core i5–6260U CPU with support for SGX version 1, 16GB of RAM and a 500GB SSD hard disk. The virtualization server is running on top of Ubuntu 15.10 Linux OS with kernel version 4.11.0.

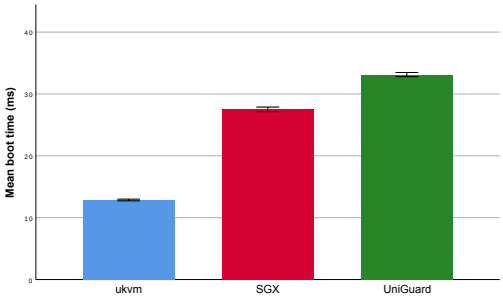


**Fig. 6: Boot times for a single unikernel, an enclave, and UniGuard**

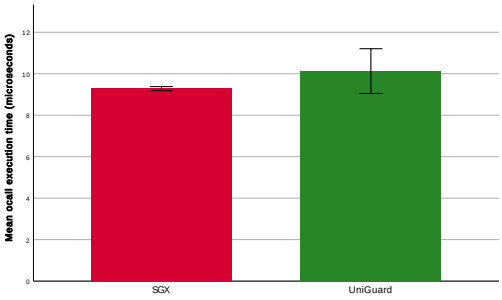


**Fig. 7: Mean execution time for ocalls**

Additionally, the virtualization server uses a custom build of the KVM hypervisor which supports SGX and MirageOS 3.0.5 for creating unikernels.

Figure 6 depicts the results from measuring the boot time of a single unikernel using ukvm, an enclave, and a unikernel using UniGuard. This performance test was executed one thousand times and afterwards the mean was calculated. Results show that booting a unikernel with *ukvm* provides the best overall performance with 11 ms boot time, $95\%$ CI $[11.1, 11.5]$. The enclave is slightly slower, booting at 27.5 ms, at $95\%$ CI $[27.1, 27.8]$. The *UniGuard* test case results show that the unikernel with the enclave boots at 33.1 ms, at $95\%$ CI $[32.7, 33.4]$. Thus, UniGuard adds a moderate $20\%$ overhead regarding boot time over the booting time of a single enclave.

Figure 7 depicts the results from measuring the execution time when an ocall is executed from an enclave and when an ocall is executed when the UniGuard shim library checks the contents of the ocalls. Results show that executing an ocall without the shim library provides the best performance, with mean execution time at 9.28 $\mu$s, $95\%$ CI $[9.18, 9.37]$. The enclave using UniGuard is slightly slower, with mean execution time at 10.1 $\mu$s, at $95\%$ CI $[9, 11.2]$. Thus, UniGuard adds a moderate $10\%$ overhead regarding average ocall execution time over the execution time of a single enclave. The reason for this overhead is the checking of the ocalls that the UniGuard shim library performs.

## VIII. Related Work

Haven [12] executes unmodified Windows applications inside a secure enclave which creates a large TCB. This means that a vulnerability in the operating system can compromise the security of the enclave. Scone [3] proposes a secure container mechanism for Docker that uses Intel SGX to protect containers from an external malicious user. Even though Scone uses a smaller TCB than Haven, it still can be reduced if it was also supporting unikernels. Graphene-SGX [4] supports a wide range of unmodified applications and offers comparable performance overhead. All of the above SGX platforms use unmodified applications, whereas in UniGuard we port the applications to unikernels and create enclaves that are specifically tailored for that application while leveraging the minimal attack surface and the VM-level isolation of unikernels. SGX-LKL [17] is another recent library OS that supports SGX and the ability to execute unmodified applications inside enclaves. This work creates a library OS based on the Linux Kernel Library (LKL) [18] project and a patched version of the musl libc [19] with support for SGX. SGX-LKL requires a disk image with an Alpine mini root environment and the application, which is linked against musl library. UniGuard uses unikernels to facilitate with the loading of the enclaves and relies on the implemented libraries to provide the OS abstractions. This work provides both a simulation and a hardware mode for executing enclaves. UniGuard, however, provides only a hardware mode.

VC3 [20] focus on MapReduce computations and uses Intel SGX to execute them in an untrusted virtualized infrastructure. VC3 has a small TCB and low-performance overhead, and it only supports the Hadoop application. Shinde et al. [5] propose micro-containers which reside isolated inside the SGX enclaves and expose the POSIX abstractions to application logic. The implementation of the OS abstractions is delegated to the OS, instead of emulating it inside the enclave.

## IX. Conclusion

This paper presents our preliminary results on protecting unikernel security-sensitive computations from an untrusted cloud. We demonstrated how we integrated unikernels with Intel SGX. Our solution enables unikernel developers to create secure applications and services in cloud computing leveraging the advantages of unikernels and the security guarantees of enclaves.

Future work includes the introduction of an automatic way of partitioning unikernels to a trusted and untrusted part using annotations. In addition, a way to improve the performance of UniGuard using asynchronous calls for the shim library. Other avenues of future work could be to add support for IncludeOS [21] unikernels and investigate, whether ARM TrustZone [22] can also be used with unikernels. Finally, creating OCaml libraries for the Intel SGX SDK, and PSW, so that they can be easier integrated with UniGuard.

## ACKNOWLEDGMENT

This work was supported in part by the EU Horizon 2020 project PrismaCloud (`https://prismacloud.eu`) under GA $n^o$ 644962.

## REFERENCES

[1] S. Checkoway, H. Shacham, S. Checkoway, and S. Checkoway, "Iago Attacks: Why the System Call API Is a Bad Untrusted RPC Interface," in *Proceedings of the International Conference on Architectural Support for Programming Languages and Operating Systems 2013*. ACM, 2013, pp. 253–264.

[2] F. Rocha, T. Gross, and A. van Moorsel, "Defense-in-Depth Against Malicious Insiders in the Cloud," in *2013 IEEE International Conference on Cloud Engineering (IC2E)*, 2013.

[3] S. Arnautov, B. Trach, F. Gregor, T. Knauth, A. Martin, C. Priebe, J. Lind, D. Muthukumaran, D. O'Keeffe, M. L. Stillwell, D. Goltzsche, D. Eyers, R. Kapitza, P. Pietzuch, and C. Fetzer, "SCONE: Secure Linux Containers with Intel SGX," in *12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16)*, Savannah, GA, 2016, pp. 689–703.

[4] C. che Tsai, D. E. Porter, and M. Vij, "Graphene-SGX: A Practical Library OS for Unmodified Applications on SGX," in *2017 USENIX Annual Technical Conference (USENIX ATC 17)*, Santa Clara, CA, 2017, pp. 645–658.

[5] S. Shinde, D. Le Tien, S. Tople, and P. Saxen, "PANOPLY: Low-TCB Linux Applications With SGX Enclaves," in *Proceedings of the Network and Distributed System Security Symposium (NDSS)*, 2017.

[6] A. Madhavapeddy, R. Mortier, C. Rotsos, D. Scott, B. Singh, T. Gazagnaire, S. Smith, S. Hand, and J. Crowcroft, "Unikernels: Library Operating Systems for the Cloud," in *Proceedings of the 18th International Conference on Architectural Support for Programming Languages and Operating Systems*, New York, NY, USA, 2013, pp. 461–472.

[7] D. E. Porter, S. Boyd-Wickizer, J. Howell, R. Olinsky, and G. C. Hunt, "Rethinking the Library OS from the Top Down," in *Proceedings of the 16th International Conference on Architectural Support for Programming Languages and Operating Systems*, ser. ASPLOS XVI, 2011, pp. 291–304.

[8] D. R. Engler, M. F. Kaashoek, and J. O'Toole, Jr., "Exokernel: An Operating System Architecture for Application-level Resource Management," in *Proceedings of the 15th ACM Symposium on Operating Systems Principles*, ser. SOSP '95, 1995, pp. 251–266.

[17] C. Priebe, "SGX-LKL," 2018. [Online]. Available: https://github.com/lsds/sgx-lkl

[9] Intel, "Intel Software Guard Extensions Programming Reference," Tech. Rep. 329298-002US, Oct. 2014.

[10] S. Kuenzer, F. Huici, and F. Schmidt, "Unicore," Sep. 2017. [Online]. Available: https://lists.xenproject.org/archives/html/mirageos-devel/2017-09/pdfu1nBH1Z1NF.pdf

[11] Y. Xu, W. Cui, and M. Peinado, "Controlled-Channel Attacks - Deterministic Side Channels for Untrusted Operating Systems." in *Proceedings of the 36th IEEE Symposium on Security and Privacy*, 2015, pp. 640–656.

[12] A. Baumann, M. Peinado, and G. Hunt, "Shielding Applications from an Untrusted Cloud with Haven," in *11th USENIX Symposium on Operating Systems Design and Implementation (OSDI 14)*, Broomfield, CO, 2014, pp. 267–283.

[13] P. L. Aublin, F. Kelbert, D. O'Keeffe, and D. Muthukumaran, "TaLoS: Secure and Transparent TLS Termination inside SGX Enclaves," Tech. Rep. DTRS17.

[14] R. Duan, L. Li, S. Jia, Y. Ding, L. Wei, and T. Chen. (2017) Rust SGX SDK. [Online]. Available: https://github.com/baidu/rust-sgx-sdk

[15] S. C. Misra and V. C. Bhavsar, "Relationships between selected software measures and latent bug-density: Guidelines for improving quality," in *Computational Science and Its Applications — ICCSA 2003*, V. Kumar, M. L. Gavrilova, C. J. K. Tan, and P. L'Ecuyer, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2003, pp. 724–732.

[16] F. Manco, C. Lupu, F. Schmidt, J. Mendes, S. Kuenzer, S. Sati, K. Yasukata, C. Raiciu, and F. Huici, "My VM is Lighter (and Safer) Than Your Container," in *Proceedings of the 26th Symposium on Operating Systems Principles*, ser. SOSP '17. New York, NY, USA: ACM, 2017, pp. 218–233.

[18] O. Purdila, L. A. Grijincu, and N. Tapus, "LKL: The Linux kernel library," in *9th RoEduNet IEEE International Conference*, June 2010, pp. 328–333.

[19] C. Priebe, "Modified musl libc for SGX-LKL," 2018. [Online]. Available: https://github.com/lsds/sgx-lkl-musl

[20] F. Schuster, M. Costa, and C. Fournet, "VC3: Trustworthy Data Analytics in the Cloud Using SGX," in *Proceedings of the 36th IEEE Symposium on Security and Privacy*. IEEE, 2015, pp. 38–54.

[21] A. Bratterud, A. A. Walla, H. Haugerud, P. E. Engelstad, and K. Begnum, "IncludeOS: A Minimal, Resource Efficient Unikernel for Cloud Services," in *2015 IEEE 7th International Conference on Cloud Computing Technology and Science (CloudCom)*, Nov 2015, pp. 250–257.

[22] T. Alves and D. Felton, "TrustZone: Integrated hardware and software security," *ARM white paper*, vol. 3, no. 4, 2004.