Trusted Execution Inside Secure Enclaves - LFX Mentorship Report

Işıl Öz

November 24, 2022

1 Hardware Trusted Execution Environment

Since remote computation has evolved as the cloud infrastructures offer high-performance and low-cost solutions, security has become a concern due to remote computer's maintenance by untrusted parties. As the security mechanisms aim to maintain protection from attackers and enable execution confidentiality and integrity, they help to achieve safety requirements as well by making sure that both program code and data are not corrupted. Therefore, memory protection solutions support safety by ensuring trusted execution, and they target domains requiring both secure and safe execution [4, 15]. Homomorphic encryption, which directly works on the encrypted data, enables the handling of user data by third parties safer. However, its computational requirements lead to impractical performance issues.

Trusted execution environments (TEEs) maintain secure computation in an isolated area of a processor with hardware-assisted technologies. They offer both performance and security improvements by exposing a smaller trusted compute base (TCB) in the environment. Confidential Computing Consortium (CCC) defines TEE as an environment that provides a level of assurance of *data integrity*, *data confidentiality*, and *code integrity* [19]. *Data integrity* is preventing unauthorized entities from data modification while in use inside TEE. *Data confidentiality* means that unauthorized entities cannot view data while in use inside TEE. *Code integrity* is preventing unauthorized entities from code modification while it is being executed inside TEE. Depending on the implementation, TEE additionally may provide the following features:

- Code confidentiality: TEE protects code while in use from being viewed by unauthorized entities. For instance, an algorithm having intellectual property issues should not be exposed to third parties.
- Authenticated launch: TEE enables the launching of a process when only it is verified to be secure.
- Programmability: TEE consists of any arbitrary code or code that is loaded by a secure source.

- Recoverability: TEE provides a recovery mechanism that resets the status from a corrupted state to a known safe state.
- Attestability: TEE measures its origin and current state and maintains evidence that the code has integrity and has not been modified.

Different hardware vendors offer TEE implementations in their processor architectures:

- Intel Software Guard eXtensions (SGX): With SGX, which exists in Intel Xeon processors, applications run in a secure memory space without access by the untrusted operating system (enclave) and store the data it uses in this memory area.
- ARM TrustZone: TrustZone is in Arm Cortex-A processors and provides a secure environment for memory, software, bus transactions, and interrupts. The applications running on TrustZone can perform trusted operations with API functions.
- AMD Memory Encryption Technology: While AMD Secure Memory Encryption (SME) is a hardware-based secure memory encryption technology found in AMD EPYC processors, Secure Encrypted Virtualization (SEV) is the implementation of SME technology used in virtualized systems.

We focus on SGX technology and open-source library operating systems that enable its functionality.

2 Intel Software Guard Extensions (SGX)

Intel's Software Guard Extensions (SGX) provide hardware-based isolation for secure computations by supporting safe execution [18]. SGX reserves isolated memory regions called Processor Reserved Memory (PRM) or enclaves for code and data. The CPU protects the memory regions from all non-enclave memory accesses, including the kernel, hypervisor System Management Mode (SMM) accesses, and DMA accesses from peripherals. Non-addressable memory pages (Enclave Page Cache (EPC) pages) inside enclaves are reserved from the physical RAM and encrypted. Execution flow can only enter an enclave by special CPU instructions, similar to the mechanism for switching from user mode to kernel mode. Enclave execution always happens in protected mode, at ring 3.

Figure 1 presents an example execution flow for the applications utilizing protected memory regions. The initial code and data in an enclave are loaded by untrusted system software. During the *loading* stage, the system software asks the CPU to copy data from unprotected memory into the enclave and assigns the pages to the enclave. After all the enclave's pages are loaded, the system software asks the CPU to mark the enclave as initialized, at which point the application can run the code inside the enclave with support for high-level language features. Other parties can perform a software attestation process to verify the identity of the enclave that it is communicating with an enclave and is running in a secure environment.

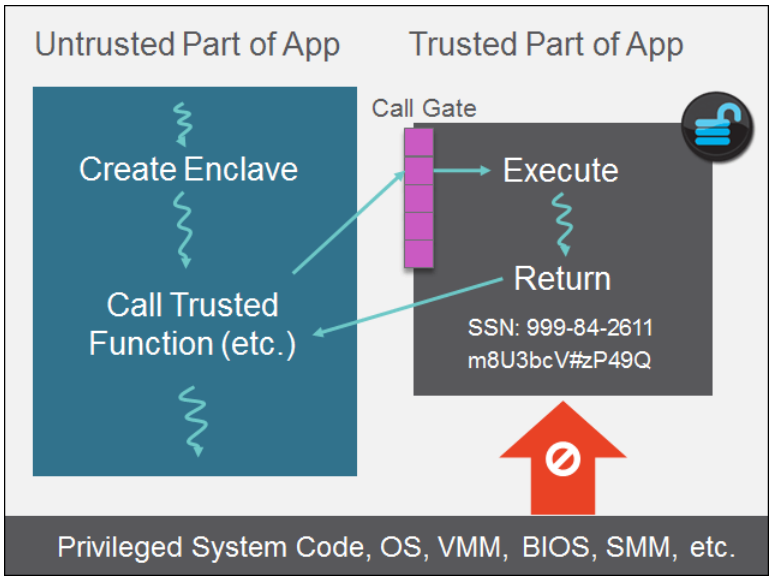


Figure 1: Trusted execution inside SGX enclave [8].

3 Open-source LibOS Projects

The library operating system (LibOS) technology enables us to take an application with little or no modifications and protect it in an SGX enclave [7]. While commercial and open-source LibOS options are available, we mention open-source projects and focus on a specific project, *Mystikos*, our mentorship is built on. The following open-source LibOS projects support Intel SGX:

- Gramine [6]: Lightweight libOS, designed to run a single application inside enclaves with minimal host requirements.
- Occlum [13]: Memory-safe, multi-process library OS (LibOS) for Intel SGX.
- Intel SGX and Linux kernel library [16]: Designed to run existing unmodified Linux binaries inside of Intel SGX enclaves.
- Mystikos [12]: Tools and runtime for launching unmodified container images in TEEs.

4 Mystikos

Mystikos [12] is a runtime and a set of tools for running Linux applications in a hardware-trusted execution environment (TEE). It currently supports Intel SGX. Mystikos aims for the following features: **1)** Enable protection of application code and data while in memory through the use of hardware TEEs, **2)** Allow users and application developers control over the makeup of the trusted computing base (TCB), **3)** Simplify retargeting to other TEE architectures through a plugin architecture.

As shown in Figure 2, Mystikos includes the following components:

- C-runtime based on musl libc: Handles passing the runtime functions from unsecure part to the secure part of the application.
- LibOS-like kernel: Implements system calls for secure execution.
- Kernel-target interface (TCALL): Communicates with the target implementations (SGX target based on Open Enclave SDK or Linux target for only verification on non-SGX platforms).
- Command-line interface: Offers an interface for the programs targeting execution inside the enclave (`myst exec-sgx` or `myst exec-linux`).
- Some related utilities: Tools for building an application for secure execution (`myst-appbuilder`), creating an archive for target program (`myst mkcpio` or `myst mkext2`), packaging an application (`myst package-sgx`).

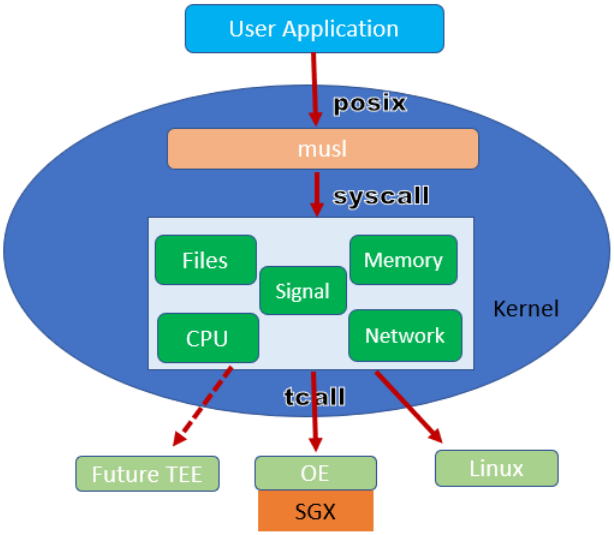


Figure 2: Mystikos architecture [12].

Figure 3 presents the application workflow when developing application in Mystikos, which consists of four steps:

1. Develop application: Develop the application using programming models of our choice.
2. Create appdir: Create and populate an `appdir` directory that contains compiled application and essential dependencies.

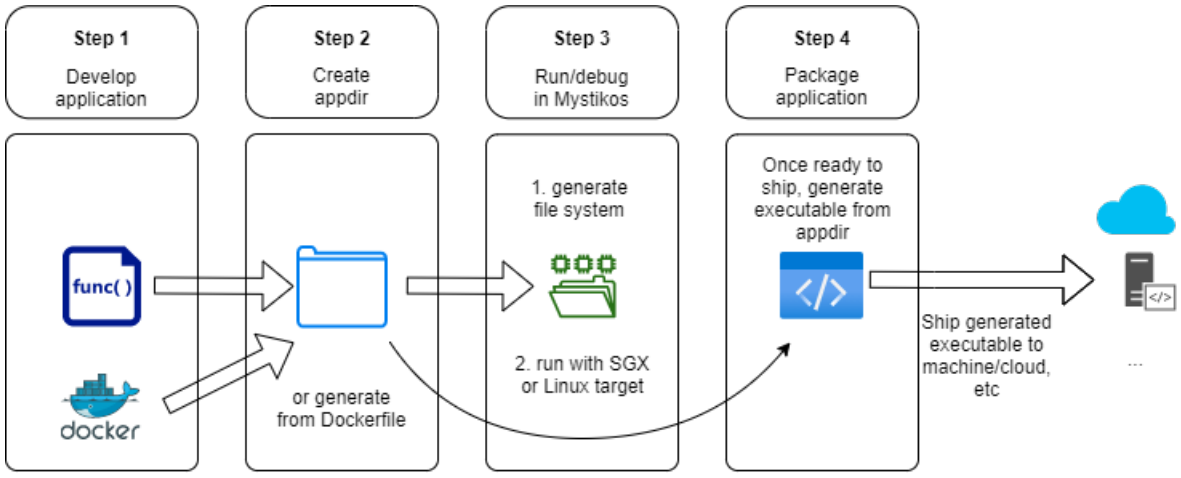


Figure 3: Application development workflow in Mystikos [12].

3. Run in Mystikos: First create a root file system (rootfs) from appdir by using `myst mkcpio` or `myst mkext2`. Then run the application by using `myst exec-sgx` or `myst exec-linux`.

4. Package application: Sign and package the application.

5 Activities During the Mentorship

At the beginning of the mentorship period, I investigated the open-source libOS projects enabling and implementing SGX features (mentioned in Section 3). With a theoretical background in operating systems and processor architecture, I tried to understand the existing implementations. In the meantime, I learned about some practical issues by following the LFX courses about Linux kernel and open-source project development (like GitHub issues and pull requests) to get hands-on experience. Specifically, I enrolled two LFX training courses: A Beginner's Guide to Open-Source Software Development (LFD102) [3] and A Beginner's Guide to Linux Kernel Development (LFD103) [2].

After deciding to work on Mystikos project, which is closely related security and safety issues, I started by examining open issues in *Mystikos* Github repository.

The first issue I have looked at was the following: `Memory leak in myst_syscall_execve` ([11]): It was about a potential bug in the function `myst_syscall_execveat` inside the file `/kernel/syscall.c`. The reason seemed to have an unfreed memory allocation in the function `myst_exec` inside `/kernel/exec.c` file. After discussion with the maintainers regarding the issue, I created a pull request and submitted it as a fix (Figure 4). However, the maintainers realized that the issue has been already fixed in the previous commits, so my commit has not been merged.

After being introduced in Mystikos and getting familiar with the code, I contacted to the maintainers. Vikas Amar Tikoo replied and offered me to work on improving debug-

gability by either writing gdb extensions, improvements to strace mechanism in Mystikos, interactive shell support, or integration with native debugging tools like valgrind. Then he created an issue on github that I can work on: `Interactive bash support issues` ([9]). The aim is to run bash shell in Mystikos. However, Mystikos complains about it. I could replicate the case in my local machine (Figure 5). The problem seems that the `ioctl` requests/commands [10] have not been supported by Mystikos. I submitted a pull request with `TIOCGPGRP` and `TIOCSPGRP` `ioclt` system calls (Figure 6). I have been working on the issue and communicating with the maintainers about the implementation details.

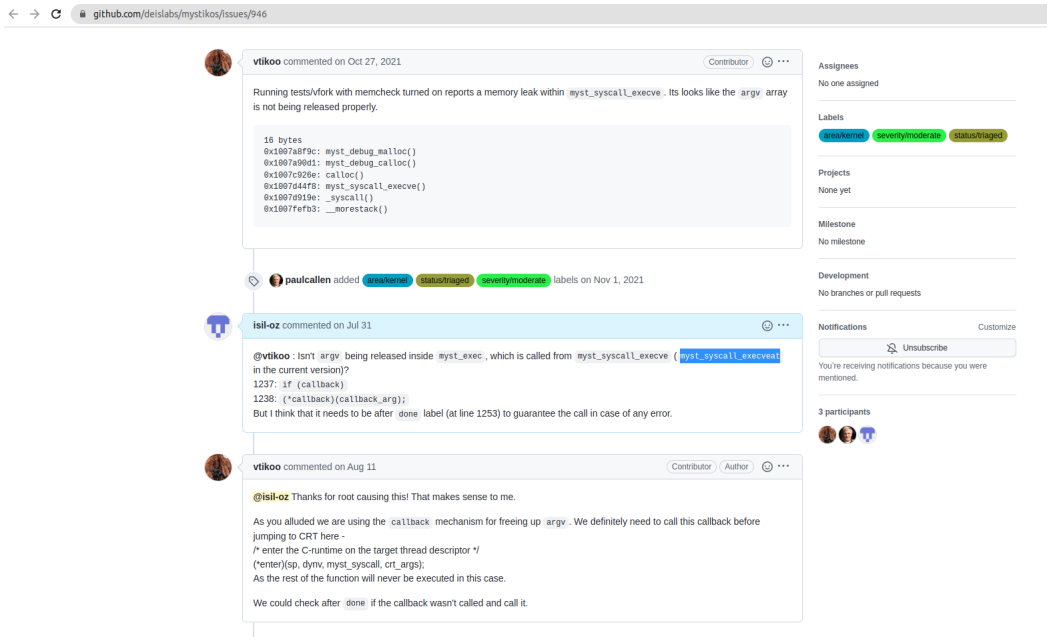
Additionally, during my mentorship period, I attended the following online activities:

- SGX Community Day Virtual Event, July 26-27, 2022 [1]: Inspirational talks from academics, startups, and big corporations about SGX implementations.
- ELISA summit, September 7-8, 2022 [5]: Introductory overview, emerging trends, and current topics in utilizing open-source software for safety-critical applications.
- SOAFEE Virtual Symposium, November 16-17, 2022 [17]: Presentations by open-source and independent software vendors organized by the Scalable Open Architecture For Embedded Edge (SOAFEE) Special Interest Group.

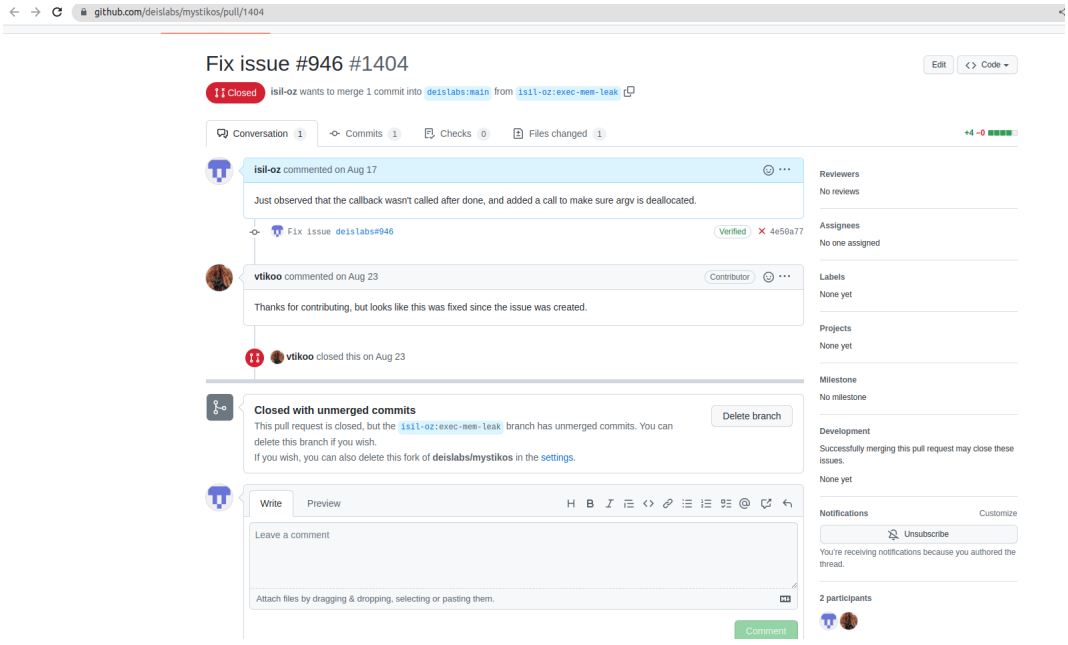
Moreover, as a researcher working in the field of computer architecture, the mentorship helped me to improve myself in the hardware security area. I have submitted a national research project titled as *Security-Performance Tradeoff Analysis for Embedded Systems with GPUs*, where we target to analyze both the security and performance of the embedded applications running inside enclaves. We aim to utilize the open-source LibOS projects and port the program executions partially into the enclaves. Mainly, we target to extend the mentorship work for safety-critical embedded CPU-GPU systems. Additionally, my mentorship encouraged my students to work on open-source projects and be included in mentorship programs.

Thanks to the LFX mentorship program supported by ELISA, I found an opportunity to be introduced to the open-source software community and gained a practical hands-on experience in open-source software development. The mentorship was helpful in terms of both enhancing my hands-on skills and learning the open-source project development process. While I have a strong background in theoretical operating systems and computer architecture area, especially my operating systems software skills have substantially improved. Moreover, the regular meetings with my experienced mentor and communication with the open-source project maintainers were very helpful for both technical and social skill development.

After getting experience on the Mystikos project, I am working on the issue and will continue working on it after my mentorship period. With this experience, I am planning to contribute to other open-source LibOS projects by combining my research interests and practical experience. After being introduced to the open-source community, I believe that I will be able to find more opportunities in the related domains requiring both security and safety. Since I learned that secure execution environments/containers maintain secure and

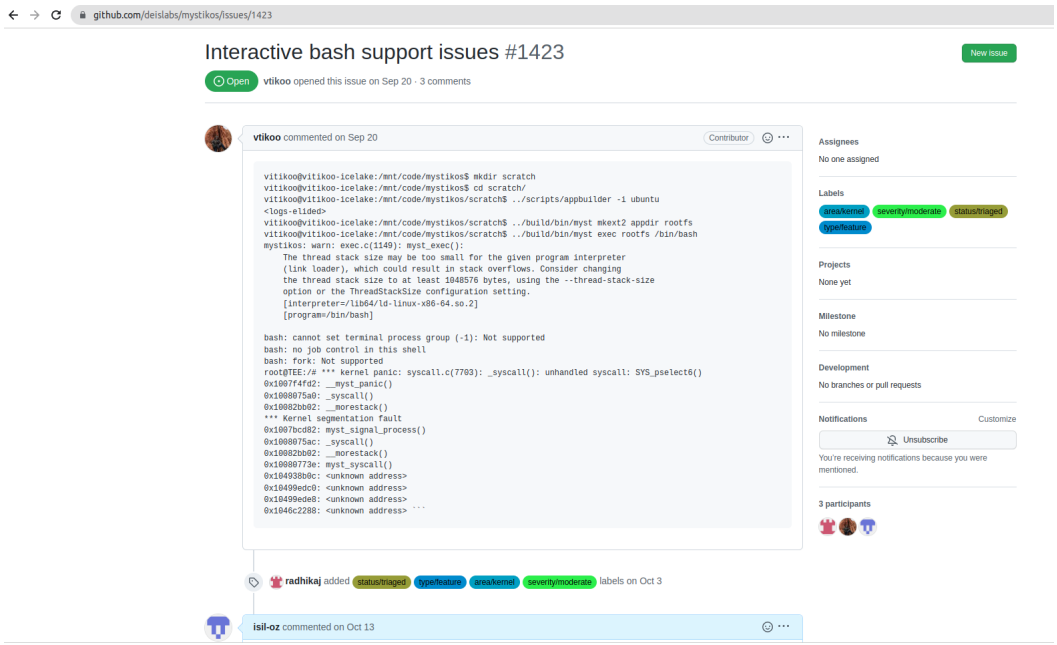


(a) Mystikos issue.

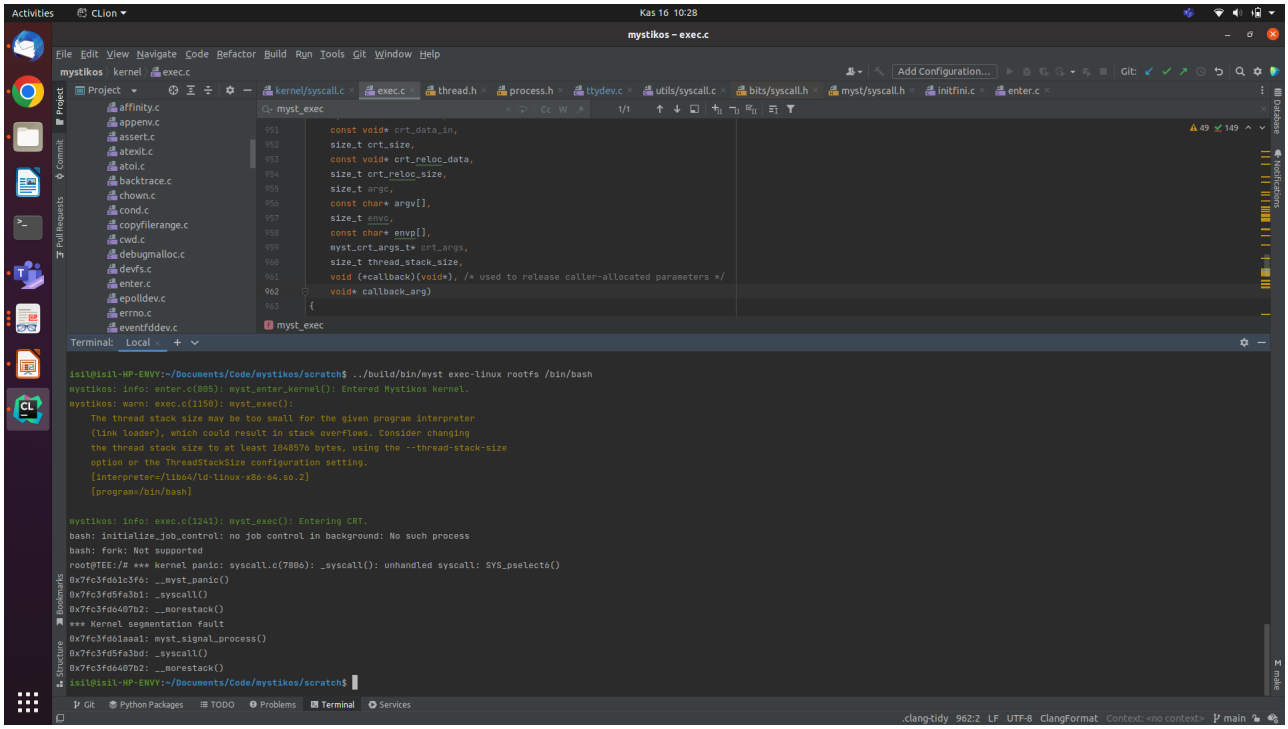


(b) Pull request for fixing the issue.

Figure 4: Working on memory leak issue [11].



(a) Mystikos issue.



(b) Replication of the issue.

Figure 5: Working on Interactive bash support issue [9].

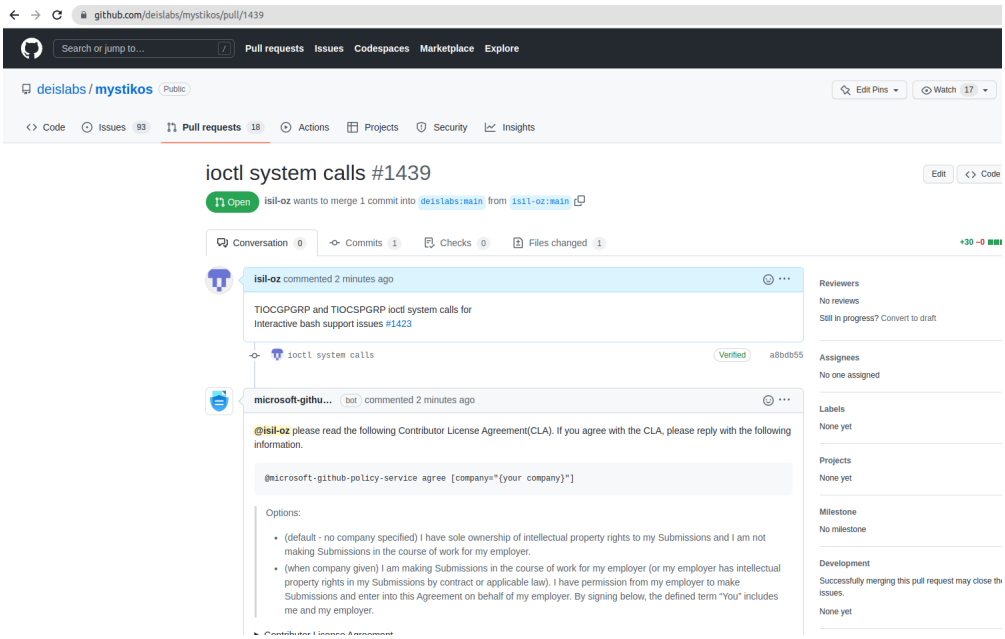


Figure 6: Pull request for ioctl system calls [14].

safe execution inside enclaves, we can utilize them for different applications in safety-critical systems.

References

[1] 3rd sgx community day virtual event. https://community.intel.com/t5/Blogs/Tech-Innovation/Data-Center/Third-SGX-Community-Day/post/1393177. Accessed: 2022-11-16.

[2] A beginner's guide to linux kernel development (lfd102). https://trainingportal.linuxfoundation.org/learn/course/a-beginners-guide-to-linux-kernel-development-lfd103. Accessed: 2022-11-16.

[3] A beginner's guide to open source software development (lfd102). https://trainingportal.linuxfoundation.org/courses/a-beginners-guide-to-open-source-software-development-lfc102. Accessed: 2022-11-16.

[4] Eclipse software defined vehicle working group. https://sdv.eclipse.org/. Accessed: 2022-11-23.

[5] Elisa summit. https://events.linuxfoundation.org/elisa-summit/. Accessed: 2022-11-16.

[6] Gramine - a library os for unmodified applications. https://gramineproject.io/. Accessed: 2022-11-16.

[7] Intel software guard extensions. https://www.intel.com/content/www/us/en/developer/tools/software-guard-extensions/get-started.html. Accessed: 2022-11-16.

[8] Intel software guard extensions (intel sgx) web-based training. https://www.intel.com/content/www/us/en/developer/articles/technical/intel-sgx-web-based-training.html. Accessed: 2022-11-16.

[9] Interactive bash support issues. https://github.com/deislabs/mystikos/issues/1423. Accessed: 2022-11-16.

[10] ioctl(2) — linux manual page. https://man7.org/linux/man-pages/man2/ioctl.2.html. Accessed: 2022-11-16.

[11] Memory leak in myst_syscall_execve. https://github.com/deislabs/mystikos/issues/946. Accessed: 2022-11-16.

[12] Mystikos. https://github.com/deislabs/mystikos. Accessed: 2022-11-16.

[13] Occlum - a library os empowering everyone to run every application in secure enclaves. https://occlum.io/. Accessed: 2022-11-16.

[14] Pull request - ioctl system calls. https://github.com/deislabs/mystikos/pull/1439. Accessed: 2022-11-24.

[15] Scalable open architecture for embedded edge (soafee) project. https://soafee.io/. Accessed: 2022-11-23.

[16] Sgx-lkl library os for running linux applications inside of intel sgx enclaves. https://github.com/lsds/sgx-lkl. Accessed: 2022-11-16.

[17] Soafee virtual symposium. https://soafee.io/blog/2022/virtual_symposium/. Accessed: 2022-11-16.

[18] Victor Costan and Srinivas Devadas. Intel sgx explained. *Cryptology ePrint Archive*, 2016.

[19] A Publication of The Confidential Computing Consortium. Confidential computing: Hardware-based trusted execution for applications and data. Technical report, January 2021.