SYSTEMATIZATION OF TRUSTED I/O SOLUTIONS FOR ISOLATED EXECUTION ENVIRONMENTS

Florin-Alexandru Stancu[1], Alexandru-Alin Mircea, Răzvan Rughiniș, Mihai Chiroiu

Nowadays, operating systems have become increasingly complex, with codebases of millions of lines inadvertently containing security bugs. Modern CPUs promise a solution to this with Trusted Execution technologies (e.g., ARM TrustZone, Intel SGX) which can be employed to isolate critical applications from unauthorized access even from more privileged actors (e.g., system kernel or hypervisor). However, as the traditional way to interface with I/O peripherals is by using drivers part an now-considered untrustworthy OS, separate solutions must be devised to secure user input or output to trusted environments by ensuring an authenticated pathway.

Our paper provides a systematization of the available trusted I/O path solutions by reviewing the scientific works in this field and extracting common attributes and differences. We categorize them by supported peripheral type and trusted platform, comparing the usability, security and complexity of their implementations. Finally, we discuss the results and give future directions for improvement.

Keywords: security, trusted execution, TEE, device virtualization, trusted I/O path, hardware

1. Introduction

In the security world, increased software complexity is often regarded as common source for vulnerabilities, introducing a large attack surface that is becoming more and more difficult to validate. Moreover, an application's security depends on the framework / libraries used, the underlying operating system [13], an optional virtual machine monitor [19] and the system's firmware and hardware. This is commonly referred to as the Trusted Computing Base (TCB) which, in the above example, is totaling to tens of millions of lines of code which may negatively affect the security of the system.

To help improve the trustworthiness of critical applications, hardware-assisted execution isolation technologies started to emerge, promising a smaller TCB and better verifiability. Popular ones include ARM TrustZone, Intel Software Guard Extensions and AMD Secure Encrypted Virtualization available in commercial CPUs. These allow pieces of code to run in isolated regions called

[1]Dept. of Computer Science and Engineering, University POLITEHNICA of Bucharest, Romania, e-mail: florin.stancu@upb.ro

Trusted Execution Environments (TEE), where their execution flow and private data are protected even against higher-privileged components such as the OS kernels or hypervisors.

This new security framework does not come without any downsides, though. Traditionally, the operating system is used to interface with the external world (e.g., network, storage, hardware peripherals for user interaction). TEEs can still leverage these services from an OS running side by side, but their behaviour is considered untrusted and must be secured by other means.

In this paper, we take on the Trusted I/O problem: establishing secure communication channels between trusted environments and user-interacting hardware peripherals. Common use cases include: keyboard / touch screen for password or PIN number input (for remote application authentication) or transaction validation (e.g., authorizing money transfer) or trusted display / printer output to prevent malware from viewing sensitive data (e.g., secret documents) or altering specific screen regions (phishing protection).

Our contributions are:

- A thorough description of the popular TEE platforms and trusted I/O path architectures;
- An extensive survey of the available I/O security solutions;
- A comparison table for the trusted path implementations by software complexity (TCB size), applicable trusted execution platform and device usability;

The rest of the paper is organized as follows: in Section 2, we give some background on current TEE technologies, in Section 3 we define the Trusted I/O Path problem and generalized approaches, in Section 4 we present the available state of the art, which we follow up with the systematization and discussion in Section 5, concluding with Section 6.

2. Background

A trusted execution environment (TEE) can be described as an isolated memory region and CPU execution context where programs can run with platform-assured confidentiality and integrity protections from the normal (rich) environment, especially against higher-privileged components (OS kernel, hypervisor, firmware or even physical tampering), often regarded as untrustworthy. Its primary goal is to protect sensitive workloads where security is important by minimizing the TCB of the system.

Although software-only TEE implementations have been tried (Virtual Ghost [3], SofTEE [10] using various instrumentation techniques for deprivileging the OS kernel), such approaches usually incur huge performance overhead upon the whole system (including normal / untrusted operation) and, arguably, still retain large TCB sizes with the added complexity of the instrumentation code. Consequently, modern TEEs are designed as a combination of software (firmware / CPU microcode) and hardware features (CPU and

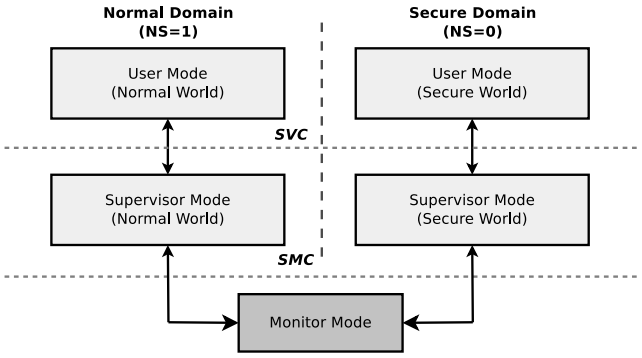

FIG. 1. TrustZone Architecture

chipset modifications, other secure elements) for realizing the platform's isolation requirements [24].

Intel Trusted Execution Technology (TxT) [8] was first introduced in 2006 as a series of CPU architectural extensions to support trusted computing principles. It made it possible for a trusted program to launch late (after the system was initialized and the normal OS has started) by calling the *SINIT* instruction, creating a Dynamic Root for Trusted Measurement and refreshing the the TCB to a clean slate. It was meant to be used together with the Intel Virtualization Technology (VMX [20]) to spawn a virtual machine monitor (VMM) to securely isolate multiple VMs, and also integrates with a Trusted Platform Module chip to provide code / data integrity measurement, sealing and remote attestation features.

Several research projects obtained trusted execution environments out of the DRTM technologies. Notable ones are **Flicker** [16], a framework for executing small pieces of application code in a isolated environment (one at a time), and **TrustVisor** [15], a secure hypervisor with a very small footprint ($\approx 6K$ LoC) providing memory isolation and integrity measurements, able to run multiple trusted applications while keeping the normal operating system responsive.

For embedded applications, ARM-based system on chips have the **TrustZone** [1] feature providing hardware-enforced separation between two domains: the Secure World, where a Trusted Execution Environment can be implemented, and the Normal World, where the rich software stack resides (i.e., the untrusted operating system and user applications), as illustrated in Figure 1. The TrustZone architecture has a new Secure Configuration Register containing the NonSecure (NS) flag used for memory / interrupt request authorization by the various on-chip peripherals. An additional privilege level is also introduced: the Secure Monitor, responsible for context switching, usually implemented as part of the trusted firmware together with the secure platform initialization.

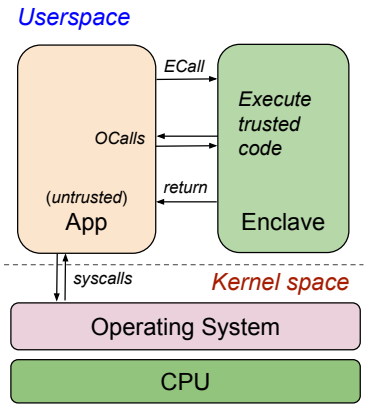

FIG. 2. Intel SGX architecture and call flow

Finally, a different approach for a TEE is taken by Intel with their newest Software Guard Extensions [9] for their general purpose x86 CPU family. SGX isolates userspace applications to run in special execution contexts called *Enclaves*, with hardware-level protections against a privileged Operating System reading their memory or altering the execution flow, resulting in a minimal TCB comprised only of the CPU hardware, its microcode / embedded firmware and the enclave software [2]. From the user's perspective, an SGX application installs and launches the same as any other program of the Operating System. For a developer, the application must be split in two major components: the enclave code (which will be executed inside the TEE), and the untrusted program (used initially to load the enclave and provide untrusted OS services, e.g. networking, file system, peripheral access). The untrusted userspace programs and their enclaves may switch back and forth using Enclave Calls and Outside Calls as present in Figure 2.

3. Problem Description

Many security-critical applications require interaction with the peripherals (e.g., keyboard, display, touchscreen) or some other devices (e.g., industrial equipment connected over serial adapters). These applications would greatly benefit from being isolated inside a Trusted Execution Environment, but the usual way they interact with the hardware is by making use of untrusted Operating System services (via its device drivers). To protect against this, either access to the specific hardware peripherals needs to be denied from the OS, or a trusted communication channel must be established with the application TEE such that a malicious Man-in-the-Middle kernel would be unable to interfere. This is defined as the *Trusted I/O Path* problem [25], and there are multiple approaches for solving it depending on the peripheral device's class and available platform features.

3.1. Attacker model

As to any security research problem, the threat model needs to be defined. The trusted I/O path assumptions are the same as the TEE platform's: any component outside of the TCB is untrusted. Notably, the OS kernel space where the device drivers usually reside is considered to be compromised, so an adversary may have privileged access to any untrusted device at any time (e.g., is able to modify internal registers, remap DMA regions, capture interrupts). The Man-in-the-Middle OS may also have a transparent (undetectable) behavior and still manage to access the user's secrets (e.g., key logging). An OS can also do denial of service attacks as to make devices unavailable to a TEE: some solutions might tackle this angle, but are usually regarded as out of scope.

Various hardware security problems such as device firmware vulnerabilities or platform-specific side channels (e.g., speculative execution) are considered as a complementary research direction and not discussed here. Also, physical access actors are ignored by most works in this field. Thus, we consider a valid trusted path solution must only guarantee confidentiality and integrity of the I/O data exchange.

3.2. Trusted path solutions

Some trusted platforms were designed with proper abstractions to support trusted paths. The obvious example here is ARM TrustZone, where a trusted firmware runs at an extra-privileged exception level (i.e., the Monitor Mode) and is able to configure hardware security registers to block I/O access (memory and interrupts) for specific devices from the Normal World, though the granularity depends on the actual System on Chip (SoC) implementation. Similarly, in virtualization-based TEE technologies (e.g., Intel TxT, Flicker and TrustVisor) the Virtual Machine Monitor also employs a privileged execution context which can be used implement device emulation (optionally sped up by CPU virtualization features) to alter the Operating System's view, though the resulting TCB complexity may be high.

Other trusted execution technologies like Intel SGX were designed such that programs are only able to access the CPU resources from the least-privileged user mode, which brings an additional problem: going through a potentially-malicious Operating System is mandatory. In this case, there are several viable approaches, each presenting downsides, e.g.: adding a small hypervisor implementing hardware access mediation to the TCB (increasing its size), or use specialized hardware (either chipset, third party middleware or end devices) supporting a trusted authentication protocol.

Furthermore, from a usability point of view, some applications require that some devices alternate between secure and normal operation. For example a keyboard or touch screen is required for normal OS utilization but also for e.g., password authentication or transaction confirmations in several trusted uses. The same for the display: it would be inconvenient to have a user own

two separate monitors. So, a trusted I/O implementation must support access multiplexing, which requires even more security safeguards; for example, an adversary might use phishing-type attacks or steal secrets using peripheral side channels (historic / cached device data).

4. State of the Art

In this section, multiple works in the Trusted I/O Path field will be presented, highlighting their novelty concepts, notable differences and improvements. Note that, to each of the articles, a single-word alias has been given, which will be used through the rest of the paper for easy identification.

Zurich Trusted Information Channel (ZTIC [21])
The paper describes one of the first implementations of an external USB device establishing a trusted path with a remote service. It is marketed as a solution for securing sensitive Internet transactions (e.g., banking applications). Although it doesn't employ a Trusted Execution Environment (uses a trusted server instead), it illustrates the concept of authenticated channel termination for taking user confirmation input (via embedded buttons) and displaying transaction data using a small LCD screen.

Bumpy: Safe Passage for Passwords and Other Sensitive Data (Bumpy [14])
Similarly to the previous paper, Bumpy takes care of sensitive input and output between a web server and a DRTM-based TEE (Flicker [16]) for protecting user input and display from keyloggers / screen scrapers. It uses a special input sentence to trigger the beginning of a secure transaction by using a custom-made USB interposer between the keyboard and the trusted environment. Instead of using the untrusted PC monitor, a smartphone application is proposed as alternative (trusted) display. The full protocol, from input to server then to output, is described and thoroughly analyzed.

Unidirectional Trusted Path (UTP [7])
The authors developed a system for authenticating that a transaction with a remote server was initiated by a human user in the face of untrusted client-side stack (malicious OS). It does so by using a trusted environment (Intel TxT / Flicker, but the solution is described as generic) and ensuring a trusted path by using hypervisor I/O mediation. They implement virtualization drivers for keyboard and a text-only VGA display mode. The unidirectionality from the solution's naming comes from the fact that it only ensures a one-way validation: from user to server, and not the other way around. Authentication is aided by Trusted Platform Module measurements of the executed client code.

Building Verifiable Trusted Path on Commodity x86 Computers (VTP x86 [25])

The paper gives comprehensive study of I/O virtualization challenges, arguing the fact that device driver-based isolation is insufficient for providing secure I/O to a TEE (due malicious re-configuration of memory maps). A new hypervisor is designed with countermeasures for all classes of attacks discovered so far, allowing for the establishment of a valid trusted path. An implementation is demonstrated using TrustVisor [15] with isolation drivers to secure a PS/2 keyboard and VGA display. Finally, a couple of architectural changes for the x86 platform are suggested to help simplify such solutions, hence decrease the size of the system's TCB.

Intel Protected Audio and Video Path (Intel PAVP [18])

Intel's PAVP is a commercial, trusted display solution for hardware-accelerated decoding of encrypted video streams from trusted sources. An authenticated session is established between a PAVP-enabled graphics chip / core and a trusted party (a TEE, in our case) resulting in a shared symmetric key which cannot be extracted even by privileged software. Unfortunately, it is only available on Intel's integrated GPUs and only usable with a commercial license agreement.

Building Trusted Path on Untrusted Device Drivers for Mobile Devices (TrustUI [11])

TrustUI is a solution for ARM mobile devices for providing an input and output trusted path for mobile devices. It employs TrustZone for a trusted execution environment, but avoids implementing full separation drivers by using several anti-grabbing / anti-spoofing techniques. For secure input, the on-screen keyboard is randomized when shown. The display output is protected by locking the framebuffer while the trusted application running and trapping normal OS calls for reading it. Additionally, anti-phishing / overlay protection is implemented by using a small RGB LED on the phone only usable by the Secure Microkernel for the user for cross-checking with a randomized background color rendered by the trusted program.

Wimpy Kernels for On-demand Isolated I/O (Wimpy [26])

The paper presents a new security architecture for having trusted on-demand I/O access (shared with the normal Operating System) by their equivalent of a TEE (called wimp apps due to their small TCB). The I/O isolation is accomplished by modifying a micro-hypervisor (a formally verified XMHF, in their implementation) and wimp kernel (addon to the untrusted OS) which outsources the device drivers to the untrusted kernel and only does a series of minimal verifications to exclude malicious actions. They demonstrate their solution by taking on Linux's USB subsystem and showing the code sharing techniques to drastically reduce the TCB of the applications.

Trusted display on untrusted commodity platforms (GSK [23])

The authors engineer a novel GPU separation kernel architecture for bringing trusted display to commodity computers. By using a micro-hypervisor, they avoid virtualizing each GPU object and simply implement address space

separation for preventing unauthorized access. To avoid memory re-mapping attacks using various GPU configuration methods, they mediate write access to the GPU command buffer while a trusted application rendering is in progress. A proof of concept is realized using a TrustVisor TEE [15] and the Intel integrated graphics, preserving the high performance standards required for graphics.

SGXIO: Generic Trusted I/O Path for Intel SGX

SGXIO offers the means for SGX applications to have trusted path to I/O devices. It uses a virtualization-based architecture, establishing an authenticated channel between the enclaves and the hypervisor while going through the untrusted OS. The I/O mediation is realized through security drivers, which the authors note that they require careful design to keeping the TCB small. The hypervisor's integrity and authentication with SGX enclaves is realized by using a TPM-assisted measured boot scheme, while proposing a user-centric protocol (typing secrets on the keyboard) for the local attestation of the TEE applications.

Bastion-SGX: Bluetooth and architectural support for trusted I/O on SGX (BASTION-SGX [17])

BASTION-SGX shows a different approach for establishing trusted I/O paths for Bluetooth-based devices: secure TEE channel termination inside the Bluetooth host controller chip. They analyze the structure of the controller's firmware and propose a series of lightweight code and protocol modifications to secure wireless peripherals for TEE applications.

PROXIMITEE: Hardened SGX Attestation Using an Embedded Device and Proximity Verification (ProximiTEE [5])

SGX enclaves' remote attestation feature allows a remote party to verify that the TEE in question is an honest one and meets the necessary conditions for the data exchange. The authors show that this process is vulnerable to relay attacks (e.g., using an attacker-controlled SGX-based intermediary). The paper brings a new solution against such attacks by introducing a small hardware device, ProximiKEY, sealing a private key and providing the remote service with its public counterpart. For each attestation session, the local device is used with a distance bounding protocol and ensure relay protection. Additionally, the authors argue that their ProximiKEY attestation process can be extended to trusted I/O path between the user and the TEE enclaves.

Secure Input/Output for Intel SGX Enclaves (TIO [6])

With TIO, we designed a portable microcontroller-based device (the size of a stick) having two USB ports (one for PC connection, one for a peripheral), establishing trusted channels with Intel SGX enclaves otherwise lacking secure I/O support. An authentication procedure is also discussed, using a one-time bootable OS for provisioning public keys between the SGX enclaves and the device. The device also takes usability into account: it supports a seamless peripheral pass-through to the OS when secure access is not required by any

application, as well as anti-phishing procedures. An implementation is realized using for a trusted keyboard interface with the enclave with almost no latency overhead.

Establishing Trusted I/O Paths for SGX client systems (Aurora [12]**)**

Aurora comes with a novel approach for a virtualization-based I/O isolation solution: it uses the System Management Mode, a highly-privileged level (above the VMM) with a special SMRAM tamper-proof memory available on commodity x86 systems (typically, only used by system firmware for critical functions like power management). Their SMVisor is able to provide Intel SGX enclaves with trusted I/O path with minimal performance impact. They also use it to implement a high-frequency trusted realtime clock required by cryptographic protocols requiring non-forgeable time.

5. Comparison and discussion

As presented in the previous section, the available trusted path solutions are diverse, with varying application classes and platform requirements. In here, we extracted their characteristics and present then in a unified table (Table 1).

We used multilateral classification by: peripheral type (Human Input Device / Display / others); the I/O isolation mechanisms used (virtualization-based - *Virt*, chipset-based access control - *Chip*, external device - *Ext*); interface / protocol: USB / Bluetooth / Network; for virtualization-based approaches, their implementation method: MMIO (memory mapped I/O) / device driver virtualization.

Note that the half-circle denotes a partial implementation of the feature; for HID, it means only a subset on input devices are usable; for display, it means text-only output. The additional TCB components are given (including the added device / chip module, besides from the CPU platform).

A first observation is that implementing trusted graphical display is a hard problem: many solutions only worked for partial, text-mode output either using an external LCD or secure console output (using BIOS-like text mode switching). As the only open full-display solution (excluding the proprietary Intel PAVP [18]), GSK [23] relies on virtualization and has a fairly high TCB. This is mainly because of the inherent complexity of computer graphics, which may become feasible once vendors implement appropriate GPU hardware security abstractions for access control on individual resources.

We note that, for the Trusted Computing Base - a desirable comparison metric, the size values were taken as-is from the papers and are unreliable for this purpose because of differences in measurement methodologies (e.g., lines of code vs binary sizes, different supported devices / feature sets, target platforms incurring framework overhead, unoptimized cryptographic libraries used). Additionally, although the latency / overhead figures were published

TABLE 1

Comparison of Trusted Path Solutions

Name	Isolation	Interface	Peripherals: HID	Peripherals: Display	Peripherals: Others	Target TEE	TCB Additions: +Hypervisor	TCB Additions: +Chip/firmware	TCB Additions: +Ext. Dev	TCB LoC
ZTIC [21]	Ext	USB	●	◐	○	*Remote*	✗	✗	✓	≈ $110KB$ [1]
Bumpy [14]	Ext	USB	●	○	○	Flicker	✗	✗	✓	≈ $8.5k$
Bumpy Display	Ext	Network	○	◐	○		✗	✗	✓	≈ $10k$
UTP [7]	Virt	Driver	●	◐	○	Flicker	✓	✗	✗	≈ $2.3k$
VTP x86 [25]	Virt	MMIO	●	◐	●	TrustVisor	✓	✗	✗	≈ $15k$
Intel PAVP [18]	Chip	GPU	○	●	○	DRTM, SGX	✗	✓	✗	n/a
TrustUI [11]	Virt	Driver	◐	◐	○	TrustZone	✓	✗	✗	≈ $10k$
Wimpy [26]	Virt	MMIO	●	○	●	DRTM	✓	✗	✗	≈ $3.5k$
GSK [23]	Virt	GPU MMIO	○	●	○	TrustVisor	✓	✗	✗	≈ $35k$
SGXIO [22]	Virt	Driver	●	○	○	SGX	✓	✗	✗	n/a
BASTION-SGX [17]	Chip	Bluetooth HCI	●	○	●	SGX	✗	✓	✗	n/a
ProximiTEE [5]	Ext	USB	●	○	○	SGX	✗	✗	✓	≈ $5k$
SecDisplay [4]	Virt	USB	●	◐	○	TrustZone	✓	✗	✗	≈ $2k$
TIO [6]	Ext	USB	●	○	●	SGX	✗	✗	✓	≈ $27k$ [2]
Aurora [12]	SMM Virt	MMIO	●	○	●	SGX	✗	✓	✗	≈ $3.3k + 696KB$ [3]

[1] only binary size given [2] counts firmware, enclave framework & full asymmetric crypto lib [3] hypervisor (LoC) + enclave library (compiled binary)

for some works, they were taken for various incomparable scenarios and are ultimately dependent on the hardware / TEE platforms used, so we left them out.

There are also important differences between hypervisor and trusted hardware approaches. For virtualization-based solutions, the flexibility is greater (support for multiple device classes), though great care must be taken to properly isolate I/O requests in order to counter all discovered attacks. Thus, a complete implementation (i.e., secure device drivers) might become even more difficult to validate. Although external device / on-chip implementations may have a limited set of protocols supported (e.g., USB) due to the increased prototyping costs, they bring better isolation and security due to the use of a clean I/O interface and running most secure processing on a separate CPU and may be easier to deploy in a personal or corporate environments (e.g., as plug and play devices, not requiring invasive / privileged software). We recall that *ARM TrustZone* allows for a hybrid isolation: it provides a special Monitor privilege level (in place of a hypervisor) which, when coupled with System-on-Chip I/O

request filtering, makes it possible to obtain smaller TCB sizes because there is no need to emulate / mediate device access in most cases (including for graphics devices). Unfortunately, this TEE platform is currently available for mobile applications only.

6. Conclusion

In this paper, we reviewed several works targetting the Trusted I/O Path problem for securing peripheral interactions in TEE applications. We have found mixed solutions for all commercially available TEE platforms (ARM TrustZone, Dynamic Root for Trusted Measurement, Intel SGX) supporting most common device types and interfaces (e.g., HID keyboard / touch as input, text or graphical display, as well as generic USB and Bluetooth).

There are several approaches for securely isolating the devices: virtualization-based or using in-chip / external hardware, each with its advantages and pitfalls especially regarding the applicable TEE platforms. As two extremes, the ARM TrustZone has native direct support for I/O isolation, while Intel SGX runs its applications in user mode and requires a privileged entity to mediate access. We have also shown that some peripherals (e.g., the display / GPUs) are more difficult than others to protect within trusted environments.

We argue that providing standardized trusted path support from the platforms is important for increasing TEE adoption and applicability. We hope that our systematization effort will help researchers in choosing the next direction.

REFERENCES

[1] ARM Holdings. ARM TrustZone Security Extensions.

[2] V. Costan and S. Devadas. Intel SGX Explained. *IACR Cryptology ePrint Archive*, 2016(086):1–118, 2016.

[3] J. Criswell, N. Dautenhahn, and V. Adve. Virtual ghost: Protecting applications from hostile operating systems. *ACM SIGARCH Computer Architecture News*, 42(1):81–96, 2014.

[4] J. Cui, Y. Zhang, Z. Cai, A. Liu, and Y. Li. Securing display path for security-sensitive applications on mobile devices. *Computers, Materials and Continua*, 55(1):17, 2018.

[5] A. Dhar, I. Puddu, K. Kostiainen, and S. Capkun. Proximitee: Hardened sgx attestation by proximity verification. In *Proceedings of the Tenth ACM Conference on Data and Application Security and Privacy*, pages 5–16, 2020.

[6] D. C. T. F. A. Stancu and M. Chiroiu. TIO - Secure Input/Output for Intel SGX Enclaves. In *International Workshop on Secure Internet of Things (SIOT)*, 2019.

[7] A. Filyanov, J. M. McCuney, A.-R. Sadeghiz, and M. Winandy. Uni-directional trusted path: Transaction confirmation on just one device. In *2011 IEEE/IFIP 41st International Conference on Dependable Systems & Networks (DSN)*, pages 1–12. IEEE, 2011.

[8] W. Futral and J. Greene. Fundamental principles of intel® txt. In *Intel® Trusted Execution Technology for Server Platforms*, pages 15–36. Springer, 2013.

[9] Intel. Intel SGX Software Guard Extensions.

[10] U. Lee and C. Park. Softee: Software-based trusted execution environment for user applications. *IEEE Access*, 8:121874–121888, 2020.

[11] W. Li, M. Ma, J. Han, Y. Xia, B. Zang, C.-K. Chu, and T. Li. Building trusted path on untrusted device drivers for mobile devices. In *Proceedings of 5th Asia-Pacific Workshop on Systems*, pages 1–7, 2014.

[12] H. Liang, M. Li, Y. Chen, L. Jiang, Z. Xie, and T. Yang. Establishing trusted i/o paths for sgx client systems with aurora. *IEEE Transactions on Information Forensics and Security*, 15:1589–1600, 2019.

[13] P. A. Loscocco, S. D. Smalley, P. A. Muckelbauer, R. C. Taylor, S. J. Turner, and J. F. Farrell. The inevitability of failure: The flawed assumption of security in modern computing environments. In *Proceedings of the 21st National Information Systems Security Conference*, volume 10, pages 303–314, 1998.

[14] J. M. McCune. Safe passage for passwords and other sensitive data. In *Proceedings of the Network and Distributed System Security Symposium, 2009*, 2009.

[15] J. M. McCune, Y. Li, N. Qu, Z. Zhou, A. Datta, V. Gligor, and A. Perrig. Trustvisor: Efficient tcb reduction and attestation. In *2010 IEEE Symposium on Security and Privacy*, pages 143–158. IEEE, 2010.

[16] J. M. McCune, B. J. Parno, A. Perrig, M. K. Reiter, and H. Isozaki. Flicker: An execution infrastructure for tcb minimization. In *ACM SIGOPS Operating Systems Review*, volume 42, pages 315–328. ACM, 2008.

[17] T. Peters, R. Lal, S. Varadarajan, P. Pappachan, and D. Kotz. Bastion-sgx: Bluetooth and architectural support for trusted i/o on sgx. In *Proceedings of the 7th International Workshop on Hardware and Architectural Support for Security and Privacy*, pages 1–9, 2018.

[18] X. Ruan. *Platform Embedded Security Technology Revealed*. Springer Nature, 2014.

[19] A. Thongthua and S. Ngamsuriyaroj. Assessment of hypervisor vulnerabilities. In *2016 International Conference on Cloud Computing Research and Innovations (ICCCRI)*, pages 71–77. IEEE, 2016.

[20] R. Uhlig, G. Neiger, D. Rodgers, A. L. Santoni, F. C. Martins, A. V. Anderson, S. M. Bennett, A. Kagi, F. H. Leung, and L. Smith. Intel virtualization technology. *Computer*, 38(5):48–56, 2005.

[21] T. Weigold, T. Kramp, R. Hermann, F. Höring, P. Buhler, and M. Baentsch. The zurich trusted information channel–an efficient defence against man-in-the-middle and malicious software attacks. In *International Conference on Trusted Computing*, pages 75–91. Springer, 2008.

[22] S. Weiser and M. Werner. Sgxio: Generic trusted i/o path for intel sgx. In *Proceedings of the seventh ACM on conference on data and application security and privacy*, pages 261–268, 2017.

[23] M. Yu, V. D. Gligor, and Z. Zhou. Trusted display on untrusted commodity platforms. In *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*, pages 989–1003, 2015.

[24] F. Zhang and H. Zhang. Sok: A study of using hardware-assisted isolated execution environments for security. In *Proceedings of the Hardware and Architectural Support for Security and Privacy 2016*, pages 1–8. 2016.

[25] Z. Zhou, V. D. Gligor, J. Newsome, and J. M. McCune. Building verifiable trusted path on commodity x86 computers. In *2012 IEEE symposium on security and privacy*, pages 616–630. IEEE, 2012.

[26] Z. Zhou, M. Yu, and V. D. Gligor. Dancing with giants: Wimpy kernels for on-demand isolated i/o. In *2014 IEEE symposium on security and privacy*, pages 308–323. IEEE, 2014.