1

INTRODUCTION

Great things are not done by impulse, but by a series of small things brought together

— Vincent Van Gogh

Interaction between human and complex software and cyber-physical systems is an important aspect of modern computing platforms. These interactions enable the users to provide inputs to the software systems and services and interpret the output of a computation. Over the last half a century, researchers in academia and industry spent a significant effort to make this interaction as effortless as possible. IO peripherals and complex user interfaces (UI) are the keys to felicitate this interaction between the user and the end-systems. User interaction through such intuitive UIs was quintessential to the wide-spread deployment of computing devices around us and the rapid adoption of remote applications and services.

Security and safety-critical remote applications such as e-voting, online banking, industrial control systems (such as remote PLCs [1]) and medical devices [2] rely upon user interaction that is typically performed through a *host* system that generally is a standard PC (specifically *x*86 architecture), which gives the host access to the raw IO data that is exchanged between the user and the remote server. The host consists of large and complex system software such as the operating system, device drivers, applications such as a browser, and a diverse set of hardware components that expose the host to a large attack surface. Due to cost and convenience, general-purpose PCs are prevalent in many safety-critical application domains such as industrial plants and hospitals. For example, the WannaCry ransomware incident showed that NHS hospitals relied on Windows XP platforms [3, 4].

Trusted path provides a secure channel between the user (specifically through the human interface devices - HIDs) and the end-point, which is typically a trustworthy application running on the host. A trusted path ensures that user inputs reach the intended application unmodified, and all the outputs presented to the user are generated by the legitimate application. The trusted path can also be extended to general peripherals such as GPU, sensors, etc. Moreover, the trusted path is not only restricted to the interaction between the peripherals and the localhost but could be extended

to a remote system. Hence, the remote trusted path to such remote systems is critical in the presence of an attacker that controls the user's computer. Such an attacker can observe and modify any user interaction data without being detected by the user or the server. The trusted path to the local host is a well-researched area where many solutions focus on using trusted software components such as a trusted hypervisor. Zhou et al. [5] proposed a generic trusted path on *x*86 systems with a pure hypervisor-based design. SGXIO [6] employs both a hypervisor and a trusted execution environment (Intel SGX). However, hypervisors are hard to deploy, have a large TCB, and are impractical in real-world scenarios as most of the existing verified hypervisors offer a minimal set of features.

Trusted external devices are another way to realize secure IO between a user and a remote server. Transaction confirmation devices [7, 8] allow the user to review her input data on a trusted device that is physically separated from the untrusted host. These approaches suffer from poor usability, security issues due to user habituation and are only limited to simple inputs. Proposals such as bump in the Ether [9], MP-Auth [10] use external embedded devices to sign input parameters. However, such solutions do not support output integrity; hence, the attacker can execute UI manipulation attacks to trick the user into providing incorrect inputs. Fidelius [11] combines the previous ideas of Bump in the Ether and trusted overlay to protect keyboard inputs from a compromised browser using external devices and a JavaScript interpreter that runs inside an SGX enclave. Fidelius maintains overlays on display, specifically on the input text boxes, to hide sensitive user inputs from the browser. We investigate the security of Fidelius and discover several issues. Fidelius imposes a high cognitive load on the users. The users need to continuously monitor different security indicators (two LED lights and the status bar on the screen) to guarantee the input's integrity and confidentiality. Furthermore, the attacker can manipulate labels of the UI elements to trick the user into providing incorrect input. The lack of mouse support, which may appear only as a functional limitation, exposes Fidelius to early form submission attacks. The host can emulate a mouse click on the submit button before the user completes all fields of a form. This allows the attacker to perform an early form submission with incomplete input - a violation of input integrity.

Trusted execution environments (TEEs) drastically reduce the trusted computing base (TCB) and provide security to applications, known as enclaves, without having to trust the operating system and hypervisor [12, 13, 14]. Thus, the attack surface is reduced by eliminating two of the largest

sources of vulnerabilities for a system [15, 16]. TEEs use isolation primitives provided by the CPU to exclude all software but a single target application from the trusted computing base (TCB). To achieve this trust model, only the CPU is considered trusted, while all the other hardware in the system is explicitly assumed malicious. Even memory is not included in the TCB and can only be used in conjunction with memory encryption and integrity protection. Such a trust model makes the TEEs ideal candidates for secure interactions with the user.

While SGX's remote attestation guarantees that the attested enclave runs the expected code, it *does not*, however, guarantee that the enclave runs on the expected computing platform. An adversary that controls the OS (or other software) on the target platform can relay incoming attestation requests to another platform. This way, the user ends up attesting to the attacker's platform rather than her own. Such also enable the attacker to see all the IO data to and from the user, e.g., sensitive user input on display. Such relay attacks are a long-standing open problem in trusted computing, as already a decade ago, Parno identified such attacks in the context of TPM attestation [17]. Hence, directly incorporating a trusted path to a TEE like Intel SGX is not a trivial issue.

Apart from the relay attack, TEEs' potential use cases are also limited. TEEs cannot communicate with any external device without going through the malicious operating system. Various applications ranging from trusted path [5, 11, 18] to data center accelerators [19] are excluded from the TEE ecosystem. Any such application with high-security requirements is usually solved by one of three approaches: i) a fully dedicated system, ii) a dedicated virtual machine (with trust in the hypervisor), or iii) blind trust in the operating system. We believe none of these approaches to be satisfactory due to either cost (i) or the need to trust codebases (ii & iii) with millions of lines of code [20, 21]. The static system architecture of TEEs limits their potential applications to trusted computing use cases, where a user possesses some secret data and wants to outsource the computation to a remote system. Many TEEs actually rely on some external peripherals already, e.g., Intel SGX and the monotonic counters in the management engine [22] or various academic proposals for trusted paths using ARM TrustZone [23, 24]. However, all of these systems are custom-tailored for a single-use case and do not allow for any flexibility as they must be configured statically at boot time.

In this dissertation, we address the three following research questions concerning the trusted path and trusted execution in modern platforms

Q1 How to build trusted path systems that provide integrity (and possibly confidentiality) guarantees without the users rely on cognitive-heavy solutions such as transaction confirmation devices while maintaining a small TCB? Are these systems truly secure?

Q2 Why existing solutions failed to provide a trusted path that provides integrity and confidentiality guarantees to the user interactions? What are the necessary steps in order to build one?

Q3 Relay attack from a local platform to a remote attacker-controlled platform is a real threat to the remote attestation of TEEs like Intel SGX. How to ensure that the attacker cannot relay the attestation to an attacker-controlled platform that exposes all the sensitive IO data?

Q4 How to extend the trusted path mechanism to a modern platform that is interconnected with a number of heterogeneous peripheral devices? How to extend this trusted path into other complex hardware devices such as GPU or accelerator that can execute programs outside the CPU cores? How to ensure the platform-wide integrity guarantee (configuration and interaction of the TEE enclaves and external peripherals) is preserved without relying on a purpose-built system?

We propose two systems: IntegriKey [25] and IntegriScreen [26] to answer **Q1**, the first research question. IntegriKey uses input signing to provide a second factor for the integrity of the keyboard input. While the latter uses a phone camera to capture the information that the user is typing on the screen to provide a second factor for user intention's integrity. Unlike exiting trusted path solutions, these proposals rely neither on any transaction confirmation devices that put a heavy cognitive load on the user nor on a hypervisor or trusted drivers that introduce a large TCB. Moreover, IntegriScreen provides a restricted form of output integrity to ensure that the attacker-controlled host does not manipulate UIs. IntegriKey uses a small embedded trusted device that runs a few hundred LoC to sign user input. In IntegriKey, we identify a new form of input manipulation attack that we name field swapping attack. In a swapping attack, the attacker can swap the levels of different input fields that accept overlapping values (e.g., blood pressure and heart rate for a medical implant). IntegriKey analyzes the type of input fields, and based on the regular expression of these fields, it can compute the overlapping fields that could be vulnerable to swapping attack and recommend the user to append a label with the input value to distinguish it. In comparison, the IntegriScreen uses a

phone to verify the user input by using text recognition and send that result to the server on a different communication channel. Even though both IntegriKey and IntegriScreen provide a significant improvement over state-of-the-art trusted path solutions, they suffer similar security and functionality pitfalls as their counterparts. They are all ad-hoc approaches focus on solving a single problem and a single input source. This leads to multiple sophisticated attacks on them.

The shortcomings of the existing literature provide the groundwork of our system named ProtectIOn that answers the first research question **Q2**. ProtectIOn is built on the following observations: i) input integrity is possible only when both input and output integrity are ensured simultaneously, ii) all the input modalities are needed to be protected as they influence each other, and iii) high cognitive load results in user habituation errors. ProtectIOn uses a trusted low-TCB auxiliary device that we call IOHub, which works as a mediator between all user IO devices and the untrusted host. Instead of implementing a separate network interface, the IOHub uses the host as an untrusted transport - reducing the attack surface. ProtectIOn ensures output integrity and confidentiality by sending an encoded UI to the host that only the IOHub can overlay on the part of the screen. The overlay is possible as the IOHub intercepts the display signal between the host and the monitor. The overlay generated by the IOHub ensures that the host cannot manipulate any output information on that overlaid part of the screen; hence, it can not trick the user. All the user interaction to this protected overlay is encrypted and signed by the IOHub. Therefore input integrity and confidentiality are preserved.

Parno [17] identified distance bounding as a candidate solution to TPM relay attacks already ten years ago but concluded that it could not be realized securely as the slow TPM identification operations (signatures) make a local and relayed attestation indistinguishable. However, the implication of the relay attack was never well-studied. In this dissertation, we investigate the implication of the relay attack and show that it could have an adverse consequence. To answer the second research question, **Q3**, We propose a new solution, called ProximiTEE, that prevents relay attacks by leveraging a simple embedded device that we call ProximiKey that is attached to the attested target platform. The ProximiKey executes a challenge-response protocol between itself and the platform and based on the latency of the protocol, and the ProximiKey determines if it is connected to the target platform physically or not. Our solution is best suited to scenarios where i) the deployment cost of such an embedded device is minor compared

to the benefit of more secure attestation, and ii) TOFU solutions are not acceptable. Attestation of servers at cloud computing platforms and setup of SGX-based permissioned blockchains are two such examples.

Finally, to answer **Q4**, the last research question, We propose a TEE with a *configurable* software and hardware TCB including arbitrary external peripheral devices, a concept that we name *platform isolation environment* (PIE). PIE executes applications in *platform-wide enclaves*, which are analogous to the enclaves provided by TEEs, except that they span several hardware components. For example, a platform-wide enclave can be composed of an output device such as a display, input devices such as keyboard and mouse, the CPU that is executing the program, the GPU that renders the graphics, and the custom code running on them. Like in traditional enclaves, a platform-wide enclave can be remotely attested. However, the PIE attestation not only reports a measurement of the software TCB but also of the hardware components that are part of the platform-wide enclave.

The shift towards configurable hardware and software TCBs has wide-ranging implications concerning integrity, confidentiality, and attestation of a platform-wide enclave. Attestation, for one, should cover all individual components of a platform-wide enclave atomically to defend against an attacker that changes the configuration in between attestations to separate components. Moreover, the untrusted OS may remap the peripheral devices at runtime with an untrustworthy device, which should not receive access to sensitive data. We carefully design PIE to not be vulnerable to such attacks and present an in-depth security analysis.

We mitigate the above-mentioned attacks with two new properties of platform-wide enclaves: *platform-wide attestation* and *platform awareness*. Platform-wide attestation expands the attestation to cover all components within a platform-wide enclave, and platform awareness enables enclaves to react to changes in their ecosystem, i.e., remapping by the OS. We achieve this by introducing two new events into the enclave lifecycle, *connect* and *disconnect*, which allow to track the liveliness of one enclave from another.

In summary, Figure 1.1 provides an overview of all the works included in this thesis.

1.1 THESIS CONTRIBUTION

The contributions of this thesis are divided into four parts as the following:

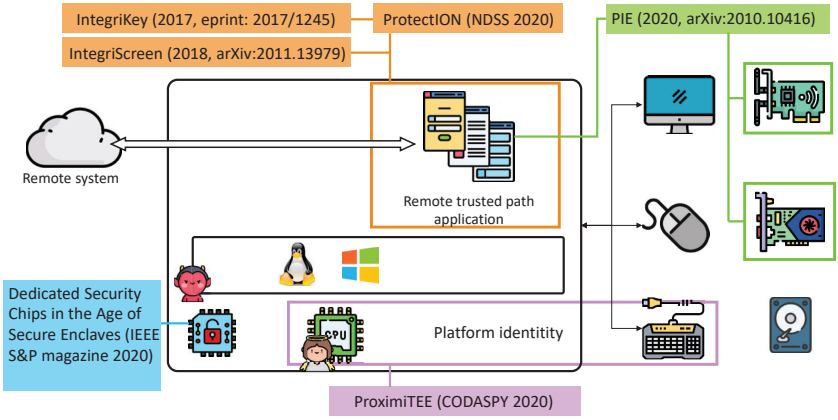

FIGURE 1.1: **Summary of the works in this thesis.** In this figure, we summarize the works that are in the thesis in a context of a modern computing platform. ① represents the works that are focused on securing the trusted path applications: IntegriKey, IntegriScreen, and ProtectION, ② represents ProximiTEE that brings additional security property to the physical communication link between the peripheral and the trusted path application running in an SGX enclave. Finally, ③ represents PIE that takes the platform as a whole: application enclave, driver enclave running on the CPU cores, and firmware running on the peripheral, and combine them in a single attestable domain - platform-wide enclave.

1. **IntegriKey, an ad-hoc approach to provide the second factor for integrity in user input through input signing**[1]

   a) *New attack.* In IntegriKey, we identify swapping attacks as a novel form of user input manipulation against simple user input matching strategies.

   b) IntegriKey. We design and implement a user input integrity protection system that is tailored for keyboard input, prevents swapping attacks, and is easy to deploy.

   c) IntegriKey tool. We develop a user interface analysis and webpage annotation tool that helps developers to protect their web

1 Note that this work presents negative results and shows how an input signing method is not secure. However, the paper also explores a new input manipulation attack - label swapping attack,

services and minimizes user effort. However, later analysis shows that the input signing is insecure. This is not due to the implementation of the input signing method, rather the fundamental pitfall of the method.

d) *Evaluation.* We verified that our tool could process UIs of existing safety-critical systems and cryptocurrency wallets correctly. Our experiments show that the performance delay of IntegriKey user input integrity protection is low. Our preliminary user study indicates that user input labeling prevents swapping attacks in most cases.

2. **IntegriScreen, second factor for user intention integrity through UI analysis captured through a smartphone.** [2]

   a) **System design.** We propose and describe IntegriScreen, a system that protects the integrity of the user's input to a remote server by using a device equipped with a camera to visually supervise the user's interaction with an untrusted client, thus preventing various advanced UI attacks that the adversary might attempt.

   b) **Prototype & experimental evaluation.** To evaluate the feasibility of the approach on recent smartphones, we build a fully functional prototype of the IntegriScreen system (IntegriScreen app and IntegriScreen server-side component) and test it with a few different devices against a range of automated attacks.

3. **ProtectIOn: Understanding trusted path and building secure trusted path systems**

   a) **Identification of new attacks:** We identify two new attacks on the existing trusted path systems. The first one is *input addition/reduction* attack on trusted path systems that uses input signing. IntegriKey is susceptible to this attack. Another attack is *early form submission* attack that targets target path systems that do not consider all modalities of input.

   b) **Identification of IO security requirements:** We identify new requirements for trusted path based on the drawbacks of the existing literature: i) unless both output and input integrity are

2 IntegriScreen was a collaboration between multiple researchers. This thesis only includes my primary contribution in IntegriScreen: the development of IntegriScreen's main idea, security analysis, and implementation of the IntegriScreens server-side component.

secured simultaneously, it is impossible to achieve any of the two, and ii) without protecting the integrity of all the modalities of inputs, none could be achieved.

c) **System for IO integrity:** We describe the design of ProtectIOn, a system that provides a remote trusted path from the server to the user in an attacker-controlled environment. The design of ProtectIOn leverages a small, low-TCB auxiliary device that acts as a *root-of-trust* for the IO. ProtectIOn ensures the integrity of the UI, specifically the integrity of mouse pointer and keyboard input. ProtectIOn is further designed to avoid user habituation.

d) **System for IO confidentiality:** We also describe an extension of ProtectIOn that provides IO confidentiality, where the user needs to execute an operation like a secure attention sequence (SAS) to identify the trusted overlay on display.

e) **Implementation and evaluation:** We also implement a prototype of ProtectIOn and evaluate its performance).

4. **ProximiTEE: Understanding Relay attacks on in Intel SGX remote attestation and building system for addressing the relay attacks in different attacker models (non-emulating vs. emulating attacker)**

a) **Analysis of relay attacks.** While relay attacks have been known for more than a decade, their implications have not been fully analyzed. We provide the first such analysis and show how relaying amplifies the adversary's capabilities for attacking SGX enclaves.

b) **ProximiTEE, a system for addressing relay attacks.** We propose a hardened SGX attestation mechanism based on an embedded device and proximity verification to prevent relay attacks. ProximiTEE does not rely on the common TOFU assumption, and hence, our solution improves the security of previous attestation approaches. Note that the distance bounding approaches are well-known in the literature, but using such a method in the context of SGX is non-trivial.

c) **Experimental evaluation.** We implement a complete prototype of ProximiTEE and evaluate it against a very strong and fast adversary. Our evaluation is the first to show that proximity verification can be both secure and reliable for TEEs like SGX.

d) **Addressing emulation attacks.** We also propose another attestation mechanism based on boot-time initialization to prevent emulation attacks. This mechanism is a novel variant of TOFU with deployment, security, and revocation benefits.

5. **PIE: Generalizing platform security in the context of TEEs and peripherals**

   a) **Security properties for platform-wide integrity** We extend traditional TEEs by introducing a dynamic hardware TCB. We call these new systems platform isolation environment (PIE). We identify key security properties for PIE, namely *platform-wide attestation* and *platform awareness*. template

   b) **Programming model** We propose a programming model that provides flexibility to the developers is comparable to the modern operating systems for developing enclaves that communicate with peripherals. The programming model abstracts the underlying hardware layer.

   c) **Prototype and experimental evaluation** We demonstrate a prototype of a PIE based on Keystone [27] on an open-source RISC-V processor [28]. The prototype includes a simplified model of the entire platform, including external peripherals emulated by multiple Arduino microcontrollers. In total, our modifications to the software TCB of Keystone only amount to around 350 LoC.

1.2 THESIS ORGANIZATION

The organization of the rest of this thesis is as the following. Chapter 2 and 3 provide details of the two ad-hoc trusted path systems based on two different second-factor methods:IntegriKey and IntegriScreen. IntegriKey is based on input signing, and IntegriScreen is based on the visual verification of user intent. In Chapter 4, we discussed ProtectIOn that explores fundamental security properties of the remote trusted path and shows attacks on existing trusted path solutions. Next, in Chapter 5, we discuss the challenges with relay attacks when we try to port an existing trusted path application to a TEE like Intel SGX, and we address this by introducing a new system based on distance-bounding mechanism: ProximiTEE. ProximiTEE shows how one can compromise a trusted path application without even compromising any security properties of the trusted path itself, but only by compromising underlying TEE properties

- remote attestation. Chapter 6 describes PIE, where we address the new challenges that arise from the disaggregated computing model where the trusted execution is not only bounded to the CPU cores, rather distributed among different hardware components of the platform such as a GPU, TPU, NIC, etc. Finally, Chapter 7 provides summary and concluding remarks.

1.3 PUBLICATION

Parts of this thesis are based on the following articles I have co-authored.

- **Aritra Dhar**, Enis Ulqinaku,Kari Kostiainen, and Srdjan Capkun, ProtectIOn: Root-of-Trust for IO in Compromised Platforms, *In Proceedings of the 27th Annual Network and Distributed System Security Symposium, NDSS 2020*.
- **Aritra Dhar**, Ivan Puddu, Kari Kostiainen, and Srdjan Capkun, ProximiTEE: Hardened SGX Attestation by Proximity Verification, *In Proceedings of the Tenth ACM Conference on Data and Application Security and Privacy (CODASPY '20)*. (🏆)
- Moritz Schneider*, **Aritra Dhar***, Ivan Puddu, Kari Kostiainen, and Srdjan Capkun, PIE: A Dynamic TCB for Remote Systems with a Platform Isolation Environment, *arXiv preprint /2010.10416 (2020)*.
- Kari Kostiainen, **Aritra Dhar**, and Srdjan Capkun, Dedicated Security Chips in the Age of Secure Enclaves, in *IEEE Security & Privacy, vol. 18, no. 5, pp. 38-46, Sept.-Oct. 2020*.
- Ivo Sluganovic, Enis Ulqinaku, **Aritra Dhar**, Daniele Lain, Srdjan Capkun, and Ivan Martinovic, IntegriScreen: Visually Supervising Remote User Interactions on Compromised Clients, *arXiv preprint 2011.13979 (2020)*.
- **Aritra Dhar**, Der-Yeuan Yu, Kari Kostiainen, and Srdjan Capkun, IntegriKey: End-to-End Integrity Protection of User Input, *IACR Cryptology ePrint Archive 2017: 1245 (2017)*.

In addition, during my Ph.D., I co-authored the following publications.

- Vasilios Mavroudis, Karl Wust, **Aritra Dhar**, and Kari Kostiainen, Srdjan Capkun:, Snappy: Fast On-chain Payments with Practical Collaterals, *In Proceedings of the 27th Annual Network and Distributed System Security Symposium, NDSS 2020*.

- David M. Sommer, **Aritra Dhar**, Luka Malisa, Esfandiar Mohammadi, Daniel Ronzani, and Srdjan Capkun, Deniable Upload and Download via Passive Participation, *In Proceedings of the 16th USENIX Symposium on Networked Systems Design and Implementation, NSDI 2019*.

- Garvita Allabadi, **Aritra Dhar**, Ambreen Bashir, Rahul Purandare, METIS: Resource and Context-Aware Monitoring of Finite State Properties, *In Proceedings of International Conference on Runtime Verification, RV 2018*. (🏆)

- Sinisa Matetic, Mansoor Ahmed, Kari Kostiainen, **Aritra Dhar**, David M. Sommer, Arthur Gervais, Ari Juels, and Srdjan Capkun, ROTE: Rollback Protection for Trusted Execution, *In Proceedings of the USENIX Security Symposium 2017*.