1 INTRODUCTION

Software applications’ security heavily depends on the security of the underlying system software. In traditional computing environments, if the operating system is compromised, the security of the applications it supports is also compromised. Therefore, the trusted computing base (TCB) of software applications include not only the software itself but also the underlying system software and hardware.

To reduce the TCB of some applications that contain sensitive code and data, academic researchers have proposed many software systems to support shielded execution—i.e., execution of a piece of code whose confidentiality and integrity is protected from an untrusted system software (e.g., [26, 28, 30, 33, 37, 44, 45, 49, 56, 68, 74, 77]). Most of these systems adopted a hypervisor-based approach to protecting the memory of victim applications against attacks from malicious operating systems. Although promising, these academic prototypes have yet to see the light of real-world adoption. Not until the advent of Intel Software Guard eXtension (SGX) [2], a hardware extension available in the most recent Intel processors, did the concept of shielded execution become practical to real-world applications. SGX enforces both confidentiality and integrity of userspace programs by isolating regions of their memory space (i.e., enclaves) from other software components, including the most privileged system software–no memory reads or writes can be performed inside the enclaves by any external software, regardless of its privilege level. As such, SGX greatly reduces the TCB of the shielded execution, enabling a wide range of applications [20, 36, 54, 59, 65, 76].

In typical application scenarios [20, 36, 76], shielded execution does not work completely alone; it communicates with remote trusted parties using secure channels, e.g., SSL/TLS protocols. Secure Sockets Layer (SSL) and its successor, Transport Layer Security (TLS), are transport-layer security protocols that provide secure communication channels using a set of cryptographic primitives. SSL/TLS protocols are expected, as part of their design goals, to prevent man-in-the-middle attackers who are capable of eavesdropping, intercepting, replaying, modifying and injecting network packets between the two communicating parties. Therefore, applications of Intel SGX [20, 36, 76] typically regard SSL/TLS modules inside SGX enclaves as basic security primitives to establish end-to-end communication security.

Attacks against the SSL/TLS protocol have been reported over the years, unfortunately. One important category of these attacks is oracle attacks [31]. In an oracle attack, the adversary interactively and adaptively queries a vulnerable SSL/TLS implementation and uses the response (or some side-channel information, e.g., the latency of the response) as an oracle to break the encryption. Well-known examples of oracle attacks include the Lucky Thirteen Attack [12], the Bleichenbacher attack [22], the DROWN attack [18], the POODLE attack [52], etc. Prior demonstration of these attacks have shown that they enable network attackers to decrypt arbitrary messages of the SSL record protocol or decrypt the PreMasterSecret of the SSL handshake protocol. We will detail these attacks in Section 2. Due to the broad adoption of the SSL/TLS protocol (e.g., in HTTPS, secure email exchanges), any of these attacks is devastating and easily headlines of the security news (e.g., [41]). Accordingly, the SSL/TLS protocol and its implementations have been frequently updated after the publicity of these attacks. A commonly used solution is to hide the oracles. For example, in cases where the oracle is the SSL Alert message indicating padding errors, the error message can be unified to conceal the real reason for the errors [35, 58] (so that the adversary cannot differentiate padding errors and MAC errors, see Section 2). As of today, almost all widely used SSL/TLS implementations are resilient to oracle attacks because the oracles have been successfully hidden from the network attackers [4, 10, 35, 58].

However, adoption of SSL/TLS in SGX enclaves brings new security challenges. Although SGX offers confidentiality protection, through memory isolation and encryption, to code and data inside secure enclaves, it has been shown vulnerable to side-channel attacks [43, 63, 73]. Side-channel attacks are a type of security attacks against the confidentiality of a system or application by making inferences from measurements of observable side-channel events. These attacks have been studied in the past twenty years in multiple contexts, most noticeably in desktop computers, cloud servers, and mobile devices where CPU micro-architectures [78, 79], software data structures [40, 57], or other system resources are shared between mutually-distrusting software components. What makes side-channel attacks on SGX different is that these attacks can be performed by the privileged system software, which enables many new attack vectors. For example, Xu et al. [73] demonstrated that by manipulating page table entries of the memory pages of secure enclaves, an adversary with system privilege could enforce page faults during the execution of enclave programs, thus collecting traces of memory accesses at the page-granularity. Recently, Lee et al. [43] demonstrated that the control flow of enclave programs can be precisely traced at every branch instruction by exploiting the shared Branch Prediction Units (BPU).

The key insight of this paper is that while SSL/TLS is designed to defend against man-in-the-middle attacks, its implementation in SGX enclaves must tackle a stronger man-in-the-kernel adversary who is capable of not only positioning himself in the middle of the two communicating parties, but controlling the underlying operating system kernel and manipulating system resources to collect execution traces of the enclave programs from various side channels. Particularly, we show that the powerful man-in-the-kernel attackers can create new decryption oracles from the state-of-theart SSL/TLS implementations and resurrect the Bleichenbacher attack and CBC padding oracle attacks against SGX enclaves.

Stacco. At the core of our work is the Side-channel Trace Analyzer for finding Chosen-Ciphertext Oracles (Stacco), which is a software framework for conducting differential analysis on the SSL/TLS implementations to detect sensitive control-flow vulnerabilities that can be exploited to create decryption oracles for CBC padding oracle attacks and Bleichenbacher attacks. Particularly, to enable automated large-scale analysis of various off-the-shelf SSL/TLS libraries, we built Stacco on top of a dynamic instrumentation engine (i.e., Pin [46]) and an open-source SSL/TLS packet generation tool (i.e., TLS-Attacker [66]), so that we can perform standard tests to multiple libraries in an automated manner. To understand the exploitability of the vulnerabilities, we also modeled three types of control-flow inference attacks, including page-level attacks [63, 73], cacheline-level attacks [23, 60] and branch-level attacks [43], and empowered Stacco to analyze vulnerabilities on each of these levels. Our analysis results suggest all the popular open-source SSL/TLS libraries we have examined are vulnerable to both types of oracle attacks, raising the questions of secure development and deployment of SSL/TLS protocols inside SGX enclaves.

To validate the vulnerabilities identified by Stacco, we demonstrated several such man-in-the-kernel attacks against the latest versions of popular cryptographic libraries: Particularly, we implemented a Bleichenbarcher attack against the latest OpenSSL library [9] running in the SGX enclaves (with the help of Graphene-SGX [70], a library OS that supports unmodified applications to run inside SGX enclaves) and completely broke the PreMasterSecret encrypted by a 4096-bit RSA public key with only 57,286 queries. We also conducted CBC padding oracle attacks against the latest GnuTLS [3] running in Graphene-SGX and an open-source SGX-implementation of mbedTLS [8] that runs directly inside the enclave, and showed that it only needs 48,388 and 25,717 queries, respectively, to break one block of AES ciphertext from TLS connections using these libraries. Empirical evaluation suggests these man-in-the-kernel attacks can be completed within one or two hours. These demonstrated attacks not only provide evidence that Stacco can effectively identify exploitable sensitive control-flow vulnerabilities in SSL/TLS implementations, but also suggest these oracle attacks conducted in a man-in-the-kernel manner are efficient for practical security intrusion.

Responsible disclosure.We have reported the vulnerabilities and demonstrated oracle attacks to Intel, OpenSSL, GnuTLS, mbedTLS. Contributions of this work include:

• The first study of critical side-channel threats against SSL/TLS implementations in SGX enclaves that lead to complete compromises of SSL/TLS-protected secure communications.

• The design and implementation of Stacco, a differential analysis framework for detecting sensitive control-flow vulnerabilities in SSL/TLS implementations, which also entails:

• A systematic characterization of control-flow inference attacks against SGX enclaves (e.g., page-level attacks, the cachelinelevel attacks, and branch-level attacks), which empowers Stacco to analyze the vulnerability with abstracted attacker models.

• A measurement study of the latest versions of popular SSL/TLS libraries using Stacco that shows that all of them, including OpenSSL, GnuTLS, mbedTLS, WolfSSL, and LibreSSL, are vulnerable to control-flow inference attacks and exploitable in oracle attacks.

• An empirical man-in-the-kernel demonstration of oracle attacks against the latest version of OpenSSL and GnuTLS running inside Graphene-SGX and an open-source SGX-implementation of mbedTLS, showing that such attacks are highly efficient on real SGX hardware.

Roadmap. The rest of this paper is outlined as follows. Section 2 introduces related background concepts. Section 3 systematically characterizes control-flow inference attacks. Section 4 describes a differential analysis framework for detecting sensitive control-flow vulnerabilities in SSL/TLS implementations.We demonstrate oracle attacks against some of the vulnerable SSL/TLS implementations to validate these detected vulnerabilities in Section 5, and then discuss countermeasures in Section 6. In Section 7, we briefly summarize related work in the field. Section 8 concludes our paper.