CUSTOS: Practical Tamper-Evident Auditing of Operating Systems Using Trusted Execution

Riccardo Paccagnella*, Pubali Datta*, Wajih Ul Hassan*,
Adam Bates*, Christopher W. Fletcher*, Andrew Miller*, Dave Tian†
*University of Illinois at Urbana-Champaign:
{rp8, pdatta2, whassan3, batesa, cwfletch, soc1024}@illinois.edu
†Purdue University: daveti@purdue.edu

***Abstract*—System auditing is a central concern when investigating and responding to security incidents. Unfortunately, attackers regularly engage in anti-forensic activities after a break-in, covering their tracks from the system logs in order to frustrate the efforts of investigators. While a variety of tamper-evident logging solutions have appeared throughout the industry and the literature, these techniques do not meet the operational and scalability requirements of system-layer audit frameworks.**

In this work, we introduce CUSTOS, a practical framework for the detection of tampering in system logs. CUSTOS consists of a tamper-evident logging layer and a decentralized auditing protocol. The former enables the verification of log integrity with minimal changes to the underlying logging framework, while the latter enables near real-time detection of log integrity violations within an enterprise-class network. CUSTOS is made practical by the observation that we can decouple the costs of cryptographic log commitments from the act of creating and storing log events, without trading off security, leveraging features of off-the-shelf trusted execution environments. Supporting over one million events per second, we show that CUSTOS' tamper-evident logging protocol is three orders of magnitude (1000×) faster than prior solutions and incurs only between 2% and 7% runtime overhead over insecure logging on intensive workloads. Further, we show that CUSTOS' auditing protocol can detect violations in near real-time even in the presence of a powerful distributed adversary and with minimal (3%) network overhead. Our case study on a real-world APT attack scenario demonstrates that CUSTOS forces anti-forensic attackers into a "lose-lose" situation, where they can either be covert and not tamper with logs (which can be used for forensics), or erase logs but then be detected by CUSTOS.

I. INTRODUCTION

Auditing is an essential component of building and maintaining secure systems. When suspicious events occur, system logs are frequently turned to as the definitive ground truth of the system's activities. Such logs have been leveraged to address a variety of security challenges, from post-mortem attack reconstruction [7], [65], [68], [76], [99] to runtime tasks such as access control [4], [6], [88], [95], execution integrity [107], [118], and intrusion detection [23], [43], [62], [64], [83]. The integrity of logs is thus a vital consideration in system security.

Network and Distributed Systems Security (NDSS) Symposium 2020
23-26 February 2020, San Diego, CA, USA
ISBN 1-891562-61-4
https://dx.doi.org/10.14722/ndss.2020.24065
www.ndss-symposium.org

Unfortunately, intruders also understand the value of system logs. Because such logs describe an attacker's method of entry, mission objectives, and further propagation within a system, attackers regularly engage in *anti-forensic* countermeasures to erase or conceal this vital forensic evidence [121], [90], [56], [9]. Penetration testing tools such as Last Door [37] and Metasploit [104] go so far as to automate this process, allowing an intruder to escalate privilege and wipe the associated logs with a single command. Perhaps worse than log removal, attackers may also edit existing events or insert new ones to confuse investigators [38], [21]. Concerningly, recent reports suggest that 72% of incident response specialists have encountered tampered logs in the course of an investigation [16], [17].

In light of this reality, it is perhaps surprising that commodity operating systems offer no special protections for their logging frameworks. Case in point, root access is often sufficient for the covert manipulation of any and all forensic evidence. While fully preventing this tampering might be impossible, detecting it should be of prime importance. Existing commercial solutions to this *tamper-evident* logging problem involve specialized Write-Once-Read-Many (WORM) storage devices [47], [85] or fully-trusted remote storage servers [58], [94]. Concurrent to commercial efforts, a variety of cryptographic solutions have been presented in the literature, spanning symmetric cryptography [10], [108], [44], digital signatures [73], [127], [41], and tamper-evident data structures [20], [101].

However, while these approaches show promise for constrained logging scenarios (e.g., securing a single application's log), they present limitations that make them ill-suited to the demands of system logging. Existing commercial solutions, while considered good practice in the industry [32], rely on either costly and exotic hardware devices or continuous, timely communication with remote trusted servers. Proposed cryptographic techniques incur excessive computational and storage overheads and do not account for practical issues such as continuity across power cycles or compatibility with upstream log analysis applications (e.g., [115]). Critically, all prior solutions suffer from issues of throughput, meaning that they are too slow to be used in practice on modern operating systems, which can generate hundreds of thousands of system calls per second [24]. For these reasons, we argue that prior solutions do not meet the practical requirements of operating systems. *For tamper-evident logging to be viable within operating systems, it must be scalable, efficient, and minimally invasive to the existing audit frameworks.*

In this work, we revisit the goal of tamper-evident logging within the context of standard operating system abstractions. We introduce CUSTOS,[1] a practical solution for the detection of tampering in system logs. CUSTOS scales to high volume logging scenarios, avoids invasive modifications to commodity audit frameworks, and is provably secure under a strong attacker model. CUSTOS is made up of the following components:

1) **Tamper-Evident Logger**: To enable the verification of log integrity, we present a minimally invasive tamper-evident logging layer for commodity audit frameworks. Our protocol satisfies the above requirements by decoupling cryptographic event commitment from logging—leveraging features of off-the-shelf Trusted Execution Environments (TEEs). Upon creation, log events are hashed inside the enclave, and the enclave asynchronously signs these hashes in response to periodic audits. We show that, without trading off security, this technique allows CUSTOS to secure up to 1,086,956 events per second, as compared to at most 1,266 events per second on the same hardware in prior work [57]. As such, CUSTOS' tamper-evident logging protocol is *three orders of magnitude (i.e., $1000\times$) faster than prior TEE-based work*. Furthermore, our logger supports third-party verifiability of log integrity, and, unlike prior work [57], [89], does not break compatibility with log analysis applications by avoiding reliance on log encryption.
2) **Real-time Decentralized Auditing**: Differently from many prior solutions (e.g., [73], [57], [41], [128], [44]), we also focus on how to discover log tampering through auditing. Intuitively, the more frequently logs are audited, the earlier anti-forensic attackers will be detected after intrusion: thus, there is a clear motivation for auditing to be a frequent operation. To this end, we introduce a decentralized auditing scheme that enables near real-time detection of log integrity violations. Our scheme employs a three-way network protocol between CUSTOS-enabled hosts and detects log tampering with very high probability even in the presence of distributed adversaries. Decentralized audits further enable CUSTOS to provide assurance of log availability for historic, integrity-verified log data. Most importantly, decentralized audits force anti-forensic attackers into a "lose-lose" situation, where they can either be covert and not tamper with logs (which can be used for forensics), or they can erase logs but then be detected by CUSTOS.

In summary, this paper makes the following contributions:

- We design CUSTOS, a comprehensive and practical solution for the detection of tampering in system logs, made up of a tamper-evident logging layer and a decentralized auditing protocol. CUSTOS minimizes the cost of securely logging an event down to a single hash update by decoupling cryptographic event commitment from logging—leveraging features of off-the-shelf TEEs. We include security analyses of our protocols to demonstrate their correctness in the presence of anti-forensic adversaries.
- To enable further experimentation and proliferation of tamper-evident logging mechanisms, we implement a prototype version of CUSTOS for the Linux Audit system that uses Intel SGX as a TEE. Our prototype is available as open source at https://bitbucket.org/sts-lab/custos/.
- We rigorously evaluate the performance and effectiveness of our system in a network of 100 hosts. Our results demonstrate that CUSTOS can secure log events $1000\times$ faster than prior work and imposes only 2% to 7% runtime and 3% network overheads over insecure logging. Our case study shows that CUSTOS enables the detection of log tampering in real-world APT attack scenarios.

[1] Custos is the Latin word for guard. It was used by the Roman poet Juvenal in the phrase "Quis custodiet ipsos custodes?" (Satire VI, lines 347–348), translated as "Who watches the watchers?".

II. BACKGROUND

1) System Logs: This paper concerns itself with detecting tampering with *system logs*. System logs, sometimes referred to as *audit logs*, are a set of records that provide documentary evidence of the sequence of activities that have affected an operating system (OS). These records contain information that serves to establish what type of event occurred, when and where it occurred, its source and outcome and the subjects associated with it [85]. System logs differ from application logs because they are not generated by the application code at the developer's will, but by the OS regardless of the application's code. As such, they are able to capture the most primitive system-level events (i.e., system calls), which include records of security-relevant events such as execution of malicious binaries and failed login attempts. Moreover, the volume of system logs is generally very large compared to application logs, since modern OSs can generate hundreds of thousands of system calls per second [24]. Because of their forensic value, system logs are critical for postmortem analyses after a break-in. As a consequence, recording system logs is a legal requirement for a number of security-related certifications (e.g., [105], [51]): in Linux, the Audit Subsystem (LAuS) [117] was introduced in version 2.6 to achieve certification under the Controlled Access Protection Profile (CAPP) common criteria [86]. System auditing will continue to grow in importance with the enactment of the European Union's GDPR [25].

2) Intel SGX: Intel Software Guard Extensions (SGX) are a set of extensions to the x86 instruction set architecture that allows for the creation of isolated execution environments called enclaves. Once an enclave has been initialized, the processor ensures that any system component outside the enclave, including the privileged software, cannot access the enclave's protected memory where its code and data reside. Untrusted applications can, however, switch into enclave mode at pre-defined entry points and execute protected instructions inside the trusted enclave. For an untrusted application to start executing trusted code inside the enclave, it needs to invoke an EENTER instruction, which performs a sequence of steps (e.g., load secure register context) before transitioning to enclave mode. Intel provides wrapper code, called *ecalls*, to prepare the environment for an EENTER instruction [49]. Current SGX hardware provides 128 MB of protected enclave memory per host. However, SGX also offers a sealing feature, which allows an enclave to persist and retrieve data on the host's unprotected memory, even after the enclave is destroyed and restarted. Sealed data is confidentiality and integrity-protected, but sealing does not provide freshness guarantees and is vulnerable to rollback attacks. To address rollback attacks, SGX supports monotonic counters that use non-volatile memory to maintain state across different sessions. However, usage of monotonic counters is limited by both their quota (256 per enclave) and

their update rate (excessive use can cause memory to wear out). Note that, while our implementation utilizes Intel SGX, our design is applicable to any enclave-like interface [116], which can alternatively be provided by other TEEs such as ARM Trustzone and Keystone (see [14], [26], [19], [60]). However, as we will demonstrate, our performance requirements rule out off-chip solutions such as the Trusted Platform Module (TPM), which is notoriously slow [80], [102].

III. Threat Model and Goals

1) Threat Model: This work envisions a large organizational environment, comprised of upwards of thousands of machines, that is the target of a sophisticated and well-funded adversary. The adversary's attack pattern follows the APT lifecycle model [83]: after an initial compromise grants unprivileged access to a host, the attacker establishes persistence and then escalates privilege in order to achieve *full system compromise*, at which point they have full control of the operating system, and can engage in anti-forensic measures (e.g., log tampering [104], [37], [90], [121], [77]) in order to hide their presence.

We assume that each host in the network is equipped with a TEE such as Intel SGX that can confidentially store cryptographic keys. We also assume that the implementations of the cryptographic functions used inside the TEE are side-channel free, meaning that they do not exhibit secret-dependent memory accesses that can leak the signing key.[2] We make the usual assumption that it is not feasible for an adversary to forge digital signatures or find collisions in cryptographic hash functions. Finally, we assume that the organization employs a system administrator or cyber analyst that maintains a key management service (KMS) and can receive and respond to security alerts.

2) Design Goals: With the adversary model and the assumptions described above in mind, we set out to design a system that satisfies the following properties:

G1 Tamper-Evident Logs. The auditing system must record log entries with provable integrity such that forgeries, omissions, and other forms of tampering can be detected. That is, after achieving full system compromise, an adversary should not be able to undetectably manipulate log messages recorded *pre-compromise*. This goal is consistent with prior work (cf. Section XIII).

G2 Third-Party Verifiability. Log verifiability should not depend on a single machine or fully trusted verifier. In forensic investigations, third parties (e.g. Court agents [52], [53], [33]) should be able to verify the correctness and authenticity of a given set of logs without being granted other privileges in the system (e.g., access to secret keys).

G3 Fine-Grained Audits. The system must support verification of subsets of log entries without the need to possess the entire log history. This permits audits over a specific time span of interest. Further, this property aids attack reconstruction by attributing integrity violations to a specific time span in the log history.

G4 Log Availability. In addition to detecting violations of log integrity, the system should provide assurance of log availability. Specifically, the system should allow to preserve and retrieve copies of historic, integrity-verified log data describing pre-compromise events.

G5 Minimal Invasiveness. The system must interoperate with commodity audit frameworks, avoid changes to the underlying OS, work efficiently enough to be deployed on systems under heavy load, and preserve compatibility with upstream log analysis applications (e.g., [115]).

[2] Custos' security in the context of micro-architectural side channels reduces to software in the TEE being able to protect secret keys during cryptographic routines (e.g., when calculating digital signatures), which is a well-studied and orthogonal problem [122], [103], [18], [12], [129], [125].

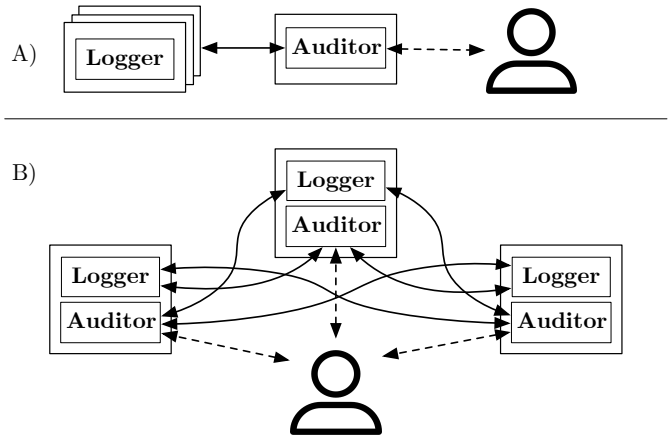

Fig. 1: Overview of Custos' components. Each rectangle represents a host. Hosts record logs using Custos' Logger (Section V). A shows a centralized auditing scenario (Section VI), where one central server (Auditor) audits the logs of other hosts and reports violations to the system administrator. B shows a decentralized auditing scenario (Section VII) where logging hosts also run Custos' Auditor component and audit each other in a peer-to-peer fashion.

3) Design Challenges: Providing the above properties is challenging given our threat model. Let us consider existing schemes that satisfy **G2** (third-party verifiability). These schemes fall short of achieving our other goals by trading off performance (violating **G5**) or security (violating **G1**).

For example, existing forward-secure signature schemes relying on the notion of $epoch$ (e.g., [73], [44], [41]) fully achieve **G1** only when configured to generate signatures at the granularity of one log event ($epoch = 1$), but generating a new key and/or signature per log event incurs impractically large computational costs (violating **G5**). On the other hand, when $epoch = n$ the most recent pre-compromise events remain vulnerable. That is, if the attacker achieved full system compromise before the n-th event of the current epoch occurred, then they would obtain the current key and be able to forge integrity proofs for all pre-compromise events belonging to the current epoch. Such a configuration would thus fail to provide **G1**. Moreover, schemes based on forward-secure sequential aggregate signatures (e.g., [73], [128]) fail to provide **G3** (fine-grained audits) in that they require the entire log history for verification.

While the aid of trusted hardware (as in [57], [89], [111]) can help overcome these issues (e.g., by protecting the secrecy of the current key after full system compromise), it is not a panacea. Interacting with a TEE still requires addressing attack vectors such as rollback attacks and protocol termination

TABLE I: Summary of notation and semantics for CUSTOS' protocols.

Notation	Description
$\mathcal{H}$	An incremental cryptographic hash-function with methods Init, Update, Final. Init() initializes and returns a hash context. Update(hash, data) updates the hash context hash with data. Final(hash) generates the hash digest h(hash).
Σ	A digital signature scheme with methods KeyGen, Sign, Verify. KeyGen() generates an asymmetric key pair $\langle \mathsf{sk}, \mathsf{pk} \rangle$, with sk private key and pk public key. Sign(sk, m) generates $\sigma_{\mathsf{sk}}(\mathsf{m})$ by signing message m with sk. Verify(pk, m, $\sigma_{\mathsf{sk}}(\mathsf{m})$) returns true if $\sigma_{\mathsf{sk}}(\mathsf{m})$ is a valid signature over m made with sk, false otherwise.
$\mathsf{m}_1 \| \mathsf{m}_2$	Concatenation of m_1 and m_2.
$\langle \mathsf{sk}, \mathsf{pk} \rangle$	Uniquely-identifying keypair for CUSTOS instance. Generated with Σ.
M_e	Ordered set of non-overlapping consecutive log entries belonging to a block with unique ID e.
Seal(in, out)	TEE function. Seals the given input data in into the encrypted output data out .
Unseal(in, out)	TEE function. Attempts to unseal the given encrypted input data in into the output data out. Returns -1 if the unsealing fails (e.g. the sealed data was tampered with).
CreateMC()	TEE function. Initializes a monotonic counter with ID mcID and value $\mathsf{mc} = 0$. Returns the tuple $\langle \mathsf{mcID}, \mathsf{mc} \rangle$.
IncrementMC(mcID)	TEE function. Increments by one the value of the monotonic counter with ID mcID.
ReadMC(mcID)	TEE function. Reads and returns the value of the monotonic counter with ID mcID.
DestroyMC(mcID)	TEE function. Destroys the monotonic counter with ID mcID.

attacks, and accounting for these issues often leads to large overheads [46], [78], [80]. However, as hosts can generate hundreds of thousands of system calls per second [24], satisfying **G5** necessitates low overheads. In addition to performance challenges, TEEs also pose reliability challenges. For example, hardware monotonic counters (updated once every 50 log events in [57]) are known to become unusable after approximately one million uses due to the wear-out of non-volatile memory [78]. Another limitation of TEEs is the limited memory available to enclaves (e.g., 128 MB in SGX), making them unusable to store large amounts of log messages.

IV. System Overview

In light of the inability of prior work to meet our goals, we now present the design of CUSTOS, an efficient *and* secure solution for the tamper-evident logging problem. CUSTOS enables the detection of tampering in system logs through the introduction of the components described below. A visual diagram of these components is provided in Figure 1.

1) Tamper-Evident Logger (Section V): As log events are generated by the operating system, they are processed by an underlying audit framework[3] that is minimally-modified to efficiently generate proofs of log integrity. Upon creation, log events are hashed inside the enclave, and the enclave asynchronously signs these hashes in response to periodic audits. The code and data (keys) responsible for producing these signatures are partitioned from the rest of the framework and executed within a trusted enclave. For proof verification, the Logger publishes its public key, which is bound to the identity of the enclave.

[3]In this work, underlying audit framework refers to the software which receives and records the log events generated by the OS.

2) Centralized Auditing (Section VI): Log tampering is discovered through auditing. To this end, we introduce two auditing protocols for CUSTOS. Our first auditing protocol is designed to support the prevailing common practice for logging in large organizations, which is is to transmit log events to a central storage server. The central server (Auditor) obtains logs and proofs from a Logger host by issuing an *audit challenge* to it. The Logger sends a *response* to the audit challenge that includes the logs and their associated integrity proofs. The Auditor uses these proofs to check the integrity of the logs included in the response, sending a security alert to the administrator if the audit fails.

3) Decentralized Auditing (Section VII): A centralized log server creates a single point of failure, which could conceivably be targeted by attackers after perimeter defenses are breached. Thus, centralized auditing jeopardizes **G4** and also creates scalability issues. To address these limitations, we present a decentralized variant of our audit protocol. Here, all network nodes include an Auditor component running inside the enclave of the host. The Auditor randomly initiates audit challenges with a parameterizable number of its peers over a specified time period. Our protocols further support secure log replication with parameterizable redundancy and include a parallelized log reconstruction algorithm.

V. Tamper-Evident Logger

We now describe CUSTOS' tamper-evident logging protocol in its five routines: (1) Initialization; (2) Startup; (3) Logging; (4) Commitment; and (5) Shutdown. Each of these routines corresponds to a call to the Logger application running inside the trusted enclave. Table I explains the notation and TEE functions we will use in our protocol descriptions.

Algorithm 1: Initialization Phase

Output: sealed-key-id, sealed-e

$\langle \mathsf{sk}, \mathsf{pk} \rangle \leftarrow \Sigma.\mathsf{KeyGen}()$; // pk is published
$\langle \mathsf{mcID}, \mathsf{mc} \rangle \leftarrow \mathsf{CreateMC}()$;
$\mathsf{e} \leftarrow 0$;
$\mathsf{Seal}(\langle \mathsf{sk}, \mathsf{mcID} \rangle, \text{sealed-key-id})$;
$\mathsf{Seal}(\langle \mathsf{e} \| \mathsf{mc} \| \mathsf{mcID} \rangle, \text{sealed-e})$;

Algorithm 2: Startup Phase

Input: sealed-key-id, sealed-e

$\mathsf{ret}_1 \leftarrow \mathsf{Unseal}(\text{sealed-key-id}, \langle \mathsf{sk}, \mathsf{mcID} \rangle)$;
if $\mathsf{ret}_1 == -1$ **then**
  raise an error;
$\mathsf{mc} \leftarrow \mathsf{ReadMC}(\mathsf{mcID})$;
$\mathsf{ret}_2 \leftarrow \mathsf{Unseal}(\text{sealed-e}, \langle \mathsf{e} \| \mathsf{mc}_e \| \mathsf{mcID}_e \rangle)$;
if $\mathsf{ret}_2 == -1 \vee \mathsf{mc}_e \neq \mathsf{mc} \vee \mathsf{mcID}_e \neq \mathsf{mcID}$ **then**
  raise an error;
$\mathsf{IncrementMC}(\mathsf{mcID})$;
$\mathsf{hash} \leftarrow \mathcal{H}.\mathsf{Init}()$;
$\mathsf{e} \leftarrow \mathsf{e} + 1$;
} $\mathsf{StartNewBlock}()$

1) Initialization phase: Algorithm 1 shows the initialization phase, used when CUSTOS is deployed on the host. This phase starts with creating an asymmetric key pair $\langle \mathsf{sk}, \mathsf{pk} \rangle$ for the Logger. Next, the TEE is used to initialize a new monotonic counter with UUID and value $\langle \mathsf{mcID}, \mathsf{mc} \rangle$, respectively, and a block ID variable (e) is initialized to zero, which will be the index of the first block of logs signed with sk. The TEE is then used to seal these values so that they can be stored on disk; they will be unsealed by the Logger during its startup phase. pk and sealed-key-id are also copied onto an administrative machine for key management and recovery purposes.

2) Startup phase: Algorithm 2 describes the startup phase, which is invoked once per application startup. This phase starts with unsealing the previously sealed key, monotonic counter ID, and block ID, and ensuring the freshness of the block ID (by checking that the mc attached to sealed-e is up-to-date).[4] The Logger then increments the value of the monotonic counter to mark the beginning of a new session. Finally, a new block is started and an incremental hash is initialized. The Logger maintains the key, monotonic counter, block ID, and incremental hash in enclave-protected memory throughout its execution until the shutdown phase is invoked. Once this phase is complete, the Logger is ready to receive log events from the underlying audit framework.

3) Logging phase: Algorithm 3 shows the logging phase, which is invoked each time the audit framework produces a new log event. As moderately loaded hosts can produce hundreds of thousands of system calls per second [24], performance is paramount in this routine. CUSTOS judiciously minimizes the cost of this phase by having it use a single, efficient operation: updating the current block's hash value with the new log event.

[4]In case of corrupted or stale data, the Logger will raise an error and will need to be re-initialized. We discuss how CUSTOS supports recovery from errors in Appendix A.

Algorithm 3: Logging Phase

Input: A log message m

$\mathcal{H}.\mathsf{Update}(\mathsf{hash}, \mathsf{m})$; // update the hash

Algorithm 4: Commitment Phase

Output: e, $\sigma_{\mathsf{sk}}(\mathsf{hash}_{\mathsf{M_e}})$

$\mathcal{H}.\mathsf{Update}(\mathsf{hash}, \mathsf{e})$;
$\mathsf{hash}_{\mathsf{M_e}} \leftarrow \mathcal{H}.\mathsf{Final}(\mathsf{hash})$;
$\sigma_{\mathsf{sk}}(\mathsf{hash}_{\mathsf{M_e}}) \leftarrow \Sigma.\mathsf{Sign}(\mathsf{sk}, \mathsf{hash}_{\mathsf{M_e}})$
} $\mathsf{CompleteBlock}()$
$\mathsf{StartNewBlock}()$;

Algorithm 5: Shutdown Phase

Output: sealed-e, e, $\sigma_{\mathsf{sk}}(\mathsf{hash}_{\mathsf{M_e}})$

$\mathsf{CompleteBlock}()$;
$\mathsf{Seal}(\langle \mathsf{e} \| \mathsf{mc} \| \mathsf{mcID} \rangle, \text{sealed-e})$;

4) Commitment phase: The commitment phase, shown in Algorithm 4, is invoked when the Logger receives an audit challenge for the current block: this process will become clearer when we introduce our audit protocol in Section VI. During this phase, the Logger signs the incremental hash that has been generated over the current block of logs (with block ID e). Finally, a new block is started. The output signature can then be stored together with the logs and used for auditing purposes. Observe that the size of each block thus depends on the frequency of the audits.

5) Shutdown phase: Algorithm 5 describes the Shutdown phase. This phase is invoked when the Logger application is terminated. In this phase, the Logger must complete the current block (regardless of whether an audit challenge was received) and seal the current block ID e together with the current value and ID of the monotonic counter. This phase ensures that (1) all log entries up to the moment of shutdown are successfully signed and (2) when the Logger is started up again it can continue with block ID $\mathsf{e} + 1$.

VI. CENTRALIZED AUDITING

In large organizations, it is common practice to periodically transmit system logs to a central storage server. Once the logs are extracted from the host, they can be analyzed by Security Information and Event Management (SIEM) products or retained for forensic analysis. In this section, we demonstrate how CUSTOS can be suitably incorporated into this workflow through the introduction of an *audit* protocol, which serves to ensure that logs were not manipulated between capture and transmission to the central server. An overview of this protocol can be found in Figure 2. The procedure works as follows:

1) The central storage server (Auditor) initiates an audit by sending a signed *audit challenge* to the Logger. An audit challenge message includes a challenge ID clD, which is a nonce, and an interval of log blocks defined by its extremities b_1 and b_2 (with $\mathsf{b}_1 \leq \mathsf{b}_2$). If b_2 is unspecified,

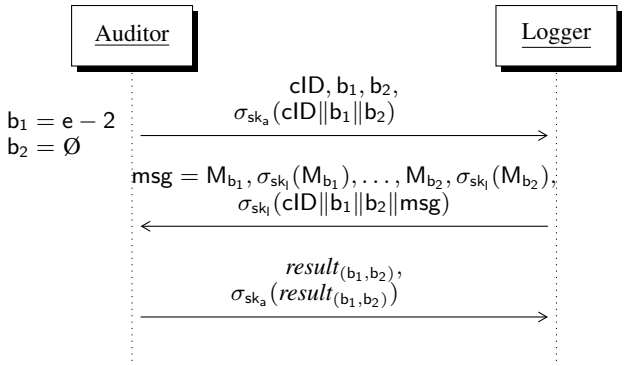

Fig. 2: Protocol summary of an audit. The Auditor initiates an audit challenge between blocks with IDs b_1 and b_2. The Logger transmits the inclusive range of log events between the IDs with their associated integrity proofs. The Auditor notifies the Logger of the result after storing and verifying the logs. $\langle \mathsf{sk_a}, \mathsf{pk_a} \rangle$ denotes the key pair of the Auditor, and $\langle \mathsf{sk_l}, \mathsf{pk_l} \rangle$ denotes the key pair of the Logger.

the audit challenge represents a request for the Logger to commit and transmit its current log block.[5]

2) The audit challenge is received by the untrusted component of the Logger, which passes it to the enclave. The enclave verifies the challenge's authenticity (by ensuring that its signature is valid) and freshness (by ensuring that cID has not been used before). If $\mathsf{b_2}$ was unspecified in the challenge, the enclave commits the current log block $\mathsf{M_e}$ and sets $\mathsf{b_2} = \mathsf{e}$. Finally, it receives from the untrusted component a copy of the requested blocks $\mathsf{M_{b_1}}, \ldots, \mathsf{M_{b_2}}$ and their associated proofs $\sigma_{\mathsf{sk}}(\mathsf{M_{b_1}}), \ldots, \sigma_{\mathsf{sk}}(\mathsf{M_{b_2}})$, and produces a signature $\sigma_{\mathsf{sk}}(\mathsf{cID} \| \mathsf{b_1} \| \mathsf{b_2} \| \mathsf{M_{b_1}} \| \sigma_{\mathsf{sk}}(\mathsf{M_{b_1}}) \| \ldots \| \mathsf{M_{b_2}} \| \sigma_{\mathsf{sk}}(\mathsf{M_{b_2}}))$. The untrusted Logger then transmits the blocks, the proofs and the generated signature as a *response* to the Auditor.
3) The Auditor validates the Logger's response signature and then iteratively verifies the integrity proof for each log block. If successful, the Auditor stores the logs and proofs. It also notifies the Logger of the successful *result*; this message will become relevant when we introduce the decentralized version of the protocol. If verification fails, the Auditor will raise an alert to notify the administrator of an incorrect Logger.

VII. Decentralized Auditing

In spite of remaining the predominant common practice for logging in large organizations, a centralized auditing strategy might be problematic when considering our powerful attacker model. In fact, while centralized auditing retains a copy of the integrity-protected logs off of the (potentially compromised) host, its log server represents a single point of failure. As a result, a viable attack strategy against centralized CUSTOS would be for the adversary to participate in the auditing protocol in an honest manner, and then attempt to compromise the log server. Once the log server is compromised, the adversary could engage in the same anti-forensic activities that they would have otherwise conducted on the host, but without raising alerts that would cause their immediate detection.

[5]The ability for the Auditor to request previously committed, already verified log blocks will become relevant for reconstruction purposes.

TABLE II: Parameters for decentralized audits, controlled by administrative policy and can be different for each participating node.

Notation	Description
T	Period between audits. That is, a node initializes a round of audit challenges every T.
w	Number of audit challenges that a node sends during each audit round.
μ	Timeout to receive a response in the protocol; μ must be less than T.
r	Number of nodes where each block of log entries needs to be replicated.

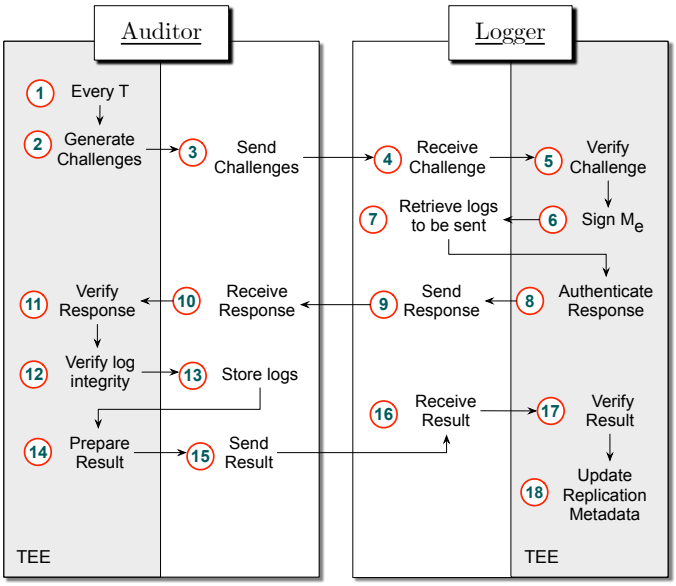

Fig. 3: System-layer diagram of an audit. The three protocol messages in Figure 2 correspond in this diagram to events 3, 9, and 15, respectively. The functionalities running inside the TEE are limited to generating protocol messages, verifying message authenticity and updating the enclave state in response to successful audits.

In this section, we address this limitation through an alternative, decentralized auditing strategy comprised of 3 principal components: (1) we now place a trusted Auditor component running inside the enclave of each host within CUSTOS' deployment network; (2) this Auditor is responsible for administering *Decentralized Audit Challenges* in concert with the other nodes in the network; (3) as this protocol results in the distributed replication of log blocks throughout the network, a *Distributed Log Reconstruction* mechanism is provided that allows administrators to efficiently retrieve a forensic record of events from a particular host over a given time span. We describe each of these components below.

1) Trusted Auditor: When first initialized, the trusted Auditor is provided a set of parameters (summarized in Table II) that govern its behavior. They include the period between audit challenges T, the number of nodes to challenge in each round w, and a timeout value μ that the Auditor should wait on a challenge response before declaring failure. These values can be fixed across all nodes or vary from node-to-node.

TABLE III: An example Global State Report (G) with $r = 3$, compiled by the administrator to track the audit results of each node, at the granularity of single log blocks. ✓ indicates a successful audit, while ✗ indicates an unsuccessful one.

Node	Block ID	Auditors
...	...	...
1	71	{4 ✓, 6 ✓, 7 ✓}
1	72	{3 ✗, 6 ✗, 7 ✗}
2	80	{7 ✓, 8 ✓, 9 ✓}
3	42	{1 ✓, 2 ✓, 5 ✓}
...	...	...

Additionally, we introduce a new parameter r for the Logger component, which specifies the number of nodes on which each of its log blocks should be replicated. The Logger and Auditor components run in the same enclave and process space and share the same identity key pair.

2) Decentralized Audit Challenges: The decentralized auditing variant builds on the same audit protocol used in the centralized version; Figure 3 shows a system-layer diagram of how the audit challenge flows through the trusted and untrusted components of two nodes, one in the Auditor role and one in the Logger role. Below, we explain how decentralized auditing wraps the audit protocol described in Section VI.

1) Every T, the Auditor selects w nodes uniformly at random from the network. It then issues to each node an audit challenge over $(\mathsf{b}_1 = *, \mathsf{b}_2 = *)$, which are wildcards to be resolved by the receiving Logger according to its replication needs. (①-③).
2) Each Logger responds to the audit challenge in the same manner described above. The Logger always commits and returns the most recent log block (i.e., $\mathsf{b}_2 = \mathsf{e}$), and it will adjust b_1 to include additional historic blocks that have not yet been replicated r times if they have not been sent to that Auditor before (④-⑨).
3) Upon receiving a response, the Auditor checks that the response is valid and then verifies the integrity proofs of the log blocks included in it (⑩-⑬). If the response is invalid or the Logger does not respond within μ, the Auditor alerts the administrator of a failed challenge. If the response is valid but the verification of log integrity fails, the Auditor alerts the administrator that the logs of the challenged node have been tampered with.
4) If both response and log integrity were valid, the Auditor sends a *result* as a confirmation to the Logger that the log was verified and replicated (⑭-⑮). If such confirmation is valid, the Logger updates its replication count for the audited blocks (⑯-⑱). If no valid confirmation is received within μ, the Logger discards the challenge. The result of each audit is also mirrored to an administrative machine for accounting purposes.

3) Distributed Log Reconstruction: Decentralized auditing prevents the log storage server from becoming a single point of failure; in the steady state of the protocol, each node's logs will be stored with r redundancy at different remote nodes in the network. However, decentralized storage also complicates the matter of inspecting the logs during post-mortem investigations. To address that, we present the following algorithm that enables the parallelized reconstruction of a target node's log history at a single point in the network. To begin, the administrator compiles the *result* messages of each audit challenge into a *Global State Report (G)*, an example of which is shown in Table III. G tracks the integrity state of each node's logs over time, but also indexes where each log block has been replicated. To reconstruct a target node v's logs over a block range s to e, the administrator invokes the Retrieve function, shown in Algorithm 6. This function parallelizes retrieval of logs across the available nodes in G. To maximize throughput, the function tracks which nodes are currently fielding requests on other blocks and minimizes the number of outstanding requests to the same node. While omitted for brevity, Retrieve also supports recovery from request timeouts and invalid responses, and ResponseCallback verifies v's signature over block M_i as in the original audit.

Algorithm 6: Pseudocode of function for decentralized log reconstruction. Min returns the minimum value in a set, Request retrieves node v's log block M_i from node n, Sum sums the values in a set, and ResponseCallback executes when node n's response to Request arrives.

```
Function Retrieve(node v, bl_id s, bl_id e, report G)
    Logs ← []
    ReqCnt ← [∀n ∈ N, 0]
    for i = s to e do
        Locations ← G[v][i]
        n ← l ∈ Locations, s.t. ∀k ∈ Locations
                                  ReqCnt[k] ≥ ReqCnt[l];
        Request(n, v, i, ResponseCallback)
        ReqCnt[n] ← ReqCnt[n] + 1
    while Sum(ReqCnt) > 0 do
        wait
    return Logs

Function ResponseCallback(node n, bl_id i, block M)
    Logs[i] ← M
    ReqCnt[n] ← ReqCnt[n] − 1
```

VIII. Implementation

We implemented CUSTOS in C for Linux Audit 2.8.2. In particular, we extended the `auditd` user space daemon [70] to be integrated with an Intel SGX enclave that supports all of CUSTOS' trusted logging and auditing functionalities. Our implementation consists of 2,254 lines of C code (excluding Makefiles, libraries, comments and blank lines), of which 658 run inside the SGX enclave. The TEE functions required by our design are provided by Intel SGX and the monotonic counter used by the Logger is an SGX-managed hardware counter.[6] We use Intel SGX's trusted cryptography library to compute hashes (with SHA-256) and digital signatures (with ECDSA), and the TPL library [39] to serialize messages before network transfer, which we implement using TCP sockets. Importantly, however, even though we only implemented it on top of Linux Audit, CUSTOS is designed to be neutral to the underlying operating system and audit framework.

[6] CUSTOS' use of the monotonic counter feature is limited to one write per server startup, which is safe with regard to the risks of memory wear-out [78].

IX. Security Analysis

We now explain how CUSTOS assures the intended security and design goals. The correctness of **G1** (*tamper-evident logs*) can be evaluated by enumerating the space of attacks against the Logger. For simplicity, we assume here that a centralized audit has been issued from block zero to the present.[7]

- *Historic Event Deletion*. An attacker cannot delete arbitrary events from a node's historic log records. Removing a subset of events from a block will invalidate its integrity proof, which cannot be forged. Attempting to remove an entire block will invalidate the response, which cannot be forged. A *Truncation Attack* on the log will similarly be detected by validating the response. This is because all protocol messages are processed inside an enclave, which protects cryptographic keys from the untrusted OS.
- *Historic Event Tampering*. An attacker cannot insert or modify events into a committed log block without invalidating its integrity proof. An attacker also cannot re-order blocks because the trusted component of the Logger signs over the chronologically-ordered challenge response.
- *Protocol Termination*. An attacker with root privilege is able to terminate the Logger process at any time. An attacker may use this ability to try to prevent a block's commitment, but will then need to restart the Logger so it can respond to future challenges. Because the current block is committed during the Shutdown Phase, which is always invoked during regular process killings (in our implementation, from auditd's SIGTERM handler), the attacker will have to force kill the process. However, by skipping the Shutdown Phase there will not be an up-to-date sealed-e on disk that will unlock to the enclave's current configuration (recall that mclD is a hardware counter that is incremented in the Startup Phase). Because the attacker cannot forge sealed-e, the Logger will raise an error, which will be detected during the audit. By the same logic, the attacker is unable to launch *Rollback Attacks* while the Logger is shutdown because this will cause parameter unsealing to fail during the Startup Phase.

CUSTOS satisfies **G2** (*third-party verifiability*) by having each node publish its public key after initialization. This key can be used to verify any log blocks produced by the node in an online challenge or an offline (e.g., Court-related [52], [53], [33]) audit. Recall that it is necessary for the organization to deploy a key management service for nodes' public keys.

Our system facilitates **G3** (*fine-grained audits*) by permitting audit challenges over ranges of log blocks. The number of log events in a block varies with the workload of the system; however, the administrator can exert control over the size of blocks by tuning the parameters T and w. This is because a block is guaranteed to be committed each time an honest node receives an audit challenge. Because audits are fine-grained (to the granularity of a single log block), our protocols can also be used to issue proof-of-retrievability challenges on subsets of historic log blocks that have previously been verified.

CUSTOS pursues goal **G4** (*log availability*) by storing integrity-verified logs at multiple remote locations during the decentralized auditing: once a log block has been audited, the attacker will not be able to erase it unless they compromise all the r nodes in which it is stored. The higher r is, the more the complexity and cost of such an attack increase as compared to the centralized scenario. Further, these logs can be retrieved using CUSTOS' log reconstruction protocol. The security of **G4** can be analyzed by enumerating the space of attacks against the decentralized auditing protocol. Let v be a compromised node that seeks to conceal events contained in block M_e. We have previously established that v will be detected if they fail to reply to an audit challenge for M_e within μ seconds, and v cannot forge a valid response over tampered logs.

- *Malicious Auditor Role*. Until v is detected, they may attempt to lie about an honest node's correctness to inject confusion into audit results. This would require v to be able to generate a $result$ message that implicates the honest node, but this is not possible because the Auditor component will only generate a $result$ message when presented with a valid authenticated audit response. Because v cannot forge the $result$ message, it is not in their interest to lie about the honest node.
- *Colluding Auditors*. Multiple compromised nodes may attempt to collude in the decentralized audit to conceal their presence. A second dishonest node k cannot force an audit challenge to v, but may randomly select v. Because the untrusted environment controls network transmission, k could then drop the $result$ message that implicates v. However, this would only delay the detection of v, whose Logger would attempt to transmit M_e again in future challenges since k did not confirm its replication. More shrewdly, v and k could both comply with the audit, then immediately delete both copies of M_e. Because v's trusted Logger component received a valid $result$ from k, it may conclude that M_e was replicated r times and stop transmitting it in future challenges. **G4** thus depends on the probability that v receives consecutive challenges by r malicious nodes and no honest nodes. We demonstrate that this probability is negligible in Section IX-A.

Finally, CUSTOS satisfies **G5** (*minimal invasiveness*) in that it is fully interoperable with any upstream applications that process Linux Audit events. CUSTOS runs in the process space of `auditd`, but its semantics are independent of the existing `audit-userspace` code base. In fact, CUSTOS requires inserting just 26 lines of code into existing source files. This makes porting CUSTOS to new versions extremely simple. In addition, while we described CUSTOS in the scenario of online auditing frameworks, CUSTOS' Logger could easily be extended to automatically generate integrity proofs offline at predefined time intervals or block sizes, without interaction with any external party.

A. Probabilistic Analysis

Let v be a compromised node that seeks to conceal events contained in block M_e. We analyze the probability that v succeeds in its mission without being reported, in the presence of a distributed adversary. Let $N+1$ be the number of nodes participating in the protocol and $f+1$ be the number of compromised colluding nodes, including v. v's enclave will attempt to replicate M_e to r auditors, in the order of challenge

[7] Analyzing the security of **G1** on either auditing protocol is analogous because centralized audits are a special case of decentralized ones.

TABLE IV: Probability ϵ that a compromised node v is not audited by any honest node under varying configurations of N, r and f. $N+1$ is the total number of nodes participating in the protocol, and $f+1 \leq N$ is the number of compromised nodes (including v).

N	$r = \lfloor \frac{N}{25} \rfloor$		$r = \lfloor \frac{N}{10} \rfloor$	
	$f = \lfloor \frac{N}{4} \rfloor$	$f = \lfloor \frac{N}{2} \rfloor$	$f = \lfloor \frac{N}{4} \rfloor$	$f = \lfloor \frac{N}{2} \rfloor$
50	5.38×10^{-2}	2.449×10^{-1}	3.74×10^{-4}	2.51×10^{-2}
100	3.23×10^{-3}	5.87×10^{-2}	1.89×10^{-7}	5.93×10^{-4}
200	9.74×10^{-6}	3.38×10^{-3}	2.92×10^{-14}	3.32×10^{-7}

arrival. v can also re-order challenges before passing them to the enclave so long as they responded to within μ seconds.

1) Case A: Suppose v receives r consecutive audit challenges from colluding compromised auditors and no challenges from honest auditors. This is the best-case scenario for the adversary as v does not even need to re-order challenges. The first challenge arrives from a colluding node with probability $\frac{f}{N}$. Given that, the second challenge arrives from a different colluding node with probability $\frac{f-1}{N-1}$. It follows that the probability of attack success (ϵ) is:

$$\left(\frac{f}{N}\right)\left(\frac{f-1}{N-1}\right)\cdots\left(\frac{f-r+1}{N-r+1}\right) = \frac{f!(N-r)!}{(f-r)!N!} = \frac{{}^{f}P_r}{{}^{N}P_r}$$

Table IV computes the value of this probability for different configurations. In a network of $N = 100$ nodes, even with $f = 50$ compromised colluding nodes, it suffices to choose $r = 4$ to have the probability of attack success $\epsilon < 5.88\%$.

2) Case B: Let us now consider the case when v re-orders challenges to keep the enclave from processing an honest node's challenge. Assume that v has received $\beta < r$ distinct challenges from colluding nodes and it is waiting for other $r-\beta$ challenges to arrive from other distinct colluding nodes. The probability of this happening is ${}^{f}P_{\beta}/{}^{N}P_{\beta}$, using the same method as in Case A. The $(\beta+1)$-th challenge will arrive from an honest auditor with probability $\frac{N-f}{N-\beta}$, making the probability of v possessing β dishonest challenges followed by one honest challenge:

$$P_1 = \frac{{}^{f}P_{\beta}}{{}^{N}P_{\beta}} \cdot \frac{N-f}{N-\beta}$$

If v chooses to delay the honest challenge while waiting for $r-\beta$ dishonest challenges, the dishonest challenges must arrive within μ before v is detected by the honest node.

We can assume that the number of challenge arrivals per unit of time follows a Poisson distribution [13]; this is because at every T the auditors send challenges to nodes randomly and thus challenges are sent to v independently of one another. Let λ be the average rate of challenge arrival and X be the random variable representing the number of challenges arriving at v over a given interval. If we observe the system from the perspective of v for a long time $\mathcal{T}$, the total number of audit challenges generated will be $\frac{\mathcal{T}}{T} \cdot N \cdot w$. There is a probability $\frac{1}{N}$ that v gets selected for each of those challenges. Therefore, the average number of challenges that arrive at v within $\mathcal{T}$ is $\frac{\mathcal{T}}{T} \cdot N \cdot w \cdot \frac{1}{N}$. It follows that the average rate at which v receives a challenge is $\lambda = \frac{\mathcal{T}}{T} \cdot N \cdot w \cdot \frac{1}{N} \cdot \frac{1}{\mathcal{T}}$, that is $\lambda = \frac{w}{T}$. Since X is Poisson distributed, it follows that the probability that v receives m challenges within μ is:

$$P_2 = e^{-\mu\lambda} \cdot \frac{(\mu\lambda)^m}{m!}$$

Now we calculate the probability P_3 that, among m challenges received by v, at least $(r-\beta)$ of them are from new colluding nodes. For exactly y auditors to be colluding among these m, distinct y nodes are chosen from remaining $f-\beta$ colluding nodes, and $m-y$ nodes are chosen from remaining $N-f-1$ honest nodes. Let random variable Y denote the number of colluding auditors among m. The probability $P(Y=y)$ will be calculated as:

$$P(Y=y) = \frac{\binom{f-\beta}{y}\binom{N-f-1}{m-y}}{\binom{N-1-\beta}{m}}, \quad y = 0, \ldots, m$$

Consequently, the probability that less than $r-\beta$ colluding nodes send challenges is:

$$P_3 = \sum_{y=0}^{r-\beta-1} P(Y=y)$$

Therefore, the probability that at least $r-\beta$ of the m challenges are from remaining colluding nodes is $1-P_3$. The probability that v receives m challenges within μ after an honest node b's challenge arrival and that at least $(r-\beta)$ among the m challenges are from distinct colluding nodes is $P_2 \cdot (1-P_3)$. Since X follows a Poisson distribution, the cumulative probability P for any m in this scenario will be:

$$P = P_1 \cdot \sum_{m=1}^{\infty} P_2(1-P_3)$$

The distributed adversary will be in the least advantageous position if the first of the r challenges to replicate M_e is from an honest auditor b, meaning that $\beta = 0$ and that to avoid detection v would have to receive r colluding challenges within μ. The best-case scenario for the adversary (other than Case A) is when the first challenge from an honest auditor b arrives after $\beta = r-1$ challenges from colluding auditors. In this case, v only needs one more challenge from a colluding node within μ to succeed in its mission. However, even in this best-case scenario and with $N = 100$, $f = 50$, $r = 4$, $w = 10$, $\mu = 15$ s, and $T = 60$ s, v will be able to avoid detection with a probability of just 4.43%.

X. Performance Evaluation

We now characterize the performance of CUSTOS. To do so, we leverage two experimental setups. In both, we configured Linux Audit to log all forensically-relevant system calls, using the same ruleset employed in [66], [31], [75].[8]

[8]This set includes the syscalls: read, readv, write, writev, sendto, recvfrom, sendmsg, recvmsg, mmap, mprotect, link, symlink, clone, fork, vfork, execve, open, close, creat, openat, mknodat, mknod, dup, dup2, dup3, bind, accept, accept4, connect, rename, setuid, setreuid, setresuid, chmod, fchmod, pipe, pipe2, truncate, ftruncate, sendfile, unlink, unlinkat, socketpair, splice.

- **Point-to-point Setup (Bare Metal)**: We deployed a Logger on a server with an Intel Core i7-7700K CPU at 4.20 GHz (4 physical cores) and 64 GB RAM running Ubuntu Server 18.04 64 bit (Linux 4.15). We deployed an Auditor on a different server with an Intel Xeon E5-2630 v4 CPU at 2.20 GHz (10 physical cores) and 64 GB RAM running Ubuntu Server 16.04 64 bit (Linux 4.4). The Logger used SGX SDK version 2.3.1 with debug mode on, while the Auditor used the same version in simulation mode. During experimentation, we observed an average latency of 176 μs between the two machines.
- **Distributed Setup (VMs)**: We deployed CUSTOS on a cluster of 100 Amazon EC2 `m4.xlarge` instances, each with 4 VCPUs (2.3 GHz Intel Xeon E5-2686 v4 or 2.4 GHz Intel Xeon E5-2676 v3) and 16 GB of RAM. Each instance was running Ubuntu Server 18.04 64 bit (Linux 4.15) and used SGX SDK version 2.3.1 in simulation mode [9]. A small script synthesized a constant workload that generated an average of 32 log events (11.8 KB of log data) per second on each node, which is a realistic (not worst-case) rate for a server [76]. During experimentation, we observed an average latency of 169 μs between any two machines in the cluster.

A. Logger Microbenchmarks

We start by using the bare metal setup to measure the time that CUSTOS' Logger takes to perform each of the five phases described in Section V. We run this microbenchmark by manually invoking each Logger's operation 500 times, including in the measurement the time required to context switch into and out of the enclave. Table V shows the results. The phases that involve interaction with a hardware counter, Initialization and Startup, are the most costly. This is because Intel SGX's monotonic counter operations are notoriously slow [78], but these operations occur only once per session. The next most costly phases, Commitment and Shutdown, involve cryptographic signatures. However, these operations are a function of challenge frequency, not the workload, and in practice will occur orders of magnitude less frequently than the Logging operation. Fortunately, Logging (ecalls) is the most efficient phase at 4.71 μs per event processed, but the performance of this operation is paramount, since it is invoked once per log event. We observed that the main cost of this phase is switching context between the untrusted OS and the trusted enclave [46]: thus, to further improve its performance, we created a second Logging implementation using Hotcalls, which were recently introduced by Weisse et al. [126]. Hotcalls provide the same security guarantees of ecalls, but allow us to reduce the cost of context-switching to the enclave, enabling a significant speed-up (0.92 μs) at the expense of running an additional background thread permanently spinning inside CUSTOS' enclave. For the rest of our evaluation, we will use the Hotcalls-based implementation of CUSTOS. In Section X-B, we will evaluate the system-wide impact of this choice.

1) Prior Work Comparison: We have argued that prior solutions for tamper-evident logging do not meet the needs of commodity operating systems. To validate this assertion, we perform a direct comparison of CUSTOS to prior work: SGX-Log [57], [123]; a logger based on the TPM2 hardware's `extend` (Logging Phase) and `quote` (Commit Phase) operations [34]; and Hartung et al.'s scheme based on BGLS signatures [41]. For SGX-Log and BGLS, we conservatively set highly-favorable parameters for performance.[10] We focus here on the critical Logging phase, which dominates performance cost. SGX-Log takes a median time of 0.80 ms (ecalls) and 0.79 ms (Hotcalls), TPM `extend` operations take 20 ms, and the BGLS-based scheme takes 31.89 ms. Further, the TPM implementation also requires a `quote` operation to produce a proof, adding an additional median cost of 734 ms per block.

TABLE V: Microbenchmarks on Logger operations. We report the median execution times over 500 runs, including results for the Logging phase using both ecalls and Hotcalls. We compare to SGX-Log [57] and a TPM2 [34] logger (same log messages), as well as BGLS signatures [40] (smaller fixed-size messages), parameterizing to optimize the performance of each. CUSTOS' logging phase is the only one that can scale to more than a million events per second.

Phase	CUSTOS	SGX-Log	TPM2	BGLS
Initialization	94.55 ms	–	–	–
Startup	109.10 ms	–	–	–
Logging	4.71 μs	0.80 ms	20 ms	31.89 ms
Logging (Hotcalls)	**0.92 μs**	0.79 ms	–	–
Commitment	128.87 μs	–	734 ms	–
Shutdown	188.98 μs	–	–	–

From these results, it can be seen that CUSTOS *outperforms existing solutions by three to five orders of magnitude.* Another way to compare the performances of these logging systems is to consider the maximum throughput they can scale to support. Using the numbers from Table V, we compute that CUSTOS could process up to 1,086,956 log events per second. On the other hand, SGX-Log could process at most 1,266 events per second, the TPM2-based solution could reach at most 50 events per second, and the BGLS approach could only sign at most 31 events per second. While the actual log throughput ultimately also depends on how many events the underlying logging framework can process and send to through the secure logging phase, if we consider that single hosts can produce hundreds of thousands of system calls per second [24], then CUSTOS is the only solution that can scale to match such throughput.

2) Vanilla Linux Audit Comparison: Finally, we instrument `auditd` to measure the average time that insecure Vanilla Linux Audit takes to process a single event as compared to CUSTOS. We run this microbenchmark by measuring the time `auditd` takes to process 40,000 identical log events. We find that CUSTOS-enabled `auditd` takes an average of 6.61 μs/event whereas Vanilla `auditd` takes an average of 5.67 μs/event. CUSTOS thus imposes an average 16.6% overhead on unmodified, insecure `auditd`. This reported overhead is conservative in that our measurement did not capture the time required for `auditd` to flush events to disk, making the overhead imposed by CUSTOS lower in practice. We conclude that CUSTOS' tamper-evident logging protocol imposes a reasonable overhead over insecure Linux Audit's log processing time. We will analyze the impact of this overhead on real applications in Section X-B.

[9] EC2 instances with hardware SGX capability are not currently offered.

[10] We configured SGX-Log to use a block-size of 1000 log messages to avoid including in our measurement the impact of SGX-Log's sealing phase, and we configured the BGLS-based scheme with $n = 1000$ and $l = 1000$.

TABLE VI: Application benchmark results. We report the medians over 10 runs. For httpd and NGINX, we used apache bench [119] configured to send 100,000 requests from a single thread and output the average time per request. For Redis, we used the built-in redis-benchmark configured to send 250,000 requests from a single thread and output the average time per request. We ran the Blast benchmark in two configurations: first, limiting it to one thread only; second, letting it use all the CPU threads available.

Test Type	Vanilla	CUSTOS	Overhead
nginx	72 μs	73 μs	1.39%
apache2	75 μs	76 μs	1.33%
redis	23,520 ns	23,932 ns	1.75%
blast	938.641 s	954.104 s	1.65%
blast-multicore	222.791 s	237.027 s	6.39%

B. Logger Macrobenchmarks

To evaluate the system-wide runtime overhead of CUSTOS, we use the point-to-point setup to measure the performance of a series of application benchmarks while running CUSTOS in the background. In particular, we benchmark three server applications (httpd [120], NGINX [87] and Redis [106]) and one scientific-computing application (Blast [27]). Our choice of server-oriented applications is dictated by the intended deployment environment (enterprise servers); further, the benchmarks we choose are known to generate larger-than-average system call loads [76]. Note that given our intensive configuration of Linux Audit, we were unable to run classic OS benchmark suites including UnixBench [72] and LMBench [82].[11]

We run each application benchmark 10 times and report the results in Table VI. The runtime overheads of our CUSTOS' implementation in the first four benchmarks are all under 2%. This is because CUSTOS does not add cycles to the execution of system processes besides `auditd`, which runs asynchronously from the processes that the audit stream describes. However, CUSTOS still incurs overhead due to its use of Hotcalls and its interaction with the Architectural Enclave Service Manager service (aesmd). This cost is most visible in the blast-multicore benchmark, which is CPU-bound and configured to use all the 8 logical cores of our experimental machine. However, even in this scenario, CUSTOS' runtime overhead against insecure Vanilla `auditd` remains at a reasonable 6.39%. We conclude that CUSTOS' extensions to Linux Audit impose acceptable system performance costs.[12]

C. Audit Protocol

We next use the point-to-point setup to characterize the performance of CUSTOS' audit protocol, used in both the centralized and the decentralized auditing scenarios. Intuitively, we expect the cost of an audit to be dominated by the time required to transmit and process log data, which are orders of magnitude larger than CUSTOS' protocol messages. To confirm that, we measure the end-to-end time required by audits from the perspective of the Auditor, which spans the moment of challenge generation to the moment of result transmission (Figure 3, ②-⑮), while varying the number of logs transmitted. We set the period between audits (T) to 1 second, then throttle our workload script to generate variable levels of logs per second. We consider 33 different workload levels ranging from 1 to 9900 logs/second, repeating each workload level 100 times. The results can be found in Figure 4. As expected, the time to complete an audit grows linearly with the size of the log.

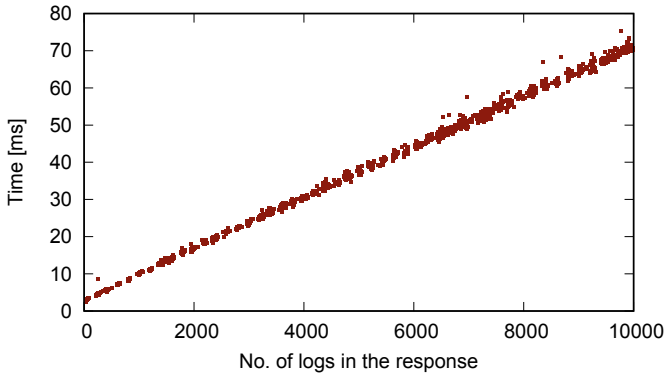

Fig. 4: Time required by an auditor to complete an audit by number of logs in the response. The measurements include all operations from the moment of challenge generation to the moment of result transmission. The cost of an audit grows linearly with the size of the response.

Another way to evaluate the practicality of our audit protocol is to ask the question "For a given T, at what log size would it take more than T seconds to complete an audit challenge?" This log size represents the breakdown point of the system, where audit challenges are now issued with greater frequency than they can be responded to. Extrapolating from the above test with T set to 1 s, we find that the Logger would need to generate 145,000 events per second for the processing time of an audit to exceed 1 s. We therefore conclude that, like the CUSTOS logging mechanism, our audit protocol is practical even for hosts under extreme load.

D. Audit Frequency

We now use our distributed setup to evaluate our decentralized auditing protocol. As the cost of an individual audit is identical in both the centralized and decentralized variants, we here turn our attention to the frequency at which each node is audited in a realistic CUSTOS' deployment. To this end, we run CUSTOS on a cluster of N=100 nodes, each with an auditing interval of T=10 s, replication factor r=1 and a fixed number of challenges per round w.[13] We instrument nodes to initiate their first audits at uniform offsets of T to smooth the network impact. We then capture the times at which each node gets audited over a period of 10 minutes of observation, repeating the experiment with w=$[1, 2, 4]$. For each challenge received by each node, we compute the time passed since the same node received the last challenge.

[11] This limitation, shared with prior work [67], [76], is due to Linux Audit not scaling to support the intensive loads of these suites.

[12] When deployed in CPUs with a low number of cores, the cost of using Hotcalls may outweigh its gains. In these cases, CUSTOS can be configured to use standard ecalls at the cost of a slightly decreased throughput (Table V).

[13] While r=1 is not the most secure configuration for **G4**, the goal of this experiment is to capture audit frequency, which does not depend on r. Similarly, our choice of a constant (not worst case) logging rate and the use of SGX in simulation mode only minimally affect audit frequency.

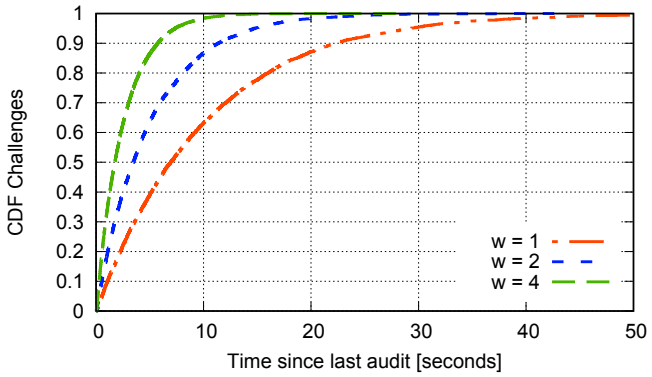

Fig. 5: Cumulative distribution function of the frequency at which nodes are audited with varying number of challenges issued per audit round (w). Results are based on a 100-node network over 10 minutes of observation, with nodes initiating audit rounds every T=10 s. When w=4, 98.4% of the time nodes were challenged again within T of receiving the last challenge.

TABLE VII: Size of protocol messages in our CUSTOS' implementation. The reported sizes include the size of one or more ECDSA signatures (64 bytes) per message.

Message Type	Size [Bytes]
Challenge	106
Result	98
Response metadata	210
Log Block	76 + 4 · number_of_logs + size_of (logs)

Figure 5 shows the results as a cumulative distribution function. Intuitively, when w increases, the frequency at which nodes in the network are audited also increases. With w=4, 98.4% of the challenges were received within 10 s ($1T$) of receiving the last challenge, and in the worst case it took only 27.96 s ($< 3T$) for a node to be audited. Another way to interpret these numbers is that in 98.4% of the cases it took less than T to audit and replicate a node's logs, and in 100% of the cases less it took less than $3T$.

E. Network Cost

Next, we evaluate the network impact of auditing at higher frequencies. In the above experiment, each node produced approximately 19,200 events (7 MB) of log data, for a total of 1,920,000 events (700 MB) of log data across the 100 nodes. Let 700 MB be our baseline for comparison, as this would be the network cost for streaming logs to a centralized server without CUSTOS. With CUSTOS and w set to 1, each node completed a total of 60 audit challenges for a total of 6,000 challenges completed system-wide. Factoring in the size of CUSTOS' protocol messages, given in Table VII, CUSTOS' total network cost for these 6,000 challenges was 711 MB (1.5% overhead). However, in the experiments with w set to 2 and 4, CUSTOS' network cost was just 714 MB (2% overhead) and 719 MB (2.7% overhead), respectively. The reason for this extremely gradual increase in cost is that the same number of log messages are transmitted in each scenario. Increasing the frequency of challenges only incurs the additional overhead of CUSTOS' small protocol messages, which are dominated by the baseline cost of log transmission.

Based on this result, it is clear that challenge frequency imposes no meaningful difference in network cost. More challenges simply decrease the size of the average log block and probabilistically reduce the time before each block is replicated (cf. Section IX-A). Because of this, it is to the advantage of the administrator to use many audit challenges (T and w) when deploying CUSTOS; doing so will verify and replicate logs sooner with minimal increase in network cost. Conversely, modifying the replication factor r would increase the network cost due to the larger amount of log data to transmit. This increase in network (and storage) requirements reflects the cost that must be paid for a stronger assurance of log availability against a distributed adversary.

F. Enclave Memory Usage

For CUSTOS to be minimally invasive, it is not enough to achieve low performance and network overheads. As we mentioned in Section III-3, an important constraint of Intel SGX is its limited amount of protected memory (128 MB on current hardware), which is shared across all enclaves on the system. To evaluate the memory usage of CUSTOS' enclave in our implementation, we use the official Enclave Memory Measurement tool (`sgx_emmt`) from the Intel SGX SDK, which returns the stack peak usage and heap peak usage in KB during enclave's execution. We launch it on our SGX-enabled machine used in the point-to-point setup, while running both Logger and Auditor on it. The reported peak memory usage of CUSTOS is 5 KB of stack and 16 KB of heap, for a total of 21 KB of memory, which is less than 0.02% of SGX's protected memory. This usage does not increase with the rate of log events, since only one event at a time is processed inside CUSTOS' enclave during its operations. We conclude that CUSTOS' utilizes enclave memory efficiently.

XI. ATTACK CASE STUDY

In this section, we empirically demonstrate the effectiveness of CUSTOS at detecting log tampering by presenting a case study based on a realistic attack scenario. In particular, we simulate the nation-state APT attack scenario "Firefox Backdoor w/ Drakon In-Memory" from the DARPA Transparent Computing program dataset [59]. This attack starts with a remote exploit of Firefox 54.0.1, which allows the attacker to open a command shell that gives them unprivileged access to the host; this shell is later used to download a malicious binary file ("Drakon") onto the victim host, which is then run to achieve root access and give the attacker persistent access to the host. However, unlike the original attack, our adversary is aware that their attack can be detected from the logs (through, e.g., [83], [23]) and therefore proceeds to erase them immediately after achieving root access. We focus on log erasure since it is known to be used in real-world attacks [56], [37], [104], [16], [17], [90], [77], [9].

We simulated this attack by replaying its trace on one CUSTOS-enabled host v, which we chose at random from the 100 nodes of our distributed setup (see Section X). We configured CUSTOS with parameters T=10 s, r=1, and w=4. The results are shown in the timeline of Figure 6. CUSTOS replicated on

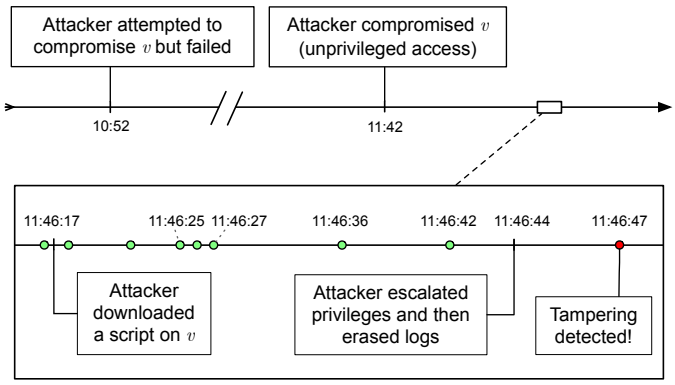

Fig. 6: Timeline of our attack case study, where host v replays the trace of the "THEIA Firefox Backdoor w/ Drakon In-Memory" attack from the DARPA TC dataset [59]. The circles in the zoomed-in view represent audits from other hosts to v: the green audits successfully verified and replicated logs of the trace; the red audit occurred after the attacker tampered with the logs and failed, causing CUSTOS to raise an alert to the system administrator.

other hosts everything but the last 2 seconds of the trace before the attacker succeeded at achieving privilege escalation. These logs described the adversary's initial compromise attempts, their methods of establishing foothold and their download of a malicious file. The attacker was able to erase the last 2 seconds of the trace before they were replicated. These included the logs describing their method of achieving privilege escalation. However, CUSTOS detected this log tampering 3 seconds later (at the next audit), and alerted the system administrator of the violation. In conclusion, CUSTOS forces attackers into a "lose-lose" situation: either they can be covert and not tamper with logs (which can be used for forensics), or they can erase logs but then be detected by CUSTOS.

XII. DISCUSSION

1) Value of CUSTOS *after full system compromise:* This work considers methods for securing audit logs in the face of a powerful anti-forensic adversary that has gained full control of the system (i.e., root privilege); this threat model does not reflect an artificially-enhanced attacker, but instead reflects the common capabilities and methods of system intruders (e.g., [16], [17], [90], [121], [104], [37], [77], [9]). Against this adversary, ensuring the correctness of log events recorded *post-compromise* is not practical. This is because, after full system compromise, no security measure can prevent the attacker from controlling which events are logged. This grants them the ability to make arbitrary claims about the events on the system. This limitation is common to all secure logging solutions (cf. Section XIII). Nonetheless, CUSTOS provides an invaluable property even against root-level attackers—the intruder is unable to undetectably edit or remove any recorded evidence of their methods of entry onto the system, reconnaissance tactics prior to escalating privilege, and their method of privilege escalation. This information may implicate another node in the network or lead to the discovery of a zero-day exploit, which are valued at tens of thousands of dollars [92] and are thus zealously guarded by attackers. Most importantly, these events are *suspicious* and may lead to immediate detection when CUSTOS is paired with a Threat Detection System (e.g., [115]) or log analysis mechanism (e.g., [23]). CUSTOS forces the attacker into a no-win situation in which they must either immediately erase this evidence (ensuring detection) or permit it to be analyzed by the TDS (risking detection).

2) Integrity of centralized logging architectures: Among existing solutions to tamper-evident logging, the most practical is continuously streaming all logs, without integrity proofs, to a centralized server. The security model for such architectures assumes that the log server is fully trusted and can therefore attest to the integrity of all logs stored on it. While an argument could be made that the log server can be hardened against attack, we do not find this to be a convincing solution to the secure logging problem. Log management utilities are complex software and there is no reason to believe that these artifacts do not also contain exploits that could be identified by well-funded attackers. In fact, while we are not aware of any documented cases of log server compromise, given the general strategies of lateral movement and anti-forensics associated with APTs, it seems highly likely that log servers are the target of frequent attack once perimeter defenses are breached.

3) Adaptive Attack Strategies: If the methods presented in this work are deployed in practice, we must conservatively assume that the intruder would adopt an optimal strategy for evading detection. Because CUSTOS probabilistically guarantees detection in the event of tampering, the attacker could decide not to erase logs until they have completed their mission objective, at which point they can erase the logs at minimal cost. This strategy underscores the importance of pairing the CUSTOS Logger with our decentralized auditing strategy, which redundantly stores evidence on additional nodes in the system. Further, this attack strategy demonstrates the importance of deploying CUSTOS alongside other security products such as Threat Detection Systems and log analysis mechanisms (e.g., [23]). Again, our ultimate goal in designing CUSTOS is to create a "lose-lose" situation for the attackers, forcing them to choose between a covert strategy and an anti-forensic strategy.

4) Reliance on TEEs: CUSTOS relies on features of TEEs and this assumption limits its potential for deployability in legacy enterprise environments without TEE support. While this is a limitation of our paper, believe it is reasonable as TEEs are enjoying increasingly wide deployment. For example, off-the-shelf client and server Intel CPUs have been shipped with SGX since 2015. Moreover, Intel recently announced the Intel SGX Card which can equip legacy systems with SGX [48].

5) Race Conditions: We have considered log integrity under two conditions: *prior* to machine compromise and *after* machine compromise. We also saw that CUSTOS can protect all events that have been recorded prior to machine compromise, and assumed that after machine compromise the attacker can take any action, including disabling the audit system. However, we have not discussed at length whether or not the log events describing the *point of compromise* are entered into the tamper-evident record before the adversary is able to erase them. In particular, if t is the moment of compromise, then for a system call x, happening at time $t_x \leq t$, the corresponding log message m_x is guaranteed to be tamper-evident only if it reaches CUSTOS' enclave before a subsequent anti-forensic action y (with $t_y > t$) is executed. As such, there exists a race

condition between the time m_x reaches the enclave and the moment the attacker can erase it. A variety of factors, including scheduling decisions, might affect whether this race condition is exploitable. Nonetheless, recall that **G1** is concerned with detecting tampering only with log messages *recorded pre-compromise*, which will likely include significant system calls (e.g., events from 10:52 to 11:46:42 in Section XI): CUSTOS, like prior work, does not seek to protect events that have not yet been recorded by the moment of compromise.

6) Diagnosing Benign Faults: Finally, one limitation of CUSTOS is that it cannot differentiate benign power failures or crashes from protocol termination attacks, creating the possibility for false-positive alerts. The root cause of this issue is that Intel SGX and other TEEs are unable to measure the runtime integrity state of the untrusted OS at the moment when an asynchronous failure occurs, and are thus unable to tell whether the failure was benign or caused by a malicious OS. This is an important problem with existing TEEs, but orthogonal to the aims of this work. Furthermore, alternate mechanisms exist to distinguish failures from attacks: for example, out-of-band information such as telemetry data from power distribution units will allow the system administrator to differentiate a power failure from an attack. The administrator can also investigate faults using techniques from computer forensics. For example, looking at logs that were not lost, open ports and running processes may reveal useful information on the type of fault which occurred; external network traffic monitors and anti-rootkit scanners are also useful tools which can help diagnose the issue. In any case, whether benign or malicious, faults still require manual intervention: we provide an in-depth discussion on how CUSTOS supports recovery from errors in Appendix A. However, in practice, we believe that it is not in an APT attacker's interests to compromise a machine and make it crash immediately before the next audit, as it will cause the system administrator to intervene and effectively prevent them from completing their mission [83].

XIII. RELATED WORK

A. Secure Logging

Many cryptographic approaches have been proposed for tamper-evident logging. Bellare et al. [10] first defined the notion of *forward integrity* for secure audit logs, which consists of generating integrity proofs in such a way that when the logging machine is compromised, previously committed logs will remain tamper-evident. To achieve forward integrity, the signing key evolves over time and expired keys are deleted from the logger. Schemes that use symmetric primitives [10], [108], [109], [114] traditionally rely on hash chains, offering computational efficiency at the cost of higher data overheads while assuming a fully trusted verifier. On the other hand, systems that rely on asymmetric primitives [44] provide third-party verifiability but incur larger computational overheads both to generate and verify proofs. To minimize these overheads, [73], [127] proposed the use of sequential aggregate signatures, but these schemes have been shown to be insecure [40]. Yavuz et al. presented an optimized signing procedure at the cost of a key size that is linear with the number of log entries and no support for fine-grained audits [128]. Most recently, Hartung et al. [41] presented a scheme that combines forward-secure sequential aggregate signatures with forward-secure signatures, but still incurs impractically large computational costs to generate proofs.

The use of various cryptographic data structures has also been proposed in the literature for storing data in a tamper-evident fashion, such as history trees [20], [101] and hash treaps [101]. The trust model for these systems is that messages generated by a host are being stored in a remote untrusted server, and the data structures provide an efficient interactive protocol to verify that a message was correctly recorded. CUSTOS also features untrusted storage servers, but in a more aggressive threat model in which any node, including the host itself, may be compromised. We thus use redundancy to probabilistically ensure that messages are not erased. It would be possible to extend CUSTOS to use these structures as a way to verify that an auditor has not erased replicated logs.

Closely related to ours is the work from Karande et al. [57], who were the first to introduce a protocol that leverages Intel SGX to protect log integrity. Their system relies on hash chains based on symmetric-key cryptography. However, SGX-Log fails to provide third-party verifiability (**G2**) and log availability (**G4**) since log access and verification rely on the particular enclave that sealed the log. Further, SGX-Log is not minimally invasive (**G5**) because encryption breaks interoperability with log analysis applications; additionally, SGX-Log's reliance on frequent writes to an Intel SGX's monotonic counter is vulnerable to memory wear-out. Finally, SGX-Log's per-event processing overhead is too costly for the high-frequency nature of system logging (cf. Section X-A).

B. Data Provenance and Attack Investigation

Related to system auditing are also [7], [31], [62], [63], [84], [97], [99], [76], which focused on techniques to accurately and efficiently collect and analyze system logs. This line of work typically parses system logs into dependency graphs (also known as provenance graphs) that allow to derive insights and scrutinize the causal relationships between events. Several methods have been proposed to automatically recognize security incidents from these graphs [23], [45], [83], [42], [11], [93], [35], [98], [124], to more precisely and accurately reason about the stream of events [67], [75], [65], [74], [5], or to more efficiently process queries to these graphs [71], [29], [30], [54], [55], [96]. Notably, all this work fully trusts the integrity of the logs used as input to their systems. CUSTOS can thus complement these existing systems by providing tamper-evidence to system logs.

C. Secure Hardware

Several works have leveraged the isolation guarantees of Intel SGX [81], [50] to protect user-level applications across domains. Representative systems that focused on "shielding" applications in SGX enclaves are Haven [8] for a lightweight OS, SCONE [1] for Docker containers and Panoply [113] for POSIX interface threads. Glamdring [69] further proposed a framework to semi-automatically partition applications to only run security-sensitive code within enclaves. Rather than secure entire applications, CUSTOS' goal is to secure system logs, thus minimizing the TCB to a small set of critical components.

Intel SGX has also served to enable applications that were either not possible or not practical otherwise, including secure

multi-party computation [3], functional encryption [28], oblivious machine learning [91], integrity assurance for Internet services [2], secure databases [100], secure network function virtualization [112], privacy-preserving cloud computing [46], secure MapReduce computations [110], [22] and access delegation [79]; or to enhance the security of systems such as Tor [61], Zookeeper [15] and Spark [130]. Analogously to these works, CUSTOS leverages secure hardware to enable an application (tamper-evident auditing) to operate efficiently under a stronger threat model that was previously possible.

D. Network Auditing

Adopting a similar threat model to CUSTOS's are the network forensic systems PeerReview [36] and SNooPy [131], which detect faults amongst byzantine nodes participating in a network protocol. The systems detect *some* faulty nodes in distributed environments, provided that a critical mass of correct hosts still exists to witness the misbehavior. Further, these systems only consider network events and cannot speak to the internal state of hosts. In contrast, CUSTOS provides tamper-evidence over considerably larger audit streams that include system-level events and is probabilistically guaranteed to detect compromised nodes engaging in anti-forensic activities.

XIV. CONCLUSION

In spite of the central importance of system logs in responding to modern security incidents such as Advanced Persistent Threats, today's commodity operating systems fail to assure the integrity of system logs beyond the use of typical access controls. CUSTOS is the first tamper-evident logging solution that supports practical operating system constraints. It works by decoupling event logging from their cryptographic commitment—without trading off security—leveraging features of TEEs which are readily available on today's hardware. We demonstrated that CUSTOS' log commitment protocol is three orders of magnitude faster than prior secure logging solutions, and imposes only between 2% and 7% runtime overhead over insecure logging on intensive workloads. Further, we showed that CUSTOS' auditing protocol can detect integrity violations with less than 3% network overhead. CUSTOS thus demonstrates a realistic path forward to achieving practical tamper-evident auditing of operating systems.

ACKNOWLEDGMENT

This work was co-funded by NSF awards #1657534, #1750024, #1909999, and by an Intel ISRA award. We thank the anonymous reviewers for their very helpful comments and suggestions. We also thank Omri Mor, who helped us implement CUSTOS at the initial stages of the project.

REFERENCES

[1] 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 *Proc. of the USENIX conference on Operating Systems Design and Implementation (OSDI)*, 2016.

[2] P.-L. Aublin, F. Kelbert, D. O'Keeffe, D. Muthukumaran, C. Priebe, J. Lind, R. Krahn, C. Fetzer, D. Eyers, and P. Pietzuch, "LibSEAL: Revealing service integrity violations using trusted execution," in *Proc. of the EuroSys Conference (EuroSys)*, 2018.

[3] R. Bahmani, M. Barbosa, F. Brasser, B. Portela, A.-R. Sadeghi, G. Scerri, and B. Warinschi, "Secure multiparty computation from SGX," in *Proc. of the International Conference on Financial Cryptography and Data Security (FC)*, 2017.

[4] A. Bates, K. R. B. Butler, and T. Moyer, "Take only what you need: Leveraging mandatory access control policy to reduce provenance storage costs," in *Proc. of the USENIX Conference on Theory and Practice of Provenance (TaPP)*, 2015.

[5] A. Bates, W. U. Hassan, K. R. Butler, A. Dobra, B. Reaves, P. Cable, T. Moyer, and N. Schear, "Transparent web service auditing via network provenance functions," in *Proc. of the International World Wide Web Conference (WWW)*, 2017.

[6] A. Bates, B. Mood, M. Valafar, and K. Butler, "Towards secure provenance-based access control in cloud environments," in *Proc. of the ACM Conference on Data and Application Security and Privacy (CODASPY)*, 2013.

[7] A. Bates, D. Tian, K. R. Butler, and T. Moyer, "Trustworthy whole-system provenance for the Linux kernel," in *Proc. of the USENIX Security Symposium (USENIX)*, 2015.

[8] A. Baumann, M. Peinado, and G. Hunt, "Shielding applications from an untrusted cloud with Haven," *ACM Transactions on Computer Systems (TOCS)*, vol. 33, no. 3, 2015.

[9] G. Belding, "Ethical hacking: Log tampering 101," https://resources.infosecinstitute.com/category/certifications-training/ethical-hacking/covering-tracks/log-tampering-101/, last accessed 02-07-2020.

[10] M. Bellare and B. Yee, "Forward integrity for secure audit logs," Computer Science and Engineering Department, University of California at San Diego, Tech. Rep., 1997.

[11] K. Berlin, D. Slater, and J. Saxe, "Malicious behavior detection using Windows audit logs," in *Proc. of the ACM Workshop on Artificial Intelligence and Security (AISec)*, 2015.

[12] D. J. Bernstein, "Curve25519: new Diffie-Hellman speed records," in *Proc. of the International Workshop on Public Key Cryptography (PKC)*, 2006.

[13] G. Bolch, S. Greiner, H. De Meer, and K. S. Trivedi, *Queueing networks and Markov chains: modeling and performance evaluation with computer science applications*. John Wiley & Sons, 2006.

[14] F. Brasser, D. Gens, P. Jauernig, A.-R. Sadeghi, and E. Stapf, "SANCTUARY: ARMing TrustZone with user-space enclaves," in *Proc. of the Network and Distributed System Security Symposium (NDSS)*, 2019.

[15] S. Brenner, C. Wulf, D. Goltzsche, N. Weichbrodt, M. Lorenz, C. Fetzer, P. Pietzuch, and R. Kapitza, "SecureKeeper: Confidential ZooKeeper using Intel SGX," in *Proc. of the International Middleware Conference (Middleware)*, 2016.

[16] Carbon Black, "Global incident response threat report," https://www.carbonblack.com/global-incident-response-threat-report/november-2018/, November 2018, last accessed 04-20-2019.

[17] C. Cimpanu, "Hackers are increasingly destroying logs to hide attacks," https://www.zdnet.com/article/hackers-are-increasingly-destroying-logs-to-hide-attacks/, last accessed 04-20-2019.

[18] B. Coppens, I. Verbauwhede, K. De Bosschere, and B. De Sutter, "Practical mitigations for timing-based side-channel attacks on modern x86 processors," in *Proc. of the IEEE Symposium on Security and Privacy (S&P)*, 2009.

[19] V. Costan, I. Lebedev, and S. Devadas, "Sanctum: Minimal hardware extensions for strong software isolation," in *Proc. of the USENIX Security Symposium (USENIX)*, 2016.

[20] S. A. Crosby and D. S. Wallach, "Efficient data structures for tamper-evident logging." in *Proc. of the USENIX Security Symposium (USENIX)*, 2009.

[21] dark laboratorys, "A better generation of logcleaners," https://web.archive.org/web/20070218231819/http://darklab.org/~jot/logclean-ng/logcleaner-ng_1.0_lib.html, last accessed 02-07-2020.

[22] T. T. A. Dinh, P. Saxena, E.-C. Chang, B. C. Ooi, and C. Zhang, "M2R: Enabling stronger privacy in MapReduce computation." in *Proc. of the USENIX Security Symposium (USENIX)*, 2015.

[23] M. Du, F. Li, G. Zheng, and V. Srikumar, "DeepLog: Anomaly detection and diagnosis from system logs through deep learning,"

in *Proc. of the ACM Conference on Computer and Communications Security (CCS)*, 2017.

[24] M. Dymshits, B. Myara, and D. Tolpin, "Process monitoring on sequences of system call count vectors," in *Proc. of the International Carnahan Conference on Security Technology (ICCST)*, 2017.

[25] European Parliament and of the Council, "Regulation (EU) 2016/679 on the protection of natural persons with regard to the processing of personal data and on the free movement of such data, and repealing directive 95/46/EC (General Data Protection Regulation)," *Official Journal of the European Union*, vol. L119, 2016.

[26] A. Ferraiuolo, A. Baumann, C. Hawblitzel, and B. Parno, "Komodo: Using verification to disentangle secure-enclave hardware from software," in *Proc. of the Symposium on Operating Systems Principles (SOSP)*, 2017.

[27] Fiehn Lab, "blast 2.7.1," http://fiehnlab.ucdavis.edu/staff/kind/collector/benchmark/blast-benchmark, last accessed 04-20-2019.

[28] B. Fisch, D. Vinayagamurthy, D. Boneh, and S. Gorbunov, "Iron: functional encryption using Intel SGX," in *Proc. of the ACM Conference on Computer and Communications Security (CCS)*, 2017.

[29] P. Gao, X. Xiao, D. Li, Z. Li, K. Jee, Z. Wu, C. H. Kim, S. R. Kulkarni, and P. Mittal, "SAQL: A stream-based query system for real-time abnormal system behavior detection," in *Proc. of the USENIX Security Symposium (USENIX)*, 2018.

[30] P. Gao, X. Xiao, Z. Li, F. Xu, S. R. Kulkarni, and P. Mittal, "AIQL: Enabling efficient attack investigation from system monitoring data," in *Proc. of the USENIX Security Symposium (USENIX)*, 2018.

[31] A. Gehani and D. Tariq, "SPADE: Support for provenance auditing in distributed environments," in *Proc. of the International Middleware Conference (Middleware)*, 2012.

[32] P. H. Gregory, *CISSP Guide to Security Essentials*, 2nd ed. Course Technology Press, 2015.

[33] R. A. Grimes, "Why it's so hard to prosecute cyber criminals," https://www.csoonline.com/article/3147398/why-its-so-hard-to-prosecute-cyber-criminals.html, last accessed 04-20-2019.

[34] T. C. Group, "ISO/IEC 11889-1:2009 information technology – trusted platform module – part 1: Overview," https://www.iso.org/standard/50970.html, 2009.

[35] Z. Gu, K. Pei, Q. Wang, L. Si, X. Zhang, and D. Xu, "LEAPS: Detecting camouflaged attacks with statistical learning guided by program analysis," in *Proc. of the Conference on Dependable Systems and Networks (DSN)*, 2015.

[36] A. Haeberlen, P. Kouznetsov, and P. Druschel, "PeerReview: Practical accountability for distributed systems," in *Proc. of the ACM Symposium on Operating Systems Principles (SOSP)*, 2007.

[37] S. Hales, "Last door log wiper," https://packetstormsecurity.com/files/118922/LastDoor.tar, last accessed 04-20-2019.

[38] K. Haniradi, "mig-logcleaner-resurrected," https://github.com/Kabot/mig-logcleaner-resurrected, last accessed 02-07-2020.

[39] T. D. Hanson, "tpl - a small binary serialization library for c," https://github.com/troydhanson/tpl, last accessed 04-20-2019.

[40] G. Hartung, "Attacks on secure logging schemes," in *Proc. of the International Conference on Financial Cryptography and Data Security (FC)*, 2017.

[41] G. Hartung, B. Kaidel, A. Koch, J. Koch, and D. Hartmann, "Practical and robust secure logging from fault-tolerant sequential aggregate signatures," in *Proc. of the International Conference on Provable Security (ProvSec)*, 2017.

[42] W. U. Hassan, S. Guo, D. Li, Z. Chen, K. Jee, Z. Li, and A. Bates, "NoDoze: Combatting threat alert fatigue with automated provenance triage," in *Proc. of the Symposium on Network and Distributed System Security (NDSS)*, 2019.

[43] S. He, J. Zhu, P. He, and M. R. Lyu, "Experience report: System log analysis for anomaly detection," in *Proc. of IEEE International Symposium on Software Reliability Engineering (ISSRE)*, 2016.

[44] J. E. Holt, "Logcrypt: Forward security and public verification for secure audit logs," in *Proc. of the Australasian Information Security Workshop (AISW-NetSec)*, 2006.

[45] M. N. Hossain, S. M. Milajerdi, J. Wang, B. Eshete, R. Gjomemo, R. Sekar, S. Stoller, and V. Venkatakrishnan, "SLEUTH: Real-time attack scenario reconstruction from COTS audit data," in *Proc. of the USENIX Security Symposium (USENIX)*, 2017.

[46] T. Hunt, Z. Zhu, Y. Xu, S. Peter, and E. Witchel, "Ryoan: A distributed sandbox for untrusted computation on secret data." in *Proc. of the USENIX conference on Operating Systems Design and Implementation (OSDI)*, 2016.

[47] IBM Knowledge Center, "Storage and analysis of audit logs," https://www.ibm.com/support/knowledgecenter/en/SSEPGG_11.1.0/com.ibm.db2.luw.admin.sec.doc/doc/c0052328.html, last accessed 04-20-2019.

[48] Intel, "Intel SGX data protections now available for mainstream cloud platforms - IT peer network," https://itpeernetwork.intel.com/sgx-data-protection-cloud-platforms/, last accessed 12-11-2019.

[49] Intel Corporation, "Intel Software Guard Extensions (Intel SGX) SDK," https://software.intel.com/en-us/sgx-sdk, last accessed 04-20-2019.

[50] ——, "Intel Software Guard Extensions programming reference," https://software.intel.com/sites/default/files/managed/48/88/329298-002.pdf, 2014, last accessed 04-20-2019.

[51] $(ISC)^2$, "Cybersecurity certification - CISSP, certified information systems security professional," https://www.isc2.org/Certifications/CISSP, last accessed 04-20-2019.

[52] M. Jarrett, M. Bailie, E. Hagen, and E. Etringham, "Prosecuting computer crimes," *United States. Department of Justice. Office of Legal Education*, 2010.

[53] M. Jarrett, M. Bailie, E. Hagen, and N. Judish, "Searching and seizing computers and obtaining electronic evidence in criminal investigations," *United States. Department of Justice. Office of Legal Education*, 2009.

[54] Y. Ji, S. Lee, E. Downing, W. Wang, M. Fazzini, T. Kim, A. Orso, and W. Lee, "Rain: Refinable attack investigation with on-demand inter-process information flow tracking," in *Proc. of the ACM Conference on Computer and Communications Security (CCS)*, 2017.

[55] Y. Ji, S. Lee, M. Fazzini, J. Allen, E. Downing, T. Kim, A. Orso, and W. Lee, "Enabling refinable cross-host attack investigation with efficient data flow tagging and tracking," in *Proc. of the USENIX Security Symposium (USENIX)*, 2018.

[56] JustLinux Forums, "server hacked!! /var/log deleted. how can i trace hacker!?!" http://forums.justlinux.com/showthread.php?123851-server-hacked-var-log-deleted-how-can-i-trace-hacker, last accessed 04-20-2019.

[57] V. Karande, E. Bauman, Z. Lin, and L. Khan, "SGX-Log: Securing system logs with SGX," in *Proc. of the ACM Asia Conference on Computer and Communications Security (ASIA CCS)*, 2017.

[58] K. Karen and S. Murugiah, "NIST special publication 800-92, guide to computer security log management," 2006.

[59] A. D. Keromytis, "Transparent computing engagement 3 data," https://github.com/darpa-i2o/Transparent-Computing, 2018, last accessed 12-12-2019.

[60] Keystone, "An open framework for architecting TEEs," https://keystone-enclave.org/, last accessed 04-20-2019.

[61] S. Kim, J. Han, J. Ha, T. Kim, and D. Han, "Enhancing security and privacy of Tor's ecosystem by using trusted execution environments," in *Proc. of the USENIX Symposium on Networked Systems Design and Implementation (NSDI)*, 2017.

[62] T. Kim, X. Wang, N. Zeldovich, and M. F. Kaashoek, "Intrusion recovery using selective re-execution," in *Proc. of the USENIX Conference on Operating Systems Design and Implementation (OSDI)*, 2010.

[63] S. T. King and P. M. Chen, "Backtracking intrusions," in *Proc. of the ACM Symposium on Operating Systems Principles (SOSP)*, 2003.

[64] S. T. King, Z. M. Mao, D. G. Lucchetti, and P. M. Chen, "Enriching intrusion alerts through multi-host causality." in *Proc. of the Network and Distributed System Security Symposium (NDSS)*, 2005.

[65] Y. Kwon, F. Wang, W. Wang, K. H. Lee, W.-C. Lee, S. Ma, X. Zhang, D. Xu, S. Jha, G. Ciocarlie *et al.*, "MCI: Modeling-based causality inference in audit logging for attack investigation," in *Proc. of the Network and Distributed System Security Symposium (NDSS)*, 2018.

[66] K. H. Lee, "Ubsi," https://github.com/kyuhlee/UBSI, last accessed 04-20-2019.

[67] K. H. Lee, X. Zhang, and D. Xu, "High accuracy attack provenance via binary-based execution partition." in *Proc. of the Symposium on Network and Distributed System Security (NDSS)*, 2013.

[68] ——, "LogGC: Garbage collecting audit log," in *Proc. of the ACM conference on Computer and Communications Security (CCS)*, 2013.

[69] J. Lind, C. Priebe, D. Muthukumaran, D. O'Keeffe, P.-L. Aublin, F. Kelbert, T. Reiher, D. Goltzsche, D. Eyers, R. Kapitza *et al.*, "Glamdring: Automatic application partitioning for Intel SGX," in *Proc. of the USENIX Annual Technical Conference (ATC)*, 2017.

[70] Linux Audit, "audit-userspace," https://github.com/linux-audit/audit-userspace, last accessed 04-20-2019.

[71] Y. Liu, M. Zhang, D. Li, K. Jee, Z. Li, Z. Wu, J. Rhee, and P. Mittal, "Towards a timely causality analysis for enterprise security," in *Proc. of the Symposium on Network and Distributed System Security (NDSS)*, 2018.

[72] K. Lucas, "UnixBench," https://github.com/kdlucas/byte-unixbench, last accessed 04-20-2019.

[73] D. Ma and G. Tsudik, "A new approach to secure logging," *ACM Transactions on Storage (TOS)*, vol. 5, no. 1, 2009.

[74] S. Ma, K. H. Lee, C. H. Kim, J. Rhee, X. Zhang, and D. Xu, "Accurate, low cost and instrumentation-free security audit logging for Windows," in *Proc. of the Annual Computer Security Applications Conference (ACSAC)*, 2015.

[75] S. Ma, J. Zhai, F. Wang, K. H. Lee, X. Zhang, and D. Xu, "MPI: Multiple perspective attack investigation with semantics aware execution partitioning," in *Proc. of the USENIX Security Symposium (USENIX)*, 2017.

[76] S. Ma, X. Zhang, and D. Xu, "ProTracer: Towards practical provenance tracing by alternating between logging and tainting." in *Proc. of the Network and Distributed System Security Symposium (NDSS)*, 2016.

[77] maldevel, "ClearLogs," https://sourceforge.net/projects/clearlogs/, last accessed 02-07-2020.

[78] S. Matetic, M. Ahmed, K. Kostiainen, A. Dhar, D. Sommer, A. Gervais, A. Juels, and S. Capkun, "ROTE: Rollback protection for trusted execution," in *Proc. of the USENIX Security Symposium (USENIX)*, 2017.

[79] S. Matetic, M. Schneider, A. Miller, A. Juels, and S. Capkun, "DelegaTEE: Brokered delegation using trusted execution environments," in *Proc. of the USENIX Security Symposium (USENIX)*, 2018.

[80] J. M. McCune, B. J. Parno, A. Perrig, M. K. Reiter, and H. Isozaki, "Flicker: An execution infrastructure for TCB minimization," *ACM SIGOPS Operating Systems Review*, vol. 42, no. 4, 2008.

[81] F. McKeen, I. Alexandrovich, A. Berenzon, C. V. Rozas, H. Shafi, V. Shanbhogue, and U. R. Savagaonkar, "Innovative instructions and software model for isolated execution." *Proc. of the International Workshop on Hardware and Architectural Support for Security and Privacy (HASP)*, 2013.

[82] L. McVoy, "LMbench," http://www.bitmover.com/lmbench/, last accessed 04-20-2019.

[83] S. M. Milajerdi, R. Gjomemo, B. Eshete, R. Sekar, and V. Venkatakrishnan, "HOLMES: Real-time APT detection through correlation of suspicious information flows," in *Proc. of the IEEE Symposium on Security and Privacy (S&P)*, 2019.

[84] K.-K. Muniswamy-Reddy, D. A. Holland, U. Braun, and M. I. Seltzer, "Provenance-aware storage systems," in *Proc. of the USENIX Annual Technical Conference (ATC)*, 2006.

[85] National Institute of Standards and Technology, "NIST special publication 800-53 (rev. 4), security controls and assessment procedures for federal information systems and organizations," 2013.

[86] National Security Agency, "Controlled access protection profile, version 1.d," https://www.niap-ccevs.org/Profile/Info.cfm?PPID=14&id=14, 1999.

[87] NGINX Inc., "Nginx 1.13.1," https://www.nginx.com/, last accessed 04-20-2019.

[88] D. Nguyen, J. Park, and R. Sandhu, "Adopting provenance-based access control in OpenStack cloud IaaS," in *Proc. of the International Conference on Network and System Security (NSS)*, 2014.

[89] H. Nguyen, B. Acharya, R. Ivanov, A. Haeberlen, L. T. X. Phan, O. Sokolsky, J. Walker, J. Weimer, W. Hanson, and I. Lee, "Cloud-based secure logger for medical devices," in *Proc. of the IEEE International Conference on Connected Health: Applications, Systems and Engineering Technologies (CHASE)*, 2016.

[90] OccupytheWeb, "How to cover your tracks & leave no trace behind on the target system," https://null-byte.wonderhowto.com/how-to/hack-like-pro-cover-your-tracks-leave-no-trace-behind-target-system-0148123/, 2013, last accessed 04-20-2019.

[91] O. Ohrimenko, F. Schuster, C. Fournet, A. Mehta, S. Nowozin, K. Vaswani, and M. Costa, "Oblivious multi-party machine learning on trusted processors," in *Proc. of the USENIX Security Symposium (USENIX)*, 2016.

[92] P. H. O'Neill, "Zero day exploits are rarer and more expensive than ever, researchers say," https://www.cyberscoop.com/zero-day-vulns-are-rarer-and-more-expensive-than-ever/, apr 2017.

[93] A. Oprea, Z. Li, T.-F. Yen, S. H. Chin, and S. Alrwais, "Detection of early-stage enterprise infection by mining large-scale log data," in *Proc. of the Conference on Dependable Systems and Networks (DSN)*, 2015.

[94] A. Ortiz Cornet and J. M. Bardera Bosch, "Method and system of generating immutable audit logs," Patent US20 090 016 534, 2009.

[95] J. Park, D. Nguyen, and R. Sandhu, "A provenance-based access control model," in *Proc. of the Annual International Conference on Privacy, Security and Trust (PST)*, 2012.

[96] T. Pasquier, X. Han, T. Moyer, A. Bates, O. Hermant, D. Eyers, J. Bacon, , and M. Seltzer, "Runtime analysis of whole-system provenance," in *Proc. of the ACM Conference on Computer and Communications Security (CCS)*, 2018.

[97] T. Pasquier, J. Singh, D. Eyers, and J. Bacon, "Camflow: Managed data-sharing for cloud services," *IEEE Transactions on Cloud Computing (TCC)*, 2015.

[98] K. Pei, Z. Gu, B. Saltaformaggio, S. Ma, F. Wang, Z. Zhang, L. Si, X. Zhang, and D. Xu, "HERCULE: Attack story reconstruction via community discovery on correlated log graph," in *Proc. of the Annual Computer Security Applications Conference (ACSAC)*, 2016.

[99] D. Pohly, S. McLaughlin, P. McDaniel, and K. Butler, "Hi-Fi: Collecting high-fidelity whole-system provenance," in *Proc. of the Annual Computer Security Applications Conference (ACSAC)*, 2012.

[100] C. Priebe, K. Vaswani, and M. Costa, "EnclaveDB: A secure database using SGX," in *Proc. of the IEEE Symposium on Security and Privacy (S&P)*, 2018.

[101] T. Pulls and R. Peeters, "Balloon: A forward-secure append-only persistent authenticated data structure," in *Proc. of the European Symposium on Research in Computer Security (ESORICS)*, 2015.

[102] H. Raj, S. Saroiu, A. Wolman, R. Aigner, J. Cox, P. England, C. Fenner, K. Kinshumann, J. Loeser, D. Mattoon *et al.*, "fTPM: A software-only implementation of a TPM chip," in *Proc. of the USENIX Security Symposium (USENIX)*, 2016.

[103] A. Rane, C. Lin, and M. Tiwari, "Raccoon: Closing digital side-channels through obfuscated execution." in *Proc. of the USENIX Security Symposium (USENIX)*, 2015.

[104] Rapid7, "Metasploit, the world's most used penetration testing framework," https://www.metasploit.com/, last accessed 04-20-2019.

[105] Red Hat Customer Portal, "System auditing," https://access.redhat.com/documentation/en-us/red_hat_enterprise_linux/6/html/security_guide/chap-system_auditing, last accessed 04-20-2019.

[106] Redis Labs, "Redis 3.0.6," https://redis.io/, last accessed 04-20-2019.

[107] R. Sailer, X. Zhang, T. Jaeger, and L. van Doorn, "Design and implementation of a tcg-based integrity measurement architecture," in *Proc. of the USENIX Security Symposium (USENIX)*, 2004.

[108] B. Schneier and J. Kelsey, "Cryptographic support for secure logs on untrusted machines." in *Proc. of the USENIX Security Symposium (USENIX)*, 1998.

[109] ——, "Secure audit logs to support computer forensics," *ACM Transactions on Information and System Security (TISSEC)*, 1999.

[110] F. Schuster, M. Costa, C. Fournet, C. Gkantsidis, M. Peinado, G. Mainar-Ruiz, and M. Russinovich, "VC3: Trustworthy data analytics in the cloud using SGX," in *Proc. of the IEEE Symposium on Security and Privacy (S&P)*, 2015.

[111] C. Shepherd, R. N. Akram, and K. Markantonakis, "EmLog: tamper-resistant system logging for constrained devices with tees," in *Proc. of the International Conference on Information Security Theory and Practice (WISTP)*, 2017.

[112] M.-W. Shih, M. Kumar, T. Kim, and A. Gavrilovska, "S-NFV: Securing NFV states by using SGX," in *Proc. of the ACM International Workshop on Security in Software Defined Networks & Network Function Virtualization*, 2016.

[113] S. Shinde, D. Tien, S. Tople, and P. Saxena, "Panoply: Low-TCB Linux applications with SGX enclaves," in *Proc. of the Network and Distributed System Security Symposium (NDSS)*, 2017.

[114] A. Sinha, L. Jia, P. England, and J. R. Lorch, "Continuous tamper-proof logging using TPM 2.0," in *Proc. of the International Conference on Trust and Trustworthy Computing (TRUST)*, 2014.

[115] Splunk Inc., "Splunk," https://www.splunk.com, last accessed 02-07-2020.

[116] P. Subramanyan, R. Sinha, I. Lebedev, S. Devadas, and S. A. Seshia, "A formal foundation for secure remote execution of enclaves," in *Proc. of the ACM Conference on Computer and Communications Security (CCS)*, 2017.

[117] SUSE Linux AG, "Linux audit-subsystem design documentation for Linux kernel 2.6, v0.1," http://uniforum.chi.il.us/slides/HardeningLinux/LAuS-Design.pdf, 2004.

[118] C. Tan, L. Yu, J. B. Leners, and M. Walfish, "The efficient server audit problem, deduplicated re-execution, and the web," in *Proc. of the Symposium on Operating Systems Principles (SOSP)*, 2017.

[119] The Apache Software Foundation, "ab 2.3," https://httpd.apache.org/docs/2.4/programs/ab.html, last accessed 04-20-2019.

[120] ——, "httpd 2.4.18," https://httpd.apache.org/, last accessed 04-20-2019.

[121] The MITRE Corporation, "Capec-81: Web logs tampering," https://capec.mitre.org/data/definitions/81.html, 2017, last accessed 04-20-2019.

[122] S. Tople and P. Saxena, "On the trade-offs in oblivious execution techniques," in *Proc. of the International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment (DIMVA)*, 2017.

[123] utds3lab, "sgx-log," https://github.com/utds3lab/sgx-log, last accessed 04-20-2019.

[124] Q. Wang, W. U. Hassan, D. Li, K. Jee, X. Yu, K. Zou, J. Rhee, Z. Chen, W. Cheng, C. Gunter, and H. Chen, "You are what you do: Hunting stealthy malware via data provenance analysis," in *Proc. of the Symposium on Network and Distributed System Security (NDSS)*, 2020.

[125] W. Wang, G. Chen, X. Pan, Y. Zhang, X. Wang, V. Bindschaedler, H. Tang, and C. A. Gunter, "Leaky cauldron on the dark land: Understanding memory side-channel hazards in SGX," in *Proc. of the ACM Conference on Computer and Communications Security (CCS)*, 2017.

[126] O. Weisse, V. Bertacco, and T. Austin, "Regaining lost cycles with HotCalls: A fast interface for sgx secure enclaves," in *Proc. of the Annual International Symposium on Computer Architecture (ISCA)*, 2017.

[127] A. A. Yavuz, P. Ning, and M. K. Reiter, "Efficient, compromise resilient and append-only cryptographic schemes for secure audit logging," in *Proc. of the International Conference on Financial Cryptography and Data Security (FC)*, 2012.

[128] A. A. Yavuz and P. Ning, "BAF: An efficient publicly verifiable secure audit logging scheme for distributed systems," in *Proc. of the Annual Computer Security Applications Conference (ACSAC)*, 2009.

[129] J. Yu, L. Hsiung, M. El Hajj, and C. W. Fletcher, "Data oblivious ISA extensions for side channel-resistant and high performance computing." in *Proc. of the Symposium on Network and Distributed System Security (NDSS)*, 2019.

[130] W. Zheng, A. Dave, J. G. Beekman, R. A. Popa, J. E. Gonzalez, and I. Stoica, "Opaque: An oblivious and encrypted distributed analytics platform." in *Proc. of the USENIX Symposium on Networked Systems Design and Implementation (NSDI)*, 2017.

[131] W. Zhou, Q. Fei, A. Narayan, A. Haeberlen, B. T. Loo, and M. Sherr, "Secure network provenance," in *Proc. of the ACM Symposium on Operating Systems Principles (SOSP)*, 2011.

Algorithm 7: Recovery Phase

Input: sealed-key-id

Unseal(sealed-key-id, $\langle$sk, mcID$\rangle$);
DestroyMC(mcID);
run Initialization Phase;

APPENDIX A
RECOVERY FROM ERRORS

Recall from Section V that if corrupted or stale data is provided as input to the Startup Phase, the Logger will raise an error. There exist multiple scenarios where this situation might happen: (1) the adversary, after achieving root access, shuts down the Logger and then erases, corrupts or replaces the sealed data stored on disk; (2) the adversary, after achieving root access, force kills the Logger causing it to skip the Shutdown Phase; (3) benign failures, such as power loss, system crash, or disk corruption, happen. We discussed why, in its current design, CUSTOS cannot distinguish between these different scenarios (they all cause an error at the next Startup, which will cause the generation of an alert at the next audit) and requires manual intervention in both the benign and the adversarial cases (Section XII). We now provide a discussion of how CUSTOS recovers from these errors.

Algorithm 7 presents the Recovery phase that CUSTOS provides to recover from errors. This phase takes as an input the sealed key and monotonic counter ID (sealed-key-id). Recall that these data were copied onto the administrative machine at the end of the Initialization Phase, and thus will not be lost, even if the adversary corrupted them on the compromised host. The enclave then unseals these data, destroys the monotonic counter ID associated with them by calling DestroyMC (cf. Table I) and runs the Initialization phase again, to create a new key-pair and monotonic counter. Finally, the administrator copies the new sealed-key-id (associated with the new key-pair and monotonic counter ID) onto their machine and registers the updated node's public key into the key management service. It is important to delete the previous monotonic counter before re-initializing CUSTOS' Logger in order not to reach the limit of monotonic counters available to SGX. Also observe that it is not in the adversary's interest to abuse the Recovery phase in an attempt to prevent detection (scenario 2), as it would result in the host utilizing an unregistered pk and this would still cause the generation of an alert at the next audit.

Once CUSTOS has been re-initialized, the integrity of the logs for which a signature was generated prior to the failure will remain verifiable using the previous public key. Future logs will be verifiable using the new public key. Only the logs which belonged to the current block at the time of the fault (in scenarios 2 and 3) will not have an associated signature, but the fact itself that CUSTOS generated an alert and the administrator had to intervene to re-initialize it with a new key pair and counter means that this tampering (whether benign or malicious) did not go undetected, and thus faults do not violate **G1** (tamper-evident logs).