Exploring granular flow integrity for interconnected trusted platforms

Adrian L. Shaw*
ARM Ltd
Cambridge, UK, CB1 9NJ
Email: als@arm.com

Hamza Attak
Hewlett Packard Enterprise
Bristol, UK, BS34 8QZ
Email: hamza@hpe.com

***Abstract*—Existing attestation solutions based on Linux Integrity Measurement Architecture treat the network as an untrusted input. Thus, they often employ strict access control mechanisms with tunneling policies to prevent network flows from tainting the system. However, these different access control policies are challenging for administrators to model and verify for different Linux deployments, making them difficult to deploy in practice. This complexity gets worse when installations may differ in behaviour.**

This paper discusses a novel method to bridge the gap between disparate information flow graphs and proposes a prototype of a new kernel-based network flow logger and attestation hooks. Results obtained show that the system impact is minimal in terms of system resources and is less complex to deploy due to reduced system complexity.

1. Introduction

It is well known that distributed systems security remains a relevant and challenging topic of research. A particularly difficult problem is being able to understand if remote systems have the presence of active malicious code, which could impersonate legitimate services. Fundamentally, there needs to be assurance that the commodity operating systems which systems are dependent on are behaving in expected ways. Without this assurance, a whole distributed system could be undermined, whether by hijacking functionality or through leakage of sensitive data.

A classic OS security issue is the confused deputy problem, where an innocent computer program (or process) is fooled into performing a privileged action on behalf of an unprivileged program, one which should be unauthorised. One way to tackle this issue is to monitor the interactions from the different processes and check for misuse by using a reference monitor. However, when services are *network facing*, they cannot be tracked easily without imposing heavy system restrictions (e.g. strict firewalling, VLAN tunneling configurations) that may hinder the deployment or functionality of a distributed system. Furthermore, these rules need to be created and verified by experts in those technologies, which is unfortunately an uncommon skill for computer experts.

Our intention in this paper is to present the design of a practical system that tracks and analyses all the interactions *across* a fixed distributed system in order to identify the source and impact of intrusions, thus gaining better end-to-end insight and assurance about internal service communications. It is critical that such tracking and monitoring remains transparent to services, such that no deployment or application modifications are needed. This work logs critical events using trusted computing techniques, process them on a remote and trusted verifier, and correlates the interactions between the system processes of the machines.

There is extensive research about establishing trust in systems, some of which has been included in standards developed by the Trusted Computing Group (TCG) [1]. The main principle of the TCG approach is that all software should be measured before executed and in a transitive fashion, starting from the first piece of code executed in the BIOS all the way to the boot loader and the OS bootstrap. Each component is responsible for measuring all executable code it loads, and is thus described as a chain of trust, with the first piece of code being the only element requiring implicit trust. All measurements are typically stored in a secure area and isolated from the main runtime. Typically, a hardware chip known as the Trusted Platform Module (TPM) [1] is used, which prevents tampering of measurements from malicious software. Since the TPM has finite storage, it is not feasible to store a potentially infinite number of measurements. Therefore, it includes a set of *platform configuration registers* (PCR), which when updated transitively includes the previous PCR value to calculate and store the result of a hash. Measurements have been extended to the operating system level [2] [3] but typically do not consider inter dependencies with other hosts on a trusted network, which is where our proposal tries to bring a practical model that can be deployed with ease.

The paper is structured as follows. Section 2 describes the fundamental concepts and challenges. Section 3 and 4 describe how information flows were captured at the host and network levels respectively. The impact of the proposed mechanisms are then evaluated in Section 5 with a discussion on the assumptions. The most relevant work is

**This work was done at Hewlett Packard Labs, Bristol, UK.*

discussed in Section 6 before ending with a summary.

2. Problem definition

Sailer *et al.* presented the Integrity Measurement Architecture (IMA) for Linux [2], which provides runtime integrity measurement for loaded Linux executable binaries. This technique is commonly known as binary attestation. Due to the large number of measurements requiring verification, it has often been described as not scalable due to the maintenance involved in managing a large whitelist database. However, there are two drawbacks with this simple approach. Firstly, creating a digest of a static binary does not necessarily capture the behaviour of the program at runtime. This behaviour can vary a lot based on the execution environment and on any data flows received. Secondly, since there is no easy distinction between important and benign binaries then all software must be measured, which may create an unnecessarily large measurement list that is full of meaningless events.

Since a UNIX-derived commodity mostly uses the filesystem for accessing resources, interactions are logged using a formal state transition model. This is for detecting which services access what resources at runtime. However, it should be noted that the commonly repeated phrase "everything is a file" should be taken with a pinch of salt, since it seems to ignore subsystems that are based on the system call interface. For instance, network level sockets are dynamically made through system calls, which result in a file descriptor corresponding to a dynamically generated file on the filesystem, but the content of the file descriptor itself is not meaningful to measure. Hence, not all files are equal, as some are just file descriptors. First of all it is important to explain the particular choice of model for this work's basis and then subsequently describe an approach to handle the case of network connections.

The Biba data integrity model [4] contrasts the Bell-LaPadula model [5], a formal model that focused on data-centric confidentiality. The Biba model is a formal state transition system that defines the notion of subjects with different integrity levels, where an information flow from a subject with a particular integrity level must never reach a subject of a higher integrity level.

One variant of Biba is the Clark-Wilson (CW) integrity model [6], it creates a more useable system by allowing low-integrity flows to reach higher integrity subjects provided that they go through a formally verified and trusted *transformation filter*. Due to the numerous technical and economical challenges in formally verifying software, an alternative and relaxed variant known as CW-Lite aims to be more practical by dropping the requirement of formal verification and leaving that task to a separate domain [7]. Instead of the formal validation, the model focuses on the trust relationships between components. The following data is required for each logged interaction in any Biba integrity model:

- **Subject**: the entity or role in the system (single entities, or a set of programs). This subject is assigned a subject label. The remote verifier decides which integrity level the label should correspond to. In our case, our verifier is simple - it looks up whether the subject is part of the TCB or not. Note that determining what a true set of TCB components for a given system is a challenge due to the number of possible permutations.
- **Object**: a resource in the system. Since the majority of resources on a UNIX inspired system can be accessed through a file system (virtual or physical), they can all be identified uniquely in the OS kernel. Each file system object is represented by a data structure called an *inode*, which can be used to hold a unique identifier.
- **Action**: the action performed on the inode, it could consist of a read, write, or an execute action. Bilateral reading and writing is also possible with "read-write", but it isn't encountered during this investigation.

This model leads to consider the formal definition of the Biba integrity model: *Given an object* o*, there is an information flow from the subject* **s1** *to* **s2** *if* **s1** *is able to write* o *and* **s2** *can read* o. Thus, if the subject, the object and the action are logged then a remote verifier can process the entries to build an information flow graph, populated by subjects as nodes and flows as edges. However, collected data and formulated graphs in most solutions have always been constrained to a single machine because the network is *untrusted*. However, today there are different ways to connect machines today that are considered trustworthy to some system owners, e.g. through isolated VLANs on software-defined network switches or TCP/IP networks over a rack-scale fabric back-plane. Prior assumptions about the network being untrusted has left a relatively unexplored space.

Applications use sockets to connect with other machines. Inputs from a network are classified as low integrity because the integrity of the remote host and the security of the associated connection must both be verified. To make matters worse, there are various distinct types of sockets, such as TCP/IP sockets, datagram sockets, UNIX sockets, and raw sockets. IMA was only designed to perform integrity measurement of inode objects, which do not contain enough metadata to fully describe high-level abstractions such as sockets. Hence, sockets measurement is out of the intended scope of IMA. In the original description of the Clark-Wilson integrity model, it is stated that low integrity inputs can only be sent to higher integrity subjects providing the inputs are sanitized by a trusted and verified *filter*. However, in practice, these filters can be both complex to write and cumbersome to set up, due to the skills required to distinguish input data as being either benign or malicious. For this reason, this work's approach tries to limit the dependence on filters. By giving machines transparent access to each other but under continuous monitoring, this can remove the need to actively mediate and influence the data flowing between them. Nevertheless, there is still a need to monitor regularly for integrity failures, as the TCB of any machine could

become untrusted at any point. That is why the aim is to correlate multiple logs (and their isolated information flow graphs) by labelling the network connections between the trusted machines. In order to achieve this, several required axioms are defined as follows:

- **A1**: Each system is running a kernel with the same TCB extensions and label type set, continuously logging the opening and closing events of network connections. The attestation agent is also included and available for serving requests.
- **A2**: Each outbound connection entry log includes the subject label which initiated the connection and a universally unique session identifier.
- **A3**: Each inbound connection entry log includes the universally unique session identifier and the label of the destination subject.
- **A4**: Each universally unique session identifier is sent with the packet data to enable A3.
- **A5**: Systems may only receive identifiers in network flows from other trusted systems.
- **A6**: The universally unique session identifiers must not be exposed prematurely to any of the subjects outside the TCB.
- **A7**: A system that does not respond with an attestation within a specified time interval must be considered untrusted to the verifier. Therefore, flows originating from the untrusted system must then be treated as low integrity.

One must consider carefully how to produce the unique identifier. A naive consideration would be to use time-stamping to associate connections. However, there is virtually no guarantee for fine-grained time synchronisation across distributed systems, and it should not be depended on for security purposes. Therefore, a different approach is needed. The approach used in this work requires that each machine keeps a tamper resistant log of all the information flows that cover reading and writing, in addition to the execution of files. This includes the subject which accessed the file, the object being accessed, and the action being performed. For a single machine, this creates an information flow graph of all the internal interactions. For labels, this work gets its inspiration from the type enforcement model first described in [8].

In order to consider network information flows between multiple machines, there needs to be a method to join multiple independent flow graphs into a single graph using typical graphing techniques. Particularly, this should show if any high integrity service on one machine receives an input from a low integrity client on another host. Since sockets are required for communication across the local network, their use should be logged. Both endpoints of the network communication should be logged also: the client socket, the destination server socket, and the respective subjects which initiated the communication. In order for the approach to be effective, it is assumed that all machines have the same logging enforcement mechanisms and have reliable communication with the verifier (i.e. in the same LAN).

The *auxiliary integrity log* (AIL) is introduced, it contains the data needed to validate the network flows. The information flows are first verified to see if an integrity violation occurred within the local system, and whether the AIL was affected by an integrity violation or not. If the AIL daemon accepts a direct input from an untrusted subject (or indirectly via tainted TCB elements) then the verifier knows that the AIL itself can no longer be trusted for determining system integrity.

A number of assumptions are made concerning the system attack surface and behaviour. Firstly, the kernel must be sufficiently trusted to preserve the integrity of the security modules and to create the required measurements correctly. Secondly, the system must employ a measured boot mechanism, providing assurance over the integrity of startup components such as the firmware and loadable modules. As is the case with all TPM-based platforms, this work also assumes that the hardware is secure and that there are no malicious hardware components, side-channels or physical attacks. This work also assumes a closed network of trusted platforms without man-in-the-middle attacks, which is not unreasonable in scenarios such as cloud computing or computer cluster environments. Finally, it is necessary for the verifier itself to be secure and trusted for performing continuous verification.

3. Implementing local information flow tracking

This section first discusses the basis for implementing the simple binary attestation, then describes the implemented basic Biba model for tracking information flows, and finally, introduces novel TPM call optimisations for reducing the burdening overhead of the PCR extend operation.

All the trusted processes ("*subjects*") on the systems must be strongly identified. To this extent, this work leverages the binary attestation capability of IMA by measuring all loaded executable programs on each system. These executables represent the subjects. An attestation agent, `attestd`, and a remote verifier are introduced, where the former sends the TPM quote and IMA's Stored Measurement Log (SML) to the verifier for analysis. The verifier contains a list of which subjects are critical to the security of the system, and are thus considered as *high integrity*. Any other subject is treated as *low integrity*.

Firstly, a list of acceptable code hashes for the platform is obtained by downloading and extracting all packages in the considered Linux distribution repositories for executable files, libraries and kernel modules. In this work, the verifier is built to handle the Debian distributions of Linux. The (i386 and amd64) packages produced 1.3 million measurements, covering all sub-distributions and occupying a modest 320MB of RAM. Secondly, these measurements were stored in Redis, an in-memory, and distributed key-value store. Any privileged service that does not come from the central repositories must be explicitly added to the database.

3.1. Information flow control

This implementation relies on the existing security module framework to log the semantics of interest. Note that, in this work, such a framework is only for its logging capabilities and not for actual access control. The Linux operating system allows for different MAC implementations as part of the general Linux Security Modules (LSM) framework [9]. Such works include well-known policy enforcement systems, such as SELinux [10] and AppArmor (previously known as SubDomain) [11]. SELinux was chosen for its type enforcement [8] capability for labeling security contexts at the object level as well as at the subject level, allowing for detailed LSM logging of who accessed a system resource, the type of resource, and the type of access. IMA is extended with a template that captures this information in each entry, consisting of approximately 200 lines of C code. Once the file system is labeled with types, the LSM data structures will contain the extended attributes created by the type information defined by the SELinux labeling system. The type information is what is required for local integrity logging, so this work takes full advantage of the existing LSM infrastructure.

3.2. Performance considerations

Acceleration through batching: Logging each file interaction is not without drawbacks. Performing an integrity measurement on each read and write event typically involves the TPM's PCR to record the event. However, the process of gathering measurements of the non-executable files themselves, which tend to be dynamic, is not very useful because there cannot be any predictable reference measurements to compare against. However, what is important is that the nature of the event is captured. The real threat is unknown software itself, and that read and write events of non-executable files do not need to be recorded by the TPM's PCR either until when new software is executed or when a remote attestation is requested. After some observations, it appears that reads and writes tend to occur in large bursts with typical applications.

The novel solution presented here is to buffer sequences of read and write events in the kernel measurement list until a new executable needs measuring. When the new executable is about to be measured, the aggregate hash of the series of read and write events in the measurement log is then used to extend the TPMs PCR register in order to protect the integrity of those log entries. Similarly, a request for remote attestation will also cause the buffer to be flushed. It should be noted that the threat model regarding IMA assumes the kernel and other TCB components are safe against runtime attacks such as buffer overflows.

Determining integrity of the system AIL: The AIL itself is created by the combination of SELinux and the commonly available `auditd` daemon. This is used for providing the remote verifier with the extra semantic data about connection information; therefore its integrity must be reported accurately against the presence of adversaries. However, measuring each appended entry to the audit log file would cause some performance degradation, since LSM hooks are in the kernel path. The main challenge of the audit log is that it is constantly being written to with an open file descriptor, so reading the log from another process will cause a Time-of-check-Time-of-Measurement (ToCToM) violation, since a reliable measurement cannot be performed. The type information is still required to perform labeling, which was taken from the open source reference policy provided by TresysTechnology [12]. The pre-defined labels simply give meaningful names to the various system files and are helpful for identifying the subjects in a more human readable manner (e.g. with labels such as `apache_t`, `openvswitch_t`, and so on). Note that this is used to label the file system for the LSM data. The access control rules are not needed, nullifying the need for extensive verification.

3.3. Continuous data-driven verification

The final element is the remote verifier, which must periodically monitor the different machines in the network. It must have a copy of all the public keys corresponding to the AIK, such that it can validate the TPM quote signatures for the remote attestation protocol. It then asks for a remote attestation and as a result, retrieves IMA's SML and the AIL. Since the verifier has no knowledge of the network topology with regard to the machines, the verifier must infer the direct network connections. This is done by processing the AIL log of each machine and looking for same label matches. Having fetched the IMA SML from each system, individual graphs are formed. Afterwards, the AILs of each machine are used to infer links that are not captured by IMA, producing a single graph showing inter-machines communications. The nodes can be annotated with their integrity level according to what is reported in the SML set. The basic CW-lite principles can then be used to calculate whether subjects on certain machines violated integrity and whether other local hosts were affected.

The full list of TCB subjects is not listed here and is subjected to the packages installed on the attesting systems. Establishing the list of trusted subjects is out of scope but is well covered in existing literature where approximately 30 subjects are identified for Linux systems [13] [14]. The presented work in this paper can provide a continual view of inter-system and intra-system information flows, which also allows for easy modification of the subject list without changing the policy on each attesting host. A simplified example is shown in Figure 1 as a graph based on the SML and AIL data of two hosts. Having discussed how to make information flow tracking feasible, the next section describes the different approaches to capture the semantics of the network connections between the distributed subjects.

4. Implementing connection tracking

The axioms defined in Section 2 need a suitable mechanism in order to track the network flows being used by the

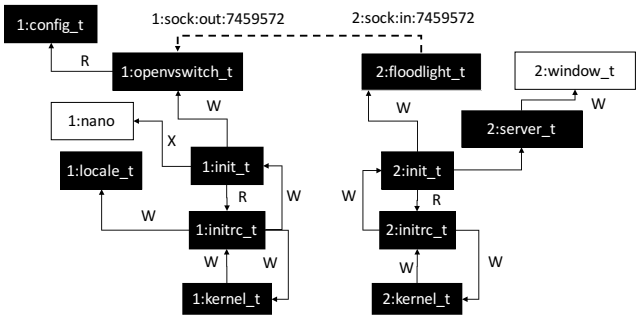

Figure 1. A simplified example of a directed dependency graph. Black elements are subjects within the distributed TCB. The white elements are subjects of lower integrity. Solid arrows show information flows reported by IMA (**R**ead, **W**rite, e**X**ecute) and the dashed arrows represent the network information flows reported by the AILs.

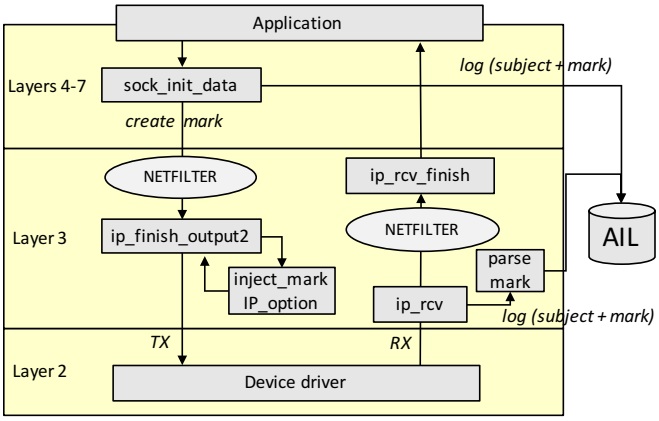

Figure 2. Simplified view of the Linux network stack with our marking subsystem hooks (egress and ingress) and their interaction with the AIL

subjects. Therefore, a "security mark" tied to each instantiated network socket is introduced. This mark is represented as a random unsigned 32-bit integer added to the sock kernel structure. This allows the sockets to be tracked over the local host. The mark is initialized in the `sock_init_data` function (see Figure 2), which instantiates the socket sock data structure in the kernel. The socket mark is generated as a pseudo-random value in the kernel space. The dynamic instantiation of the sockets in the Linux kernel is tracked through the *'socket'* and *'shutdown'* Linux system calls. The implementation relies on the audit framework to log both events and their associated socket's security mark. To meet axiom A1 described in Section 2, the mark must be transiting between trusted nodes with specific TCB extensions. This is achieved through packet marking. In fact, every packet transiting through a socket holds the sockets mark as part of its headers.

To achieve this goal, IP options [15] are leveraged in this work. Several IP options exists and are implemented as part of the Linux IP stack. In this context, there are two main selection criteria to pick the right IP option. Firstly, it has to implement at least a 32-bit field in which the secure mark should fit without triggering undesirable behavior by the current Linux kernel. The second criterion implies that the field is left unhandled by the IP option processing code in the Linux kernel, to be sure that it won't be removed or causing side effects. This second point is discussed later. Different solutions were studied, based on the US Department of Defense (DoD) basic security option [16] or the Commercial IP Security Option (CIPSO) [17]. Both of these IP options seemed relevant, but during tests it appeared that depending on the values put on their specific options fields, the kernel was dealing with some packets differently. For CIPSO, not having a Domain Of Interpretation (DOI) value defined on both the receiving and sending machines causes the connection to break and was quickly rejected with a specially formed ICMP packet. The same behavior happens with the DoD basic security option, if a datagram is received with no basic security option and the system security configuration parameters require an option field to hold the network port via which the datagram was received, then an ICMP parameter rejection packet will be transmitted.

The SECMARK solution in the Linux kernel [18] has also been considered, but unfortunately requires a specific SELinux rule for each network flow, which the flexibility goal of this work. Since a random value must be dynamically assigned to each new socket, the SECMARK solution cannot easily be used (due to having to update a static policy for each new value). In fact, it would also have added more configuration to attest on the system, and the SELinux policy constitutes a verification problem by itself.

The study of all these existing solutions and their restrictions led this work towards the creation of a new custom IP option, fitting exactly the needs of this paper's solution. As an IP option, the first byte represents the type, its value is picked among the non-reserved ones. Being also a mutlti-byte kind of IP option, the second byte is the total option length, then the security mark appears as a 4-byte field. By using a custom IP option type and subsequently, custom fields, the security marking also avoids adding unnecessary overhead of existing fields that are not meaningful to this work's objectives. For example, the CIPSO solution is represented by at least 10 bytes whereas the new security mark option is fixed at only 8 bytes. In fact, this new type adds the minimum overhead possible for a multi-byte IP option which is the type, the length and the necessary padding to reach a multiple of 8 bytes.

The new IP option is only managed by the new hooks, which therefore reduces the risk of having packets managed differently depending on the IP option value. However, there is one drawback from using a custom type. On some routers, unrecognized IP options either lead to options stripping or the packet being completely discarded. It is a vendor and firmware specific case, but is nevertheless an important factor to consider in deployment. This constitutes a risk in using IP options in general (and not only this new custom type), depending on the traversed router, its brand or even firmware version, there might be configurations set up to reject or ignore packets bearing IP options. However, this

has not been noticed while evaluating the implementation.

Since this approach requires marking every single packet going out of the host system, the marks are inserted at the IP layer to avoid misconfiguration at higher layers. Thus, the mark is inserted right at the end of the layer 3 processing, just before sending the packet to the device driver code in layer 2. Figure 2 depicts a simplified view of where the connection tracking hooks are in the IP kernel stack. Importantly, this figure shows how critical the placement of the marking code is. The function `ip_finish_output2` in Linux has the advantage of being one of the last egress functions of the IP stack. It is also located after all the NETFILTER hooks, which allows this solution to mark the packet without any regards to the current IPtables (routing and firewall) configuration on the host. The marking code is called from the beginning of this function. Fundamentally, it checks if there is enough headroom on the current packet to process, retrieves the security mark from the packets associated socket, injects the mark to the packet IP header with the custom IP option type and finally updates the checksum of the IP headers.

In order to retrieve the socket mark from the incoming packets (to meet axioms A1 and A3), the IP options are parsed by a custom code put in the very first function handling the incoming traffic in the Linux IP networking stack: `ip_rcv`. This function has the advantage of being right before the first ingress NETFILTER hook (`NF_INET_PRE_ROUTING`) and therefore avoids the packet from being modified or ignored by any subsequent routing and firewall rules from IPtables. The additional code for logging the incoming packets custom IP option simply verifies the IP option existence first, then directly accesses the IP option fields and logs the incoming mark in the AIL. A conceptually better location in the Linux kernel would be to place the hook in the `ip_rcv_options` function, as this function is already managing incoming packets that contain IP options. Unfortunately, `ip_rcv_options` is called in `ip_rcv_finish`, which is located after the very first NETFILTER hook occurring on the ingress traffic, and it would put the packet at risk of being modified by a netfilter rule. Both locations have been tested during the implementation, which exhibit similar performance impact. However, there is a small drawback. By logging the marks of packets before the routing stage then the system may log the marks of packets which are intended for a different destination. This detail can be solved by simply inspecting the destination address, but an elegant solution will require further investigation. Overall, the new network and AIL logging extensions added to the kernel have been minimal, requiring only approximately 200 lines of additional C code.

5. Evaluation and Discussion

In order to evaluate the presented solution a number of benchmarks were conducted to assess the overhead and impact of logging, the TPM overhead, verification delay when performing remote attestation, and the marking impact on bandwidth and latency.

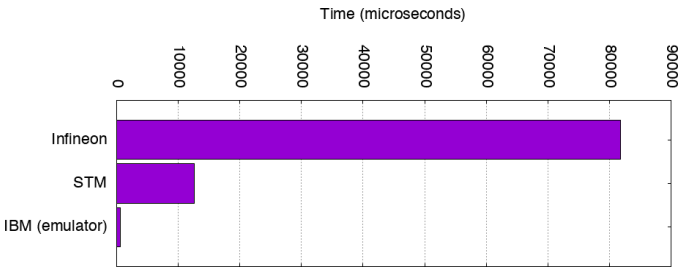

Figure 3. Average overhead of a PCR extension operation (based on 10,000 attempts) with two hardware TPMs (version 1.2) and a software-based TPM.

5.1. Remote attestation overhead

Figure 4 shows the performance of the remote attestations when having to process large line counts of IMA's SML. Network transfer time remains relatively constant and verification speed is approximately proportional to the line count of the IMA SML. The download time includes the log transfer time and the TPM quote operation, which shows to be fairly constant even with larger differential logs. The transfer of the AIL for network flow verification is comparable to the performance of receiving network logs of a typical web server application.

The evaluation platforms used to produce the test results are all equipped with TPMs manufactured by Infineon, which are an expected bottleneck in system performance, since a TPM is a slow device. A quick observation across two different TPM models shows that there is some variability in chip and device driver implementations. Figure 3 shows the average overhead produced by 10,000 TPM PCR extend operations. An open-source TPM emulator is also included in the comparison to show the speed improvements when no slow I/O operations are involved (executed on an Intel(R) Xeon(R) CPU X5650 @ 2.67GHz). The tests specifically measure the time taken to request a PCR extension via the open-source TCG Software Stack (TSS) API[1]. Therefore, saving unnecessary PCR extensions is important, which has been addressed with the batching approach described in Section 3.2.

As was mentioned earlier, not all files are equal. These experiments exclude a few Linux subsystems which are either not true files or just generally do not quite fit with the semantics of the integrity model, such as kernel interfaces in `sysfs`, `securityfs`, and `procfs`. Since the kernel is an element of the TCB, it must be trusted to filter these inputs correctly. In addition, `tmpfs` could not be accurately measured due to concurrency issues of multiple writers, resulting in a ToCToM violation. Since the AIL is implemented using the auditd package of Linux, a test was conducted to see the *worst case* impact if every event on the system is logged including socket connections and IPC calls. We noticed the worst case showed logging overhead could reach up to 20% overhead when in permissive mode. Therefore, foregoing SELinux permissive logging and using

1. Tested with the TrouSerS TSS: http://trousers.sourceforge.net/faq.html

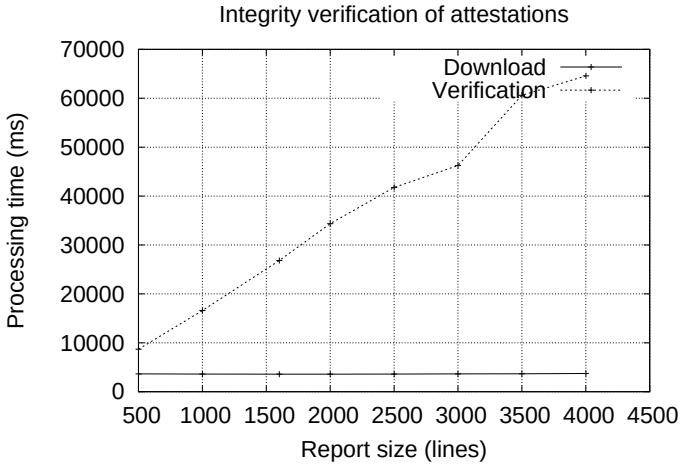

Figure 4. Performance graph of differential remote attestations, including network overhead, TPM overhead and verification speed.

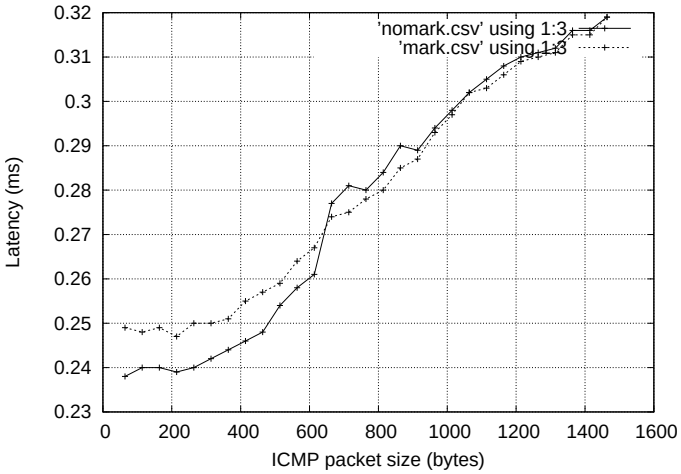

Figure 5. Average latency introduced when marking packets of different sizes (1000 ping probes each)

our own socket logging LSM hooks reached less than 2% worst case overhead with load on an Apache web server.

5.2. Network overhead

Figure 5 shows the latency increase for different packet sizes between marking and non-marking kernels. A single hop between two machines, both of which have the marking kernels (bi-directional marking), is used to obtain these results. As expected, for small packet sizes the overhead is explained by the static size of the new IP option. The kernel has to process the additional IP options for each packet, the cost of the options being fixed, it is therefore more evident with the smaller packet sizes in the benchmarks, and the difference slowly fades away with higher packet sizes. The few potential anomalies appearing on the figure (when the marking kernel appears to respond faster than the non-marking one) can be explained by the high variance of the `ping` command. For small packets, the variance between the two setup is clear and shows the difference depicted in Figure 5, but for bigger packets the variance of both graphs tends to converge. To assess bandwidth, the Linux `iperf` utility is used on a 1Gbit link with the default transfer size (1.06GBytes). The results show a difference of less than 1% when using the marking-enabled kernel compared to the non-marking one. The difference is constant for different TCP window sizes and shows constant time for when the kernel processes the fixed size marking IP option.

5.3. Software and replay attacks

Since every stage of the boot process and userspace programs are measured by the TPM then the integrity of the system can be reported. However, the dynamic data generated at boot time is out of scope of this paper. Note that through the proposed mechanisms, the presence of covert channels [19] cannot be detected and it is also not within the scope of this work. The remote attestation protocol requires a challenge from the verifier (also known as a nonce) to be included in the TPM Quote signature, which prevents replay attacks on the protocol. For the marked packets, the entropy and general strength of the random number generator must be reliable on each machine. Hardware-based random number generators are recommended to have better assurance about entropy generation. The choice of a 32-bit size representation for random numbers is what is offered by the Linux kernel API. Although a 32-bit random number seems sufficient to prevent replay attacks, this size could be adjusted depending on the strength requirement. Since this work focuses on software-based attacks, attacks over the wire are not considered, although this can easily be mitigated by using trusted channels to encrypt the communication and check the software posture of the remote peer. It is worth noting that this approach decouples the measurement system from the specific trusted channel mechanism, allowing flexibility for different trusted transport mechanisms, such as virtual LANs or the backbone of local clustered computers.

6. Related work

In particular, we note the most relevant work of Jaeger *et al.* who identified the minimal set of kernel dependencies to provide continuous runtime *enforcement* of the CW-Lite integrity model [3]. The completeness of the PRIMA solution is sound (it also mediates all types of IPC flows, more than network access and files) and is robust against attack by denying a bad access. While PRIMA reduces the number of measurements required for verifying the information flows of the TCB, it does not reduce the overall complexity of general integrity measurement. This is because a mandatory access control (MAC) policy must be carefully defined and applied in order to actively protect the critical subset of the operating system, particularly the flows that could affect the integrity of the TCB should be actively blocked. Identifying the dependencies for a given system does not appear to be

a simple task, as it requires a thorough understanding of the whole TCB design and the expected runtime behaviours. Any errors in defining the MAC policy or the occurrence of an unverified system upgrade could also cause system instability due to the blocking of necessary access requests from critical subjects, such as when a TCB component update is not accounted for. Sassu *et al.* [20] use IMA to track information flows but do not extend to the network. McCune *et al.* [21] presented Shamon, a system for distributed MAC across different physical systems. The proposed system involves deploying a special VM as a reference monitor on each host to mediate communications between different VMs. Each VM is equipped with a virtual TPM for remote attestation and the reference monitor only permits communication to hosts that satisfy particular integrity requirements. The system must enforce a MAC policy in the hypervisor as well as inside the VMs, meaning that all these policies need to be verified for correctness. Network traffic is restricted based on flow label and is locally enforced using IPSec gateways. The authors are optimistic that verification of the MAC policies is feasible (there were efforts [22]), but this seems impractical given the expert knowledge required to do so [23]. Our approach trades off robustness in favour of improving deployment flexibility.

7. Summary

This paper introduced a new framework for bridging information flows between hosts in a trusted network using TPM mechanisms. We find that prior trusted mechanisms are too dependent on access control policy enforcement, which are inflexible to TCB changes and still require significant expertise to verify. Through logging and centralised analysis, this approach: *A)* reduces the amount of bespoke policy verification needed for each host, *B)* avoids the need to update each machine policy when the TCB changes, and *C)* tolerates the absence of transformation filters if the incoming flow is of the same integrity level or higher as the receiver. We have shown that our proposed set of mechanisms have minimal impact on the host systems. Finally, by relaxing the trust assumptions of the network we are able to decouple this solution from particular transport mechanisms,

References

[1] G. Proudler, L. Chen, and C. Dalton, *Trusted Computing Platforms: TPM2.0 in Context*. Springer Publishing Company, Incorporated, 2015.

[2] R. Sailer, X. Zhang, T. Jaeger, and L. van Doorn, "Design and implementation of a tcg-based integrity measurement architecture," in *Proceedings of the 13th Conference on USENIX Security Symposium - Volume 13*, ser. SSYM'04. Berkeley, CA, USA: USENIX Association, 2004, pp. 16–16.

[3] T. Jaeger, R. Sailer, and U. Shankar, "Prima: Policy-reduced integrity measurement architecture," in *Proceedings of the Eleventh ACM Symposium on Access Control Models and Technologies*, ser. SACMAT '06. New York, NY, USA: ACM, 2006, pp. 19–28.

[4] K. J. Biba, "Integrity considerations for secure computer systems," *Technical Report MTR-3153*, Apr. 1977.

[5] L. J. LaPadula and D. E. Bell, "Mitre technical report 2547, volume ii," *J. Comput. Secur.*, vol. 4, no. 2-3, pp. 239–263, Jan. 1996.

[6] D. D. Clark and D. R. Wilson, "A comparison of commercial and military computer security policies," in *Security and Privacy, 1987 IEEE Symposium on*, April 1987, pp. 184–184.

[7] U. Shankar, T. Jaeger, and R. Sailer, "Toward automated information-flow integrity verification for security-critical applications," in *Proceedings of the 2006 Network and Distributed Systems Security Symposium*, Feb. 2006, pp. 267–280.

[8] W. E. Boebert and R. Y. Kain, "The extended access matrix model of computer security," *SIGSOFT Softw. Eng. Notes*, vol. 10, no. 4, pp. 119–125, Aug. 1985.

[9] C. Wright, C. Cowan, S. Smalley, J. Morris, and G. Kroah-Hartman, "Linux security modules: General security support for the linux kernel," in *Proceedings of the 11th USENIX Security Symposium*. Berkeley, CA, USA: USENIX Association, 2002, pp. 17–31.

[10] P. Loscocco and S. Smalley, "Integrating flexible support for security policies into the linux operating system," in *Proceedings of the FREENIX Track: 2001 USENIX Annual Technical Conference*. Berkeley, CA, USA: USENIX Association, 2001, pp. 29–42.

[11] C. Cowan, S. Beattie, G. Kroah-Hartman, C. Pu, P. Wagle, and V. Gligor, "Subdomain: Parsimonious server security," in *Proceedings of the 14th USENIX Conference on System Administration*, ser. LISA '00. Berkeley, CA, USA: USENIX Association, 2000, pp. 355–368.

[12] "SELinux Reference Policy," accessed: 2016-11-09. [Online]. Available: https://github.com/TresysTechnology/refpolicy

[13] T. Jaeger, R. Sailer, and X. Zhang, "Analyzing integrity protection in the selinux example policy," in *Proceedings of the 12th Conference on USENIX Security Symposium - Volume 12*, ser. SSYM'03. Berkeley, CA, USA: USENIX Association, 2003, pp. 5–5.

[14] T. Jaeger, X. Zhang, and A. Edwards, "Policy management using access control spaces," *ACM Trans. Inf. Syst. Secur.*, vol. 6, no. 3, pp. 327–364, Aug. 2003.

[15] J. Postel, "Internet protocol," Internet Requests for Comments, RFC Editor, STD 5, September 1981, http://www.rfc-editor.org/rfc/rfc791.txt. [Online]. Available: http://www.rfc-editor.org/rfc/rfc791.txt

[16] S. Kent, "U.s. department of defense security options for the internet protocol," Internet Requests for Comments, RFC Editor, RFC 1108, November 1991.

[17] I. C. W. Group, "Commercial IP Security Option (CIPSO 2.2)," Internet Engineering Task Force, Internet-Draft draft-ietf-cipso-ipsecurity-01, Aug. 1992, work in Progress. [Online]. Available: https://tools.ietf.org/html/draft-ietf-cipso-ipsecurity-01

[18] J. Morris, "New secmark-based network controls for selinux," 2016, accessed: 2016-11-09. [Online]. Available: http://james-morris.livejournal.com/11010.html

[19] M. Peter, M. Petschick, J. Vetter, J. Nordholz, J. Danisevskis, and J.-P. Seifert, *Undermining Isolation Through Covert Channels in the Fiasco.OC Microkernel*. Cham: Springer International Publishing, 2016, pp. 147–156.

[20] R. Sassu, G. Ramunno, and A. Lioy, "Practical assessment of biba integrity for tcg-enabled platforms," in *2014 IEEE 13th International Conference on Trust, Security and Privacy in Computing and Communications*, Sept 2014, pp. 495–504.

[21] J. M. McCune, T. Jaeger, S. Berger, R. Caceres, and R. Sailer, "Shamon: A system for distributed mandatory access control," in *2006 22nd Annual Computer Security Applications Conference (ACSAC'06)*, Dec 2006, pp. 23–32.

[22] B. D. Payne, R. Sailer, R. Cáceres, R. Perez, and W. Lee, "A layered approach to simplified access control in virtualized systems," *SIGOPS Oper. Syst. Rev.*, vol. 41, no. 4, pp. 12–19, Jul. 2007.

[23] Z. C. Schreuders, T. McGill, and C. Payne, "Empowering end users to confine their own applications: The results of a usability study comparing selinux, apparmor, and fbac-lsm," *ACM Trans. Inf. Syst. Secur.*, vol. 14, no. 2, pp. 19:1–19:28, Sep. 2011.