Abstract—Remote Attestation (RA) is a popular means of detecting malware presence on embedded and IoT devices. It is especially relevant to low-end devices that are incapable of protecting themselves against infection. Malware that is aware of ongoing or impending RA and aims to avoid detection can relocate itself during computation of the attestation measurement. In order to thwart such behavior, prior RA techniques are either non-interruptible or explicitly forbid modification of storage during measurement computation. However, since the latter can be a time-consuming task, this curtails availability of device’s other (main) functions, which is especially undesirable, or even dangerous, for devices with time- and/or safety-critical missions.

In this paper, we propose SMARM, a light-weight technique, based on shuffled measurements, as a defense against roving malware. In SMARM, memory is measured in a randomized and secret order. This does not impact device’s availability – the measurement process can be interrupted, even by malware, which can relocate itself at will. We analyze various malware behaviors and show that, while malware can escape detection in a single attestation instance, it is highly unlikely to avoid eventual detection.

1. Introduction

In the past, malware primarily targeted general-purpose computers and smartphones. In recent years, the number and variety of specialized computing devices has increased dramatically. This includes all kinds of embedded devices, cyber-physical systems (CPS) and Internet-of-Things (IoT) gadgets. They can be encountered in many diverse “smart” settings, such as home, office, factory, automotive and public venues. Unfortunately, these devices represent natural and attractive targets for malware, mainly because they are often numerous, connected to the Internet (and/or interconnected), and their security is poor or non-existent.

As society becomes increasingly accustomed to being surrounded by, and deriving benefits from, such devices, their well-being becomes a paramount concern. In the context of actuation-capable devices, malware can impact security and safety, e.g., as demonstrated by Stuxnet [18]. Whereas, for sensing devices, malware can undermine privacy by obtaining ambient information. Also, malware can turn vulnerable IoT devices into zombies that can become sources for DDoS attacks.

Security is typically not a priority for low-end device manufacturers, due to cost, size or power constraints, as well as the rush-to-market syndrome. It is thus unrealistic to expect low-end IoT devices to have the means to prevent malware attacks. The next best thing is detection of malware presence. This typically requires some form of Remote Attestation (RA) – a security service whereby a trusted entity (called Vrf) securely verifies the status of a remote device (called Prv) and detects malware presence on the latter. If malware is detected, Prv’s software can be re-set or rolled back and measures can be taken to prevent similar infections.

1.1. Remote Attestation (RA)

RA techniques generally operate as follows: Vrf sends an attestation request (that contains a challenge) to Prv. Once it receives the request, Prv invokes some trusted attestation code (AttC) that computes a measurement over Vrf-specified memory region M. This measurement (often called an attestation token) is typically realized as a Message Authentication Code (MAC) over M computed using a secret key K, which is protected from unauthorized access by the security architecture of Prv. (The goal of Prv’s underlying security architecture is to prevent malware Mal from forging measurements.) The measurement is then returned to Vrf which, in turn, determines whether Vrf is in a healthy or compromised state.

RA typically comes in three flavors: software-based, hardware-based and hybrid. Since this work is related to hybrid attestation, we refer to [1] for a discussion of the other two types. Hybrid attestation refers to RA techniques that aim to minimize hardware requirements on Prv, in part to be suitable for low-end single-core devices. Notable hybrid attestation architectures are: SMART [8], TrustLite [14], and TyTAN [3].

SMART is the first hybrid RA architecture. It stipulates that attestation code AttC and attestation key K are stored in ROM and guarded by hard-wired MCU access control rules.

1. About a year ago, hackers used a multitude of compromised “smart” cameras and DVRs to mount a massive-scale DDoS attack.
These rules enforce that only $\text{AttC}$ has access to $K$, and that $\text{AttC}$ is atomic, i.e., non-interruptible, and executed as a whole. Security properties of SMART are detailed in [10].

TrustLite differs from SMART in that interrupts are allowed and handled securely by the CPU Exception Engine. Also, access control rules can be programmed using an Execution-Aware Memory Protection Unit (EA-MPU). TyTAN further builds on TrustLite, notably featuring dynamic configuration of access control rules.

1.2. Motivation

Since low-end devices are often used in real-time and safety-critical applications, it is important to minimize the impact of security on normal operation (i.e., availability) of such devices. In particular, it might be undesirable to allow $\text{AttC}$ on $Prv$ to run without interruption, considering that computing a measurement over a substantial amount of memory might take a relatively long time. In other words, $\text{AttC}$ should be interruptible by a legitimate, time-critical application. However, $Prv$ might have been compromised and the same time-critical application might contain malware ($Mal$). $Mal$ presumably wants to evade detection. When confronted with attestation, it may want to simply erase itself, perhaps in order to reappear later. Alternatively, it might remain on $Prv$ and try to avoid detection by relocating itself during attestation. This behavior corresponds to roving malware.

Therefore, this paper focuses on reconciling two seemingly contradicting objectives: resistance against roving malware and minimizing the impact of attestation on $Prv$’s availability.

Prior hybrid RA designs had different motivations. SMART avoids roving malware by enforcing non-interruptibility of the measuring process, which fully sacrifices interruptibility by a time-critical task. Although TrustLite allows secure interrupts, it fails to detect roving malware. Whereas, TyTAN protects against roving malware by enforcing a rule that the process being measured cannot interrupt $\text{AttC}$, while other processes can. Nonetheless, this approach lacks transparency of attestation, i.e., if a time-critical process is being measured, it cannot interrupt the attestation process. Also, TyTAN might not detect roving malware if process isolation is compromised (e.g., by a kernel bug) resulting in process collusion. Furthermore, TrustLite and (to a lesser degree) TyTAN require more advanced hardware features than SMART, which translates into extra cost for low-end platforms.

2. SMARM: RA via Shuffled Measurements

To mitigate the conflict between roving malware ($Mal$) detection and critical mission of $Prv$, we adopt an approach whereby the attestation process is interruptible, while the order in which $M$ is measured is determined randomly and privately, by the attestation process. The rationale is that, if $Mal$ remains unaware of what portions of $M$ have been already measured (covered), it can not decide where to relocate itself to escape detection. $Mal$’s optimal strategy (i.e., where and when to relocate) depends on its knowledge of the attestation coverage. However, based on reasonable assumptions about security of the attestation architecture, we show that $Mal$ is detectable with significant probability.

Furthermore, since individual attestation instances are independent, the compound probability for $Mal$ to evade detection can be made negligible. As discussed in Section 6 below, although this increased level of security comes at the cost of running several measurements, no additional hardware features are needed. This results in the first low-cost and secure hybrid attestation technique that has no impact on $Prv$’s availability during attestation.

Shuffled (or random) memory coverage has been already suggested in the context of software-based attestation. However, it was done differently from SMARM, in several ways. First, random coverage of memory in software-based attestation is not secret, i.e., $Mal$ is fully aware of the sequence of memory blocks traversal. In contrast, SMARM assumes secrecy of this traversal pattern (shuffling), since it is generated based on $\forall rf$’s one-time challenge and a secret key shared by $\forall rf$ and $Prv$, which is inaccessible to $Mal$, as part of the underlying SMART architecture. Also, as described in [17], [16], [15], memory blocks are measured several times before all are processed at least once. This redundant coverage is likely due to size restrictions and non-optimizable constraints of $\text{AttC}$.

In the rest of this paper, in the context of $Prv$ implementing SMARM, we analyze different evasion strategies based on roving $Mal$’s varying degree of knowledge about the progress of shuffled measurements.

3. SMARM: Model and Assumptions

We assume that $Prv$’s memory is divided into $n$ blocks $M_1, ..., M_n$. We require $Prv$ to conform to the SMART architecture, as described in [8] and summarized above, augmented by an anti-DoS extension proposed in [4], which requires $Prv$ to maintain either a reliable read-only clock (ROC) or a monotonic secure counter. This is needed to authenticate and detect replayed or reordered $\forall rf$’s attestation requests. In addition, SMARM entails one important change with respect to SMART:

We relax the atomicity requirement of SMART such that the measurement process implemented by $\text{AttC}$ can be interrupted after it measures each memory block. Let $\sigma$ be a permutation randomly selected for a given attestation instance (measurement) of $M$. That is, block $M_{\sigma(i)}$ is measured at step $i$. Let $t_1, \ldots, t_n$ be the times at which blocks $M_1, \ldots, M_n$ are measured, respectively.

Let $M^*$ be $M$ in benign state. Let $R^* = F_K(M^*)$ be the measurement corresponding to the healthy state, computed by the measurement routine $F$ using key $K$. Finally, let $R$ be the measurement actually computed over $M$ in a given attestation instance.

2. Since the measurement process takes a non-negligible amount of time and since $M$ can change during that time, we can not write $R = F(M)$. 

We define a *benign measurement* as the event \( R = R^* \). We further define the probability of \( \text{Mal} \) evading detection with strategy \( S \) as:

\[
P_S = \Pr(R = R^* \mid M \neq M^*)
\]

We assume each \( M_i \) is measured under the same constraints as SMART. Namely, execution of \( \text{Attc} \) (over \( M_j \)) is atomic, access to \( K \) is protected, and memory is cleaned up (such that no traces of \( K \) remain) after the measurement of \( M_i \) is done.

Potential interruptions by other tasks running on \( Prv \) must thus be scheduled between the measurement of two blocks. Let \( t_{\text{max}} \) denote the “maximum non-interruptibility interval”. The size of a memory block \( M_i \) is set such that the time to measure it is at most \( t_{\text{max}} \).

4. Roving \( \text{Mal} \) Evasion Strategies

Since \( \text{Mal} \)’s goal is to avoid detection, it must restore each block where it resides to a benign state before that block is measured. This section considers the optimal strategy for \( \text{Mal} \) under various assumptions about its capabilities and knowledge, and the associated probability of detection.

We initially assume that \( \text{Mal} \) occupies a single block. The case where it resides in multiple blocks is discussed later in Section 5.3. Without loss of generality, we also assume that \( \text{Mal} \) is active during the entire attestation process, and can thus interrupt it and make changes to any block of \( M \) at any point as long as it does so between the intervals of attestation process measuring a single block. Reactive \( \text{Mal} \), and \( \text{Mal} \) with restricted number of interruptions are discussed in Sections 5.4 and 5.5 respectively.

4.1. Erasure

One trivial evasion strategy for \( \text{Mal} \) is to simply erase itself as soon as possible, perhaps to re-infect \( Prv \) at a later time. Assuming that \( \text{Mal} \) is aware of the incoming attestation request from \( Prf \), or it interrupts the attestation process before it starts (or early on during its execution), erasure seems difficult, if not impossible, to mitigate. In the rest of this paper, we focus on strategies whereby \( \text{Mal} \) attempts to remain on \( Prv \) while evading detection.

4.2. Relocation Techniques

Clearly, if \( \text{Mal} \) remains where it is, it cannot escape detection. Otherwise, it must relocate itself, at least once. We identify and explore three intuitive \( \text{Mal} \) flavors (which vary in the degree of knowledge) and their associated probabilities of successful evasion. As mentioned earlier, we assume below that \( \text{Mal} \) occupies a single memory block.

4.2.1. Knowledge of Future Volume (KFV). During attestation, the volume (size) of memory that has not yet been measured is the least amount of actionable information that \( \text{Mal} \) might have. This knowledge can be acquired by measuring the time elapsed since the start of attestation and estimating the number of memory blocks already measured. This is based on a realistic assumption that \( \text{Mal} \) is aware of: (1) time when attestation began, and (2) time to measure one memory block. We refer to this degree of knowledge as the KFV model.

**Theorem 1.** The optimal strategy for KFV \( \text{Mal} \) is to relocate after every memory block is measured. It would thus move a total of \( n-1 \) times, assuming that it can interrupt the attestation process at each block boundary. The probability of evasion is:

\[
P_{\text{KFV}} = \left(1 - \frac{1}{n}\right)^n \approx e^{-1} \approx 0.37
\]

**Proof.** Let \( M_{m_i} \), denote the block containing \( \text{Mal} \) at \( t_i \), for \( 1 \leq m_i \leq n \). Having the ability to interrupt the attestation process between \( t_i \) and \( t_{i+1} \), \( \text{Mal} \) can either stay put or relocate. Let \( p_i \) be the probability of \( \text{Mal} \) getting “caught” exactly at \( t_{i+1} \):

\[
p_i = \Pr(m_{i+1} = \sigma(i+1) \mid m_k \neq \sigma(k), 1 \leq k < i),
\]

for \( 1 \leq i < n \).

If \( \text{Mal} \) relocates, two outcomes may occur: either (1) its new location \( M_{m_{i+1}} \) has been already measured (there are \( i \) such blocks), in which case it will certainly not be caught, or (2) \( M_{m_{i+1}} \) was not measured yet (there are \( n-i \) such locations), in which case it will be caught with probability \( \frac{1}{n-i} \). Consequently:

\[
p_{\text{move}}^i = \frac{i}{n} \cdot 0 + \frac{n-i}{n} \cdot \frac{1}{n-i} = \frac{1}{n}.
\] (1)

If \( \text{Mal} \) does not relocate, two situations may occur. Let \( j \) be the last interval when \( \text{Mal} \) moved. (If it never moved, \( j = 0 \).) Again, \( M_{m_{i+1}} = M_{m_{j+1}} \) might have been already measured, in which case \( \text{Mal} \) will not be caught. This occurs with probability \( \frac{j}{n} \). Indeed, since \( M_{m_{j+1}} \) cannot have been measured in the last \( i-j \) steps (since we assume \( \text{Mal} \) has not been detected so far), it must have been measured in the \( j \) first ones.

Otherwise, if \( M_{m_{j+1}} \) has not been measured yet (which occurs with probability \( 1 - \frac{j}{n} \)), \( \text{Mal} \) will be caught with probability \( \frac{1}{n-i} \) (for the same reason as in Eq. [1]). Consequently,

\[
p_{\text{stay}}^i = \frac{j}{n} \cdot 0 + \frac{n-j}{n} \cdot \frac{1}{n-i} = \frac{n-j}{n(n-i)}.
\]

If \( j < i \), \( p_{\text{stay}}^i > p_{\text{move}}^i \). Therefore, it is always beneficial for \( \text{Mal} \) to relocate. Interestingly, the new location \( M_{m_{j+1}} \) does not matter, as long as \( m_{i+1} \neq m_i \). Relocation at each interval leads to an overall evasion probability of:

\[
P_{\text{KFV}} = (1 - p_i)^{n-1} \cdot \Pr(m_1 \neq \sigma(1)) = \left(1 - \frac{1}{n}\right)^n
\]

\( \Pr(m_1 \neq \sigma(1)) \) is the probability that the first block measured is not \( M_1 \). The approximation to \( e^{-1} \) results from the limit definition of \( e \). \( \square \)
4.2.2. Knowledge of Future Coverage (KFC). In addition to knowing how many blocks have been measured, Mal might also learn which blocks have been measured. That is, after \( t_1 \), Mal knows \( \{\sigma(1), \ldots, \sigma(i)\} \). Based on this, Mal can infer \( \{\sigma(i+1), \ldots, \sigma(n)\} \). This is different from knowing precise values \( \sigma(i+1), \ldots, \sigma(n) \); see Section 4.2.3 below. We refer to this as the KFC model.

This greater knowledge might stem from a side-channel vulnerability in the implementation of the attestation process, e.g., if measured blocks can be distinguished from non-measured ones from the point of view of Mal. Alternatively, it could be the result of the attestation process carelessly storing \( \{\sigma(1), \ldots, \sigma(i)\} \) (see Section 7).

**Theorem 2.** The optimal strategy for KFC Mal is to change its location to \( M_{\sigma(1)} \) immediately after \( t_1 \). The probability of evasion is:

\[
P_{FC} = 1 - \frac{1}{n}
\]

**Proof.** Since \( \sigma(1) \) is unknown before \( t_1 \), Mal can not make any informed decision, and thus would be discovered if \( n_1 = \sigma(1) \). After \( t_1 \), Mal knows that \( M_{\sigma(1)} \) was measured. Thus, it can safely relocate there and remain until the end. The result follows trivially: \( P_t(n_1 = \sigma(1)) = \frac{1}{n} \).

4.2.3. Knowledge of Future Order (KFO). In addition to future coverage, Mal might know \( \sigma(i+1), \ldots, \sigma(n) \), and thus also \( \{\sigma(1), \ldots, \sigma(i)\} \). Whether Mal also knows the past order of coverage \( \sigma(1), \ldots, \sigma(i) \) is irrelevant; see Section 5. Mal might acquire this additional knowledge due to \( \sigma(i+1), \ldots, \sigma(n) \) being insecurely stored by the attestation process; see Section 5. The leakage can also occur due to some vulnerability in the random block selection process, e.g., in case of a weak random number generation. We refer to this as the KFO model.

**Theorem 3.** The optimal strategy for KFO Mal is to relocate to any \( M_{j} \) with \( j \neq \sigma(1) \) before \( t_1 \) if \( M_{\sigma(1)} \) is its initial location, and then to \( M_{\sigma(1)} \) immediately after \( t_1 \). The probability of evasion is:

\[
P_{FO} = 1
\]

**Proof.** If \( \sigma(1) \) is located in \( \sigma(1) \) (the next block to be measured), it must relocate. After \( t_1 \), Mal knows that \( M_{\sigma(1)} \) was measured, so it can safely relocate there and remain to the end. Consequently, Mal must move once (or twice, if \( M_{\sigma(1)} \) is its initial location) in order to evade detection.

5. **Discussion**

We now discuss the viability of aforementioned malware flavors and several related issues.

5.1. **Realistic Scenarios**

We believe that KFV is quite realistic. It requires little insight and sophistication from Mal. It is viable even with a secure hybrid attestation framework and a secure handling of \( \sigma \). As discussed in Section 5.2 even when Mal’s timing estimates are inaccurate, it can attain the probability of evasion close to \( e^{-1} \). In contrast, KFC and KFO Mal flavors are more far-fetched, as long as \( \sigma \) is placed in secure storage, or stored elsewhere in an encrypted and authenticated form; see Section 7. Therefore, the rest of this paper focuses on KFV Mal.

5.2. **Unknown Timing**

Recall that KFV Mal is assumed to know \( (t_1, \ldots, t_n) \). This may not always be the case in practice. For instance, other active processes might distort the timing, or spurious delays might be introduced by the AttC, as a potential countermeasure.

One possible strategy for Mal is to relocate as often as possible. As follows from the proof of Theorem 2, as long as Mal relocates at least once between two blocks being measured, the predicted probability of successful evasion is reached. While precise values for \( (t_1, \ldots, t_n) \) may not be known, Mal can easily determine the lower bound on the time to measure one block. If Mal relocates with a frequency of that lower bound, it can attain the evasion probability of \( e^{-1} \).

5.3. **Mal in Multiple Blocks**

Despite the earlier assumption, Mal might not fit in a single memory block. Let \( s \) denote the number of blocks occupied by KFV Mal. Its evasion probability can be viewed as evasion by \( s \) separate single-block pieces of malware, with the probability of:

\[
P_{FV}(S = s) = P_{FV}(S = 1)^s \approx e^{-s}.
\]

5.4. **Active vs Reactive Malware**

So far, we assumed that Mal is active throughout the attestation process. We now broaden this to include Mal which wakes up (becomes active) at some point during attestation. Let \( w \) be the number of memory blocks measured thus far. Let \( P_{active}(n, s) \) be the probability of evasion of an active Mal of size \( s \) for a given strategy on measurement of \( n \) blocks. The probability \( P_{reactive}(n, s, w) \) of Mal following the same strategy reacting after \( w \) blocks is:

**Theorem 4.**

\[
P_{reactive}(n, s, w) = \binom{n-s}{w} \frac{n-s}{n} P_{active}(n-w, s)
\]

**Proof.** The \( \binom{n-s}{w} \) factor is the probability of none of Mal’s \( s \) blocks being chosen in first \( w \) measurements. It is a special case of a hypergeometric distribution. The second factor is the probability that an active Mal escapes detection, with \( n - w \) blocks remaining.
It follows trivially that $P_{\text{reactive}}(n, s, 0) = P_{\text{active}}(n, s)$. Figure 1 shows the effect of $\mathcal{M}_{\text{al}}$ that is only active after $w$ steps.

5.5. Limited # of Interruptions

We now consider the case when KFV $\mathcal{M}_{\text{al}}$ is limited to at most $k$ interruptions during the entire attestation process instance. This might be motivated by $\mathcal{M}_{\text{al}}$ aiming to reduce its timing footprint. Indeed, one possible countermeasure for the attestation process (with SMART) that suspects roving $\mathcal{M}_{\text{al}}$ presence is to measure its total elapsed (wall-clock) time. If it diverges significantly from the expected time, and if there is no legitimate justification, it can be an indication of activity by frequently relocating $\mathcal{M}_{\text{al}}$.

**Theorem 5.** The optimal strategy for KFV $\mathcal{M}_{\text{al}}$ limited to $k$ interruptions is to change its location after each group of $n/(k + 1)$ blocks is measured. It thus moves $k$ times and its probability of evasion is:

$$P_{\mathcal{P}_{\mathcal{F}}}(K = k) = \left(1 - \frac{1}{k + 1}\right)^{k+1}$$

**Proof.** See Appendix A.

6. Reliable Detection

Using SMART, the probability that one measurement does not detect $\mathcal{M}_{\text{al}}$ presence is non-negligible. However, if multiple measurements are taken, the overall false negative probability decreases exponentially. Multiple measurements can be obtained via: (1) independent consecutive attestation instances, (2) batch mode, e.g., Vrf sends $m$ challenges at once and receives $m$ measurements, or (3) self-measurements by $\mathcal{P}_{\mathcal{F}}$ itself, as described in [5]. Given $m$ independent measurements, each with probability $P$ of a false negative, the overall false negative probability is $P_{\mathcal{F}} = P^m$. That is, with $P = e^{-1}$ this gives, e.g., $P_7 < 10^{-3}$, and $P_{13} < 10^{-6}$.

7. Block Permutation in Practice

This section discusses a trial implementation of random block shuffling and its security implications.

7.1. Permutation Computation and Storage

Recall that the random one-time permutation $\sigma$ is computed at the start of the attestation process, prior to measuring any memory blocks. One way to compute it efficiently is by using the well-known Fisher-Yates shuffle method.

$\mathcal{P}_{\mathcal{F}}$’s PRNG is seeded with $H(c, K)$ where $c$ is Vrf’s challenge, $K$ is the key securely stored by $\mathcal{P}_{\mathcal{F}}$ (as part of SMART), and $H$ is a hash function. As mentioned in Section 5 we assume that $\mathcal{P}_{\mathcal{F}}$ authenticates each attestation request. Once an attestation request is validated, $\mathcal{P}_{\mathcal{F}}$ knows that $c$ (contained therein) is unique; hence, $H(c, K)$ is unique as well. This guarantees that random values $\sigma_i$ produced by PRNG are both fresh and secret, i.e., unpredictable by $\mathcal{M}_{\text{al}}$.

Once computed, $\sigma = \{\sigma(1), \ldots, \sigma(n)\}$ is stored in secure memory. The underlying security architecture ensures that (similar to $K$), $\sigma$ can be written and read only by $\mathcal{A}_{\text{ttC}}$. This requires $n \log n$ bits of storage. When the measurement of block $M_{\sigma(i-1)}$ is completed (or when attestation starts at $i = 1$), $\sigma(i)$ is read and $M_{\sigma(i)}$ is fed to the MAC function. Once all blocks are measured, $\sigma$ remains in secure storage. It is then over-written at the start of the next attestation instance.

If the size of the additional secure storage is a concern, the following variant is an alternative. The permutation $\sigma$ is instead stored encrypted and authenticated in insecure storage, and no additional secure storage is required. Each index $\sigma(i)$ is encrypted individually as a block of the block cipher. Since efficiency is a concern, the block cipher operates in CTR mode [6], so that random access to individual blocks is efficient. This requires $n \log n$ bits of storage. When the measurement of block $M_{\sigma(i-1)}$ is completed (or when attestation starts at $i = 1$), $\sigma(i)$ is read and $M_{\sigma(i)}$ is fed to the MAC function. Once all blocks are measured, $\sigma$ remains in secure storage. It is then over-written at the start of the next attestation instance.

If the size of the additional secure storage is a concern, the following variant is an alternative. The permutation $\sigma$ is instead stored encrypted and authenticated in insecure storage, and no additional secure storage is required. Each index $\sigma(i)$ is encrypted individually as a block of the block cipher. Since efficiency is a concern, the block cipher operates in CTR mode [6], so that random access to individual blocks is efficient. This requires $n \log n$ bits of storage. When the measurement of block $M_{\sigma(i-1)}$ is completed (or when attestation starts at $i = 1$), $\sigma(i)$ is read and $M_{\sigma(i)}$ is fed to the MAC function. Once all blocks are measured, $\sigma$ remains in secure storage. It is then over-written at the start of the next attestation instance.

7.2. Memory Overhead

We estimate memory overhead for storing $\sigma$ on IMX6-SabreLite [2], a popular and inexpensive development board representative of low- to medium-end IoT devices.
As discussed in Section 3, the maximum non-interruptibility interval \( t_{\text{max}} \) influences block size and their number. Given total memory size \( |M| \) and throughput \( \theta \), we generate the random permutation \( \sigma \) using the Fisher-Yates shuffle method, discussed in Section 7. The PRNG is implemented using AES-256 in CTR Mode, seeded with SHA-256. In the variant where secure storage is not available, we implement the underlying block cipher and MAC function as AES-128 and SHA-256-based HMAC respectively.

8. HYDRA Implementation

8.1. Overview

We implemented \( \text{SMARM} \) in the context of HYDRA on an I.MX6-SabreLite. HYDRA implements a hybrid RA design for devices with a Memory Management Unit (MMU). It builds upon the formally verified seL4 microkernel, which ensures process memory isolation and enforces access control to memory regions. Using access control rules can be implemented in software and enforced by the microkernel.

HYDRA stores \( K \) and \( \text{AttC} \) in a writable memory region and configures the system such that no other process, besides the measurement process, can access these memory regions. Access control configuration in HYDRA also involves the measurement process having exclusive access to its thread control block as well as to memory regions used for key-related computations. The measurement process starts with the highest maximum controlled priority (MCP). This ensures execution of the measurement process will run uninterruptedly as long as its “priority” remains the highest. Note that, in seL4, a priority is the effective priority of a process, which can be increased and decreased at run-time as long as it does not exceed its MCP value. In contrast, a process cannot increase its MCP after it is set (but a decrease is possible).

The formally verified version of seL4 uses a simple preemptive round-robin scheduler. Processes with the same priority take turns to execute for the same time-slice of length \( t_{\text{slice}} \), unless pre-empted by a higher-priority process.

The measurement procedure in the HYDRA-based \( \text{SMARM} \) implementation is as follows. At the start of a time-slice, the measurement process is set to highest priority (so that it can not be interrupted). Once the measurement of the block is done (this takes \( t_{\text{max}} \) seconds), the measurement process decreases its priority and yields (via the \text{seL4_Yield} system call) its remainder of the time-slice. This allows another process (which may include \( \text{Mal} \)) to run immediately after each block is measured with a fresh time-slice. Once the next time-slice of the measurement process begins, the next block is measured.

8.2. Experimental Results: \( \text{SMARM} \) with/without Secure Storage

We generate the random permutation \( \sigma \) using the Fisher-Yates shuffle method, discussed in Section 7. The PRNG is implemented using AES-256 in CTR Mode, seeded with SHA-256. In the variant where secure storage is not available, we implement the underlying block cipher and MAC function as AES-128 and SHA-256-based HMAC respectively.

4. Technically, each interrupting process, besides the measurement process, is allowed one time-slice (round-robin scheduling). However, for simplicity we assume a single (possibly infected) such process performing \( \text{Prv} \)’s main task.
Figure 4: Runtime of SMARM with/without secure storage for $\sigma$ as a factor of $n$. $|M|$ is fixed to 256MB. It is also assumed that no other process exists besides the measurement process.

Figure 4 shows overall runtime of these two variants (i.e. SMARM with/without secure permutation storage) compared to that of the baseline measurement. With enough secure storage, SMARM incurs 41% overhead over the baseline measurement with $n = 2048$ (or $t_{\text{max}} = 0.006s$). This overhead decreases as $n$ decreases; it achieves 0.8% at $n = 32$ (or $t_{\text{max}} = 0.4s$). On the other hand, without secure storage for $\sigma$, SMARM spends significant amount of time performing additional encryptions and MAC computations. The overhead in this case can be as high as 10,636% (at $n = 2048$) and as low as 2% (at $n = 1$). Thus, it might be more beneficial to deploy a lower number of memory blocks if a device does not have room for secure storage.

8.3. Experimental Results: Different $t_{\text{slice}}$ and $t_{\text{max}}$

Figure 5 illustrates the worst-case runtime of SMARM on this HYDRA implementation, compared to the uninterrupted measurement of $M$, as a factor of $t_{\text{slice}}$, $t_{\text{max}}$, and $|M|$. The overall SMARM runtime: (1) increases as $t_{\text{slice}}$ increases, and (2) decreases as $t_{\text{max}}$ increases. We found that the overhead can be up to a factor of $1 + \frac{t_{\text{slice}}}{t_{\text{max}}}$, while another process can run for up to $t_{\text{slice}}$ seconds every $t_{\text{slice}} + t_{\text{max}}$ seconds.

9. Conclusion

This paper presents SMARM, an attestation technique based on measuring $P_{rv}$’s memory in a random and secret order.

Contrarily to most hybrid attestation schemes, the measurement process can be interrupted and the attested memory modified. This makes attestation having no practical impact on the availability of $P_{rv}$ for its main operation. This is a requirement in time- and/or safety-critical applications.

Roving malware normally presents a threat to interruptible measurement over malleable memory. However, since in SMARM the memory is measured in unpredictable and secret order, malware can not make informed decisions to escape detection with certainty.

Using a secure framework, such as HYDRA, allows to keep the permutation unknown to potential malware. This caps the evasion probability to $e^{-1} \approx 37\%$. Since each attestation session is independent, the probability that a piece of malware present on $P_{rv}$ consistently escapes detection after multiple attestation sessions drops exponentially. After 13 checks, that probability is below $10^{-6}$.

SMARM thus results in: (1) increased time to attain a negligible probability of false negatives, and (2) additional memory to store the current permutation, as compared to
techniques using deterministic coverage order with non-interruptibility or non-malleability. However, to the best of our knowledge, SMARM is the only RA technique that mitigates roving malware for time- and/or safety-critical applications.

SUPPORT: This work was supported in part by DHS, under subcontract from HRL Laboratories, and ARO under contract: W911NF-16-1-0536, as well as NSF WiFiUS Program Award #: 1702911.

References


Appendix

Proof of Theorem 3. Theorem 1 showed that relocating is always preferable for Mal to remaining in the same block. What remains to determine is when to optimally make theses interruptions.

Let \( n_1, \ldots, n_k \) be the number of blocks measured before each interruption, and let \( n_{k+1} = n - \sum_{i=1}^{k} n_i \) be the remaining blocks after the last interruption. The probability for Mal to escape detection is thus:

\[
P = \prod_{i=1}^{k+1} \left( 1 - \frac{n_i}{n} \right)
\]

Let \( P \) be the probability pertaining to the strategy \( n_1, \ldots, n_k \). Let an alternative strategy \( n_1', \ldots, n_k' \) (and the corresponding probability \( P' \)) be defined as:

\[
n_i' = \begin{cases} n_a + c & \text{if } i = a \\ n_b - c & \text{if } i = b \\ n_i & \text{else,} \end{cases}
\]

for \( a \neq b : n_a \geq n/(k+1), n_b \leq n/(k+1) \). Such a pair is guaranteed to exist because \( n_{k+1} = n - \sum_{i=1}^{k} n_i \). We thus have:

\[
P' = \frac{(1 - \frac{n_a}{n}) \left( 1 - \frac{n_b}{n} \right)}{(1 - \frac{n_a + c}{n}) \left( 1 - \frac{n_b - c}{n} \right)}
= \frac{(1 - \frac{n_a}{n}) \left( 1 - \frac{n_b}{n} \right)}{(1 - \frac{n_a}{n}) \left( 1 - \frac{n_b}{n} \right) + \frac{c}{n} \left( \frac{n_a}{n} - \frac{n_b}{n} \right) - \frac{c^2}{n^2}}.
\]

Since \( n_a \geq n_b \), the denominator is smaller than the numerator, and thus \( P \geq P' \). This shows that any strategy that diverges from \( n_i = n/(k+1) \) will be sub-optimal.

\[\square\]