ERASMUS: Efficient Remote Attestation via Self-Measurement for Unattended Settings

Xavier Carpent
UC Irvine
xcarpent@uci.edu

Norrathep Rattanavipanon
UC Irvine
nrattana@uci.edu

Gene Tsudik
UC Irvine
gene.tsudik@uci.edu

***Abstract*—Remote attestation (RA) is a popular means of detecting malware in embedded and IoT devices. RA is usually realized as a protocol via which a trusted *verifier* measures software integrity of an untrusted remote device called *prover*. All prior RA techniques require *on-demand operation*. We identify two drawbacks of this approach in the context of unattended devices: First, it fails to detect *mobile malware* that enters and leaves the prover between successive RA instances. Second, it requires the prover to engage in a potentially expensive computation, which can negatively impact safety-critical or real-time devices.**

To this end, we introduce the concept of *self-measurement* whereby a prover periodically (and securely) measures and records its own software state. A verifier then collects and verifies these measurements. We demonstrate a concrete technique called ERASMUS, justify its features, and evaluate its performance. We show that ERASMUS is well-suited for safety-critical applications. We also define a new metric – *Quality of Attestation* (QoA).

I. Introduction

In recent years, embedded and cyber-physical systems (CPS), under the guise of *Internet-of-Things (IoT)*, entered many aspects of daily life, such as: homes, office buildings, public venues, factories and vehicles. This trend of computerizing previously analog devices and then inter-connecting them brings many obvious benefits. However, it also greatly expands the *"attack surface"* and turns these newly computerized gadgets into natural and attractive attack targets. As recent incidents demonstrated (e.g., Mirai [2] and Reaper[1]), IoT devices can be infected with malware and used as bot-controlled *zombies* in Distributed Denial-of-Service (DDoS) attacks. Also, IoT-borne malware can snoop on device owners (by sensing) or maliciously control critical services (by actuation), as happened with Stuxnet [17].

One key component in securing IoT devices is malware detection, typically attained via Remote Attestation (RA). RA is a distinct security service that allows a trusted party, called *verifier*, to securely verify the internal state (including memory and storage) of a remote untrusted and potentially malware-infected device, called *prover*. RA is usually realized via a protocol between prover and verifier. A typical example is described in [5]: (1) verifier sends an attestation request to prover, (2) prover verifies the request[2] and (3) computes a cryptographic function of its internal state, (4) sends the result to verifier, and finally, (5) the latter checks the result and decides whether prover is infected.

This general approach represents *on-demand attestation* and all current RA techniques adhere to it. In this paper, we identify its two important limitations: First, it is a poor match for unattended devices, since malware that "comes and goes" (i.e., mobile malware [12]) can not be detected if it leaves prover by the time attestation is performed at verifier's demand. Second, for a device working under time constraints (real-time or safety-critical operation), on-demand RA requires performing a time-consuming task while deviating from prover's main functions.

Motivated by this, we design ERASMUS: **E**fficient **R**emote **A**ttestation via **S**elf-**M**easurement for **U**nattended **S**ettings. ERASMUS is based on *self-measurements*. In it, a device (prover) measures its state at scheduled times. Measurements are stored in prover's *insecure* memory. Verifier occasionally collects and validates these measurements in order to establish the *history* of prover's state. Notably, with this general approach, verifier imposes negligible real-time burden on prover. ERASMUS also offers better quality-of-service than prior RA techniques, since verifier obtains prover's entire measurement history, since the last request. In other words, ERASMUS decouples (1) frequency of verifier's requests from (2) frequency of prover's measurements, which are equivalent in on-demand RA. Finally, ERASMUS simplifies RA design for prover: authentication of verifier requests is no longer needed, since computational DoS attacks do not apply.[3] We also introduce a new notion of *Quality of Attestation* (QoA) which captures: (1) how a device (prover) is attested, (2) how often its state is measured, and (3) how often these measurements are verified.
FULL PAPER: The extended version of this paper [6] contains more detailed discussions of: self-measurement techniques, applications to swarm/group settings, authenticated erasure, as well as ERASMUS prototype implementations and experimental results.
NOTE: ERASMUS is not intended as a replacement for on-demand RA, mainly because, for some devices and some settings, real-time on-demand RA is mandatory, e.g., immediately before or after a software update, or for secure erasure/reset.

[1] https://www.arbornetworks.com/blog/asert/reaper-madness/

[2] Since attestation is a potentially expensive task for prover, this verification mitigates computational DoS attacks.

[3] This goes counter to requirements in [5] that stipulate (potentially expensive) prover authentication of verifier's requests.

These two approaches are not mutually exclusive and may be used together to increase QoA.

II. Remote Attestation (RA)

RA techniques fall into the three main categories: (1) *Hardware-based* [16], [13] uses dedicated hardware features such as a Trusted Platform Module (TPM) to execute attestation code in a secure environment. Even though such features are currently available in personal computers and smartphones, they are considered a relative "luxury" for very low-end embedded devices. (2) *Software-based* [14], [15] requires no hardware support and performs attestation solely based on precise timing measures. However, it limits prover to being one-hop away from verifier, so that round-trip time is either negligible or fixed. It also relies on strong assumptions about attacker behavior [1] and is typically only used for legacy devices. (3) Finally, *hybrid* [8], [11], [4], based on a software/hardware co-design, provides RA while minimizing its impact on underlying hardware features. SMART [8] is the first hybrid RA design with minimal hardware modifications to existing microcontroller units (MCUs). Its key features are:

- Attestation code is immutable: it is located in, and executed from, ROM.
- Attestation code is safe: its execution always terminates and leaks no information other than the attestation result.
- Attestation is atomic: (1) its execution is uninterruptible, and (2) it starts from the first instruction and exits at the last instruction. This is realized via hard-wired MCU access controls and disabling interrupts upon entering attestation code.
- A secret key (K) stored in a secure memory location and accessible only from the attestation code: K is stored in ROM and is guarded by MCU rules.

[5] extended SMART to defend against denial-of-service (DoS) attacks on prover. We refer to this extended design as SMART+. [5] additionally requires prover to have a Reliable Read-Only Clock (RROC), needed to perform verifier authentication and prevent replay, reorder and delay attacks.

TrustLite [11] security architecture also supports RA for low-end devices. It differs from SMART in two ways: (1) interrupts are allowed and handled securely by the CPU Exception Engine, and (2) access control rules can be programmed using an Execution-Aware Memory Protection Unit (EA-MPU). TyTAN [4] adopts a similar approach while providing additional real-time guarantees and dynamic configuration for safety- and security-critical applications.

HYDRA [9] is a hybrid RA design for medium-end devices devices with a Memory Management Unit (MMU). It builds upon a formally verified micro-kernel, seL4 [10], to ensure memory isolation and enforce access control to memory regions.

In this paper, we use SMART+ and HYDRA as base security architectures for ERASMUS. However, ERASMUS is equally applicable to other on-demand RA techniques, such as TrustLite [11] or TyTan [4].

III. Self-Measurements

As discussed earlier, on-demand RA is a time-consuming activity that takes prover away from its primary mission. In contrast, ERASMUS divides RA into two phases. In the *measurement* phase, prover performs self-measurements based on a pre-established schedule and stores the results. In the *collection* phase, verifier (whenever it wants) contacts prover to fetch these measurements. This phase is very fast since it requires practically no computation by prover. In particular, since measurements are based on a MAC computed with a key shared between prover and verifier, no extra protection is needed when prover sends these measurements to verifier. Furthermore, since there is no threat of computational DoS on prover, there is also no need to authenticate verifier's requests. This is in contrast with on-demand RA. A prover's measurement M_t computed at time t is defined as:

$$M_t =< t, H(mem_t), MAC_K(t, H(mem_t)) >$$

where H is a suitable cryptographic hash function and mem_t represents prover's memory at time t. The computation of $H(mem_t)$ and MAC is done in the context of the security architecture, e.g., SMART or HYDRA.

From here on, $\mathcal{V}$rf and $\mathcal{P}$rv are used to denote verifier and prover, respectively. Although ERASMUS assumes a symmetric key K shared between $\mathcal{V}$rf and $\mathcal{P}$rv, a public key signature scheme could be used instead, with no real impact on security, except for higher measurement cost.

A. Quality of Attestation

Quality of Attestation (QoA) is determined by two parameters: (1) time T_M between two successive *measurements* on $\mathcal{P}$rv, and (2) time T_C between two successive requests by $\mathcal{V}$rf to collect measurements from $\mathcal{P}$rv.

Exactly how T_C and T_M are determined depends on $\mathcal{P}$rv's hardware platform, mission and deployment setting. Security impact of these parameters is intuitive. Smaller T_M implies smaller window of opportunity for mobile malware to escape detection. Smaller T_C implies faster malware detection. If either value is large, attestation becomes ineffective. Meanwhile, though low values increase QoA, they also increase $\mathcal{P}$rv's overall burden, in terms of computation, power consumption and communication.

We assume that, in most cases, $T_C > T_M$. If $T_C \leq T_M$, $\mathcal{V}$rf would collect same measurements more than once. Alternatively, $\mathcal{V}$rf can **explicitly request** $\mathcal{P}$rv to produce a measurement before collection. In that case, $\mathcal{V}$rf's request must be authenticated and checked for freshness (as in SMART+ [5]) before on-demand measurement is computed. These activities clearly incur additional real-time overhead and delays. We refer to this variant as ERASMUS+OD.

Without loss of generality, we assume that measurements and collections occur at regular intervals. However, in some cases, it might be advantageous to take measurements at irregular intervals, since doing so might give prover a bit of an extra edge against mobile malware (see Section III-E).

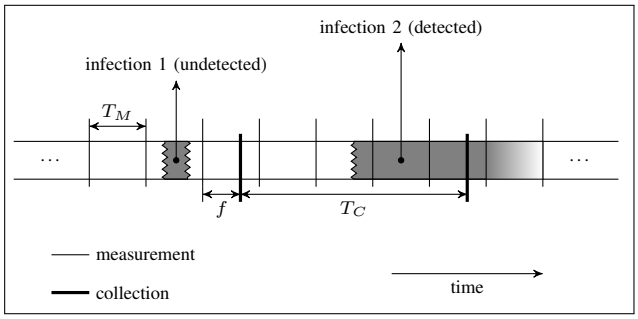

Figure 1. QoA illustration: Infection 1 by mobile malware is undetected; Infection 2 is detected. T_M – time between two measurements, T_C – time between two collections, and f – measurement's freshness.

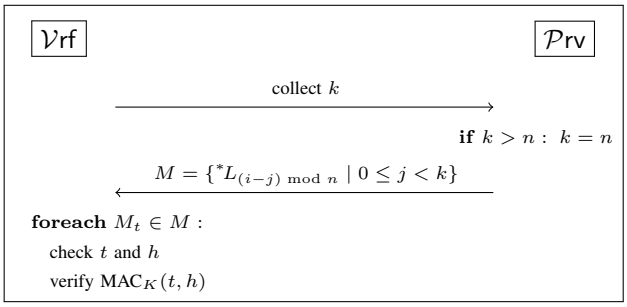

Figure 2. ERASMUS collection protocol.

Another ERASMUS parameter is the number of measurements (denoted as k) obtained by $\mathcal{V}$rf in each collection phase. It can range between *one* (latest measurement) and *all*. In an ideal setting, $\mathcal{P}$rv's history size should be set such that each measurement is collected exactly once, i.e., $k = \lceil T_C/T_M \rceil$.

Finally, the collection phase involves the notion of *freshness* of $\mathcal{P}$rv's latest measurement. Depending on the application, maximal freshness might be required, e.g., right before or after a software update. Maximal freshness is attainable via on-demand RA. In ERASMUS, freshness of a measurement (denoted as f) ranges between T_M and 0, which correspond to minimal and maximal freshness, respectively. We expect that $f = T_M/2$. Figure 1 shows an example with two infections of $\mathcal{P}$rv. In the first, malware covers its tracks and leaves before any measurement takes place. In the second, malware persists.

B. Measurements Storage & Collection

A naïve way for $\mathcal{P}$rv to store measurements is to keep track of them indefinitely. However, this will eventually consume a lot of $\mathcal{P}$rv's storage. Therefore, ERASMUS uses *rolling measurements*. A fixed section of $\mathcal{P}$rv's insecure storage is allocated as a windowed (circular) buffer for n measurements and i-th one is stored at location $L_{i \bmod n}$. However, it is expected that $\mathcal{V}$rf collects measurements sufficiently often, such that no location is over-written. That is, the time between successive collections should be $\le\ T_C \le n \cdot T_M$.

Interaction between $\mathcal{P}$rv and $\mathcal{V}$rf is very simple: $\mathcal{V}$rf asks for k latest measurements, which $\mathcal{P}$rv simply reads from the buffer and returns. The collection phase does not involve any change of state on $\mathcal{P}$rv and returned measurements are not encrypted. (Though, recall that they are authenticated, since each measurement is computed using K). It also does not trigger any significant computation on $\mathcal{P}$rv; in contrast with on-demand RA, no cryptographic operations are required in the collection phase.[4]

Self-measurements can be stored in $\mathcal{P}$rv's unprotected storage. This allows malware (possibly present on $\mathcal{P}$rv) to tamper with measurements, by modifying, re-ordering and/or deleting them. However, since malware (by design of underlying SMART) can not access K, it cannot forge measurements. Thus, it is easy to see that any tampering will be detected by $\mathcal{V}$rf at the next collection phase and malware presence would be immediately be noticed. For the same reasons, code that handles request parsing as well as storage and transmission of measurements does not need to run in a secure environment or be stored in ROM. Whereas, code that performs self-measurement must be protected by the underlying security architecture, as in on-demand RA.

Scheduling in ERASMUS can be done in a very simple and stateless manner. Let t be the RROC reading at the time of measurement M_t, and let T_M be the time between two successive measurements, as configured in $\mathcal{P}$rv. The windowed buffer slot L_i, used to store M_t, is determined by: $i = \lfloor t/T_M \rfloor \bmod n$.

ERASMUS collection protocol is shown in Figure 2. Notation $^*L_j$ refers to contents of location L_j.

C. ERASMUS+OD: ERASMUS with On-demand RA

As mentioned in Section III-A, ERASMUS may be combined with on-demand RA to benefit from advantages of both approaches. This variant, ERASMUS+OD, records $\mathcal{P}$rv's state history to detect mobile malware, and uses on-demand RA to obtain better freshness. Freshness is particularly relevant whenever real-time RA is mandatory, e.g., immediately before or after a software update.

The measurement phase is unmodified, while the collection phase is combined with on-demand attestation request, as follows. First, as part of each request, $\mathcal{V}$rf now computes and includes an authentication token and specifies k. As in SMART+ [5], $\mathcal{V}$rf authentication protects $\mathcal{P}$rv against computational DoS attacks. Next, after checking that a request is valid, $\mathcal{P}$rv computes a fresh measurement which it returns to $\mathcal{V}$rf, along with k previous ones.

D. Security Considerations

Security of the measurement phase itself is based on the underlying security architecture, e.g., SMART+ or HYDRA, which: (1) provides measurements code with exclusive access to K, (2) ensures non-malleability and non-interruptibility of the measurement code, and (3) performs memory-cleanup after execution.

Timestamps used in the measurement phase *must* be based on the RROC which (by definition) can not be modified by

[4]However, in the ERASMUS+OD variant mentioned in Section III-A, $\mathcal{V}$rf's request must be authenticated and checked for freshness, and a current measurement must be computed.

non-physical means. If RROC value could be modified, the following attack scenario would become possible: malware enters at time t_0 and remains active long enough so that a measurement at time $t_0 + \delta$ (with $\delta < T_M$) is taken. Before leaving, malware discards that measurement and resets the counter to t_0. Soon after δ (so that a measurement, valid this time, has been taken for $t_0 + \delta$), malware returns and resets the counter to time elapsed since t_0. Though this example works for one T_M window, it can be extended to arbitrarily many. It requires an additional assumption that no collection took place during presence of malware. Fortunately, RROC is already required in the underlying SMART+ security architecture, for a totally different reason. In SMART+, RROC helps prevent replay and computational DoS attacks on $\mathcal{P}$rv. Thus, ERASMUS does not require any architectural changes or additions.

E. Irregular Intervals

A natural extension to ERASMUS is to use irregular measurement intervals, instead of a fixed T_M. One motivating factor is that mobile malware aware of fixed scheduling knows when to enter/leave $\mathcal{P}$rv in order to stay undetected. One way to implement irregular intervals is by using a cryptographically secure Pseudo Random Number Generator (PRNG) initialized or seeded with the secret key K. Output of PRNG can be truncated such that T_M is upper- and/or lower-bounded. For example, after computing M_{t_i}, $\mathcal{P}$rv can set the measurement timer to $T_M^{\text{next}} = \text{map}(\text{PRNG}_k(t_i))$, where map is a function that maps PRNG output to seconds, e.g., $\text{map} : x \mapsto x \bmod (U - L) + L$, where U and L are upper and lower bounds, respectively. The timer itself must be read-protected to ensure that T_M^{next} is unknown to malware potentially present on $\mathcal{P}$rv. PRNG code must be protected the same way as measurement collection.

IV. Discussion & Extensions

Due to strict size limitations, some topics could not be discussed here. They are summarized below and discussed in detail in the full version of this paper [6].

Implementation: We built two prototype implementations of ERASMUS on two hybrid RA architectures: SMART+ and HYDRA. Experimental results show that: (1) ERASMUS does not require extra features (or larger ROM) than that in SMART+, and (2) $\mathcal{P}$rv-$\mathcal{V}$rf interaction is appreciably faster than in on-demand RA.

Swarm Attestation: Some applications require attesting a group (or swarm) of interconnected embedded devices. SEDA [3] presents the first swarm attestation scheme, which relies on hybrid RA architectures. SEDA combines them with a request-flooding and response-gathering protocol. SEDA was improved and further specified in LISA [7]. ERASMUS can be used instead of on-demand RA in swarm attestation protocols. In particular, $\mathcal{P}$rv self-measurements can be coupled with a collection protocol, e.g., LISA-α, where the latter only relays reports and perform no computation. This can yield a clean and conceptually simple approach to swarm attestation that inherits all benefits of ERASMUS.

V. Conclusions

This paper presents ERASMUS as an alternative to current on-demand RA techniques for low-end devices. It is based on scheduled self-measurements, which is more friendly for time- or safety-critical applications. ERASMUS also provides detection of mobile malware, which is not possible with on-demand techniques. Its other major advantage is that it requires no cryptographic computation by $\mathcal{P}$rv as part of its interaction with $\mathcal{V}$rf during the collection phase.

In addition, we defined a notion of Quality-of-Attestation (QoA) as a measure of temporal security guarantees given by an RA technique. We show that timing of measurements and timing of verifications (that are conjoined in on-demand RA) are two distinct aspects of QoA. They are treated as distinct parameters in ERASMUS. We also discuss the possibility of using on-demand RA as part of ERASMUS collection phase to obtain maximal freshness. Finally, we show that ERASMUS is a promising option for attesting groups/swarms of devices.

SUPPORT: This work was supported in part by DHS, under subcontract from HRL Laboratories, and ARO under contract W911NF-16-1-0536.

References

[1] T. Abera et al. Invited: Things, trouble, trust: on building trust in IoT systems. In *DAC*, 2016.
[2] M. Antonakakis et al. Understanding the mirai botnet. In *USENIX*, 2017.
[3] N. Asokan et al. SEDA: Scalable embedded device attestation. In *CCS*, 2015.
[4] F. Brasser et al. TyTAN: tiny trust anchor for tiny devices. In *DAC*, 2015.
[5] F. Brasser et al. Remote attestation for low-end embedded devices: the prover's perspective. In *DAC*, 2016.
[6] X. Carpent et al. ERASMUS: Efficient Remote Attestation via Self-Measurement for Unattended Settings (Full Version). arXiv:1707.09043v1, 2017.
[7] X. Carpent et al. Lightweigh swarm attestation: a tale of two LISA-s. In *ASIACCS*, 2017.
[8] K. Eldefrawy et al. SMART: Secure and minimal architecture for (establishing dynamic) root of trust. In *NDSS*, 2012.
[9] K. Eldefrawy et al. HYDRA: Hybrid Design for Remote Attestation (Using a Formally Verified Microkernel). In *WiSec*, 2017.
[10] G. Klein et al. seL4: Formal verification of an OS kernel. In *SIGOPS*, 2009.
[11] P. Koeberl et al. TrustLite: A security architecture for tiny embedded devices. In *EuroSys*, 2014.
[12] R. Ostrovsky and M. Yung. How to withstand mobile virus attacks. In *PODC*, 1991.
[13] D. Schellekens et al. Remote attestation on legacy operating systems with trusted platform modules. *Electronic Notes in Theoretical Computer Science*, 2008.
[14] A. Seshadri et al. SWATT: Software-based attestation for embedded devices. In *S&P*, 2004.
[15] A. Seshadri et al. SCUBA: secure code update by attestation in sensor networks. In *WiSe*, 2006.
[16] F. Stumpf et al. A robust integrity reporting protocol for remote attestation. In *WATC*, 2006.
[17] J. Vijayan. Stuxnet renews power grid security concerns. http://www.computerworld.com/article/2519574/security0/stuxnet-renews-power-grid-security-concerns.html, 2010.