Comments on "Refutation of *On the Difficulty of Software-Based Attestation of Embedded Devices*"

Aurélien Francillon, Claude Castelluccia, Daniele Perito, Claudio Soriente

October 20, 2010

Abstract

In [2] we published several attacks against software-based attestation techniques on embedded systems. This document is a response to a refutation [8] from Perrig and van Doorn. We provide detailed explanations that shows that most of the comments in [8] are either inaccurate, overlooking some fundamental properties of our attacks, or discussing solutions provided for platforms other than the ones we evaluated. Some of the comments however are legitimate and, for this reason, we will make available an updated version of the paper on our web pages. This document is intended to be constructive and productive toward improving the current state of the art of software-based attestation, which we believe is a promising primitive.

However, we believe that our paper contains no "*factual errors and inaccuracies,*". At best, we mis-attributed some claims or overlooked some suggestions. Attacks we presented in [2] are legitimate, implemented, working and do not involve any unrealistic assumptions. Moreover we believe that these attacks represent useful contributions to the community. Our attacks pertain mainly to the SWATT protocol [14] and ICE primitive [11], and are limited to embedded devices. We did not consider other techniques, such as Pioneer [12] or other protocols for non embedded platforms.

In summary, we fully agree that software-based attestation is a very useful primitive and offers previously unachieved properties. Many challenges remain, and we hope that our work will stimulate more research in this area.

1 Introduction

This paper discusses the "refutation" of our attacks in [8].

All attacks presented in [2], were performed using algorithms described in literature. Performing an independent implementation from scratch allowed us to get into the intimate details of the SWATT protocol. We believe that it was the right strategy, as this would be the path followed by an implementer of the scheme.

Our paper focused exclusively on embedded platforms and more specifically on the AVR architecture (except for section 4.2 in [2]). Therefore our attacks are relevant only to [14, 13] and not [9, 6, 11, 12, 10] as claimed in [8]. Section 4.2 of [2], that focused on the ICE primitive (as described in SCUBA [11]), is relevant to the following publications [9, 11, 10], but not to [12].

We understand that [14] (and to a lesser extent [10, 11]) in an early work on software-based attestation and that it was later improved in PIONEER and PIONEER-NG. However, PIONEER is **not** relevant to our work [2]. Once again, we only address low-end embedded systems, while PIONEER was designed for the netburst Pentium4 architecture. Therefore, while some of our attack do not apply to PIONEER-based protocols, they are fully applicable to the SWATT protocol [14].

2 Return-oriented rootkit

Code integrity and control flow integrity From Section 2.1 of [8]:

> *In this attack, Castelluccia et al. make several assumptions about software-based attestation: (1) software-based attestation achieves control-flow integrity, [...] software-based attestation was primarily designed to achieve code integrity, but not control-flow integrity. In particular, the SWATT function's main purpose was to validate the code memory. Hence, the presented "attack" is on a property that is not attempted in general software-based attestation mechanisms.*

We never assumed that software-based attestation would provide control-flow integrity. However, we believe that this attack is a direct consequence of *return-oriented programming* (ROP): code integrity without control flow integrity is at risk of being of very limited value in practice. We demonstrate this and thus show the *Time Of Check to Time Of Use* (TOCTOU) problem at its extreme: in the simplest realization of the ROP Rootkit the malicious code is removed right before attestation and reinstalled right after the checksum is produced. This leaves the verifier with a checksum value that does not correspond to the actual state of the device a few milliseconds after that value is sent.

While one could argue that it is not an attack on the software-based attestation itself (as suggested in [8]), it still violates the properties that are expected to be provided, i.e., detection of malicious code on the device. Moreover, the following paragraphs in Section 3.5 of [14] highlights the relevance of our attack:

> *Note that an attacker cannot reconstruct the correct memory image before verification and undo changes afterward. Doing this will require the code that does the undo operation to be resident in the memory of the device. This change in memory contents will be detected by SWATT.*
>
> *In the Harvard architecture, the program memory is separate from the data memory and typically different instructions are required to access the program and data memory. Since the data memory is not executable, we usually only need to verify the program memory. The program memory typically contains executable code and static data. Hence, its contents will be known to the verifier a priori. This makes it feasible to verify a running device.*

To conclude, our ROP Rootkit attack clearly demonstrates that it is not necessary to have *code* in program memory to

> *[...]reconstruct the correct memory image before verification and undo changes afterward*
> .

Untampered execution environment From Section 2.1 of [8]:

> *Because the attestation sets up an untampered execution environment, the ROP would not gain control because we still did not call return.*

We would like to present the following counter-arguments :

- Our ROP-based attack was intended for code attestation schemes that only check program memory and that run on AVR architectes. Once again, it was not intended to be used for code attestation that works on more sophisticated platforms.
- *Untampered execution environments* have never been discussed and proposed for AVR-based architectures, since they are in practice far from being trivial on these platforms. For example, one of the most important features of *untampered execution environments* is to include the PC in the memory checksum. However, the AVR Instruction Set does not allow a direct access to the Program Counter. This makes the construction of an "untempered" environment difficult.

- The idea of *untampered execution environments* appeared after the publication of [14], in [10, 11]. Once again, we did not claim that our attack is relevant to these schemes. Actually, ROP Rootkits are unnecessary on von Neumann architecture based devices [1] as all the data memory is executable (no nx-bit or $W \oplus X$ like functionality is present). One would therefore not need return-oriented programming at all. We believe this is the reason why ICE was implemented to check only a small portion of memory to set up an *untampered execution environment*.
- The rootkit reliance on the modification of the return address used right after attestation to launch the return-oriented programming was only done for ease of demonstration. Indeed, return-oriented programming used to restore malicious code can be launched in several other ways. For example, by corrupting *any* return address on the stack (or stacks if multi-threaded environment is present), any function pointer (stored in data memory) in the program or by memory corruptions that, e.g., would only allow partial control of the PC or SP (and therefore might not allow standard code injection). Therefore, checking only return addresses on the stack before returning is not a sufficient countermeasure. The correct countermeasure is to attest data memory or erase it completely as in [7]. Moreover, the return-oriented programming data (or stack) is not stored on the normal system stack but at arbitrary free position in the data memory. Finally, return address checking has been shown not to be sufficient [3].
- In SWATT [14], it is never mentioned that data memory (e.g., return addresses) needs to be checked. It is even argued in Section 3.5 that this is unnecessary on this platform:

  > *Since the data memory is not executable, we usually only need to verify the program memory.*

- While we experimented with the return-oriented rootkit attack on an Atmega128 and not on the Atmega163L (which was used in the original SWATT paper) both have the same functionalities (almost identical instruction set, read/write program memory, bootloader support etc....). Therefore our attack is still relevant.
- As we point in our paper, if a check for data memory (pointers, stack or any non program memory region) is done, it must be performed using the same method as for program memory. The loop that performs random memory traversal has to check other memories at the same time. Performing it before or after this loop allows the attacker to bypass it in constant time, as long as it can take control back, e.g., using return-oriented programming (with [1] or without [3, 15] "returns") and might remain undetected.

As discussed in [2]:

- We therefore implemented such modifications on the AVR Architecture (discussed in Section 4.1.3 of [2]) to evaluate the feasibility of modified SWATT that would attest other memories. We found that this required profound changes to the algorithm and reduced the overhead of an attack as a result of the increase in size of the algorithm. Hence, those solutions did not appear convincing.

Finally, performing some additional operations before using a return instruction (e.g. check the stack) is not a definitive solution, since, for example, return-oriented programming has recently been shown to be practical (on several architectures) without the use of any *return* instruction (with [1] or without [3, 15] "returns").

On the other hand, we discuss several mitigation techniques. For example, the attack would not be possible if no boot-loader is present in memory.

[1] Specifically the MSP430 family of microcontrollers

3 Memory Shadowing Attack

From Section 2.1 and Section 2.2. of [8]:

> *It is certainly convenient to assume a weak function and then attack it. In our implementation,* [..]
>
> *Section 4.1 presents a memory shadowing attack [1]. In a nutshell, the authors re-implement SWATT on a different platform, and attack their new implementation. Once the attack on their re-implementation is successful, they conclude that SWATT is also vulnerable.*

We implemented SWATT as it is described in [14] and the shadowing attack applies to the original algorithm as described in [14].

We now describe in more detail:

- For devices that use the ATMEGA163L (Section 4.1.1 in [2]), the only modification we made to the SWATT algorithm was to correct the RC4 implementation bug (footnote 8 in [2]) and to use numerical register names instead of aliases (zh aliases to r31, zl aliases to r30, etc.). Figure 10 of [2] presents the memory shadowing attack for a device with a full 16 bit of address space mapped to 64Kb physical memory. It is trivial to adapt this attack to the exact model used in the original paper [2]
- We also ported the SWATT algorithm to devices with more memory and discussed the implications (Section 4.1.2). We believe that this is crucial for practitioners. (For example, the Atmega128 is used by the MicaZ Wireless sensor node.)

From Section 2.2. of [8]:

> *Since the 16K program memory on an Atmega163l system is almost always completely filled with code, their attack would not work on a practical system.*

The fact that half of the memory has to be empty is a clear assumption of our attack and we did not hide it. We believe this is a realistic assumption since many simple TinyOS programs often use less than 8Kb. The condition that at least half of the memory has to be used by code does not seems to be an appropriate property for a security system.

From [8]:

> *Moreover, their attack still exhibits a 7.4% overhead, which is much lower than our 13% overhead. However, in our implementations of software-based attestation systems we assume that an adversary can design an attack function with half the overhead of our optimal attack function, thus, we set the threshold to 6.5%. This point is illustrated well by the Pioneer system, where we set the threshold even below half of the attackers overhead [12]. This also indicates that we do not claim to have invented the best possible attack we always point*

[2] For devices with less memory such as the ATMEGA163L with 13 bits of useful address to 16Kb of physical memory the shadowing is performed between addresses : 00000 base address
`0x07FF` end of 1^{st} quarter of memory
`0x1000` end of 2^{nd} quarter of memory
`0x17FF` end of 3^{rd} quarter of memory
`0x1FFF` end of 4^{th} quarter of memory

In order to shadow the 4^{th} quarter of the memory by the 3^{rd} quarter, the 12^{th} bit of the program address needs to be written to zero when the 13^{th} bit of the address is one, this translate to:
sbrs r31, 4
cbr r31, 3

we recall that the address is in 16bit register Z which is an alias for the 2 8-bits registers r31:r30, bits in registers are counted from zero for instructions sbrs and cbr.

out that we need to prove the run-time optimality of the checksum function as well as of the best attack code.

Again this 50% tradeoff was not discussed in the SWATT paper. Also note that with the return-oriented rootkit attack this overhead is below 1%.

From [8]:

> *This also indicates that we do not claim to have invented the best possible attack we always point out that we need to prove the run-time optimality of the checksum function as well as of the best attack code.*

Provability of such properties for software-based attestation is clearly an important goal. Our contribution provided further motivation that proving such properties is of paramount importance. However, this has never been done on real world processors and for most architectures it's unrealistic (considering the complexity of the datasheets for e.g. the ARM or the Intel processors for which documentation is of several thousands of pages).

More generally and depending on the context there are many properties that have to be proven (software tamper resistance, time optimality, best possible attack, etc.)

From Section 2.2. of [8]:

> *However, in the SWATT paper in Section 3.5 [13] we state: Empty regions of memory are often filled with zeros. So, if an attacker places malicious code in the empty memory regions, it can suppress the read to these memory locations and substitute it with zero. Also, the attacker need not compute the exor operation when computing the checksum of a zero-valued memory location. Together, the time saved by not performing these two operations may offset the time for an extra if statement. To prevent this attack, we suggest that empty memory regions be filled with a pseudo-random pattern. Filling the empty memory regions with the pseudo-random pattern is another reason why the proposed attack does not work on the SWATT system as presented in the paper. The positive aspects of this section though is that the authors have indeed identified a faster attack. Although we have attempted to explore possible attacks based on all branch instructions, the attack based on the SBRS instruction seems to have eluded us. That faster attack is a good contribution.*

We will add a reference to this in an updated version of our paper. However, we note that the reason behind this argument seems completely unrealistic (and impossible to implement). Moreover, this is clearly presented as a "suggestion" of "Considerations for Practical Use" . We believe that after our attack it should be a mandatory requirement.

Randomization of unused memory areas does not change anything for the return oriented rootkit.

In our implementation the malicious code in program memory is "erased" by writing as many blocks of $0xFF$ [3] those blocks of memory could very well have been stored with the original randomness prior to the installation of the malicious code in flash.

4 Attack on ICE

From Section 2.3 of [8]:

> *ICE : This attack illustrates that we need better mechanisms to formally validate these functions, a point that we make repeatedly in our papers.*

We fully agree with this sentiment. Unfortunately this has only been accomplished on rather primitive platforms [5, 4].

[3] as it is indeed the value of non programmed flash memory

An apparent paradox for timed software-based attestation is that it appears that an extremely fast cryptographic primitive is needed to compute memory checksum. This is necessary both to make any change apparent and visible to an external verifier and to make it easier to verify optimality or *quasi-optimality* of the code. The problem is that such fast cryptographic primitives appear to be vulnerable to online attacks that allow to compute valid checksums in illegitimate ways. This is a central point we made in our paper.

5 Well-known Recommendations and Various Inaccuracies

Porting SWATT to variants of the AVR or ICE to the AVR From Section 2.4 of [8]:

> *From Section 2.2 [1]: "Indisputable Code Execution (ICE) based schemes rely on an attestation procedure being performed on the attestation routine itself, including the program counter in the computation. ... Unfortunately, not all platforms make the program counter available to software. This is the case, for example, of the AVR family of micro-controllers 4 used on MicaZ devices. Porting ICE on this family of processors would require complex changes or would just not be feasible." From Section 4.1.2 [1]: "We conclude that the security of SWATT relies on some unique characteristic of the devices considered by the authors to run their experiments. Porting SWATT on a new device with a new instruction set or a different memory size, dramatically changes the rules for both the attacker and the verifier, which can undermine the security of the scheme."*
>
> *In all our papers we emphasize that a shortcoming of software-based attestation is the tight coupling between the attestation function and the hardware architecture. Consequently, for each specific hardware architecture, a specific attestation function needs to be designed.*

While we evaluated attacks on the original SWATT algorithm we also tried to fix and to extend it to other variants of the Atmega microcontroller.

We believe that our paper clearly demonstrated the serious drawbacks in designing a modified attestation function for just a variant of the Atmega163L, which we believe to be a genuine contribution. We will clarify this text and add (one more) reference to [14] in an updated version of the paper.

Fastest implementation claims From Section 2.4 of [8]:

> *From Section 2.2.2 [1]: "They claim to have implemented the fastest checksum function and to have considered the fastest redirection routine and show that it would still introduce a considerable overhead to checksum computation."*
>
> *We were always careful not to claim that we have found the fastest checksum implementation. In all our papers we emphasize that formal analysis is required to prove optimality.*

We were mislead by the following statement in section 3.2 of the SWATT paper:

> *Due to the small code size of the loop body of the verification procedure we were able to hand-optimize it to be highly efficient. In Section 3.3 we argue why our code sequence cannot be optimized further.*

However, we will modify this claim in the updated version of the paper. Basically, the security of the scheme relies on the availability of a proof of the presence of a fastest attestation routine and known attack.

Data memory checks From Section 2.4 of [8]:

> *The SWATT paper actually does point out that the data memory may need to be verified for some applications, for example for Von Neumann architectures [13].*

The point we make in our paper is that data memories **always** have to be attested, and not only for some applications, both on Von Neumann and on Harvard architecture devices (where data memory is not executable and on a separate address space, as on the AVR). We believe this is an important distinction.

6 Conclusion

Finally, our paper should not be taken as a rebuttal against software-based attestation in general, it is not, but as a strong warning that in practice many attacks have to be considered in a software-based attestation system. We believe that this was clear in the title of the paper itself (which contains "difficulty" and not "impossibility"), indeed both [8] and this document are only making our point stronger. However, we discussed in [2] and recalled here that most of the attacks are not as easy to fix as suggested in the refutation paper [8].

7 Bibliography

References

[1] E. Buchanan, R. Roemer, H. Shacham, and S. Savage. When good instructions go bad: generalizing return-oriented programming to RISC. In *Proceedings of CCS '08*. ACM, 2008.

[2] C. Castelluccia, A. Francillon, D. Perito, and C. Soriente. On the difficulty of software-based attestation of embedded devices. In *CCS '09: Proceedings of the 16th ACM conference on Computer and Communications Security*, New York, NY, USA, November 2009. ACM.

[3] S. Checkoway, L. Davi, A. Dmitrienko, A.-R. Sadeghi, H. Shacham, and M. Winandy. Return-oriented programming without returns. In *To appear at CCS 2010*, 2010.

[4] J. Franklin and M. C. Tschantz. On the (im)possibility of timed tamper-evident software in (a)synchronous systems. Unpublished, August 2007.

[5] V. Gratzer and D. Naccache. Alien vs. quine. *IEEE Security and Privacy*, 5(2):26–31, 2007.

[6] C. Kuo, M. Luk, R. Negi, and A. Perrig. Message-in-a-bottle: user-friendly and secure key deployment for sensor nodes. In *SenSys '07: Proceedings of the 5th international conference on Embedded networked sensor systems*. ACM, 2007.

[7] D. Perito and G. Tsudik. Secure code update for embedded devices via proofs of secure erasure. In *ESORICS*, 2010.

[8] A. Perrig and L. van Doorn. Refutation of on the difficulty of software-based attestation of embedded devices, 2010.

[9] A. Seshadri, M. Luk, and A. Perrig. SAKE: Software attestation for key establishment in sensor networks. In *DCOSS '08: Proceedings of the 4th IEEE international conference on Distributed Computing in Sensor Systems*, 2008.

[10] A. Seshadri, M. Luk, A. Perrig, L. van Doorn, and P. Khosla. Using fire and ice for detecting and recovering compromised nodes in sensor networks. Technical Report CMU-CS-04-187, CMU, December 2004.

[11] A. Seshadri, M. Luk, A. Perrig, L. van Doorn, and P. Khosla. SCUBA: Secure code update by attestation in sensor networks. In *WiSe '06: Proceedings of the 5th ACM workshop on Wireless security*. ACM, 2006.

[12] A. Seshadri, M. Luk, E. Shi, A. Perrig, L. van Doorn, and P. Khosla. Pioneer: verifying code integrity and enforcing untampered code execution on legacy systems. In *SOSP '05: Proceedings of the twentieth ACM symposium on Operating systems principles*. ACM, 2005.

[13] A. Seshadri, A. Perrig, L. van Doorn, and P. Khosla. Using SWATT for verifying embedded systems in cars. In *Proceedings of Embedded Security in Cars Workshop (ESCAR 2004)*, Nov. 2004.

[14] A. Seshadri, A. Perrig, L. van Doorn, and P. K. Khosla. SWATT: SoftWare-based ATTestation for embedded devices. In *IEEE Symposium on Security and Privacy*. IEEE Computer Society, 2004.

[15] V. F. Tyler Bletsch, Xuxian Jiang. Jump-oriented programming: A new class of code-reuse attack. Technical report, NCSU, 2010. NCSU Technical Report TR-2010-8.