Refutation of “On the Difficulty of Software-Based Attestation of Embedded Devices”

Adrian Perrig
CyLab/CMU

Leendert van Doorn
AMD

Abstract

The paper “On the Difficulty of Software-Based Attestation of Embedded Devices” had been published at the ACM CCS 2009 conference [1]. Although the paper contains many useful points, unfortunately, it also contains numerous errors and inaccuracies which we would like to rectify with this note.

1 Context

Software-based attestation is a promising technique, which enables a verifier to check the integrity of software that is executing on an untrusted device. In contrast to hardware-based attestation, software-based attestation does not require any special hardware support. Instead, software-based attestation relies on the execution properties of a verification function, that enables external detection if the verification function deviates from the intended execution.

Before we discuss the Castelluccia et al. paper in detail, we would like to provide some context on our software-based attestation research since early in year 2003 when we started working on SWATT. Until recently, the development of software-based attestation techniques was accompanied by numerous doubts proclaiming that “the current techniques were not useful in practice and that the next step would not be feasible.” We are glad to note that today the community has largely accepted the beneficial properties and the feasibility of software-based attestation, and that we now need to work on approaches that provide provable properties. However, several research results were necessary to get to this point.

When our first paper (SWATT) was published at the IEEE Security and Privacy Symposium [13], researchers mentioned that verifying the entire memory is impractical and that it would be impossible to only verify a subset of the memory. A common criticism also was the claim that these techniques would never work on more complex hardware architectures. Many researchers we interacted with also did not believe that these techniques would be useful in practice and that the community should dedicate a concerted research effort to mitigate the shortcomings.

Since we firmly believed in the utility of these mechanisms, we continued our research to further develop this research. Initially, our main goal was to demonstrate the *usefulness* and broad applicability of these techniques, and that software-based attestation was *possible* on a variety of architectures. By attempting to convince the community of the utility and feasibility of software-based attestation, our hope was that more researchers will join the effort and work on proving the correctness of these approaches.

One work that represented a breakthrough for our team was the paper “Fire and ICE” (which was only published as a TR [8]), in which we found an approach for only checking a small subset of the total memory, which addressed many of the shortcomings of the SWATT function. The ICE function could essentially be used to establish a dynamic root of trust for executing security-sensitive code on an untrusted embedded system. The function was later published in our works on SCUBA [11] and SAKE [10].

To address the criticism that software-based attestation was impossible on high-end processors, we worked on an implementation on a Pentium IV processor. That work was published as Pioneer [12]. Although that work did not address all possible issues, such as SMM-based malware, or multi-core and multi-threaded processors, the work nevertheless represented a leap forward for us because we addressed a significant number of issues that have been suspected to be fundamental. We are deeply indebted to the SOSP program committee for recognizing the contributions of the work without dwelling on the remaining shortcomings. Unfortunately, many of the hallway discussions after the Pioneer presentation still revolved around criticisms that software-based attestation lacks useful applications and that it would be impossible to obtain a provably secure function for high-end processors that covers all issues. Fortunately though, Greg Nelson who was attending the conference gave us hope that his Denali optimizer could be used to formally prove the optimality of a software-based attestation function. While still on the mission of convincing the community of the utility and feasibility, we continued our work in that direction, while emphasizing that formal properties need to be shown to apply these techniques in practice.

In his thesis, Arvind Seshadri worked on Pioneer-NG, a software-based attestation function that addresses all the issues on a high-end AMD processor (with the exception of overclocking the processor, which needed to be detected based on the physical artifacts of overclocking), including attacks based on SMM mode and multi-core execution [9]. Simultaneously, we also continued our work on realistic applications, to demonstrate use in embedded car networks [14], SCADA networks [16], and sensor networks [11].

An exciting result was based on an insight by Virgil Gligor, that software-based attestation performs attestation without secrets. This is a significant advantage of software-based attestation over hardware-based attestation, because the absence of secrets enables use in settings where the adversary has compromised the secrets on a system. Moreover, since secrecy is difficult to achieve over the lifetime of the secret, in particular it is often impossible to detect whether secrecy has been breached in the past. As a consequence of the absence of secrecy, software-based attestation can be used for secure recovery after compromises. Our SAKE protocol leverages the fact that the device checksum can be used as a short-lived shared secret, which can be used to bootstrap a long-term shared secret [10]. We found these to be exciting results, because it demonstrates that software-based attestation provides fundamental properties that cannot be achieved with existing techniques.

This overview was written from our perspective, several other groups have been working on software-based attestation and have faced similar issues [2–7, 17–19].

Fortunately, many people in the community now agree on the utility of software-based attestation. While it is clear that software-based attestation does not solve all of our security problems and is not applicable in all settings, it has emerged that it is a useful technique for some specific applications in some specific settings. The next challenge now is to design software-based attestation schemes that provide formally verifiable security guarantees.

2 The Castelluccia et al. Paper

The recent paper by Castelluccia et al. revisits several prior works in software-based attestation and presents perceived shortcomings [1]. Such research is very important, because independent analysis and validation of published work is crucial for sound scientific progress.

Their paper contains several technical contributions, but alas, also many factual errors and inaccuracies, which prompted the production of this document.

2.1 Rootkit / ROP-based Attack (Section 3.1 [1])

From Section 3 [1]: "The first attack circumvents malware detection by moving malicious code between program memory and non-executable memory, during the code attestation procedure. This is achieved using a technique called Return-Oriented Programming."

The idea behind the attack is to leverage Return-Oriented Programming (ROP) [15] to subvert the control flow *after* software-based attestation executed. In ROP, jump targets are listed on the stack that are executed after the main function returns. In this attack, Castelluccia et al. make several assumptions about software-based attestation: (1) software-based attestation achieves control-flow integrity, (2) the software-based attestation function does not verify stack information.

To the first assumption, 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.

The specific property that software-based attestation provides is the execution integrity of the verification function. Under execution integrity, we understand code integrity, a defined entry point into the code, and an untampered execution environment where no other external code can tamper with the execution. The verification function can be crafted to establish an untampered execution environment, which can then be used to execute some piece of code.

Of course, all data used (including jump targets, stack, etc.) are validated and verified before use. Similarly, the untampered execution environment can be used to validate the stack, or to re-create a compliant stack.

To the second point, some software-based attestation mechanism do verify the stack information. For example, in the Pioneer [12] paper in Section 6.2, list point 3, we state: “The KMA also verifies that the return address on the stack points back to the kernel/LKM code segment.” Hence, if such properties were required, the code that executes right after the verification function can validate the stack integrity. The attack is thus based on a naive implementation, which utilizes stack inputs that are unverified. This is hardly an attack on software-based attestation itself, but rather on a flawed implementation of a system that leverages software-based attestation as a primitive.

We now discuss their attack in some more detail. The authors propose the following checksum code in Section 3.1 [1]: “Figure 4 presents a generic attestation function. In our prototype, we insert a hook to the rootkit bootstrap code, by replacing the first instruction of the attestation function with a jump.”:

```
void receive checksum request(uint8 t nonce){
  uint8 t checksum[8];
  prepare checksum(nonce);
  do checksum(checksum);
  send(checksum);
  return;
}
```

It is certainly convenient to assume a weak function and then attack it. In our implementation, we perform security-sensitive operations right after performing the attestation. Because the attestation sets up an untampered execution environment, the ROP would not gain control because we still did not call return. The security-sensitive operations can also verify areas in memory, as we did in a rootkit detector for Pioneer [12], which would also validate the sanity of the stack, preventing an ROP-based hook.

2.2 Memory Shadowing Attack

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.

Unfortunately, there are several issues with this approach. Foremost, the SWATT system was implemented for the Atmel ATMEGA163L microcontroller, an 8-bit Harvard Architecture with 16K of program memory and 1K of data memory [13]. The authors implemented their attack on the ATMega128L micro-controller, which has 128 KBytes of program memory. Since their attack requires that the latter half of the memory is set to zero,[1] the much larger memory size is quite convenient for the attack. 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.

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 attacker’s overhead [12]. 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.

Furthermore, footnote 9 claims [1]: “This attack would therefore not be possible if the free program memory is used or filled with randomness (as in [5, 33]), but this is not the case with SWATT.”

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

[1]Actually, a sufficient condition is that the contents of the third quarter of the memory equals the contents of the fourth quarter of memory. In the unlikely and highly construed case where the first quarter also equals the second quarter, no conditional statement would even be necessary. The attack would work by copying the most significant memory address bit into the second most significant bit, which in turn would enable the attacker to use the second and third quarter of memory for its code, further speeding up the attack compared to the one presented.

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.

2.3 Attack on ICE

The attack on the ICE function in Section 4.2 indeed works [1]. [2] Unfortunately, we missed two opportunities that we were aware of that would have defended against this attack. First, we did not use the entropy of the carry bit in the ICE function. The attack would likely be trivially prevented by using ADDC instead of ADD. Second, the attack was enabled by an omission of a mechanism that we planned to use, which was to alter the computation of the checksum in each loop. Pioneer uses this approach, but unfortunately, our ICE implementation ended up with all the same computations in each iteration, most likely due to time constraints when building a prototype system. This attack illustrates that we need better mechanisms to formally validate these functions, a point that we make repeatedly in our papers.

2.4 Well-known Recommendations and Various Inaccuracies

The Castelluccia et al. paper lists numerous recommendations that were already well-known in the literature, without attributing those recommendations to the prior publications.

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.

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. Moreover, in our implementations we assume that the adversary can find a function with a 50% lower overhead than our best attack function as we discuss above. So clearly, we never claim that we found the fastest checksum implementation nor the fastest attack function.

From Section 2.2.2 [1]: "Moreover, as SWATT does not attest data memory nor external storage, the prover could store malicious code in one of those memories and restore it after attestation using ROP (Section 3.1)."

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].

[2]We have actually proposed an analogous attack on the Genuinity system [6] in our SWATT paper [13], which exploits the fact that XOR and ADD are analogous for the MSB.

3 Conclusion

While the work by Castelluccia et al. contains useful contributions, it unfortunately also has numerous technical factual errors and inaccuracies. These issues could have been easily mitigated through inclusion of domain experts after acceptance of the paper. For example the conference shepherd of the paper could have asked for additional input, or the authors themselves could have requested validation for their very strong claims. By removing the errors, everyone would have benefited. For the audience at the conference, more time for questions during the Q&A session (although some people probably found the exchange amusing); for the authors and conference PC, a stronger and factually more correct paper; for us, not needing to write this paper.

In summary, software-based attestation is a very useful primitive that offers previously unachieved properties. It is our hope that researchers will agree on the importance of these methods and work on addressing the remaining challenges in this area.

References

[1] C. Castelluccia, A. Francillon, D. Perito, and C. Soriente. On the difficulty of software-based attestation of embedded devices. In *Proceedings of ACM Conference on Computer and Communications Security (CCS)*, November 2009.

[2] Y.-G. Choi, J. Kang, and D. Nyang. Proactive code verification protocol in wireless sensor network. In *Proceedings of ICCSA*, 2007.

[3] Vanessa Gratzer and David Naccache. Alien vs. quine, the vanishing circuit and other tales from the industry's crypt. In *Proceedings of Eurocrypt*, May 2006.

[4] Markus Jakobsson and Karl-Anders Johansson. Assured detection of malware with applications to mobile platforms. DIMACS Technical Report 2010-03, `http://dimacs.rutgers.edu/TechnicalReports/abstracts/2010/2010-03.html`, 2010.

[5] Markus Jakobsson and Karl-Anders Johansson. Assured detection of malware with applications to mobile platforms. In *HotSec*, August 2010.

[6] Rick Kennell and Leah H. Jamieson. Establishing the genuinity of remote computer systems. In *Proceedings of the 12th USENIX Security Symposium*, pages 295–308. USENIX, August 2003.

[7] T. Park and K. G. Shin. Soft tamper-proofing via program integrity verification in wireless sensor networks. *IEEE Transactions on Mobile Computing (TMC)*, 4(3), 2005.

[8] 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, School of Computer Science, Carnegie Mellon University, December 2004.

[9] Arvind Seshadri. *A Software Primitive for Externally-verifiable Untampered Execution and its Applications to Securing Computing Systems*. PhD thesis, Electrical and Computer Engineering Department, Carnegie Mellon University, 2009.

[10] Arvind Seshadri, Mark Luk, and Adrian Perrig. SAKE: Software attestation for key establishment in sensor networks. In *Proceedings of International Conference on Distributed Computing in Sensor Systems (DCOSS)*, June 2008.

[11] Arvind Seshadri, Mark Luk, Adrian Perrig, Leendert van Doorn, and Pradeep Khosla. SCUBA: Secure code update by attestation in sensor networks. In *Proceedings of ACM Workshop on Wireless Security (WiSe)*, September 2006.

[12] Arvind Seshadri, Mark Luk, Elaine Shi, Adrian Perrig, Leendert van Doorn, and Pradeep Khosla. Pioneer: Verifying integrity and guaranteeing execution of code on legacy platforms. In *Proceedings of ACM Symposium on Operating Systems Principles (SOSP)*, pages 1–16, October 2005.

[13] Arvind Seshadri, Adrian Perrig, Leendert van Doorn, and Pradeep Khosla. SWATT: Software-based attestation for embedded devices. In *Proceedings of the IEEE Symposium on Security and Privacy*, May 2004.

[14] Arvind Seshadri, Adrian Perrig, Leendert van Doorn, and Pradeep Khosla. Using SWATT for verifying embedded systems in cars. In *Proceedings of Embedded Security in Cars Workshop (ESCAR)*, 2004.

[15] Hovav Shacham. The geometry of innocent flesh on the bone: Return-into-libc without function calls (on the x86). In *Proceedings of ACM CCS*, October 2007.

[16] Aakash Shah, Adrian Perrig, and Bruno Sinopoli. Mechanisms to provide integrity in SCADA and PCS devices. In *International Workshop on Cyber-Physical Systems Challenges and Applications (CPS-CA)*, June 2008.

[17] M. Shaneck, K. Mahadevan, V. Kher, and Y. Kim. Remote software-based attestation for wireless sensors. In *Proceedings of ESAS*, 2005.

[18] Diomidis Spinellis. Reflection as a mechanism for software integrity verification. *ACM Transactions on Information and System Security*, 3(1):51–62, February 2000.

[19] Y. Yang, X. Wang, S. Zhu, and G. Cao. Distributed software-based attestation for node compromise detection in sensor networks. In *Proceedings of IEEE SRDS*, 2007.