Verifying Trusted Code Execution using ARM Trustzone

R.Sebas Sujeen **Sridhar Periasami**

College of Engineering, Anna University, Chennai – 600025, India

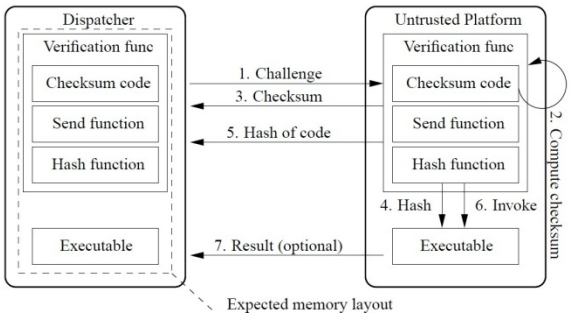
Summary
In this paper, we present a comprehensive analysis of a software based attestation system, Pioneer [1] which was designed for the x86 processor architecture and show how it would fail for RISC processor architecture like ARM. We then present an overview of the Security Extensions implemented in Cortex-A9 processors and higher, referred to as Trustzone[2] and how it can be leveraged to guarantee trusted code execution even on untrusted systems. We also discuss TOCTOU (Time of Check, Time of Use) issues with remote attestation and how it can be resolved leveraging Trustzone. We conclude with a discussion of how this can be used to implement a Kernel Integrity monitor that can be used to detect sophisticated malware like rootkits.

Key words:
remote attestation, Trustzone, TOCTOU attack, NS bit, rootkit, Kernel Integrity Monitor

1. Introduction

Even with a plethora of security products available, both commercial and open source, very few behave appropriately when the malware is running with the same privileges as the security software. Some notable products like MS-Patchguard and Mandiant's MIR has features for performing a self-checksum over the security product itself to verify that the product hasn't been tampered by the malware. But in reality, it just adds another layer of indirection, the malware can target the self-checksum routine of the security product and the security product would never know that it has been tampered with. Patchguard v2 had been subverted by Skywing in [3] who subjected it to a rigorous analysis by reverse engineering the product. Pioneer used software based attestation by a trusted verifier to achieve trusted code execution on x86 processor architecture. Pioneer is based on a challenge response protocol between an external trusted entity called the dispatcher and an untrusted computing platform.

As depicted in fig(1), the dispatcher sends a nonce uniformly chosen at random to the untrusted platform. Pioneer uses a nonce to prevent pre-computation attacks. The untrusted platform uses the checksum code and the nonce it received to compute the checksum of the verification function. The checksum code also sets up the execution environment in which the send function and the hash function can execute untampered regardless of the presence of malware in the untrusted platform. The send function sends the computed checksum to the dispatcher. It is assumed by Pioneer that the dispatcher knows the exact hardware configuration of the untrusted platform and therefore can compute the time required for the checksum to be calculated. If the time exceeds a particular threshold value, then the untrusted platform is compromised and further actions that are implementation defined are taken. Pioneer makes sure that the checksum code runs with the highest privileges, the interrupts are turned off to avoid pre-empting the checksum code, all the NMI interrupt handlers are replaced with an **iret** instruction when the checksum code is executed and the empty slot issues in the super scalar architecture of the processor are eliminated.



1. D: $t_1 \leftarrow$ current time, $nonce \xleftarrow{R} \{0,1\}^n$
   $D \rightarrow P$: $\langle nonce \rangle$
2. P: $c \leftarrow \text{Checksum}(nonce, P)$
3. $P \rightarrow D$: $\langle c \rangle$
   D: $t_2 \leftarrow$ current time
   if $(t_2 - t_1 > \Delta t)$ then exit with failure
   else verify checksum c
4. P: $h \leftarrow \text{Hash}(nonce, E)$
5. $P \rightarrow D$: $\langle h \rangle$
   D: verify measurement result h
6. P: transfer control to E
7. $E \rightarrow D$: $\langle$result (optional)$\rangle$

Fig 1. Pioneer protocol

The checksum itself is strongly ordered to prevent instruction level parallelism. A strongly ordered function is a function whose output differs with high probability if the operations are evaluated in a different order. For example, if a1, a2, a3, a4 are inputs then a1 $\oplus$ a2 + a3 $\oplus$ a4 is strongly ordered. For example the correct order of

Manuscript received October 5, 2013
Manuscript revised October 20, 2013

evaluating the function is (((a1 ⊕ a2) + a3)⊕ a4), If the adversary tries to parallelize the computation by computing (a1 ⊕ a2), (a3 ⊕ a4) in parallel and adding the intermediate results to get the final result, the result will be different with high probability for both the cases.
We give a brief overview of memory copy attacks and how it can be prevented by Pioneer in the x86 architecture and then proceed to show how it can be subverted in a RISC processor architecture like ARM.

2. Attacker model

The adversary is assumed to have complete privileges over the untrusted platform and can tamper with all the software including the operating system itself.

3. Pioneer Architecture

3.1 Data Pointer and Program Counter

Pioneer defines a Data Pointer (DP) and an Program Counter (PC). The Data pointer DP is the pointer to the memory from where the bytes, whose checksum needs to be computed, are fetched. And PC points to address of the next instruction that needs to be executed. In Pioneer, because we perform self-checksum, both the DP and the PC point to the same memory location initially. Pioneer assumes that the external dispatcher knows the correct value of PC and DP.

3.2. Structure of the checksum code

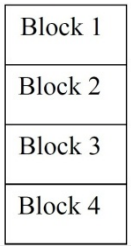


Fig 2. Structure of the checksum code

The checksum code is constructed as a series of 4 blocks, each 128 bytes in length as depicted in fig[2]. The blocks are aligned in memory so that the first instruction in each block is at an address that is a multiple of 128. This simplifies the jump target address generation akin to one depicted below.

Jump target address = base address of block 0 + (((Block-size) * PRN)& 0x3)

3.3. External dispatcher

The external dispatcher is assumed to be trusted and has a copy of the verification function. It also knows the correct DP and PC value. Since the dispatcher generates the nonce based on a PRNG (Pseudo Random Number Generator),it can compute the correct checksum as it already has the verification function code and also knows the correct DP and PC value. The checksum sent by the untrusted platform is verified with this generated checksum and also the time taken for the checksum to arrive is noted based on which the conclusion as to, whether the verification function in the untrusted platform is tampered or not, is derived.

4. Memory copy attacks

Memory copy attacks occur as a result of the adversary modifying either the DP or PC or both. As depicted in fig(3)

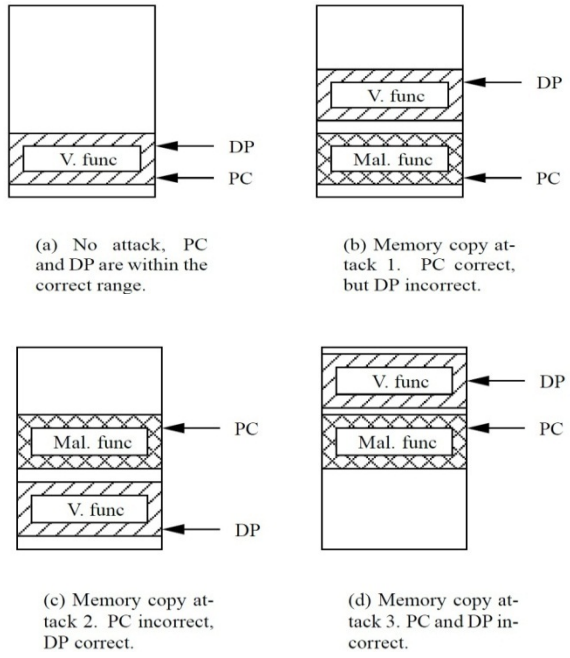


Figure 3. Memory copy attacks

5. How Pioneer protects against memory copy attack in x86 architecture

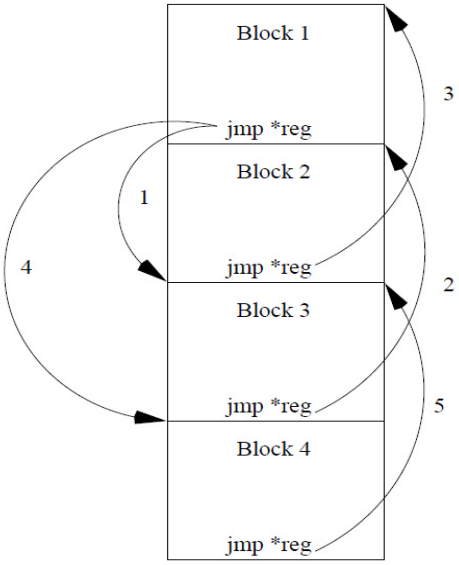


Figure 4. Pseudo random memory traversal

Scenario 1: DP points to incorrect memory location where untampered code for the verification function is stored. PC points to the correct memory location but the checksum code is tampered with.

An adversary can take advantage of the segmentation of the x86 architecture to have the processor add a displacement automatically to the data pointer DP for each iteration of the checksum loop[*].

To illustrate with an example, let us assume that the correct value of the data pointer DP is 0x2000 and the modified DP is at 0x1000. So the adversary needs to add 0x1000 to the DP for each iteration of the checksum loop. But that would normally take an additional **add** instruction which would cause an execution overhead unless we take advantage of the segmentation to automatically add an offset to the DP as in:

mov DP,gs:0x1000; gs = 0x1000 initially

The introduction of **gs** register in the instruction doesn't increase the execution time, however, it increases the instruction length by a byte. Therefore with just the [1]overhead of one instruction that would initially set the value of the segment register **gs** to 0x1000 , we would get the correct checksum. This overhead is too low to cause a significant delay in the timings. Pioneer constructs defenses against these type of memory copy attacks by taking advantage of the increase in instruction length caused by the introduction of the **gs** register and by using pseudo random memory traversal as depicted in fig[4]. Pioneer places an instruction with a memory reference such **add mem, reg** as the first instruction in each of the four checksum blocks. In each of the jump target address computation, we either jump to the starting address of the block or to the address that is 3 bytes away from the starting address of the block. In an unmodified code this works perfectly fine but in a modified checksum code, because of the addition of an extra byte, the second instruction in the next block cannot begin before the 4th byte of the block. Thus 50% of the jumps would land in the middle of the first instruction resulting in the processor to generate an **illegal opcode exception**.

The adversary can overcome this by aligning the checksum code blocks on 256 byte boundaries and separating the first and second instruction of a block by 8 bytes. The adversary has to left shift the jump target address generated by one to jump to the next block and then immediately right shift the address by one to preserve the correct jump target address because the jump target address is also used in the checksum computation. Thus this adds a latency of two instructions for each iteration of the checksum loop. The timing delay is noticeable once the checksum loop is computed. The timing delay can be made more significant by repeating the checksum loop for say, a million times as pointed out by [4].

In the second memory attack, the adversary keeps the unmodified verification function at the correct memory location but computes the checksum using a modified checksum code that runs at a different memory location. In short, DP and its contents are untampered whereas PC and its contents are tampered. Since Pioneer includes the jump target address in its checksum computation, the adversary needs to generate the correct jump target address which could be done in a similar way as mentioned above, but with the added latency of a few instructions.

In the third memory attack, both PC and DP are tampered with and therefore would cause even more latency to generate the right checksum.

6. ARM Overview

ARM processor belongs to a RISC architecture. Using a RISC-based approach to computer design, ARM processors require significantly fewer transistors than processors that would typically be found in a traditional computer. The benefits of this approach are reduced costs, heat and power usage compared to more complex chip designs, traits which are desirable for light, portable,

[*]*The adversary can also use a processor mode which adds an immediate or register value to the data pointer. All general purpose registers are used by the checksum code, so the adversary can only add an immediate value to the DP.*

battery-powered devices such as smartphones and tablet computers [5].

Since ARM is a RISC architecture, all instruction are of same length. In ARM execution mode, each instruction is of size 4 bytes.

6.1. ARM Inline Barrel shifter

The ARM ALU has a 32 bit barrel shifter that is capable of shifts and rotates.

This supports:-

(i) Scaled addressing
(ii) Multiplication by an immediate value
(iii) Constructing immediate values

MOV r0, r0, LSL #1 ; multiply r0 by 2

ADD r1, r1, r1, LSL #4 ; multiply r4 by 17 (r4 = r4 + 16 * r4)

RSB r2, r2, r2, LSL #5 ; multiply r5 by 31 (r5 = 32 * r5 - r5)

7. Subverting Pioneer in ARM

Consider memory copy attack scenario one where the DP points to incorrect memory location containing the untampered verification function whereas the PC points to the correct memory location containing the modified checksum code.

To illustrate this, let the correct value of DP be 0x2000 and the modified version of DP is 0x1000. So for each iteration of the checksum loop, the modified DP value should be added by 0x1000 to get the correct DP value and hence derive the correct checksum to be sent to the external dispatcher. This can be done without any extra memory requirements or execution overhead in the checksum loop by using the ARM Inline barrel shifter as follows

MOV r0, r1, LSL #1 ; r0 is the DP and r1 is the incorrect memory location 0x1000 which is left shifted by 1 using the inline barrel register to get the correct value of DP with no compromise in space and time.

The same technique can also be used for adjusting the jump target address to get the correct checksum in the memory copy attack scenario two and scenario three.

This increases the code length by the size of the immediate.

8. ARM Trustzone overview

(i) The security of the system in Trustzone is achieved by partitioning all of SoC's hardware and software resources so that they exist in either of the two worlds - the **Secure world** for the security subsystem and the **Normal/Non Secure world** for everything else.
(ii) Trustzone splits a physical processor into two virtual processors one running in Secure world and the other running in Non Secure world.
(iii) AMBA3 AXI system bus introduced a new control signal (NS bit) for each of the read and write channel of the main system bus as depicted in Table 1.

	ARPROT	AWPROT
0	Secure	Secure
1	Non Secure	Non Secure

Table 1. AMBA3 AXI control signals for Trustzone

Non Secure world bus masters set NS bit to high thereby prohibiting access to secure slaves.

(iv) Each physical processor core has 2 virtual cores one running in the Secure world and the other in the Non Secure world and a mechanism called **Monitor mode** to switch between these two worlds. The value of the NS bit is derived from whether the secure world or the Non Secure world is making a memory access. Context switch between worlds occur through SMC (secure monitor call) or through exceptions like aborts, FIQ, IRQ

In a ARM processor implementing Security Extensions (Trustzone), there are 3 Exception Vector Tables.

(i) Non Secure World Exception Vector Table (Non Secure World VBAR holds the exception table base address)[**]
(ii) Secure World Exception Vector Table (Secure World VBAR holds the exception table base address) [**]
(iii) Monitor mode Exception Vector Table (MVBAR holds the exception table base address)[2]

8.1 Avoiding denial of service attacks by the Non secure world

The Non Secure world can mask the IRQ, FIQ, Aborts by masking the CPSR.{A,I,F} bits. The secure

[**] - If SCTLR.V == 1, then the exception vector table is always at 0xffff0000 and cannot be remapped. Only when the SCTLR.V == 0, the exception vector table can be remapped. For monitor mode, SCTLR.V is ignored and MVBAR always holds the exception table base address.

world can prevent the modification of the CPSR {A,F} bits by setting the Secure Configuration Register SCR.{FW, EA} bits to 1.

A->Abort

I->IRQ

F->FIQ

The Non Secure world can still mask the IRQ bit of the CPSR. A common mechanism recommended is to make FIQ as the Secure world interrupt and IRQ as the Non Secure world interrupt.

8.2 Determine which world the processor is executing

SCR.NS bit determines in which mode the processor is executing but in monitor mode the processor is always executing in Secure World regardless of the value of the NS bit.

8.3 Securing the MMU

Two virtual MMUs are provided for each of the two virtual processors, so the Non Secure world and the Secure world have their own set of virtual address to physical address translations. In the secure world MMU, the secure world translations has NS bit set to 0. It is possible for the Secure World to access Non Secure World's memory by creating a mapping for the Non Secure world's physical address in the Secure World MMU but with the NS bit set to 1 instead of 0 to denote that it is a Non Secure Memory.

TLB (Translation Lookaside Buffer) is tagged with NSTID bit to avoid redundant invalidates. Cache is also tagged with NS bit to avoid redundant data in the cache.

9. Leveraging Trustzone to achieve trusted code execution on untrusted systems

Since we know the physical address of the Verification function (DP and PC) in the untrusted platform, we can create a mapping for the Non Secure world physical address in the Secure world page table by creating either a section entry or a page entry in it and also setting the NS bit to 0x1.

There is no need for a nonce here since the checksum is computed by the Trustzone which acts as an external verifier in this case. For efficiency, the Integrity Monitor running in the Trustzone can have a copy of the Verification function, pre-compute the checksum and validate the pre-computed checksum with the checksum computed from the Non Secure world's memory. If the checksum matches with the pre-computed checksum, then the Verification function is not tampered with resulting in trusted code execution in an untrusted platform , if not , the verification function has been tampered with and implementation defined actions are taken.

10. Detecting Kernel Rootkits

Most of the rootkits like adore [5] and earthworm [6] target the system call table or the Exception Vector Table. These rootkits can be detected by having a hash of the system call table and the Exception Vector table of the Non Secure world and then mapping the physical address of the system call table and the exception vector table of the Non Secure world to the Secure World MMU. We can then compare the hashes of these data structures with the pre-computed hash stored in the Secure World to check for any inconsistencies.

11. TOCTOU attacks

TOCTOU (Time of Check, Time of Use) can subvert remote attestation as pointed out by [7], [8], [9]. If the adversary knows in advance when the check is going to happen, then the adversary can roll back any malicious changes he made to the Verification Function so that the hash computed in the Secure world would match with the pre-computed hash and once the check is over, overwrite the Verification function so that it behaves maliciously once again. This is a classic example of a TOCTOU attack. This can be prevented by checking the validity of the Verification function in a pseudo random manner so that it cannot be predicted by the adversary. Also, the Secure world has a separate Secure timer which cannot be tampered by the adversary which makes the defense against the above mentioned TOCTOU attack feasible.

12. Conclusion

In this paper we have provided a comprehensive analysis of Pioneer and how it can be subverted in a RISC processor architecture like ARM. We have also provided an overview of Trustzone and how it can be leveraged to provide attestation of software running in the Non Secure world. We have also provided key insights on how the above mentioned can be combined to develop a Kernel Integrity Monitor to detect advanced malware like rootkits. TOCTOU attacks were mentioned briefly and we also discuss defenses against them leveraging the secure timer feature of ARM Trustzone.

References

[1] Arvind Seshadri, Mark Luk, Elaine Shi, Adrian Perrig, Leendert van Doorn and Pradeep khosla. “Pioneer: Verifying Code Integrity and Enforcing Untampered Code Execution on Legacy Systems." 20thACM Symposium on Operating Systems Principles (SOSP 2005).
[2] ARM Security Technology “Building a Secure System using TrustZone® Technology”
[3] Skywing, “Subverting patchguard version2”. Uninformed Magazine
[4] Xeno Kovah, Corey Kallenberg, Chris Weathers, Amy Heroz, Mathew Albin, John Butterworth. “New Results for Timing-based Attestation”. The MITRE Corporation
[5] Packet storm security, “adore-ng rootkit”.http://packetstormsecurity.com/files/32843/adore-ng-0.41.tgz.html
[6] Dong-hoon you, “Android platform based linux kernel rootkit.” Phrack Magazine.
[7] C. Castelluccia, A. Francillon, D. Perito, and C. Soriente. On the difficulty of software-based attestation of embedded devices. In Proceedings of the ACM conference on Computer and Communications Security, CCS, pages 400–409, 2009
[8] U. Shankar, M. Chew, and J. D. Tygar. Side effects are not sufficient to authenticate software. In Proceedings of the conference on USENIX Security Symposium, SSYM, pages 7–7, 2004.
[9] G. Wurster, P. C. van Oorschot, and A.Somayaji. A generic attack on checksumming based software tamper resistance. In Proceedings of the IEEE Symposium on Security and Privacy, pages 127–138, 2005.

R.Sebas Sujeen received his B.E degree from the College of Engineering, Anna University Chennai. Currently employed in a startup specializing in ARM Trustzone and Hypervisor development. His research interest includes operating systems, system and information security.

Sridhar Periasami received his B.E degree from the College of Engineering, Anna university, Chennai.Currently employed in a startup specializing in ARM Trustzone and Hypervisor development. His research interest includes operating systems, web application and system security.