Inspecting data from the safety of your trusted execution environment

John Williams
johnwwil [at] u.washington.edu
June, 2015

Abstract: This paper presents a proof of concept that uses ARM TrustZone to perform introspection of a Linux kernel running in the normal world from within a secure-world system. Techniques from existing volatile-memory analysis applications are repurposed for application in real-time through asynchronous introspection of the normal world. The solution presented here leverages open-source software that is actively being developed to support additional architectures.

1 INTRODUCTION

As technology becomes more prevalent in our day-to-day lives, we increasingly rely on mobile devices for both business and personal purposes. Emerging security threats, such as rootkits that can scan memory for protected PCI information, have been responsible for large-scale breaches. It has been predicted that targeted exploit kits and attack services will soon exist for mobile platforms, which could enable the exploitation of mobile devices on an unprecedented scale [1].

Techniques for mitigating such threats to mobile devices have largely evolved in parallel with industry demand for them. While some companies have indeed made progress through continued addition of product features that increase the security of devices, other companies have simply pivoted their marketing efforts to emphasize or recast certain aspects of their existing capabilities.

In order to select the right products and features for protecting sensitive data and activities, effective segmentation is key. While segmentation within devices has traditionally been implemented at the operating system level, in recent years architectural modification at the chip level has emerged as a way to provide hard segmentation between execution environments. It has not been trivial for researchers to use these extensions in the prototyping of security solutions that protect against rootkits and malware. This paper explores the implementation of complex features for mobile devices that take advantage of this hardware segmentation.

2 BACKGROUND

As it stands today, the complexity and attack surface of the average operating system kernel is large. The goal of this project is to assess the groundwork for the development of novel security capabilities which enable inspection of the normal world operating system from the secure world environment. ARM

TrustZone extensions are focused on here due to their pervasiveness and flexibility in mobile applications.

2.1 ARM TRUSTZONE

TrustZone provides as set of architectural design elements for partitioning execution environments into separate "worlds": a normal world and a secure world. TrustZone has been in production since 2003, and its capabilities have been included within devices with varying levels of support dictated by the integrator of the specific system-on-chip. TrustZone provides full partitioning of hardware resources, for example:

> "TrustZone technology is tightly integrated into Cortex®-A processors but the secure state is also extended throughout the system via the AMBA® AXI™ bus and specific TrustZone System IP blocks. This system approach means that it is possible to secure peripherals such as secure memory, crypto blocks, keyboard and screen to ensure they can be protected from software attack"[2].

2.2 TRUSTED EXECUTION ENVIRONMENTS

"Devices developed with TrustZone technology, according to the recommendations of the Trusted Base System Architecture specification, enables [*sic*] the delivery of platforms capable of supporting a full Trusted Execution Environment (TEE) and security aware applications and secure services, or Trusted Applications (TA). A Trusted Execution Environment is a small secure kernel, and normally developed with standard APIs, developed to the TEE specification evolved by the Global Platform industry forum" [2].

For the purposes of this paper, I use the term Trusted Execution Environment (TEE) loosely to mean a secure environment that is isolated from the "rich" normal world environment rather than one that is Global Platform standards compliant. Today, TEEs are designed to securely implement applications such

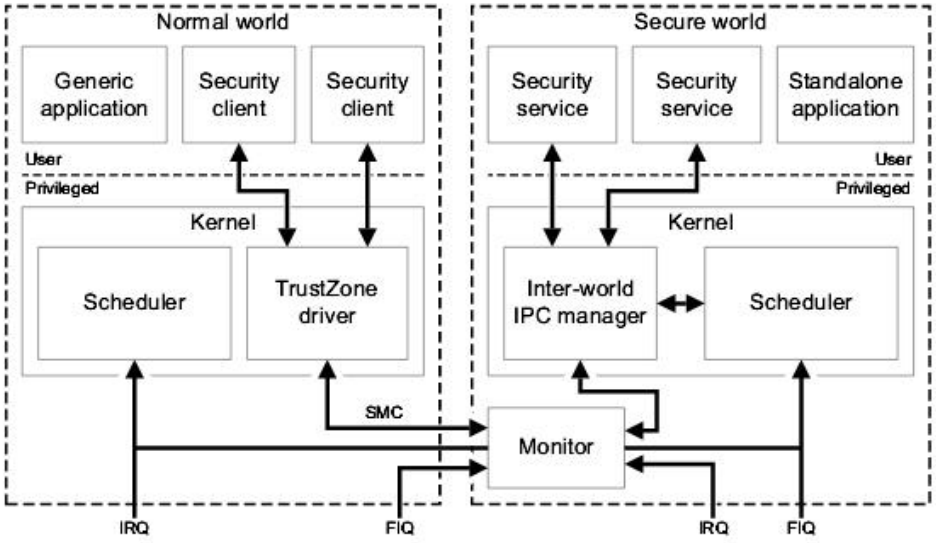

Figure 1. Secure/Normal world split [3]

as DRM and secure payments.

A TEE is designed to be minimally complex, providing a trusted computing base (TCB) which may be treated as a cryptosystem providing secure services. While the ultimate goal is to minimize complexity,

the trusted environment is fully capable of taking advantage of the full power of devices which increase substantially with every generation. This allows for the development of more significant applications to provide novel security functions which may be designed in such a way to minimize the attack surface of the trusted world.

As shown in Figure 1, a complex operating system running parallely in the secure world may take advantage of traditional privilege separation in the secure world for providing protection between the secure services. It may also share execution and storage resources with the normal world.

	sp<n> field controls if the TZASC permits access for the following AXI transactions			
sp<n> field[a]	Secure read	Secure write	Non-secure read	Non-secure write
b0000	No	No	No	No
b0100	No	Yes	No	No
b0001, b0101	No	Yes	No	Yes
b1000	Yes	No	No	No
b0010, b1010	Yes	No	Yes	No
b1100	Yes	Yes	No	No
b1001, b1101	Yes	Yes	No	Yes
b0110, b1110	Yes	Yes	Yes	No
b0011, b0111, b1011, b1111	Yes	Yes	Yes	Yes

a. See *Region Attributes <n> Register* on page 3-20 for programming information.

Figure 2. Secure/Non-secure operating system configuration [3]

This structure is a natural complement to the running of a separate high-assurance operating system providing a basis for services with minimal attack surface, but which may not be able to compete with the features of standard operating system from a user perspective.

2.3 HARDWARE REQUIREMENTS FOR SECURE-WORLD ENVIRONMENT

In order to simultaneously instantiate both a secure world and a normal world environment, all of the resources used by the secure world must be protected. Modern ARM System-on-Chips (SOC) that leverage the ARMv7 architecture provide TrustZone extensions in the CPU.

However, support in the MMU (for protecting memory) and in the interrupt controller (for enabling secure interrupts) are only provided by some manufacturers. Additionally the boot ROM must provide access to the secure mode to allow configuration of system resource.

2.4 REQUIREMENTS FOR INTROSPECTION

With the proper support from supported IP blocks, the ARM TrustZone architecture allows for controlling of memory regions within the system through the use of a TrustZone Address-Space Controller (TZASC). The TZASC allows specific memory spaces to be declared as either secure or non-secure. Figure 2 shows the possible configurations in the TZASC for memory regions not including those covered by memory

inversion.

In order to achieve the target application of performing introspection on the normal world from the secure world it is required to have the memory protection configured to allow the secure world access to the non-secure regions. It should also be noted that the configuration of the TZASC does not necessarily need to be static for the lifetime of the system and may be dynamically configured at runtime.

2.5 VOLATILE MEMORY-ANALYSIS TOOLS

The development of forensics techniques for mobile technology has enabled investigations into security incidents, malware analysis, and criminal activity. Open-source tools like Volatility [8] and Rekall [9] have significantly advanced analysis in this area. Furthermore, there are many robust modules provided between these tools that are commonly used for malware analysis and incident response. These tools have laid the groundwork for bridging the semantic gap between raw memory and the structures contained within and provide established methods for how to learn about a system without relying on its APIs.

3 RESEARCH CHALLENGES

Investigations into TrustZone by other researchers have identified significant challenges in experimenting with and prototyping security functions that leverage a secure/normal world split, namely the availability of hardware that: a) has the right hardware features, b) is not manufacturer-locked, and c) is relatively affordable.

End-consumer platforms tend to be locked down and often contain proprietary implementations of secure world systems which are locked down to prevent access and modification. Even when these proprietary systems can be modified, they often do not contain the ability to perform memory segmentation as previously mentioned. Official ARM development boards are highly supported but are not affordable and do not necessarily provide a platform that is representative of commodity hardware.

4 SECURE-WORLD OPERATING SYSTEM

In order to provide effective security, the secure-world operating system must be of minimal size and complexity. An open-source design is also advantageous to facilitate broader development of secure-world applications. Additionally, the typical internal TEE interface provides a custom interface designed for specific use cases and does not necessarily provide a POSIX-compatible API that can be used to develop portable applications.

Anticipating leveraging existing POSIX-compatible applications, Genode was targeted due to the degree of flexibility and hardware support at present.

4.1 GENODE

For the purposes of this research, I sought an open-source trusted environment that provides support for hardware that is widely accessible. I evaluated several different solutions and have found significant fragmentation in the area of hardware platform support. Ultimately Genode was chosen based on the flexibility of the solution and its active development community.

“Genode is a novel OS architecture that is able to master complexity by applying a strict organizational structure to all software components including device drivers, system services, and applications” [10]. Genode implements a microkernel-like architecture that allows for capability-driven security. A

compelling demo solution has been provided which demonstrates how to leverage TrustZone for advanced virtualization-like capabilities on the i.MX53 [11].

4.2 NOUX

The Genode project also provides Noux, an "experimental implementation of a UNIX-like API that enables the use of unmodified command-line based GNU software" [12]. Noux provides a minimum-complexity libc interface, 'noux_libc,' designed for running POSIX applications within the Genode environment. Noux provides segmentation between user and system within the secure world and also isolation between application instances in the secure user space.

4.3 EXISTING 'TZ_VMM' APPLICATION

The sample 'tz_vmm' target allows for the basic bring-up of a system in the normal world. This demo is designed to configure the system to run Genode within the secure world and to properly configure and boot a Linux kernel in the normal world. I use this application as a basis for developing the components used for introspection. See [11] for details on the initial construction of this system.

4.4 HARDWARE USED

The solution was developed and tested on the i.MX53 platform which includes the USBArmory board as well as the i.MX53 Quick Start Board (QSB). Unofficially, there is support available for the i.MX6 platform which would bring this solution to a significantly wider base of devices and allow for broader experimentation with this platform. The USBArmory takes advantage of existing work to provide a platform for experimentation on the i.MX53 platform.

5 EXTENDING GENODE TO SUPPORT INTROSPECTION

5.1 LEVERAGING TZ_VMM AS BLOCK DRIVER

In order to access the normal world memory from within the Noux environment, a block driver was developed to provide access to Noux applications through a block-level interface. The block driver was developed by combining the tz_vmm application with the ram_fs driver.

The block driver also serves to provide an abstraction from the hardware implementation and provides finer control over accesses made to the normal world memory — the only knowledge required in order to perform memory lookups is the normal world memory base address.

This block driver is exposed to all applications running under Noux within the secure world however access permissions can be tailored based on the specific functionality required.

5.2 ANDROID/LINUX GUEST

In order to successfully run the Linux kernel in the normal world with Genode in the secure world, Linux must be modified to prevent the normal world operating system from accessing hardware and memory that is marked as secure. This requires that any secure hardware be accessed indirectly through the secure monitor rather than directly via traditional access methods.

5.3 PYTHON

The goal of this project was to experiment using a platform for the development of fairly complex security application. For maximum code reuse, ideally Python would be available for running the existing modules. I was able to successfully compile Python 2.6 which is supported in the Genode libports for x86.

5.4 WORLD SWITCHING

Common usage of a secure world system is to provide services to the normal world which can be accessed synchronously through the usage of the secure monitor entry command, SMC. The SMC command allows the normal world to make a request or otherwise switch to the secure world. For the purpose of this application, we require asynchronous transition of control back to the secure world without relying on an explicit transition from the normal world. There are multiple ways to achieve preemption of the normal world:

5.4.1 Hardware timer

Hardware timer is designed as secure which allows the secure world to schedule itself and to preempt the normal world. An alternative approach on other platforms such as the i.MX6 would involve developing a kernel module that uses the SMC instruction to synchronously transition to the secure world and to configure the secure TZ Watchdog 2 to require world switch to prevent DoS.

5.5 SYSTEM CONFIGURATION

The configuration system used by Genode allows for the flexible assignment of resources within the system. In my test, I configured the system to use the noux_libc and to have access to the hardware timer for scheduling purposes. The directories containing my binaries are also loaded using a tar filesystem. Noux loads the block driver which is configured in the run file and exposes the block device to various applications. For more details on the exact configuration used see Appendix A. Genode Configuration.

The i.MX53 uses the Multi-Master Multi-Memory Interface (M4IF) rather than the TZASC for protection of memory and the TrustZone configuration within Genode by default allows for secure world access to non-secure memory. The UART was configured to allow output from both worlds in order to use the normal system while simultaneously receiving output from the secure world.

6 DEVELOPING INTROSPECTION APPLICATION

The previous section outlined the steps for developing the system to enable support complex applications. In this section, I discuss the path traversed for developing a defensive security application that leverages the introspection capabilities to perform verification of the system call table.

6.1 VOLATILITY 'CHECK_SYSCALL_ARM' MODULE

As a starting point for my new introspection application, I use the techniques employed in the 'check_syscall_arm' plugin included in Volatility. This plugin is designed to be used for offline analysis of a memory dump to detect modifications to the system call table in Linux. The operation of this application can be broken down into (1) discovery of the parameters of the table from the System.map and parsing the code section of memory, and (2) comparison of the system call table to ensure that each of the addresses are included in the System.map.

6.2 EXPERIMENT RUNNING PYTHON MODULE

Genode supports the compiling of applications for execution within the Noux environment natively. Given that Python 2.6 is a supported target, I first attempted to execute the module within Python running natively. After a few hurdles such as compiling the normally dynamic standard native Python modules statically within Python, modifying Noux to support the number of file descriptors required, and packaging the standard libraries, I was able to run the module natively. However, the performance was very poor on the target tested and it became quickly apparent that the performance would not be sufficient for real-time applications.

It may still be that a faster target and optimization of the implementation would allow for this to be a viable solution. However, even if the execution time were to be sufficient with this test, the module itself is not written to be executed in real-time nor to be dynamically analyzed using test cases.

6.3 NATIVE INTROSPECTION APPLICATION

Given that the Python implementation on the target is not currently viable, the techniques used in 'check_syscall_arm' were used as a starting point for the development of a native application that performs this check at runtime. Porting this application allowed us to tailor the approach to the specific application at hand by optimizing the implementation for being run in real-time and to develop code that lends itself to dynamic coverage analysis.

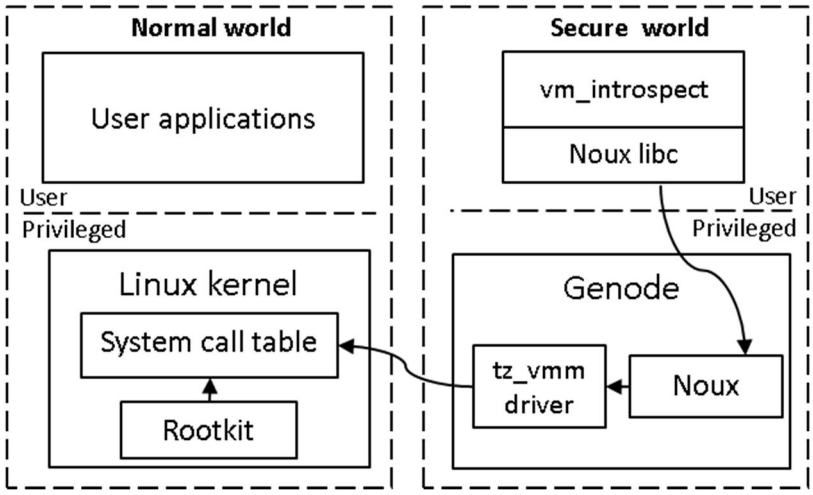

Figure 3. Architecture for introspection application PoC

6.3.1 System.map incorporation

The System.map file containing the normal-world kernel symbol table is required to be packaged. This provides us with the set of acceptable addresses and is incorporated into a tar file which is loaded into the image.

6.3.2 Application execution

The module itself has three distinct states: (1) secure world boot initialization, (2) initial normal-world runtime execution, and (3) steady-state operation. In the secure world boot initialization stage, the application is the last item to be configured. Figure 3 above shows the architecture of the solution.

6.3.2.1 Secure-world boot initialization

The initial execution of Genode in the secure-world environment is responsible for configuring secure access to system resources and constructing the various services. Because the tz_vmm is now a block driver which is initialized before the custom application, I configure the module to only start after it receives a signal from the custom application so that I can perform initialization tasks.

The initial construction activities consist of parsing the System.map text file to determine the addresses for the required runtime symbols and creating a table of valid symbol addresses to reference when performing validation of the system call table.

6.3.2.2 Normal-world runtime initial execution

The initial execution of the application provides the first opportunity to run after the normal world operating system has constructed. We must ensure that the delay is large enough for the normal world OS to fully construct. This allows my application to perform initial validation of the system call table.

6.3.2.3 Steady-state operation

In the steady-state runtime operation, the execution of the application continues to rely on the preemption of the normal world by the secure world through the use of the hardware timer. The introspection application may perform a faster validation than initially by checking the CRC of the table, however this was not implemented.

6.4 PERFORMING VALIDATION OF ROOTKIT DETECTION

In order to validate the detection of the system call hooking detection the Mindtrick [13] rootkit developed specifically for Android devices was employed to perform validation that the system call table hooking was effectively detected. The introspection application test involves hooking the system call table and reading out the modified entries. We are able to detect these modifications.

7 LIMITATIONS AND FUTURE WORK

The current solution is designed and tested on the on i.MX53, a Cortex-A8 SOC which is a single-core system. Enhancements can be made to this solution to make it more robust including reading the System.map from the normal world directly. The initial runtime execution could also be determined programmatically rather than through a delay.

Additionally, the proof-of-concept code is written in such a way as to facilitate dynamic code analysis under test which would be expected for a high-assurance security function. Writing to the normal world from the secure world is also implemented but untested. This allows secure response through modification of the normal world memory. Ultimately this demonstrates a platform for analysis of guest systems to which new features can be added to provide a secure agent that implements host-based intrusion detection capabilities.

8 CONCLUSION

In this paper I have demonstrated how ARM TrustZone, along with the correct hardware support, allows for introspection of the normal world system and supports performing complex analysis in real-time.

I have presented a solution that is built on top the Genode operating system framework to allow for the asynchronous execution of unprivileged applications in the secure world which have complete access to

the normal world memory.

Finally, I have enumerated the steps necessary for taking advantage of existing volatile memory analysis techniques to be employed in the creation of a custom application that implements validation of the normal world system call table.

9 BIBLIOGRAPHY

1. "Sophos Trends and Predictions 2015." https://www.sophos.com/en-us/threat-center/medialibrary/PDFs/other/sophos-trends-and-predictions-2015.pdf.
2. TrustZone - ARM, from http://www.arm.com/products/processors/technologies/trustzone/index.php
3. "Trusted Execution Environment." 2015. https://www.trustonic.com/products-services/trusted-execution-environment.
4. ARM Security Technology Building a Secure System using TrustZone Technology. http://infocenter.arm.com/help/topic/com.arm.doc.prd29-genc-009492c/PRD29-GENC-009492C_trustzone_security_whitepaper.pdf
5. "Reflections on Trusting TrustZone." https://www.blackhat.com/docs/us-14/materials/us-14-Rosenberg-Reflections-on-Trusting-TrustZone.pdf.
6. "CoreLink TrustZone Address Space Controller TZC-380 Technical Reference Manual" http://infocenter.arm.com/help/topic/com.arm.doc.ddi0431c/DDI0431C_tzasc_tzc380_r0p1_trm.pdf.
7. Inverse Path - USB armory. http://inversepath.com/usbarmory
8. The Volatility Foundation - Open Source Memory Forensics. http://www.volatilityfoundation.org/
9. Rekall Memory Forensic Framework., http://www.rekall-forensic.com/
10. Genode - Genode Operating System Framework. http://genode.org/
11. "An Exploration of ARM TrustZone Technology". http://genode.org/documentation/articles/trustzone
12. "Genode components overview." http://genode.org/documentation/components
13. "Mindtrick Android Rootkit". https://github.com/ChristianPapathanasiou/DEFCON-18-Android-rootkit-Mindtrick.
14. "Multi-Tiered Security Architecture for ARM via the Virtualization and Security Extensions." https://www.sec.in.tum.de/assets/Uploads/lengyelshcis2.pdf.
15. "Trustworthy Execution on Mobile Devices: What security properties can my mobile platform give me?" http://users.ece.cmu.edu/~jmmccune/papers/VaOwZhNeMc2012.pdf.
16. "SeCReT: Secure Channel between Rich Execution Environment and Trusted Execution Environment. http://gsis.kaist.ac.kr/cysec/publications/NDSS_2015_SeCReT.pdf.
17. Azab, Ahmed M., Peng Ning, Jitesh Shah, Quan Chen, Rohan Bhutkar, Guruprasad Ganesh, Jia Ma, and Wenbo Shen. "Hypervision Across Worlds: Real-Time Kernel Protection from the ARM TrustZone Secure World." In *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security*, 90–102. CCS '14. New York, NY, USA: ACM, 2014. doi:10.1145/2660267.2660350.
18. "Exploring Virtualization Platforms for ARM-Based Mobile Android Devices." http://moss.csc.ncsu.edu/~mueller/ftp/pub/mueller/theses/ramasubramanian-th.pdf.
19. "SPROBES : Enforcing Kernel Code Integrity on the TrustZone Architecture". http://arxiv.org/ftp/arxiv/papers/1410/1410.7747.pdf.
20. "A Scalable High-Performance In-Memory Key-Value Cache using a Microkernel-Based Design." http://www.sra.samsung.com/assets/Uploads/TR-SRA-SV-CSL-2014-1.pdf.
21. "TrustDump: Reliable Memory Acquisition on Smartphones." http://www.cs.wm.edu/~ksun/csci780-f14/papers/0-trustdump.pdf.
22. "Building Trusted Path on Untrusted Device Drivers for Mobile Devices". http://www.liwenhaosuper.com/blog/wp-content/uploads/2014/06/trust-ui.pdf.

10 APPENDIX A. GENODE CONFIGURATION

This section contains the Genode configuration which runs the vm_introspect proof-of-concept.

```
<config verbose="yes">
	<parent-provides>
		<service name="ROM"/>
		<service name="RAM"/>
		<service name="IRQ"/>
		<service name="IO_MEM"/>
		<service name="CAP"/>
		<service name="PD"/>
		<service name="RM"/>
		<service name="CPU"/>
		<service name="LOG"/>
		<service name="SIGNAL"/>
		<service name="VM"/>
	</parent-provides>
	<default-route>
		<any-service><any-child/><parent/></any-service>
	</default-route>
	<start name="timer">
		<resource name="RAM" quantum="1M"/>
		<provides><service name="Timer"/></provides>
	</start>
	<start name="terminal">
		<binary name="log_terminal" />
		<resource name="RAM" quantum="1M"/>
		<provides><service name="Terminal"/></provides>
	</start>
	<start name="ram_fs">
		<resource name="RAM" quantum="10M"/>
		<provides><service name="File_system"/></provides>
		<config>
			<policy label="noux -> root" root="/" />
		</config>
	</start>
	<start name="noux">
		<resource name="RAM" quantum="100M"/>
		<provides>
			<service name="Noux"/>
		</provides>
		<config verbose="yes">
			<fstab>
				<tar name="pylibs.tar" />
				<tar name="vm_introspect_server.tar" />
				<dir name="ram"> <fs label="root" /> </dir>
				<dir name="dev">
					<terminal name="terminal" label="terminal_fs" />
					<block name="blkdev0" label="block_session_0" />
				</dir>
			</fstab>
			<start name="/bin/vm_introspect_server"> </start>
		</config>
	</start>
	<start name="tz_vmm">
		<resource name="RAM" quantum="14M"/>
		<provides><service name="Block" /></provides>
	</start>
</config>
```