

# VERIAL: Verification-Enabled Runtime Integrity Attestation of Linux

Michael Neises <sup>a</sup>  
*neisesmichael@gmail.com*

**Keywords:** Runtime Attestation, Remote Attestation, Virtual Machine Introspection, seL4, Operating Systems, Integrity, Confidentiality.

**Abstract:** Runtime attestation is a way to gain confidence in the current state of a remote target. That confidence is valuable in safety-critical systems such as those involving health-care or finance. Layered attestation is a way of extending that confidence from one component to another. Solutions for layered attestation require strict isolation. Without that isolation, the boundaries between components are blurred, and trustworthy components become intermingled with untrustworthy ones. The seL4 is uniquely well-suited to offer kernel properties sufficient to achieve such isolation. While not the world's only kernel with some formal verification, the seL4 is significantly more advanced than any other solution while uniquely offering the possibility of real-time computation. I design, implement, and evaluate introspective measurements and the layered runtime attestation of a Linux kernel hosted by the seL4. VERIAL can detect diamorphine-style rootkits with performance cost comparable to previous work.

## 1 INTRODUCTION

Network-connected devices are ubiquitous. Estimates for such devices number in the tens of billions, and that number grows by billions every year (?). A significant portion of these are autonomous “Internet of Things” (IoT) devices, which can be operated remotely. A popular ironic phrase has emerged: “the S in IoT stands for security,” and it underscores the common neglect of security in IoT system design (?; ?). And yet these devices find popular use in many sectors: agriculture, energy, finance, health-care, manufacturing, retail, hospitality, and transport and logistics (?). In 2018 it was determined that more than a third of people claim they depend on their devices to get through their daily lives (?). Network-connected devices are everywhere, and they are critical to many people and occupations.


Some network-connected devices perform trust-critical operations. In the health-care setting, IoT devices are commonly used to monitor patient health; the monitoring machine must be trusted to provide accurate reports. In the agriculture and transportation sectors, IoT devices control large machines that sometimes have human passengers; these large machines must be trusted to operate normally. In the finance sector, many people use network-connected devices to accomplish their banking and investing tasks; these

software must be trusted to handle money appropriately. In manufacturing, network-connected machines must be trusted to build items to specification. Seeing potential for networked voting to bring accessibility to the polls, we must build trustworthy mechanisms if this is ever to happen. Many people routinely engage in risky behavior by trusting network-connected devices.

Attestation is a method by which trust in a remote target can be established (?). During attestation, a remote target makes claims about its properties by supplying evidence that is appraised by another so that the other might come to trust the target. In the simplest case, if a remote target attests that its kernel is unmodified, we can afford to trust it more than if we had no such report. However, that trust expires; just because we completed one attestation does not mean the system cannot be later attacked. Even if a machine boots into a correct state, it is important to maintain and verify that correctness over time. In any case, remote attestation is a valuable tool with which we can increase the trustworthiness of networked devices upon which so many depend.

A critical part of attestation is measurement. The measurer creates evidence by collecting certain data about its target. Evidence collection primarily concerns the integrity of critical subsystems. For example, an unexpected change to a kernel's internal code indicates the possibility of unauthorized access. A

---

<sup>a</sup>  <https://orcid.org/0000-0002-5464-3269>

more advanced measurement might detect changes to shared libraries such as what permitted the XZ-utils backdoor (?). Evidence collection can also concern verifying the nonexistence of unauthorized software. For example, the appearance of an unknown program on a server with a well-known workload indicates the possibility of intrusion. Measurement should show that extant systems have integrity and that no other systems are present.

Soundness of measurement is a primary concern of this dissertation, so VERIAL was designed and implemented with the principles of attestation in mind (?). There are five principles for attestation architectures: fresh information, comprehensive information, constrained disclosure, semantic explicitness, and trustworthy mechanism. “Trustworthy mechanism” was the primary guiding principle for this work. Here, trustworthy mechanism refers to the separation of measurement tools from the measurement target, so that a possibly corrupted target cannot interfere with the measurement being taken. A good remote attestation architecture must be able to protect itself. If the attestation tools are corrupted, they are no longer trustworthy. For that reason, the measurer must be separated from its target.

The separation of the measurer from its target can be understood in terms of confidentiality and integrity. The measurer has confidentiality if its target cannot read its data. The measurer has integrity if its target cannot change its data. Confidentiality and integrity are traditional security properties that are basic goals of all trustworthy systems. Effectively enforced separation limits communication to only those components explicitly allowed to communicate. In general, inter-component access is not permitted, but special exceptions are sometimes made.

Software solutions for attestation typically provide separation by involving a virtual machine (VM). Without virtualization there is only one kernel, and the measurer has no separation from its target. But with a VM, a measurer need not depend on its target because it can use a technique called virtual machine introspection (?). Virtual machine introspection allows a host to view the memory contents of a virtual machine. This provides some separation between the measurement tools and their target. However, this technique has proven vulnerable to a class of attacks called “Virtual Machine Escape” exploits (?). In addition, although never before seen in the wild, there is a style of attack called “Hyper-jacking,” in which a virtual machine manager is subverted from within to allow programs to run on the system beneath the manager (?). It is clear that different OS technologies are required to create a more trustworthy system.

An architecture built on seL4 (?) is one way to faithfully implement the separation principle for remote attestation. The seL4 microkernel provides the separation required for measurement tools to be considered trustworthy. The seL4 is machine-proven to correctly implement the properties of integrity and confidentiality (?). This means the data of one component cannot be changed or read by another component without prior explicit permission. These properties permit strong separation between measurement tools and their subject. In this work, these permissions are afforded exactly once at compile-time; the permissions are not changed during run-time. This is ideal for an attestation architecture that needs to make measurements during run-time.

The contribution of this work is an **seL4-based operating system with an embedded measurement suite capable of detecting integrity violations in the Linux kernel and its userspace data**. The operating system is called VERIAL: Verification-Enabled Runtime Integrity Attestation of Linux. This system is distinguished from prior work by its foundation on and inclusion of verified software. The seL4 provides a foundation upon which entire classes of vulnerabilities can be discarded, from buffer overflows to virtual machine escape exploits. The measurements suffice to reveal a variant of the diamorphine (?) rootkit and other unexpected changes of vulnerable data. We achieve the goal of VERIAL by achieving the following subgoals:

- An seL4-based OS implementing this attestation architecture.
- An introspection library and many measurements over Linux.
- An implementation of a rootkit.
- Evaluation against various data integrity breaches

## 2 RELATED WORK

### 2.1 Software-Based Introspection

Software-based introspection is the process of examining a guest virtual machine from an independent task for the purposes of attesting to the running state of the guest VM. The removal of the measurement tools from the guest VM ensures a degree of separation between the two. This is advantageous because it helps to ensure that if the VM is compromised, the measurement tools will be safe. Software-based introspection has the disadvantage of historically not providing sufficient separation. Namely, virtual machine escape exploits have proven this separation weak.

Volatility (?) is a python project that facilitates the inspection of memory snapshots. It implements several plugins for the investigation of the memory snapshot. Each plugin implements some investigative procedure that acts on the memory snapshot. Volatility is not a hypervisor-based solution, and cannot be used to report on a running system. To create a memory snapshot requires loading a specific kernel module. However, the process of interpreting the memory remains the same in hypervisor-based solutions. One advantage of Volatility is that snapshots can be stored or even moved for inspection at a later time.

There exists an application that facilitates the introspection of VM memory managed by seL4's virtual machine manager (VMM) (?). In particular, it arranges permissions such that an seL4 process is able to read the memory associated with the VM's Linux instance. That memory is provided in terms of a pointer to its first byte. In this way, such an seL4 process can be independent from a VMM, yet have complete read-access to its VM's memory.

VERIAL is forked from this example application. A trivial measurement that could be taken is the complete snapshot of Linux memory. Unfortunately this measurement will be not be useful because even a small change in any bit of VM memory should result in a totally different hash digest; it would be infeasible to create a collection of acceptable digests. So a more contextual approach to measurement is desirable from the standpoint of appraisal, and LKIM is a great example of contextual measurement (?).

A certain degree of abstraction is desirable to improve the utility of the measurement. Therefore, the memory snapshot should be partially interpreted, and only certain parts of memory should be collected for measurement, depending on the state of the system in question.

Garfinkel et al (?) presents the idea of virtual machine introspection, which they define as an "approach to intrusion detection that co-locates an Intrusion Detection System on the same machine as the host it is monitoring and leverages a virtual machine monitor to isolate the IDS from the monitored host." This approach renders the IDS resistant to attack, because it is isolated from the monitored host, and resistant to evasion, because the IDS still maintains high visibility of the target. VERIAL capitalizes on resistance to attack by its foundation on seL4.

Terra (?) is founded upon a trusted virtual machine manager. This TVMM is used to partition tamper-resistant hardware into a set of virtual machines. These virtual machines can either have open or closed semantics. Open semantics refers to running a commodity OS, a general purpose system where

anything can happen. Closed semantics, on the other hand, refers to a virtual machine that runs just one application, for which certain appropriate security assurances can be given statically. In this way, one machine can host several applications and operating systems at different levels of assurance.

Terra is a trusted way to partition system resources. Terra does not perform or specify any integrity measurement on the open systems, but only provides measurements of the closed systems. There is a question of whether Terra or seL4 provides a more trustworthy VMM. We believe the movement of data within seL4 is more well-understood than in the Terra case. However, Terra always leverages "tamper-resistant hardware," which is something allowed yet not prescribed by the seL4.

Baiardi et al (?) uses virtual machine introspection to monitor the run-time state of a target process. The authors were required to modify the monitored kernel in order to trace system call invocations. In doing so, it allows the monitor kernel to verify that all system calls follow valid invocation paths. The core idea of this technique is that all security-relevant action happens during system calls. For this reason, it makes sense to check the actual sequences of system calls, and to make sure these actual sequences are valid by way of static analysis on the source code.

The measurement technique of tracing system call invocations is only valid for a given program, but it would appear that any program could be measured and appraised in this way. This describes a distinct style of user-space measurement, and it would be a useful part of any such toolkit. Perhaps this trace analysis technique could be applied to measurements for kernel integrity. This type of introspection requires modification of the target kernel. This causes additional expense because modifications must be propagated through every kernel update. While the ability to trace system call invocations has value, this additional cost might not always be acceptable.

## 2.2 Hardware-Based Introspection

Hardware-based introspection is much like the software-based case, but the separation between the measuring agent and the guest VM is at the hardware level. Because these two parties are executing on separate processors, it becomes much more difficult, if not impossible, for the guest VM to exploit the measuring agent.

Beyond isolation, hardware solutions can also benefit from speed. If measurement is performed by another machine, then the target need not sacrifice any resources. Sometimes the initial cost of a hardware

solution can be outweighed by its lifetime benefits.

Hardware solutions are expensive because they require two machines in order to measure one machine. In some cases, two machines are shipped in the same platform, and one measures the other. In other cases, a single machine is responsible for measuring many machines.

The primary disadvantages of hardware-based introspection are that it is greatly limited in the systems to which it can apply and even in those systems it is expensive. This technique has been separately implemented using an onboard co-processor, a PCI device, and a smart NIC device. Solutions are varied, but the requirement for hardware support limits applicability.

Copilot achieves isolated kernel integrity monitoring by way of having an independent machine to monitor another using a PCI card (?). Copilot depends on having an IBM PC-compatible PCI bus and is only targeted at Linux virtual memory. At the time of publication, Copilot could reportedly detect a rootkit within 30 seconds of corruption.

Zhang et al (?) proposes “building intrusion detection systems using external secure co-processors,” much like what is done in the Copilot approach. They give several examples of monitoring applications that can be implemented with a co-processor: kernel invariants can be checked, file integrity can be checked, and memory can be scanned for viruses.

Zhou et al achieves hardware-based introspection needs via the Intel Management Engine (?). The Intel Management Engine is an autonomous subsystem has been integrated in virtually all of Intel’s processor chipsets since 2008. However, many researchers consider the engine to be a backdoor (?). In fact, the engine is a popular target for hackers because it has top-level access to devices and utterly bypasses the OS. As such, several types of exploits have been demonstrated that effect large swaths of Intel machinery.

Hongyi Liu et al employ a technique called Remote Direct Memory Access (RDMA) to perform introspection(?). RDMA requires a network interface controller (NIC) that is specially equipped to access the target memory. RDMA permits zero-copy networking by enabling applications to directly access application memory in another machine. This new technique called RDMA performs introspection using RDMA, and it uses one machine to measure several dozen with impressive performance results.

The RDMA technique is primarily used in high-performance computing (HPC) environments. RDMA is advantageous in cluster computing because the CPU of the target takes no action before, during, or after its memory is read. Typical network costs are bypassed because cluster nodes forsake their isolation

in union.

RDMA is problematic for internet-based attestation. The NIC can filter RDMA requests, but if the NIC provides an abstraction over memory, then the access is not direct. The ability to choose the supplied abstraction over memory is a principle of attestation: constrained disclosure. The RDMA framework could be adapted to permit these decisions to be made in the NIC, in which case the solution would look like Copilot.

## 2.3 Attestation

Attestation can involve introspective measurement, but it is also studied independent of introspection. Architectural design is a critical consideration for attestation systems, and such designs can abstract away the details of particular measurements. In addition, some measurements blur the line between software and hardware and will be mentioned in this section.

Eldefrawy et al claim to be the first to introduce an architecture for measurement using the seL4 microkernel (?). HYDRA can be used to report on the state of a single specific seL4 process. This approach is sound because the footprint of a seL4 process is simple, but for that same reason this approach lacks flexibility. In contrast, our work is aimed at reporting on the state of a Linux kernel that is virtualized by seL4. Not only this, but we also report on the state of all those programs executing in the Linux context. Ultimately, we believe our approach to combine the trustworthiness of the HYDRA approach with the general-purpose functionality of VM introspection techniques.

Paul Rowe introduces a convenient and natural way to analyze layered systems. (?) The essential idea presented here is that an adversary must pay the price of corrupting a “deep” (harder to corrupt) component if they wish to avoid performing a “recent” corruption. In order to ensure this arrangement, the measurements in a layered attestation system must take place in a certain way. Namely, those components deeper in the system should measure first and should measure the shallower components. This measurement strategy is called “bottom-up.” Rowe’s work is a primary motivator for VERIAL because it justifies the high utility of a system of layered attestation.

“SWATT externally attests the code, static data, and configuration settings of an embedded device.” (?) SWATT is an attestation technique that uses two machines: an embedded device and a verifier. “SWATT uses a challenge-response protocol between the verifier and the embedded device.” Both the

verifier and the embedded device undergo the same pseudo-random procedure, which is seeded by the verifier's choice of pseudo-random number. The trustworthiness of this approach hinges on the fact that the procedure has tight timings, such that any deviation made necessary by an intrusion causes a noticeable increase in the computation time. SWATT could be beaten by an adversary with knowledge of the target's hardware. If the adversary, for example, poses as the target but has a faster processor, the execution time argument falls apart.

SWATT is applicable in situations with exclusive direct connections between a pair of devices, but it faces general attestation problems. SWATT requires the verifier to have total knowledge of the hardware of the target. This not only prevents constrained disclosure, but also makes the solution infeasible over the internet. SWATT depends on measurements of execution time for its trust argument, and it can be thwarted by a target with different hardware than the verifier expects.

Haldar et al (?) uses a trusted virtual machine to execute platform-independent code. This type of virtual machine differs from the kind used by a virtual machine monitor (VMM) that virtualizes hardware architecture. The measurements taken here are called "semantic" because they deal with finer-grained, richer properties of programs than simply a hash of their executable. They include "the run-time state of the program and properties of the input of the program."

This type of virtual machine is different from the type used in the work proposed here. The semantic richness afforded by the trusted virtual machine of this work is desirable. These measurements are of interest, and it appears this kind of measurement tool would be best inside the measurement target of our work in order to work. This is an exciting avenue for a "shallow" measurement in our layered architecture.

Kennell et al (?) describes challenges that authorities can pose to remote systems in order to establish their genuinity before being allowed to access shared resources. The challenges have a test for software genuinity and a test for microprocessor genuinity. In other words, there is a test to ensure exactly the right software is loaded and executing and a test to ensure that the machine hardware is as expected. The results of the challenges can be transmitted over an insecure network as long as a certain cryptographic protocol is followed.

The tests for software genuinity regard random samples of instructions that are to be executed. This is akin to sampling the executable segments of the Linux kernel. On the other hand, the tests for hard-

ware genuinity are widely applicable but require confidence that the test implementation is known exactly.

Sailer et al describes a system whose execution is completely measured by a Trusted Platform Module (TPM) (?). Starting with Secure Boot, every piece of the operating system is measured with SHA-1 hash before being loaded. The entire OS is measured with SHA-1 in this way. From that point, every new executable is measured as well before being executed. In practice, this led to a fingerprint of about 500 measurements for one author's system, and about 250 measurements for a typical server workload (?). In this way, to completely measure everything that executes is surprisingly feasible, especially when results are cached and used when nothing has changed. This design creates a static root of trust for measurement (SRTM). An SRTM is important because it builds confidence that the system booted correctly. However, it does not address the question of what happens after an executable is loaded into memory. It does not report on the current state of the system.

## 2.4 Measurement Techniques

Measurement can be static or contextual. Static measurement techniques depend on vulnerable data being in known locations and cannot convincingly associate that data with data truly in use by the target. On the other hand, contextual measurement inspects dynamic data structures and depends on a deep understanding of the target. Contextual inspection is a concept championed by LKIM to improve on the state of the art attestation systems via the comprehensiveness, freshness, and semantic explicitness principles of attestation (?).

LKIM (?; ?) is a software system for the baselining, measurement, and appraisal of an operating Linux kernel. VERIAL takes after LKIM in many ways, and VERIAL also has baselining, measurement, and appraisal steps. Baselining is the process by which the appropriate run-time states of a Linux system are described. During measurement, certain details about the state of the system are collected. Finally during appraisal, the measurement results are compared with the baseline results. If the measured evidence is in line with the baseline, the system is said to maintain integrity.

LKIM can be executed within the Linux kernel context, but it is best used with VM introspection. LKIM used the Xen hypervisor, but VERIAL uses seL4. The Xen hypervisor was for 11 years vulnerable to the VENOM flaw, but seL4 is formally verified to be free of bugs (?; ?).

One distinct difference between LKIM and VE-

VERIAL is that LKIM was developed for the x86 ISA but VERIAL was developed for ARM64. This creates significant differences in the data structures maintained by the Linux kernel and necessitates a totally different way of translating virtual addresses to physical offsets into memory. While x86 may still be the market-dominant ISA, ARM has been rapidly increasing its share since LKIM was first released (?). In particular, ARM dominates the mobile and embedded device market (?).

LKIM measurement results are an abstraction of the target's state into a graph of data values; this offers significant flexibility to appraisal agents. This is in contrast to the naive measurement: a complete snapshot of system memory. Instead, the high level approach of LKIM makes it easy to write new appraisal procedures over the graph of system data. VERIAL adopts this approach of measuring distinct targets and creating distinct evidence for each.

One high level goal of LKIM is to verify the integrity of Linux kernel execution paths. An execution path describes a collection of kernel executable data necessary for a certain job. For example, kernel function pointers should truly reference measured kernel executable segments.

LKIM achieves the ability to detect unknown rootkits, and VERIAL achieves the same. LKIM detects a variant of the adware-ng rootkit, and VERIAL detects a variant of the diamorphine rootkit. In both cases, this is an encouraging result that could warrant future research.

LKIM provides a scheduling service so that no single measurement overwhelms the Linux target. However, this creates concerns about data consistency and system quiescence. A non-atomic measurement over a data structure suffers from possibly failing to represent the data structure as it existed at any single point. LKIM asks questions about when the Linux target is non-active so that it may be measured, but VERIAL side-steps these oddities. VERIAL's measurements are atomic because the Linux target is paused during execution.

LKIM admits the difficulty of comprehensively measuring any OS (?). It also admits the necessity of using sometimes imprecise heuristics to characterize integrity policies for the Linux kernel (?). Both of these are also true in the experience of developing VERIAL. Both LKIM and VERIAL operate as a foundation for and ease the future development of new measurements to be added.

Petroni et al (?) suggests that monitoring the control flow integrity (CFI) of an operating kernel is an effective way to detect rootkits. The authors argue that a large class of rootkits violate CFI, and that mon-

itoring CFI in an abstract form imposes low overhead. The authors actually monitor "state-based" CFI, by which they "periodically examine the kernel's state and validate it as a whole." First they validate all static branches of the kernel code, ensuring nothing has changed. Then they traverse the heap and validate every function pointer they find.

CFI is one type of measurement that can be used to appraise the trustworthiness of an operating system. Petroni's work shares techniques with LKIM, in which function pointers are validated and static analysis of the kernel image is conducted.

The USIM toolkit (?) is "a set of integrity measurement collections tools capable of detecting advanced malware threats." User-space integrity measurement validates platform trustworthiness by ensuring the consistency of a set of invariants. In this case, "user-space" refers to everything necessary for a system to operate normally except the executing kernel. Such measurements include hashes of important files and hashes of executable memory segments mapped in certain processes, as well as memory mappings and namespaces.

The USIM toolkit is related to VERIAL because it includes a collection of measurements that can be used to appraise the trustworthiness of an operating system. However, it only applies to "user-space", so it cannot be trusted to detect rootkits. The measurements and appraisal strategies discussed herein represent opportunities for future measurements to be implemented in this effort. Combined with effective measurement and appraisal techniques for detecting rootkits, this user-space measurement toolkit could become part of a complete measurement suite.

Petroni et al (?) introduces "a novel general architecture for defining and monitoring semantic integrity constraints using a specification language-based approach." Namely it implements a defense to two attacks that change dynamic kernel data structures to leverage control. It implements these defenses via a specification written by an expert that describes "an abstract model for low-level data structures and the relationships between them."

Tripwire is a measurement tool suite that targets UNIX file systems. (?) It creates a database of unique identifiers for each file monitored. Then it periodically verifies that either a checklist or a signature is valid for each file monitored. In this way, it is possible to both determine if a file has been altered or if files have been added or deleted. However, as a user-level application, Tripwire is vulnerable to kernel subversion.

File system integrity is a significant part of integrity measurement. Tripwire represents an impor-

tant integrity check for user-space. Although it cannot detect corruption in the kernel, it can detect corruption in long term storage, wherein backdoors and stepping stones can be hidden. VERIAL provides a coarse-grained measurement of file systems; it validates the presence of known files and their size (in pages), and it validates the absence of unknown files.

Nentwich et al (?) follows up on a tool for checking consistency of documents with a tool that allows for repair actions to be taken on documents deemed inconsistent. The core of the paper was a semantics for deriving a complete and correct set of repair actions. The tools of this paper were implemented for UML objects. Consistency checking of documents is much like integrity measurement of objects in a file system. On the other hand, repair actions are not approached in VERIAL because they are not a feature of attestation.

Pioneer (?) is founded upon a dynamic root of trust for measurement (DRTM). The DRTM is “instantiated through a self-checking function that computes a check-sum over its own instructions.” The self-checking function is designed such that any attempts to manipulate its execution cause a significant increase in execution time. In this way, a sufficiently fast correct check-sum is an argument for the correctness of the self-checking function that establishes the “dynamic root of trust.” Pioneer then extends this trust to the measurement of an apparently isolated program binary.

Pioneer is interesting because it does not depend on external hardware or even on a virtual machine manager. Instead, it is a completely software-based single-kernel attestation architecture. Like SWATT, Pioneer depends on a race: in this case to compute a check-sum. Furthermore, Pioneer requires the program it measures to be self-contained, having no external dependencies.

Like Pioneer, VERIAL constructs a DRTM, but the usage and purpose of the DRTM differs in VERIAL. Pioneer builds a DRTM to increase its own trustworthiness. On the other hand, VERIAL builds a DRTM to increase the trustworthiness of layered measurements. It is the focus on layered attestation that distinguishes VERIAL’s use of the DRTM.

When combined with a TPM, the Linux Integrity Measurement Architecture (IMA) can provide evidence of a Linux system’s run-time integrity. IMA is integrated in the Linux kernel. It measures every file into a hash digest before it is opened or executed, and it collects these digests in kernel space. If such a measurement results in a previously unknown digest, operation on that file is prohibited. This protects against depending on a file that was surreptitiously changed.

The Linux IMA has utility for validating file integrity, but its presence within the Linux kernel leaves it vulnerable to rootkits. Like Pioneer, Tripwire, and the USIM toolkit, the Linux IMA is vulnerable to kernel-level attacks. However, the Linux IMA is a natural target for layered attestation in VERIAL. If VERIAL can provide evidence that the Linux kernel and the IMA have integrity, then the IMA can subsequently provide evidence that the file system has integrity.

Ko et al introduces a monitoring technique by which audit trails are inspected to determine whether an attack has occurred (?). In this case, the programs are “setuid root programs in Unix,” such as `rdist`, `sendmail` and `fingerd`. The core idea of this paper is that such programs are given special privileges but are expected to use them in only narrowly-defined and benign ways. That is, the usage of privilege is logged in audit trails, and misuse of privilege can be detected in these trails. This approach benefits from being reasonably expected to detect unknown attacks on the programs whose proper behavior has been specified with sufficient precision.

Demsky et al describes a system by which a developer can write consistency constraints that are automatically checked during execution and repaired if necessary (?). This approach is considered useful in systems that cannot afford total failure, yet can sustain the imperfect repair that follows one. That is, a repaired system might not be operating as a perfectly executing system would be, but the repair actions might have prevented a total failure, and a repaired system might be able to get back to perfect condition.

Some measurement techniques require support from their target. For example, LKIM deploys certain hooks in the Linux kernel in order to LKIM to react to kernel module load events. Some control flow integrity strategies like PRIMA require their target to be “PRIMA-aware.” These dependencies on the target reduce the applicability of these strategies. In contrast, VERIAL operates on an unmodified Linux kernel.

## 2.5 Rootkits and other Malware

A rootkit is a self-obscuring mechanism. Rootkits can be malware designed to elevate access and evade detection. But rootkits can also have innocent purposes; rootkits are sometimes used for intrusion detection systems and for anti-cheat mechanisms in online gaming. Different rootkits can be responsible for hiding processes, files, or even network traffic. Some rootkits exist in persistent storage so that they main-

tain control across reboot. Other rootkits exist only in volatile memory, limiting their lifespan and detectability. In all cases, rootkits are designed to avoid detection.

There have been user-level rootkits that do not require kernel privilege. To replace a binary with a malicious version can be a rootkit. For example, to replace the “ps” tool with a subverted copy can hide processes. A rootkit can also hijack a process, allowing the rootkit to execute code with improper authorization. However, user-level rootkits can always be detected by kernel tools.

A kernel-level rootkit is a subversion of kernel tools. A kernel tool can be hooked so that it cannot discover its own corruption. For example, the Adoreng rootkit and the Phalanx2 rootkit modify the system call table to hook critical kernel functions such as “open.” Some techniques involve changes to kernel executable code, but others such as DKOM permit data-only attacks.

Finally, a hypervisor-level rootkit provides privilege levels superior to the victim kernel. The Blue Pill rootkit used hardware virtualization features to lift the entire target kernel in a virtual machine hosted by a thin hypervisor. However, such a hypervisor must exist in memory and can be detected by timing attacks due to the extra expense of context switching in virtualization. Other examples are the Vitriol and SubVirt rootkits.

VERIAL has been shown to detect a variant of the diamorphine rootkit. The diamorphine rootkit is a loadable Linux kernel module (LKM) that modifies the kernel executable and data segments and hides itself from user-level tools. To defend against hypervisor-level rootkits, VERIAL depends on the verified nature of seL4 to prevent unauthorized access to data.

CacheKit is a rootkit that avoids detection by hiding itself entirely within the CPU cache (?). The CacheKit rootkit is not detectable by either software or hardware-based introspection techniques. Fortunately, with sufficient knowledge of the inner workings of the rootkit, a detection can still be made in the way of anomaly discovery or repair. Namely, the rootkit has a hook in the “interrupt vector table,” although it could just as well have a hook in the system call table. Discovering anomalous entries or simply periodically flushing the cache or validating these tables can suffice to deal with such a rootkit.

Virtual machines uniquely suffer from many styles of exploits (?). These attacks include communicating between VMs, escaping a VM, denying service to the host or to another VM, and modifying either the VM or the hypervisor. Fortunately, most of these at-

tacks are mitigated by the proper implementation of access rights in the form of seL4 capabilities. Notably, each VMM in seL4 manages only one VM; no two VMs have communication privileges by default.

Of particular interest is the on-system denial-of-service attack. This is an attack by which, without having communication privileges, a guest VM can disrupt service to other tasks running on the host. Fortunately, with seL4, it is easy and in fact required to limit the resources available to the guest VM. That is to say, only a static amount of resources will ever be allocated to the VM; the memory is not dynamically allocated (?). In this way, we can ensure there will always be resources available to other seL4 tasks that need them.

### 3 CONCLUSIONS

I design and implement VERIAL, an seL4-based operating system with an embedded measurement suite capable of detecting integrity violations in the Linux kernel and its userspace data. VERIAL is implemented for Linux LTS versions 4.9 through 6.1 on Arm64. I show the utility of VERIAL by implementing and subsequently detecting a rootkit (?). I explain how VERIAL detects the addition, removal, or change of kernel read-only data, kernel modules, user processes, and user files. In doing so, I advance the state-of-the-art for remote attestation using formally verified components.

This architecture behaves similarly to other software solutions for VM attestation, but its foundation on seL4 provides distinct benefits. First of all, seL4 is the “world’s only hypervisor with a sound worst-case execution-time analysis” (?). Second, seL4 is formally verified to provide data integrity and confidentiality, although these results come with assumptions laid out in Section ???. These properties are leveraged to create separation for VERIAL’s measurement routines, preventing the Linux kernel from interfering with them. Finally, seL4 is proven, among other results, to be free of bugs and to not be vulnerable to buffer overflows. Among kernels, seL4 is fast and side-steps entire classes of vulnerabilities.

Another key difference between VERIAL and prior work, such as LKIM or HYDRA, is its use of the Copland language to express attestation procedures (?; ?; ?). Two Copland attestation managers are used in VERIAL to provide verified attestation tools in both the seL4 and Linux domains. The Copland AM supports layered attestation, and VERIAL takes advantage of that (?).

VERIAL includes a concise and novel measure-



ment suite that can detect breaches of integrity including the presence of a diamorphine-style rootkit. VERIAL performs contextual measurement of its Linux VM at run-time and comprehensively verifies the integrity of the kernel code and read-only data. VERIAL is also equipped with several measurements over mutable kernel data. For example, one of VERIAL's most rapid measurements measures mutable system call table data that can reveal the presence of a rootkit.

A remote adversary is restricted by VERIAL's layered attestation in the ways laid out by Rowe (?). Such an adversary is successful only when conducting a malware attack that is "deep or recent." A deep attack is targeted against VERIAL's architecture, and a recent attack occurs in the window between measurement and action. The failure case for VERIAL is this: the target is measured, then the target is compromised, then the target takes trusted action. This window of opportunity can be reduced by various means such as repeated measurements or event-induced measurements like those described in LKIM (?).

VERIAL has comparable performance costs to Copilot, with about a 1% penalty if run every 30 seconds (?). Like Copilot, VERIAL can be amortized to reduce this penalty, as discussed in Section ???. VERIAL comes with both static and dynamic penalties. The static penalties are unchanging architectural costs for using a hypervisor and for using the Copland AM. The dynamic penalties are a result of the measurement suite chosen for a given attestation. For the system measured, the time cost of attestation was anywhere from 0.17s to 1.16s.

Neither VERIAL nor LKIM have claimed to measure the Linux kernel comprehensively (?). VERIAL targets Linux due to its popularity and widespread deployment. However, the Linux kernel is not completely specified and neither does it exist simply; it is complex in both source and run-time memory. Measurement of a Linux kernel is a complex undertaking, but government, industry, and consumers use billions of Linux devices every day, so the large time investment is worthwhile.

Future work could be done to improve VERIAL. More measurements can be written to cover more parts of the Linux kernel. Some unverified parts of VERIAL might benefit from being re-written in Rust for the sake of memory safety. VERIAL's architecture could be re-written using Microkit instead of the CAMkES architectural description language (?; ?). The introspection library could be expanded to support instruction set architectures other than ARMv8-A AArch64. VERIAL's existing measurements could have amortization routines embedded into them.

## REFERENCES

- Moore, R. and Lopes, J. (1999). Paper templates. In *TEMPLATE'06, 1st International Conference on Template Production*. SCITEPRESS.
- Smith, J. (1998). *The Book*. The publishing company, London, 2nd edition.

## APPENDIX

If any, the appendix should appear directly after the references without numbering, and not on a new page. To do so please use the following command: `\section*{APPENDIX}`