The trust problem in modern network infrastructures

Ludovic Jacquin (*), Antonio Lioy (+), Diego R. Lopez (%), Adrian L. Shaw (*), and Tao Su (+)

(*) Hewlett-Packard Laboratories (Bristol, UK)
(+) Politecnico di Torino (Torino, Italy)
(%) Teléfonica I+D (Madrid, Spain)

Abstract. SDN and NFV are modern techniques to implement networking infrastructures and can be used also to implement other advanced functionalities, such as the protection architecture designed by the SECURED project. This paper discusses a couple of techniques – trustworthy network infrastructure monitoring and remote attestation of virtual machines – useful towards a trusted and secure usage of SDN and NFV.

Keywords: SDN, NFV, remote attestation, trust, security.

1 Introduction

Network infrastructure is quickly evolving from a hardware-based switch-only layer to a full-fledged computational system able to perform several tasks, switching packets being just one, although a very important one. This evolution is permitted by the advent of two new architectures, namely SDN and NFV.

SDN (Software-Defined Networking) is one particular approach to provide virtualised traffic routing and unified network flow management across hardware and software-based networking components. The principal design of SDN is to virtualise the existing control and data planes by moving the control part away from all network elements to a centralised node in the network, known as the *SDN controller*.

NFV (Network Functions Virtualisation) proposes to virtualise several classes of network node functions into generic building-blocks (to be run as virtual machines on commodity hardware) to be connected for creating various network services. NFV typically exploits SDN to create custom overlay networks connecting the various network functions and in turn SDN can use NFV to host its controllers and applications.

SDN and NFV can be used also for non network-related functions: an example is provided by the SECURED project [1] which uses SDN to create a custom network path for each user to interconnect its network-hosted security controls, that may be executed in a NFV infrastructure. Purpose of this project is to create a user-oriented security environment, protecting the user's traffic independent

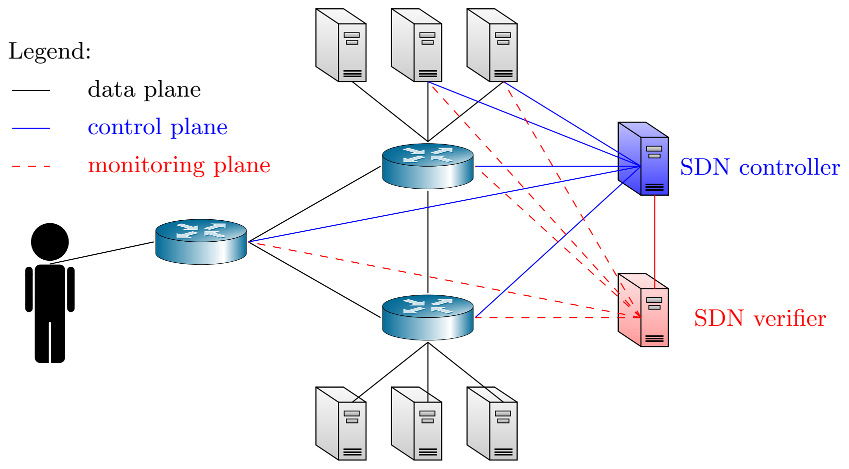


Fig. 1. A typical SDN topology, enhanced with a SDN verifier.

of the specific user device and network access point. As SDN and NFV are critical elements for SECURED, it is investigating techniques to improve their trustworthiness and security and a first analysis and proposal is reported here.

2 Trustworthy monitoring architecture for SDN

The usage of SDN introduces new network abstractions and high-level primitives but this creates a trust gap for administrators as they cannot easily assess the correctness of enforced device configurations. We present here a monitoring architecture for SDN to bridge this semantic gap, with the additional goal to be both trustworthy and automated, such that administrators only need to act upon detection of faulty behaviour. The proposed monitoring architecture introduces an out-of-band *SDN verifier*, from the control plane perspective, to automatically and continuously attest the enforced SDN rules by the network elements.

2.1 Introducing the SDN verifier

In a typical SDN topology, network elements are both hardware- and software-based, with a hierarchy of controllers to program them all. As we want to address the security and trust concerns of this new layer, we define the following attacker model:

1. an attacker can modify the software stack of a network element;
2. hardware attacks cannot be performed, in particular physical links are deemed secure;

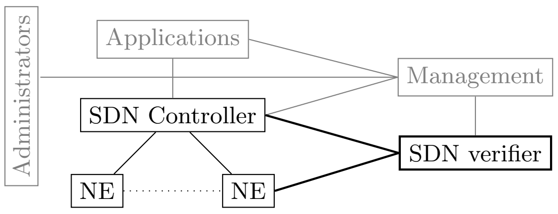


Fig. 2. Logical overview of the proposed monitoring architecture

3. the SDN controller is secure and trusted – whilst it is an important and central element of SDN, the security of the controller is an orthogonal issue to what is addressed here.
4. the SDN verifier is secure and trusted.

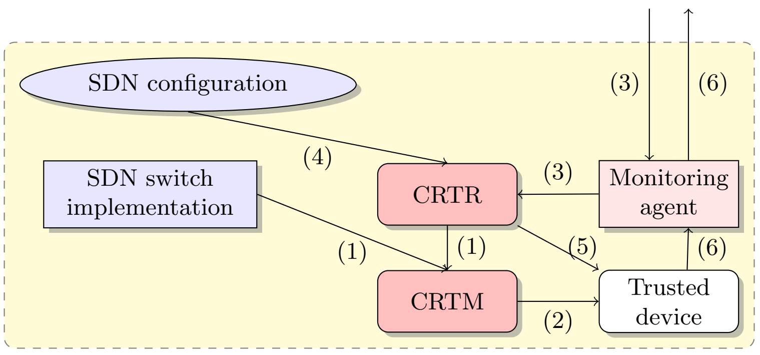
The introduction of the SDN verifier requires a new monitoring plane which will be used to exchange attestation data from the network elements. A special part of this monitoring plane is the connection between the verifier and the controller(s): it is used by the verifier to retrieve the expected SDN configuration of the network elements. With regards to the attacker model considered, we keep the same assumption as the controller.

In the control hierarchy, the verifier remains in the same network as the SDN controller and the network elements, as depicted in Fig. 2. We envision that it would report to the management components, which in turn inform the administrator and potentially SDN applications. The application has the logic and knowledge of the network topology, to act on faulty behaviour automatically if needed.

From a functional perspective, the SDN verifier focuses on re-establishing trust in the network elements by attesting their software stack and the SDN configuration they are currently enforcing at the dataplane. Since the main tasks of the controller and network elements are to route the traffic through the dataplane and do performance critical tasks, we choose to alleviate this pressure by offloading the computational complexity to the SDN Verifier. Whilst the verifier–controller interface is quite trivial as most implementation already support that through a web interface, the proposed monitoring architecture needs an embedded monitor inside each network element.

2.2 Network element monitoring

The verifier relies on the attestation agent installed in each network element to locally retrieve and package the monitoring data back to the verifier. Since a software-only solution would be prone to a wide range of software attacks, our design uses a trusted device inside the network element. This trusted device is generally immutable and is used as a basis for trust, which is leveraged by the verifier to attest the network element and its behaviour. The trusted device must provide a hardware-based identity and enable the creation of a *Core Root of Trust*



1: Attestation measurements
2: Secure storage of measurements
3: SDN verifier challenge
4: OpenFlow rules inspection
5: Dynamic configuration measurements
6: SDN verifier response

Fig. 3. Network element component architecture target

for Reporting (CRTR), a component which measures the dynamic configurations related to network flow rules that are currently enforced.

Hardware-based identity. The network element identity is primarily used to derive a pair of public/private keys used to provide integrity and authentication of the monitoring data. For instance, the IEEE 802.1AR already provides examples of facilitating secure device identity provisioning. The identity is provisioned at installation time inside the trusted device with the corresponding derived private key. This identity allows the SDN verifier to check the authenticity of the attestation proof coming from the trusted device and to check the integrity of the measurement reports. The methods of identity distribution and management are not covered as part of this paper.

Measured Boot. One way of assessing the software state of a computer is to measure every trusted piece of software before it is loaded during device start-up. This is known as a *measured boot*. This is achieved by augmenting the boot process of the machine to cryptographically measure executable software at load time and securely report the result to a trusted device prior to transferring execution to the loaded software. The subsequent piece of measured software can in turn load and measure more software. This process is repeated for each piece of software loaded on the platform, creating what is commonly known as a *chain of trust*. Since the trusted device keeps logs of all software executed on the platform, it is able to report all the measurements of the configurations since early in the boot process. Recursively, the measured boot relies on a small combination of hardware and software that must be implicitly trusted to measure software running on the platform. This minimum combination constitutes the *Core Root of Trust for Measurement (CRTM)*. It is fundamental that the CRTM implements a secure storage capability that cannot be tampered with by other software on the platform, such that any potentially malicious code cannot erase or overwrite logs that have already been reported and stored in the trusted device. As an example, on a PC platform the CRTM has typically been composed of both the BIOS and a Trusted Platform Module (TPM), both of which must be trusted in order to provide secure and assurable storage of software measurements and secrets.

Remote verification stage. Once all measurements have been reported by each component to the trusted device, a verifier is able to then remotely query the trusted device. The cryptographically signed response from the trusted device should include the measurement log provided by the CRTM and all subsequent measured pieces of software. The verifier can then compare the received logs with well-known measurements corresponding to a measurement database of trusted software components (and states). It is important to note that with a measured boot approach, the verifier must attest each piece of software to verify that nothing previously loaded was unknown or untrusted software. Thus, in order to verify if the operating system is in a trusted state, all the logs of previously executed software must be asserted as known and trusted software.

Attestation agent. Its main goal is to relay the incoming challenges and outgoing responses between the verifier and the CRTR. It is not a trusted component in the monitoring architecture since the verifier's challenge is used by the trusted device when creating the attestation proof. Any tampering of the challenge or the response would be detected by the SDN verifier.

Core Root of Trust for Reporting (CRTR). The role of the CRTR is to inspect the forwarding table state of the SDN switch implementation in a network element and report it to the SDN verifier. The CRTR is eventually loaded as part of the platform's trusted computing base. Ideally, the CRTR needs to be one–way isolated from the network element implementation. For instance, the CRTR must be able to inspect the VLAN routing tables and system configurations, but the rest of the switch system must not be able to interfere or influence the CRTR functionality, The isolation of the CRTR is either enforced using a hardware mechanism, or attested through a CRTM. The full combination of the CRTR, the agent and the trusted device are illustrated in Fig. 3.

With the addition of the SDN Verifier and the embedded reporting mechanism inside the network elements, a software-defined network can be more strongly attested for (A) correctness of firmware and software at the data plane, and (B) correct enforcement of dynamic configurations, such as VLAN forwarding tables.

2.3 Prototype implementation

Let us now focus on the trade-offs done during the implementation of a hardware switch prototype and show the equivalent implementation for a virtual SDN switch, such as OvS [2].

For the prototype, we made two major technological choices: (i) we rely on a TPM as the trusted device (and we use the well-known Trusted Computing Group methods for the CRTM), and (ii) the monitoring agent uses SNMP to communicate with the SDN verifier. Both technologies present the main advantage to be widely deployed world wide. The downside of the TPM is its performance: in our hardware switch prototype, the remote attestation exchange (over SNMP) takes around one second, where most of the latency comes from the digital signature during the TPM Quote operation (which can take as long as 600 ms).

In the meantime, we are evaluating three approaches for a software SDN switch attestation. TPM remains our choice for the trusted device, especially with the existing software environment around it, namely TrustedGRUB [3] and IMA [4]. The main point we are investigating is the CRTR to introspect the "vswitchd" context. We will evaluate three different approaches: (i) full kernel-based integrity monitor, that routinely inspect the memory of the OvS process; (ii) split integrity monitor, that relies on a small kernel module for the memory introspection and moves the processing logic in a user–space agent; (iii) full user–space monitor, if the monitor can access directly the OvS memory space.

3 NFV and virtual machine attestation

Any NFV service deployment requires the onboarding, activation and start-up of a set of virtualised elements that will be run on a uniform infrastructure supporting the virtualised execution environment (the *NFV Infrastructure, NFVI*). Furthermore, these elements have to be connected to other elements according to a given network topology, that will be dynamically created by requesting it to the *Virtual Infrastructure Manager (VIM)*, which typically would create it by means of SDN. In this environment it is obvious the need for applying procedures to verify the integrity of the system (the whole NFV service deployment) by the appropriate attestation of the NFV architectural elements, including software and firmware images and associated supporting security sub-systems that will run to instantiate individual *VNFs (Virtual Network Functions)* and their composition into a NFV service. Since these procedures will have to be executed by the *Management and Orchestration (MANO)* stack in charge of the service deployment they have to support remote attestation mechanisms and, more specifically, they have to apply cryptographic techniques to verify system integrity.

NFV remote attestation requires identifying the root(s) of trust, establishing a chain of trust for the NFVI, the individual VNFs, and the MANO sub-systems, and verification of the trust chain, so the MANO stack components can verifiably establish a sufficient level of assurance in the different software elements constituting the VNFs and the service(s) that use them. While exist standards and best practices for attestation in physical environments (TPM, TCG, . . .), a detailed assessment of their applicability is needed due to the extensive use of virtualisation techniques, the scale of VNF composition, and the requirement to perform network topology attestation. The NFV Security Problem Statement [5] describes secure boot and secure crash among key issues for guaranteeing a secure NFV operation, while the NFV Security and Trust Guidance [6] directly mentions attestation mechanisms. Several NFV use cases [7] and reported experiments [8] describe situations in which VNFs are dynamically on-boarded, updated or modified, and a proper verification of their correct provenance is an essential step in these procedures.

The attestation steps may be specific to the level of assurance to be established, which, in turn, depends on the nature of the particular network function, the service it supports, and the different parties involved in its instantiation. Furthermore, different local and remote procedures may apply depending on whether the elements in the supporting infrastructure are trusted. To establish a set of common NFV attestation technologies it will be necessary to address the following aspects:

– define the required levels of assurance;
– identify the assumed capabilities in the NFVI (e.g. TPM, secure boot, . . .);
– assign the operational procedures to be applied at each layer of the MANO stack;
– specify how attestation requirements will be expressed in the different NFV descriptors, for the NFVI, individual VNFs, and services;
– specify the information model to exchange attestation requests and data at the reference points in the NFV architecture framework.

Among all these challenges, we currently focus on attestation of virtual machines as it is a fundamental problem for NFV trustworthiness.

3.1 Virtual machine attestation

Attestation of physical computing platforms is possible in various ways. A very common one is to to exploit the Trusted Platform Module (TPM), a special chip available on most hardware platforms. It is used to provide secure storage, integrity measurement, and reporting. The TPM offers secure storage in the form of *Platform Configuration Registers* (PCRs) that can only be accessed with specific commands. Integrity measurement consists of computing the digest of target files and accumlating the values into the PCRs with a specific command (*extend*).

The action of reporting the integrity of the platform is called *Attestation* and is mostly useful in its *Remote Attestation* form, which is requested by a different network entity that wants evidence about the current software status of the attested platform. The TPM then makes a digital signature over the values of a subset of PCRs to prove to the remote entity the integrity and authenticity of the platform configuration. In this way, the evidence provided to other party is reliable and authentic. It is bound with the hardware TPM, which cannot be forged by others.

Attestation comprises three phases: *measurement*, *attestation* and *verification*. Previous studies about attesting VMs mainly focus on the first two steps, how to measure the system [4] [9], and how to properly attest the results [10] [11], but few address the verification problem. Actually, most works do not provide information of how to verify the measurement properly. Binary-based attestation is the most popular solution, which extends the measurements into PCRs residing inside the TPM. Subsequently, verifier compares these PCR binary values with golden ones. This approach can provide high security assurance, but need complex management because the PCR values are order sensitive. Unlike the booting process (where the components are loaded one after another in a specific order), during normal usage the execution of software happens in random order, so PCR values verification could easily fail even though integrity is not compromised.

Attesting VMs is a difficult task as they do not have direct access to the hardware and hence to the TPM, which is the critical component for attestation. Even if direct access to the TPM could be provided, the number of PCRs would be insufficient for the number of VMs normally activated on a single hardware platform.

A first approach to solve this problem is to emulate the whole functionality of a physical TPM by using a custom software module. In this way, this module can be used to attest multiple VMs with just one hardware TPM. This is the *vTPM* [10] approach (Fig. 4) where each VM has a *client side TPM driver*, which VMs send their TPM commands to. A *server side TPM driver* is running in a special VM on top of the hypervisor; this server-side driver collects the data from the client-side driver, and sends them to the vTPM manager. The vTPM manager is in charge of creating vTPM instances and multiplexing requests from VMs to their associated vTPM instances. Since the vTPM instance number is prepended on the server side, a VM cannot forge packets and try to get access to another vTPM instance not associated with itself. This solution has been implemented in XEN [12].

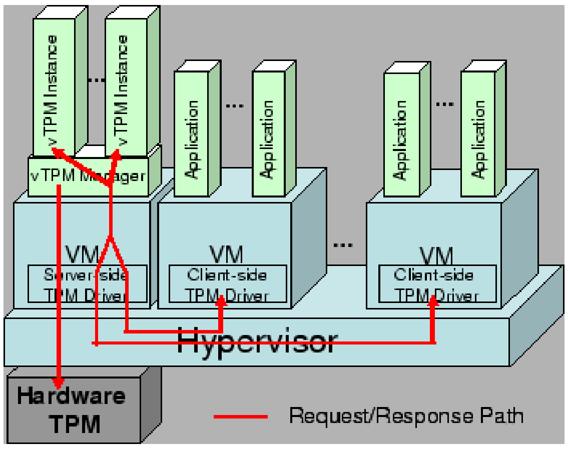


Fig. 4. The vTPM architecture [10].

Another work [11] addresses the scalability issue by extending the vTPM model to reduce the complexity of software attestation. Since the traditional periodic polling model does not scale well (each VM adds effort to the attestation cost), the authors propose an event-based monitoring and pushing model. The benefit of the pushing model is that it will eliminate the problem of Time-of-Measure to Time-of-Report attacks and TPM reset attack (i.e. fast rebooting the system after the malicious script execution, to reset TPM PCR values).

The architecture proposed by the authors is shown in Fig. 5. The client TPM driver normally executes the TPM extend commands through the vTPM manager into its own child state. Now the vTPM manager can repeat the same extend operation into the parent state of the child state (which works like the PCRs in the hardware TPM). One parent state may create multiple child states, so that it can monitor multiple guest virtual machines at the same time. Every time the parent state is modified, the vTPM manager notifies the users that subscribe to it, thus achieving event-based attestation.

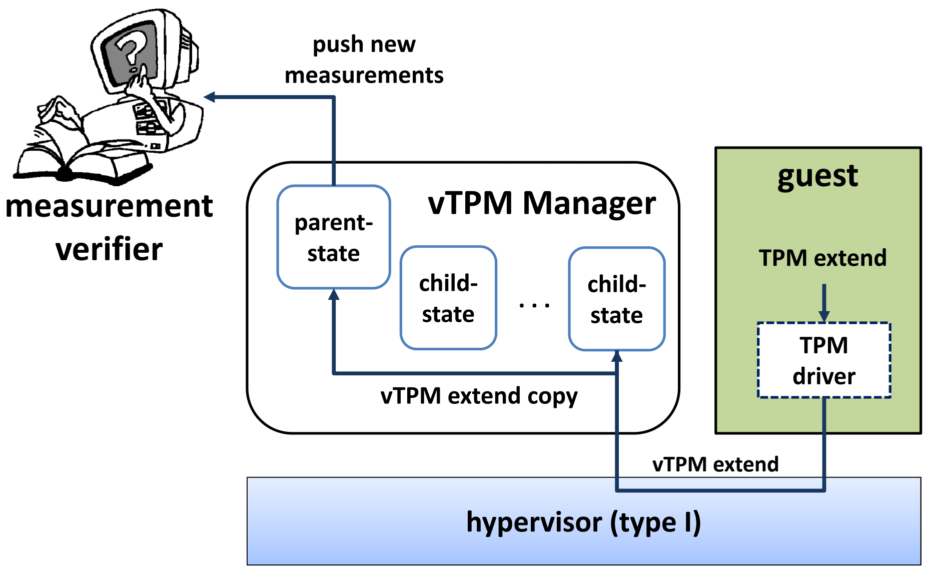


Fig. 5. Solution architecture in [11].

The obvious benefit for this solution is its scalability and the possibility to eliminate the ToM-ToR and TPM reset attacks. It can easily support thousands of VMs.

In general event-based monitoring is more convenient and feedback time can be much faster. In a virtualised environment, the hypervisor can be modified to support event-based monitoring. Following this idea, in [13] the authors propose to verify VM integrity by an *Integrity Verification Proxy (IVP)* embedded in the hypervisor. They chose QEMU/KVM as the hypervisor to be modified. The VMs running on top of KVM are in *Debug Mode*, so a debugging tool (e.g. `gdb`) can be used to set watchpoints (e.g. locations in memory) that are triggered by integrity-relevant operations such as Integrity Measurement Architecture (IMA) operations. Once the watchpoint is triggered, the VM will be paused and no outgoing/incoming traffic is possible. Until the module finishes to assess whether the new event violates an integrity criteria, the VM is not permitted to resume execution. This solution suffers a penalty due to executing the VMs in debug mode. Applying the same technique in normal mode is currently an open challenge.

3.2 IMA-based attestation

To support remote attestation in virtualised environment, the supporting component (SC) can be put in three different places (Fig. 6).

The first place is in the hosting system with a type II hypervisor.

The second option is to put it in a special VM (as done by vTPM). In this case, since the hosting system is not monitored, only a type I hypervisor can be used but neither the hypervisor nor the OS running in the VMs require modification.

The third option is to embed the supporting component within the hypervisor (either type I or type II) so that the supporting component can provide event-based attestation, but the hypervisor needs modification.

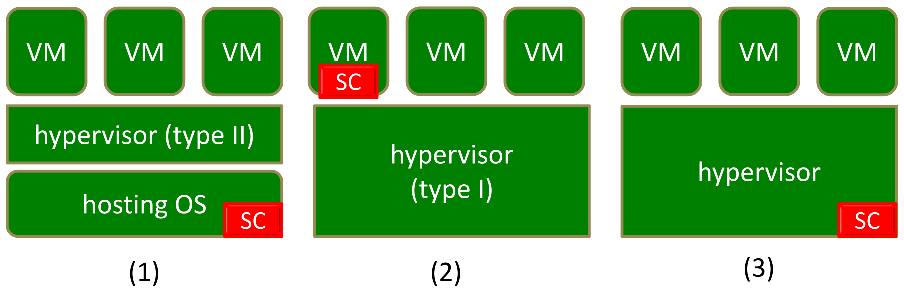


Fig. 6. Possible placement of the attestation supporting component.

We are currently exploring the first option: a remote attestation component running inside the hosting system underlying a type II hypervisor. This relies on the IMA measurements generated by the Linux kernel. It also requires a database of executable digests to show what software is running in the VMs and whether it is trusted or not. Since in SECURED the VM images are customised and their initial state is known, the only files that need to be checked are the configurations and the executable files loaded inside the VM.

The idea is to place one *attestation proxy* in the hosting system and use it to retrieve the IMA measurements from the VMs running inside it (Fig. 7). Since

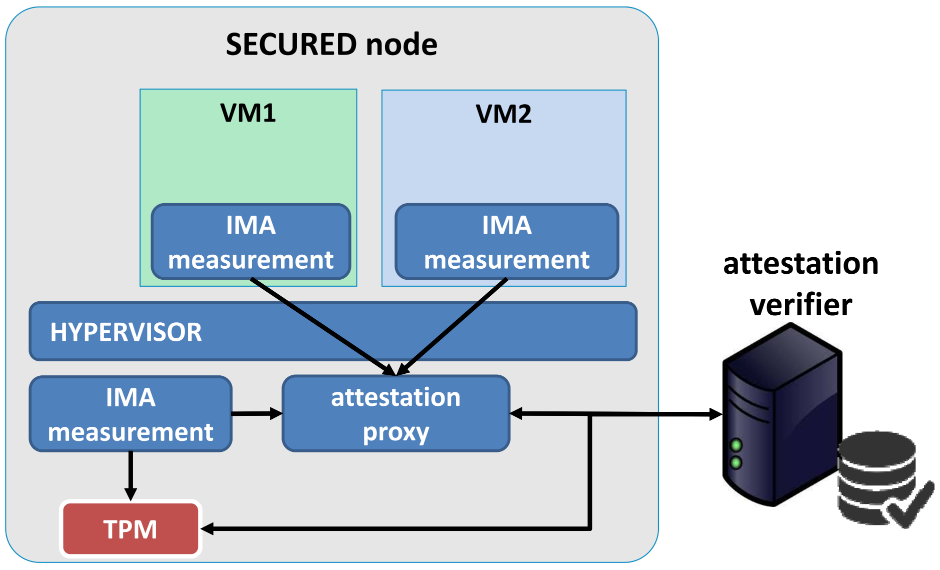


Fig. 7. IMA-based attestation architecture.

IMA measures all the executables invoked in VM, based on its policy, the IMA measurement is the key to ensure the integrity status of the VM.

The physical TPM is used to attest the integrity of the hosting OS, including the attestation proxy and the hypervisor.

Since the proxy is doing what it is expected to do (thanks to the physical TPM), it is guaranteed that the verifier will retrieve the IMA measurements and compare them with its database. In this way, we can set a short time period (like a few seconds) to ask the proxy to retrieve the IMA measurements from each VM, and compute the digests of each of them. If the digest has changed, then compare the new measurement with the database in the verifier.

A big challenge here is how to associate each IMA measurement with its VM. Luckily, this can be solved with the VM-id assigned by the hypervisor when it starts the VM, and this id cannot be modified from the VM internally.

In order to improve the scalability, it is a good option to use the push model. So if the verifier detects that one VM integrity status is compromised (some unknown scripts were executed or some unknown configurations loaded), it can inform the attestation proxy which in turn can notify the user or simply shut down the VM.

4 Conclusions

SDN and NFV are useful technologies increasingly used to implement modern networking infrastructures. However, as they are heavily relying on various software components distributed by several actors, we need proper techniques to guarantee their trustworthiness and integrity. In this paper we have presented two possible approaches based on remote attestation to measure the integrity state of SDN switches and virtual machines executing critical network functions. However much more work is needed to solve practical issues (e.g. performance, management of cryptographic identity) as well as theoretical ones (e.g. fast and secure migration of VMs while maintaining their attestation state).

Acknowledgement

The research described in this paper is part of the SECURED project, co-funded by the European Commission (FP7 grant agreement no. 611458).

References

1. Dalton, C., Lioy, A., Lopez, D., Risso, F., Sassu, R.: Exploiting the Network for Securing Personal Devices. In: Cyber Security & Privacy Forum 2014, Athens (Greece), May 21-22, 2014, pp. 16-27
2. Open vSwitch, https://github.com/openvswitch/ovs
3. TrustedGRUB, http://sourceforge.net/projects/trustedgrub/
4. Sailer, R., Zhang, X., Jaeger, T., van Doorn, L.: Design and Implementation of a TCG-based Integrity Measurement Architecture. In: 13th USENIX Security Symposium, San Diego (CA, USA), August 9-13, 2004, pp. 223-238
5. ETSI NFV ISG: NFV Security / Problem Statement. Report ETSI GS NFV-SEC 001 (V1.1.1), October 2014. http://www.etsi.org/deliver/etsi_gs/NFV-SEC/001_099/001/01.01.01_60/gs_NFV-SEC001v010101p.pdf
6. ETSI NFV ISG: NFV Security / Security and Trust Guidance. Report ETSI GS NFV-SEC 003 (V1.1.1), December 2014. http://www.etsi.org/deliver/etsi_gs/NFV-SEC/001_099/003/01.01.01_60/gs_NFV-SEC003v010101p.pdf
7. ETSI NFV ISG: NFV / Use Cases. Report ETSI GS NFV 001 (V1.1.1), October 2013, http://www.etsi.org/deliver/etsi_gs/NFV/001_099/001/01.01.01_60/gs_NFV001v010101p.pdf
8. ETSI NFV ISG: NFV Proofs of Concept. http://www.etsi.org/technologies-clusters/technologies/nfv/nfv-poc
9. Jaeger, T., and Sailer, R., Shankar, U.: PRIMA: Policy-Reduced Integrity Measurement Architecture. In: 11th ACM Symposium on Access Control Models and Technologies, Lake Tahoe (CA, USA), June 7-9, 2006, pp. 19-28
10. Berger, S., Sailer, R., Goldman, K.A.: vTPM: Virtualizing the Trusted Platform Module. In: 15th USENIX Security Symposium, Vancouver (B.C., Canada), July 31–August 8, 2006, pp. 305-320
11. Goldman, K., Sailer, R., Pendarakis, D., Srinivasan, D.: Scalable Integrity Monitoring in Virtualized Environments. In: 5th ACM Workshop on Scalable Trusted Computing, Chicago (IL, USA), October 4-8, 2010, pp. 73-78
12. Barham, P., Dragovic, B., Fraser, K., Hand, S., Harris, T., Ho, A., Neugebauer, R., Pratt, I., Warfield, A.: Xen and the Art of Virtualization. In: 19th ACM Symposium on Operating Systems Principles, Bolton Landing (NY, USA), October 19-22, 2003, pp. 164-177
13. Schiffman, J.,Vijayakumar, H., Jaeger, T.: Verifying System Integrity by Proxy. In: 5th Int. Conf. on Trust and Trustworthy Computing, Vienna (Austria), June 13-15, 2012, pp. 179-200