Stay Thrifty, Stay Secure: A VPN-Based Assurance Framework for Hybrid Systems

Marco Anisetti[1][a], Claudio A. Ardagna[1][b], Nicola Bena[1] and Ernesto Damiani[2][c]

[1]*Dipartimento di Informatica, Università degli Studi di Milano, Milan, Italy*

[2]*Artificial Intelligence and Intelligent Systems Institute (AIISI), Khalifa University, Abu Dhabi, UAE*

{*marco.anisetti, claudio.ardagna*}*@unimi.it, nicola.bena@studenti.unimi.it, ernesto.damiani@kustar.ac.ae*

Keywords: Assurance, Hybrid System, Security, Virtual Private Network

Abstract: Security assurance provides a wealth of techniques to demonstrate that a target system holds some non-functional properties and behaves as expected. These techniques have been recently applied to the cloud ecosystem, while encountering some critical issues that reduced their benefit when hybrid systems, mixing public and private infrastructures, are considered. In this paper, we present a new assurance framework that evaluates the trustworthiness of hybrid systems, from traditional private networks to public clouds. It implements an assurance process that relies on a Virtual Private Network (VPN)-based solution to smoothly integrate with the target systems. The assurance process provides a transparent and non-invasive solution that does not interfere with the working of the target system. The performance of the framework have been experimentally evaluated in a simulated scenario.

1 INTRODUCTION

We live in a pervasive and connected society, where users as well as enterprises are engaging with digital technologies to carry out day-to-day activities and business processes. Recent years have been characterized by a continuous and fast evolution of communication and computation technologies towards public infrastructures, moving from service-based architectures to the cloud, and more recently to microservices and Internet of Things (IoT). At the same time, the importance of private infrastructures has a comeback pushed by an increasing need of data protection, which resulted in new regulations such as the General Data Protection Regulation (GDPR) in Europe. In this complex scenario, made of hybrid systems mixing public and private infrastructures, new concerns emerged undermining the users' perceived trust (e.g., (Teigeler et al., 2017)), as well as their confidence in the security of the overall systems.

The problem of guaranteeing the trustworthiness of such systems has been extensively studied by the research community in the last couple of decades, to the aim of fully unleashing their potential and foster widespread adoption. Security assurance stands out as the way to gain justifiable confidence that IT systems will consistently demonstrate one or more security properties, and operationally behave as expected, despite failures and attacks (Anisetti et al., 2017). It implements processes and techniques, based on audit, certification, compliance, supporting the assessment and verification of a target system behavior against security properties and requirements (Ardagna et al., 2015). Assurance solutions have been recently applied to service-based systems, including cloud and IoT systems (Ardagna et al., 2015; Baldini et al., 2016), introducing new frameworks addressing peculiar requirements such as scalability, multi-layer evaluation, and continuous monitoring. Although they provide outstanding benefits on the perceived trust, they lack of generality and cannot be easily adapted to current scenarios, where different services deployed on hybrid public and private infrastructures are composed at run time. Many of these solutions are in fact ad hoc (Cheah et al., 2018; Elsayed and Zulkernine, 2018), meaning they cannot handle a modern IT system as a whole. Moreover, existing assurance techniques, and corresponding frameworks, require some effort for being integrated with the target system, interfering with its normal operation (e.g., performance), and introducing not-negligible (monetary and business) costs.

[a] https://orcid.org/0000-0002-5438-9467
[b] https://orcid.org/0000-0001-7426-4795
[c] https://orcid.org/0000-0002-9557-6496

In this paper, we aim to fill in these gaps by proposing a new assurance framework enabling a centralized security assurance targeting both public and private infrastructures, including public and private cloud as well as traditional private systems. It implements an assurance process that relies on a Virtual Private Network (VPN)-based solution for a smooth integration with the target system, minimizing the interferences of the framework on the target system functioning. Our contribution is threefold. We first define the requirements a security assurance framework and corresponding process have to fulfill in our scenario made of hybrid systems. We then propose a novel VPN-based assurance framework and corresponding process addressing these requirements. To this aim, we introduce several modifications to a standard VPN configuration, including the so-called *Server-side NAT*, *Client-side NAT*, and a custom protocol to resolve conflicts between the networks in the VPN. We finally propose an experimental evaluation of our framework and comparison with the state of the art according to the identified requirements.

The remaining of this paper is organized as follows. Section 2 defines an assurance process and identifies the requirements it has to fulfill. Section 3 presents our assurance framework. Section 4 describes the VPN-based approach at the basis of our assurance process in Section 5. Section 6 presents an experimental evaluation of the framework performance in a simulated scenario. Section 7 proposes a comparison with the state of the art according to the identified requirements. Section 8 draws our final remarks.

2 ASSURANCE REQUIREMENTS

The advent and success of cloud computing and Internet of Things (IoT) are radically changing the shape of distributed systems. Hybrid systems, building on both private and public technologies, introduce new requirements and challenges on security assurance techniques, which must take a step forward for being applicable to modern architectures. In particular, the definition of new assurance processes is crucial to fill in the *lack of trustworthiness* that is one of the main hurdles against the widespread diffusion of such systems.

Despite targeting complex systems, a security assurance process should be lightweight and not interfere with the normal operation of the system under verification. The need of a lightweight process is strictly connected to its *psychological acceptability* (Saltzer and Schroeder, 1975), meaning that final users are more willing to *accept* to perform assurance activities that preserve the behavior of the system and do not increase overall costs. In fact, although the undebatable advantages given by a continuous evaluation of system security, users are recalcitrant with respect to a process perceived as heavy and costly (West, 2008).

Cost management and optimization are the foundation of assurance adoption. Costs refer to *monetary costs* in terms of additional human and IT resources, as well as *performance and business costs* in terms of overhead, latency, and reliability. *Monetary costs* include the need of highly specialized personnel, on one side, and resources allocated and paid on demand on the other side, which are spent to manage non-functional aspects of the system often considered as superfluous. *Performance costs* include the need of continuously verifying the security status of a system. They intrinsically introduce a not-negligible overhead and latency, an assurance process has to cope with. Assessment activities are only viable if they take resource demands under control, avoiding scenarios in which they become a source of attack. *Business costs* are partially overlapped with performance costs and model how much assurance activities interfere with the normal operations of a business process. On one side, changes required to the system to connect an assurance process should be reduced to the minimum, and mostly work at the interface level. On the other side, an assurance process cannot threat itself the system. For example, run-time verification of a system security status cannot increase the risk of system unavailability by performing penetration testing on the production system. A good balance between active and passive testing/monitoring should be provided.

We identify the main requirements an *assurance process* has to satisfy (MUST/SHOULD) to address the peculiarities of modern systems, as follows.

Transparency: it MUST not interfere with the normal operation of the business processes, being transparent to the final user of the system where the assurance process is performed.

Non-invasivess: it MUST require the least possible set of changes to the target system.

Safety: it MUST not introduce (or at least minimize) new risks on the target system.

Continuity: it SHOULD provide a continuous process, verifying the status of security while the system is operating and evolving.

Lightness: it SHOULD be lightweight and cope with systems having limited resources.

Adaptivity: it SHOULD be dynamic and incremental to adapt to changes in the system under verification and its environment.

Such requirements should be supported by a centralized framework tuning each aspect of the assurance evaluation. The *framework* itself has its own requirements (Anisetti et al., 2016), which are summarized in the following.

Evidence-based verification: it SHOULD implement a verification built on evidence collected on the target system, to get the real picture of its security status.

Extensibility: it MUST inspect hybrid targets, from traditional private networks to public clouds, as well as hybrid clouds and IoT.

Multi-layer: it SHOULD assess system security at different layers, from network protocols to application-level services.

Scalability: it SHOULD support a scalable process, able to manage an increasing number of assurance processes and evaluations.

Generally speaking, an assurance framework MUST *at least* implement a process that has the lowest possible impact on the target resources and normal system activities (*transparency*), do not modify the current ICT infrastructure or at least require very few modifications (*non-invasiveness*), do not affect security by introducing new risks (*safety*), while being generic enough to address peculiarities of hybrid systems (*extensibility*).

3 ASSURANCE FRAMEWORK

We extend our assurance framework in (Anisetti et al., 2018), designed for the assessment of *cloud* systems, to satisfy the requirements in Section 2 and address the peculiarities of modern environments. The framework in (Anisetti et al., 2018), in fact, does not fully satisfy property *extensibility* and needs to be modified to target those *private* deployments not directly reachable from the outside (e.g., *traditional* private corporate networks and private clouds). The simplest solution of moving assurance controls to the private network, directly connecting them to the target system, is not viable because it would cause the violation of different requirements in Section 2. For instance, it would affect properties *transparency* and *non-invasiveness*, since one or more backdoors should be coded in the target system, also affecting property *safety*. It would also interfere with property *adaptivity* constraining the ability of adapting the process to changes in the system. It would also increase costs, violating property *lightness*.

The framework in this paper aims to provide a lightweight solution based on Virtual Private Network (VPN) that embraces peculiarities of distributed systems, including cloud, microservices, and traditional private networks. In particular, it adopts a layer-3 VPN that connects the framework with the private deployments under verification (i.e., the *target networks*).

The architecture of the new assurance framework is presented in Figure 1, adding three components to the one in (Anisetti et al., 2018): *VPN Server*, *VPN Client*, and *VPN Manager*. In a nutshell, different *VPN Servers* are installed within the framework, each one responsible to handle isolated VPN tunnels with client devices placed in the target networks. A single VPN connection consists of a *VPN Client* directly connected to the target network, and a *VPN Server* installed in the framework. The framework manages an assurance process (Section 5) that consists of a set of evaluation rules (evaluations in the following). Each evaluation is a Boolean expression of test cases, which are evaluated on the basis of the evidence collected by *probes* and *meta-probes*. *Probes* are self-contained test scripts that assess the status of the given target by collecting relevant evidence on its behavior. They return as output a Boolean result indicating the success or failure of the test case. *Meta probes* are defined as *probes* collecting *meta-information*, such as the response time of a service. The framework components are summarized in the following.

Execution Manager manages the assurance process. Upon receiving an evaluation request, it selects the relevant *probes* and executes them. There are two types of *Execution Managers*: one targeting public clouds (*Public Execution Manager*), and one targeting private deployments (*Private Execution Manager*). The only difference between them is the way in which traffic is routed to the destination.

Evidence Analyzer produces the overall result of an evaluation by collecting the results of the single test cases composing the evaluation, and validating them against the Boolean expression of the evaluation.

Evidence Database stores the results of probe execution, including both the collected evidence and the Boolean results.

Dashboard is the user interface used to configure new evaluations and access their results.

VPN Server is a dedicated VM running the VPN software. It handles several VPN tunnels, one for each private network, which are strictly isolated.

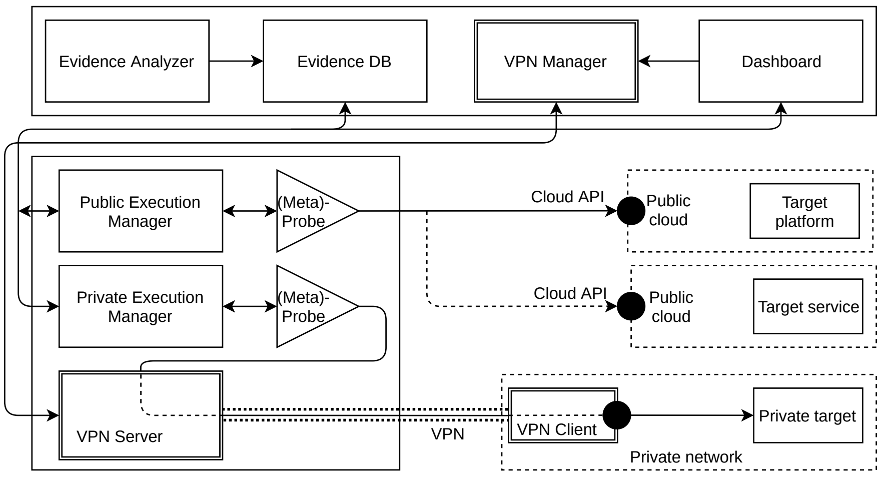


Figure 1: Our framework architecture. Double line rectangles highlight new components.

It acts as a default gateway for multiple *Private Execution Managers*.

VPN Client is physically located into the target network. It establishes a VPN connection with the *VPN Server* in the framework, traversing the firewall protecting the private network.

VPN Manager is a REST API service that manages the automatic configuration of the VPN. It automatically generates configuration files and handles all activities needed to manage VPN connections.

VPN Client and *VPN Server* are the stubs mediating the communication between the target system and the framework, respectively. They act as intermediaries supporting protocol translation and VPN working, and interacting with the *VPN Manager* for the channel configuration.

Example 3.1 *Let us consider an assurance evaluation targeting a public website composed of two test cases chained with a logic AND: i) a test case evaluating compliance against Mozilla best practices for websites and ii) a test case evaluating the proper configuration of HTTPS. The Execution Manager manages the assurance process as follows. Two probes are executed to collect the evidence needed to evaluate the two test cases, producing two Boolean results. Those results are then evaluated by the Evidence Analyzer according to the evaluation formula, a conjunction (AND) of test cases* i) *and* ii)*. As such, the overall evaluation is successful if and only if both test cases succeed.*

4 VPN-BASED METHODOLOGY

The assurance methodology implemented by the framework in Figure 1 builds on Virtual Private Network (VPN) to address the *must-have* requirements (i.e., *transparency*, *non-invasiveness*, *safety*, and *extensibility*) in Section 2. The goal is to provide a VPN-based solution that smoothly integrates our framework with the target system. In the following of this section, we briefly present the basis of VPN and discuss the reasons why it cannot be used as is to achieve our requirements; we then describe our VPN-based solution and how it differentiates from a common VPN configuration.

4.1 VPN in a Nutshell

Virtual Private Network (VPN) stands for a set of technologies used to build overlay networks over the public network. It provides hosts with remote access to a corporate network, or connects several geographically-distributed networks like they are separated by one router (Alshalan et al., 2016).

In this paper, we focus on *Site-to-Site VPN*, where several networks are connected using the VPN – rather than connecting one single host with a remote network. Usually there is one host per network connected to the VPN that acts as a *VPN gateway*, mediating traffic between internal hosts within its network and *other networks*. It routes traffic coming from internal hosts to the other *VPN gateways* and back. *VPN gateways* are called either *VPN clients* or *VPN servers*, where servers can handle connections to multiple clients, while a client establishes a single tunnel with a server.

VPNs usually combine a *virtual network interface*

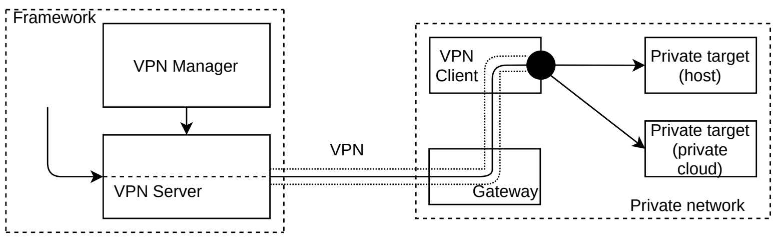


Figure 2: Architecture of VPN-Based Solution.

card (virtual NIC) and a socket-like connection. A virtual NIC is a NIC that has no physical correspondence, and is associated with a userspace process – in this case the VPN software. Packets *sent* by such process to its virtual NIC are *received* by the operating system (OS), and further processed just like a real network packet. At the same time, the OS can *send* packets to it, and the VPN software, through its NIC, will be the receiver. The socket-like connection is used to transmit packets between *VPN gateways* using a cryptographic VPN protocol. The virtual NIC is used to send and receive packets coming from and whose destination is the host's network.

Virtual NICs of the same VPN have IP addresses belonging to the same subnet, called *VPN subnet*. When the operating system of the *VPN gateway* handles a packet whose destination is a host in the *VPN subnet*, it sends the packet to the local virtual NIC, like a normal routing operation. Two sets of routing rules have to be defined: *i)* on each network, a rule on the default gateway that specifies to route traffic for *other networks* to the local *VPN gateway*; *ii)* on each *VPN gateway*, a rule that specifies to route traffic for *other networks* to the local virtual NIC.

Our final goal is to realize a Site-to-Site VPN between the framework and the private targets. However, a traditional VPN implementation does not permit to address many of the requirements in Section 2. The aforementioned routing rules, in fact, must be installed on both sides of the communication. Setting up these routes on the targets' default gateways requires access to the devices to alter their configurations. This violates properties *non-invasiveness*, *transparency*, and *safety*. Our approach avoids this, by adding several configurations on top of a standard VPN setup.

4.2 VPN-Based Solution

Figure 2 presents the VPN-based solution at the basis of our assurance framework. This solution is composed of two main parts: *i)* the mapping between the assurance framework and the private target (*framework-to-system mapping*), *ii)* the management of IP conflicts between the framework network and the private networks (*conflict-resolution protocol*).

4.2.1 Framework-to-System Mapping

The mapping between the assurance framework and the private target builds on two components: *VPN Client* and *VPN Server*.

VPN Client establishes a VPN connection with the server, exposing its network to the framework. It realizes a *Client-side NAT* that avoids setting up routing rules on the target network. The issue is that packets generated by the framework and injected by the *VPN Client* into the target network have a source IP address belonging to the framework network. As such, responses to such packets would be routed to the target network default gateway (because they appertain to a different network than the current one) instead of the *VPN Client*. To address this, we propose a lightweight approach based on network address translation (*NAT*), which does not require to configure default gateways. Once packets are received by the *VPN Client* from the framework through the VPN, it translates their source IP address in the *VPN Client* IP address. Since this belongs to the same subnet of the target hosts, no routes need to be configured. Responses can directly reach the *VPN Client*, where the destination IP address of the packets is translated back. We implemented this address translation with *nftables*, available in Linux-based operating systems.

VPN Server handles VPN tunnels with several clients; each tunnel is isolated to each other. It implements a *Server-side NAT*, to provide higher dynamics. There are two problems behind *Server-side NAT*, both involving routing configuration. On one side, *VPN Clients* need to know the network IP address of the framework (Section 4.1), on the other side, these routes must be known a priori, an assumption not trivial in our scenario. The network IP address of the framework, in fact, can change, for example, if the framework moves to a different cloud provider or for security reasons. We address the aforementioned problems by setting up different NAT rules on the *VPN Server*. They modify packets coming from

```
INPUT
$s \in S$: VPN Server
$n_O$: network to map

OUTPUT
$n_M$: mapped version of $n_O$

MAP_NET
available_nets ← db_query_select($s$);
if length(available_nets) != 0 then
  pair ← ⟨available_nets[0], $n_O$⟩;
  db_query_insert(pair);
else Error();
return pair;
```

```
INPUT
$n_O.j$: $j$-th IP address $\in$ network $n_O$

OUTPUT
$n_M.j$: $j$-th corresponding
      IP address $\in$ network $n_M$

MAP_IP
$n_O$ ← net_id($n_O.j$);
host_id ← host_id($n_O$, $n_O.j$);
$n_M$ ← get_corresponding_net($n_O$);
$n_M.j$ ← build_address($n_M$, host_id);
return $n_M.j$;
```

```
INPUT
$n_M.k$: $k$-th IP address $\in$ network $n_M$

OUTPUT
$n_O.k$: $k$-th corresponding
      IP address $\in$ network $n_O$

REMAP_IP
$n_M$ ← net_id($n_M.k$);
host_id ← host_id($n_M$, $n_M.k$);
$n_O$ ← get_corresponding_net($n_M$);
$n_O.k$ ← build_address($n_O$, host_id);
return $n_O.k$;
```

Figure 3: IP Mapping: Pseudocode

the framework just before being received by the virtual NIC of the VPN software. These rules change the source IP address of packets by replacing it with the virtual NIC IP address of the server. Thus, packets received by a *VPN Client* have a source IP address belonging to the current *VPN subnet*. Then, corresponding responses generated by the target hosts, after the application of *Client-side NAT*, have a destination IP address appertaining to the *VPN subnet*. Recalling that a *VPN Client* knows how to handle packets generated – or appearing to be generated – directly from the *VPN subnet*, the *VPN Client* OS can route those packets to the local virtual NIC, without additional configurations. They are then received by the VPN software and finally sent to the server. *Server-side NAT* is implemented as a set of *nftables* rules on the *VPN Servers*.

4.2.2 Conflict-Resolution Protocol

A mandatory requirement for a Site-to-Site VPN is that the network IP addresses of each participating network *must* be non-conflicting. Guaranteeing this assumption is necessary to allow a single VPN server to connect multiple networks together – in our case to allow a single *VPN Server* to handle several target networks. In corporate VPNs, it is trivial to assert this property, since the networks are under the control of the same organization. This assumption is not valid in our scenario, where two target networks could have the same network IP address, or a target network could conflict with the framework one. We propose an approach called *IP Mapping* to solve this issue.

IP Mapping is based on the the concept of *mapping* the *original* network to a new one, called *mapped* network and guaranteed to be unique. Each IP address of the *original* network is translated into a new one, belonging to the corresponding *mapped* network. This translation is reversible, and the *mapped* address is specified by the framework as the target when executing a new evaluation. *IP Mapping* is realized through three functions whose pseudocode is described in Figure 3. The overall protocol is completely transparent to the final user and works as follows. For simplicity, we consider an evaluation composed of a single test case.

First, when a new target network is being registered, the function *map_net* is invoked by the framework, to obtain a non-conflicting version of the *original* target network. The pair ⟨*original*, *mapped*⟩ is saved into the database. This function is offered through a REST API by *VPN Manager*.

When a user issues a new evaluation, it enters the *original* target IP address. The framework calls *map_ip* to obtain its *mapped* version, and builds the corresponding test case using this IP address as destination. The test packets are then sent through the VPN. Function *map_ip* is offered by *VPN Manager*.

The *VPN Client* receives the packets and calls *remap_ip* to get the *original* version of the destination IP address of the packets. This address is then set as the destination address: packets can now be sent to the target.

When corresponding responses reach back the *VPN Client*, the latter invokes *map_ip* to obtain the *mapped* version of the current IP source address; the result is set as the new IP source address. This second translation is issued in order to re-apply *IP Mapping* and let packets becoming correct responses to the ones generated by the framework. Finally, they are sent along the VPN and reach the framework.

Functions *map_ip* and *remap_ip* are implemented by a set of NAT rules using *nftables*.

The soundness of the overall VPN setup passes from *IP Mapping*, which, using the terminology in

Table 1: Comparison of a standard layer-3 VPN and a layer-3 VPN with our modifications on top.

	Standard layer-3 VPN	Our approach
Client-side requiring configuration	Yes	No (*Client-side NAT*)
Server network known a priori	Yes	No (*Server-side NAT*)
Conflicting networks	Not allowed	Allowed (*IP Mapping*)
Address conflict resolution	Manual	Automatic (*VPN Manager*)
Plug-and-play integration	No	Yes

Figure 3, must support the following properties.

1. *Mapping uniqueness*: let $A \subseteq n_M \times S$; $\forall\ a_i, a_j \in A$, $(a_i.s = a_j.s \wedge a_i \neq a_j) \Rightarrow (a_i.n \neq a_j.n)$
2. *Mapping correctness*: $\forall\ n_O\ \forall$ address $\in n_O$ $remap\_ip(map\_ip(address)) = map\_ip(address)^{-1}$
3. *Implementation correspondence*: $\forall\ n_O$, $\forall$ address $\in n_O$, $map\_ip'(address) = map\_ip''(address)$

The first property expresses that no conflicts can happen, that is, two *mapped* networks with the same network IP address attached to the same *VPN Server* cannot exist. The second property expresses the reversibility of the translation process. It guarantees that a response to *mapped* packets generated by the framework is correct, that is, the source IP address of a response is equal to the destination IP address of a request. The third property expresses the need of having two implementations of *map_ip* (as a REST API or NAT rule) with the same behavior. We note that the pseudocode shown in Figure 3 is a possible implementation of the three functions.

Table 1 summarizes the differences between a standard VPN and the one described in this paper. Our solution does not require any configurations on the target network, thanks to *Client-side NAT*; it also does not require to know the network IP address of the framework, thanks to *Server-side NAT*. Moreover, the networks participating in the VPN can have conflicting IP addresses, which are automatically disambiguated by *IP Mapping* and *VPN Manager*. To conclude, our solution allows a plug-and-play integration between the framework and the target network.

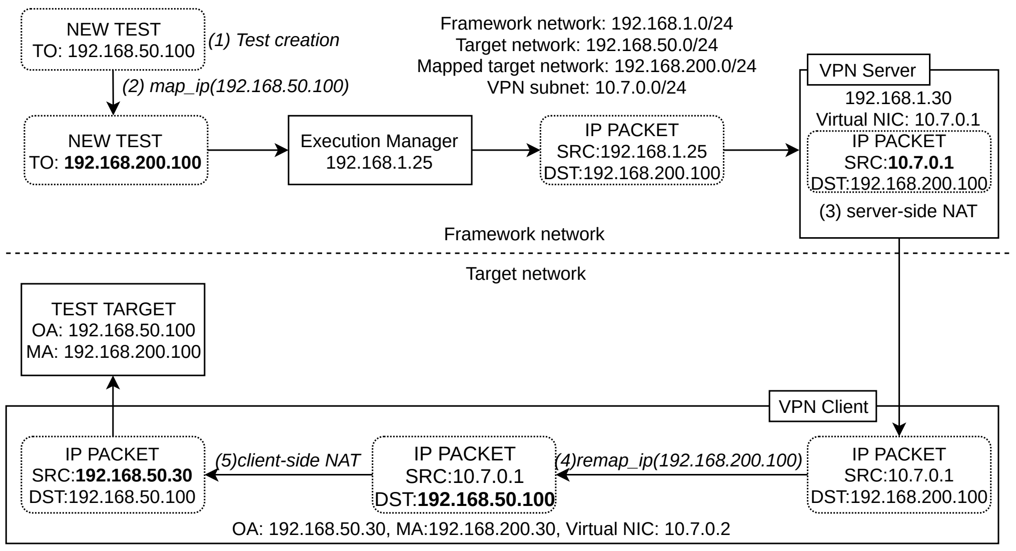
5 ASSURANCE PROCESS

The overall assurance process is composed of three phases: *i)* connection setup, *ii)* assurance request (Figure 4(a)), *iii)* assurance response (Figure 4(b)). We note that, in Figure 4, we denote *Mapped Address* and *Original Address* as *MA* and *OA*, respectively.

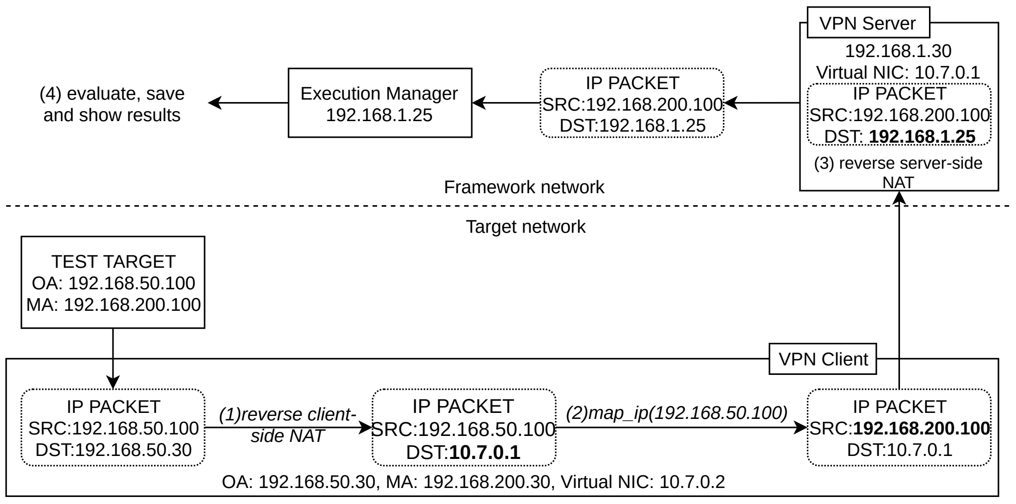
Connection setup. It starts with the user registering the net ID of a new network in the framework. The framework calls the REST API *map_net* and retrieves the *mapped* version of the input network. As described in Section 4.2.2, this mapping is stored in the framework database and triggers the creation of a new *VPN Client*. *VPN Manager* also configures *VPN Server* to support connections from that client. The client device is then moved into the correct location and connected to the server. For the sake of discussion, we consider the following sample network configuration: framework net ID `192.168.1.0/24`, target net ID `192.168.50.0/24`, mapped target net ID `192.168.200.0/24`, and VPN subnet net ID `10.7.0.0/24`; we also consider an evaluation with a single test case.

Assurance request. The assurance request in Figure 4(a) starts with the user submitting an evaluation request to the framework (*Step (1)* in Figure 4(a)), specifying the IP address of the target (`192.168.50.100` in our example). The framework then calls the *VPN Manager* REST API *map_ip* (*Step (2)*), obtaining the *mapped* version of target address (`192.168.200.100`). *Private Execution Manager* executes the *probe* corresponding to the requested test case against the *mapped* IP address. The packets generated by the *probe* are then sent to the *VPN Server*. Upon receiving them, *VPN Server* applies *Server-side NAT* (*Step (3)*), which changes the source IP address of the packets to its virtual NIC address (`10.7.0.1`). Modified packets are then sent through the VPN, finally reaching *VPN Client*. At this point, *VPN Client* executes function *remap_ip* (*Step (4)*), which replaces the destination IP address of the packets with their *original* version. It then applies *Client-side NAT* (*Step (5)*), which changes the source IP address from the *VPN Server* virtual NIC address to its IP address (`192.168.50.30`). Finally, test packets reach their target.

Assurance response. The assurance response process in Figure 4(b) starts when the test target sends back response packets to the *VPN Client*. This phase applies the assurance request steps in the reverse order, to correctly forward responses to the *probe*. *VPN Client* first executes the reverse of *Client-side NAT*, by replacing the destination IP address of the packets with the *VPN Server* virtual NIC (*Step (1)* in Figure 4(b)). It then applies *map_ip* to change the source IP address (`192.168.50.100`) with the correspond-



(a) Assurance request



(b) Assurance response

Figure 4: Packet flow: (a) assurance request, (b) assurance response.

ing *mapped* version (`192.168.200.100`) (*Step (2)*). Next, packets are forwarded to the *VPN Server*. Upon their reception, *VPN Server* applies the reverse of *Server-side NAT* (*Step (3)*). This step changes the destination address of the packets from the address of the *VPN Server* virtual NIC (`10.7.0.1`) to the address of the *Execution Manager* (`192.168.1.25`). Finally, packets reach the *probe*, and results are evaluated and stored (*Step (4)*).

It is important to note that the involvement of users in this process is very limited; users need only to provide the IP address of the target. The framework then manages *IP Mapping* and performs every step in a transparent way, guaranteeing *transparency* and *non-invasiveness* of the process, avoiding any configurations on the target system.

6 EXPERIMENTAL EVALUATION

We provide a performance evaluation of our framework and corresponding assurance process.

6.1 Settings

Our framework has been realized on top of *OpenVPN*, a flexible and open-source VPN solution that permits to tune every aspect of a VPN tunnel. In particular, we configured a layer-3 VPN, using TCP as the encapsulating protocol, to maximize the probability of traversing firewalls in the path from the framework to the target system. *Client-side NAT*, *Server-side NAT* and *IP Mapping* have been implemented as NAT rules with *nftables*. *Execution Manager* and *VPN Server* have been installed in two virtual machines (VMs) running the operating system *CentOS 7 x64*, both equipped with 1 CPU and 4 GBs of RAM. *VPN Server* runs *OpenVPN* version *2.4.6* and *nftables* version *0.8*. Both VMs have been deployed on a Dell PowerEdge M360 physical host that features 16 CPUs Intel® Xeon® CPU E5-2620 v4 @ 2.10 GHz and 191 GBs of RAM.

The target system has been deployed on AWS EC2 and was composed of two virtual machines *t2.micro*, both with 1 vCPU and 1 GB of RAM. The first one, *VPN Client*, with operating system *Ubuntu 18.04 x64*, *OpenVPN* version *2.4.7*, and *nftables* version *0.8*. The second one, test target, with operating system *Ubuntu 16.04 x64* offering *WordPress* version *5.2.2*.

We finally setup two experiments with the goal of computing the overhead of our solution, which vary the target system deployment: *i)* public deployment exposing the target on the public network, *ii)* private deployment using the approach in Section 4.2. We run the same evaluations measuring the overhead our approach adds on top of public deployment verification.

6.2 Performance and Discussion

We executed the following evaluations against the two deployments of the target system.

- *Infowebsite* that extracts as much information as possible from a target website. It is denoted as `E1` in Figure 5.
- *Observatory-Compliance* that checks whether a website has implemented common best practices, such as HTTPS redirection and cross-site-scripting countermeasures. It is denoted as `E2` in Figure 5.
- *SSH-Compliance* that checks the compliance of a SSH configuration against Mozilla SSH guidelines. It is denoted as `E3` in Figure 5.
- *TLS-strength* that evaluates whether the TLS channel has been properly configured, such as avoiding weak ciphers and older versions of the protocol. It is denoted as `E4` in Figure 5.
- *WordPress-scan* that scans the target *WordPress*-based website looking for *WordPress*-specific vulnerabilities. It is denoted as `E5` in Figure 5.

We chose these evaluations to maximize test coverage and diversity, from the evaluation of web re-

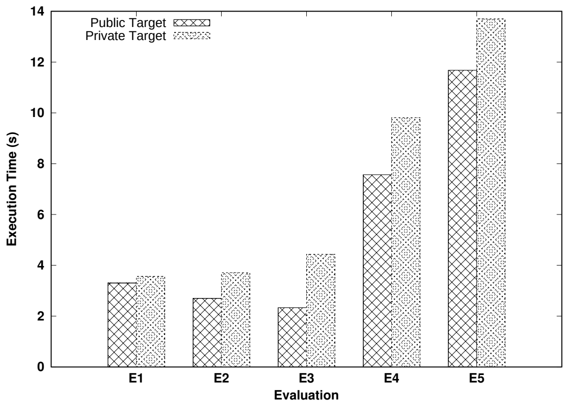


Figure 5: Execution times of evaluations `E1` – *Infowebsite*, `E2` – *Observatory-Compliance*, `E3` – *SSH-Compliance*, `E4` – *TLS-strength*, `E5` – *WordPress-scan*.

sources (*Infowebsite*, *Observatory-Compliance*), to the evaluation of protocol configurations (*SSH-Compliance*, *TLS-strength*) and specific applications (*WordPress-scan*). Each evaluation was executed 10 times and the average time was computed. Figure 5 presents the average execution time of evaluations `E1`–`E5`. It shows that, as expected, the execution time in the private scenario is higher than the same in the public scenario, with an overhead varying between ≈0.3s and 2s. More in detail, evaluation `E1` (*Infowebsite*) experienced a very low overhead, less than a second. Evaluation `E2` (*Observatory-Compliance*) experienced an overhead of approximately 1 second. Evaluations `E3`, `E4` and `E5` (*SSH-Compliance*, *TLS-strength*, *WordPress-scan*, resp.) experienced a higher overhead, approximately 2 seconds, increasing execution time from ≈2s to ≈4s for `E3`, from ≈8s to ≈10s for `E4` and from ≈12s to ≈14s. Overall, the increase in the execution time was globally under control, never exceeding 2 seconds. This overhead can be tolerated in all scenarios supporting requirements in Section 2.

To conclude, there is a subtlety to consider when an assurance process for hybrid systems is concerned: the accuracy of the retrieved results. There could be some cases in which the evidence collected by a *probe* on a public endpoint is different from the one collected by the same *probe* on a private endpoint. For instance, evaluation `E1` (*Infowebsite*), in the private scenario, failed to discover the version of the target *WordPress* website. This was due to a partial incompatibility between the *probe* implementation and our VPN-based solution. Being our approach *probe*-independent, this issue can be solved by refining the *probe* associated with *Infowebsite*. In our experiments, evaluation `E1` was the only experiencing such problem, while the other evaluations have been able to collect

Table 2: Comparison of frameworks for security assurance with the one in this paper.

References	Transparency	Non Invasiveness	Safety	Continuity	Lightness	Adaptivity
(Alcaraz Calero and Aguado, 2015)	✓	∼	∼	✓	✗	✓
(Cheah et al., 2018)	∼	✗	∼	✗	✗	✗
(Ciuffoletti, 2016)	✓	∼	✓	✓	✓	∼
(De Chaves et al., 2011)	∼	∼	✓	✓	✗	∼
(Elsayed and Zulkernine, 2018)	∼	✓	∼	✓	✗	∼
(Jahan et al., 2019)	✗	✗	∼	✓	✗	✓
(Ouedraogo et al., 2010)	✗	✗	✗	✓	✗	✗
(Povedano-Molina et al., 2013)	✓	∼	✗	∼	∼	✗
(Wu and Marotta, 2013)	✗	✗	✗	✗	✓	✓
(Anisetti et al., 2018)	✓	∼	∼	✓	∼	∼
This paper	✓	✓	∼	✓	∼	✓

(a) Process requirements

Reference	Evidence-based verification	Extensibility	Multi-layer	Scalability
(Alcaraz Calero and Aguado, 2015)	∼	✗	✓	∼
(Cheah et al., 2018)	✓	✗	∼	✗
(Ciuffoletti, 2016)	∼	✗	∼	✓
(De Chaves et al., 2011)	∼	∼	∼	∼
(Elsayed and Zulkernine, 2018)	∼	✗	∼	✓
(Jahan et al., 2019)	✗	✗	✗	✗
(Ouedraogo et al., 2010)	✗	∼	∼	∼
(Povedano-Molina et al., 2013)	∼	✗	✓	✓
(Wu and Marotta, 2013)	✗	∼	∼	✗
(Anisetti et al., 2018)	✓	✗	✓	∼
This paper	✓	∼	✓	∼

(b) Framework requirements

the same evidence in both private and public deployments. We leave the analysis of this issue for our future work.

7 COMPARISON WITH EXISTING SOLUTIONS

Many solutions for security assurance have been presented in literature, moving from software-based systems (Herrmann, 2002) to service-based environments (Ardagna et al., 2015), providing certification, compliance, and audit solutions based on testing and monitoring. We analyzed the main assurance frameworks and processes, which can be classified according to the following categories: *monitoring-based*, *test-based* and *domain-specific*. Table 2 provides a comparison of these frameworks, including the one in this paper, with respect to requirements in Section 2.

Monitoring-based frameworks. (Aceto et al., 2013) provided a comprehensive survey of assurance solutions based on monitoring. They first considered property *intrusiveness*, which is similar to our requirements *transparency* and *non-invasiveness*, and found that many commercial monitoring tools do not address such property. They then considered requirement *lightness*, because monitoring tends to be expensive in term of resource consumption. Two monitoring frameworks have been presented in (Alcaraz Calero and Aguado, 2015; De Chaves et al., 2011), both building on monitoring tool *Nagios*. Due to the intrinsic nature of monitoring, these frameworks can easily satisfy the requirement *continuity*. Moreover, the work in (Alcaraz Calero and Aguado, 2015) can achieve a very good *adaptivity* and offers a monitoring platform both for cloud providers and users. Nevertheless, they require a significant effort in terms of setting up the monitoring infrastructure and resources for maintaining it, thus violating requirements *non-invasiveness* and *lightness*. Framework in (De Chaves et al., 2011) has also proven to suffer of *extensibility* and *scalability* issues (Taherizadeh et al., 2018). (Povedano-Molina et al., 2013) described a monitoring framework called *DARGOS*, built with scalability and flexibility in mind. Being fully distributed, it supports *scalability* and can be enriched with more sensors. However, being specifically tailored for the cloud, it cannot be easily adapted to other scenarios. (Ciuffoletti, 2016) presented a novel approach, where a simple, cloud-independent API-based solution has been used to configure monitoring. Such cloud-agnosticism is realized through an OCCI (*Open Cloud Computing Interface*) extension, designed towards

Monitoring-as-a-Service. The author also proposed how the cloud providers should implement such functionalities in their backends.

Test-based frameworks. (Wu and Marotta, 2013) presented a work-in-progress testing-based framework that instruments client binaries to perform cloud testing. The main issue is that binaries instrumentation may not be always feasible, and might also introduce undesired behavior in modified programs. As such, the framework fails to satisfy requirements *transparency*, *non-invasiveness*, and *safety*. (Ouedraogo et al., 2010) presented a framework that uses agents to perform security assurance, although agents themselves need to be properly secured. (Greenberg et al., 1998) claimed that, to protect hosts from agent misuse or attacks, several techniques need to be properly employed. Agents also pose a maintenance problem: they have to be kept updated, and things can only become worse as the number of agents increases. Also, they introduce substantial costs since they need to be physically installed on each host/device to be assessed and coordinated, introducing not-negligible network traffic. For these reasons, the agent-based framework in (Ouedraogo et al., 2010) does not satisfy requirements *transparency*, *non-invasiveness* and *safety*. (Jahan et al., 2019) discussed *MAPE-SAC*, a conceptual approach for security assurance of self-adaptive systems, where the system itself changes, and security requirements must adapt to these changes. While it is not possible to completely evaluate our requirements due to the lack of a real, implemented framework, *MAPE-SAC* fulfills requirements *adaptivity* and *continuity*. A different solution has been given in our work in (Anisetti et al., 2018), where we described the cloud-ready framework briefly summarized at the beginning of Section 3. As already discussed, the proposed approach is based on *probes* and *meta-probes*, and fails to address requirements *extensibility*, *non-invasiveness*, and partially, *safety*.

Domain-specific frameworks. They most recent category of assurance frameworks. (Elsayed and Zulkernine, 2018) described a distributed framework for monitoring cloud analytics applications, based on analyzing logs produced by such applications. The proposed approach requires very few configurations at the cloud side, and can be offered through the *Security-as-a-Service* paradigm. (Cheah et al., 2018) considered the automotive world, where cases are generated after evaluating the severity of threats. Threats are found through threat modeling and confirmed with a penetration testing. The usage of penetration testing violates requirement *non-invasiveness*. Usually, solutions in this category cannot claim property *extensibility*.

To conclude, the comparison in Table 2 shows that the existing frameworks (and corresponding processes) do not even come close to addressing the requirements in Section 2. In general, existing solutions mainly target *continuous evaluation* and *multi-layer* infrastructures, as well as *transparency* and *adaptivity*, failing to achieve *non-invasiveness*, *safety*, *lightness*, and *extensibility*. The framework in this paper, instead, provides a first boost in this direction addressing, at least partially all requirements in Table 2. Following the comparison therein, this paper leaves space for future work. We will first aim to extend our framework towards Big Data and IoT environments, further improving *extensibility*, *lightness*, and *scalability*. We will also focus on strengthening the *safety* of the framework and its components, for example the *Execution Manager* that can easily become a single point of failure/attack.

8 CONCLUSIONS

Security assurance is increasingly adopted as the solution to verify whether a distributed system holds some security properties and behaves as expected. Current approaches and, when available, frameworks often target a specific system lacking extensibility and have a not-negligible impact on the functioning of the system target of the verification. These issues represent big hurdles towards assurance adoption, especially when hybrid systems are considered. In this paper, we presented a VPN-based assurance framework that smoothly integrates with hybrid systems, from private networks to public clouds. The framework implemented an assurance process with limited impact and costs on the target system, while providing a safe and scalable approach. The present work leaves space for further work. First, our VPN-based approach can be extended to increase *extensibility*, addressing more domains and their peculiarities (e.g., IoT). Second, it can be refined to increase the *safety* of the framework and its core components (i.e., *Execution Manager*, *VPN Server*).

ACKNOWLEDGMENTS

Research supported, in parts, by EC H2020 Project CONCORDIA GA 830927 and Università degli Studi di Milano under the program “Piano sostegno alla ricerca”.

REFERENCES

Aceto, G., Botta, A., de Donato, W., and Pescapè, A. (2013). Cloud monitoring: A survey. *Computer Networks*, 57(9):2093 – 2115.

Alcaraz Calero, J. M. and Aguado, J. G. (2015). Monpaas: An adaptive monitoring platformas a service for cloud computing infrastructures and services. *IEEE TSC*, 8(1):65–78.

Alshalan, A., Pisharody, S., and Huang, D. (2016). A survey of mobile vpn technologies. *IEEE COMST*, 18(2):1177–1196.

Anisetti, M., Ardagna, C., Damiani, E., and Gaudenzi, F. (2016). A certification framework for cloud-based services. In *Proc. of ACM SAC*, Pisa, Italy.

Anisetti, M., Ardagna, C., Damiani, E., and Gaudenzi, F. (2017). A semi-automatic and trustworthy scheme for continuous cloud service certification. *IEEE TSC*.

Anisetti, M., Ardagna, C., Damiani, E., Ioini, N. E., and Gaudenzi, F. (2018). Modeling time, probability, and configuration constraints for continuous cloud service certification. *Computers & Security*, 72:234 – 254.

Ardagna, C., Asal, R., Damiani, E., and Vu, Q. (2015). From security to assurance in the cloud: A survey. *ACM CSUR*, 48(1):2:1–2:50.

Baldini, G., Skarmeta, A., Fourneret, E., Neisse, R., Legeard, B., and Le Gall, F. (2016). Security certification and labelling in internet of things. In *2016 IEEE 3rd World Forum on Internet of Things (WF-IoT)*.

Cheah, M., Shaikh, S. A., Bryans, J., and Wooderson, P. (2018). Building an automotive security assurance case using systematic security evaluations. *COSE*, 77:360 – 379.

Ciuffoletti, A. (2016). Application level interface for a cloud monitoring service. *CS&I*, 46:15 – 22.

De Chaves, S. A., Uriarte, R. B., and Westphall, C. B. (2011). Toward an architecture for monitoring private clouds. *IEEE Comm. Mag.*, 49(12):130–137.

Elsayed, M. and Zulkernine, M. (2018). Towards security monitoring for cloud analytic applications. In *Proc. of IEEE BigDataSecurity/HPSC/IDS 2018*, Omaha, NE, USA.

Greenberg, M. S., Byington, J. C., and Harper, D. G. (1998). Mobile agents and security. *IEEE Comm. Mag.*, 36(7):76–85.

Herrmann, D. (2002). *Using the Common Criteria for IT security evaluation*. Auerbach Publications.

Jahan, S., Marshall, A., and Gamble, R. (2018). Self-adaptation strategies to maintain security assurance cases. In *Proc. of IEEE SASO 2018*, Trento, Italy.

Jahan, S., Pasco, M., Gamble, R., McKinley, P., and Cheng, B. (2019). Mape-sac: A framework to dynamically manage security assurance cases. In *Proc. of IEEE FAS*W 2019*.

Ouedraogo, M., Mouratidis, H., Khadraoui, D., and Dubois, E. (2010). An agent-based system to support assurance of security requirements. In *Proc. of SSIRI 2010*, Singapore.

Povedano-Molina, J., Lopez-Vega, J. M., Lopez-Soler, J. M., Corradi, A., and Foschini, L. (2013). Dargos: A highly adaptable and scalable monitoring architecture for multi-tenant clouds. *FGCS*, 29(8):2041–2056.

Saltzer, J. H. and Schroeder, M. D. (1975). The protection of information in computer systems. *Proceedings of the IEEE*, 63(9):1278–1308.

Taherizadeh, S., Jones, A. C., Taylor, I., Zhao, Z., and Stankovski, V. (2018). Monitoring self-adaptive applications within edge computing frameworks: A state-of-the-art review. *JSS*, 136:19 – 38.

Teigeler, H., Lins, S., and Sunyaev, A. (2017). Chicken and egg problem: What drives cloud service providers and certification authorities to adopt continuous service certification? In *Proc. of WISP 2017*, Seoul, South Korea.

West, R. (2008). The psychology of security. *Commun. ACM*, 51(4):34–40.

Wu, C. and Marotta, S. (2013). Framework for assessing cloud trustworthiness. In *Proc. of IEEE CLOUD 2013*, Santa Clara, CA, USA.