Building a Framework for Automated Security Testbeds in Cloud Infrastructures

Roman-Valentyn Tkachuk
Blekinge Institute of Technology
roman-valentyn.tkachuk@bth.se

Dragos Ilie
Blekinge Institute of Technology
dragos.ilie@bth.se

Kurt Tutschku
Blekinge Institute of Technology
kurt.tutschku@bth.se

***Abstract*—When exposed to the network, applications and devices are exposed to constant security risks. This puts pressure on hardware and software vendors to test even more than before how secure applications and devices are before being released to customers.**

We have worked towards defining and developing a framework for automated security testbeds. Testbeds comprise both the ability to build on-demand virtual isolated networks that emulate corporate networks, as well as the ability to automate security breach scenarios, which accelerates the testing process. In order to accomplish both features of the testbed, we have based the framework on well-established cloud and orchestration technologies *e. g.*, OpenStack and Ansible. Although many of these technologies are powerful, they are also complex, leading to a steep learning curve for new users. Thus, one of the main goals of the developed framework is to hide the underlying complexities through a template approach and a simplified user interface that shortens the initial training time.

In this paper, we present the full stack of technologies that were used for constructing the testbed framework. The framework allows us to create entire virtual networks and to manipulate network devices started in it, via comprehensive yet simple interfaces. Also, we describe a specific testbed solution, developed as a part of the Test Arena Blekinge project.

***Keywords*-Security Testbed; Cloud Infrastructures; Infrastructure as a Service; Infrastructure as Code;**

I. Introduction

As we witness an increase in the frequency of security incidents, ordinary defensive mechanisms, *e. g.*, *firewalls* or *IDSs/IPSs*, must be complemented by offensive actions, *e. g.*, *penetration testing* or *vulnerability analysis*. The reason is that many modern applications rely on multiple distributed components (*e. g.*, cloud services) interconnected over the Internet and located in different administrative domains. This makes perimeter-based defenses less effective as attackers try to exploit vulnerabilities in the application logic to laterally move towards where the assets are located. The aforementioned offensive testing actions help to proactively find and patch vulnerabilities that enable this type of intrusions. However, when executed towards a production environment, these actions may disrupt or degrade the operation of the application. It would be useful if one can replicate the application within a testbed where it can be subjected to offensive actions without disturbing the production system.

A testbed is a controlled environment in which one can execute a set of available actions in a secure and isolated manner without prejudicing the process being controlled [1], while being able to observe and log the outcome of the actions.

There has been a considerable amount of work done in the area of security testbeds, but due to space reasons we provide only two examples. In [2], a testbed framework for IoT devices has been outlined and a unified structure of the test has been proposed. However, virtual infrastructure creation was not considered as far as this testbed is designed to test physical IoT devices. In [3], authors describe a technical solution of cloud-based testbed but do not describe a generic framework nor security test methods, which would help in building new testbeds for security testing.

Infrastructure as a Service (IaaS) solutions, *e. g.*, *OpenStack* [4], permit the setup and deployment of scalable environments for rapid and flexible service testing. Also, IaaS has broad capabilities for provisioning network devices, which allows us to emulate corporate networks given sufficient resources in the cloud. The configuration and deployment of cloud-based systems can be done manually, using friendly graphical user interfaces (GUI). However, this approach can become a highly time consuming and error-prone process when provisioning must be done repeatedly, as in scenarios where the application is developed using a continuous integration, testing and deployment (CI/CT/CD) approach. In order to make provisioning process more efficient, IaaS Systems employ the idea of *Infrastructure as Code (IaC)* [5], which is the process of provisioning of computing infrastructure and its configuration through machine-readable definition files. The advantages of IaC code have been addressed in [6], along with scalability demonstration of one of the tools covered by the study.

While IaC makes the process of network devices provisioning more efficient, one still needs the special knowledge to write provisioning scripts, maintain the provisioned infrastructure and operate the cloud. In order to decrease the initial training period to use the testbed, a comprehensive interface has to be built. This interface is a part of the framework which allows us to connect to the IaaS cloud and use it as a platform for security tests while having no need to know how the cloud operates internally. Security test's process flow is shown in Fig. 1. While there is only a GUI exposed to the User, the task of the framework is to translate settings from GUI to machine-readable definition files. Further, definition files are translated into executable commands by IaC and executed on the IaaS

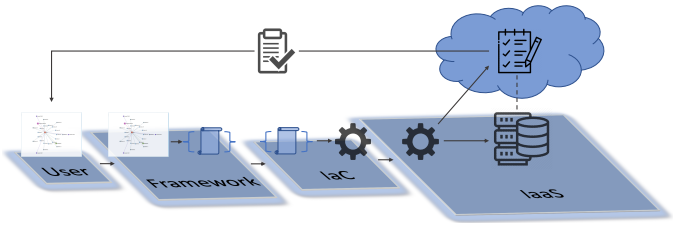

Fig. 1. Security test's process flow.

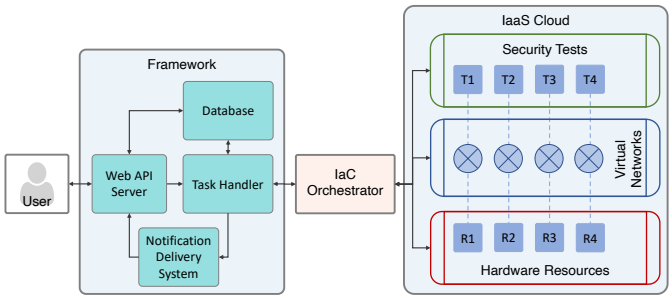

Fig. 2. Testbed Architecture.

cloud. When execution is finished, the results are displayed on the User's GUI.

As part of the Test Arena Blekinge project that was carried out in Karlskrona, Sweden, we have investigated architecture and technologies suitable for automating a testbed for security. Here we present the results of our study. The reminder of the paper is structured as follows. Section II discusses requirements for the architecture of the security testbed and the framework we put forth to meet those requirements. Section III introduces the implemented testbed, technologies that have been used and challenges that authors met during the implementation. Section IV outlines some the lessons we learned on the capabilities and limitation of virtualized testbeds for security testing. Finally, Section V summarizes the developed framework, capabilities of the implemented testbed, and provides an outlook to future research.

II. Testbed Architecture

The testbed has to meet a set of requirements in order for the security tests to be conducted in a safe and automated manner, and provide the possibility to gather and organise the information about test results. First, the testbed has to provide a possibility to fully control the test network infrastructure, providing isolation from the Internet and from other physical or virtual networks that co-exist on the same IaaS Cloud. Second, all devices and services emulated inside the virtual network must be accessible through an interface which enables control of the objects under test. Third, the testbed must offer an interface for collecting log information from all test objects so that it is possible to evaluate the state of the network and services at a specific moment or after the test completes.

The focus of this work is to define a framework that addresses these requirements. Fig. 2 depicts the test framework in the context of the testbed architecture. The architecture consists of three distinct entities, *Framework*, *IaC Orchestrator* and *IaaS Cloud*, which are described below.

A. Framework

The *Framework* is the entity that is exposed to *User* and contains the interface to perform all needed actions to conduct security tests. The main aim of the Framework is to hide the complexity of the *IaaS Cloud* through a human comprehensive and interactive interface. To interact with the testbed, the User sends requests to *Web API Server*. The Web API Server exposes a unified API, through which the User can configure the test infrastructure and security scenario. The set of supported API calls comprises virtual network creation, network devices configuration, security test scenario automation, and data collection. In order for the Web API Server to be able to associate User's security tests to his/her identity, the User has to provide a set of credentials on the initial request to Web API Server. These credentials are saved in the *Database*, along with security test settings and debug information.

Network infrastructure instantiating and device provisioning are tasks that may take a considerable amount of time to execute. The *Task Handler* is the architectural component that handles the execution of long duration tasks. After the User configures all settings and starts a test, the Web API Server transfers the settings to the Task Handler, which carries out all further actions. The Framework components must be able asynchronously notify each other about various events that occur during the test execution. The *Notification Delivery System* handles the delivery of all notifications and enables live on-demand status updates on the User side.

While the security test is being executed, Task Handler executes commands on IaC Orchestrator according to the settings provided by the User.

B. IaC Orchestrator

The *IaC Orchestrator* (further *Orchestrator*) is an entity that contains logic for interacting directly with the IaaS Cloud. We have chosen to implement this functionality outside the Task Handler to decouple cloud functionality from framework functionality. This allows us in the future to modify the IaC orchestrator independently of the other components when we want to support different forms of IaaS. The Orchestrator performs all security test provisioning and automation actions. Its operation is controlled through machine-readable definition files, which are translated by its engine into actions that are performed on the IaaS Cloud.

C. IaaS Cloud

The *IaaS Cloud* (further *Cloud*) is the entity where the actual execution of the security test takes place. Thus it is important that all malicious actions are concealed within the virtual network that is dedicated to the test. User has to be authorized to use the amount of cloud resources needed to conduct a security test of a certain scale. In turn, the Cloud has to provide a possibility for Orchestrator to create and configure virtual networks, along with the ability to separate

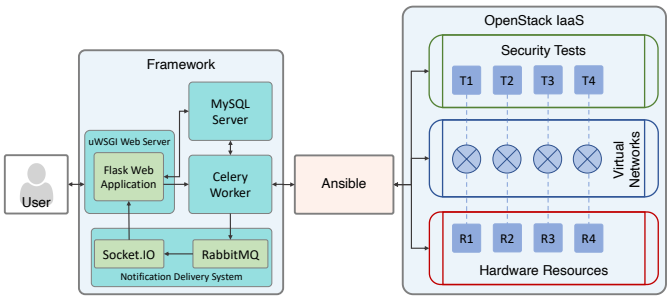

Fig. 3. Implemented Testbed.

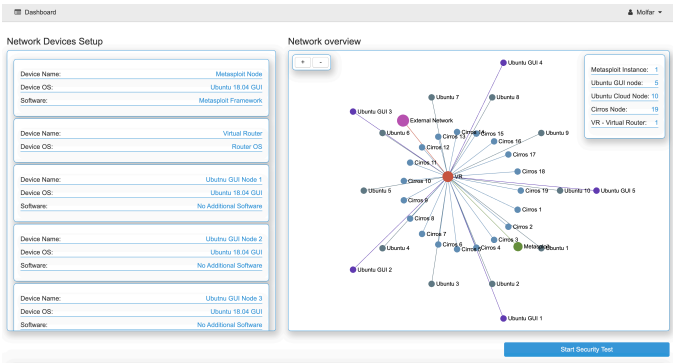

Fig. 4. Testbed's graphical user interface.

security tests on both hardware and network level. Hardware-level separation is important when running multiple tests simultaneously to avoid having one security test corrupt the test environment and thus, cause other tests to fail (when they should not). Similarly, network-level separation isolates traffic generated by concurrent tests, to avoid that traffic from one test (*e. g.*, DoS attack) influences the outcome of other tests (*e. g.*, tests that evaluated the robustness of specific services or applications).

III. Implementation

Our current implementation of the framework has enough functionality to conduct both automated and manual security tests. Also, it features a Web-based GUI, which allows the operator to pre-configure security tests, follow the network infrastructure setup process, and collect all necessary data during the test and after its finish. Fig. 3 shows the technologies used for implementing the testbed architecture.

For IaaS Cloud we have chosen *OpenStack IaaS* [4]. It is a well-proven, powerful IaaS solution that fulfills all requirements outlined in Section II. It has the capability to construct multilevel virtual networks and to isolate security experiments on both hardware and network level. Also, it is distributed with an open-source license, which is a major factor when considering the costs of implementation. Our OpenStack IaaS runs on the university campus and comprises *24 VCPUs*, *70 GB of RAM* and *5 TB of Storage*.

For IaC Orchestrator, we have chosen *Ansible IaC* [7], a cloud-agnostic solution that is well suited for provisioning OpenStack IaaS. The choice is based on our previous study that we described in [6], where we compared different industrial IaC tools. The reason for this choice is that Ansible supports both declarative and imperative programming approaches. Also, its YAML-based language for machine-readable definition files is easy to start with. Finally, Ansible's server-only architecture decreases the amount of software and configuration that is needed to be deployed and maintained.

A. Framework Implementation

The user interacts with the Framework through a *Flask Web Application* served by a *uWSGI Web Server*. Flask Web Application is a server-side lightweight RESTfull Web Application that is well integrated with all technologies that were used to build the framework. For user and security test data storage, *MySQL Server* has been used. One of the challenges that we faced was in building the *data access layer (DAL)* for Flask Web Application to support a large amount of database operations. MySQL Server's capability to store some of the functionality inside of the database engine in a form of *stored procedures* eased the task of DAL creation and made Flask Web Application less monolithic.

The Task Handler was implemented with a *Celery Worker*. Celery is a Python-based task executor framework, which can be used as a separate service running on-site or remotely from the Flask Web Application. The advantage of Celery Worker is that it can use all Python libraries, thus enabling us to reuse the DAL from the Flask Web Application.

The *Notification Delivery System* consists of two technologies that work as part of a whole. *Rabbit Message Queue (RabbitMQ)* is a service that runs continuously and accepts the messages from Celery Worker. At the moment the message is received and registered in the queue, the *Socket.IO* event communicator transfers the message to the Flask Web Application, which delivers it to the User through the uWSGI Web Server.

The GUI of the implemented framework is shown in Fig. 4. In the left pane *Network Devices Setup*, one can see the information about each device included in the test. There are only a few types of devices available in the current testbed library. However, their descriptions can be used as templates to create new devices from scratch, or by cloning existing images and adding additional software on top. The right pane *Network Overview*, shows graphically the network topology used for the test.

B. Supported Security Tests

Currently, we have implemented two security experiments in our testbed. The first one is about scanning IPv6 networks to discover active hosts. The scanning can be useful, for example, for penetration testing exercises. However, the tremendous size of IPv6 networks poses a serious challenge for classic scanning algorithms that iterate sequentially over the entire address space. We have developed a set of IPv6 scanning heuristics [8] to overcome this challenge by trading accuracy for execution time. We are using the testbed to deploy virtual IPv6 networks and evaluate the performance of the heuristics.

This test is fully automated and when the test completes it returns the set IPv6 addresses that were found in the subnet.

Second test is about using Metasploit [9] to exploit network devices and services that are part of the test. The Metasploit framework is an exploitation and vulnerability validation tool that is widely used for penetration testing. When the network devices are started in the testbed, the user can define the security experiment scenario through remote VNC desktop to every network device.

IV. Lessons Learned

The implementation of the testbed permitted insights beyond the technical understanding how the different hard- and software components interact. The implementation and the execution of security tests taught additionally how such virtualized testbeds can be used in general for automatic testing of security and, thus, enhancing the methodology of using virtual testbeds. Next, the findings and learned lessons will be described.

Lesson 1 – Consideration of Stakeholders: The implementation, operation and use of a testbeds has to be based on an analysis of its stakeholders and roles, such as testbed operator, testing company, or device developer. The availability of a GUI makes the testbed accessible for less security-experienced users, *e. g.,* device developers. However, the simplifications by the GUI may limit scope of the investigations. This characteristic might lead to a trade-off between the generality of the testbed and the time to achieve test results. In addition, the operation of the underlying virtualization systems, here OpenStack, is still difficult and a role of a testbed service provider is advised.

Lesson 2 – Challenges in the Inclusion of Cyberphysical Systems-under-Test (CP-SUTs): Virtual testbeds have outmarked advantages in being able to scale experiments to a very large number of nodes. However, the scalability with regard to CP-SUTs might be not given or limited since not sufficient physical implementations of CP-SUTs are available or since the accuracy of the digital virtualized counterparts (sometimes denoted as "digital twins") is too low.

Lesson 3 – Automation Supports Fast and Different Types of Scalability: The use of automation and orchestration techniques support new, various ways of scalability in virtual security testbed. Obviously, the first category of scalability is the increased in number SUTs. Additionally, the automation permits the algorithmic generation of a large number test scenarios and, thus, permitting a fast execution of a spectrum of test scenarios. However, there might be a considerable trade-off between the required efforts for accurate automatic configuration generation and the manual generation of threat and test scenarios.

V. Conclusions and Future Work

In this work, we defined the components of a security testbed framework, outlined an architecture for such testbed, detailed an specific implementation and discussed the capabilities, limitations and lessons learned from the testbed implementation for security testing.

The suggested framework is targeted to ease the tasks of virtual network creation and automation of security tests through comprehensive interface which hides the complexity of IaaS clouds. While defining the framework, three main requirements have been considered: a) *full control over the virtual network infrastructure and separation of security tests on hardware and network levels*, b) *full control over the network devices instantiated in virtual network*, c) *data collection capability from all network devices*. By fulfilling these requirements, the testbed has enough functionality to emulate corporate networks and devices installed in them, and is capable to provide enough information about the security and robustness of the tested service.

Also, we describe the implementation of the testbed which is deployed on site of Blekinge Institute of Technology. It contains enough functionality to be able to conduct both automated and manual security tests as well as fulfills all requirements defined for the testbed.

The current architecture of the testbed, although is designed to be cloud-agnostic, can use only one IaaS provider at a time. For future research we would like to investigate the possibility of federated testbed, connecting multiple IaaS clouds at a time, and having a possibility to run cross-cloud security tests.

Acknowledgements

The authors would like to thank the funding project that is supporting the work presented in this paper: The work was partly sponsored by the *Swedish Agency for Economic and Regional Growth (Tillväxtverket)* under the umbrella of European Structural and Investment Funds (ESI) covered by grant agreement No 20201213 (Test Arena Blekinge). The project is run in collaboration with Blue Science Park in Karlskrona, Sweden.

The authors would also like to thank Mårten Holmberg from Blue Science Park for his support in defining the test infrastructure.

References

[1] INCIBE. (2017, Jan.) Analysing security without risk: Testbeds. Accessed: 2020-03-20. [Online]. Available: https://www.incibe-cert.es/en/blog/analysing-security-without-risk-testbeds

[2] S. Siboni, V. Sachidananda, Y. Meidan, M. Bohadana, Y. Mathov, S. Bhairav, A. Shabtai, and Y. Elovici, "Security testbed for internet-of-things devices," *IEEE Transactions on Reliability*, vol. 68, no. 1, pp. 23–44, Mar. 2019.

[3] R. L. Grossman, Y. Gu, M. Sabala, C. Bennett, J. Seidman, and J. Mambretti, "The open cloud testbed: A wide area testbed for cloud computing utilizing high performance network services," *CoRR*, Jul. 2009.

[4] OpenStack. (2020) What is openstack? Accessed: 2020-03-20. [Online]. Available: https://www.openstack.org/software/

[5] K. Morris, *Infrastructure As Code: Managing Servers in the Cloud*, 1st ed. O'Reilly Media, Inc., 2016.

[6] R.-V. Tkachuk, D. Ilie, and K. Tutschku, "Orchestrating future service chains in the next generation of clouds," in *Proc. SNCNW*, Luleå, Sweden, Jun. 2019.

[7] Red Hat. (2020) Ansible. Use Case: Orchestration. Accessed: 2020-03-20. [Online]. Available: https://www.ansible.com/use-cases/orchestration

[8] E. Bergenholtz, D. Ilie, A. Moss, and E. Casalicchio, "Finding a needle in a haystack – a comparative study of IPv6 scanning methods," in *Proceedings of IEEE ISNCC*, Istanbul, Turkey, Jun. 2019.

[9] Rapid7. (2020) Metasploit: Documentation. Accessed: 2020-03-20. [Online]. Available: https://metasploit.help.rapid7.com/docs