Cassiopée Project 2018-2019: Development and deployment of an automated IT security audit tool in a virtualized environment.

Aurélien Duboc, Pierrick Gorisse, Lucas Martin
Supervisor: Hervé Debar

Contents

List of Figures

1 Start of the project

Large companies, as well as SMEs are subject to a security obligation for their information systems. Our project proposes a tool that allows the management of the main vulnerabilities that security auditers usually look for. This tool identifies weaknesses and / or configuration vulnerabilities.
The main interest lies in the automated analysis of a large number of machines. It could also propose automated corrections associated with these weaknesses. Our solution could be equivalent to a security audit for a company.

1.1 Objectives

1.1.1 Infrastructure

In order to provide a realistic framework for the implementation of such a tool, it is necessary to establish an active infrastructure resulting from the use of network services such as Virtual Private Networks (VPN), Domain Name Servers (DNS), reverse proxy, monitoring servers... It is also resulting from the use of personal services and data, as some users of this infrastructure host web and ftp servers on it.

1.1.2 Setting up a security scan tools

This tool mainly focused on the deployment of security audit tools for computer systems running Linux at least. Windows security audit is an optional feature that is a project-building perspective.

1.1.3 Management of the logs

The management of the logs associated with these audits, helps detect evidence of an attack in the logs of network devices, servers, and applications. The web interface aggregates and manages log data from built-in detection capabilities and from logs produced by other devices in your environment. It automatically executes advanced analysis, producing normalized events and correlating them to produce actionable intelligence, alerting us of any threats to the environment.

1.1.4 Web interface

The development of a web interface allowing data management and visibility over the data produced by the log management tool.

1.1.5 Build an opensource project

This tool is opensource but need some competencies required. It needs to master UNIX-like platforms (Linux, MacOS) and associated tools, including software development tools (editors, interpretors, etc.). Coding in JavaScript and / or scripting languages, including referenced frameworks for JavaScript development (e.g. Node.js, Angular.js, etc.) might be usefull. As Elasticsearch and Kibana are working as APIs, the frontend development part works as a REST API (Representational State Transfer), the REST standards has to be mastered as well in order to contribute on this project.

1.1.6   Learning objectives

Processing of accessible textual data (coherence management, etc.) , the presentation and analysis of textual data as well as the understanding of cybersecurity software and integration.

1.1.7   Documentation

The use of English for our project is essential. Indeed, all the technical documentation associated with the resources used is in English and we are used to working with resources in English because they are much more complete. In addition, the community providing these resources exchanges mainly in English. In order to offer a tool that is widespread and easy to use, writing the documentation in English then appears as a better choice.

1.2   Technical solutions

1.2.1   Setting up a hypervisor for managing virtualized content

The chosen solution is Proxmox, an open source hypervisor that can provide container based virtual environment, using OpenVZ. It supports guest operating systems like Linux (KVM), Windows. It is enabled by the presence of integrated backup service. It delivers full system virtualization by the use of KVM. The Proxmox management interface can function using a normal browser. Proxmox is using the cluster mode: from a single page multiple servers can be managed. It can also perform a direct migration from one host to another. Lastly, Proxmox provides shell access to the KVM directly from its interface, using a Debian system.
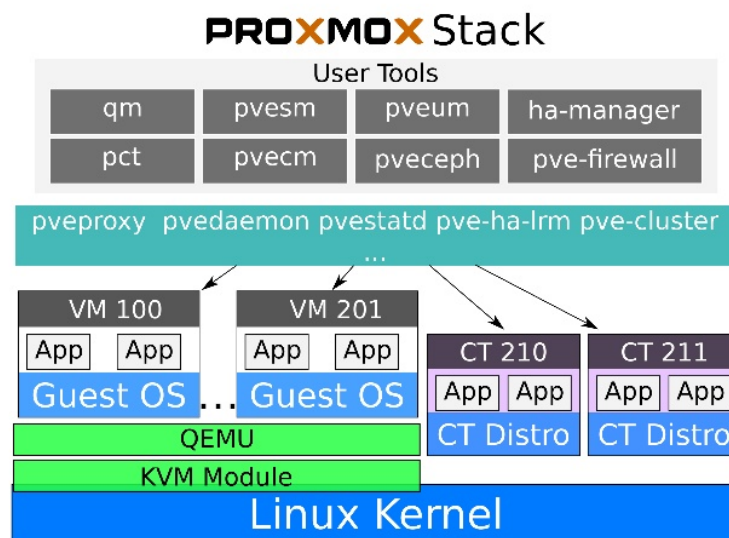
Figure 1: Proxmox stack architecture

Proxmox VE tightly integrates KVM (Kernel-based Virtual Machine) hypervisor and LXC containers, software-defined storage and networking functionality on a single platform, and easily manages high availability clusters and disaster recovery tools with the built-in web management interface.

Qemu is running with the support of the virtualization processor extensions, via the Linux KVM module. In the context of Proxmox VE, Qemu and KVM can be used interchangeably. Indeed, Qemu will always try to load the KVM module in Proxmox.

*https://www.proxmox.com/en/

1.2.2   Setting up a directory to manage users and associated access control policies

We use an LDAP directory [RFC 4510]. This allows a standardized representation of information (database - LDAP directory) as well as a standard query protocol widely deployed for this database. The chosen solution is the OpenLDAP software version 2.4.46, because of its maturity and protocol compliance. PhpLDAPadmin is the web interface that allows the management of user accounts. One of the most important fields is sshPublicKey because all the servers are configured to do LDAP queries during an SSH connection to list the authorized RSA keys.
We could have used a PAM setup with libpam-ldap but it seemed easier to specify in ssh configuration files an AuthorizedKeyCommand that will reach all the RSA public keys on the LDAP server.
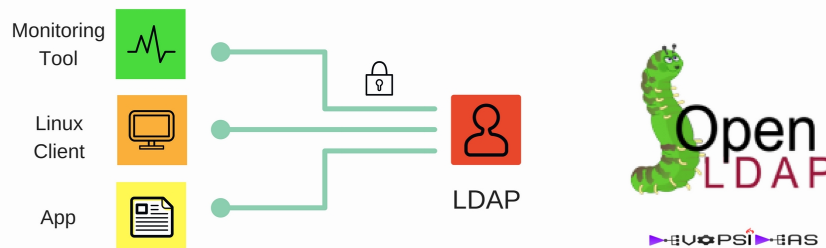*`https://ldap.com/basic-ldap-concepts/`

Figure 2: Web Based LDAP Client

1.2.3   Use of a tool allowing access to all virtualized machines while using the access control protocol cited above

We chose Ansible, an open source software that automates software provisioning, configuration management, and application deployment. Ansible connects via SSH, remote PowerShell or via other remote APIs.
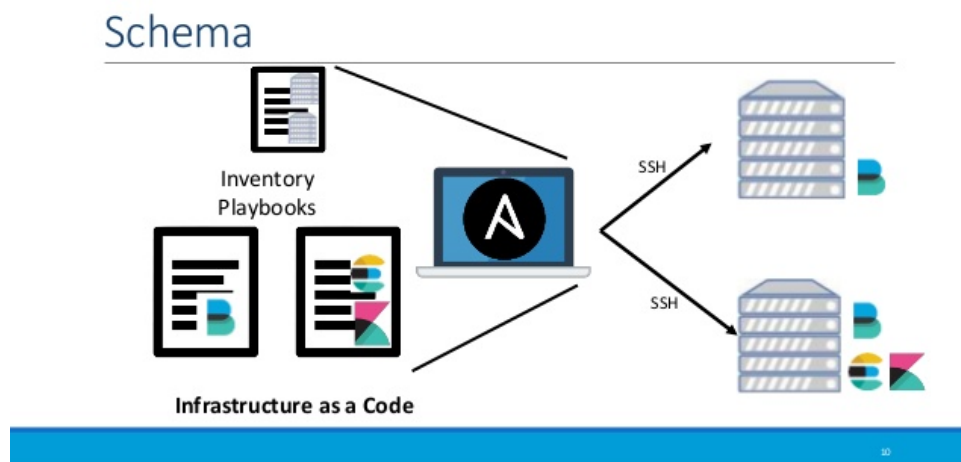* (`https://www.ansible.com/`)

Figure 3: Ansible playbooks deployment

1.2.4   Implementation of a list of tools allowing the automated audits added to our personal contribution

For now, Lynis is the best candidate. We will probably combine several tools later. Lynis is an extensible security audit tool for computer systems running Linux, FreeBSD, macOS, OpenBSD, Solaris, and other Unix-derivatives.

Lynis scanning is opportunistic, meaning it will only use what it can find, like available tools or libraries. The benefit is that no installation of other tools is needed, so you can keep your systems clean. By using this scanning method, the tool can run with almost no dependencies. Also, the more it finds, the more extensive the audit will be. In other words: Lynis will always perform scans that are customized to your system and two audits will never be the same.

Many vulnerability scanners perform on a network level (outside of the system). They can detect missing security patches due to discovered weaknesses. However, in many cases leaks can be present, and their detection via the network close to impossible. An additional downside is version banners on which some of the tools rely, providing you with a false positive when the software vendor is using a patched version.

Lynis focuses on scanning from the inside, on the system itself. This doesn't mean it has to be installed on the system though. Lynis can run from local or external storage and only requires root permissions. The big benefit from running it on the system itself is that all information is available, including running processes, open network ports, being able to discover user accounts etc. Depending on your needs and how in-depth a security scan has to be, scanning from the inside might be a preferred method. More information will be available, while the chance of getting false positives is lower as well.

Figure 4: Output from Lynis scan

1.2.5   Management of the logs

ELK is the most famous tool for log processing and analysis, so naturally we will use this tool to generate our logs. ELK is the acronym for three open source projects: Elasticsearch, Logstash, and Kibana. Elasticsearch is a search and analytics engine. Logstash is a server-side data processing pipeline that ingests data from multiple sources simultaneously, transforms it, and then sends it to a stash like Elasticsearch. Kibana lets users visualize data with charts and graphs in Elasticsearch.
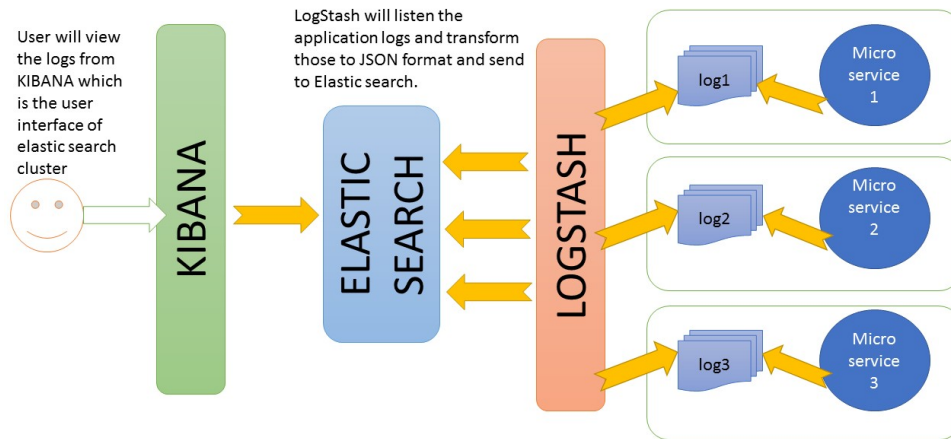* (https://www.elastic.co/fr/elk-stack)

Figure 5: ELK stack interaction with different applications based on Log file

1.2.6   Web interface development

The use of a framework associated with a certain number of libraries will allow us to obtain a modular and easy to use application. Symfony is a PHP framework that we use to work with, which is why we chose to use this one. Moreover, we were able to verify that there were many libraries usable by this framework allowing us to interface ELK with our web application.
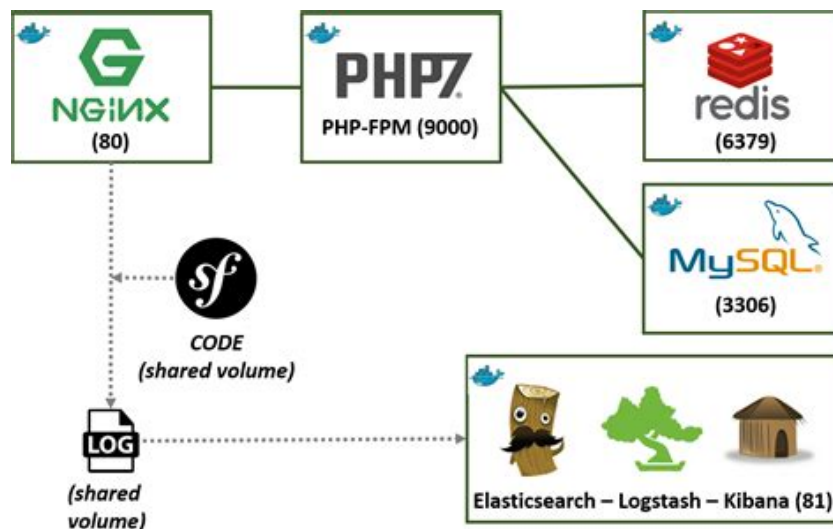* (https://pehapkari.cz/blog/2017/10/22/connecting-monolog-with-ELK/)

Figure 6: First web application architecture

2 Implementation of the project

2.1 Update about the web framework

We finally chose to use Flask framework instead of Symfony, after further study of the necessary features:

- Elegant syntax, clear namespaces (good code-maintainability in long run).

- Rich ecosystem and libraries (sqlalchemy, numpy, scipy, matplotlib).

- Out of box support for everything (such as large floating point numbers).

- Package management using pip is a breeze.

The use of Proxmoxer (`https://github.com/swayf/proxmoxer`) is also one of the main reasons for switching to a Python framework.

Proxmoxer is a wrapper around the Proxmox REST API v2. It was inspired by slumber, but it is dedicated only to Proxmox. It allows to use not only REST API over HTTPS, but the same API over ssh and pvesh utility as well. Like Proxmoxia it dynamically creates attributes which responds to the attributes you've attempted to reach.

```python
from proxmoxer import ProxmoxAPI
proxmox = ProxmoxAPI('proxmox_host', user='proxmox_admin',
                                    backend='ssh_paramiko')
for node in proxmox.nodes.get():
    for vm in proxmox.nodes(node['node']).openvz.get():
        print "{0}. {1} => {2}" .format(vm['vmid'], vm['name'], vm['status'])
```

Figure 7: Snippet to get LXC and Qemu data over Proxmox API

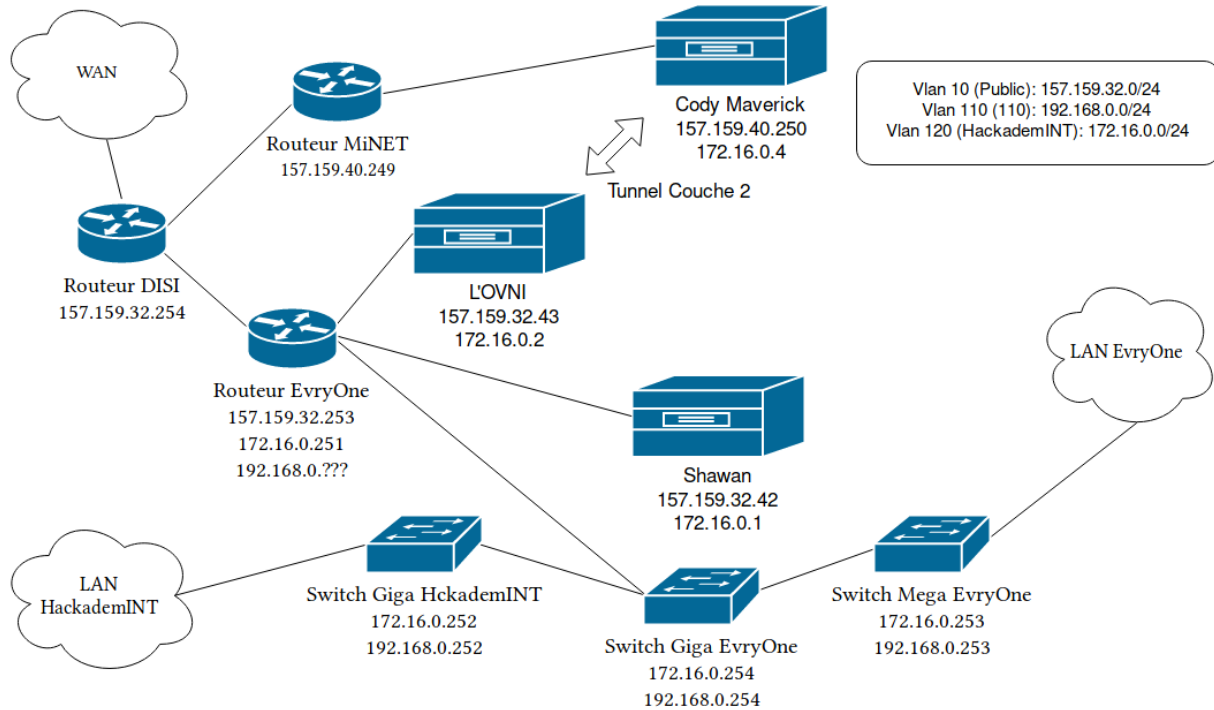## 2.2   Network Architecture

Figure 8: Schema of the network architecture

Three physical servers (whose names are "Cody-Maverick", "L'OVNI" and "Shawan") have been clustered on which are deployed the Proxmox hypervisor. These are DELL PowerEdge 1950 servers. The two Gigabit switches are Cisco SG500-52P 52-port Gigabit POE Stackable Managed Switch. The Megabit one is a Cisco Catalyst 3750.

To connect to different machines in the infrastructure, the application knows the IP addresses of the different machines, thanks to the API of the hypervisor. RSA private keys must be inserted in by the system administrator within the application to allow connection to different machines.

2.2.1   Technical specs of the servers

The dual processor Dell PowerEdge 1950 delivers next generation performance in a 1U, rack dense chassis with Quad-Core Intel Xeon processors. This high concentration of computing power and redundancy makes the PowerEdge 1950 the perfect choice for high performance computing clusters (HPCC), SAN front-end, web and infrastructure applications. Outstanding manageability in this latest generation 1U server delivers new functionality for managing the servers from remote locations.

2.2.2 Virtual LANs

A virtual LAN (VLAN) is any broadcast domain that is partitioned and isolated in a computer network at the data link layer (OSI layer 2). LAN is the abbreviation for local area network and in this context, virtual refers to a physical object recreated and altered by additional logic. VLANs work by applying tags to network frames and handling these tags in networking systems – creating the appearance and functionality of network traffic that is physically on a single network, but acts as if it is split between separate networks. In this way, VLANs can keep network applications separate despite being connected to the same physical network, without requiring multiple sets of cabling and networking devices to be deployed.

The three clustered servers are separated by three routers and we do not have access to one of them, so we have established a network infrastructure in which we have installed VLANs while the subnets are schemas-filled. In order to realize the cluster between the servers, we set up an Ethernet layer 2 bridge (see next subsection).

2.2.3 Layer 2 ethernet bridging

Layer-2 bridging works by putting one physical and one virtual Ethernet adapter into a mode where they can receive traffic that is not destined for their address. This traffic is selectively sent onto the other network according to the IEEE 802.1D standard, known as "bridging" the frames. Frames transmitted by virtual Ethernet adapters on the same VLAN as the bridging virtual Ethernet adapter can be sent to the physical network. Frames sent from the physical network can be received by adapters on the virtual network.

2.3   Application Architecture

Here is the detail of the architecture as currently designed for this project:
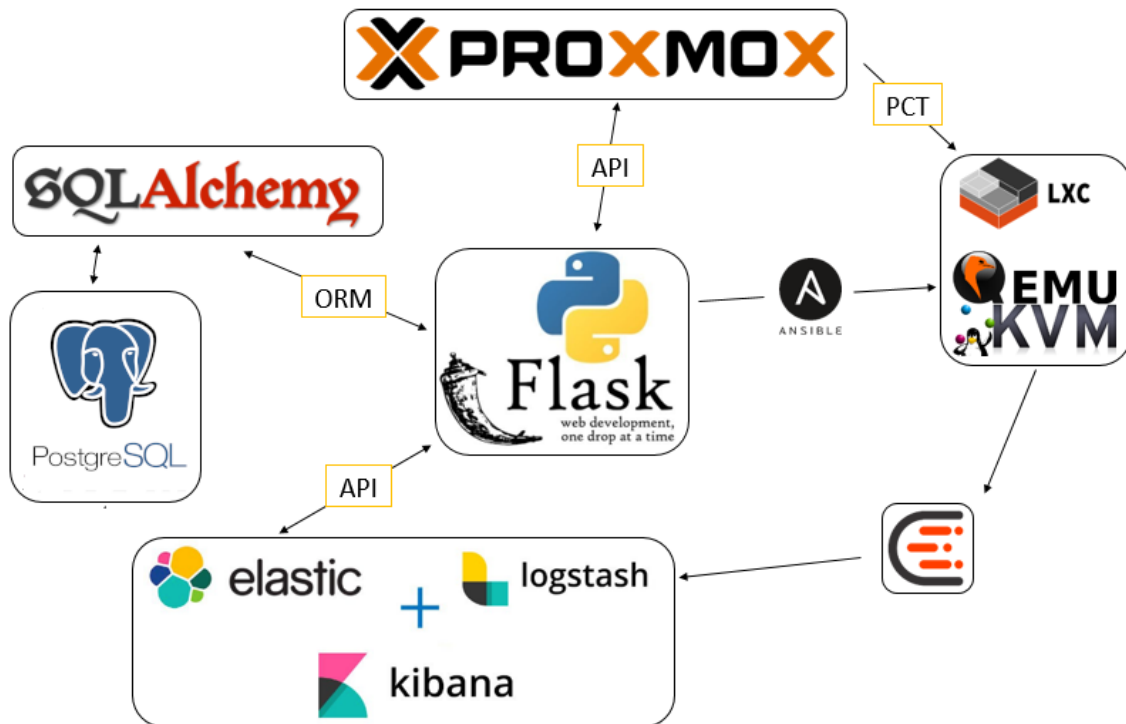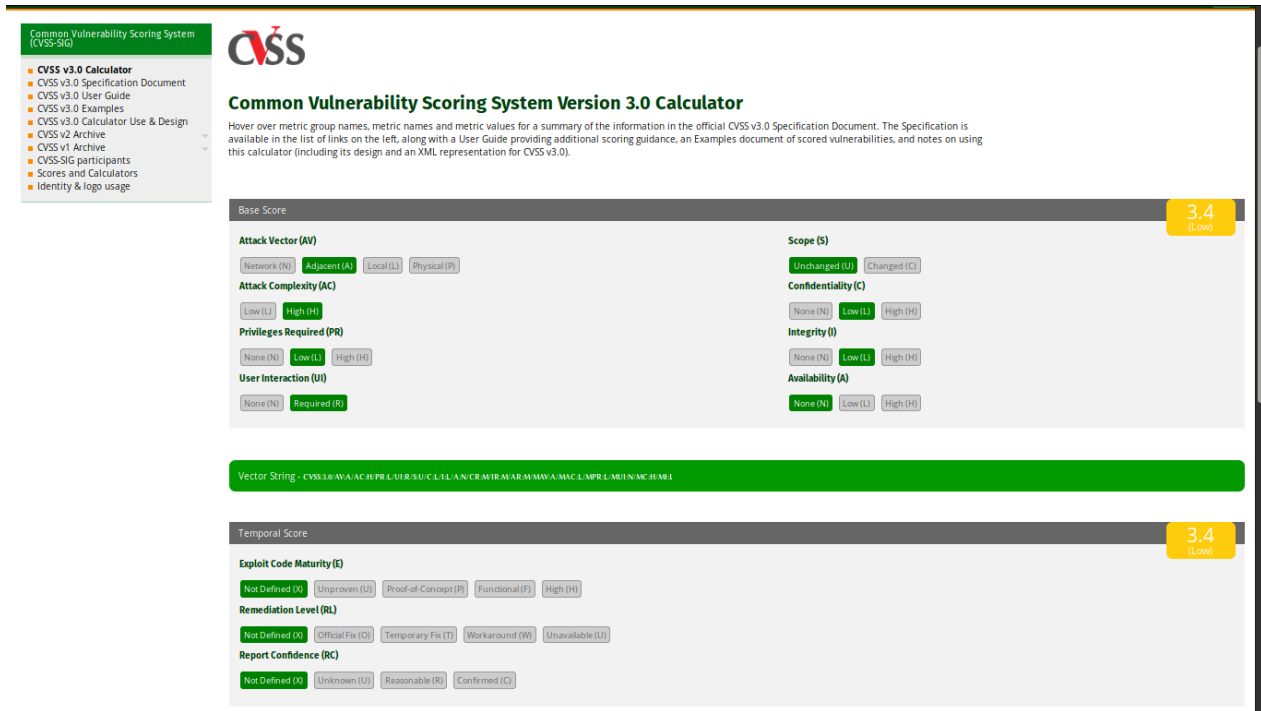

Figure 9: Application architecture schema

- Flask application interacts with this hypervisor through its API. We so use proxmoxer as a wrapper to interface with it.

- It is possible to access containers and virtual machines through pct (Tool to manage Linux Container (LXC) on Proxmox VE) and qmu (Qemu / KVM Virtual Machine Manager).

- By using the SSH network protocol, the Ansible tool helps with the execution of Lynis security audit tool on the different containers. This generates a log file which will then be parsed in the ELK stack, itself interfaced with the application. We thus obtain a complete report of the security of every machine within the application.

- User management of the application is performed at a base level postgresql data that the application accesses via the SQlachemy ORM.

2.4   Scoring and standardization

With Lynis, many tests are part of common security guidelines and standards, completed with additional security tests. After the scan a report will be displayed with all discovered vulnerabilities. Our application will map these vulnerabilities with a CVSS Base, Temporal and Enrironmental Score.

Figure 10: Common Vulnerability Scoring System Version 3.0 Calculator

The Common Vulnerability Scoring System (CVSS) is a free and open industry standard for assessing the severity of computer system security vulnerabilities. CVSS attempts to assign severity scores to vulnerabilities, allowing responders to prioritize responses and resources according to threat. Scores are calculated based on a formula that depends on several metrics, that approximate ease of exploit and the impact of exploit. Scores range from 0 to 10, with 10 being the most severe. While many utilize only the CVSS Base score for determining severity, temporal and environmental scores also exist, to factor in availability of mitigations and how widespread vulnerable systems are within an organization, respectively.

https://www.first.org/cvss/calculator/3.0

3 Results

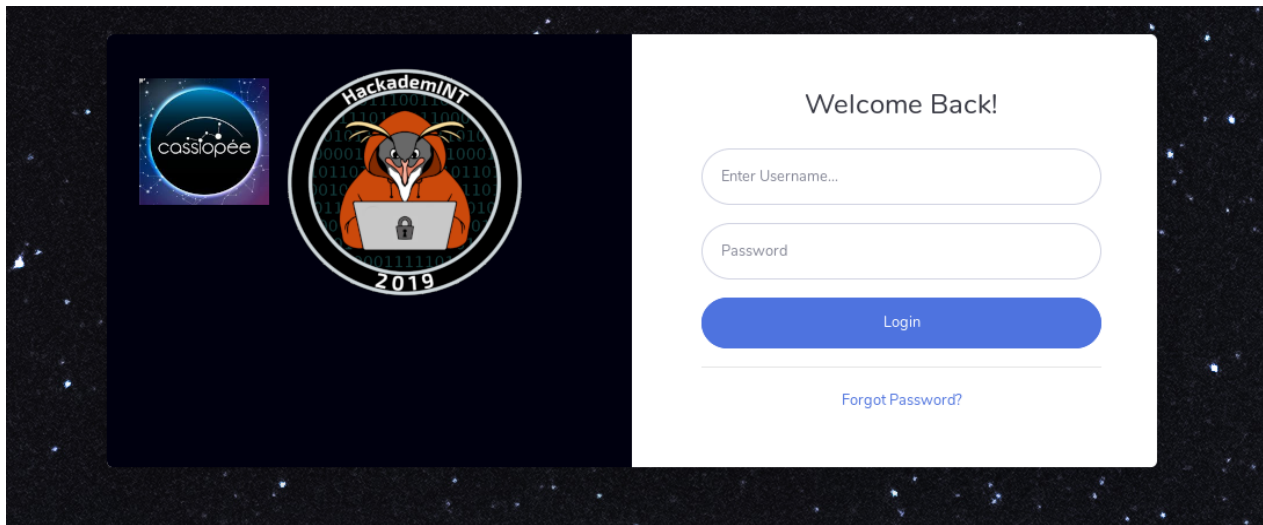## 3.1 Login View

url: `https://cassiopee.hackademint.org/login`

Figure 11: Screenshot of Login View

3.2 Admin Panel

url: `https://cassiopee.hackademint.org/admin/users/`

Figure 12: Screenshot of Admin Panel

This admin panel allows to manage the administrators of the platform

3.3   Forgot View

url: `https://cassiopee.hackademint.org/forgot`
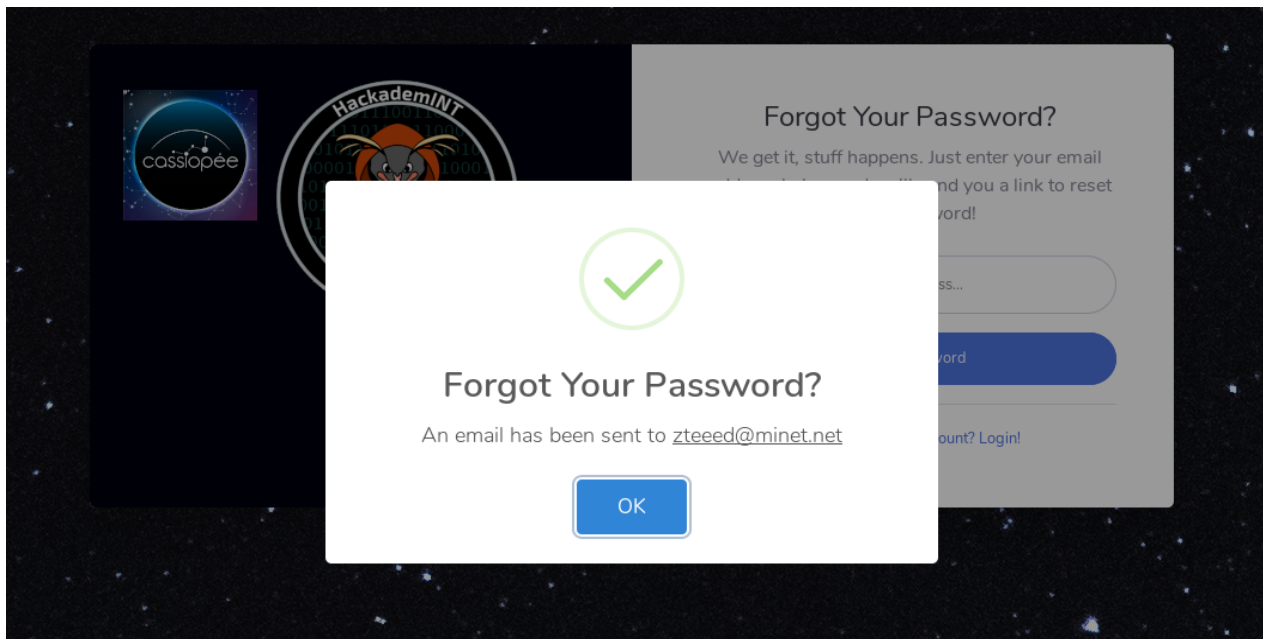
Figure 13: Screenshot of Forgot View

Figure 14: Screenshot of Email sent by Forgot View

3.4   Index View

Figure 15: Screenshot of Index View

3.5   CT/VM view

LXC 104: ns1 | Node: hackademint | CONFIG

JSON: {'dns': {}, 'data': {'hostname': 'ns1', 'arch': 'amd64', 'rootfs': 'stockage:104/vm-104-disk-0.raw,size=8G', 'ostype': 'debian', 'onboot': 1, 'swap': 512, 'digest': 'eab00f3b80fbfe0bc7a7af181305fd15bc193e4a', 'cores': 1, 'memory': 512}, 'interfaces': OrderedDict([('net0', {'type': 'veth', 'ip': '10.10.10.4/24', 'gw': '10.10.10.1', 'bridge': 'vmbr0', 'hwaddr': 'E2:20:9B:1D:D3:7A', 'name': 'eth0'}), ('net1', {'type': 'veth', 'ip': '10.20.17.4/24', 'gw': '10.20.17.1', 'bridge': 'vmbr17', 'hwaddr': '22:1B:33:AB:0F:AF', 'name': 'eth17'}), ('net2', {'type': 'veth', 'ip': '10.20.18.4/24', 'gw': '10.20.18.1', 'bridge': 'vmbr18', 'hwaddr': 'FE:92:65:28:A6:A2', 'name': 'eth18'}), ('net3', {'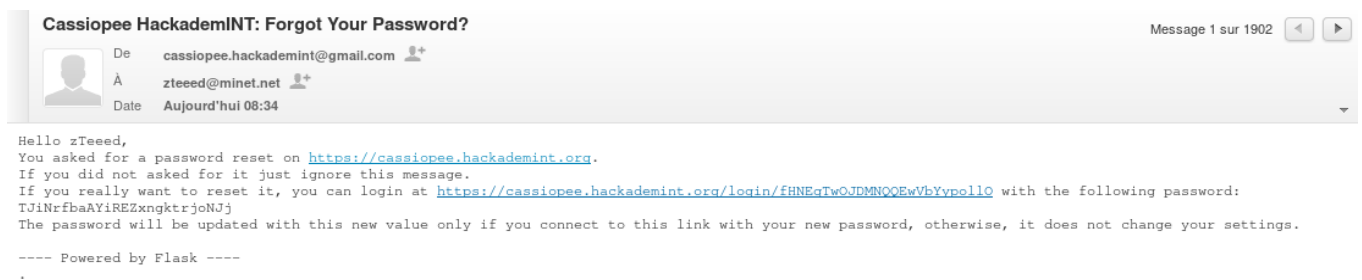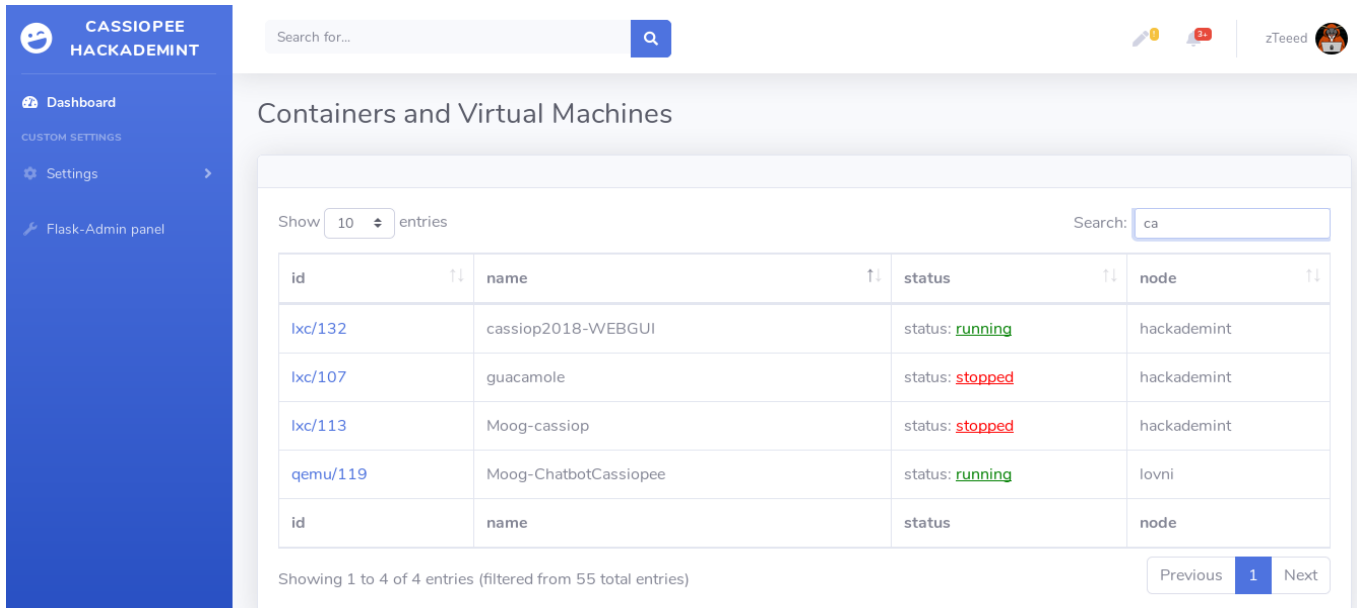type': 'veth', 'ip': '10.20.19.4/24', 'gw': '10.20.19.1', 'bridge': 'vmbr19', 'hwaddr': '76:89:34:F0:17:2B', 'name': 'eth19'}), ('net4', {'type': 'veth', 'ip': '10.0.33.4/16', 'gw': '10.0.33.1', 'bridge': 'vmbr0', 'hwaddr': '16:B8:C1:BA:91:59', 'name': 'eth33'}), ('net5', {'type': 'veth', 'ip': '172.16.0.4/24', 'gw': '172.16.0.3', 'bridge': 'vmbr120', 'hwaddr': '8E:C9:BB:4C:36:A6', 'name': 'eth42'})])}

Network interfaces

name	bridge	ip	gateway	hwaddr	type
eth0	vmbr0	10.10.10.4/24	10.10.10.1	E2:20:9B:1D:D3:7A	veth
eth17	vmbr17	10.20.17.4/24	10.20.17.1	22:1B:33:AB:0F:AF	veth
eth18	vmbr18	10.20.18.4/24	10.20.18.1	FE:92:65:28:A6:A2	veth
eth19	vmbr19	10.20.19.4/24	10.20.19.1	76:89:34:F0:17:2B	veth
eth33	vmbr0	10.0.33.4/16	10.0.33.1	16:B8:C1:BA:91:59	veth
eth42	vmbr120	172.16.0.4/24	172.16.0.3	8E:C9:BB:4C:36:A6	veth

Config data

ostype: debian	arch: amd64	rootfs: stockage:104/vm-104-disk-0.raw,size=8G
cores: 1	memory: 512G	swap: 512G

Figure 16: Screenshot of CT/VM View 1

LXC 104: ns1 | Node: hackademint | DATA

JSON: {'status': 'running', 'maxdisk': 8388009984, 'cpu': 0, 'maxcpu': 1, 'node': 'hackademint', 'vmid': 104, 'diskread': 203419648, 'uptime': 2812915, 'id': 'lxc/104', 'name': 'ns1', 'mem': 180355072, 'netin': 10166315643, 'diskwrite': 8118272, 'netout': 78880605, 'maxmem': 536870912, 'template': 0, 'disk': 1093820416, 'type': 'lxc'}

mem: 180355072	maxmem: 536870912	disk: 1093820416
diskread: 203419648	diskwrite: 8118272	maxdisk: 8388009984
cpu: 0	maxcpu: 1	uptime: 2812915
netin: 10166315643	netout: 78880605	status: running

Disk Usage — Styling for the donut chart can be found in the /static/assets/js/proxmox/chart-pie-disk.js file.

RAM Usage — Styling for the donut chart can be found in the /static/assets/js/proxmox/chart-pie-mem.js file.

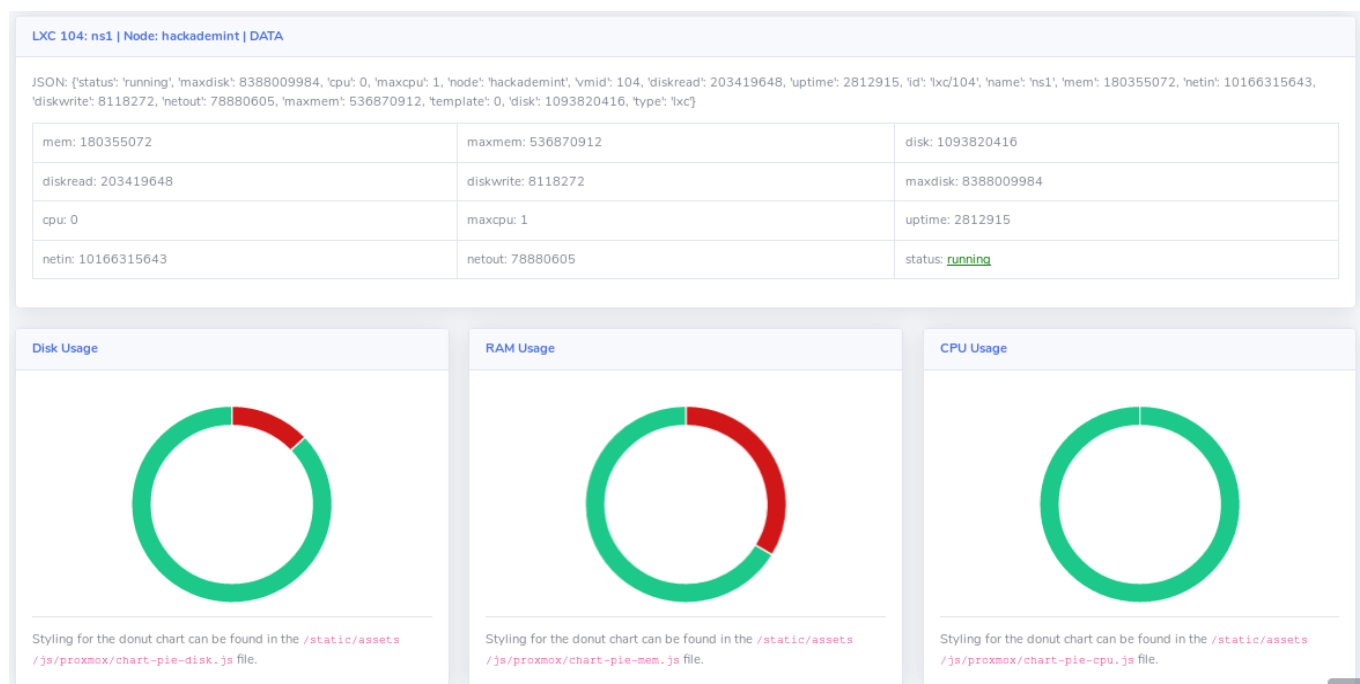CPU Usage — Styling for the donut chart can be found in the /static/assets/js/proxmox/chart-pie-cpu.js file.

Figure 17: Screenshot of CT/VM View 2

4 Conclusion and perspectives