Slovak University of Technology in Bratislava
Faculty of Informatics and Information Technologies

FIIT-XXXXXX-82385

Tomáš Belluš

Bait network based monitoring of malicious actors

Master's thesis

Supervisor: Ing. Tibor Csóka, PhD.

2021, January

Slovak University of Technology in Bratislava

Faculty of Informatics and Information Technologies

FIIT-XXXXXX-82385

Tomáš Belluš

Bait network based monitoring of malicious actors

Master's thesis

Study program: Information Security

Field of study: 9.2.4 Computer Engineering

Training workplace: Institute of Computer Engineering and Applied Informatics

Supervisor: Ing. Tibor Csóka, PhD.

Departmental Advisor: Ing. Katarína Jelemenská, PhD.

2021, January

Annotation

Slovak University of Technology Bratislava
FACULTY OF INFORMATICS AND INFORMATION TECHNOLOGIES
Degree Course: Information Security

Author: Tomáš Belluš
Master's Thesis: Bait network based monitoring of malicious actors
Supervisor: Ing. Tibor Csóka, PhD.
Departmental advisor: Ing. Katarína Jelemenská, PhD.
2021, January

Internal network, demilitarized zone (DMZ) or data pipelines have been compromised by a threat actor and the information gathered from this incident is minimal. Knowing the threat actor's agenda (entering, potentially leaving and fulfilling a goal) is more valuable, because it may lead to enforcing the system perimeter or endpoint security. By deliberately baiting access to highly monitored isolated networks, all further activities may be learned and enlighten a security engineer. This thesis, by utilizing state of the art container orchestration mechanism - Kubernetes, designs and implements a monitored isolated environment. Understanding and logging all file system changes, process executions helps to create a timeline of events constructing a possible incident. The resulting implementation is a robust proof of concept virtualized and completely automated immutable infrastructure monitored on the host machine level. Container observation mechanisms are capable, but not accurate enough to yet determine the order of file system events. Further design and implementation is required to identify the *point of enter* and *exit*.

Anotácia

Slovenská technická univerzita v Bratislave
FAKULTA INFORMATIKY A INFORMAČNÝCH TECHNOLÓGIÍ
Študijný program: Informačná bezpečnosť

Autor: Tomáš Belluš
Diplomová práca: Sledovanie zlovoľných činiteľov nástražným systémom
Vedúci diplomovej práce: Ing. Tibor Csóka, PhD.
Pedagogický vedúci: Ing. Katarína Jelemenská, PhD.
Január 2021

Interná sieť, demilitarizovaná zóna DMZ alebo zreťazené spracovanie údajov sú kompromitované útočníkom, a získané informácie o tomto incidente sú minimálne. Vedieť útočníkovú agendu (od vstup cez vykonanie agendy až po prípadný výstup) je velmi cenné, lebo to môže viesť k vylepšeniu informačnej bezpečnosti sieťového okruhu alebo cieľových staníc. Úmyselným lákaním útočníkov na prístup k vysoko sledovaným a izolovaným sieťam, poskytuje možnosť pre bezpečnostného experta sa poučiť zo zlovoľných aktivít. Diplomová práca, využitím vyspelého orchestračného mechanizmu kontajnerov - `Kubernetes`, sa venuje návrhu a implementácii sledovania izolovaného prostredia. Porozumenie a zaznamenávanie všetkým zmenám súboroého systému a vykonávania procesov napomáha vytváraniu časovej osi udalostí spájaných do prípadného incidentu. Výsledná implementácia je rozsiahlým dôkazom predstavy ako kompletne virtualizovaná a automatizovaná nemenná infraštruktúra monitorovaná na úrovni hostiteľa. Mechanizmy sledovania kontajnerov sú funkčné, ale nie sú dostatočne presné v udávaní poradia výskytu udalostí o zmene v súborovom systéme. Preto je nevyhnutný daľší návrh a implementácia aj pre konkrétne idetifikovanie *bodu vstupu* a *výstupu* zo systému.

Tu vložiť zadanie diplomovej práce

Potom, vložiť finálny návrh zadania diplomovej práce

Contents

Chapter 1

Analysis

This chapter introduces malware analysis (see section 1.1) and differentiates the techniques (see subsection 1.1.1), outlines and describes mechanisms and environments utilized by cyber security professionals and companies (see subsection 1.1.2), describes the virtualization technology KVM (see subsection 1.2.1) and the orchestration mechanism Kubernetes (see subsection 1.2.2).

1.1 Malware analysis

Monitoring a malicious actor, sample or an entity in any environment is interchangeable from the analyst point of view. Therefore, knowing the malware analysis techniques is crucial for secure monitoring of malicious actors. Malware analysis may be static and dynamic with varying tools and mechanisms [5] [3].

1.1.1 Dynamic and static

Dynamic analysis is a process of actively monitoring, ideally in an isolated environment, the execution of a malware sample. Static analysis, as the name implies, inspects the function calls, readable strings, control and data flow of a malware sample (i.e. binary, program code).

Dynamic analysis may be unsafe and devastating, unless environment or system isolation is applied, but it's more precise and bypasses the reversing of self-modified code. Even though, dynamic analysis does not explore all execution paths, there are techniques resolving this drawback (i.e. virtual machine snapshots [3]). Furthermore, dynamic analysis in a isolated or virtualization environment is exposed to the risk of [isolation] detection [2].

Static analysis is safe [and inspects all execution paths of the binary], but may be difficult to interpret when malicious actors utilize the obfuscation, compression or encryption techniques. Advanced malware sample poses a challenge for a malware analyst due the usage of control-flow flattening [10] and other methods. It suggests that dynamic malware analysis bypasses this troublesome procedure and discloses the sample outcome or agenda.

Manual Egele, et al. in their article [5] highlight the problems of static malware analysis approaches. The authors introduce multiple state-of-the-art techniques and tools dedicated to dynamic malware analysis - Function Call Monitoring, Function Parameter Analysis focusing of the final values, Information Flow Tracking, Instruction Trace and Auto start Extensibility Points (monitor startup programs, cron jobs, etc.).

1.1.2 Mechanisms [and environments and tools]

These mechanisms must provide solution to dynamic malware analysis limitations in order to effectively capture the malicious actor's agenda. It should be a robust system implying the impression of a production environment.

1.1.2.1 Honeypot and honeynet

Honeypot is a bait service, system or a even whole network (honeynet) usually hosted on public server. Its main purpose is to be scanned, attacked or compromised by the malicious actor. Every honeypot provides the desired functionality of the target resource, to mimic the production environment, leaving the malicious actor unaware of the honeypot [7].

Based on the ENISA honeypot study [7], honeypots are classified from the level of interaction view and based on the attacked resource type. The Low Interaction Honeypot (LIH) provides very low availability of the host OS. Most services and application are mocked and simulated in a static environment. Everything accessible is controlled by a decoy application with absolutely minimal in-depth features (e.g. shell, configuration files, other programs etc.). LIHs are more secure for the host, but far less capable or useful for malware/attack detection and inspection.

On the contrary, High Interaction Honeypot (HIH) is a fully responsive system with live applications and services with minimal to none emulated functionalities. It provides the attacker a wide attack surface ranking the HIH far less secure with the whole OS at the malicious actor's disposal. The idea is to make HIHs believable as possible and isolating it from production environment including virtualization [16].

Honeypots are differentiated by the type of attacked resource. The server-side honeypot is the well-known honeypot with running service(s) and monitoring the activity of the server-side connections. The attacked resources are the services listening on the dedicated ports. It's main purpose is to detect and identify botnets and forced authentication/authorization attempts.

The client-side honeypot is deployed as a user application, which utilizes the server's services. The monitored subject is the application (e.g web-browser, document editor). It's main purpose is to detect client-side attacks originating from the application (i.e. web-browser attacks via web pages and plugins).

1.1.2.2 Sandbox

1.2 Virtualization technology

1.2.1 Kernel-based Virtual Machine

1.2.2 Kubernetes

1.2.3 Docker

1.3 Environment monitoring

Knowing what and how to extract and monitor is vital to understanding the threat actor and the agenda. There are various mechanisms and possibilities to effectively observe container file system, networking and process execution. Regarding the expected setup of virtual machines hosting the Kubernetes cluster, the possible

monitoring strategies are:

- from the container in form of an agent reporting events
- from the node as agent-less programs "spying" on the containers
- bpftool, bcc tools

Container monitoring from the container itself may appear more straightforward, but it's similar to monitoring of a honeypot, where agents could be discovered or in other way uncovered by the threat actor. Since containers are wrapped environments hosted on the host machine, the file system, networking processes are readable. The apparent advantage is the transparency for the threat actor, since the container is observed from the hosting operating system. For example docker with the *overlayfs* driver has a predefined directory for each container with the root FS.

1.3.1 Existing solutions

`Monks` [1] works as a kernel module hijaking system calls on the target system. "Monks is like strace, but tracing all and every single process from any user, at any level" [14]. `execmon` [2] is a similar utility intercepting the *execve* syscalls in kernel and notifying the user. It's a kernel module combined with a user-land utility receiving events. For file system (FS) monitoring there is a CLI program `fswatch` [3] acting as the API to the *libfswatch*, which is ultimately an API to *inotify*. It provides real time FS monitoring with the accuracy of one second. Fswatch distinguishes events based on the action (e.g. created, renamed, removed, owner changed).

[1] `https://github.com/alexandernst/monks`
[2] `https://github.com/kfiros/execmon`
[3] `https://github.com/emcrisostomo/fswatch`

Chapter 2

Related Work

Honeypot, sandbox and deception technology make up the leading techniques in dynamic malware analysis. They differ in the scope of knowledge before the analysis at hand. Some know the filename, malware type, other artifacts and possible the expected outcome (in which case the tool observes the behavior). Other analyze the behavior of all activity - searching for anomalies and malicious behavior indicators. The following sections briefly introduce the malware analysis, deception technology, honeypot and other existing solutions and studies related to the thesis topic.

2.1 Malware file analysis solutions

Malware file analysis is a dynamic procedure when the filename and malware type is known. In comparison to the scope of this thesis, all use similar techniques of malicious activity observation/monitoring, but differ in use case scenarios.

2.1.1 Cuckoo sandbox

A most common sandbox environment for malware analysis by executing a given file in a sandboxed environment with reporting of the outcome. All files affecting mainstream operating systems i.e. Windows, MacOS, Linux and Android are supported. In addition to known artifacts, cuckoo has no interfering processes, so all traces must be followed and provide insight to the behavior. Based on the official cuckoo documentation, the system produces various results (the following artifacts are copied from the documentation site):

- Traces of calls performed by all processes spawned by the malware.
- Files being created, deleted and downloaded by the malware during its execution.
- Memory dumps of the malware processes.
- Network traffic trace in PCAP format.
- Screenshots taken during the execution of the malware.
- Full memory dumps of the machines.

Despite all differences, cuckoo's architecture consists of the management software (host machine) and a number of virtual/physical machines for analysis. It's a tool for different use case, so a comparison is insignificant.

2.1.2 Droidbox

Another open source tool, sadly discontinued several years ago, droidbox utilizes analyzes android applications using Android Virtual Devices (AVD) and the android emulator 4.1.1_rc6, which enables the android activity monitoring. Analyzed applications are sandboxed in the AVDs and afterwards reports the following results (the following artifacts are copied from the documentation site):

- Hashes for the analyzed package
- Incoming/outgoing network data
- File read and write operations
- Started services and loaded classes through DexClassLoader
- Information leaks via the network, file and SMS
- Circumvented permissions
- Cryptographic operations performed using Android API
- Listing broadcast receivers
- Sent SMS and phone calls

Droidbox introduces a simple way of analyzing android applications via an existing API of the emulator.

2.1.3 Virustotal

Similarly to cuckoo, virustotal utilizes both static and dynamic malware analysis. "VirusTotal's aggregated data is the output of many different antivirus engines, website scanners, file and URL analysis tools, and user contributions" [9].

2.1.4 Falcon sandbox

A direct concurrency to virustoal is the `Hybrid Analysis` [1] tool powered by the Falcon sandbox. Again, it's similar to cuckoo, except the anti-evasion feature [4], which allows, even sandbox-aware malware, to be analyzed despite their evasion techniques.

[1] `hybrid-analysis.com`

2.2 Active analysis

This section explores existing honeypot/honeynet technologies and a recently emerged concept - deception technologies. These technologies may be divided into two categories - dynamic [13] and static, where the environment adapt to the scenarios or remains unchanged respectively.

2.2.1 Honeystat

Honeystat [2] is a honeypot solution observing the behavior of the Blaster worm and may be used to detect zero day worm threats. The authors assume the infection may be described in a systematic way, so by knowing the worm agenda and steps they model the monitoring procedure. The observation is event-based with memory, disk and network events. Since there are no regular users in the system, the memory events are e.g. interesting violations as buffer overflows and other. Disk events are file system modifications and network events should always be infection related outgoing traffic. Worms require a multi-host network to have spreading possibility, so honeystat is deployed in a multihomed VMWare environment (64 VMs * 32 IP addresses = 2^{11} IP) with minimal honeypots.

The procedure when events are encountered is:

- The honeystat is capturing memory and disk events
- If a network event occurs, the honeypot is reset to stop further spread of the worm to other machines/honeypots.
- Any previous memory/disk event is updated with additional information from the network event.

[2] `https://people.engr.tamu.edu/guofei/paper/honeystat.pdf`

- Resets ought to be faster in virtual environment. Host VM is not rebooted, only the virtual disk (VD) is kept in a suspended state before it's replaced with a fresh copy of a VD. The reset always completes before a TCP timeout.
- Other steps include an analysis node, which is out of scope of this thesis.

This solution does not introduce any isolation techniques beside utilizing virtualization and the emulation mechanisms are exposing the virtualized environment via e.g. BIOS strings or MAC address. All features and considerations for honeystat are purely for worm infection detection, other infection types could require more observables.

2.2.2 Honeypoint

Service emulation is what Honeypoint [3] utilizes to lure malicious actors and detect their agenda. Production services lie in the same environment as the robust architecture of Honeypoint, which can mimic a complex network environment for deceiving an attacker. The Microsolved CEO Brent Huston claims [11] that having a honeypot is a great deception technology with almost no false positives, since it is expected that no legitimate user interacts with it. It means that any recorded activity should be considered suspicious, if the honeypot targets malicious actors scanning the Internet regardless of possible domain - randomly trying IP addresses and looking for a services ought to have malicious intent. Consists of various components [12] that could be replicated in the Kubernetes architecture design.

[3] `https://www.microsolved.com/honeypoint`

2.2.3 Cybertrap

A purely documented (commercial) solution Cybertrap [6] operates as a deception technology luring attackers away from production systems. Looking apart from that services in Honeystat are emulated, Cybertrap's deployed services cannot be distinguished by the attacker. Once the malicious actor gets inside such network, all his/her movements are tracked. In addition, the Cybertrap's network is inaccessible by regular users, so any activity within the simulated environment is consider malicious - minimal to none false positives. Cybertrap is close to the idea of the goal of this thesis - sandboxed honeynet.

2.2.4 A distributed platform of high interaction honeypots and experimental results

A case study [15] serving as a proof of concept in live Internet traffic observing malicious actors' trends and agenda. As a monitoring technique they patched the kernel's `tty` and `exec` modules to intercept the keystrokes and system calls respectively. The architecture is 4 machines anywhere in the world working as relays to the authors' local setup of VM honeypots. The traffic incoming to the public interface of the relay is routed to a GRE tunnel connected to the local VM.

In a SSH scenario the created a new syscall and modified the SSH server to use it in order to intercept the login credentials. Logged data is periodically copied from the VM disk to the host disk (such extractions should be undetected by the malicious actors). All login data is stored to the database of this structure:

- data from each ssh login attempt
- data from each successful ssh connection - tty buffer content and tty name

- data of programs executed with parameters and the terminal in which it ran
- session data grouping ssh connections

2.2.4.1 Experiment

- in the period of 30 days, they monitored what are the most common login-password pairs when no accounts are created
  - they found that for most attempts the login and password were the same
- then for almost half a year they monitored the time it took the attacker to successfully login and to login with commands entered
  - in some cases the attacker managed to get root via system vulnerable exploit
- they encountered attackers changing passwords of other accounts on the system
- they sorted their findings by country (mostly China, USA, Germany, UK, Russia, Romania, Japan, Brazil, France, South Korea and Netherlands)
- analyzed the intrusions and commands
  - mostly they tried to download programs from the same country the source IP originated from
- general trends of attacker behaviors:
  - check if i am alone on the system
  - system recon - OS name and version, processor characteristics
  - changes the password of current user
  - install an IP scan program and scans the IP range to recon for potential lateral movement
  - IRC client setup for receiving instructions
  - privilege escalation attempt

- general trends of attacker behaviors (with root):
  - change the root password
  - setup backdoor - open another port
  - checkout info about legitimate users of the computer via custom installed software
  - one attacker replaced the ssh client binary

2.2.5 SIPHON

A case study [8] on Scalable High-Interaction Physical Honeypots (SIPHON), similar to the study before, serving as a proof of concept in live Internet traffic observing the IOT related malicious intents. Leveraging Shodan to appear visible and legitimate in the eyes of malicious actors, the honeypots where based on real devices. The architecture is divided into physical IOT devices, wormholes exposed to the Internet forwarding to the IOT devices via the proxy forwarder. Technically are devices separated using VLANs 802.1Q and the wormhole to forwarder connection are via reverse ssh tunnels. As compromise countermeasures the suricata IPS and IDS features are enabled in the local netowork and periodic resets of IOT devices.

They observed the influence of device listing in Shodan. The number of scans/connection attempts on the device has tripled between 'one week before listing' and 'one week after listing'. It proves that being visible by Shodan increases the possibility of attack reconnaissance on device at hand. Although, after two week after listing in Shodan, the connection attempts has decreased, which good piece of knowledge before implementation.

2.3 Kuberentes specific solutions

2.3.1 alcide

Chapter 3

Design

This chapter focuses on the overall design of the solution, depicting the crucial parts. Firstly, the base system of the monitored environment. Secondly, container remotely controlled event-based monitoring of processes, file system changes and networking. Thirdly, deployment and emplacement of the isolated environment and additionally connecting it over network in a target facility or system.

3.1 Specification

This short section summarizes all specifications and assumptions considering the design. The target operating system is always Linux distribution Ubuntu 18.04.5+ with hardware enablement (HWE) or 20.04.x. The kernel specification is important for the machine hosting VMs to satisfy eBPF tools. In addition, the KVM (with QEMU) is used with the libvirt management library, which fully satisfies the choice of virtualization technology.

Regarding the VMs, there are no kernel specification, but they have Ubuntu

18.04.5+ or 20.04.x. Additionally, each VM has the minimum memory size of 4 GB and 2 virtual processors (vCPU). Although, this can vary depending of the host system resources.

3.2 Environment architecture

The environment has a solid underlying architecture considering a proper isolation layer and other prevention mechanisms to secure the hosting system. These operations and precautions are in compliance with the ISO 27002 standard. The bait environments within, are deliberately vulnerable in some way, therefore they are not designed to abide much of any of the ISO 27000 standards. The main goal is a mimicking production environment build atop of Kubernetes cluster. Sequentially, this section covers all in a bottom-up fashion through Kubernetes cluster design to system environments.

The considered specific control categories of the "Information technology – Security techniques – Code of practice for information security controls" - ISO 27002 are[1]:

12. Operations security
    1. Operational procedures and responsibilities
    2. Protection from malware
    3. Backup
    4. Logging and monitoring
    5. Control of operational software

[1] All of the outlined categories are derived from the official standard [1] with proper numbering preserved

13. Communications security
    1. Network security management
14. System acquisition, development and maintenance
    2. Security in development and support processes

Before implementation, the whole solution including program code, configurations and operation process must be documented with proper backups in terms of these control categories. According to ISO-27002 category 12.1 part "12.1.1 Documented operating procedures", the solution must have a technical documentation of the operations, which means including installation, setup and configuration procedures whether they are automated or not. This category links to ISO-27002 category 14.2, which discusses the security of version control systems and remote repositories used for sharing and archiving. All developed programs have a dedicated repository with no sensitive configurations, which could compromise the system. Last but not least, ISO-27002 category 12.3 refers to creating and maintaining proper backup procedures of valuable assets. Partly it correlates with the previous categories and only together function as secure, documented, archived and reproducible in case of any failure.

3.2.1 Base system

The lowest layer is an Ubuntu host machine with KVM virtual machines (VM). Deriving from a Kubernetes cluster design in production environment, which often isn't a single-node architecture, the base system must be a set of coexisting VMs. Choosing the right number of VMs with respect to the overall resource capacity is not in scope of this thesis. Nevertheless, there are three base VMs serving as the base of the deceiving system.

The main specification defines that all VMs are identical and configured for remote operations, have functional inter-VM communication and meet all requirements (e.g. kernel version, basic security settings, hardening). Creating multiple identical VMs can be achieved by preceding mechanisms (e.g. *cloud-init*[2]), sharing the same image or by provisioning mechanisms (e.g. *Vagrant*, *Ansible*).

It depends on the implementation, but since these VMs have a static and simple setup, preceding is not necessary. This technique is fully configurable and must be automated or manually ran for each VM with deviating variables e.g. host name, IP address. This is not a complicated process, but a more suitable approach is using a shared pre-configured image combined with a dedicated tool - Vagrant for seamless provisioning and installation. KVM with libvirt and Vagrant create a "VM as code" concept efficient in provisioning, preceding and whole VM management.

Setup

Base system has several dependencies and requires a custom setup of networking and host machine altogether. As mention in section 3.1, the host depends on QEMU, KVM and libvirt to run VMs. Additionally, the VMs are connected to a libvirt-managed management network (MGMT network) and a standard inter-VM network (NODE network) also simulating public IP address pool for the Kubernetes cluster. Both of these networks are represented as Linux network bridges and are segregated to comply with ISO-27002 category 13.1.

Given that, the deceiving environment is isolated twice (Kubernetes-managed containers and VMs), the host machine is not expected to experience any malicious activity. Even though, a set of precautions is advised in case of any suspicious

[2]https://github.com/canonical/cloud-init

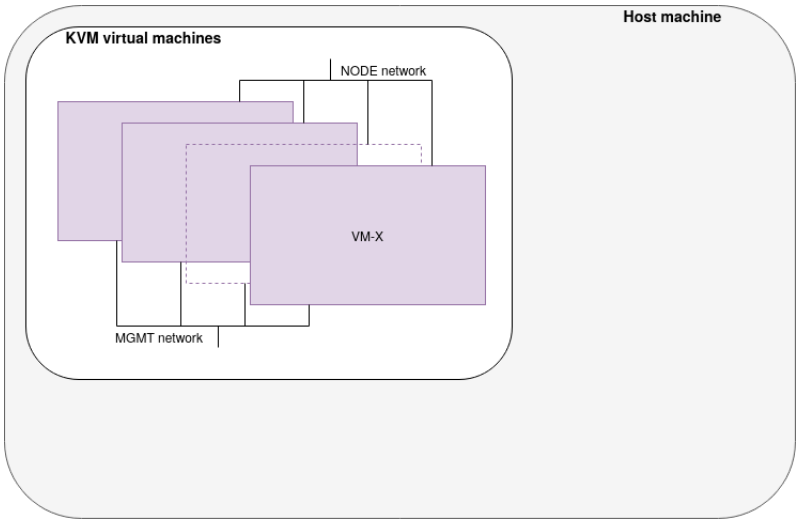

Figure 3.1: Base system visualization. !!!!REVIEW ME!!!!

activity occurs otherwise. But only passive techniques are suitable, because all of activity from the isolated environment is routed through the host system, which must be allowed. Network security monitoring (NSM) solution designed for detection rather than prevention should effectively provide the sufficient visibility over the host system. Although, this is not the main thesis objective, so there are no specific requirements and it depends on the implementation.

Vagrant

Sharing a custom base image (Vagrant box), saves time on configuration and is less prone to error. Utilizing a pre-configured Vagrant box does not necessarily mean the configuration is immutable, all can be changed via Vagrant post-deploy commands and provisioning in the *Vagrantfile*. Security-wise, creating a custom Vagrant box from the official Ubuntu image is recommended over publicly available Vagrant boxes from unverified owners. Not knowing the whole agenda of those boxes a full audit would be appropriate to approve their usage. Therefore, the all

VMs have been created with a specific Vagrant box.

Each Vagrant box is a minimal Ubuntu (of satisfying version defined in section 3.1) installation with the following configuration and setup:

- user setup including password protected *root*
- kernel parameters
  - enabled IP forwarding - `net.ipv4.ip_forward`
- VM routing table entry to satisfy reverse path check - new route through the NODE network to the administration network or machine
- SSH daemon configuration
- custom dependencies based on the implementation

Additionally everything else is done within the Vagrantfile, which is configurable with environment variables or other type of arguments. The input variables are the NODE network bridge name, IP prefix for the NODE network, number of nodes and vagrant box identifier. Altogether, the Vagrantfile creates all requested nodes as functional and remotely accessible VMs ready for Kubernetes installation and the deception environment configuration.

3.2.2 Kubernetes cluster

Kubernetes is complex and highly configurable, therefore a simple configuration is sufficient. There are many Kubernetes installation techniques and various derivatives meant for minimal setup and development (e.g. *microk8s*[3], *minikube*[4], *k3s*[5]).

[3] `https://microk8s.io/`
[4] `https://minikube.sigs.k8s.io/docs/`
[5] `https://k3s.io/`

For this thesis a full Kubernetes ecosystem is preferred to mimic a production environment as much as possible.

Setup

Considering Kubernetes as a cloud-only platform, there are few differences to take into account when deploying microservices and applications on a bare-metal variant. Most importantly, Kubernetes is installed on those three base VMs in an arrangement of one master node with two worker nodes. Which briefly means, that the master node controls the cluster and does not provide any computational resources like the worker nodes do.

Things like storage and load balancing network traffic, are cloud provider services, which need to be substituted. Regarding persistent storage for demanding applications, the cluster is scaled up with additional dedicated node (data node) for storage. Data node is not part of the Kubernetes cluster, it serves as an external Network File System (NFS) server providing persistent storage. This node is setup in the same way as the other cluster nodes, except for the Kubernetes installation. Instead, setup with simple NFS server.

The Kubernetes ecosystem is installed via *kubespray*[6], which is a full Ansible skeleton for a complete cluster setup. Kubespray installs all dependencies, configures networking and assigns roles (master vs. worker) to all nodes. The cluster is ready for creating new objects, applications and environments.

Only after the cluster is fully functional, the before mentioned load balancing can be setup. For a bare-metal installation, the *MetalLB*[7] load balancer effective and provides full Kubernetes loadbalancer capabilities. MetalLB is setup in layer 2

[6] `https://kubespray.io/`
[7] `https://metallb.org/`

mode, which refers to the data link layer of the RM OSI model. This mode utilizes the address discovery protocol (i.e. ARP) to route between nodes and turns one node into a point of enter. It is not a true load balancing technique, but it serves the key purpose of publishing common ports to the outside network.

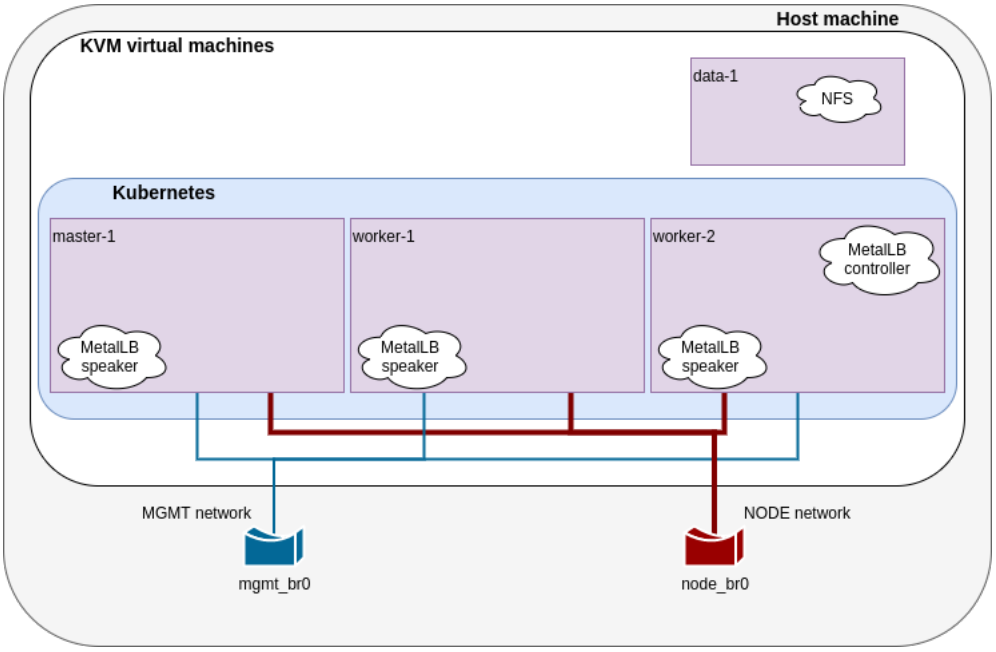

Figure 3.2: Desired system setup with an environment ready Kubernetes.

Environments

Each environment covers a whole system with multiple applications and subsystems. In terms of Kubernetes, the environment is a collection of compatible objects/resources. Some with simple functions and dedicated applications (e.g. FTP server, private Docker registry, Gitea/GitLab, mail server or Nextcloud) others with connected active processes or dependent applications (e.g. data pipelines or microservice ecosystem).

Technically, an environment in terms of this thesis is a collection of YAML files

stored in a remote repository easily deployed to the cluster. More specifically, Kubernetes *pods*, *services*, configuration (via e.g. *configmaps*, *secrets*), mounted volumes and/or permission-related objects. Practically, pods are replaced with *deployments* and published services are using service type *LoadBalancer* (utilizes MetalLB).

3.3 Container monitoring

Monitoring a container requires to observe container file system, networking and process execution in the most efficient way. Inspired by some related solutions, this section describes the monitoring mechanisms, tools and techniques utilized in this thesis.

- Points of enter, such as honeypots that lure the threat actors to the environment. Could be local to the environment or remote anywhere in the Internet.

- 3 Ubuntu server nodes are the base to Kubernetes cluster holding and orchestrating the whole environment.

- There are to be multiple environments.

- Any environment is automated and deployed to the cluster via Ansible playbooks.

- The core monitoring tools and programs are deployed on the hosting nodes

3.3.1 Activity events

The traced events, which are used to identify the malicious actor's agenda are file system, network and process related. Each type of event has a different repetition period and importance. For example, network and process events are also self-occurring (with no user initialization activity), therefore not all are held as valuable as file system changes. Files are changed less frequently and selectively choosing valuable directories, the monitoring agents gain visibility over the desired parts of the file system. Nevertheless, the down side to all types of these events are those interpreted as false positive.

According to ISO-27002 category 12.4 part "12.4.4 Clock synchronisation" "the clocks of all relevant information processing systems within an organization or security domain should be synchronised to a single reference time source" [1]. All events are tagged with a timestamp to correctly create a timeline of all events together. Since the designed system is wrapped by VMs, which share the host machine, the time is identical each VM.

File system events

Any modifications or removal of existing and creation of new files in the specified locations of the file system are detected and logged. Additional events possible to monitor, are file permissions, owners, access lists and renaming. Since, "Everything is a file" mostly applies, there are expected to be many uninteresting events.

Process execution events

It depends on the implementation, but process events are any new instances that are readable in the output of "`ps waxu`" command. Including process and par-

ent process IDs (PID, PPID resp.), user, command line including arguments and optionally memory and CPU usage creates an image of observed activities.

Network events

Malicious actors use network to exfiltrate data, send commands or connect to the target service/system. Malicious programs propagate and mutate through network. For all these scenarios a different protocol or technique may be used. Although, in Kubernetes pod/container network services must be published over Kubernetes service objects, therefore setting up a backdoor listing on arbitrary port requires access to *kubectl* or a vulnerability in Kubernetes itself. Meaning only existing services are of interest in addition to general exfiltration network protocols e.g. FTP, SMTP, HTTP/S, DNS, SMB according to the MITRE technique - "Exfiltration Over Alternative Protocol" [0].

3.3.2 Tools and techniques

Whether it is monitoring from the host machine or directly from Kubernetes, a list of target containers is vital. In case of monitoring from within the containers, an agent/daemon is predefined as part of the pod in its object definition. On the other hand, for a host-based monitoring some kind of reconnaissance must occur to identify all targets in an automated way. The designed tools are described in depth in the following sections.

Host-based monitoring

All events are detectable, to some extend, from the host. Since all activity originates from a Docker container, by design the file system, processes and network

traffic is accessible.

Firstly, dynamically getting all the active containers is crucial for inspecting any of the discussed events. Using the control tool *kubectl* get the cluster nodes, which host the pods. Combining *kubectl* and *docker-inspect*, it is simple to retrieve all containers and additional metadata. Altogether the designed tool maps all application containers to nodes for further processing.

Docker container metadata provide all the necessary information to locate the container root directory and mounted file systems. Using a standardized technique, these directories or selective subdirectories may be actively monitored for changes. More specifically, '`.[0].GraphDriver.Data.MergedDir`'[8] gives the top level file system of the *overlayfs* and '`.[0].Mounts[].Source`'[8] returns the mounted directories located on the host. This tool extension executes file system monitoring for each container utilizing tools like *fswatch*, *inotify* or *opensnoop*.

Detecting executed processes is a more sofisticated issue, but using the right technique is important. eBPF enables the tool *execsnoop* to tap into the kernel and record system calls. Fortunately, execsnoop allows various filtering options. Specifically the mount namespace efficiently isolates the observable system calls within a single container. For execsnoop the mount namespace identifier is shared over the BPF map objects using *bpftool*, which are available for a latest kernel version requirement mentioned in section 3.1.

Last of the event types - network traffic monitoring is much more intuitive and native to the architecture and technology used. Since the applications within the cluster are virtualized, the pod network interfaces can be traced to host accessible interfaces. Consequently, a tool taps on to the interface and effectively scans the whole traffic including a pod-to-pod and pod-to-external communications. This

[8] jq filter

leaves open possibilities to tools like *DPDK* or netflow collectors like *suricata*, Although the design expects many pods with many interfaces, so the network traffic collector should have a minimal overhead.

Control tool

The main control (*API server*) tool is for managing and controlling all of the above mentioned tools and functionality. It is a centralized API server, which has access to the execsnoop, fswatch and nettool (**TODO change**) for basic management. Additionally, API server provides additional threat hunting functionality such as:

- running process information
- file information **MUST IMPLEMENT**
- **TODO other**

Kubernetes monitoring

The minimal monitoring of the whole cluster is an ideal job for the *Prometheus* toolkit combined with *Grafana* dashboards for visualization. Prometheus provides visibility of the golden metrics[9].

Apart from the network and pod monitoring, other events are not being tracked from within the cluster.

[9] "Golden metrics (or "golden signals") are the top-line metrics that you need to have an idea of whether your application is up and running as expected."[0]

3.4 Deployment and emplacement

This section focuses on the design of deployment methods and ideal emplacement of the system. The focus is mainly on *Ansible*, *Vagrant*, *kubespray* and *kubectl*. The main goal is an Ansible skeleton with playbooks and roles and applicable division by hosts in custom inventory(s).

3.4.1 Deployment

The prerequisite are cluster nodes, which are deployed and pre-configured using Vagrant as discussed in section 3.2.1. While the Vagrantfile defines cluster nodes, Ansible executes scripts, additional configuration and provisioning of Kubernetes and installation of tools' dependencies.

Following the Ansible design patterns, any compact deployment is packed in an Ansible playbook or Ansible role. section 3.2.2 already addresses kubespray as the Kubernetes deployment mechanism, which is not integrated to the main Ansible skeleton.

Apart from the Kubernetes deployment all required setups (both Kubernetes-related and not) and tools are provisioned by playbooks or roles. Each is fully configurable by Ansible group and host variables, which define a specific environment setup and can be cloned to configure an environment with different properties.

3.4.2 Emplacement

An abstract visualization in Figure 3.3 shows the bait system behind a firewall next to a organization's production environment. This one of possible emplacements of

the system in a network.

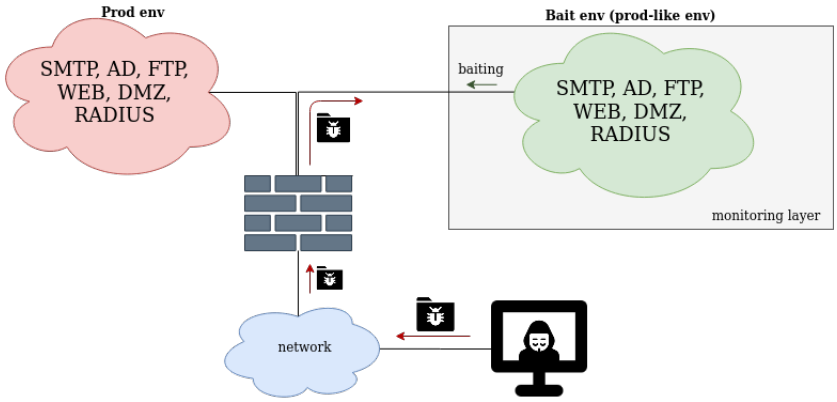

Figure 3.3: Abstract idea of this thesis system emplacement.

Chapter 4

Implementation

- Manage base setup of Kubernetes nodes via Vagrant.
- Deployment of those Vagrant based virtual machines (including networking setup) and Kubernetes (via kubespray [1]) is done by Ansible.
- Kubernetes environments are maintained [2] separately.
- Sneakpeek [3] properly parses the docker container information. It groups nodes with user defined containers created by the Kubernetes orchestration from deployments and pods. Ultimately used for getting the container root FS directory.
- Fswatch [4] is a fork with simple improvements and features. It's ran on every acquired container root directory.
- All is automated with Ansible.

- network traffic is monitored by tapping to the mapped interface on the host system 1. get the docker container pid 2. us nsenter to execute 'ip addr' to get the

[1] https://kubespray.io/
[2] https://github.com/tomas321/k8s-environment-scenarios
[3] https://github.com/tomas321/sneakpeek
[4] https://github.com/tomas321/fswatch

main net interface 3. from that get the mapped interface 4. tap on that interface with tcpdump or some other software

Literature

[1] ISO/IEC 27002:2013. *Information technology – Security techniques – Code of practice for information security controls*. Standard. Geneva, CH: International Organization for Standardization, 2013. URL: https://www.iso.org/standard/62711.html.

[0] Alfredo Abarca, ed. *Exfiltration Over Alternative Protocol*. Version 1.2. The MITRE Corporation. Mar. 28, 2020. URL: https://attack.mitre.org/techniques/T1048/ (visited on 05/29/2021).

[2] Aditya Anand. "Malware Analysis 101 - Sandboxing. Cons of using a VM". Sept. 29, 2019. URL: https://medium.com/bugbountywriteup/malware-analysis-101-sandboxing-746a06432334 (visited on 03/27/2020).

[3] Ujaliben Kalpesh Bavishi et al. "Malware Analysis". In: *ijarcsse* (2017). URL: https://ijarcsse.com/index.php/ijarcsse/article/view/507 (visited on 03/26/2020).

[4] "Defense in Depth: Detonation Technologies". Mar. 12, 2018. URL: https://inquest.net/blog/2018/03/12/defense-in-depth-detonation-technologies (visited on 10/11/2020).

[5] Manuel Egele et al. "A Survey on Automated Dynamic Malware-Analysis Techniques and Tools". In: *ACM Computing Surveys. Article 6* 44.2 (Feb. 8, 2012). Article 6, pp. 5–21. DOI: https://dl.acm.org/doi/10.1145/

2089125.2089126. URL: https://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.640.6356&rep=rep1&type=pdf (visited on 03/26/2020).

[6] "ENDPOINT DECEPTION". URL: https://cybertrap.com/en/solutions/#endpointdeception (visited on 10/05/2020).

[7] Tomasz Grudziecki et al. "Proactive Detection of Security Incidents - Honeypots". enisa study. Nov. 2012. URL: https://www.enisa.europa.eu/publications/proactive-detection-of-security-incidents-II-honeypots.

[8] Juan Guarnizo et al. "SIPHON: Towards Scalable High-Interaction Physical Honeypots". In: (Jan. 12, 2017). URL: https://arxiv.org/pdf/1701.02446.pdf (visited on 09/25/2020).

[9] "How it works. Many contributors". Mar. 23, 2020. URL: https://support.virustotal.com/hc/en-us/articles/115002126889-How-it-works (visited on 10/11/2020).

[10] Vladislav Hrčka. "Stantinko's new cryptominer features unique obfuscation techniques". Mar. 19, 2020. URL: https://www.welivesecurity.com/2020/03/19/stantinko-new-cryptominer-unique-obfuscation-techniques/ (visited on 03/23/2020).

[11] Brent Huston. *State Of Security Episode 15*. Apr. 5, 2019. URL: https://www.podbean.com/media/share/pb-cgwhv-ad161e?utm_campaign=w_share_ep\&utm_medium=dlink\&utm_source=w_share (visited on 10/05/2020).

[12] Brent Huston. "What is this HoneyPoint Thing Anyway?" Dec. 6, 2012. URL: https://stateofsecurity.com/what-is-this-honeypoint-thing-anyway/ (visited on 10/05/2020).

[0] Risha Mars. *Kubernetes observability with a service mesh. What are golden metrics and why are they important?* URL: https://buoyant.io/2021/01/11/kubernetes-monitoring-with-a-service-mesh/.

[13] Yifat Mor. "Using Dynamic Honeypot Cyber Security: What Do I Need to Know?" Oct. 10, 2018. URL: https://www.guardicore.com/2018/10/dynamic-honeypot-cyber-security/ (visited on 10/11/2020).

[14] Alexander Nestorov. "What is Monks". Feb. 24, 2015. URL: https://github.com/alexandernst/monks (visited on 12/04/2020).

[15] Ivan Studnia et al. "A distributed plateform of high interaction honeypots and experimental results (extended version)". In: (June 10, 2012). DOI: https://hal.archives-ouvertes.fr/hal-00706333. URL: https://www.researchgate.net/publication/262277761_A_distributed_platform_of_high_interaction_honeypots_and_experimental_results (visited on 09/26/2020).

[16] Lenny Zeltser. "5 Steps to Building a Malware Analysis Toolkit Using Free Tools". SANS Institute instructor. Mar. 2015. URL: https://zeltser.com/build-malware-analysis-toolkit/#allocate-virtual-systems.