Hounterfeit: A Virtual Self-Defending Infrastructure with Transparent Relocation to Honeypots

Mihai-Alexandru Bogatu Adrian-Răzvan Deaconescu Cătălin-Adrian Leordeanu
Department of Computer Science
POLITEHNICA București
București, Romania
e-mail: mihai.bogatu@stud.acs.upb.ro {razvan.deaconescu | catalin.leordeanu}@upb.ro

***Abstract*—Advanced Persistent Threats (APTs) pose the most sophisticated cyber-attacks, some of which reside in the internal networks over an extended period of time. Intrusion Detection and Prevention Systems (IDS / IPS) strive to keep up with the newest attacks; however, they are often updated only after a 0-day causes impact to businesses. As APTs continue to evolve, we propose Hounterfeit - a self-defending infrastructure that deceives attackers into revealing their payloads on production-looking Honeypot systems. The architecture makes use of Software-Defined Networking (SDNs) alongside process migration through Checkpoint/Restore In Userspace (CRIU) to achieve a transparent relocation to a honeypot environment, with low network overhead, while also maintaining scalability. Once common attack indicators are detected, the malicious actor is transferred transparently alongside its application and network session, to a replica of the server that masquerades sensitive data. The infrastructure can be used as a base to deceive attackers to expose attack methods tailored for the real live systems.**

***Keywords*-SDN; CRIU; IPS; Honeypots.**

I. Introduction

The attacker-defender ecosystem in cybersecurity is constantly evolving, attackers finding new ways to break into systems while defenders having to keep up with them. While traditionally most exploits were centered around a single application or service, nowadays defenders have to take into consideration complex interactions, such as supply-chain attacks. Immediately blocking attackers in a deep packet inspection scenario once malicious activity is detected limits the amount of damage done to the systems. However, it also reveals to the attackers possible security filters inside the systems. A proactive approach to model threats is deployment of honeypot systems. Arguably implementing honeypots helps defenders reveal attackers' payloads and methodologies, however such systems are usually implemented at companies offering services or products in cybersecurity. The honeypot systems deployed may not have any technology present on a specific's company system that needs protection. Revealing payloads to other types of applications would have less value for said organizations. Finding a way to capture malicious payloads in an environment resembling the real production system, this paper proposes an infrastructure that creates honeypots from the base production applications, which with the attackers establish communication, while preserving other system data confidentiality, integrity and availability.

Hounterfeit implements a *Software-Defined Networking* (SDN) infrastructure that scans packets for malicious activity. Once attacks are detected, the traffic between the hosting server and the malicious user is dropped, a process with already established network connection with the attacker is cloned and transferred between the host and the Honeypot while the virtual firewall connection is redirected to the new system. This method achieves transparent Transmission Control Protocol (TCP) redirection but also preserves any in-process state that the attacker made until that point. This solution preserves sufficient environment such that attacks can continue on a Honeypot system, prolonged interaction with such system allowing for better analysis of the threat.

In this paper, we make the following contributions:

- we propose the Hounterfeit self-defending infrastructure;
- we implement a replicable build for testing process relocation in honeypot scenarios;
- we evaluate the impact of the infrastructure by measuring the process migration time via a secure communication.

The rest of the paper is structured as follows: Section II presents the bases of an infrastructure capable of preventing attacks and learning from them; Section III mentions state-of-the-art research related to Hounterfeit's design; Section IV details the logical design and proposed technologies to integrate with Hounterfeit; Section V notes the testing procedure and resulting metrics; Section VI provides takeaways for the experiment while proposing extended usage in closed-loop scenario, while also listing some limitations of the approach; and Section VII summarizes the article and the possible directions for Hounterfeit's extended development.

II. Background

One of the major adversary to the security of information technology systems are *Advanced Persistent Threats* (APTs). An APT is usually composed of a group of individuals who posses both financial and technological resources, with deep understanding of security systems. Major motives for APTs are financial and/or political, any may be sponsored for cyber-attacks. The term persistent, comes from the fact that APTs will, most of the times, break into the systems stealthily, move through the infrastructure and may reside inside organization's systems for a prolonged period of time. Initial foothold in the infrastructure may be gained through the use of 0-days, vulnerabilities that have not yet been disclosed publicly. Due to the complexity of such attacks, and the potential impact

of a breach, it is imperative to understand the tactics of such threats, preferably without the risk of any major damage.

To add resilience to any system, isolating components at risk is essential. An isolated environment is often called a sandbox. From a host's perspective there are 3 common isolation methods that provide resilience without the need of application modification: emulation, virtual machines and containers. Emulation refers to an architecture in which the hardware is modeled through software. While emulation provides most control, it is usually not implemented in publicly-facing production environments due to high resource consumption. *Virtual Machines* (VMs) abstract some of the hardware and allow multiple *Operating Systems* (OS) to run at the same time while restricting some operations that are managed by a *Virtual Machine Manager* (VMM) and which require hypervisor privileges. VMs are typically used in production environments as they share the same hardware, with some restrictions, providing fast access while isolating the systems. Containers are implemented on top of an OS, leveraging security features in the Operating System to achieve isolation such as namespaces, control groups, capabilities, etc. Traditionally, containers share the same kernel, which provides a faster access to resources than VMs but reduced isolation. The mentioned methods for resilience remain secure as long as the implementation is correct and isolation is not broken, which otherwise may result in a sandbox escape.

An infrastructure that is able to defend itself against attacks is designed to autonomously detect, prevent and respond to cybersecurity threats. It is commonly implemented with network and/or host *Intrusion Prevention Systems* (IPS), which may contain *Deep Packet Inspection* (DPI) capabilities. The detection at network level can be implemented via plaintext filters such as tokens or regular expressions, behavior analysis or machine learning. The rules are usually constructed based on real attacks, if a major incident happened and network/application logs are recoverable or after a security advisory is released, which is prone to delays in analysis, publication and finally rule implementation. Protection against 0-days is limited to the degree of similarity with previous known attacks, such as having common attack kill chain patterns or payloads.

A. Honeypot Systems

A proactive approach to learn new attack techniques and payloads is to set up a dedicated system constructed specifically to be exploited, to collect as much information as possible, known as honeypots for individual hosts/services or honeynets for multiple inter-connected hosts. Figure 1 presents a minimal example of a self-defending architecture that contains autonomous honeypot systems. All traffic towards the servers will be inspected through an IPS. If malicious activity is detected, the traffic is dropped and the attacker may be blacklisted. In other deployments that a regular user will not otherwise find by legitimate actions, the honeypots are unprotected and left as a distraction and possibly for payload capture. Sometimes, honeypots are deployed behind a firewall, and the scope of

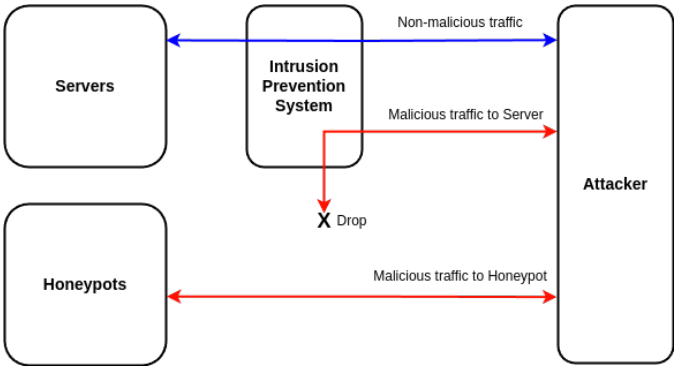

Figure 1. Self-defending infrastructure with honeypots.

deployment is attacker identification for blacklisting, requiring some sort of IDS. Exploitation of the honeypot systems is intended and some virtually created vulnerabilities may be introduced to provoke malicious actors engagement.

Honeypot systems can be classified by the interaction level they provide. *Low-Interaction Honeypots* (LIH) implement a minimal system. They are the safest to deploy and maintain, however they are not very realistic, as an advanced attacker can make the difference between a real system and a Low-Interaction Honeypot. They are a mechanism to capture malicious entities such as malware or other automated scanning and exploitation tools rather than manual attacks. Usually, Low-Interaction Honeypots do not have the sense of maintaining sessions and persistent actions such as uploading files, modifying a database value or keeping an internal application state. *High-interaction Honeypots* (HIH) give the most realistic sandboxing environment. They allow cybersecurity experts to monitor and trace complex attacks and develop new defense mechanisms by observing the behavior of malicious entities or actors. The drawback is that the system might contain vulnerabilities that could be exploited and then leveraged further to attack the internal systems by pivoting on the compromised server. As such, a high-interaction Honeypot must be properly isolated from the rest of the network and the abstracted from physical machine it is deployed on, by using sandboxing techniques. Another approach to create honeypot systems is to migrate attacked services into an isolated segment and observe attack interactions. While this poses a high risk as real systems are under exposure, carefully selecting which part of the system will be migrated can mitigate potential sensitive information leaks and allow a high-interaction honeypot system with a high fidelity to the production environment.

B. Reactive Migration Process

While having ready decoy machines to deceive attacks away from the real systems can be helpful, wanting to capture payloads directed to the real systems without revealing current defense mechanisms can pose some challenges. An approach inspired by current desktop / mobile antivirus software is to run suspicious client code in a sandbox environment, quarantine the app if malicious activity is detected and send a sample

to dedicated environments for further analysis, as it can be re-run. For server-based applications, stopping an attack early will reveal little information about threats that could be used defensively. An approach is to redirect the traffic dynamically to honeypot systems. One problem that might arise is redirection fingerprinting. For attackers to be migrated transparently, network stack protocols must be corrected appropriately, such as in the case of TCP communication, sequence and acknowledgement numbers must synchronized with the new honeypot communication.

While a transparent redirection solves the problem of traffic fingerprinting, which otherwise may stop some automated attacks early, in the case of manual attacks or attacks that make use of application-level sessions, there still is the problem of app fingerprinting. By having the same application in an isolated environment, specifically deployed without sensitive production data, or otherwise specifically crafted decoy data, can solve the fingerprinting problem and will be more likely to capture payloads crafted for the production. To preserve application-level sessions, a solution that can be applied without modifying the application is based on a live migration of the process the attacker was communicating with in the production environment to the isolated honeypot environment. This migration must be a reaction to attack indices such as fuzzing detection, mass vulnerability scans or known payloads detection. This reactive migration process must be accompanied by a network redirection as well. Such migration can be aided by utilizing a programmable network, as in the case of Software Defined Networks.

Migrating systems or system components is a relevant topic for infrastructures that implement Quality of Service (QoS). While used to ensure high availability, live migration methods can be used in cybersecurity systems as well, as detailed in the next section. VM live migration is often achieved via taking a snapshot of the machine and resuming it on a new physical system. A faster method of VM migration is detailed in the next section. On the other hand, pausing and resuming a container, such as in the case of Docker, could be done via freezer cgroup [1], or via checkpoint restore with *Checkpoint/Restore In Userspace* (CRIU) [2]. Furthermore, CRIU could be used standalone to checkpoint processes, or process trees alongside relevant information such as established TCP communication or other system handles.

III. Related Work

While a lot of research focuses on redirecting attackers from low-interaction level honeypots to high-interaction level honeypots based on the persistence of the attacks [3]–[7], some research focuses on redirecting attackers from the real production systems to honeypots [8][9]. Furthermore, some research focus is put on the transparent migration process as well, by utilizing Software Defined Networks infrastructure. While some research proposes constructing new TCP handles by simulating network traffic and recalculating synchronization numbers at the switch level [5] or at the controller level [6], other approaches [8] propose a mix of SDN and operating system functions alongside auxiliary proxy systems to reconstruct TCP sessions.

INTERCEPT+ [10] created a reactive VM deployment using QEMU in kernel-based virtualization. To migrate attackers, the infrastructure makes a full clone of the Virtual Machine (VM) hosting the production environment, while handling the traffic redirection with an SDN approach. To achieve better performance, the authors used Copy-on-Write (CoW) mechanisms for the deployment of the Honeypot system. The article suggests to use a separate database for the Honeypot, to avoid data leaks. Since the systems are perfect copies, at the moment of deployment, the TCP connection states are preserved between the Honeypot and the initial VM, thus achieving true transparent redirection, after the network traffic is routed to the honeypot port. While the VM approach handles the environment isolation issue, by making complete clones of the environment may inevitably leak production data present in the cloned VM. An improvement to the security of the infrastructure, which may limit the information disclosed, would be to use a small VM images and dissect the work between them. In this situation, if an attack is detected, a minimal portion of the infrastructure should be cloned. This approach can make use of micro-VMs approaches such as Unikraft [11].

Sandnet [9] uses live-cloning of containers to achieve a confined network segment used as a Honeynet, by mirroring parts of the production segment. The paper defined the Quality of Deception (QoD) as a metric to express the similarities between the environments. The infrastructure will clone sufficient containers to give the attackers an attractive bait while limiting the fingerprinting between the real and honeypot infrastructure by using a mathematical formula to express QoD. This approach is most likely to keep attackers hooked to the sandbox infrastructure, defenders gaining an advantage on information about vulnerable assets. The infrastructure relies on the use of microservices for optimal defense functioning. The infrastructure is capable of transferring attackers with a from 0.5s to a maximum of 3s overhead per microservice. The infrastructure tested the speed of process migration with CRIU vs docker checkpoint for process tree cloning. The results show that CRIU had a faster migration.

WARP [12] is an acronym used for a self-defending Kubernetes architecture. While it did not aim to facilitate a honeypot systems, it is an interesting case of asset protection. The infrastructure would migrate attackers before they can perform a malicious action on other assets. It maintained a lower false-positive detection rate by continuously predicting the attack model and next steps. WARP utilizes ML models on the security logs with associated MITRE ATT&CK tags, by predicting the probability of the next attacker action. WARP compared 2 migration delays: a) Docker-based migration by moving the pod after it is closed and b) CRIU based inside the pod at run SDN IPS with Process Migration to Honeypots time. The results show that live CRIU-based migration is faster, showing migration delays of 0.4s to 3.1s. While CRIU is identified to be faster than a container pause and resume,

similar to [9], it comes at the cost of privileges. To migrate one or more processes rather than full containers special capabilities must be added for the containers, which may allow persistent attackers to escape the isolation.

Mfhoney [7] is an infrastructure centered around honeypots. The infrastructure uses CRIU to switch between low- and high-interaction honeypots by profiling them and having a ready-deployment template that would be changed for the specific attacker. By changing the network state of a CRIU image dump, mfhoney could migrate attackers between processes that reside inside a single machine. This allows changing the fidelity level of the honeypots based on the interaction level of the attackers. It achieves that by monitoring in-system interactions such as syscalls, to schedule a live-migration to a higher interaction level honeypot. Having pre-built image dumps, the authors observed delays of interactions with the applications after a process restoration with TCP-handshake rewrite around 210ms for Hypertext Transfer Protocol (HTTP) services and 100ms for HTTPS services. The honeypot process switching via CRIU is done on a single local machine.

While Hounterfeit is similar to [9][10][12], instead of migrating live containers, the infrastructure selects the exact processes attackers communicate with. Furthermore, similar to [10], the infrastructure benefits of higher virtualization isolation for the cloning of processes between VMs, while furthermore allowing migrations multiple times by using a process monitoring tool to extract unused migrated processes. With its internal design, Hounterfeit also benefits from scalability, allowing multiple attackers to be redirected to the same environment or multiple environments while preserving application states.

IV. Implementation

The infrastructure is broken down into components that could be virtualized independently. Figure 2 shows the minimum required components. While having the honeypot migration system at all times is ideal to capture and deceive attacks, the infrastructure design is to be used on production environments rather than research laboratories. A Migrator service is decoupled from the network control to prioritize infrastructure uptime over security research. In the case of malfunctioning or overloading, the infrastructure should fallback into a simple IPS mode, blocking communication with identified attackers. Software-defined network paradigm of planes is used to separate the infrastructure. The data plane is used for normal network traffic while the control plane for network control. In this situation, a Migrator service is placed on the same plane as the SDN Controller. The Migrator's job is to trigger CRIU dump and restore commands and facilitate secure transfer of process dumps. A connection between the Migrator and the SDN Controller is required for work synchronization and process identification.

The server and honeypot contain the same build template, including Media Access Control (MAC) address and Internet Protocol (IP) address allocation, while being differentiable by the SDN switch port they are virtually connected to. They contain a predefined number of services already deployed,

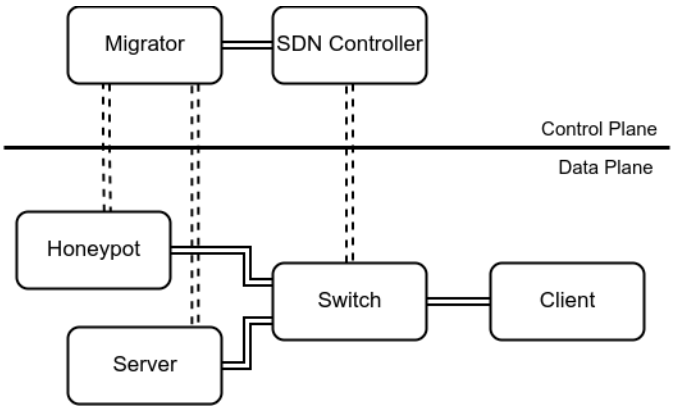

Figure 2. Minimal Hounterfeit infrastructure.

split into multiple processes for work balance. The work allocation and client connectivity is handled by the operating system. The honeypot requires a clean-up service to remove processes, that have been migrated into the system, which no longer contain active connections. In this situation, the operating system handles load balancing, as such once a session terminates or expires and its underling process is killed, an attacker can still make a new connection with the pool of pre-spawned processes inside the honeypot system. This approach will prevent resource exhaustion, while allowing transparent communication migration.

To allow extension of the infrastructure, the SDN switch and SDN controller are set to apply reverse Network Address Translation (NAT) for the honeypot/server. The SDN switch configuration is set to send all packets that do not match internal OpenFlow table rules to the controller. The controller will take the role as an Intrusion Prevention System, and also will take the responsibility to migrate the traffic to a honeypot. To take the load off the controller, once an attacker is detected and transparently redirected, further communication will be handled directly by the OpenFlow switch. Further analysis of payloads after the initial detection should be done by an auxiliary system, that could be set to monitor the traffic. The switch will have two roles: to forward packets for analysis in the communication line of the server-client; and to apply OpenFlow rules for detected malicious actors. The table rules for an identified attacker will contain: a rule to drop packets from the server destined to the attacker; and 2 rules to apply reverse NAT between the attacker and the honeypot, for each direction.

A. Detection Procedure

The infrastructure is meant to redirect attacks to honeypot systems. Taking this into consideration, it should aim to provide almost the same security level as a system that would only block attacks rather than migrate them to an isolated system.

The detection procedure will be broken down into two separated cases: ingress traffic and egress traffic. Figure 3 shows the communication flow when traffic is received and a response is sent from the servers.

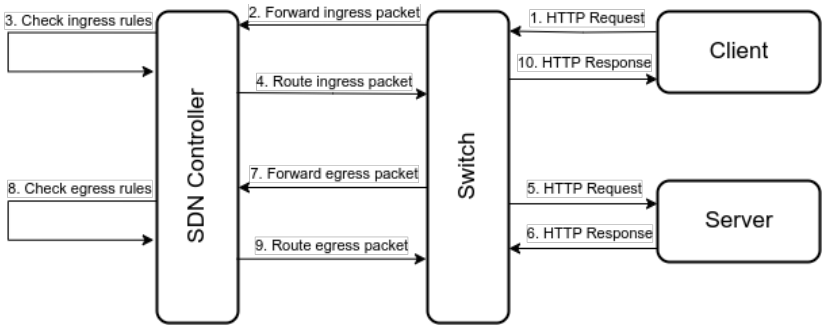

Figure 3. Normal communication flow.

- Ingress traffic is scanned against common payloads and attack techniques. Such payloads can include application specific or network specific malicious requests, as an example for web applications: *Cross Site-Scripting* (XSS) attacks, *Server Side Request Forgery* (SSRF), Command Injection, etc.
- Egress traffic is scanned against information disclosure. Some common sensitive information it can scan for is password hashes (provided the format used in database/applications is known), specific parts of system files (ex: /etc/hosts, private keys, /etc/shadow), *Application Programming Interface* (API) keys, etc.

To detect if a host is compromised and it is used for attackers to move laterally, some ingress detection rules for payloads could be scanned for egress traffic as well.

Detecting an attacker on the egress case though, instead of migrating the attacker alongside the process itself, the egress traffic is dropped. In this situation, if migration was done successfully, the attacker would still divulge information from the production environment.

After the detection procedure, a mutex is acquired and further traffic before the full migration will be ignored for the identified attacker. Typically in a normal scenario packets will be retransmitted by applications or the operating system when a connection is non-responsive, especially in the case of reliable TCP connections. After the migration is done and rules are installed in the OpenFlow switch, further packets will be directly handled in the data plane by the switch and will not reach the SDN controller and will not need further management. Figure 4 shows the process flow in three timelines: T1: normal traffic (such as TCP handshakes); T2: initial payload where the transparent migration process takes place; T3: further traffic after initial payload. Unlike a deployed honeypot visible to the internet, Hounterfeit hides the honeypot before the SDN Switch. It will route traffic to the honeypot system only if attacks are detected towards the production environment. By not completely dropping the traffic, unlike standalone IPS, Hounterfeit will redirect the entire communication to the sandbox environment.

B. Tools and Technologies

Checkpoint/Restore in Userspace (CRIU) is used for process migration in the current scenario. The tool supports reconstruction of live TCP connections, however requiring higher privileges and capabilities for this operation. While exposing additional capabilities in a VM will not break virtualization, container-based approaches use capabilities for environment isolation, making it unsuitable for a secure deployment, as the additional capabilities may allow container escape attacks.

Selected technologies for scenario simulation are HTTP servers composed of Nginx, Nodejs and Flask. While migration is not a problem in a new environment without conflicting processes, in this scenario the same process may be cloned multiple times. Furthermore, the honeypot itself must contain some pre-spawned processes to provide multiple simultaneous connections similar to a production environment. Taking the constraints into consideration, Linux systems offer a socket feature supported by Nginx as well, which makes the infrastructure reusable: `SO_REUSEPORT`. The flag on a listening socket allows other processes spawned by the same user to listen to the same port, provided they have the flag set as well. The load balancing in this situation is handled directly by the operating system. While not apparent, this has great implications in the simplicity of the infrastructure. If without that flag, a master process is used to load-balance communication for a multi-processing host, in the case of migrating the attacker communication and session to an other server, the worker process must be migrated, but the master process will have listening port conflict with an existing one on the target migration system. Even if migrating only the worker process, a new connection between the worker and new master must be established, which may require software modifications. By using software that naively supports `SO_REUSEPORT`, the problem of master synchronization can be avoided, as the services can work standalone. While Flaks and Nodejs do offer direct support to the flag, the call to the socket function was intercepted with `LD_PRELOAD` mechanism and the mentioned socket option was added. In a migration, only the socket must be reconstructed, and with it, the communication with the initial attacker will be re-established. At the moment of writing, CRIU technology uses TCP_REPAIR to migrate active TCP connections, by restoring them.

Another issue that may arise in a migration scenario is *Process Identifier* (PID) reuse. The production environment may contain multiple worker processes and usually are long-

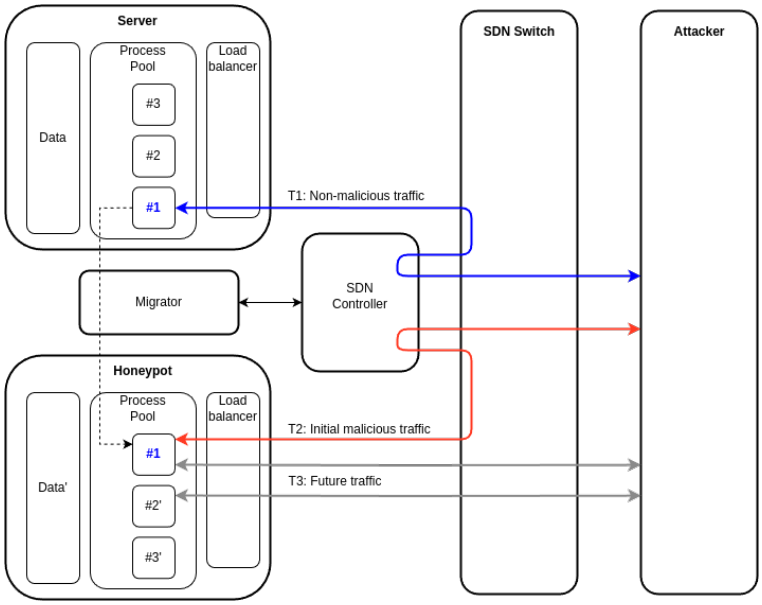

Figure 4. Attack flow on Hounterfeit.

lived. If a process is migrated to a system that already has an other process with the same PID as the migrated process, the CRIU migration will halt. Modification of the image dump can be made to assign a random unused PID at migration. However, to ensure that the honeypot system is not overloaded, a process must collect unused workers.

For process transfer, *Secure Copy Protocol* (SCP) 3-way communication was used. For minimum overhead, all *Secure Shell* (SSH) communication is multiplexed with a control file. This way for any command (CRIU dump, SCP transfer, etc.) the communication will be reused, not requiring a full TCP and *Transport Layer Security* (TLS) handshake each time. The initial handshake is established before any attack. This approach improved the migration time considerably, as the SSH handshake would sometimes take over 500ms.

V. Attack Scenario

A simple HTTP server was chosen for a simulation of the infrastructure. For real production environments, multiple backends and databases might be in use. To protect data, which processes are migrated should be carefully chosen, to avoid migrating sensitive information that might be cached in memory. For the scenario, an attacker would send legitimate traffic, which will reach the server and eventually will send a payload which will trigger the migration process. During this time all his communication is paused, the controller installs flows in the switch to drop server to attacker communication, sends source IP and port to the Migrator service. The Migrator service then identifies the process the attacker was communicating with, makes a CRIU dump of the process, transfers it over a 3-way SCP to the honeypot, then restores the process. Afterwards the Migrator notifies the SDN controller that the action is completed. The SDN controller installs new flows in the switch to NAT the traffic between the attacker and the honeypot.

The infrastructure was tested over 100 times per technology in the following scenario: controller is reset and OpenFlow switch tables are emptied, and 2 HTTP server processes are initiated both on the server and on the honeypot for each technology. An attacker sens a malicious payload as the first HTTP request.

A garbage collector was implemented to remove migrated processes from the Honeypot after they would terminate all TCP connections. The hardware used for testing consists of an *Intel Core i5-4590 @ 4x 3.7GHz* with 16 GB RAM. The base operating system runs on *x86_64 Linux 6.8.0-47-generic* with *Ubuntu 22.04 jammy* while the VMs run on the latest Ubuntu image at that time from Vagrant repository of *x86_64 Linux 5.4.0-193-generic* on *Ubuntu 20.04.6 LTS*. The following resources were allocated for the actors: 0.5 CPU cores with 1 GB RAM for each of server/honeypot/attacker and 2 CPU cores with 1 GB RAM for the controller hosting a Open vSwitch (OVS) as well. The Migrator service ran directly on the host machine. HTTP server packages verions contain *nginx/1.18.0 (Ubuntu)*, *Flask/3.0.3 (Python/3.8.10)* and *node/10.19.0*, while *CRIU v3.18* was compiled on both the server and the honeypot. Both the server and the honeypot were built separately.

Table I results show the mean time and standard deviation (σ) for the migration tests. Aligned with other research works that use CRIU technology [7][9][12], the infrastructure contains similar benchmark results on process migration time (considering the process transfer time as well for some of the papers).

A further approach to compress the Nginx process dump before being sent over SCP, taking into consideration the compression/decompression overhead, did not show improvements for the transfer time in the current lab scenario. However, the network links in the lab scenario are virtualized, with the default

TABLE I. MEAN AND STANDARD DEVIATION OF STEP PER TECHNOLOGY, MEASURED IN MS.

Service	Criu dump	SCP transfer	Criu Restore	Total
Nginx	39.57 ǀ 2.16(σ)	55.82 ǀ 13.19(σ)	13.47 ǀ 0.22(σ)	108.87 ǀ 13.39(σ)
Node	52.40 ǀ 13.62(σ)	263.72 ǀ 20.92(σ)	13.63 ǀ 0.93(σ)	329.76 ǀ 23.01(σ)
Flask	66.70 ǀ 27.53(σ)	442.40 ǀ 32.78(σ)	14.40 ǀ 3.31(σ)	523.50 ǀ 38.65(σ)

VirtualBox/Open vSwitch bandwidth, which realistically may be less performant if physical separation of the hosts is in place, depending on the hardware in place.

VI. FURTHER DISCUSSION

While having an infrastructure capable of migrating attacks to a system without sensitive information wastes attackers time, making use of the information they provide such as network and system traces for developing new threat prevention rules is the desired outcome of Hounterfeit's design. A fully self-defending infrastructure will require inputting the extracted information from the honeypot system back into the network detection module and closing the loop, as shown in Figure 5. Note that the IPS rules with the Detection Module can be replaced by a machine learning algorithms, as having this loop makes this natural for continuous learning. For this to work, intelligent seasons must be implemented that would recognize potential payloads from usual traffic. Furthermore, a CI/CD system should check new rules extracted from an attack against test-cases such that the normal functioning of the infrastructure would not be affected. Additionally, each new rule will further slow down the normal traffic, as the controller is designed in IPS mode as a man-in-the-middle system. A way to compress and optimize and integrate new rules should be implemented. The optimization system should not make a new detection rule easy to bypass.

Furthermore, the infrastructure can benefit from, and contribute to *Threat Intelligence Platforms* (TIPs) by sending new *Tactics Techniques and Procedures* (TTPs) attackers adopt, since it has the capability to capture attacks and redirect them to a system suitable for tracking and monitoring malicious activity. Such systems also allow for tracking personalized payloads while not affecting the sensitive data and services.

While the infrastructure works well for single timed attacks, for multiple attackers the migration mutex may be visible, by measuring the time between traffic. Furthermore, during the CRIU the production system may be paused temporary. This may be a big concern if attackers use multiple machines to trigger migrations. However, such attacks will require a high number of IPs. Redirecting an IP to the honeypot may also cause problems for legitimate clients in the case of NAT-based Internet Service Providers (ISP). The quantum of time an attack is sent to the honeypot should be carefully chosen. The SDN controller should also apply some packet application-layer buffering for the detection rules to prevent attack evasion via packet fragmentation.

On the other hand, in the case of false positive detection, users will be redirected to the Honeypot, which may expose them to an environment that attackers may operate on. While

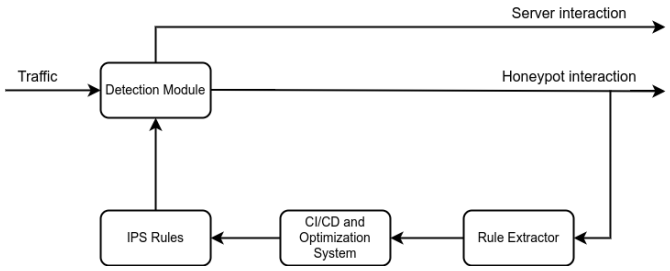

Figure 5. Closed IPS loop.

this does not affect static interactions with the services, in case of data insertions, modifications or deletions, the users will not be able to replicate them in the real environment nor receive updated entries from other entities, which can be considered a *Denial of Service* (DoS) for the migrated legitimate users. Sharing the environment with the attackers, may also leak user information and activity. To combat such problems, the detection system should be audited for potential false positive detection rules.

Furthermore, since the infrastructure already supports process dumping in the honeypot, periodic images of the process could be taken alongside network traffic to better understand and able to reproduce attacks. Certain triggers such as a reverse TCP connection or specific system calls can trigger the logging mechanism to permanently save the attack process and network snapshot and for further analysis. This approach will provide a better snapshot of attack interaction any may allow full replay of the attack in some scenarios.

VII. CONCLUSION AND FUTURE WORK

In this paper, we present Hounterfeit: based on network and operating system function virtualization, it provides an insight into a self-defending infrastructure that is able to migrate live attacks from real production system into a honeypot environment.

The scenarios measured results suggests that the migration time of the process between separated virtual machines is adequate, allowing redirection to the honeypot before standard application sessions expire. For simple production technologies, the infrastructure design may be used in real systems after some adaptations.

Future work on Hounterfeit will focus on process dependencies, adding a framework to support complex interactions between the OS and other *Interprocess Communication* (IPC) requirements. Furthermore, a framework will be proposed for real time sensitive data detection and counterfeiting for transferred processes that depend on in-memory or at-rest generated data.

REFERENCES

[1] Docker, *Docker pause command*, Accessed: Mar. 04, 2025. [Online]. Available: https://docs.docker.com/reference/cli/docker/container/pause/.

[2] CRIU, *Checkpoint/restore in userspace*, Accessed: Mar. 04, 2025. [Online]. Available: https://criu.org/.

[3] H. Wang and B. Wu, "Sdn-based hybrid honeypot for attack capture", in *2019 IEEE 3rd Information Technology, Networking, Electronic and Automation Control Conference (ITNEC)*, 2019, pp. 1602–1606. DOI: 10.1109/ITNEC.2019.8729425.

[4] S. Kyung *et al.*, "Honeyproxy: Design and implementation of next-generation honeynet via sdn", in *2017 IEEE Conference on Communications and Network Security (CNS)*, 2017, pp. 1–9. DOI: 10.1109/CNS.2017.8228653.

[5] W. Fan and D. Fernandez, "A novel sdn based stealthy tcp connection handover mechanism for hybrid honeypot systems", in *2017 IEEE Conference on Network Softwarization (NetSoft)*, 2017, pp. 1–9. DOI: 10.1109/NETSOFT.2017.8004194.

[6] W. Fan, Z. Du, M. Smith-Creasey, and D. Fernández, "Honeydoc: An efficient honeypot architecture enabling all-round design", *IEEE Journal on Selected Areas in Communications*, vol. 37, no. 3, pp. 683–697, 2019. DOI: 10.1109/JSAC.2019.2894307.

[7] J. C. Acosta, "Locally-hosted fidelity-adaptive honeypots with connection-preserving capabilities", in *MILCOM 2022 - 2022 IEEE Military Communications Conference (MILCOM)*, 2022, pp. 154–159. DOI: 10.1109/MILCOM55135.2022.10017548.

[8] V. A. Cunha, D. Corujo, J. P. Barraca, and R. L. Aguiar, "Using linux tcp connection repair for mid-session endpoint handover: A security enhancement use-case", in *2020 IEEE Conference on Network Function Virtualization and Software Defined Networks (NFV-SDN)*, 2020, pp. 174–180. DOI: 10.1109/NFV-SDN50289.2020.9289898.

[9] A. Osman *et al.*, "Sandnet: Towards high quality of deception in container-based microservice architectures", in *ICC 2019 - 2019 IEEE International Conference on Communications (ICC)*, 2019, pp. 1–7. DOI: 10.1109/ICC.2019.8761171.

[10] A. Hirata, D. Miyamoto, M. Nakayama, and H. Esaki, "Intercept+: Sdn support for live migration-based honeypots", in *2015 4th International Workshop on Building Analysis Datasets and Gathering Experience Returns for Security (BADGERS)*, 2015, pp. 16–24. DOI: 10.1109/BADGERS.2015.013.

[11] S. Kuenzer *et al.*, "Unikraft: Fast, specialized unikernels the easy way", in *Proceedings of the Sixteenth European Conference on Computer Systems*, ser. EuroSys '21, Online Event, United Kingdom: Association for Computing Machinery, 2021, pp. 376–394, ISBN: 9781450383349. DOI: 10.1145/3447786.3456248. [Online]. Available: https://doi.org/10.1145/3447786.3456248.

[12] S. Bagheri *et al.*, "Warping the defence timeline: Non-disruptive proactive attack mitigation for kubernetes clusters", in *ICC 2023 - IEEE International Conference on Communications*, 2023, pp. 777–782. DOI: 10.1109/ICC45041.2023.10278632.