Sandnet: Towards High Quality of Deception in Container-based Microservice Architectures

Amr Osman*, Pascal Brückner*, Hani Salah*, Frank H. P. Fitzek*, Thorsten Strufe*, Mathias Fischer†
*TU Dresden, Germany
firstname.lastname@tu-dresden.de
†University of Hamburg, Germany
mfischer@informatik.uni-hamburg.de

***Abstract*—Responding to network security incidents requires interference with ongoing attacks to restore the security of services running on production systems. This approach prevents damage, but drastically impedes the collection of threat intelligence and the analysis of vulnerabilities, exploits, and attack strategies. We propose the live confinement of suspicious microservices into a sandbox network that allows to monitor and analyze ongoing attacks under quarantine and that retains an image of the vulnerable and open production network. A successful sandboxing requires that it happens completely transparent to and cannot be detected by an attacker. Therefore, we introduce a novel metric to measure the Quality of Deception (QoD) and use it to evaluate three proposed network deception mechanisms. Our evaluation results indicate that in our evaluation scenario in best case, an optimal QoD is achieved. In worst case, only a small downtime of approx. 3s per microservice (MS) occurs and thus a momentary drop in QoD to 70.26% before it converges back to optimum as the quarantined services are restored.**

***Index Terms*—Network Deception, Decoy Networks, Network Security, Live Network Sandboxing, Honeypots, Honeynets.**

I. Introduction

As cyber attacks continue to evolve in their sophistication and stealthiness, and evasion techniques, traditional countermeasures, e.g., honeypots and Intrusion Detection Systems (IDS), have become no longer sufficient [1]. Many IDS rely on well-known signatures, and behaviors [2]. Thus, these systems cannot detect unknown and stealthy attacks as well as internal adversaries performing lateral movement, i.e., who leverage internal services to pivot through the network and obtain unauthorized access to other services.

Current honeypots can be easily fingerprinted according to protocol signatures, i.e., header fields such as user agent strings, fake credentials, enclosed fake data, as well as their pre-programmed dummy behavior [3], [4]. Additionally, Key Performance Indicators (KPIs) such as system load and network throughput indirectly translate to its importance from an adversarial standpoint, and could alarm the adversary about a honeypot presence. Advanced Persistent Threats (APTs) that attack multiple services and systems over longer time spans therefore pose a challenge to traditional honeypots.

This work takes a novel approach to tackle the deception problem. Our approach uses the live network as deception network, and combines the state of the art in Software Defined Networking (SDN), and live cloning of microservices via check-pointing and restore (C/R) to confine adversarial network presence to its own network micro-segment that we coin "*Sandnet*" without endangering the live production network. In particular, we make the following contributions: (1) the design and concept of Sandnet's architecture, (2) the implementation and evaluation of three new strategies to perform live network deception: actively, reactively and proactively, (3) a metric that we coin the *Quality of Deception* (QoD) to quantify the effectiveness of network deception given various KPIs that could be observed by the adversary to distinguish whether it has been confined, (4) a thorough investigation of the feasibility and QoD of Sandnet using different microservices (MSes) as macro-benchmarks.

The remainder of this paper is structured as follows: Section II defines our assumptions, system architecture and adversary model. Next, Section III describes our methodology and three proposed strategies for live cloning and network deception along with the QoD metric. Section IV describes our evaluation. Section V reviews the related work. Finally, we conclude our work in Section VI.

II. Sandnet's concept

A. System design

Our system design is shown in Figure 1. It is designed to protect a network which is under the control of one administrative domain. We refer to this as the *Production Network* (PN). Within the PN, a number of interconnected MSes exist. We assume this topology is connected on layer 2 through a highly-available SDN switch mesh. For simplicity, we represent this as only one switch in the PN and the *Sandbox Network* (SN) respectively. When the IDS detects an intrusion and/or a Suspicious Container (SC), it will notify the controller. The controller will then sandbox network connections from and to the SC.

We assume a deployment of applications following the microservices pattern, where the overall application is decomposed into container-based uni-purpose units which communicate over a network-based API such as Remote Procedure Calls (RPC) or other TCP/IP based protocols such as HTTP Representational State Transfer (REST). A single process is confined using a single Linux container (i.e. Kernel-level isolation between processes). Contrary to Virtual Machines (VMs; i.e. Hypervisor-level isolation), containers enjoy a

978-1-5386-8088-9/19/$31.00 ©2019 IEEE

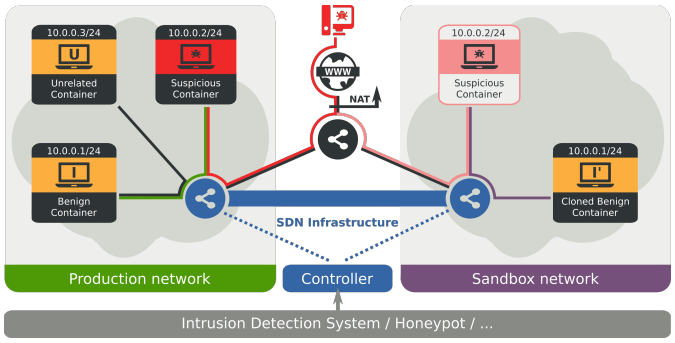

Figure 1. Live network deception concept

lower resource footprint [5], and usually contain less state. By relying on the MSes model, we can reap these benefits and additionally apply our network sandboxing mechanism to a finer level of granularity on the service-level instead of the VM-level.

Our key objective is to mirror the PN into an isolated solitary confinement network, i.e., SN, and re-route internal and external connections involving the SC to the SN without disrupting IP reachability and existing stateful Layer 4 connections such as TCP. This entails that the adversary is completely isolated from the PN and may only cause damage within the boundaries of the SN.

B. Adversary model

We limit the threat model to a single "pivot" adversary. That is an external adversary who becomes internal by compromising a MS with the intent to explore the network seeking to infect or control other MSes. It is capable of scanning the network, conducting reconnaissance, performing traffic analysis to fingerprint or counter the network sandbox, actively crafting malicious packets to forged network destinations, and connecting to containers within its reach as well as leverage remote exploits in them. The adversary is stealthy and aims to remain undetected. Hence, it behaves like the MS it attacks which allows it to evade internal firewalls by masquerading as a benign container.

We consider the following scenarios out of scope and leave them for future work. First, we assume that the adversary cannot break out of the container-sandbox and we rely on the security model followed by Linux containers. Second, we do not consider Denial of Service attacks on Sandnet itself where the adversary can trigger multiple concurrent cloning pipelines leading to performance degradation, or resource exhaustion. Third, we do not consider adversaries with multiple entry points to the PN, nor the presence of multiple adversaries at the same time.

III. Live Cloning and Network Deception

When a container on the PN is suspected as malicious, a fresh clone of the malicious container is spawned from a clean state, and all benign connections are redirected to it. After that, we weaponize our network sandboxing methodology in order to clone other containers within reach of the SC to SN and then confine the adversary's connections.

Two key concepts lie at the heart of our approach: A network-oriented live *cloning strategy* for containers and a dynamic *network confinement* mechanism. Both must operate within the framework of a *deception life cycle*. This section highlights the aforementioned three aspects of our methodology then introduces our QoD metric.

A. Deception life cycle

Sandnet relies on a 3-phase live deception life cycle: An *initiation* phase, an *expansion* phase, and a *shrinking* phase. The first defines when SN is created. The second defines how the SN grows as the adversary expands its reach or presence. The third defines if the cloned containers could be put in a sleeping state, or removed from the SN and when. This work focuses on the *initiation* and the *expansion* phases only and leaves *shrinking* for future work.

B. Cloning strategy

To employ Sandnet, we introduce and distinguish between three main live-cloning strategies:

- **Active cloning (AC):** A hot clone corresponding to each container within the PN is pre-spawned in the SN, regardless of the semantics of its connection to the adversary. When deception is due, no cloning is necessary. Only re-routing traffic through the sandboxed clone is.
- **Reactive cloning (RC):** Upon first contact with the SC, each benign container is cloned on-demand to the SN and then network confinement is applied by redirecting the adversary's traffic to the clone(s).
- **Proactive cloning (PC):** In this mode, we leverage existing historical data about recurring network flows using SDN so as to pre-clone carefully selected containers in the form of hot spares beforehand. This is a hybrid approach between AC and RC. As a fall-back in case of false negatives, RC cloning is used.

For PC, we use a heuristic function that considers the container size, the bytes transmitted and received within a discrete time window, and the duration of network flows to pre-clone active or highly interactive containers, and their k nearest neighbors. More advanced heuristics are left for our future work.

The cloning process achieves persistence across three dimensions: The *in-memory state*, the *local FS state*, and the *network state*. It is described in Algorithm1. We preserve the in-memory state by check-pointing the memory, and state of the process tree to disk using CRIU in user space[1] (L12-14) and to restore the state on the cloned counterpart (L22-23). The local FS state is preserved by explicitly getting the difference of the file system compared to the original image and CoW layers underneath, and then copying these files over to the container clone (L8-11). We also differentially copy

[1]http://criu.org

the externally mounted volumes as necessary (L4-7). Finally, to preserve the network state, we leverage the `docker inspect` command to get the container's network identifiers such as IP, MAC addresses, as well as the networks it is attached to (L3) then using the same exact configuration when creating the container clone on the Sandnet (L17).

Algorithm 1 Live container cloning

Input: Container to be cloned C
Output: Cloned container C'
Constants:
$snIP$, $volumesPath$, $changedFilesPath$, $checkPointsPath$

```
1. begin
2.   ▷Get the container conf using 'docker inspect'
3.   var config = getContainerConfigOnPN(C)
4.   for all volume ∈ config
5.     ▷Copy volume using 'rsync'
6.     deltaCopy(volume, snIP , volumesPath, C)
7.   end for
8.   ▷File changes using 'docker diff'
9.   for all file ∈ getChangedFiles(C)
10.    deltaCopy(file, snIP, changedFilesPath, C)
11.  end for
12.  ▷Checkpoint process tree using 'CRIU' and 'docker checkpoint'
13.  var checkpoint = checkpointPtree(C)
14.  deltaCopy(checkpoint, snIP, checkPointsPath, C)
15.
16.  ▷Honor container conf using 'docker create'
17.  C' = createContainerOnSNFromConfig(config)
18.  ▷Move changed files into the container root FS
19.  for all file ∈ getFilesInChangedFilesPath(C)
20.    deltaCopyHostToContainer(file, C')
21.  end for
22.  ▷Start container using 'docker start –checkpoint'
23.  C' = startContainerFromCheckpoint(C', checkpoint)
24. end
```

C. Network confinement mechanism

We identify two network confinement modes:

- **Atomic confinement:** Confinement of connections to sandboxed clones is on an all-or-nothing basis. Connections are not re-routed unless all clones are in a ready state. This mode is less likely to incur packet loss or connectivity degradation because re-routing to the clones only happens once all the clones are ready. However, it leaves the adversary potentially longer on its attacked containers.
- **In-process confinement:** Re-routing occurs iteratively as clones are made ready in the sandbox. One issue with this mode is the fragile deception surface during the transitional states. While the adversary's network flows are re-routed as soon as as possible, packet loss, and network throughput degradation are likely due to the temporary absence of dependent containers in the SN.

We chose atomic confinement in favor of a hypothetically higher QoD. Our network confinement methodology can be seen in Figure 2. The packet matching happens on the PN's hosts and their attached OpenVswitch (OVS) instances. The routes between the PN and the SN consist of appropriate rules to allow for packets to traverse the two broadcast domains, and for the containers to reach each others.

The semantics of confinement are as follows: packets egressing the SC are redirected to the SN whose OVS instance in turn delivers packets to the corresponding container clones. Broadcast packets (e.g. ARP requests) and unicast packets to the SC are not delivered to it but rather its substitute fresh clone on PN if they originate from containers within the PN because the SC is now virtually confined to the SN. Therefore, any packets from the SN to the PN are also only delivered to the SC to maintain isolation.

We also mirror network traffic (i.e. workload) within the PN to the clones so that CPU / IO changes are not distinguishable to the adversary. We distinguish between two kinds of traffic: *external* traffic and *internal* traffic. The external traffic is due to external clients (e.g. on the Internet) consuming the MSes on the PN. This causes a chain of subsequent internal requests between the MSes. Internal traffic could originate between the MSes without the need to serve an external request. For example, using timely tasks or cron jobs like health checks.

The internal traffic is implicitly mirrored due to cloning. External ingress traffic is mirrored by the edge switches to both the PN and the SN. The corresponding egress responses from the SN are then blocked from reaching the external network so as not to pollute the benign responses from the PN to the benign external clients. Responses from the PN to the adversary's external host are also blocked, in favor of responses from the SN to maintain deception.

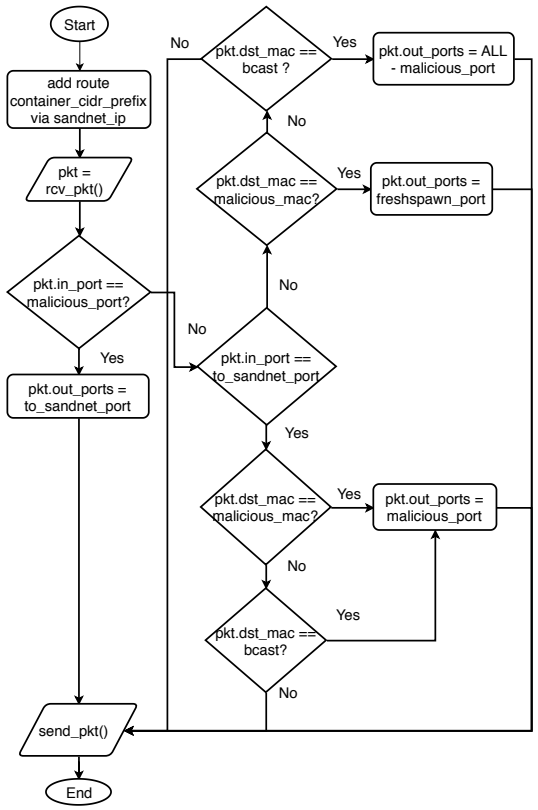

Figure 2. Packet confinement (We assume that containers are placed in the *container_cidr_prefix* network)

D. Quality of deception

To successfully deceive an adversary, Sandnet must be able to minimize the changes in KPIs observable to an attacker with a pre-image of the normal operation of the network. Hence, we introduce our deception metric which we coin the Quality of Deception (QoD).

We denote a sequence of KPI measurements of length n as:

$$KPI = [v_1, ..., v_n] \tag{1}$$

We then define the image of KPI as the tuple:

$$I = (f_1(KPI), ..., f_m(KPI)) \tag{2}$$

where I consists of m unique features computed as a function of KPI, e.g, the statistical variance or degree distribution of the network. In Sandnet, we choose $I = (\mu, \sigma)$ where μ and σ represent the mean and the standard deviation of KPI respectively. Three images are defined in Sandnet:

- *Expected image* (I_e): represents the attacker's blind expectation of a network's behavior. Past studies can be used to formulate I_e.
- *Calculated image* (I_c): is calculated by the attacker over a limited time interval when it successfully pivots into the network.
- *Observed image* (I_o): is the cumulative image observed by the attacker throughout its presence on PN and after confinement to SN.

The adversary's educated pre-image of the network is therefore:

$$I_p = g(I_e, I_c) = (p_1, ..., p_m) \tag{3}$$

where $g(x, y)$ is a function of x and y. We choose $g(x, y) = median(x, y)$ such that $I_p = (\frac{\mu_e+\mu_c}{2}, \frac{\sigma_e+\sigma_c}{2})$ where $I_e = (\mu_e, \sigma_e)$ and $I_c = (\mu_c, \sigma_c)$. The adversary can therefore use I_c to correct its expected image.

Let $I_o = (o_1, ..., o_m)$. Then, from (3):

$$\begin{aligned} MAX &= (max(p_1, o_1), ..., max(p_m, o_m)) \\ MIN &= (min(p_1, o_1), ..., min(p_m, o_m)) \end{aligned} \tag{4}$$

The normalized value of a feature $x_i \in I$ is therefore:

$$norm(x_i) = \frac{x_i - min_i}{max_i - min_i}, \; where: \tag{5}$$

$i \in [1, ..., m] \wedge min_i \in MIN \wedge max_i \in MAX$

We normalize I_p and I_o to $\widetilde{I_p}$ and $\widetilde{I_o}$ in $[0 - 1]$ using:

$$\begin{aligned} \widetilde{I_p} &= (norm(p_1), ..., norm(p_m)) \\ \widetilde{I_o} &= (norm(o_1), ..., norm(o_m)) \end{aligned} \tag{6}$$

For optimal deception, the attacker should not observe a difference between $\widetilde{I_p}$ and $\widetilde{I_o}$. We represent this difference as:

$$D = |\widetilde{I_o} - \widetilde{I_p}| = (d_1, ..., d_m) \tag{7}$$

The total change C in the features is therefore:

$C = (\sum_{i=1}^{m} d_i)/m$

Hence, $0 \leq C \leq 1$. C is maximum when there are significant differences between $\widetilde{I_p}$ and $\widetilde{I_o}$. Therefore, given k KPIs, we formulate the QoD as:

$$QoD = 1 - (\sum_{i=1}^{k} C_i)/k \tag{8}$$

where C_i is the total change in features for KPI_i. As the total changes across KPIs increase, the QoD decreases. Therefore, the attacker can infer that it got sandboxed when QoD is below a certain threshold $t \in \mathbb{R}$.

We evaluate Sandnet's QoD according to KPIs related to both: the local container's state, as well as the network I/O traversing the container from SC's perspective. We formulate Sandnet's measured KPIs as the tuple: $(R_{L2}, R_{L3}, T, L, D, FS)$ where R_{L2} is the set of reachable MAC addresses from the SC, R_{L3} is the set of reachable IP addresses, T is the network throughput between the adversary's container and one ore more other containers, L is the network latency to another container, D is the experienced down time, and FS is a recursive file system diff between the original container and the clone.

It is worth mentioning that FS entails a lot of local information about the container that is stored in the $/proc$ file system, e.g., the list of running processes, list of file descriptors of processes, memory and CPU utilization, the routing table, the ARP table, as well as the list of network interfaces along with their TX and RX counters. We exclude the memory maps and pagefile maps from our calculation, as these are usually subject to randomization (e.g. ASLR) as a part of memory protection.

IV. Evaluation

A. Evaluation setup

We conducted our evaluations on a testbed consisting of 3 Intel NUCs representing the PN, the SN and the external network. Each NUC has 8GB of RAM, a 256GB SSD, a Core i5-7260U processor with two hyper-threaded cores and a 1Gbps network connection between them. A second 1 Gbps NIC is used for the management, and control traffic related to the initiation of live network deception. The NUCs were running Linux kernel v4.9.0, Debian Linux v8, docker v17.06, Openvswitch v2.3.2 with an appropriate docker network plug-in, and version 3.10 of the CRIU software.

B. Microservice workload under test

The MS application under test consists of a 2x replicated 3-layer container-based deployment which is comprised of a *frontend* (FE) web server running the nginx caching web server, a *backend* server (BE) performing the business logic using the flask web framework running as a WSGI application within Gunicorn which in turn persists its data to a *database* (DB) tier represented by a redis key-value store. The versions of the docker images used were as follows: redis *version 5.0*, python-2.7-slim *version 19.6* for the flask application container, and nginx version *1.15.5-alpine*. All six containers are connected via $openvswitch$ as illustrated in Figure 1

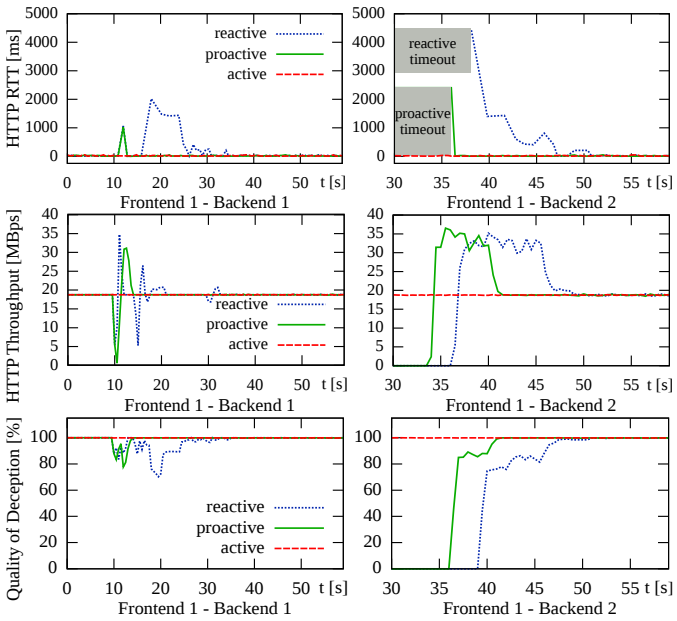

Figure 3. QoD relative to KPIs

In all our experiments we emulated adversarial presence on the FE MS and employed our three aforementioned cloning strategies along with the network confinement mechanism accordingly. The flow of the experiment runs in two phases: First, the FE ($frontend\,1$ or FE1) internally accesses the BE to serve incoming end user requests, which in turn generates a R/W load on the DB to retrieve and/or modify data. After 10s, the IDS detects the adversarial incident and triggers the live network deception routine. At the 30s mark, the adversarial FE changes its behavior to further access the BE of the second replica group ($backend\,2$ or BE1). This generates a similar workload for the second replica group.

We computed the QoD w.r.t. the compromised FE, and the benign FE of the second replica group ($frontend\,2$ or FE2) was used as a baseline to compare the deception quality of the cloned containers on SN to their non-cloned counterparts which reside on the PN.

C. Quality of deception

We used the tools $httping$, and $fasthttploader$ to measure the HTTP latency as Round Trip Time (RTT), and the HTTP throughput accordingly. We then inferred the total down time as the time at which the MS under test is unresponsive and packet loss occurs. To compute the changes in the FS before and after cloning, we used the $diff$ tool to recursively compare the FS trees. The network reachability from the SC were measured using $arp-scan$ and $nmap$ accordingly.

Figure 3 shows the impact of the three live cloning strategies we proposed on the HTTP RTT and the throughput (y-axis) through the experiment's lifetime (x-axis). RC is drawn in blue, whereas AC & PC are drawn in red and green. Overall, AC outperforms PC and RC, and PC shows a gain over RC. This can be reflected in their corresponding QoD accordingly.

At the 10s mark, when the IDS has signaled the intrusion, a small latency spike (in green) with the magnitude of 1s can be observed for both AC and RC. Shortly after, a 2s spike is observed for only RC. This is because the PC strategy has marked the DB as eligible for pre-cloning and clones both DB1 and its dependent BE1 concurrently. However, RC first clones BE1, and serially clones DB1 when the first HTTP request is made to it, which causes a significant latency disruption for a longer duration.

As our approach favors atomic network confinement, the adversary's connections are re-routed to the SN only when the corresponding clones are ready. For this reason, only ~1s of latency overhead is experienced for PC which is roughly the time for CRIU to freeze the memory state of BE1 and DB1 to dump it to disk. It does not include the other overheads due to copying and restoring on the SN.

At t=30s, when the adversary contacts BE2, a much larger downtime is experienced for PC and RC. At this time, the adversary's connections have been confined to only the SN and may no longer cross the isolation boundary between the SN and the PN. As the newly contacted containers BE2 and DB2 have not yet been cloned, the adversary experiences the cloning time in its totality. Namely, checkpoint creation, followed by a container FS file diff creation, copying, and restoration.

A similar observation can be drawn about the HTTP throughput. When cloning is imminent, a momentary decrease in throughput due to downtime occurs. This is followed by a quick rise of throughput over the norm due to retransmissions. The remaining KPIs; namely, the link-layer and network-layer reachability, and the FS difference varied in the transient state of RC and PC (i.e. while a container was being cloned), but stabilized after completion. They were not plotted individually due to space limitations.

The bottom plot of Figure 3 shows how the aforementioned KPIs affects the QoD. AC achieves 100% QoD, whereas PC on average achieves a higher QoD than RC due to its lower downtime and the concurrent pre-cloning of strongly connected containers. During downtime, the QoD is zero as the total difference between the normalized adversary's pre-image of the KPIs and their actual values is at its maximum value: one.

D. Breakdown of the live cloning workflow

In this subsection we attempt to answer the question: "which step of the live cloning workflow is the most time consuming?". Table I sheds light on the mean time costs for each step involved when cloning both The BE and the DB. The majority of the time was spent on restoring the memory checkpoint on the cloned container, as well as on updating the changed files onto the cloned container. As the BE is significantly more complex and larger in image size than the DB, it results in a larger memory snapshot which is more costly to copy over the network. These results also align with past work [6].

Table I
DETAILED TIME COSTS OF LIVE CLONING

Step	Backend	Database
Checkpoint creation	0.5824s	0.4759s
FS diff create & copy	0.5492s	0.0168s
Checkpoint copy	2.2015s	0.7425s
FS diff application	0.7163s	0.7865s
Checkpoint restoration	0.8494s	0.7653s

E. Resource overhead

Achieving a higher QoD comes at the cost of resource duplication due to cloning and traffic mirroring. In this section, we estimate the resources overhead in terms of time complexity due to the three cloning strategies as well as the two network confinement mechanisms.

1) Traffic mirroring: To keep the state between the PN and the SN synchronized, and keep the containers' workload similar, traffic mirroring between the PN and the SN is required. This adds an extra $\mathcal{O}(f)$ of overhead, where f is number of forwarded packets at the edge switch.

2) Network confinement: Due to network confinement (Figure 2), an extra packet processing overhead of $\mathcal{O}(p)$ is endured, where p is the number of forwarded packets at the OVS instance of the PN. This is due to the extra SDN header matching logic. Additionally, in case of atomic confinement, a time overhead of $\mathcal{O}(1)$ is added due to rerouting the packets once cloning is complete. For in-process confinement, this overhead is calculated as $\mathcal{O}(k)$, where k is the number of cloned containers. After cloning each container, the corresponding flows are redirected to it.

3) Live cloning: For AC, extra $\mathcal{O}(n), n = number\,of\,containers$, of computation, network, and storage resources are necessary because all containers on the PN are pre-cloned. PC, and RC require $\mathcal{O}(n-k_1)$ and $\mathcal{O}(n-k_2)$ extra resources respectively, where k_1 is the number of containers which were *not* pre-cloned according to the chosen heuristic, and k_2 is the number of containers which were *not* contacted by the SC. We further note that $k_1 \leq k_2$ as PC might elect containers for pre-cloning irrespective of them being contacted by SC.

4) Total: We quantify the total overhead as the sum of the above. Hence:

$$O_{total} = O_{cloning} + O_{traffic\,mirr.} + O_{net\,conf.} \quad (9)$$
$$= \mathcal{O}(n-c) + \mathcal{O}(f) + \mathcal{O}(p) + \mathcal{O}(1),\ 0 \leq c \leq n$$

where c is the number of containers which were not cloned. The overhead due to deploying Sandnet is linear in the number of external ingress packets, the number of internal packets within the PN, and the number of cloned containers. Through future work, we plan to examine and reduce this overhead.

F. Impact on the production network

The impact on containers in the PN can be observed in Figure 4. Throughout the lifetime of the experiment, the frontend of the second replica group continues to consume the backend and the database accordingly. At 30s, a downtime of roughly 1s can be observed, through which a latency spike and

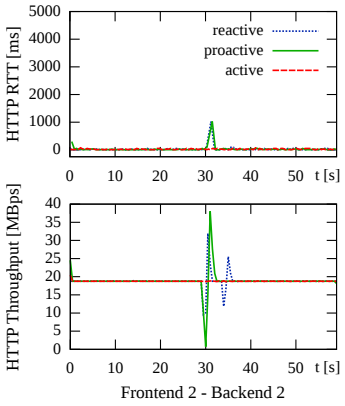

Figure 4. Impact of live cloning on the production network

throughput degradation occurs. This downtime is accounted for due to the memory snapshotting phase of our live cloning routine using CRIU. During this time, it is necessary to freeze the memory of the running container, and dump it to disk for cloning. This effectively freezes the running processes tree within the container. Hence, we conclude that the impact on the running MS on the PN is minimal.

G. Discussion

From an attacker's perspective, the compromised service appears to reside on the PN. However, in fact it is confined to the SN. We conclude that an optimal QoD is achievable. However, this comes at the cost of a significant resource overhead. In data center and highly-available deployments, MSes are typically replicated across data centers, availability zones and hosts to ensure maximum resilience. This can be capitalized on to reap the benefits of detecting and thwarting new attacks by deploying Sandnet in AC mode which provides a constant optimal QoD.

In resource-constrained environments, we provide two alternatives: PC and RC which achieve a reasonable QoD. The latter incurs a worst-case QoD of 70.26% for a very short period and a downtime of ~7s, while the former incurs a worst-case QoD of 77.70% and a downtime of ~3s. Both quickly reach a 100% QoD after stabilization and recover once the live deception process is complete. In fact, the ability of the adversary to distinguish normal degradation in KPIs from degradation due to deception remains in favor of Sandnet [7], because KPIs can still be emulated to add noise to the adversary's perception. Traffic could also be redirected to standby honeypots in transition, and critical confidential data on the SN could be indistinguishably obscured [8].

V. RELATED WORK

In this section, we survey the related work. We classify past literature into three main directions: traditional honeypots, network deception, and live cloning and migration.

A. Traditional honeypots

Traditional honeypots only purvey limited functionalities to the adversary, are intentionally vulnerable and identifiable [4]. Moreover, decoys using Virtual Machine Monitor (VMM) introspection are susceptible to various limitations and are detectable [9]. Ongoing work deployed multiple honeypots in a network (Honeynets) [10] and evaluated the combination of different kinds of honeypots in that context [11]. Honeynets remain fingerprintable, and are limited in the face or propagating attacks. Therefore, dynamic intelligent honeypot selection and routing has become a problem of interest. Systems like HoneyMix [3] and HoneyProxy [12] address these limitations by re-configuring the network [13] and co-deploying honeypots with real servers.

B. Network deception

Altering the attacker's perception of the network is one way to alleviate the need for topologically correct decoys. This problem lies at the heart of network deception [14]. Moving target defense techniques conceal topological information by transparently changing the network identifiers of the hosts (i.e. IP addresses). Hence, reconnaissance information gathered by an adversary become of no value as the attack surface changes [15]. Other work makes more network properties indistinguishable to the adversary such as the connectivity information between the hosts, and the traffic patterns [16].

C. Live migration & cloning

VM live migration algorithms could be modified to create realistic on-demand decoys which serve as isolated copies from the originals [17]. However, this approach suffers from a large turnaround time (several seconds) due to copying large VM snapshots and connectivity disruptions. Further improvements to preserve VM network identifiers [18] and speeding up cloning [19] have been suggested. Recently, only limited approaches studied the live migration of SDN topologies [20] and migrating containers [6]. However, this was neither performed in an adversarial setting nor considered realistic deceptive measures.

VI. Conclusion

This work introduced a deception network - *Sandnet* - which sandboxes an internal adversary to a network segment using SDN and actively clones containerized applications to its solitary network domain in a microservice deployment. This allows for a more complete investigation of evolutionary behaviors in network attacks such as APTs. Our contributions included an implementation of Sandnet, a novel metric to measure the Quality of Deception (QoD), and a qualitative evaluation between three introduced strategies for live deception. Our results show that for the most resource-consuming strategy, an optimal QoD can be achieved, and for the least resource-consuming strategy, only a small downtime of approx. 3s per container was experienced which led to a worst-case QoD of 70.26%. Our future work is aimed at optimizing the QoD for multiple co-present adversaries, and evolving the system to handle complex adversaries such as colluding entities or adversaries with multiple entry points to the production network.

Acknowledgments

This work in parts has been funded through the Cluster of Excellence EXC 2050 "CeTI" and BMBF FASTcloud.

References

[1] C. Modi, D. Patel, B. Borisaniya, H. Patel, A. Patel, and M. Rajarajan, "A survey of intrusion detection techniques in cloud," *J. Netw. Comput. Appl.*, vol. 36, no. 1, pp. 42–57, jan 2013.

[2] E. Vasilomanolakis, S. Karuppayah, M. Muhlhauser, and M. Fischer, "Taxonomy and survey of collaborative intrusion detection," *ACM CSUR*, vol. 47, no. 4, pp. 1–33, may 2015.

[3] W. Han, Z. Zhao, A. Doupé, and G.-J. Ahn, "HoneyMix," in *ACM SDN-NFV Security*, 2016.

[4] A. Vetterl and R. Clayton, "Bitter harvest: Systematically fingerprinting low- and medium-interaction honeypots at internet scale," in *USENIX WOOT*, Baltimore, MD, 2018. [Online]. Available: https://www.usenix.org/conference/woot18/presentation/vetterl

[5] W. Felter, A. Ferreira, R. Rajamony, and J. Rubio, "An updated performance comparison of virtual machines and linux containers," in *Performance Analysis of Systems and Software (ISPASS)*. IEEE, 2015, pp. 171–172.

[6] S. Nadgowda, S. Suneja, N. Bila, and C. Isci, "Voyager: Complete container state migration," in *IEEE ICDCS*, jun 2017.

[7] P. Casas, P. Fiadino, S. Wassermann, S. Traverso, A. D. Alconzo, E. Tego, F. Matera, and M. Mellia, "Unveiling network and service performance degradation in the wild with mplane," *IEEE Communications Magazine*, vol. 54, no. 3, pp. 71–79, mar 2016.

[8] A. Webster, "An application of jeeves for honeypot sanitization," Tech. Rep., 2018.

[9] T. Garfinkel and M. Rosenblum, "A virtual machine introspection based architecture for intrusion detection," in *NDSS*, 2003, pp. 191–206.

[10] W. Fan, Z. Du, and D. Fernandez, "Taxonomy of honeynet solutions," in *SAI Intelligent Systems Conference (IntelliSys)*. IEEE, nov 2015.

[11] M. Vrable, J. Ma, J. Chen, D. Moore, E. Vandekieft, A. C. Snoeren, G. M. Voelker, and S. Savage, "Scalability, fidelity, and containment in the potemkin virtual honeyfarm," in *ACM SoSP*, 2005.

[12] S. Kyung, W. Han, N. Tiwari, V. H. Dixit, L. Srinivas, Z. Zhao, A. Doupe, and G.-J. Ahn, "HoneyProxy: Design and implementation of next-generation honeynet via SDN," in *IEEE CNS*, oct 2017.

[13] M. P. Stoecklin, J. Zhang, F. Araujo, and T. Taylor, "Dressed up," in *ACM SDN-NFV Security*, 2018.

[14] X. Han, N. Kheir, and D. Balzarotti, "Deception techniques in computer security," *ACM CSUR*, vol. 51, no. 4, pp. 1–36, jul 2018.

[15] V. E. Urias, W. M. Stout, and C. Loverro, "Computer network deception as a moving target defense," in *International Carnahan Conference on Security Technology (ICCST)*. IEEE, sep 2015.

[16] K. Park, S. Woo, D. Moon, and H. Choi, "Secure cyber deception architecture and decoy injection to mitigate the insider threat," *MDPI SYMMAM*, vol. 10, no. 1, p. 14, jan 2018.

[17] A. Hirata, D. Miyamoto, M. Nakayama, and H. Esaki, "INTERCEPT+: SDN support for live migration-based honeypots," in *4th International Workshop on Building Analysis Datasets and Gathering Experience Returns for Security (BADGERS)*. IEEE, nov 2015.

[18] T. K. Lengyel, J. Neumann, S. Maresca, and A. Kiayias, "Towards hybrid honeynets via virtual machine introspection and cloning," in *Network and System Security*. Springer Berlin Heidelberg, 2013, pp. 164–177.

[19] H. A. Lagar-Cavilla, J. A. Whitney, A. M. Scannell, P. Patchin, S. M. Rumble, E. de Lara, M. Brudno, and M. Satyanarayanan, "SnowFlock," in *ACM EuroSys*, 2009.

[20] Y. Zhao, S. Lo, E. Zegura, M. Ammar, and N. Riga, "Virtual network migration on the GENI wide-area SDN-enabled infrastructure," in *IEEE INFOCOM*, may 2017.