Proceedings of the 53rd Hawaii International Conference on System Sciences | 2020

Delivering Honeypots as a Service

Jafar Haadi Jafarian
University of Colorado Denver
haadi.jafarian@ucdenver.edu

Amirreza Niakanlahiji
University of Illinois Springfield
aniak2@uis.edu

Abstract

The effect of honeypots in slowing down attacks and collecting their signatures is well-known. Despite their known effectiveness, these technologies have remained underutilized, especially by small and medium-sized enterprises, because internal hosting and configuration of honeypots requires extensive expertise and infrastructure, which is unjustifiably expensive especially for small or medium-sized enterprises. In this paper, we propose a novel security approach that enables a security service provider to offer honeypot-as-a-service (HaaS) to customer enterprises. The HaaS service is offered by a plug-and-play gateway and incorporates a network of moving high-interaction honeypots into unused address space of client enterprises. These honeypots are configured tailored to the mission and type of services offered by the customer enterprise to blend in the surrounding network for maximum believability while looking vulnerable enough to engage potential attackers. As a contribution, we formulate and solve the problem of strategic configuration planning of a group of honeypots for a given input network. We also provide the necessary infrastructure and mechanisms for realizing the model and offering it to client enterprises without affecting their regular operations. Using experimental and analytical modeling, we evaluate our approach and show its robustness against honeypot mapping attacks, and its effectiveness in slowing down large-scale cyber intrusion attacks on enterprise networks.

1. Introduction

Honeypots are invaluable for detection and analysis of network attacks. In spite of their proven advantages in attack deterrence, honeypots have not been widely adopted by enterprises, because deployment, configuration, maintenance, and analysis of honeypots are expensive, thus making their adoption financially unjustifiable. Existing honeypot technologies suffer from a cost-benefit paradox; cheap and scalable emulated (low-interaction) honeypots are easily detectable and thus not really effective against real attacks [1], while stealthy real-machine (high-interaction) honeypots are expensive to build and maintain [1], and with careless maintenance, they may even make the network more vulnerable [2].

This gives rise to our novel honeypot deployment model, based on the "software as a service" initiative, in which a third-party service provider offers "honeypots as a service" (HaaS) to customers, by populating their external and internal address space with a group of high-fidelity, hard-to-detect, and customer-tailored honeypots. This model allows small- and medium-sized enterprises to unleash the full potentials of honeypots in deterring sophisticated cyber threats, without needing to purchase expensive hardware and software and hiring professionals for honeypot configuration, analysis, and maintenance.

The technological and theoretical aspects of developing honeypots as a service lead to new research challenges. The *first* fundamental question that we answer is how the HaaS provider can incorporate honeypots in a customer network, in a way that is scalable, flexible, secure and more importantly non-intrusive to network operation and sessions, as well as devices and protocols. In Section 3, we present necessary architectural components and protocols for the realization of the HaaS model with the aforementioned properties.

The *second* and more fundamental question is defining criteria and techniques for effective design and configuration of these honeypots. To address this question, in Section 4, we first identify three seminal criteria for designing an effective deception plan. Then, we provide a scientific foundation for realizing each of these criteria in

URI: https://hdl.handle.net/10125/63966
978-0-9981331-3-3

the honeypot planning problem. We show that the resulting planning problem is NP-complete. Thus, we provide a modeling of the problem based on Satisfiability Modulo Theories (SMT) [3]. The model is then solved using off-the-shelf SMT solvers (*e.g.*, Microsoft Z3 [3]) to determine the optimal honeypot network plan for a customer network.

To evaluate the model, we use two types of evaluation methodologies, the result of which are presented in Section 5. First, we implemented the model in a tested and used a red-teaming experiment with a group of security experts to evaluate the accuracy and effectiveness of our criteria and techniques in a small-scale testbed. The subjects were asked to differentiate between honeypots and real machines in different setups. These experiments show that the honeypots generated by our methodology were the least recognizable in comparison with honeypots that are either low-interaction, configured in isolation (with no planning) and include some arbitrary services that may not blend well in the surrounding network, or suffer from poor (unrealistic) design, configuration, or content. In the second stage, we generalized the red-teaming results in an analytical framework to quantify and show the effectiveness of the HaaS model for slowing down large-scale network attacks on enterprises with different sizes.

2. Related Work

Based on level of interaction, honeypots are classified to *low-interaction*, and *high-interaction*. While low-interaction honeypots are usually implemented as a daemon (`honeyd` [1], `dionaea` [4]) that emulates behaviors of a real system, high-interaction honeypots are usually implemented as real machines, with full-fledged operating system and applications. Low-interaction honeypots require less computational resources, but their limited implementation of protocols is easily detectable by skilled attackers [5, 6]. In contrast, while realistic nature of high-interaction honeypot makes identifying them harder, (1) they are susceptible to being taken over by attackers, thus increasing the risk [2]; (2) their deployment and maintenance is extremely costly in terms of expertise and resources [1]; and (3) they are not scalable, as each high-interaction honeypot can only be one machine with a static configuration [1, 7].

A line of work close to the HaaS model are centralized honeypot architecture [8, 9] called *honeyfarm* [8, 10, 11]. These honeypots are centrally operated while each of them may virtually belonging to different network domains.

Another class of work includes distributed honeypot models with sensors distributed across multiple networks to observe global trends [12, 13].

Another class of work that has commonalities with our proposed model focused on dynamic honeypot solutions [14, 15]. A dynamic honeypot passively [16] or actively [17] discover production systems and network events, and based on that information create or adapt honeypot systems.

3. HaaS Infrastructure

In this section, we describe the architecture, components, and protocols of the HaaS model.

3.1. Architecture

The HaaS model aims to thwart attackers by populating the unused address space with honeypots. However, these honeypots are not physically hosted on the customer's network. Rather, they are merely a redirection to unmodified and full-fledged platforms/services that are hosted in a cloud of machines which are controlled by the HaaS provider. The architecture of the HaaS model is depicted in Figure 1. The HaaS gateway, deployed at in front of critical subnets (*e.g.*, DMZ) of customer networks, is responsible for (1) mutating addresses and updating DNS replies issued by the authoritative DNS, as well as (2) redirecting designated flows to the HaaS provider network. The gateway has two interfaces: the `pcap/libnet` interface that performs packet capturing and forwarding, and the *tunneling* interface that tunnels designated packets to the HaaS provider network. One mutation gateways must be located in front of each subnet, including DMZ and internal subnets. The DMZ HaaS gateway protected the network against external attackers, while internal HaaS gateways target internal threats.

On the provider's side, honeypot cloud is, in essence, a data center of high-end machines, each hosting a multitude of virtual machines (VM). VPN gateway is responsible for decapsulating incoming tunneled packets and forwarding them to cloud VMs, as well as encapsulation and forwarding of response packets. In this architecture, the production traffic is never routed to the HaaS network. Moreover, the HaaS firewall ensures that only traffic to/from the predetermined IP/port pairs are permitted and any other packet is dropped. This prohibits an attacker on the Internet or

even in the customer network (insider) to access an unauthorized machine or service in the HaaS network. Moreover, the firewall does not allow any new flow (*e.g.*, TCP SYN) from the HaaS provider machines to the customer network to ensure that if a HaaS network machine is compromised, it does not pose any risk to the customer network. The combination of tunneling and firewalling ensures complete isolation of customer network from the HaaS provider network, except for whitelisted packets.

The HaaS model provider offers services to customer networks based on a service-level agreement (SLA). This SLA determines the traffic rate and the bandwidth that is allocated to this customer.

3.2. Communication Protocols

To protect honeypots from mapping attacks, the HaaS model constantly randomizes or mutates IP addresses of network hosts (both real ones and honeypots) at regular intervals. In Section 4, we provide a more detailed justification for why these IP address mutations are central to achieving a robust honeypot-based defense.

In order to keep address mutations transparent to end-hosts, devices, and protocols, the HaaS gateway keeps the actual IP addresses of real hosts unchanged. When a user queries an authoritative DNS for the IP address of a host, the DNS response provides a temporary IP address instead of the host's real IP address to the querying resolver. This temporary IP is randomly chosen from the unused address space and temporarily assigned to a host, and the assignments are updated at regular intervals (*e.g.*, every 5 min). The user will communicate with this temporary IP as the destination address. However, this temporary destination IP is translated to the host's real IP address before packets are delivered to the host. The opposite translation is done by the HaaS gateway for source IP addresses in egress packets from the host. The HaaS gateway performs these translations for the whole duration of a flow to ensure end-to-end reachability and session integrity. Figure 2 provides a detailed description of how legitimate communication with a host occurs in a HaaS-protected network.

3.3. Proxy Honeypots

The HaaS gateway uses the IP/port translation to generate proxy honeypots on designated IP addresses. These proxy honeypots are generated by redirection of incoming traffic to different virtual machines in the HaaS provider's cloud based on a pre-planned table. For example, in Figure 3, while from the attacker's perspective, there exists a host on address IP_{rand} with ports 80 and 21 open, in reality this is a proxy honeypot that is generated by redirecting traffic destined to port 80 to one honeypot VM, and traffic to port 21 to another VM.

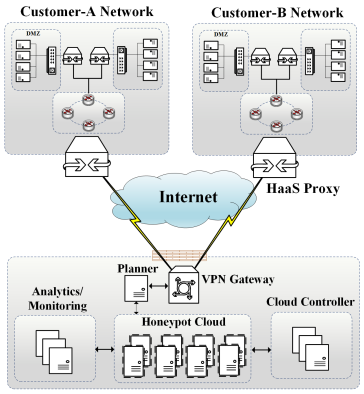

Figure 1. Architecture of the HaaS model

4. HaaS Honeypot Network Planning

In this section, we define and solve the problem of effective, mission-oriented, and customer-tailored honeypot network planning. To this aim, first, we identify three criteria for effective honeypot-based deception planning. Then, we discuss how each of these criteria could be scientifically modeled. Finally, we formally define and solve the planning problem.

Once the honeypot network plan is created, it is possible for an enterprise to host it locally or to use VMs from other cloud-based models like infrastructure as a service. However, design, configuration, hosting, and maintenance of high-fidelity honeypot machines requires expensive infrastructure and human resources which may not be affordable for a small- or medium-sized enterprise. The HaaS provider, on the other hand, can reduce these costs through unified management, aggregation, and sharing of infrastructures and resources among different customers, thus making this solution affordable for smaller enterprises. Moreover, by hosting a variety of different operating systems, servers and applications inside a consolidated and closely-monitored network, the HaaS provider can create honeypot networks that

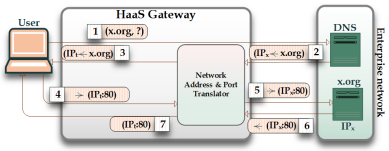

Figure 2. communication protocol for regular access

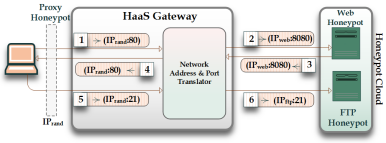

Figure 3. Proxy honeypots: redirection of designated flows to Honeycloud

have machines with very diverse designs and configurations and very believable and high-fidelity contents.

4.1. Criteria for Effective Deception

Investigating literature on designing effective deceptive paradigms points to two important properties for successful deception: (1) *plausible* deception, by conforming deception to attackers' expectations [18], and (2) *appealing* deception to provide temptation for attackers to engage [19]. In addition to these two, we show that (3) *moving* is also necessary to defeat honeypot mapping attacks [20].

Plausible Deception. Achieving a plausible deception is tantamount to making honeypots indistinguishable from production hosts in the customer network. To achieve such indistinguishability, a honeypot must have three properties: (a) plausible in its compliance with legacy protocols (high-interaction), (b) plausible design (using expected platforms and services), and (c) plausible configuration and content.

Appealing Deception. A honeypot or group of honeypots are appealing when they provide incentives for adversaries to interact with them. To this aim, (1) they must offer a diverse range of platform and services to engage attackers with different capabilities and agendas, and (2) they must include platforms and services that are sufficiently vulnerable to provide incentives for attackers to engage.

Moving Deception. Any deception scheme is susceptible to being detected [21]. One of the major limitations of existing honeypot paradigms is that once a honeypot is detected, it loses its value [1]. Mutating (randomizing over time) addresses of hosts (honeypots and real) disallow adversaries from mapping and blacklisting honeypots. Moving is handled by the address mutation component of the HaaS model, as discussed in Section 3. In the next subsections, we define the other two criteria of plausible and appealing honeypot design and configuration, followed by a formulation of the problem.

4.2. Plausible Honeypots

We identify three factors for building a plausible honeypot: (1) high-interaction OS and services, (2) plausible design (platforms and services), and (3) plausible configuration and content.

High-interaction. High-interaction honeypots [2] are usually implemented as real systems, consisting of a full-fledged operating system and unmodified services.

Plausible design. Honeypots must be designed in a manner that is plausible in general, and also in the context of its surrounding network. Firstly, having (a) outdated services (*e.g.*, `Windows XP`), or abnormally vulnerable services would seem suspicious to human attackers. This means that every honeypot platform/service must be *individually* plausible, irrespective of the customer network. Plausible services are determined by investigating the occurrence frequency of services in a large number of diverse networks. Implausible services should not be included in designing honeypots.

Secondly, proxy honeypots must conform to the posture of the enterprise network. To this aim, (b) each honeypot must only include services that are relevant to the mission and type of the network in which they are deployed. Also, (c) every honeypot must include a reasonable number of services. For example, hosting too many services to a honeypot could raise suspicion.

Plausible Configuration. After discovering the configuration or fingerprint of a host, a skilled attacker would interact with a honeypot service in an attempt to compromise it. If the deployed content on honeypot services are not believable, both individually and with regard to the type and mission of the surrounding network, it could raise suspicion. Therefore, honeypot services must be properly configured and populated with meaningful and plausible data and content. On the other hand, an FTP server with anonymous login enabled, or a Web application filled with all types of

XSS and SQL injection vulnerabilities would look suspicious to a skilled human attacker, especially in comparison with the hardened real hosts.

4.3. Appealing Honeypots

As discussed in Section 4.1, appealing honeypots must have diverse types of services and applications and also include a sufficient number of vulnerabilities to engage attackers.

High Diversity. Attackers use a multitude of exploits, each targeting a vulnerability in a specific service. Such exploits are only useful when one of the targeted hosts is running the vulnerable service for that exploit. Therefore, having a diverse set of services in the network would increase the probability that an attacker is engaged with a honeypot.

The *inverse Simpson index* is a well-known metric in ecology and economics to calculate diversity when entities are classified into types. The value of a diversity index increases both when the number of types increases, and when evenness increases. Both factors are important for good deception; the number of services increases the possibility that the attacker engages with a honeypot, and evenness prohibits a honeypot service to stand out, due to a low occurrence. Using this metric, we ensure that the diversity of services does not fall below an acceptable threshold.

High Vulnerability Level. Every honeypot must be constructed in a way that lures adversaries to engage with it. A honeypot that includes a few hardened/secured services would seem too hard to compromise, thus discouraging adversaries from engaging. However, if the services are too vulnerable, it may also raise suspicion and reveal their decoy nature.

4.4. Problem Formulation

Generation of honeypots with customer-tailored, plausible and appealing configurations is a planning problem that must be solved by the HaaS provider. This planning determines the number and configuration of honeypots that are provided to the customer network. The problem is defined as follows: : given (1) goals of the model (designing plausible and appealing honeypots), (2) number of available (unused) addresses, (3) configuration of the customer network,and (4) the agreed-upon service-level agreement (SLA), how many proxy honeypots and with what configurations must be allocated to the customer's network.

This problem is a generalization of the unbounded knapsack problem, where each service type has a value (based on its vulnerability level and diversity) and weight (expected attack rate), and the SLA traffic rate defines the maximum total weight. The knapsack optimization problem is NP-complete. Therefore, to solve this planning problem, we first reduce it to a satisfiability problem by formulating it using satisfiability modulo theories (SMT) logic [3]. Then, we use an off-the-shelf SMT solver (Microsoft Z3) to solve the NP-complete problem.

Suppose m denotes the number of available and routable addresses in the address space of the customer network, and n denotes the number of production hosts in this address space; therefore, $m - n$ remaining addresses could be used for honeypots.

Physical network services in the HaaS provider cloud are categorized into classes $\{S_1, \ldots\}$. Each S_j represents a set of services of the same class; either implementing the same protocol or offering the same functionality (*e.g.*, FTP server, Web server, operating system, *etc.*). Network services are associated with a default port; *e.g.*, port 80 for Web services, 21 for FTP Servers, *etc.*For example, S_1 denotes the set of operating systems; that is, S_1 denotes the set of candidate OSs that could be observed on any honeypot in the customer network. In our formalization, variable $\boldsymbol{P}_{i,j}$ denotes the decoy service of class S_j that is assigned to i^{th} decoy host. These variables are assigned by the SMT solver. $\boldsymbol{P}_{i,j} = \phi$ denotes that no service is assigned.

The formal definition of the problem is as follows: determine a satisfiable assignment to variables of the form $\boldsymbol{P}_{i,j}$ for every production host and proxy honeypot, such that the given constraints are satisfied. Eq. 11 to 10 show the formulation of this problem using Satisfiability Modulo Theories (SMT) logic. Next, we provide a quick explanation of how this formulation captures and models the aforementioned criteria and constraints.

Maximum number of honeypots: The total number of production hosts and honeypots cannot exceed the number of available addresses, m. Eq. 1 defines variable $\boldsymbol{L}_i$ to denote whether a decoy machine is assigned to i^{th} unused IP address. Based on this equation, $\boldsymbol{L}_i$ is true (or 1) if and only if at least one decoy service is assigned to it. Eq. 2 denotes the constraint that the number of decoy machines must be equal to $m - n$.

Plausible Design: Designing plausible honeypots is the core objective of this planning.

Eq. 3 denotes that every decoy machine must have an operating system. Eq. 4 ensures that the number of services assigned to each decoy machine does not fall below or exceed the lower and upper bounds l_{min} and l_{max}, respectively. Eq. 5 represents a set of constraints that denote the set of inconsistent services that must not be simultaneously assigned to the same decoy machine. This equation constrains assignment of services of a type S_k on a honeypot depending on the service of type S_j that is assigned to it. For example, if for a decoy machine, $\boldsymbol{P}_{i,1} = Linux$, then $\boldsymbol{P}_{i,80} \neq IIS$, because `IIS` is a Windows service.

Appealing Design: The honeypot design is appealing when it includes a diverse set of services for engaging different types of attackers with different techniques or agenda and sufficiently high vulnerability to provide an incentive for attackers to engage. For every service r (*e.g.*, Windows, Linux, IIS), Eq. 6 sets $\boldsymbol{Q}_r$ to the number of occurrences of r in all honeypots over the total number of services in all honeypots. Thus, $\boldsymbol{Q}_r$ denotes the occurrence ratio of service r. Then, Eq. 7 calculates Simpson index and puts a lower bound threshold, denoted as Φ, on it to ensure that the overall set of services assigned to all honeypots is highly diverse.

To ensure sufficiently high - but not unrealistically high - vulnerability, we assume the input value γ_r denotes the vulnerability score of service r, which is calculated based on the *Lai and Hsia*'s model [22]. Eq. 8 sets variable $\boldsymbol{\Gamma}_{i,j}$ to the vulnerability score of the service assigned to it. Then, Eq. 9 determines the average vulnerability score of a decoy machine i. This is denoted as Γ_i. Finally, Eq. 10 ensures that the expected vulnerability score of each decoy machine is within the acceptable bounds denoted by $[\gamma_{min}, \gamma_{max}]$.

Compliance with service-level agreement: The expected traffic rate between customer and provider networks must respect SLA. To model this, the HaaS provider assigns an expected flow rate for each service class S_j, which is denoted as δ_j. Given the agreed-upon aggregate traffic threshold, denoted as Δ and expected flow rates for each service (δ_j), Eq. 11 ensures that the aggregate expected rate to the HaaS network remains close to the threshold, Δ.

5. Evaluation

In this section, we present our results on evaluating effectiveness of honeypots that are designed by the HaaS model.

[Maximizing Number of Honeypots]

$$\boldsymbol{L}_i \leftrightarrow (\bigvee_j (\boldsymbol{P}_{i,j} \neq \emptyset)) \tag{1}$$

$$\sum_i \boldsymbol{L}_i = (m - n) \tag{2}$$

[Plausible Design]

$$\boldsymbol{L}_i \leftrightarrow (\boldsymbol{P}_{i,1} \neq \emptyset) \tag{3}$$

$$l_{min} \leq \left((\sum_j (\boldsymbol{P}_{i,j} \neq \emptyset) \right) \leq l_{max} \tag{4}$$

$$(\boldsymbol{P_{i,j}} = p) \rightarrow (\boldsymbol{P_{i,k}} \neq q) \tag{5}$$

[Appealing Design]

[Service Diversity]

$$Q_r = \frac{\sum_{i,j} (\boldsymbol{P}_{i,j} = r)}{\sum_{i,j} (\boldsymbol{P}_{i,j} \neq \emptyset)} \tag{6}$$

$$(\boldsymbol{S} = \sum_i \boldsymbol{Q}_i^2) \geq \Phi \tag{7}$$

[High Vulnerability]

$$(\boldsymbol{P}_{i,j} = r) \rightarrow (\boldsymbol{\Gamma}_{i,j} = \gamma_r) \tag{8}$$

$$\boldsymbol{\Gamma}_i = (\sum_j \boldsymbol{\Gamma}_{i,j}) / (\sum_j (\boldsymbol{P}_{i,j} \neq \emptyset)) \tag{9}$$

$$\gamma_{min} \leq \boldsymbol{\Gamma}_i \leq \gamma_{max} \tag{10}$$

[SLA Compliance]

$$\Delta - \epsilon \leq \left(\sum_{i,j} \delta_j \cdot (\boldsymbol{P}_{i,j} \neq \emptyset) \right) \leq \Delta + \epsilon \tag{11}$$

5.1. PoC Implementation

We deployed a proof-of-concept (POC) implementation of the model in a testbed network. `VLAN` settings on `switch` was used to divide the network into three logical parts: (1) customer network, (2) HaaS network, and (3) user network. The HaaS gateway has three interfaces, each configured for one of the networks. HaaS gateway is implemented on a `Debian` platform and has three main components: `farp` daemon, `iptables`, and a *Planner* script. The gateway manages all the traffic between the external network and customer and honeypot cloud networks. A `farpd` daemon on the gateway enables the gateway to receive all the traffic, destined to customer and HaaS networks. The gateway acts as 1 : 1 NAT server, which is implemented by adding `DNAT` and `SNAT` rules to `iptables`. These `DNAT` and `SNAT` rules are inserted

by *Planner*. The planner is, in fact, a bash script which is executed periodically using a `cronjob` daemon. In each run, the planner script performs two specific operations. Firstly, it determines a new address mapping for all production hosts and honeypots. Secondly, it flushes all rules in `nat` table, and insert the new NATing rules into it based on the new mapping.

5.2. Effectiveness

We conducted our evaluation of the effectiveness of the HaaS model planning in two phases: first, we performed a small-scale evaluation with subject matter experts; and then, we generalized the results from the first stage in an analytical framework to evaluate the HaaS model effectiveness for large-scale scenarios.

Small-Scale Evaluation with Human Subjects. Using our Proof-of-concept implementation, we created a network with a class `C` address space, consisting of following host types.

- **Real**: 20 VMs were designated as real hosts.
- **Low-interaction**: 5 low-interaction virtual honeypots. These honeypots were created using `honeyd` [1].
- **Poorly-designed**: 5 high-interaction honeypots with implausible (unbelievable) platforms and services; these honeypots included services that were either outdated (`Windows XP`), too vulnerable (an unpatched`Print Spooler` service), or too different (`Plex Media Server`) in the context of the network.
- **Misconfigured**: 5 proxy honeypots that were designed using our planner, but services had weak configurations (FTP server that allows anonymous users), and dummy data (Web app included only a few HTML files or FTP server that hosted text files with dummy contents).
- **HaaS**: 5 proxy honeypots that were designed using our planner, and configured properly. These proxy honeypots were generated as redirection to 3 VMs that included services from several classes, including FTP servers, SSH servers, sharing services (SMB), Web Servers, DNS servers, and mail servers, hosted on different versions of `Windows` and `Linux` operating systems.

Six security experts with similar hands-on experience in penetration testing were recruited to this aim. Participants were asked to determine for each of the 40 hosts in the address space if each one is *real* or *honeypot*. The participants could use any tools or script to probe machines in order to determine whether the machine is real or emulated; whether the machine OS and remotely-accessible services have versions and vulnerability levels that are believable (*e.g.*, not outdated); whether applications (*e.g.*, Web app) and their configuration (*e.g.*, SSH credentials) look real (not too vulnerable); and also whether accessible applications' contents look realistic.

In addition, for each host participants announced the amount of time they spent on reaching their decision, in addition to justifying their choices. The number and distribution of real and different honeypot types were not disclosed to participants.

Figure 4 shows the number of honeypots from each type that were correctly distinguished by participants for different honeypot configuration scenarios. For each scenario, the figure shows the minimum and the maximum number of detected honeypots among all experts as well as the averages and median of reported times. Accordingly, Figure 5 shows the times that participants announced to have spent on distinguishing each honeypot type (whether correctly or incorrectly); and finally Figure 6 shows the probabilities of missing a honeypot for each scenario.

Note that on average, each participant correctly identified low-interaction honeypots in an average time of 6 minutes for each honeypot, with a mean miss probability of 0.07. For poorly-designed type, note that on average almost 3 out of 5 honeypots were distinguished correctly by participants, resulting in a miss probability of 0.21. Also, probing time increased to 33 minutes, because participants needed more probes and fingerprinting in order to make their decisions. For most correct identifications, participants' justified their choice by marking services that they found unexpected, either because they included platforms/services that were too outdated, too vulnerable, or "unrealistic" to be executed by a production host.

For misconfigured type, on average slightly less than 2 hosts were correctly identified by participants, resulting in a miss probability of 0.56, and probing time of 43 minutes. Based on participants' reports, correct cases were identified by distinguishing (1) meaningless and dummy contents and (2) unrealistically-weak configurations.

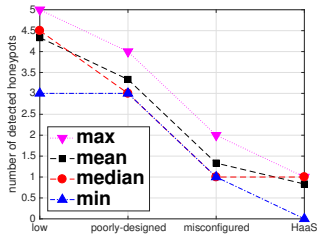

Figure 4. No. of distinguished honeypot based on type

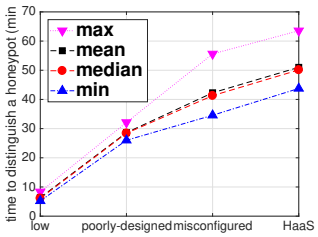

Figure 5. Time to distinguish honeypots based on type

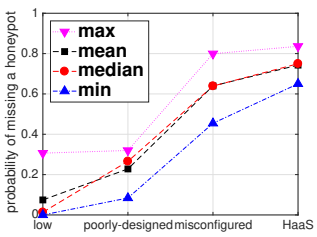

Figure 6. Prob. of missing a honeypot based on type

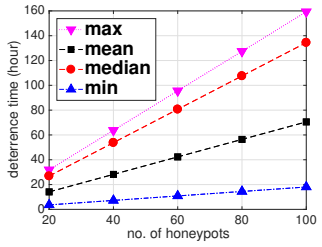

Figure 7. Deterrence in attack completion time

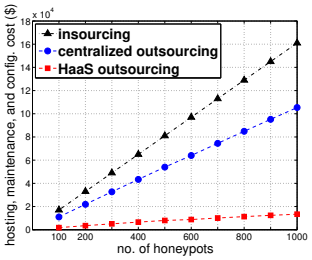

Figure 8. Expected cost for various honeypot deployment models

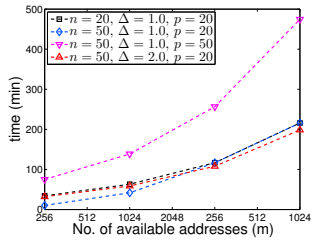

Figure 9. Planning overhead for various scenarios

For the HaaS type, the results were promising. Participants, on average, identified less than 1 honeypot correctly, resulting in a miss probability of 0.75. Probing, on average, required 57 minutes, which was higher than all other cases. This significant miss probability proves that our honeypots are indeed effective in providing a deception that is plausible even to elite attackers.

Large-Scale Evaluation using Simulation. Building on these experimental results with human subjects, we used simulation to understand the effect of our model on prolonging network attacks. We simulated the following well-known network scanning threat model: assume a network is populated with 20 real hosts, and h honeypots of different types. An attacker is stealthily attempting to compromise network hosts in the address space. To this aim, they (1) first attempts to distinguish honeypots from real hosts and blacklist them, and (2) then attempts to compromise those hosts they identified as real.

Assuming the experimental miss probability, and assuming an attacking time of 60 minutes to compromise a real host, we simulated the time that a skilled attacker would need in order to attack all real hosts in the network. This deterrence emanates from the fact that the attacker would not be able to distinguish a HaaS honeypot from a real system, thus squandering their time on attacking them. Figure 7 shows how long such attacker is *deterred* in scenarios with different honeypot types and numbers. Firstly, note that for a network with 20 production hosts, deploying 20 low-interaction honeypots would only deter a skilled attacker for 4 extra hours. Even deploying 100 such honeypots do not slow down the attack more than 18 hours. In reality, we expect this time to be much lower, because emulated services on low-interaction honeypots are very limited in number and type, and as the attacker's experience increases, identifying low-interaction honeypots becomes very straightforward.

Secondly, deploying 20 effective honeypots can slow down attack up to 32 hours. In contrast, this number increases to 159 hours for 100 effective honeypots. This 5-times increase is linear with the 5-times increase in the number of honeypots. This suggests that the increase in attack's completion time is *linear* to the number of honeypots. However, there is an upper limit on the number of honeypots that could be deployed in any given subnet, which is determined by the number of available (unused) IP addresses in the address ranges assigned and routed to that subnet. Given the limited availability of addresses in IPv4 address family, this upper bound could limit the applicability of the HaaS model. While the number of available addresses is considered as an input to the HaaS planning formulation, more investigation is required to understand the effect of this on achieving desired

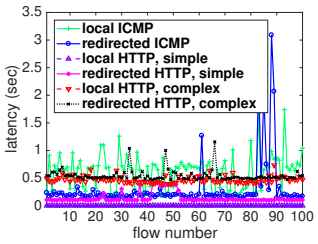

Figure 10. Comparing RTT for local and redirected flows

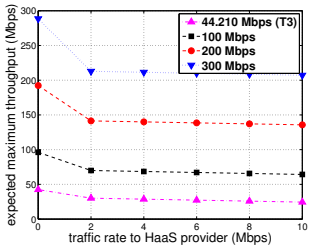

Figure 11. Network throughput for various traffic rates

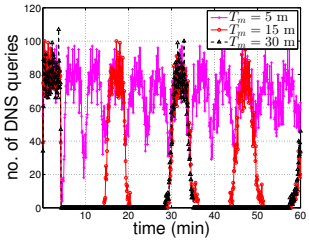

Figure 12. No. of DNS queries for various mutation intervals

effectiveness in scenarios with a very limited number of available addresses.

5.3. Overhead

In this section, we evaluate the primary overheads of the HaaS model.

Deployment Cost. Deploying effective honeypots incurs high cost in terms of hosting, design, and maintenance. Figure 8 provides an average estimation of deployment costs for three models: (1) insourcing honeypot deployment inside a customer's network; (2) outsourcing honeypots to a centralized provider but without proxy honeypots, honeyfarm [9], and (3) outsourcing to a HaaS provider. The following assumptions are made for deriving these estimations: 1000$ per each hardware machine, a yearly cost of 1000$ for maintaining each machine, where each machine has the capability of hosting 10 VMs, and each service are shared only with 5 customers, and each honeypot includes 4 services on average. Compared to the insourcing model, both centralized deployment and the HaaS model are more affordable. However, since the HaaS model shares honeypot services among customer networks, the overall cost of deployment is considerably less than the centralized model.

Planning Overhead. We use the `Z3` SMT solver [3] on a Quad Core processor (`3.3GHz`, `8M cache`) and `16GB DDR3 RAM`. Figure 9 shows the required time for solving the SMT instance, given various settings and network sizes. For all tested scenarios, unless stated otherwise, the model is solved for a network with $n = 20$ hosts, $p = 20$ ports, and $\Delta = 1$ Mbps admissible traffic rate to the HaaS provider network. Firstly, note the running time is still affordable even for fairly large network sizes, especially since the SMT instance is only solved once. Secondly, note that the running time is exponential with regard to the address space size, m. Also, the number of considered service classes (p) has a significant effect on running time. This is because the running time of the SMT instance largely depends on the number of variables, which in our model is of order $\theta(m \cdot p)$. The number of production hosts (n) has a negligible effect on running time.

Latency and Throughput. Figure 10 shows the round-trip time (RTT) of ICMP and TCP flows for both local (customer) and remote (HaaS provider) networks. This RTT includes propagation, queuing, processing and other delays at routers and end hosts. Simple TCP flows are basically HTTP requests to retrieve a simple HTML file from a `Apache` Web server. In contrast, complex TCP flows denote HTTP requests to a database-backed `PHP` application that requires non-trivial processing. As shown in the Figure, RTT of flows that are redirected to the HaaS provider network is slightly higher than that of local flows. This difference becomes less noticeable for HTTP flows. To disallow latency analysis to differentiate proxy honeypots from real hosts, HaaS gateway will include a latency emulator module which buffers outgoing packets of randomly chosen flows (in our example, $\leq$ 20%) for a short period of time (in our example, between $100 - 200$ msec). Assuming an average latency of 150 msec for 20% of production flows, Figure 11 shows the expected maximum throughput that is achieved for networks with various bandwidths and traffic rates to HaaS provider network.

DNS Overhead. When the address of a host is mutated, the new IP address must be reflected in DNS responses that are issued by the authoritative DNS of that host. Since TTL values of DNS queries are small, the authoritative DNS has to handle more DNS queries. Using our PoC testbed, we measured the DNS overhead for mutation intervals of 5, 15, and 30 minutes. Figure 12 shows the number of queries destined to the authoritative DNS of a HaaS-protected network with 20 externally-reachable server and

1,000 clients. The average inter-arrival time for the requests made by each client is 5 seconds. In the Figure, note that higher mutation rates result in a higher number of DNS queries. This clearly represents the trade-off between the benefit that is achieved as a result of faster mutations, and the cost of handling a larger number of DNS queries.

6. Conclusion

In this paper, we presented honeypot-as-a-service (HaaS) model, a *proactive* security paradigm which aims to make honeypots both effective against skilled attackers, and affordable by enterprises. To this aim, we defined and formulated the problem of designing and configuring effective honeypots, and offered architecture, communication protocols and implementation guidelines for realizing the HaaS model. Our evaluation shows that honeypots generated by HaaS honeypots were misidentified as real production hosts with a likelihood of almost 80%; and for some investigated scenarios our model results in prolonging attacks for up to 160 hours.

References

[1] Niels Provos et al. A virtual honeypot framework. In *USENIX Security Symposium*, volume 173, 2004.

[2] Xuxian Jiang and Xinyuan Wang. out-of-the-box monitoring of vm-based high-interaction honeypots. In *Recent Advances in Intrusion Detection*, pages 198–218. Springer, 2007.

[3] Microsoft. *Z3: An Efficient Theorem Prover*, 2012. `http://research.microsoft.com/en-us/um/redmond/projects/z3/`.

[4] Tomas Sochor and Matej Zuzcak. Study of internet threats and attack methods using honeypots and honeynets. In *Computer Networks*, pages 118–127. Springer, 2014.

[5] Xinwen Fu, Wei Yu, Dan Cheng, Xuejun Tan, Kevin Streff, and Steve Graham. On recognizing virtual honeypots and countermeasures. In *Dependable, Autonomic and Secure Computing, 2nd IEEE International Symposium on*, pages 211–218. IEEE, 2006.

[6] Xinyuan Zhang and Lianqing Zheng. Delude remote operating system (os) scan by honeyd. In *Computer Science and Engineering, 2009. WCSE'09. Second International Workshop on*, volume 2, pages 503–506. IEEE, 2009.

[7] Kyi Lin Lin Kyaw. Hybrid honeypot system for network security. *International Journal of Computer, Electrical, Automation, Control and Information Engineering*, 2(12):4085–4089, 2008.

[8] Lance Spitzner. Honeypot farms, 2003.

[9] Xuxian Jiang, Dongyan Xu, and Yi-Min Wang. Collapsar: A VM-based honeyfarm and reverse honeyfarm architecture for network attack capture and detention. *Journal of Parallel and Distributed Computing*, 66(9):1165–1180, 2006.

[10] Pragya Jain and Anjali Sardana. Defending against internet worms using honeyfarm. In *Proceedings of the CUBE International Information Technology Conference*, pages 795–800. ACM, 2012.

[11] Pragya Jain and Anjali Sardana. A hybrid honeyfarm based technique for defense against worm attacks. In *2011 World Congress on Information and Communication Technologies*, pages 1084–1089. IEEE, 2011.

[12] F Pouget, M Dacier, VH Pham, et al. On the advantages of deploying a large scale distributed honeypot platform. In *proceedings of the e-crime and computer evidence conference*, 2005.

[13] Jakub Safarik, MIroslav Voznak, Filip Rezac, Pavol Partila, and Karel Tomala. Automatic analysis of attack data from distributed honeypot network. In *Mobile Multimedia/Image Processing, Security, and Applications 2013*, volume 8755, page 875512. International Society for Optics and Photonics, 2013.

[14] Rahmat Budiarto, Azman Samsudin, Chuah Wee Heong, and Salah Noori. Honeypots: why we need a dynamics honeypots? In *Information and Communication Technologies: From Theory to Applications, 2004. Proceedings. 2004 International Conference on*, pages 565–566. IEEE, 2004.

[15] Daniel Fraunholz, Marc Zimmermann, and Hans D Schotten. An adaptive honeypot configuration, deployment and maintenance strategy. In *2017 19th International Conference on Advanced Communication Technology (ICACT)*, pages 53–57. IEEE, 2017.

[16] Iyad Kuwatly, Malek Sraj, Zaid Al Masri, and Hassan Artail. A dynamic honeypot design for intrusion detection. In *Pervasive Services, 2004. ICPS 2004. IEEE/ACS International Conference on*, pages 95–104. IEEE, 2004.

[17] Christopher Hecker, Kara L Nance, and Brian Hay. Dynamic honeypot construction. In *10th Colloquium for Information Systems Security Education, University of Maryland, USA*. Citeseer, 2006.

[18] Fiorella De Rosis, Valeria Carofiglio, Giuseppe Grassano, and Cristiano Castelfranchi. Can computers deliberately deceive? *Computational Intelligence*, 19(3):235–263, 2003.

[19] Neil C Rowe. A model of deception during cyber-attacks on information systems. In *Multi-Agent Security and Survivability, 2004 IEEE First Symposium on*, pages 21–30. IEEE, 2004.

[20] Vinod Yegneswaran and Chris Alfeld. Camouflaging honeynets. In *In Proceedings of IEEE Global Internet Symposium*, 2007.

[21] Neil C Rowe. Counterplanning deceptions to foil cyber-attack plans. In *Information Assurance Workshop, 2003. IEEE Systems, Man and Cybernetics Society*, pages 203–210. IEEE, 2003.

[22] Yeu-Pong Lai and Po-Lun Hsia. Using the vulnerability information of computer systems to improve the network security. *Computer Communications*, 30(9):2032–2047, 2007.