Multi-dimensional Host Identity Anonymization for Defeating Skilled Attackers

Jafar Haadi Jafarian, Amirreza Niakanlahiji, Ehab Al-Shaer, Qi Duan
Department of Software and Information Systems
University of North Carolina at Charlotte
Charlotte, NC, USA
{jjafaria, aniakanl, ealshaer, qduan}@uncc.edu

ABSTRACT

While existing proactive-based paradigms such as address mutation are effective in slowing down reconnaissance by naive attackers, they are ineffective against skilled human attackers. In this paper, we analytically show that the goal of defeating reconnaissance by skilled human attackers is only achievable by an integration of five defensive dimensions: (1) mutating host addresses, (2) mutating host fingerprints, (3) anonymizing host fingerprints, (4) deploying high-fidelity honeypots with context-aware fingerprints, and (5) deploying context-aware content on those honeypots.

Using a novel class of honeypots, referred to as *proxy* honeypots (high-interaction honeypots with customizable fingerprints), we propose a proactive defense model, called (HIDE), that constantly mutates addresses and fingerprints of network hosts and proxy honeypots in a manner that maximally anonymizes identity of network hosts. The objective is to make a host untraceable over time by not letting even skilled attackers reuse discovered attributes of a host in previous scanning, including its addresses and fingerprint, to identify that host again. The mutations are generated through formal definition and modeling of the problem.

Using a red teaming evaluation with a group of white-hat hackers, we evaluated our five-dimensional defense model and compared its effectiveness with alternative and competing scenarios. These experiments as well as our analytical evaluation show that by anonymizing all identifying attributes of a host/honeypot over time, HIDE is able to significantly complicate reconnaissance, even for highly skilled human attackers.

1. INTRODUCTION

Network reconnaissance, as a recurring process of identifying, enumerating and fingerprinting network hosts, is a precursory step to majority of advanced external and insider attacks against enterprise networks [1, 2]. Reconnaissance is performed by gathering information from a series of scanning probes [3], where the objective is to identify attributes of network hosts, including their names, interfaces and their associated MAC/IP addresses, and fingerprints (operating system, open ports and running services). Collectively, these individual information pieces could be envisioned as a hypothetical dataset that describes network hosts.

Given this hypothetical dataset, we can restate the problem of defeating adversarial reconnaissance as dataset anonymization [4], *i.e.*, anonymizing every attribute of a host that could serve as an identifier or quasi-identifier [4] for that host. Achieving this basically means that the reconnaissance data which have been collected about a host could not be used later to identify that host.

The need for this anonymization has been the underlying justification for a novel class of proactive defense techniques based on moving target defense (MTD) paradigm, referred to as address mutation/randomization. These approaches mainly focus on anonymizing host addresses by mutating them over time [1, 5, 6]. However, while address anonymization is sufficient to defeat automated and naive attackers, skilled adversaries could still use other identifying attributes of a host, especially a host fingerprint at network or application level, to identify it later. For example, if the adversary knows that the target host is the only one that is running a specific distribution of `Apache` Web server on port 80, she can use this to identify the target host, even after its address is mutated.

In this paper, we show that effective anonymization of host identities against reconnaissance by skilled attackers entails synergistic composition of five dimensions: (1) mutating host addresses and domain names (only against reverse-DNS queries) [1], (2) anonymizing host fingerprints, using the concept of k-anonymity from data privacy [4], (2) mutating host fingerprints, (4) deploying high-fidelity (high-interaction and context-aware) honeypots and (5) placing context-aware and believable contents and data on these honeypots.

To this aim, we propose a proactive defense model, called Host IDEntity anonymization (HIDE), that integrates these five dimensions in a way that synergistically anonymizes host identities in a transparent manner and without affecting regular operations of the network; *i.e.*, breaking active sessions or manipulating production traffic. Specifically, for each interval HIDE updates attributes of production hosts, including their addresses, name, and fingerprints, both in public (DMZ) and internal address space. For each interval, it also populates these spaces with a set of honeypots with fingerprints that are carefully designed to anonymize identity of production hosts. As part of our work, we formally define strategies for generating fingerprints that satisfy the

MTD'16, October 24 2016, Vienna, Austria

DOI: http://dx.doi.org/10.1145/2995272.2995278

anonymization criteria. We model this NP-complete problem as a constraint satisfaction problem, and solve it using off-the-shelf Satisfiability Modulo Theories (SMT) solvers [7].

To realize our model in an enterprise network, we rely on *proxy* honeypot services; high-interaction platform/services that are hosted on a collection of honeypot machines in a consolidated network called *honeypot cloud*. Accordingly, we define the notion of *proxy honeypot* as an imaginary machine that is pretended to be located on a specific address, but is merely a set of meaningful redirections to platform and services that may physically belong to different machines in honeypot cloud. Using customizability of proxy honeypots, we also introduce the concept of *high-fidelity* honeypot as a decoy host with configuration and behavior that is indistinguishable from a production host, both individually and with regard to the mission and context of its surrounding network.

To realistically evaluate effectiveness of HIDE in deceiving human attackers, we performed a thorough red teaming experiment with six participants with penetration testing and ethical hacking backgrounds. Our objective was primarily to understand how HIDE affects reconnaissance activities of skilled human attackers, and also how indistinguishable (believable) our high-fidelity honeypots are to these attackers. These experimentations, as well as our analytical modeling, confirms that our five-dimensional defense strategy is very effective against human attackers. Moreover, our assessment shows that the proposed high-fidelity honeypots are not distinguishable from production hosts. Mutation of fingerprints and addresses resulted in effective anonymization of both production and honeypot hosts across subsequent intervals. This constantly invalidated participants' previous reconnaissances. Using a metric called *mean time to compromise*, we show that even in a small testbed, HIDE deployment significantly deters attack progression and completion.

2. RELATED WORK

Proactive techniques for disrupting reconnaissance could be broadly divided into two categories: MTD-based and deception-based approaches. Moving target defense relies on randomizing static system parameters, to invalidate attacker's reconnaissance information. In this direction, several approaches have been proposed [1, 8, 9] for mutation of IP addresses, especially over time. NASR [5] relies on DHCP to randomize IP addresses over time. RHM [1] and OF-RHM [8] propose using DNS for address randomization without changing the actual IP address of network hosts, in order to prevent TCP/UDP sessions from being broken. RHM [1] identifies additional need for mutating MAC addresses to defeat LAN-level reconnaissance, and mutating domain names to counter reverse-DNS queries. While these works are effective against automated scanners and worms, in our evaluation we show that they have limited effectiveness against human attackers.

In contrast, *deception-based technologies*, rely on luring attackers away from critical hosts, in order to prolong adversarial reconnaissance and attack progress. It has been shown that honeypots are highly effective in slowing down worm breakouts, and decreasing amount of attacks on production hosts [10].

Budiarto [11] propose the idea of dynamic honeypots, which passively [12] or actively [13] discover production systems on a network and uses this information to create honeypots that are similar to production hosts and blend in the surrounding network. Anagnostakis *et al.* [14] proposes *shadow* honeypots, an identical copy of production servers, as an strategy to deploy honeypots in a production environment. Cohen *et al.* [15] investigate several intuitive honeypot configuration strategies, including shadow honeypots, and shows their effectiveness in deceiving attackers via thorough red teaming experiments with human subjects.

Some recent works have realized the need for using IP mutation to defeat honeypot mapping attacks [16–18]. These attacks aim to detect and blacklist honeypots in a target address space before initiating the attack, using various probing methods [19, 20].

Authors in [6] correctly identify the need for strengthening address mutation with decoy services against sophisticated adversaries, but their approach neglects important theoretical aspects of anonymization such as mutation and anonymization of host/honeypot fingerprints. Moreover, efficacy of their proposed model is not tested against real human attackers.

In summary, none of the existing approaches recognizes the need and synergy of combining the five-dimensional proactive defense for defeating adversarial reconnaissance performed by skilled adversaries.

3. METHODOLOGY

In this section, we first present the threat model addressed by our approach, as well as the defense objective against this threat model. Then, we provide an overview of HIDE architecture, protocols, and algorithms.

3.1 Threat Model, Objectives, and Metrics

3.1.1 Threat Model

According to the intrusion kill chain [21], reconnaissance is the primary and initial step of any advanced and persistent intrusion on enterprise networks. At network level, reconnaissance refers to the process of (1) discovering and enumerating network hosts, (2) scanning these hosts at network- and transport-level (detecting OS and open ports), or application-level (identifying services names), and (4) discovering exploitable vulnerabilities for each host [3].

If attacker knows name of a host, she will obtain its IP address from DNS and proceed to attacking that host. These classes of attacks are not the focus of our approach.

However, usually only a few domain names for public servers are known to attackers, and these hosts may not be exploitable. More importantly, intrusion attacks are usually multi-staged; they are usually initiated with compromising a vulnerable public server, and moving inward toward critical assets deep inside the target network [1]. Even though domain name for public servers might be known, domain names for internal servers are usually unknown to external parties.

Therefore, in order to discover other hosts (especially internal ones) in the enterprise network, the attacker needs to scan the address space, which results in issuing probes to non-existing addresses and closed ports. This behavior is obviously in contrast with how a regular user communicates with a server, which is initiated by obtaining the server's IP address from its authoritative DNS. However, after discovering an active IP address, the attacker may also issue

a reverse-DNS query to obtain the domain name from the host's address.

After identifying active hosts, the attacker uses further reconnaissance to identify host fingerprints and vulnerabilities [3]. This information is usually achieved by aggregating individual information pieces derived from a set of network probes that could be generated by tools such as `Nmap` and `Nessus` [3].

While address mutation and rDNS name mutation [1] are disruptive to automated worms and naive adversaries, our discussion will be focused on raising the bar against skilled human adversaries, who would use any available information such as a host's fingerprint to re-identify that host after its addresses have been mutated. In fact, while mutating addresses of a host would significantly complicate adversarial reconnaissance, a host identity can still be recognized and traced by its fingerprint. In other words, a skilled attacker can fingerprint a host and use that fingerprint in future to distinguish that host, even after its address is mutated. For example, if adversary knows that the target host is the only host in the network that is running a specific distribution of `Debian` OS with an `Apache` Web server on port 80, she can use this information to identify that host, even after its addresses are mutated.

3.1.2 Defense Objectives

In a conventional network, the reconnaissance stage of the kill chain [21] occurs only once, because attacker's search space constantly shrinks: once attacker scrutinizes a host and fails to compromise it (*i.e.*, she finds no exploitable vulnerability on it), she does not need to re-probe that host again. Therefore, attacker's sampling from address space is random *without replacement*, because her search space constantly shrinks. The idea of address mutation techniques is to disallow adversary's search space from shrinking, by rapid changing of their network addresses. However, a skilled human attacker still can use other identifying attributes of a host to recognize and avoid probing already-probed hosts, thus shrinking her search space.

If every time an attacker aims to randomly select a host for probing, all hosts (production or honeypot) have equal probability of being chosen, then the distribution that attacker selects her next target changes to uniform sampling *with replacement*. This is achieved when (1) attacker's gained knowledge does not shrink her search space (information gathered in previous reconnaissance is not usable), and (2) attacker is not able to distinguish honeypots from production hosts throughout the reconnaissance.

The former objective, disallowing attacker's knowledge from shrinking her search space, could be violated in three specific ways:

- **Address mutation** refers to the act of changing host addresses over time. With no *address mutation*, attacker can use a host IP/MAC addresses to identify it and avoid re-probing that host. Correspondingly, with no *rDNS name mutation*, the attacker can issue a reverse DNS, discover domain name of a host, and use that name to identify that host later.

- **Fingerprint anonymization**: refers to the notion of hiding a host fingerprint in a pool of honeypots with same fingerprints, thus not letting a skilled adversary trace a host by its fingerprint. With no host fingerprint anonymization, a skilled attacker can use a potentially unique fingerprint of a host to identify it later.

- **Fingerprint mutation** refers to the act of mutating fingerprints of network hosts over time, thus not letting attacker identify groups of hosts with same fingerprints. More details on this is provided in Section 3.3.

The latter objective, making honeypots indistinguishable, especially to skilled human attackers could only be achieved when honeypots are not differentiable from network hosts. Our investigation and experimentation with real human attackers show that to achieve such indistinguishability, a honeypot must have *high-fidelity* (a) in compliance with network protocols (traditionally called high-interaction), (b) within the context of its surrounding network (context-aware configuration), and (c) in providing believable and meaningful application content/data (context-aware content):

- **High-interaction**: high-interaction honeypots [22] are usually implemented as real system, consisting of a full-fledged operating system and unmodified services. In contrast, low-interaction honeypots simulate a limited subset of the functionality of a real system [23].

- **Context-aware configuration**: honeypots must be configured with regards to the specificities of their networks. To this aim, the type and number of platform and services must be believable and consistent with respect to the mission and configuration of the target network. For example, an `IIS` Web server in a `Unix`-based enterprise is very distinguishable. Or, a host that is running too many services stands out from production hosts.

- **Context-aware content**: after fingerprinting a host (detecting its OS, service names and versions), a skilled attacker would interact and fingerprint the content/data of a honeypot service in an attempt to compromise it. For example, the attacker would interact with a Web application to see if it can find a SQL injection or XSS vulnerability on it. If the same content is located on multiple honeypot services, or this content does not correspond to the type and mission of the target network, it makes that honeypot easily distinguishable. For example, in an educational domain, honeypot Web applications must have contents that is believable in that context. Therefore, honeypot services must have context-aware contents; which is (a) individualized and (b) consistent with the mission and type of the network. Generation of context-aware content for honeypot applications is a novel paradigm that is beyond the scope of this paper. However, we emphasize its necessity and also present a preliminary approach to this aim.

3.1.3 Metrics

Mean Time to Compromise (MTTC) [24] is a metric that measures the average time required to gain some privilege level on a target. Our primary objective is to understand how our approach impacts mean time to compromise (MTTC) of skilled human attackers, as compared to (a) legacy static networks, (b) honeypot-protected networks with

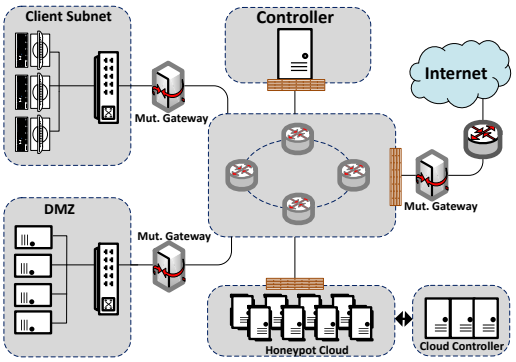

Figure 1: Architecture of the HIDE model

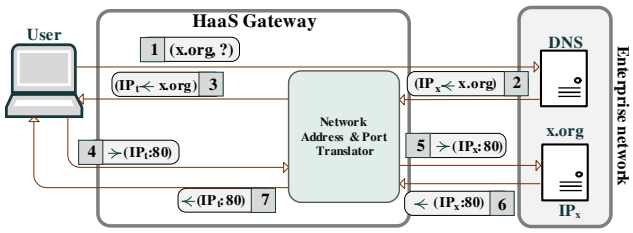

Figure 2: communication protocol of the HIDE model

no address mutation, (c) mutating networks with no honeypots, and (d) mutating network with arbitrary and static honeypots. Our secondary objective is to understand how increasing the mutation rate and number of honeypots would affect MTTC.

Mean number of attempted compromises is another metric that denotes the average number of hosts that attacker must attempt to compromise before discovering (and successfully compromising) her target in the address space. This number quantitatively shows whether attacker's search space has been shrinking during the attack or not. Effective anonymization is achieved when for a network with n hosts and h honeypots, this number is $n + h$. This means that attacker's sampling from address space is without replacement; *i.e.*, the search space is not shrinking.

3.2 Overview, Architecture, and Communication Protocol

Our approach mutates attributes of network hosts after relatively short durations (*e.g.*, 5-30 minutes), referred to as *mutation interval*. This includes externally-observable parameters of a host, including IP and MAC addresses associated with its network interfaces, domain name (that is provided to reverse-DNS queries), and its fingerprint (operating system, open ports and their corresponding service names).

Figure 1 depicts the architecture for deploying HIDE in a TCP/IP enterprise network. In this architecture, a central entity called *Controller* is responsible for determining new mutations for network hosts, and announcing them to *mutation gateways*, which are distributed entities located at the boundaries of physical subnets (between subnet switch and default router). These gateways act similar to NAPT (network address and port translation) devices, translating address/port pairs based on new mutations received from the Controller.

A mutation gateway is located in front of each critical subnet, including DMZ as well as internal subnets. Therefore, configuration of all public and internal network hosts are mutable. This is especially important for thwarting sophisticated insider threats, which are usually prone to less security and inspection [1].

For fingerprint mutation, new fingerprints are generated by forwarding flows which are destined to unused ports of production hosts or unused addresses to honeypot services in the *honeypot cloud*. Honeypot cloud is a consolidated subnet of high-end machines, each hosting a number of virtual machines. These VMs are running a diverse set of platforms and network services, including an instance of every platform/service that is being executed on production hosts. This cloud is maintained by a cloud controller that uses an underlying centralized virtualization framework, such as Vagrant [25], for unified and automated management of these virtual machines. The cloud controller uses VM introspection techniques (*e.g.*, VMScope [22], NFM [26]) to detect and handle infected VMs.

Using the underlying infrastructure, HIDE anonymizes configuration of all (production or honeypot) hosts. Specifically, for each interval, we determine a new *mutation plan*, which consists of (1) new addresses and names, and (2) new fingerprints for all production and honeypot hosts.

3.2.1 Address mutation and communication protocol

Mutation of IP/MAC addresses and domain names are performed as described in our previous work [1]. Mutating MAC addresses are straightforward and requires a layer-2 proxy that also handles ARP queries. Mutating domain names for countering reverse-DNS queries while allowing their use for logging and spam detection is performed by generating temporary but well-formed domain names [1].

Mutation of IP addresses is more challenging due to routing and addressing complexities. For IP mutation, we keep the actual IP addresses of real hosts unchanged. Instead, we select a short-lived temporary IP address from the unused ranges of the address space and provide it to clients via DNS. For public hosts, this address is chosen from the public address space, which may be limited due to lack of available addresses in IPv4. However, for private hosts this address is chosen from the private address space, which is usually sparsely populated [1].

Regular communication with network hosts occurs via their domain names. When a user queries an authoritative DNS for IP address of a host, the DNS reply provides the temporary IP address of that host to the user. Figure 2 provides a step-by-step description of how communication with a host via name occurs in our network.

In Figure 2, Network Address and Port Translator (NAPT) module is responsible for two tasks: (1) updating DNS replies that are issued by an authoritative DNS by changing the real IP address (IP_x in our example) to the current temporary one (IP_t); and (2) updating destination address-port pair for incoming traffic (IP_t to IP_x) and source address-port pair for outgoing traffic (IP_x to IP_t). This means that address mutation is completely transparent to DNS and end-hosts. Moreover, addresses are mutated before flow enters core of

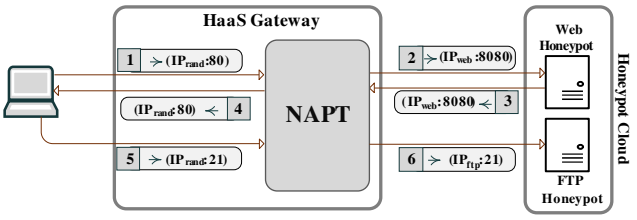

Figure 3: redirection of flows (destined to inactive address-port pairs) to honeypot cloud for generating proxy honeypots and new fingerprints

the network (at the edges), and therefore address mutations are transparent to routing devices.

3.2.2 Fingerprint mutation and generating proxy honeypots

In order to mutate fingerprints of a network host, we never manipulate traffic to actual production services. This is because, while several works have suggested sanitization and scrubbing of responses generated by production services for hiding or changing their fingerprints [27, 28], these changes are in essence low-interaction and thus easily detectable, since only responses to probes with known signatures are changed [29].

Therefore, new host fingerprints are generated by redirecting flows destined to closed or non-existent ports to predetermined machines in the honeypot cloud. For example, assume port 21 of a production server is closed. In one interval, the incoming traffic to this port may be redirected to a FTP service in the honeypot cloud, while in another the port may be announced as closed.

In addition to mutating fingerprints of production hosts, we also use this redirection to generate *proxy honeypots* on inactive IP addresses. These proxy honeypots are generated by redirection of incoming traffic to a specific address to different virtual machines in the honeypot cloud. For example, in Figure 3, while from 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 3 shows how this redirection occurs. Assume an attacker, as part of her reconnaissance, randomly selects an IP address IP_{rand} and probes it to discover if it is running a Web application on port 80. If attacker is lucky and IP_{rand} is assigned to a Web server for this mutation interval, then the communication proceeds as in Figure 2. However, if IP_{rand} is not assigned to a Web server, which is intuitively far more probable, the NAPT module redirects them to the predetermined Web application in the honeypot cloud, which resides on port 8080 of a VM called Web honeypot. Probing the same address on port 21 is redirected to another VM called FTP honeypot. Again, note that no changes is required to be applied to any legacy protocol or device.

3.3 Mutation Planning

3.3.1 Algorithm

In order to effectively anonymize host identities for each mutation interval, we need to formally define our criteria for generating anonymized fingerprints, as well as providing methodologies for generation of such fingerprints.

Given existing configuration of network hosts, available address space, and admissible capacity of honeypot traffic as input, the controller determines a new *mutation plan* for each interval, such that the following objectives are satisfied: (1) host configurations (names and fingerprints) are unpredictably anonymized; (2) generated honeypots are believable and context-aware (to satisfy high-fidelity requirements), and (3) the overhead is bounded. This mutation plan is generated by Controller for each interval, and announced to mutation gateways.

To this aim, Controller periodically probes internal and public address spaces of the network to determine changes in hosts' configurations (*e.g.*, new services installed), as well as to discover inactive addresses-ports in the public address space. This makes our mutation planning adaptive to changes in the context (network), and adding or removing new hosts or services would be immediately reflected in the planning. In this characterization, the controller would exclude exceptional cases such as ports used for peer-to-peer communications. For example, port range $6881 - 6889$ are used by `BitTorrent` and are excluded prior to planning.

The modeled problem is a constrained matching problem, which is known to be NP-complete. We model this problem as a satisfiability problem using generalized Boolean/arithmetic format of satisfiability modulo theories (SMT) [7]. An SMT instance is a generalization of a Boolean SAT instance in which various sets of variables are replaced by predicates from a variety of underlying theories. While there are some heuristics for constrained matching problem, SMT is natural to solve this class, because the SMT search algorithm naturally fits the matching problem.

3.3.2 Problem Definition and Formalization

Suppose the controller aims to determine the mutation plan for interval μ_t, for a network with n production hosts. The maximum number of hosts on the address space is denoted as m, which is basically the number of unused and available addresses. This means that the network can have up to m active hosts for mutation interval μ_t. Among these, n are real, and the remaining active ones are high-fidelity proxy honeypots.

Appendix I presents our SMT formalization for this problem. Assume $H = \{h_1, \ldots, h_m\}$ denotes the set of all potential hosts. Input value $real_i$ demonstrates whether h_i is real or not; *i.e.*, $(real_i = true)$ means that h_i is a production host. Network services are categorized into classes $\{S_1, \ldots S_z\}$; *e.g.*, `operating system`, `Web server`, `FTP server`, *etc.* Each S_i consists of all services of a certain class that are installed on any machine in honeypot cloud. For example, S_1 is the set of all operating systems in honeypot cloud. Except for OS, all other service classes are associated with a port number; *e.g.*, 80 for Web servers, and 21 for FTP services. Attribute $srv_{i,j}$ describes name of the service from class S_j that is running on host h_i. The input value $srv_{i,j}$ denotes the actual service of class S_j that is running on host h_i, whereas $\boldsymbol{plan}\langle srv_{i,j}\rangle$ denotes the service of class S_j that *will* appear to be running on h_i based on the mutation plan.

The objective of the model is to discover effective assignments to variables of form $\boldsymbol{plan}\langle srv_{i,j}\rangle$. This determines what OS and services must be advertised to be running on each (production or honeypot) host, in order to achieve the

given objectives. Input value $plan^s\langle srv_{i,j}\rangle$ $(s < t)$ denotes the advertised service of class S_j that *was* announced to be running on host h_i in mutation interval μ_s. We only mention superscript s when needed.

3.3.3 Objective I: Anonymizing Host Identities

The final product of the reconnaissance process could be assumed as a hypothetical dataset where each element characterizes a network host in terms of IP/MAC addresses, open/closed ports, service names, exploitable vulnerabilities, *etc.*

As a result, anonymizing this hypothetical dataset would defeat reconnaissance because attackers would not be able to associate previously collected data with any host, which basically makes this dataset useless. So, the problem of defeating reconnaissance could be restated as a *data anonymization* problem, where our objective is to anonymize any potential reconnaissance data, such that no host is individually identifiable in this dataset. This includes (1) anonymizing hosts' unique identifiers, including their addresses and names, as well as (2) anonymizing their fingerprints which may serve as quasi-identifiers [4]. Quasi-identifiers are pieces of information that are not of themselves unique identifiers, but are sufficiently well-correlated with an entity that they can be combined with other information to create a unique identifier [4].

Mutation of addresses and name are carried out as explained in Section 3.2. To address the problem of quasi-identifiers in data privacy, the concept of k-anonymity [4] is defined and enforced on the data. A release of data is said to have the k-anonymity property if the information for each entity contained in the release cannot be distinguished from at least $k-1$ entities whose information also appear in the release. For our problem, the k-anonymity means that the fingerprint of a (production or honeypot) host must be the same as at least $k-1$ other hosts at any point. This idea also coincides with the concept of *shadow* honeypots [14], which has empirically shown to be effective for attack slowdown. Since honeypot cloud includes at least one instance of every real service in the network, k-anonymity can definitely be achieved. In Appendix I, Eq. 6 defines fingerprint equality for two hosts h_i and $h_{i'}$. Eq. 7 guarantees that for each host there exists at least $k-1$ other hosts with same fingerprints.

In addition to being a quasi-identifier, fingerprint of a host could serve as an identifier for the group of k hosts with same fingerprints. If fingerprint of a host is not changed after each interval, a skilled attacker would be able to immediately narrow down search for a specific fingerprint to these k hosts. Therefore, for a group of k hosts not to be recognizable in consecutive mutation intervals, the fingerprint of each host must be different from its fingerprint in the previous interval. We consider fingerprint of a host to be significantly different from its previous fingerprint when at least half of its services are different from its previous fingerprint.

For production hosts, this difference is established only by assigning new services to their closed ports. In Appendix I, Eq. 8 counts the number of different services between current fingerprint and host h_i's fingerprint in previous interval (μ_{t-1}); and eq. 9 ensures that this number is not less than a threshold α.

Another factor contributing to defeating skilled reconnaissance is the number of honeypots. Having more honeypots increases the probability that a probe hits a honeypot rather than a production host, thus slowing down attack progression.This number depends on the number of available addresses, and is discussed in the overhead Section (3.4).

3.3.4 Objective II: Generating Context-aware Honeypots

In section 3.1, we defined high-fidelity honeypots as ones that are camouflaged so well that even skilled attackers would not be able to distinguish them from production hosts. One of the criteria for such honeypots is context-aware configurations; configuring a group of honeypots with respect to specific properties of the network in which they are being deployed, in a way that seamlessly blends them in the surrounding environment.

Believable Fingerprints: since proxy honeypots are generated by redirection to a group of platform and services that may be physically running on different VMs, it is necessary for our mutation planning to generate believable honeypots. To this aim, (a) every proxy honeypot must have an operating system (Eq. 13). More importantly, (b) it must include services that are feasible. Unfeasible service pairs are services that cannot occur in the same fingerprint together. For example, assigning `IIS` to a proxy honeypot with `Debian` OS is not believable. In Appendix I, Eq. 15 ensures that unfeasible service pairs are not assigned to the same host.

Context-aware Fingerprints: the proxy honeypots must conform to the specificities of the enterprise network. To this aim, (a) each honeypot must only include services that are believable based on the mission and type of the network in which they are deployed. For example, having a `Linux` OS in a network with `Windows` servers is out of context and suspicious. In Appendix I, Eq. 12 ensures that all assigned services of type S_j belong to the pre-computed set of services of this type that are acceptable within the context of this network, denoted as M_j. Also, (b) every honeypot must include a believable number of services. Eq. 14 ensures that the number of services on each host is between minimum (l_{min}) and (l_{max}) number of services in the network.

3.4 Objective III: Bounded Overhead

Following capacity constraints ensure that the mutation plan does not violate budgetary limitations of the network.

Bounded service sharing: intuitively, the more proxy honeypot services represented by one physical honeypot service, the higher the resource requirement of that service. Therefore, for each service we constrain the maximum number of proxy honeypots that can share (redirect to) that physical service. Specifically, for $s_l \in S_j$, a threshold π_l is defined as the maximum number of proxy honeypots that can share that physical service. Thus, if more than π_l proxy services of type s_l are required, more than one physical honeypot VM must include s_l. For each physical service s_l, the input value ψ_l denotes the number of physical honeypots that offer service s_l. Eq. 16 ensures that the number of proxy services of type s_l is less than the given capacity of that physical service.

Bounded traffic rate: admitting honeypot traffic into the network increases load on network links and routers, thus incurring nontrivial cost on the network. Therefore, mutation planning must bound the number of honeypot services, by considering the flow rate that is expected to be processed by each service class. For a service class S_j, the input value

δ_j determines the expected flow rate to services of that class S_j, which is estimated based on several factors, including experiences with similar Enterprises in terms of size and mission, as well as current dark and production traffic observed at the network.

Given the expected flow rates for each service class, the planner must ensure that the average expected traffic rate to honeypot cloud does not surpass threshold Δ. Moreover, since having more services (and thus more honeypots) contributes to defeating reconnaissance, we must ensure that the expected traffic is close to the threshold. Eq. 17 ensures that the expected flow rate remains within the ϵ bound of the threshold Δ.

4. EVALUATION

In this section, we present our results on evaluating effectiveness of the model, which are mainly achieved through real experiments with white-hat human attackers. We also evaluate planning overhead of the model.

4.1 PoC Implementation

We deployed a proof-of-concept (PoC) implementation of the model in a testbed network. We used `VLAN` settings on a regular `switch` to divide the network into three logical parts: (1) enterprise network, (2) honeypot network, and (3) clients (and attackers) 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 all subnets. A `farpd` daemon on the gateway enables it to receive all the traffic destined to enterprise and honeypot 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 *Controller*. The Controller is in fact a bash script which is executed periodically using a `cronjob` daemon. In each run, the controller 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.

4.2 Effectiveness: Theoretical Analysis

Every time an attacker chooses a host for attacking, she requires a certain amount of time to (1) fingerprint its OS and services, and (2) attempts to exploit its vulnerabilities. For the sake of our analysis, we assume that the attacker would be able to compromise the target once she selects it and attempts to compromise it. We also assume that the attack finishes after compromising the target.

Assume T_a denotes the average time a skilled attacker would need to perform these steps. Also, assume that host addresses and fingerprints are mutated every T_m minutes. An attacker can on average scan $l = \lfloor T_m/T_a \rfloor$ hosts in each mutation interval. During an interval, attacker's knowledge regarding which hosts have already been scanned is not expiring. In other words, attacker's search space shrinks within an interval.

Since attacker cannot differentiate honeypots from hosts, and also she cannot use reconnaissance from previous intervals in this interval, the probability that attacker misses the target in one interval in a network with n production hosts and h honeypots is:

$$\prod_{i=0}^{l-1}(1-\frac{1}{n+h-i})$$

Therefore, the probability that the target is compromised after attacker has probed k hosts would be:

$$P_{HIDE}(k,n,h,T_m) = (\prod_{i=0}^{l-1}(1-\frac{1}{n+h-i}))^{p-1}. (\prod_{i=0}^{k'-2}(1-\frac{1}{n+h-i}))(\frac{1}{n+h-(k'-1)}) \quad (1)$$

where $l = \lceil \frac{T_m}{T_a} \rceil$, $p = \lfloor \frac{k}{l} \rfloor$ and $k' = k-(p-1)l$.

In a conventional network with no mutation, but with $h > 0$ high-fidelity honeypots, referred to as **C-1** network, the compromise probability is:

$$P_{C_1}(k,n,h) = (\prod_{i=0}^{k-2}(1-\frac{1}{n+h-i}))(\frac{1}{n+h-(k-1)}), \quad k < (n+h) \quad (2)$$

In a static network with no mutation and no honeypots, which we refer to as **C-0** network, the same compromise probability is:

$$P_{C_0}(k,n) = P_{C_1}(k,n,0) \quad (3)$$

For a network only with address mutation, referred to as **C-2** network [1], compromise probability depends on the skill level of the attacker. For naive attacks such as automated attacks or script kiddies, address mutation is effective enough for anonymizing host identities. However, for a skilled attacker, this is the same as a static network, because host identities are still recognizable via host fingerprints:

$$P_{C_2}(k,n,t_m) = P_{C_0}(k,n) \quad (4)$$

In a network only with address mutation and high-interaction honeypots, but no fingerprint mutation and anonymization, called **C-3** network, again it is adversary's skill level that determines the compromise probability. For skilled adversaries, we do not expect address mutations to be individually effective and lack of anonymization and fingerprint mutation allows adversaries to recognize hosts from their previous reconnaissance. This assumption was confirmed in our experimentation with white-hat skilled human attackers.

$$P_{C_3}(k,n,h,t_m) = P_{C_0}(k,n+h) \quad (5)$$

4.3 Effectiveness: Red-teaming Experimentation

Using our Proof-of-concept implementation, we created a network with a class `C` address space, consisting of 5 production hosts (VMs), and a differing number of proxy honeypots, generated by redirection to 18 honeypot VMs in the cloud. Appendix II provides a short summary of platforms services deployed on production and honeypot VMs.

On each production machine, at least three of these services were included and each service had individualized content; for example, no two Web servers were running the same Web application. Honeypot VMs included all these

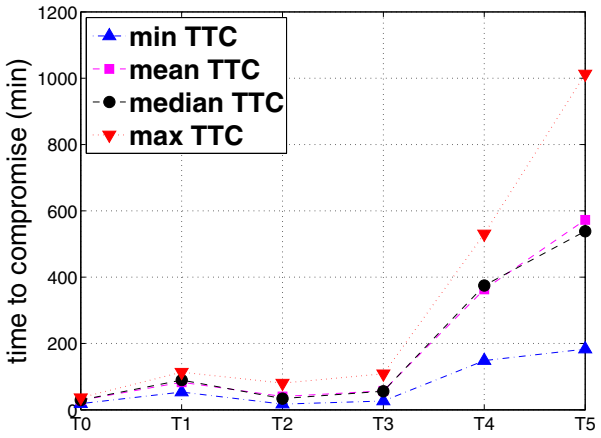

Figure 4: time-to-compromise for each test

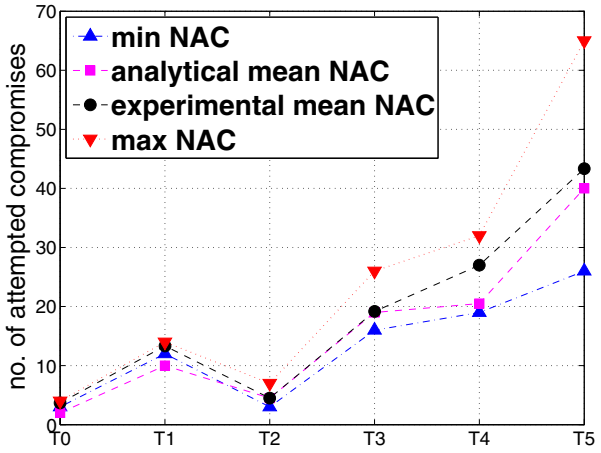

Figure 5: no. of attempted compromises for each test

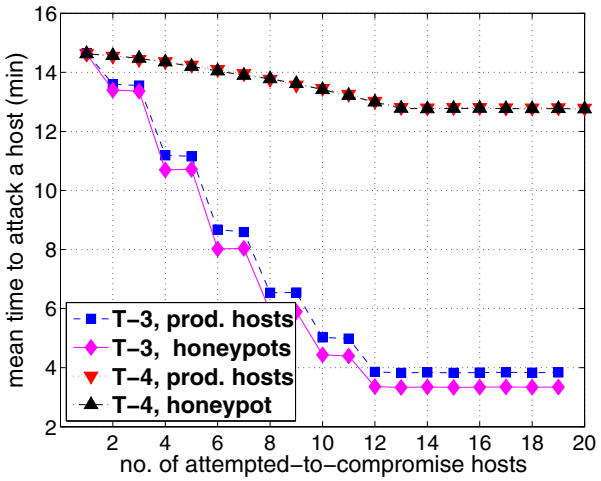

Figure 6: time spent on attacking honeypots vs. hosts for T-3 and T-4

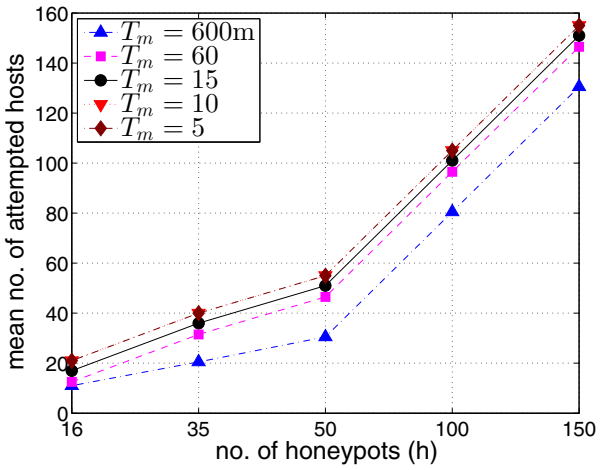

Figure 7: mean time spent compromising honeypots vs. prod. hosts

platforms and services, but each machine was running exactly one service. Following *context-aware content* principle, each physical service had randomly generated, but unique content.

Six security experts with similar hands-on experience in penetration testing were recruited to this aim. Participants were allowed to use or develop any tool, framework or script to automate the attack process.

A total of 6 tests, named **T-0** to **T-5** were held for each participant. Each test was held similar to a capture-the-flag (CTF) experiment. For each test, we randomly chose one of the services and made it vulnerable, usually through misconfiguration. In addition, a random string called *flag* was hidden in the content of this vulnerable application.

All vulnerabilities were of the same intensity, and preliminary information about the types of vulnerabilities was provided to participants to ensure that applicants have similar skill levels in detecting and exploiting various vulnerability services.

The participants were asked to identify and compromise this vulnerable service, discover the flag and submit it. However, the injected vulnerabilities were easy to exploit, in order to minimize the time variations related to the difficulty level of the exploitation process itself. This was necessary to ensure that our results are a correct reflection of the effort level required at the reconnaissance stage of the attack.

Test **T-0** was done on a **C-0** network, where the vulnerable service was a misconfigured FTP server with a weak password, and the flag was in a file hosted on the FTP server.

T-1 was done on a **C-1** network, populated with 16 static honeypots, to achieve 3-anonymity. The vulnerable service was a misconfigured `SSH` server with weak password.

T-2 was done on a **C-2** network with a 15-min interval address mutation, but no honeypots. The vulnerable service was a Web application with a simple XSS vulnerability, and the flag was hidden in the cookie.

T-3 was done on a **C-3** network, with 15-min mutation intervals and populated with 16 static honeypots (with default configurations). The vulnerable service was again a misconfigured FTP server with weak password.

T-4 was done on a **HIDE** network with 15-min mutation intervals and 16 mutating honeypots to achieve the same

3-anonymity. The vulnerable service was a misconfigured shared folder that included the flag file.

T-5 was done on a **HIDE** network again, but with shorter 5-min mutation intervals and a higher number (35) of mutating honeypots, to achieve 5-anonymity. The vulnerable service was a XSS-vulnerable Web application.

4.4 Results

The objective of our evaluation was to provide answers to the following three questions:

- **Question 1**: How effective our five-dimensional proactive defense is in (a) increasing MTTC and (b) disallowing attacker's search space to shrink, compared to those of **C-0** to **C-3** networks?
- **Question 2**: How distinguishable our proposed proxy honeypots are (from production hosts) to a skilled human attacker?
- **Question 3**: How (a) decreasing mutation interval and (b) increasing number of honeypots would affect MTTC?

In our experiments, we solely measure MTTC for generic one-step attacks, where the first compromised host is the final target. However, real attacks are usually multi-staged and the attacker usually has to traverse (compromise) a path of multiple hosts in the attack graph of the network [1], in order to arrive at a final target. While it is intuitive that due to the need for repeated reconnaissance at each step HIDE would even be more effective against multi-stage attacks, its experimental evaluation is left to future work.

4.4.1 Question 1: Effectiveness of HIDE

The first question focuses on evaluating effectiveness of HIDE-protected network in (a) increasing MTTC and (b) disallowing attacker's search space to shrink, compared to **C-0** to **C-4** networks.

Increasing mean time-to-compromise (MTTC): Figure 4 denotes the minimum, maximum, median, and mean time-to-compromise for all tests.

The test **T-0** was completed in a relatively short time by all participants. This led credibility to our preliminary assumption that *static and non-deceptive networks are highly vulnerable to skilled adversarial reconnaissance.*

T-1 slightly raised the bar for participants, due to the increased overhead for probing honeypots. However, the static configurations simplified reconnaissance by requiring participants to investigate each host (production or proxy honeypot) at most once.

T-2 was also completed by all participants in a relatively short time. The address mutation resulted in only a slight increase in completion time. This result was also consistent with our expectation that while effective against hitlist attacks [5] and automated worms [1], *address mutation* has limited effectiveness against skilled human attackers.

The test **T-3** slightly increased participants' workload, but since honeypots had arbitrary and static configurations, distinguishing them was straightforward for participants, especially as their knowledge regarding the test increased over time. This led credibility to our assumption that *low-fidelity, static, or arbitrarily-configured honeypots would not look persuasive to skilled human attackers.*

The test **T-4** on HIDE network resulted in a significant surge in participants' workload. The average MTTC increased from 28 minutes for a **C-0** conventional network and 40 minutes for a **C-2** (RHM [1]) network to to 363 minutes for HIDE network. For test **T-5** with more honeypots (35 honeypots to achieve 5-anonymity) and faster mutation (every 5 minutes), MTTC was increased to 572 minutes.

Comparing **T-1** and **T-4** shows the significance of address mutation and fingerprint mutation. This is because with static addresses and fingerprints, the attacker needs to probe each honeypot only once. Also, comparing **T-3** and **T-4** shows that the significance of high-fidelity honeypots. Although deploying high-interaction honeypots with arbitrary and static configurations would definitely increase attacker's reconnaissance workload, a skilled attacker would easily differentiate a honeypot after several interactions and blacklist it.

Increasing mean number of attempted compromises: Figure 5 compares the experimental minimum, maximum, and mean number of attempted compromises, *i.e.*, number of individual hosts, either production or honeypot, that participants attacked and attempted to compromise for each test. To this aim, in the traffic log generated by participants' network-level activities, all network flows destined to a host (an IP address) during one mutation interval are counted as one attack (attempt to compromise), because the information gathering process during an interval is incremental.

Figure 5 also compares the analytical mean (achieved via Eq. 1 to Eq.5) with the experimental mean. Firstly, note that experimental and analytical means are very close for all test cases, which shows correctness of our analytical modeling. More importantly, note that for tests **T-4** and **T-5**, which are focused on HIDE networks, the average MNAC (before target is attacked and compromised) is $n+h$ (which is the total number of production hosts and honeypots). This means that attacker's sampling is *with replacement*. Comparatively, for test **T-3**, MNAC is $(n+h)/2$, which shows that attacker's sampling is *with replacement*.

With respect to *question 1(b)*, this means that attacker's search strategy for target in the address space changes from uniform without replacement to uniform with replacement.

4.4.2 Question 2: Goodness of High-fidelity Honeypots

In our discussion regarding question 1(a), we emphasized the susceptibility of low-fidelity honeypots (high-interaction honeypots but with arbitrary and/or configuration, as in the test **T-3**) to being distinguished by skilled attackers.

Figure 6 shows how participants' average time for attempting to compromise a production host and honeypot changes over time.

Note that for in **T-3**, the time-to-attack decreases sharply after initial attacks. This is because the non-anonymized and non-mutated honeypot fingerprints allows participants to identify hosts even after address mutation. Therefore, the amount of time attacker invests on attacking a host decreases sharply over time, even with address mutation. In comparison, for **T-4** the time-to-attack for production and honeypot hosts remains the same over the course of test. This again shows that (1) our high-fidelity honeypots are not distinguishable from real production hosts, and (2) our anonymization (3-anonymity scheme) is effective in hiding

identity of previously attacked hosts against skilled human attackers.

Also note that especially for **T-4** as participants' experience increases, the average time dedicated to attacking each host decreases over time, up to a certain threshold. This is due to *experience curve* notion from economics, which states that as experience gained by repeating a task increases, the time required to do that task in subsequent iterations decreases.

4.4.3 *Question 3: effects of mutation rate and honeypot cardinality*

Comparing **T-4** and **T-5** in Figure 4 shows that decreasing mutation interval from 15 to 5 minutes and increasing the number of honeypots from 16 to 35 (both with 3-anonymity) almost doubles MTTC. Also, Figure 5 shows that the number of hosts that need to be attacked increases almost linearly with the number of honeypots. This linear increase emanates from indistinguishably of honeypots from production hosts.

Using our analytical modeling, we calculated the effect of shorter intervals and higher number of honeypots for the same $n = 5$ production hosts. Note that for higher number of honeypots, the MNAC is linear to $n + h$. Moreover, as our analytical modeling shows, while shorter mutation intervals are expectedly more effective, reducing the interval below mean time-to-attack (between 10 to 15 minutes in our experiments (Figure 6)) does not change MTTC.

4.5 Planning Overhead

We use the `Z3` SMT solver [30] on a Quad Core processor (`3.3GHz`, `8M cache`) and `16GB DDR3 RAM`. Figure 8 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 flow rate aimed at achieving 5-anonymity.

Firstly, note the running time is still affordable even for fairly large network sizes, especially since the SMT instance is solved with a low frequency (*e.g.*, on a daily basis).

Secondly, note that the running time is exponential with regard to 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 $\theta(m \cdot p)$.

Thirdly, note that the number of production hosts (n) has negligible effect on running time, especially for larger networks.

Finally, note that having a larger traffic bound (Δ) slightly improves computation time, because it relaxes the traffic rate constraint.

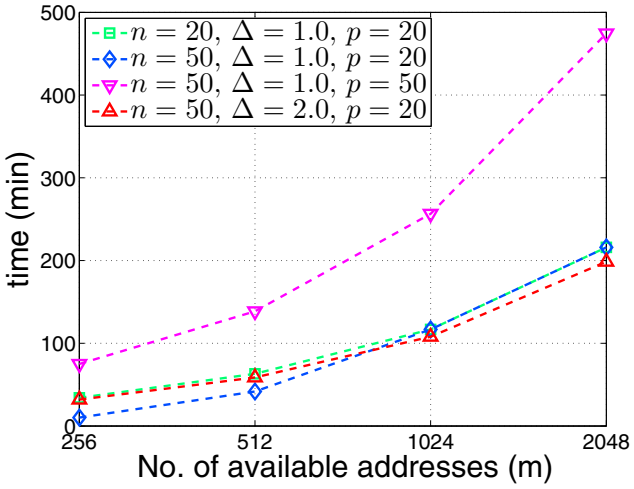

Figure 8: mutation planning overhead for various network settings

5. CONCLUSION

In this paper, we presented a novel five-dimensional proactive technique for defeating reconnaissance by skilled attackers, through mutating configuration (name, addresses, fingerprint) of network hosts, while populating address space with moving honeypots with strategically-designed but randomly-changing configurations. We also presented architecture and communication protocols for deployment of HIDE on enterprise networks. We showed that HIDE is transparent to all network protocols and devices as well as network sessions and regular operation of the network. Furthermore, we presented both analytical and experimental evaluation of HIDE effectiveness against skilled attackers. Experimentation was performed through CTF-inspired challenge experiments with a group of security specialists.

Our experiments showed that while deceptive technologies (honeypots) and address mutation techniques are both individually effective, their strategic composition creates a synergistic effectiveness against human attackers. Even in a very small-scale experiment, HIDE increased participants workload for completing their reconnaissance challenge up to 15 times (20 minutes vs. 5 hours).

While our evaluation provides insight on overall effectiveness of HIDE, understanding the individual effect of each dimension, as well as effect of HIDE on multi-stage APT attacks requires more tests and is left to future works. In addition among our five dimensions of HIDE, the fifth dimension of generating context-aware application content requires more research. In fact, crafting carefully-designed applications that include a diverse set of vulnerabilities and deceptive hooks based on network's mission and configuration is a very interesting and important research problem which is left to future works.

6. APPENDIX I

In this Section, we present the SMT formalization for mutation interval μ_t.

Anonymization

[Fingerprint k-anonymity]

$$\boldsymbol{sim}_{i,i'} \leftrightarrow \bigwedge_j (\boldsymbol{plan}\langle srv_{i,j}\rangle = \boldsymbol{plan}\langle srv_{i',j}\rangle) \tag{6}$$

$$\sum_l \boldsymbol{sim}_{i,l} \geq k-1 \tag{7}$$

[Fingerprint Mutation]

$$\boldsymbol{dif}_i = \sum_j (\boldsymbol{plan}^t\langle srv_{i,j}\rangle \neq plan^{t-1}\langle srv_{i,j}\rangle) \tag{8}$$

$$\boldsymbol{dif}_i \geq \alpha \tag{9}$$

High-Fidelity

[non-intrusiveness]

$$(srv_{i,j} \neq \emptyset) \leftrightarrow (\boldsymbol{plan}\langle srv_{i,j}\rangle = srv_{i,j}) \tag{10}$$

[Context-aware Fingerprints]

$$(\bigcup_{i,j} \boldsymbol{plan}\langle srv_{i,j}\rangle) = (\bigcup_{i,j} plan^{t-1}\langle srv_{i,j}\rangle) \tag{11}$$

$$\boldsymbol{plan}\langle srv_{i,j}\rangle \in M_j \tag{12}$$

[Believable Fingerprints]

$$\boldsymbol{plan}\langle live_i\rangle \leftrightarrow (\boldsymbol{plan}\langle srv_{i,1}\rangle \neq \emptyset) \tag{13}$$

$$\boldsymbol{plan}\langle live_i\rangle \rightarrow l_{min} \leq (\sum_j (\boldsymbol{plan}\langle srv_{i,j}\rangle \neq \emptyset)) \leq l_{max} \tag{14}$$

$$(\boldsymbol{plan}\langle srv_{i,j}\rangle = s_l) \leftrightarrow (\boldsymbol{plan}\langle srv_{i,k}\rangle \neq s_{l'}); \forall \langle s_l, s_{l'}\rangle \in X_{j,k} \tag{15}$$

Bounded Overhead

[Bounded Service Sharing]

$$(\sum_i \boldsymbol{\beta}_{srv_{i,j}} = s_l) \leq \pi_l/\psi_l \tag{16}$$

[Bounded Traffic Capacity]

$$\Delta - \epsilon \leq \left(\sum_{i,j} \delta_j \cdot (\boldsymbol{\beta}_{srv_{i,j}} \neq \emptyset \wedge \tau_{srv_{i,j}} = \emptyset)\right) \leq \Delta + \epsilon \tag{17}$$

7. APPENDIX II

List of platforms/services used in our evaluation testbed:

- Operating Systems: `Windows` (`Windows Server 2008 R2` and `Windows Server 2012`) and `Linux` (`Ubuntu 14.04` and `CentOS 7.x`) platforms were hosted. A
- FTP servers (`IIS FTP 7.5 & 8.0`, `vsftpd 3.0.3`, `ProFTPD 1.3.5`)
- SSH Servers (`OpenSSH 7.1p1`, `FreeSSHd 1.3.1`)
- sharing services (`SMB 3.0.2` `Samba 4.4.0`)
- Web servers (`Apache 2.4.18`, `IIS 7.5 & 8.0`)
- DNS servers (`BIND 9.10.2`)
- mail servers (`Exchange Server 2013`, `Sendmail 8.15.2`)

8. REFERENCES

[1] Jafar Haadi Jafarian, Ehab Al-Shaer, and Qi Duan. An effective address mutation approach for disrupting reconnaissance attacks. *IEEE Transactions on Information Forensics and Security*, 10(12):2562–2577, 2015.

[2] Michael Gregg. *Certified Ethical Hacker Exam Prep.* Que, 2006.

[3] Stuart McClure, Joel Scambray, George Kurtz, and Kurtz. *Hacking exposed: network security secrets and solutions*, volume 6. McGraw-Hill/Osborne New York, 2005.

[4] Latanya Sweeney. k-anonymity: A model for protecting privacy. *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems*, 10(05):557–570, 2002.

[5] S. Antonatos, P. Akritidis, E. P. Markatos, and K. G. Anagnostakis. Defending against hitlist worms using network address space randomization. *Comput. Netw.*, 51(12):3471–3490, 2007.

[6] Jianhua Sun and Kun Sun. Desir: Decoy-enhanced seamless ip randomization.

[7] N. Bjørner and L. de Moura. $z3^{10}$: Applications, enablers, challenges and directions. In *Sixth International Workshop on Constraints in Formal Verification*, 2009.

[8] Jafar Haadi Jafarian, Ehab Al-Shaer, and Qi Duan. Openflow random host mutation: transparent moving target defense using software defined networking. In *Proceedings of the first workshop on Hot topics in software defined networks*, pages 127–132. ACM, 2012.

[9] Jafar Haadi H. Jafarian, Ehab Al-Shaer, and Qi Duan. Spatio-temporal address mutation for proactive cyber agility against sophisticated attackers. In *Proceedings of the First ACM Workshop on Moving Target Defense*, MTD '14, pages 69–78. ACM, 2014.

[10] Neil C Rowe, E John Custy, and Binh T Duong. Defending cyberspace with fake honeypots. *Journal of Computers*, 2(2):25–36, 2007.

[11] 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.

[12] 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.

[13] 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.

[14] Kostas G Anagnostakis, Stelios Sidiroglou, Periklis Akritidis, Konstantinos Xinidis, Evangelos P Markatos, and Angelos D Keromytis. Detecting targeted attacks using shadow honeypots. In *Usenix Security*, 2005.

[15] Fred Cohen, Irwin Marin, Jeanne Sappington, Corbin Stewart, and Eric Thomas. Red teaming experiments with deception technologies. *IA Newsletter*, 2001.

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

[17] Jin-Yi Cai, Vinod Yegneswaran, Chris Alfeld, and Paul Barford. An attacker-defender game for honeynets. In *Proceedings of the 15th Annual International Conference on Computing and Combinatorics*, pages 7–16. Springer-Verlag, 2009.

[18] Andrew Clark, Kun Sun, Linda Bushnell, and Radha Poovendran. *Decision and Game Theory for Security: 6th International Conference*, chapter A Game-Theoretic Approach to IP Address Randomization in Decoy-Based Cyber Defense, pages 3–21. Springer, 2015.

[19] John Bethencourt, Jason Franklin, and Mary Vernon. Mapping internet sensors with probe response attacks. In *USENIX Security*, 2005.

[20] Moheeb Abu Rajab, Fabian Monrose, and Andreas Terzis. Fast and evasive attacks: Highlighting the challenges ahead. In *Recent Advances in Intrusion Detection*, pages 206–225. Springer, 2006.

[21] Eric M Hutchins, Michael J Cloppert, and Rohan M Amin. Intelligence-driven computer network defense informed by analysis of adversary campaigns and intrusion kill chains. *Leading Issues in Information Warfare & Security Research*, 1:80, 2011.

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

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

[24] Miles A McQueen, Wayne F Boyer, Mark A Flynn, and George A Beitel. Time-to-compromise model for cyber risk reduction estimation. In *Quality of Protection*, pages 49–64. Springer, 2006.

[25] Richard Delaney. Vagrant. *Linux Journal*, 2014(244):1, 2014.

[26] Sahil Suneja, Canturk Isci, Vasanth Bala, Eyal de Lara, and Todd Mummert. Non-intrusive, out-of-band and out-of-the-box systems monitoring in the cloud. In *ACM SIGMETRICS Performance Evaluation Review*, volume 42, pages 249–261. ACM, 2014.

[27] Md Arifur Rahman, Mohammad Hossein Manshaei, and Ehab Al-Shaer. A game-theoretic approach for deceiving remote operating system fingerprinting. In *Communications and Network Security (CNS), 2013 IEEE Conference on*, pages 73–81. IEEE, 2013.

[28] Guillaume Prigent, Florian Vichot, and Fabrice Harrouet. Ipmorph: fingerprinting spoofing unification. *Journal in computer virology*, 6(4):329–342, 2010.

[29] David Barroso Berrueta. A practical approach for defeating nmap os-fingerprinting. *Retrieved March*, 12:2009, 2003.

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