Active Deception Model for Securing Cloud Infrastructure

Albert Brzeczko, A. Selcuk Uluagac, Raheem Beyah, John Copeland
School of Electrical and
Computer Engineering
Georgia Institute of Technology
Atlanta, Georgia 30332–0250

***Abstract*—The proliferation of cloud computing over the past several years has led to a variety of new use cases and enabling technologies for enterprise and consumer applications. Increased reliance on cloud-based platforms has also necessitated an increased emphasis on securing the services and data hosted within those platforms. From a security standpoint, an advantage of cloud platforms over traditional production networks is that they have a dynamic, mutable structure that can change as a result of a variety of factors, so reconnaissance on the part of an attacker is far less predictable. In this work, we propose a novel technique that leverages the amorphous nature of cloud architectures to deceive and redirect potential intruders with decoy assets implanted among production hosts. In this way, attackers encounter and probe decoys that lead them to reveal their motives and cause them to be less likely to compromise their intended target, particularly once they have revealed their tactics against decoy assets. We show that our technique, after having been exposed to live traffic for approximately one month, detected 1,255 highly malicious hosts and was able to divert 97.5% of malicious traffic from these hosts. This traffic would have otherwise reached production hosts and potentially led to compromise.**

Index terms - Intrusion Deception, Cloud Computing, Information Security, Network Decoys, Honeynets

I. Introduction

Since Amazon introduced the Elastic Compute Cloud in 2006 [1], the transition to cloud computing for enterprise and consumer services has become a prevailing trend in the information technology (IT) industry. According to Gartner, approximately 38% of all enterprises use some form of public cloud infrastructure, and that number is set to increase to 80% within the next twelve months [2].

Cloud computing has become such a prevailing trend because it commoditizes computing hardware and allows infrastructure to be provisioned to large scale, elastically, and without rigid constraints upon network topology. Effectively, cloud services succeed at abstracting many of the hard problems of IT management away from the enterprise and shifting them onto a specialized hosting provider. These intrinsic features of the cloud also make it a highly advantageous place to employ the strategy of deception. The scalability and elasticity of typical cloud services provides computational head room for ephemeral, non-production assets (e.g. decoy hosts) that can be provisioned when production demand on the platform is below saturation, and de-provisioned so as to free resources as production demand requires. Furthermore, given the dynamism and interchangeability of individual hosts within the cloud, intruders will be able to infer far less from the way the network is structured (aside from domain name service records), and their task of reconnaissance is thus made more difficult, necessitating more aggressive techniques. The fact that the allocation of hosts on the network can and does change in very short intervals further implies that the relevance of the attacker's aggressive reconnaissance is more short-lived than in traditional network deployments. For these reasons, targeted attacks against cloud-hosted assets are considerably more difficult to perform, and an attacker is very likely to encounter a decoy in his search for production assets.

In spite of the considerable difficulty of targeted attacks against a cloud computing environment, the threat of automated attacks (e.g. botnets) remains a very real one [3]. Furthermore, while an intruder may not have knowledge of the specific allocations of hosts on a cloud's public subnet, it can reasonably infer that the subnet has targets of high value. Depending upon the cloud provider's policies, the hosted services likely have heterogeneous security measures, meaning the adversary may find a weak link that gains them access to considerable computing, storage, and network resources. For this reason, the subnets of cloud providers are highly attractive to automated attacks [3].

In this work, we demonstrate a technique whereby a cloud hosting environment responds to network-based attacks by allocating unused computing resources and network space toward decoy hosts and exposing them to the network under attack. These hosts are provisioned in priority order according to the class of attacker detected. In this way, decoys are tailored to the types of attacks that are most prevalent. For instance, if a large number of connection attempts are classified as SSH probes, a proportionate number of fake, vulnerable SSH honeypots are deployed on the unused IP space and used to collect application-layer intelligence on the attackers, and use that intelligence for further classification. From this intelligence, a threat index for each attacking host is computed and updated progressively as that host interacts with our system. If an attacker's threat index exceeds a certain threshold, connection attempts originating from that host are redirected away from production assets toward decoys that mimic the functionality of the production host. We show that our technique, after having been exposed to live traffic for

978-1-4799-3088-3/14/$31.00 ©2014 IEEE

approximately one month, detected 1,255 highly malicious hosts and was able to divert 97.5% of malicious traffic from these hosts while maintaining dialog with the intruder as a source of further intelligence post-detection. This traffic would have otherwise reached production hosts and potentially led to compromise. To the best of our knowledge, this is the first work to use active deception techniques to secure cloud computing infrastructures.

The paper proceeds as follows: an overview of related and background work is presented in Section II. A formal definition of the problem and our proposed solution is presented in Section III, followed by a detailed description of the testbed setup used to evaluate the proposed solution in Section IV. In section V, the results of the experimentation are discussed and conclusions about the results and plans for future extensions to the research are presented in Section VI.

II. Related Work

This section contains a synopsis of the related prior work that our proposed solution extends.

A. Existing Cloud Security Measures

In the cases of Intrastructure-as-a-service (IaaS) and Platform-as-a-service (PaaS) cloud models, the security of the hosted assets is typically the responsibility of the customer. That is, the enterprise must implement its own security policy on any cloud assets thus hosted. Several of the leading public cloud providers provide or offer the option of enabling built-in security measures such as traffic filtering, traditional IDS, client browser integrity checking, and visitor reputation whitelisting/blacklisting [4][5][6][7][8][9][10]. It is worth noting that each provider offers slightly different sets of security features, meaning that the customer must self-implement those required measures that are not offered on their particular hosting provider. Many organizations even prefer to have the control over their own security, rather than rely on the hosting provider to implement security measures for them [11]. However, this strategy results in a heterogeneous set of security implementations in a particular cloud, which can be difficult to audit effectively.

A recent proposal to cloud security best practice is known as the software defined perimeter (SDP) [12], which "aims to give application owners the ability to deploy perimeter functionality where needed." The main objective of the SDP is, similarly to the traditional fixed perimeter model, to hide internal assets and disallow external users from accessing them. In contrast to the fixed perimeter, SDP allows for finer grained control of the logical perimeter, allowing it to exclude untrusted devices that move into the physical enterprise (e.g. bring-your-own-device) as well as include trusted assets that reside outside (e.g. cloud-hosted services). Our work, which dynamically modifies the perimeter configuration to enable deception, presents a complementary value to emerging SDP technology.

B. Deception Using Honeynets

Honeypots, or collections of honeypots known as a honeynet, are key enabling technologies that have been used within the information security research community to gather intelligence on widespread, novel attack campaigns [13][14]. The intrinsic value of a honeypot, from the security researcher's perspective, lies in the fact that it has no production value [15]. In other words, legitimate users would have no reason to interact with a honeypot, as it offers them no usable service or content. Therefore, any traffic seen by it is automatically considered to be malicious. This characteristic has the effect of filtering out the large volume of production traffic normally seen on a host and allowing attack traffic to be monitored in isolation. It also allows for richer interaction with an attacker as an asset having no resources worthy of protecting can safely be allowed to be exploited by an attacker. By enabling the attacker to succeed, far more information can be gathered about the attacker's motives, rather than simply blocking its traffic at the firewall.

Therefore, honeypots and honeynets provide a rich complement to existing security measures in that they extend the attack surface of a particular network to allow deeper information gathering on threats. However, in traditional production environments, this sort of strategy can impose a great deal of overhead and require diversion of already scarce IT resources away from production tasks. Furthermore, the allowance of active attacks must be controlled and monitored very carefully, and those who do not have this particular skill set stand a reasonable chance of incurring liability for themselves through misconfiguration of the deceptive assets [16][17]. For these reasons, enterprises are unlikely to integrate such a technology into their security posture unless it is seamlessly integrated and managed for them.

C. Clustering of Connection and Activity Features for Malicious Network Traffic Detection

A framework known as BotMiner [18] has been presented that very effectively identifies the behaviors of malicious bots by passively monitoring connection (C-plane) and activity (A-plane) characteristics on an existing subnet. By using an unsupervised machine learning algorithm known as X-means clustering on the C-plane and A-plane, and by performing cross-plane correlation between these two, the creators of BotMiner were able to achieve 100% detection of six of the eight botnets that had infiltrated hosts on the network under test, with detection rates of 99.6% and 75% on the other two. This approach is clearly very effective for botnet detection, but the authors acknowledge several areas of their work that could use improvement, namely that their "A-plane clustering implementation utilizes relatively weak cluster features" and that the observed A-plane activities were not exclusive to botnets, thus the possibility of generating "a lot of false positives." Therefore, the BotMiner framework's efficacy would be improved significantly by a richer source of A-plane features and a way to filter out the majority of benign traffic, two objectives that honeypots excel in. Our work uses a deceptive layer of infrastructure to gather exactly the types of activity features that extend the A-plane feature set of BotMiner's prior work.

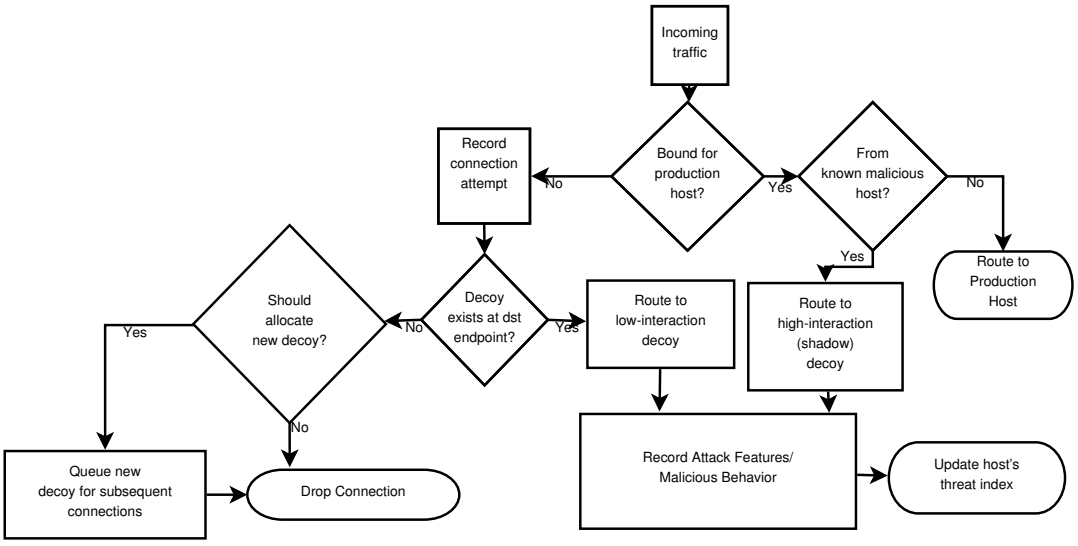

Fig. 1: Flow Diagram of Incoming Traffic Handling

III. System Overview

In this section, we present a high-level overview of the problem space, a threat model describing what our proposed solution addresses, and a detailed discussion of the proposed solution itself.

A. Problem Space

The scalability and elasticity of typical cloud services provides computational head room for ephemeral, non-production assets (e.g. decoy hosts) that can be provisioned when production demand on the platform is below saturation, and deprovisioned so as to free resources as production demand requires. Furthermore, given the dynamism and interchangeability of individual hosts within the cloud, intruders will be able to infer far less from the way the network is structured (aside from domain name service records), and their task of reconnaissance is thus made more difficult, necessitating more aggressive techniques. The fact that the allocation of hosts on the network can and does change in very short intervals further implies that the relevance of the attacker's aggressive reconnaissance is more short-lived than in traditional network deployments. For these reasons, targeted attacks against cloud-hosted assets are considerably more difficult to perform, and an attacker is very likely to encounter a decoy in his search for production assets.

A cloud computing platform, whether it be private, Intrastructure-as-a-Service (IaaS), Platform-as-a-Service (PaaS), Software-as-a-Service (SaaS), or otherwise [19], must host production assets on a publicly available subnet that is subject to all the threats typically levied against public internet-accessible hosts. Traditional perimeter defenses and intrusion deception systems go a long way to mitigating many of these threats, but the often heterogeneous security practices implemented on a multi-tenant cloud system lend themselves to a set of vulnerabilities caused by configuration and inconsistent access controls across the cloud infrastructure. These vulnerabilities can be discovered (and remediation strategies offered) via network and behavioral analysis. A layer of assets having no production value can be used to carry out deception against the potential attacker, causing him to reveal his techniques. Given the scalability and elasticity of most cloud platforms, the opportunity arises to deploy this deceptive layer in the cloud's excess computational capability without causing a negative performance impact on its production capabilities. Furthermore, the dynamic nature of the cloud infrastructure allows this deceptive layer to blend in more easily, and an attacker is likely to perform aggressive tactics against the deceptive layer that exists solely for monitoring and capture of malicious activity. The attacker thereby leaves traces of the sorts of vulnerabilities he is targeting. From the information thus gathered on the decoy assets, the attacker can be classified as malicious and diverted away from production assets on which he can do real harm.

B. Threat Model and Assumptions

In our work, we assume the following threat model:

1) Purpose: (a) Establish a set of content that has no production value, but that attracts attackers to interact with it and reveal intelligence about their motives. (b)Use this gathered intelligence to identify that a host is malicious and divert its traffic away from production.

2) Assets: (a) Cloud hosting environment, (b) Production content hosted on the cloud, (c) Decoy coordinator, (d) Decoy content hosted on the cloud.

3) Assumptions: (a) Hosting environment has elastic but finite resources. (b) Production content is not required to expose hooks, nor participate in coordinating deception. (c) Decoy coordinator has control over router/firewall configuration to perform content redirection. (d) Decoy content is never commissioned in such a way that imposes adverse performance impact on production content.

Decoy Type	Feature Type	Threat Index
dionaea	accept (httpd)	0.1
	accept (ftpd, mssqld, mysqld)	0.5
	accept (smbd, epmapper)	1.0
	profile	1.0
	mysql_command	1.0
	dcerpcrequest	2.5
	dcerpcbind	2.5
	offer	2.5
	reject	0.1
	login	2.5
	mssql_fingerprint	1.0
	mssql_command	1.0
	downloads	2.5
	service	2.5
	listen	2.5
	connect	1.0
	sip_command (cmd)	2.5
	sip_command (addr, via)	0.5
glastopf	request_raw	0.1
	request	0.1
kippo	login_succeeded	5.0
	terminal_size	0.5
	lost_connection	0.5
	client_version	0.5
	login_failed	1.0

Fig. 2: Threat Index Contributions of Common Attack Features

4) Threats: (a) The attacker identifies decoy content and avoids it. This condition reduces efficacy of the intelligence gathering technique, but is worst case net-neutral over existing methodologies. (b) The attacker leverages decoy computational resources to perform further malicious activity.

C. Proposed Solution

Our proposed solution to the problem leverages the existing dynamism of cloud orchestration to provision a layer of deception within the cloud itself. As cloud infrastructures enable scalability and elasticity, they feature resources (both computational and network) far beyond those demanded by production assets during their normal mode of operation. We therefore propose that these extra resources be allocated toward ephemeral decoy assets that carry out the deception. We mitigate Threat (a) in our threat model by replicating production configurations for high-interaction decoys where possible, and updating/moving low-interaction decoy assets periodically within the subnet. We mitigate Threat (b) by placing stringent resource allocation policies toward decoy content such that the attacker has very little computational power to leverage even in the unlikely event of a compromise.

Our implementation of this strategy is based upon Canonical's Juju [20] service orchestration framework. Juju was chosen for its ability to interface with nearly all existing cloud APIs as well as for its extensibility via service modules known as *charms*. In order to maintain a wall of separation from the real production assets, the decoys are placed within a different internal subnet than that of the production hosts and communications between the two subnets are disallowed. On this secondary internal subnet, we deploy an environment with several existing Juju charms to provide infrastructure for the decoy subnet. Furthermore, we have implemented our own custom charms that enable reliable deployment of decoy assets within this environment. For the purposes of our proof-of-concept prototype, our custom charms enable us to deploy several different honeypots (Glastopf [21], Dionaea [22], and Kippo [23]), and we gain the ability to incorporate virtually any service as a decoy either by using existing Juju charms or writing our own.

While it is possible to pre-define statically what is deployed within the decoy environment, the innovation of our strategy comes in its ability to blend in with the dynamic nature of the cloud. For the scope of this work, an automatic deployment strategy was implemented that surveys the N most frequently accessed decoy-eligible (e.g., unused) public subnet endpoints and responds by automatically provisioning an applicable decoy at this network endpoint. For instance, if our cloud is located on the subnet 3.4.5.0/24 and we observe a comparatively large volume of SYN/RST interactions with 3.4.5.158 on tcp/22, we are well served to deploy a Kippo (SSH) honeypot on this endpoint to gather more intelligence on the nature of these communications. The value of N is configurable and based upon the size of the subnet in question, as well as the extent to which cloud resources need to be economized. In our implementation, we devoted 20% of the unused endpoints toward decoys.

Once deployed within the cloud via Juju, decoy assets are made available on the public network via a configurable firewall based upon iptables. At the perimeter, firewall rules are modified automatically to expose provisioned decoy assets as though they existed on the production net. This is accomplished either via selective port forwarding, or in some cases 1:1 NAT is more appropriate. In addition, production assets themselves can optionally be shadowed by a decoy, to which traffic identified as originating from malicious hosts to the production asset is redirected. The logic by which incoming connections are routed either to production or the decoy infrastructure is shown in Figure 1.

The intrinsic benefit to this strategy is that the decoy assets carry out no production goal whatsoever, so any traffic that reaches them is considered to be suspicious. Furthermore, rather than simply denying the traffic from entering the firewall, and therefore gaining no further knowledge from it, these attack interactions are allowed to proceed against valueless assets that gather behavioral information. This behavioral information is then used to build a detailed vector of attack features for each host. Hosts are then grouped by these features and their relation to other hosts are determined by the Jaccard index of similarity [24] between their feature sets. From these groupings and similarity metrics, we form associations among hosts by the nature of the attack(s) perpetrated.

In addition to the similarity metrics, we compute a composite threat index for each host from observed attack behavior according to the rubric in Figure 2. Once this threat index reaches a sufficiently high value (in our case, 50.0 is the threshold), subsequent traffic from that host is automatically diverted away from production assets and toward shadow decoys.

IV. Experimental Testbed Setup

A detailed explanation of the hardware and software setup and testing methodologies we used to evaluate our proposed solution follows in this section.

We implemented a prototype of our proposed solution in python and deployed it inside a local Linux containers (LXC) environment managed by the Juju framework. We also hosted example production assets on an unfiltered /25 public IPv4 subnet on our academic network. In this way, the production assets are subject to typical threats faced by internet-facing services on real networks. To represent legitimate network traffic, we crafted a simulator that carries out requests to the production assets within the expected use cases of the content being hosted. To supplement the existing probing and malicious traffic coming in over the /25, as well as provide a ground truth of detection capabilities, we subjected the system to further malicious traffic by using the tools in the BackTrack5 R3 penetration testing suite to perform reconnaissance and attacks against our /25 subnet.

V. Results

This section presents our findings as a result of applying the aforementioned testing methodologies.

After allowing the active deception system to remain exposed to active threats on our public subnet for approximately 1 month (Dec 2013), our decoy infrastructure encountered connections from 6,739 unique remote hosts whose observed attack behaviors fell into 537 distinct groupings. These groupings ranged from having 1 feature to having over 800. Among the 193 one-feature (singleton) groups, all the behavior features involved an accepted or rejected connection attempt on a particular port, indicating a single-port probe. Due to this characteristic, the singleton group class is representative of the intelligence gathered by logging disallowed connections at the perimeter. The single-port scanners represented 5,703 of the encountered hosts (84.6% of the total). Of the single port scanners, 4,882 (or 85.6%) amassed a very low threat index under 1.0. This low threat index indicates a non-determined attacker that has not presented enough of a threat to warrant blacklisting. Nevertheless, an interesting characteristic emerged from these low-threat entities, which is that the two most commonly accessed ports by this class of attacker were 3389/tcp (52.2%) and 4899/tcp (29.3%). In other words, over 80% of hosts responsible for our most benign traffic patterns may have proven not to be so benign if our attack surface included the Microsoft and Radmin screen-sharing protocols that operate on these ports. This result underscores the fact that our technique gains considerably more intelligence by

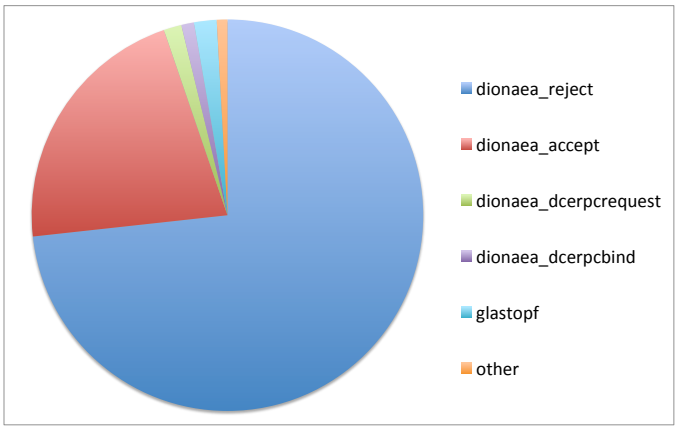

Fig. 3: Attack Features for Hosts with ThreatIndex $\in$ [1.0, 50.0)

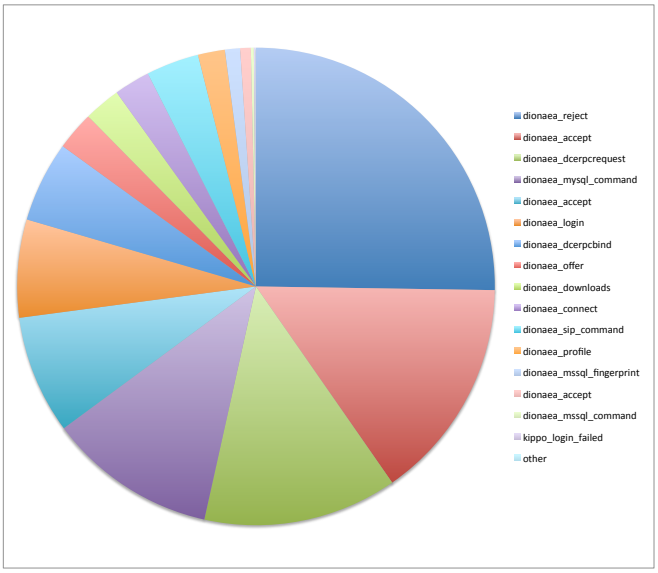

Fig. 4: Attack features for hosts with ThreatIndex $\in$ [50.0, $+\infty$)

expanding the attack surface vs. merely rejecting packets at the perimeter. Even within the class of singleton features, several hosts amassed very high threat scores (some as high as 8178.0) through massively repeated probing of low-interaction decoy services such as SIP and smbd.

We observed 1,255 hosts with threat indices in the range [1.0, 50.0). Among this set of sub-threshold hosts, we observed 52,858 attack events having the feature breakdown observed in Figure 3. The vast majority of the detected behaviors were connection rejects (73%) and connection acceptances (21%), still indicative of probing behavior. Hosts in this threat range representing this probing behavior totaled 1,141 (91%). The remaining features in this threat index range began to implicate hosts exhibiting low volumes of decidedly malicious behavior against the decoy hosts (e.g. *dcerpcrequest*s and *dcerpcbind*s).

Our system assigned 533 hosts a threat index of greater

than or equal to 50, meaning that these hosts were classified as decidedly malicious and that production assets should be protected from their network traffic. These hosts generated a set of 810,336 features consisting of the feature classes shown in Figure 4. In contrast to the sub-50 threat index feature set, the rejections (25%) and connections (15%) comprised much smaller shares of the total. This relative reduction is caused by the fact that hosts in this subset are recorded performing malicious activities in much greater quantities. In particular, the *dcerpcrequest*/*dcerpcbind* events, commands executed against the *mysql* decoy service, and login attempts now represent 5% to 13% of the total feature set. This drastically different set breakdown that occurs above threat index of 50.0 led to its choice as a reasonable threshold for blacklisting.

Our technique succeeded in redirecting traffic from these 533 hosts away from production hosts once they surpassed the threat index threshold of 50.0. The attack campaigns of these 533 hosts lasted, on average, 2 days, 19 hours. On these campaigns, our technique's overall time until detection averaged approximately 12 hours, and on average, after the offending host carried out 31.43% of its attack events against our decoy sensors. Given that after detection, the host that exceeds threat index of 50.0 is diverted from production hosts, *these detection characteristics resulted in redirection of 1241.39 MiB of 1273.17 MiB bound from these malicious hosts, representing an overall reduction of attack traffic by 97.5%.*

VI. Conclusion & Future Work

This section presents conclusory statements and a discussion of what additional work can be done in the future to expand upon the research already performed.

The results show that our technique is successful at using deceptive infrastructure to monitor attackers' behavior and use it to classify them as malicious. Our proof-of-concept prototype, which we implemented in Python, deployed in LXC with Juju, and exposed to live internet traffic for the month of December 2013, achieved a 97.5% reduction of malicious traffic bound for production hosts by detecting and redirecting attack campaigns well before they concluded.

In our live setup, we relegated the decoy library to Kippo (SSH), Glastopf (Web) and Dionaea (numerous protocols), again to serve as proof of concept. In the future, we will provide decoy services for many additional protocols, thus increasing the attack surface. In particular, we noted a large number of connection attempts against the Microsoft RDP and Radmin screensharing protocols, which indicates that these protocols are likely to be ripe vectors for enhanced feature detection. We calculated our composite threat index for a given host based upon the sum of individual attack features that (mostly) had threat indices based upon the class of feature involved. For example, our sensors attributed the same threat index contribution of 0.1 to an http request for "/" as an http request for "//phpmyadmin/scripts/setup.php". The latter should contribute a higher value to the overall threat index, and in future works, we will adjust the threat index contribution based on the content as well as the class of feature. In future iterations that have both a higher traffic volume and a significantly larger corpus of attack features, we anticipate much higher computational complexity involved with fully searching the attack features as we have done in this work. Therefore, we plan to make enhancements to our correlation approach and rely on it to make probabilistic decisions that fit in a real-time computational envelope, while buffering the higher fidelity calculations for completion in the background.

References

[1] C. Evangelinos and C. Hill, "Cloud computing for parallel scientific hpc applications: Feasibility of running coupled atmosphere-ocean climate models on amazon's ec2." *ratio*, vol. 2, no. 2.40, pp. 2–34, 2008.

[2] J. Rivera and R. Van Der Meulen. Gartner says the road to increased enterprise cloud usage will largely run through tactical business solutions addressing specific issues. [Online]. Available: http://www.gartner.com/newsroom/id/2581315

[3] Y. Chen, V. Paxson, and R. H. Katz, "What's new about cloud computing security," *University of California, Berkeley Report No. UCB/EECS-2010-5 January*, vol. 20, no. 2010, pp. 2010–5, 2010.

[4] Cloudflare. Cloudflare security. [Online]. Available: https://www.cloudflare.com/features-security

[5] Microsoft. Technical overview of the security features in the windows azure platform. [Online]. Available: http://www.windowsazure.com/en-us/support/legal/security-overview/

[6] BlueLock. Cloud security features and functionality. [Online]. Available: http://www.bluelock.com/virtual-datacenters/cloud-security/security-features/

[7] Joyent. Securing joyent cloud. [Online]. Available: http://www.joyent.com/developers/security

[8] Salesforce. Security overview. [Online]. Available: https://trust.salesforce.com/trust/security/

[9] Rackspace. Security solutions and services. [Online]. Available: http://www.rackspace.com/security/domains/

[10] Amazon. Aws security center. [Online]. Available: http://aws.amazon.com/security/

[11] CenturyLink. Cloud security: Cios have more pressing things to worry about. [Online]. Available: http://www.centurylink.com/business/asset/white-paper/cloud-security-cios-have-more-pressing-things-to-worry-about-cm101361.pdf

[12] A. Boehme, B. Flores, J. Schweitzer, and J. Islam, "Software defined perimeter," Cloud Security Alliance, Tech. Rep., December 2013.

[13] Z. Li, A. Goyal, and Y. Chen, "Honeynet-based botnet scan traffic analysis," in *Botnet Detection*. Springer, 2008, pp. 25–44.

[14] O. Thonnard and M. Dacier, "A framework for attack patterns' discovery in honeynet data," *digital investigation*, vol. 5, pp. S128–S139, 2008.

[15] L. Spitzner, *Honeypots: tracking hackers*. Addison-Wesley Reading, 2003, vol. 1.

[16] ——, "The honeynet project: Trapping the hackers," *Security & Privacy, IEEE*, vol. 1, no. 2, pp. 15–23, 2003.

[17] M. Dornseif and S. May, "Modelling the costs and benefits of honeynets," *arXiv preprint cs/0406057*, 2004.

[18] G. Gu, R. Perdisci, J. Zhang, and W. Lee, "Botminer: Clustering analysis of network traffic for protocol- and structure-independent botnet detection," in *Proceedings of the 17th Conference on Security Symposium*, ser. SS'08. Berkeley, CA, USA: USENIX Association, 2008, pp. 139–154. [Online]. Available: http://dl.acm.org/citation.cfm?id=1496711.1496721

[19] B. P. Rimal, E. Choi, and I. Lumb, "A taxonomy and survey of cloud computing systems," in *INC, IMS and IDC, 2009. NCM'09. Fifth International Joint Conference on*. Ieee, 2009, pp. 44–51.

[20] Ubuntu juju. [Online]. Available: https://juju.ubuntu.com

[21] Glastopf. [Online]. Available: http://glastopf.org

[22] Dionaea — catches bugs. [Online]. Available: http://dionaea.carnivore.it

[23] Kippo. [Online]. Available: https://code.google.com/p/kippo/

[24] R. Real and J. M. Vargas, "The probabilistic basis of jaccard's index of similarity," *Systematic biology*, vol. 45, no. 3, pp. 380–385, 1996.