Received: 29 March 2019 | Revised: 7 March 2020 | Accepted: 24 March 2020

DOI: 10.4218/etrij.2019-0155

ORIGINAL ARTICLE

ETRI Journal WILEY

New framework for adaptive and agile honeypots

Seamus Dowling[1] | Michael Schukat[2] | Enda Barrett[2]

[1]Mayo Campus, Galway Mayo Institute of Technology, Mayo, Ireland

[2]Discipline of IT, College of Engineering and Informatics, National University of Ireland Galway, Galway, Ireland

Correspondence
Seamus Dowling, GMIT, Mayo Campus, Castlebar, Mayo, Ireland.
Email: seamus.dowling@gmit.ie

Funding information
None reported.

This paper proposes a new framework for the development and deployment of honeypots for evolving malware threats. As new technological concepts appear and evolve, attack surfaces are exploited. Internet of things significantly increases the attack surface available to malware developers. Previously independent devices are becoming accessible through new hardware and software attack vectors, and the existing taxonomies governing the development and deployment of honeypots are inadequate for evolving malicious programs and their variants. Malware-propagation and compromise methods are highly automated and repetitious. These automated and repetitive characteristics can be exploited by using embedded reinforcement learning within a honeypot. A honeypot for automated and repetitive malware (HARM) can be adaptive so that the best responses may be learnt during its interaction with attack sequences. HARM deployments can be agile through periodic policy evaluation to optimize redeployment. The necessary enhancements for adaptive, agile honeypots require a new development and deployment framework.

KEYWORDS
adaptive, agile, framework, honeypots, reinforcement learning

1 | INTRODUCTION

The implementation of new and evolving technological concepts increases the opportunity to exploit new cyber-attack surfaces. Every web-facing device or service is vulnerable when connected to the Internet. Previously independent devices are becoming accessible through new software protocols and physical communication channels. Internet of things (IoT) significantly increases the attack surface available to malware developers. In 2016, Mirai [1] compromised IoT devices and contributed to large-scale distributed denial-of service (DDoS) attacks. To react to new changes in malware evolution, cyber-security measures should also evolve. Zero-day attacks become major malware threats if not promptly discovered. If established, malware uses highly automated and repetitive methods to propagate globally, infecting and compromising hosts, and forming large-scale botnets. These botnets can be used to launch extensive DDoS attacks or can further propagate and change into new variants. Kaspersky states that the number of malware modifications targeting IoT devices in the first half of 2018 was greater than the total number of such modifications in 2017 [2]. To understand malware and its variants, in 2003, Spitzner [3] suggested capturing attack data to understand the motives of developers and the behavior of the corresponding tools. The resulting *honeypots* uncover attack behavior with longitudinal deployments, capturing large datasets for retrospective analysis. Early versions of honeypots had low interaction capabilities. They were simple devices simulating Internet services and detecting the presence of an attacker. Mid- and high-interaction honeypots (MiHPs and HiHPs, respectively) allowed more interaction with an attacker. To adapt to new malware methods, honeypots have evolved to utilize new technologies.

Honeypots, regarded as dynamic, real-time analytical tools, have certain limitations. Moreover, their design, configuration, and operation require careful consideration. A compromised honeypot could itself inadvertently participate in further attacks; therefore, honeypot operations require constant monitoring. Honeypots facilitate attack interaction with scripted responses to attack-command streams. If the honeypot encounters an attack command it cannot process, then the attack terminates. Automated malware also employs honeypot detection mechanisms within its code. Anti-honeypot checks cause attacks to fail when processes are exposed [4]. Once honeypot functionality has been exposed, malware such as botnets will cease the attempted compromise. New malware variants employ similar techniques to evade detection by known honeypots. This reduces the potential size of a captured dataset and the subsequent analysis. With the growth of new attack surfaces and vectors for malware developers, cyber-security measures, such as honeypots, should dynamically adapt to new threats. New methods of honeypot design and deployment are required to overcome the limitations against evolving malware. Honeypots are required to be adaptive and agile so that better datasets for faster forensics may be provided. Reinforcement learning can be used in conjunction with honeypot operations to provide adaptability. The state–action space formalism outlined in Section 4 is designed to target automated and repetitive malware. Deployment strategies should be re-examined to provide agility for new variants. A new framework is required to facilitate honeypot development. In this respect, the contributions of this study are the following:

- A new framework for honeypots is proposed. Existing taxonomies are assessed for relevance. Updated classes and values are generated, incorporating adaptive and agile functionality into honeypot development and deployment.
- Data captured on adaptive honeypots can be used to evaluate reinforcement-learning algorithms and policies. Agile honeypot deployment is facilitated by Q-learning and SARSA (state, action, reward, state, action) under a variety of policy configurations.

2 | PREVIOUS RESEARCH

2.1 | Honeypot evolution

Since their introduction in the 1990s, honeypots have evolved to meet the changing landscape of cyber threats. The number of deployments of bots and malicious code targeting IoT end-devices has increased [5]. In 1992, USENIX conferences presented work on captured *cracker* activities [6]. Terminal machines were deployed to lure unauthorized users and monitor their activity. Spitzner defined a honeypot as a "security resource whose value lies in being probed, attacked or compromised" [3]. Research on honeypots and honeynets has since increased. *Honeyd* appeared in 2003 and is an easy-to deploy, low-risk honeypot [7], which can safely deploy virtual honeypots with different IP numbers. Honeyd is considered a low-interaction honeypot (LiHP), gathering information on the activities of an attacker in a virtual, confined space. At that point in their evolution, honeypots could be compromised and inadvertently partake in subsequent attacks [8]. Terminology such as low, medium, and high interaction came into honeypot parlance. LiHPs capture base information such as IP addresses, port numbers, and services. They will not permit the installation or execution of downloaded malware, and their implantation is considered low risk. By contrast, HiHPs provide to the attacker more scope for installing malware and exploring the operating system and file structure. This has the advantage of maintaining the interest of attackers and capturing more information on their behavior. HiHPs are real systems, often mirroring live production systems. The obvious disadvantage of a more interactive honeypot is the potential for compromising the honeypot itself. This could allow access to live production networks or participation in subsequent attacks. Nepenthes [9] is a LiHP emulating known vulnerabilities used by worms to spread across the Internet. Argos [10] is a HiHP providing real system functionality through the guest OS to capture zero-day attacks. Virtualization enabled the low-risk deployment of HiHPs [11], isolating attack traffic from connected hardware and networks [12]. Owing to the popularity of honeypots, comprehensive surveys of their technology have appeared, with the most relevant being the most recent [13–15]. The *European Network and Information Security Agency* is a center of network and information security expertise for European Union member states [16]. In 2012, it issued the *Proactive Detection of Security Incidents* report, which presented the test and evaluation results for over 30 existing honeypots. The report indicated difficulties with honeypot usage, documentation, software stability, and developer support, and made recommendations for the future of honeypot development.

Emerging networking technologies have opened up new directions in honeypot deployment. Real-time visualization of global attacks is provided by Deutsche Telekom Honeypot Project [17]. This honeypot-development community produced *T-Pot*, which is used for collectively capturing and visualizing attacks on multiple well-known honeypots. Other substantial advances in honeypot deployment include the use of *software-defined networking* to design and deploy flexibly next-generation honeynets [18–21].

2.2 | Taxonomies

Zhang [22] introduced a taxonomy to standardize the development and deployment of honeypots. In this taxonomy, the function of honeypots as security mechanisms was classified as follows: *prevention*, *detection*, *reaction*, or *research*. As more devices were connected to the Internet, and threats accordingly evolved, Seifert introduced an updated taxonomy in 2006 [23]. Six core classes were proposed, each with its own set of values, as can be seen in Table 1. Technology and malware as well as honeypot development have since evolved. The heterogeneous nature of new devices coupled with emerging attack vectors affects the relevance of this taxonomy and is explored in Section 3. Further research has classified honeypot operations and has proposed a taxonomy to gain insight into honeynet architecture [24].

2.3 | Malware evolution

The development of the connected world of ARPANET and personal computers in the 1980s provided increased opportunities to create malicious software. ARPANET terminals infected by the *Creeper* worm posted a message and opened new connections to other terminals [25]. Fred Cohen coined the term *virus* in 1987, referring to a computer program that could infect a computer, make a copy of itself, and spread to other machines [26]. HTML facilitated the creation and expansion of the World Wide Web in the early 1990s. This expansion, in turn, facilitated the spread of malicious software.

TABLE 1 Seifert's taxonomy for honeypot development

Class	Value
Interaction level	High
	Low
Data capture	Events
	Attacks
	Intrusions
	None
Containment	Block
	Diffuse
	Slow down
	None
Distribution appearance	Distributed
	Standalone
Communication interface	Network interface
	Non-network interface
	Software API
Role in multitier architecture	Client
	Serve

Evolutionary web paradigms gave rise to new methods and malware variants.

It is a great irony that a botnet provided a census of connected routers on the Internet. The Carna botnet scanned the IPv4 address space to generate an image of fixed-line Internet connectivity [27]. This automated and repetitive program globally propagated and compromised devices, predominately routers, to measure the extent of Internet access. The original Creeper worm used an automated and repetitive method to propagate and infect terminals. The human factor involves designing, coding, and launching the malware. Infected machines may communicate with the command-and control (C&C) center, which is also operated by a human *botmaster*. Human naivety contributes to local compromise, as end users unwittingly enable content, but global infection predominately uses automated and repetitive methods. Botnets provide a mechanism for global propagation and control of cyber-attack infection. They are defined as large networks of compromised machines used to carry out further attacks [28]. Real-time systems are employed to detect and prevent malware infection. Firewalls, IDS, IPS, anti-virus, and access lists are some of the commonly used and widely accepted measures implemented to negate the effect of malware. Malware can exploit vulnerabilities at all layers of the open-system interconnection model. For example, physical wireless IoT infrastructure can be vulnerable to *war-driving* [29], whereas application services such as SQL and HTML are often targeted [30]. Coupled with this is the vast array of potential attack vectors available to malware developers. Every wired and wireless communication protocol becomes a potential entry point. Brute-force and dictionary attacks gain entry for subsequent exploitative software. From a security perspective, one benefit of deploying honeypots is knowing that all captured activity is malicious.

2.4 | Malware capture on honeypots

Honeypots actively seek to interact with cyber attacks by simulating vulnerable Internet services and devices. This is their raison d'être and can result in honeypot datasets rich in repetitive, automated attack sequences [31]. The analytical value associated with datasets of this type is longitudinal, providing spatiotemporal information on attack patterns [32]. Accordingly, honeypot detection tools and evasion techniques are designed into malware [33]. For example, *honeypot hunter* identifies honeypot functionality by generating false services and observing their execution [34]. Successful execution of these services identifies the connected device as a honeypot. Virtual-machine (VM) detection techniques identify the presence of virtual infrastructure by executing simple kernel commands [35]. Conficker and Spybot scan for the presence of a VM and terminate or modify their attack methodology accordingly [36]. Consequently, honeypot effectiveness is compromised, and the quality of the resultant dataset is reduced. Inadvertent termination of attack interaction truncates the attack sequences captured by the honeypot.

A vast collection of LiHPs, MiHPs, and HiHPs simulating server and client services are available on multiple attack vectors [14]. To coordinate this diversity, a versatile virtual-honeynet framework focusing on the management of automatic honeypot deployment has been proposed [37]. Predominately, LiHPs and MiHPs are simulations providing scripted responses. They are not at risk of being compromised and provide basic information for analysis, such as IP addresses and timestamps. These can be deployed quickly and require little maintenance. For better analysis, HiHPs are real systems actively engaging with the attacks. These honeypots gather additional information on attack code, C&C communication, and downloaded files, and they require time for their proper configuration, deployment, and maintenance. Malware developers use obfuscation to avoid the detection of installed software and C&C communications [38]. When it was released, the Mirai botnet spawned multiple variants with similar attack methods. A sample of the Mirai bot is shown in Table 2. Mirai is the dominant malware type captured by the adaptive honeypot detailed in Section 4. Owing to its dominance, it is used to standardize the optimization of the adaptive honeypot in Section 5. The type of malware captured will therefore depend on the interaction level, attack vectors, and services configured on the honeypot. The latest related survey found that most researchers tend to pose questions related to the attack source, target, and frequency [14].

TABLE 2 Mirai bot sample [39]

Sequence	Bot Command
1	/gweerwe323f
2	sudo/bin/sh
3	/bin/busybox
4	/gweerwe323f
5	mount
6	/gweerwe323f
7	echo -e '\x47\x72\x6f\x70'>//.nippon
8	cat//.nippon
--	---------------
38	/gweerwe323f
39	cat/bin/echo
40	cd/
41	wget http:// <RedactedIP>/bins/usb_bus.x86-O ->usb_bus; chmod 777 usb_bus
42	chmod 777 usb_bus
43	./usb_bus
44	/gweerwe323f

Thus, significant deployment periods are required to capture the necessary data and perform longitudinal analysis.

3 | CONCEPTUALIZED FRAMEWORK

The classes from Seifert's taxonomy in Table 1 are evaluated for relevance and are incorporated into the new framework. Adaptive and agile honeypots can be developed using the cyclic processes of (a) adaptive honeypot development, (b) time-limited deployment and data capture, and (c) honeypot optimization. The framework shown in Figure 1 facilitates honeypot development, expedites the capture of complete datasets, and ultimately leads to improved cyber forensics.

The top half of the framework uses five classes and one process for adaptive honeypot generation. The functionality and performance of the adaptive element are detailed in Section 4. The bottom half of the framework uses one class and two processes to enable agile functionality. The functionality and performance of the agile element are detailed in Section 5.

To mitigate against detection and inadvertent termination, and to handle repetitive truncated datasets, it is incumbent on honeypot developers and operators to implement alternative measures. These measures provide the adaptability to learn from attack interactions and the agility ensuring that honeypots may be expeditiously deployed, optimized, and redeployed. The existing taxonomy in Table 1 should be revisited to consider the relevance of its classes and actions. The automated and repetitive characteristics of malware affect this relevance, and therefore the classes and values should be examined for evolving malware threats. Even though some elements of this

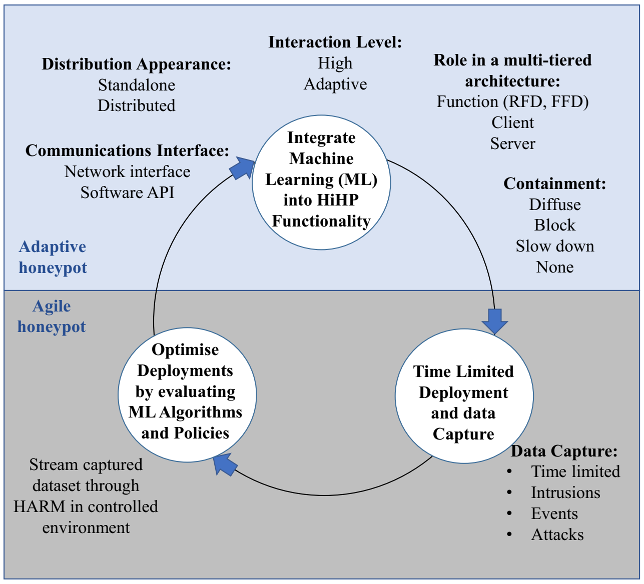
Interaction Level:
High
Adaptive
Distribution Appearance:
Standalone
Distributed
Role in a multi-tiered architecture:
Function (RFD, FFD)
Client
Server
Communications Interface:
Network interface
Software API
Integrate Machine Learning (ML) into HiHP Functionality
Containment:
Diffuse
Block
Slow down
None
Adaptive honeypot
Agile honeypot
Optimise Deployments by evaluating ML Algorithms and Policies
Time Limited Deployment and data Capture
Stream captured dataset through HARM in controlled environment
Data Capture:
Time limited
Intrusions
Events
Attacks

FIGURE 1 Framework for adaptive, agile honeypot development, and deployment

taxonomy remain valid, updated classes and values should be considered for a new framework for adaptive, agile honeypots.

- *Interaction*: Although LiHPs are available, they only simulate Internet services and capture basic interactions. Machine-learning functionality can be used to learn from attack code interacting with the honeypot. This gives rise to a new value for the "interaction level" class, namely, *adaptive*. HiHPs with adaptive abilities should be deployed for faster realization of attack sequences.
- *Data capture*: All automated interactions are captured by honeypots. Longitudinal deployment results in datasets containing repetitive data. The new value *time limited* should be added to the "data capture" class to improve cyber forensics. Adaptive honeypots prolong interaction and capture more relevant information. Section 5 demonstrates that adaptive datasets realize attack sequences faster. Continuing to deploy an adaptive honeypot is redundant. Optimization and redeployment ensure agility for such a honeypot. *Containment*: The values for this class remain highly relevant. Virtualization has abstracted the honeypot from the underlying architecture and has mitigated the ethical concerns regarding operations [8]. Virtual and cloud platforms improve containment by providing mechanisms that allow or restrict specific traffic types and protocols.
- *Distribution appearance*: Post compromise, malware will scan the environment for other potential addresses and services. New networking paradigms such as IoT will have a highly different distribution appearance and will encounter evolving propagation methods. Malware could evolve to exploit mesh networks or other non-traditional models. The adaptive functionality of honeypots can respond so that this interaction may safely be prolonged in a virtualized environment.
- *Communication interface*: The non-network interface is a redundant value. Automated and repetitive malware uses Internet communication protocols and software application programming interfaces (APIs) to propagate. Wired and wireless Internet protocols ensure permanent access to online services and are considered attack vectors. Non-network interfaces on devices are not Internet facing.
- *Role in a multitier architecture*: Malware does not discriminate post compromise. If a vulnerable device is accessible on an attack vector, malware will launch complex code structures to compromise the underlying architecture, regardless of whether the honeypot advertises client or server services. An adaptive honeypot will learn the best responses to realize all commands in an attack sequence. With IoT deployments gathering pace, reduced- and full-function devices give rise to complex mesh networks requiring communication and gateways to Internet services. Traditional client and/or server models should be expanded to include *function*.

4 | ADAPTIVE HONEYPOT

The development of artificial-intelligence and machine learning libraries heralded renewed interest in deploying honeypots. *Supervised learning* is ideally suited to retrospective analysis, as the algorithm can learn from or be trained by existing data. This learning is then used to classify new occurrences. Some examples of classifiers used on honeypot datasets are *linear regression* [40], *naive Bayes*, *support vector machines*, *decision trees*, *random forest* and *nearest neighbor* [41,42]. Similarly, *unsupervised learning* can organize data in different ways. Given a dataset, an unsupervised model can analyze the data to find underlying structures [43]. In *reinforcement learning*, an agent learns through trial-and-error interactions with its environment. The learning agent selects its actions based on previous experiences. Reinforcement-learning problems can generally be modeled using *Markov decision processes* (MDPs). Honeypots are examples of real-world problems in an incomplete environment. Reinforcement-learning methods facilitate the handling of MDPs, where the model can often be unknown or difficult to approximate. The proposed honeypot for automated, repetitive malware (HARM) involves the integration of reinforcement learning into existing honeypot technology to exploit the characteristics of malware. The reinforcement-learning state–action space and reward function are designed to increase the number of commands from the attack sequence.

During the reinforcement-learning process, the agent can select an action that exploits its current knowledge, or it can opt for further exploration. Reinforcement learning provides parameters so that the learning environment may determine the reward and exploration values. Throughout deployment, the honeypot is considered to be an environment augmented with reinforcement-learning functionality. The honeypot *states* in this environment are examined and changed using bash scripts. The states are represented as bash commands, such as *wget* and *sudo*. The reinforcement-learning agent performs actions on these states, such as *allow*, *block*, or *substitute* the execution of the attack scripts. The environment issues a reward to the agent for performing that action. The repetitive nature of the automated attack sequences facilitates learning over time. Eventually, the agent learns the optimal policy $\pi*$, which maps the optimal action to be taken for each state s. The learning process will converge as the honeypot is rewarded for each attack episode. This is a temporal difference method for on-policy learning and uses the transition from one state–action pair to the next so that the reward may be derived. HARM uses SARSA for the implementation of on-policy reinforcement learning (1). The reward policy Q is estimated for a given state s_t and a given action a_t. The environment is explored using a random component ε or exploited using the learnt Q values. The estimated Q value is updated through a received reward *rt* and an estimated future reward $Q(s_{t+1}, a_{t+1})$ that is discounted (γ). A learning rate parameter is also applied (α). The policy is evaluated at the end of each attack episode.

$$Q\left(s_t, a_t\right) \leftarrow Q\left(s_t, a_t\right) + a\left[r + \gamma Q\left(s_{t+1}, a_{t+1}\right) - Q\left(s_t, a_t\right)\right]. \quad (1)$$

From a functional perspective, HARM has the following elements:

- A Kippo honeypot modified to pass variables to the learning agent. Depending on the action selected by the learning agent, the honeypot will allow, block, or substitute attack commands.
- SARSA learning agent. This module receives the required variables from HARM and calculates $Q(s_t, a_t)$ using (1). It determines the responses selected by HARM and learns over time which actions yield the greatest amount of reward. PyBrain is a machine-learning library and is used to facilitate this reinforcement-learning functionality [44].

All external attack-command sequences interact with the honeypot only. HARM was developed to generate rewards on 75 states and is publicly available [45]. Figure 2 shows the relationship between HARM elements, the captured data, and the subsequent use and analysis of these data. Both Kippo and PyBrain are written in Python, providing seamless interaction. The attack commands are parsed as the current *state* and passed to the PyBrain module, which selects the action that will return the maximal value for a given state. The PyBrain module is separate from the Kippo honeypot and only communicates with the honeypot. Previous approaches, such as Heliza [46] and RASSH [47], have also used reinforcement learning to prolong the attack duration. However, these approaches assumed human attackers, and therefore they are inadequate for automated, repetitive malware. HARM demonstrated improved learning and reward functionality over previous approaches [48]. AmazonWeb Services EC2 was used to facilitate the deployment of Internet-facing honeypots: a normal Kippo honeypot and HARM. Kippo, PyBrain, MySQL, and other dependencies were installed on the adaptive honeypot EC2 instance, which was accessible through SSH for a 30-day period and immediately started to record repetitive malware activity. Initially, it logged dictionary and brute-force attempts and then captured other malware traffic, including a Mirai-like bot (Table 2). These commands represent post-compromise interactions. This bot became the dominant attacking tool over a 30-day period until over 100 distinct, repetitive attacks were recorded on both honeypots. Other SSH malware interacted with the honeypot as well. The novel state–action space formalism demonstrated improved adaptive learning and an increase in the number of commands captured by the honeypot compared with standard honeypots [48]. By further examining the malware code interactions, it can be seen that the adaptive honeypot

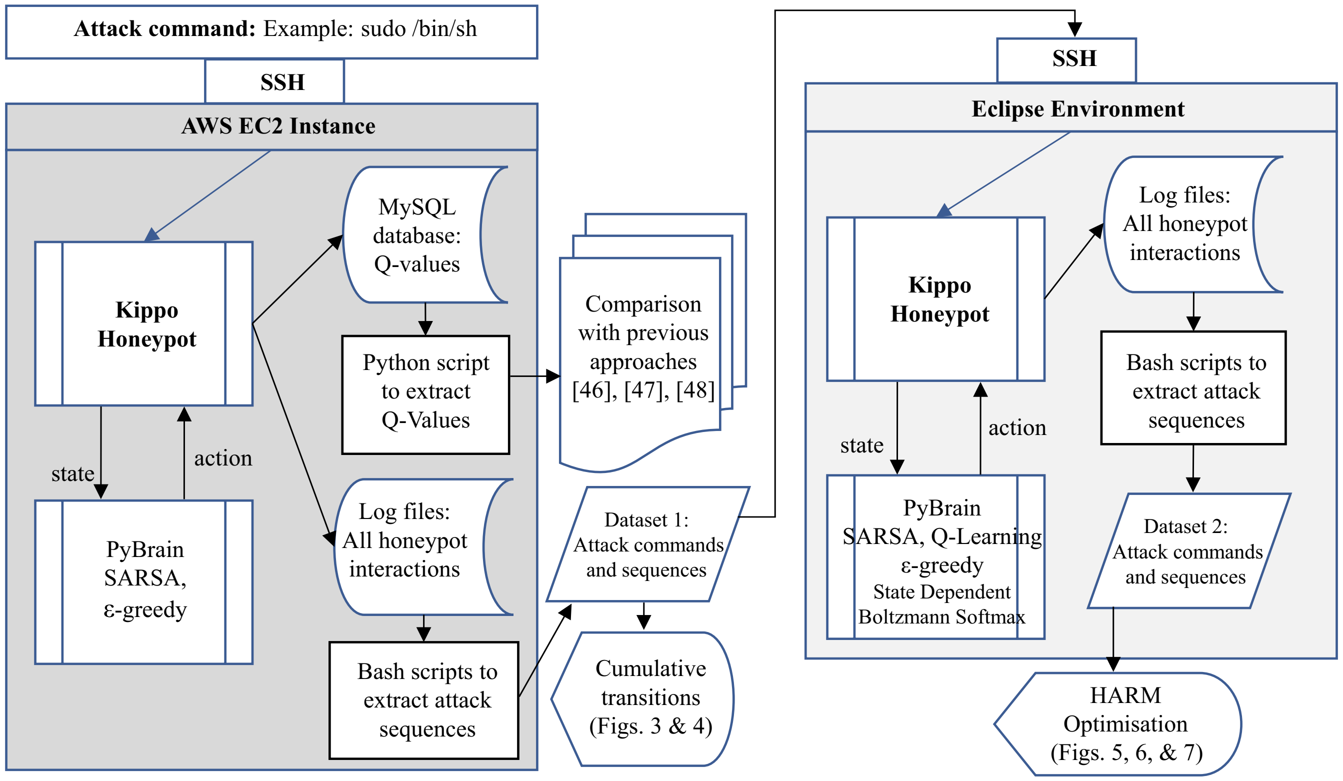
Attack command: Example: sudo /bin/sh
SSH
AWS EC2 Instance
Kippo Honeypot
state
action
PyBrain SARSA, ε-greedy
MySQL database: Q-values
Python script to extract Q-Values
Log files: All honeypot interactions
Bash scripts to extract attack sequences
Comparison with previous approaches [46], [47], [48]
Dataset 1: Attack commands and sequences
Cumulative transitions (Figs. 3 & 4)
SSH
Eclipse Environment
Kippo Honeypot
state
action
PyBrain SARSA, Q-Learning ε-greedy State Dependent Boltzmann Softmax
Log files: All honeypot interactions
Bash scripts to extract attack sequences
Dataset 2: Attack commands and sequences
HARM Optimisation (Figs. 5, 6, & 7)

FIGURE 2 Adaptive elements and data capture in HARM

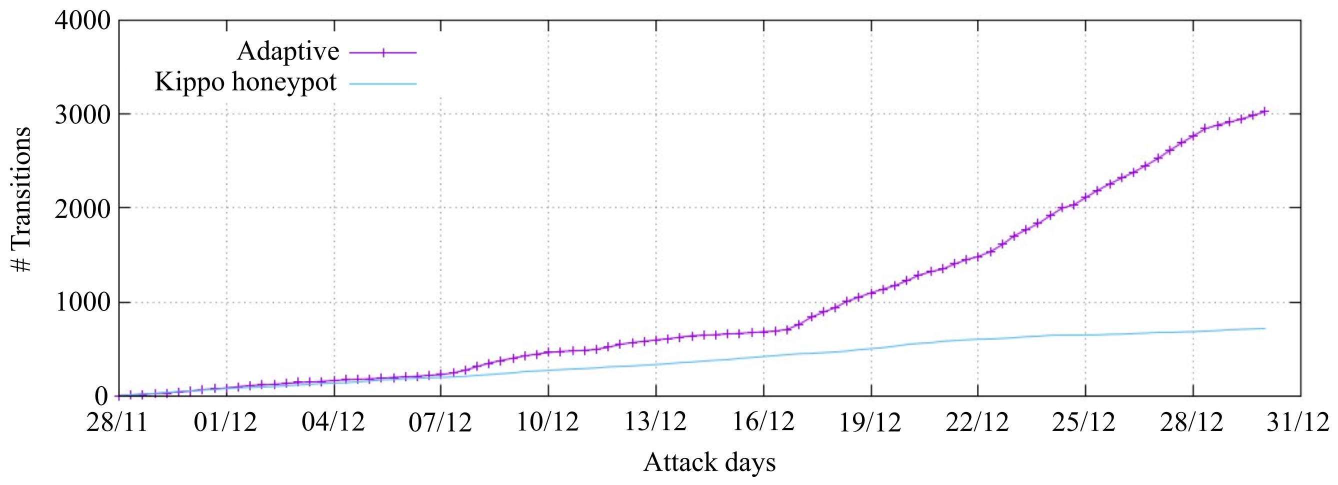

FIGURE 3 Cumulative-transition comparison

overcame detection techniques employed by malware [49]. When standard honeypots encounter these techniques, the captured datasets contain truncated sequences of automated and repetitive attacks. Figure 3 shows that, compared with standard honeypots, the adaptive honeypot exhibited prolonged interaction and improved data capture.

5 | AGILE HONEYPOT

Data capture on HARM can be used to evaluate reinforcement-learning algorithms and policies. Agile honeypot deployment is facilitated by Q-learning and SARSA under a variety of policy configurations. The adaptive ability of HARM results in the capture of a dataset containing prolonged attack interactions. Therefore, to ascertain whether continuing with the deployment is relevant, the deployment period should be considered. Optimizing and redeploying HARM may expedite the capture of more relevant information.

5.1 | Deployment period

The diversity of malware was discussed in Section 2.3. To further enhance the efficacy of honeypot technology, honeypot deployment should be considered. Longitudinal honeypots collect large datasets of repetitive attacks. These operate for long periods capturing repetitive, automated, and incomplete attack sequences. Analyzing the dataset captured by the honeypots provides further information (Dataset 1, Figure 2). The standard honeypot only interacted with the first eight commands in the Mirai attack sequence. This accounts for the linear evolution of cumulative transitions in Figure 3. It failed to realize the entire attack sequence. It should be noted that the repetitive, automated variant had 44 commands in the attack sequence (Table 2). Figure 4 shows the increases in attack interactions and identifies the distinct attacks resulting in this increase.

It is seen that the state–action space formalism for automated, repetitive malware rewarded the learning agent until

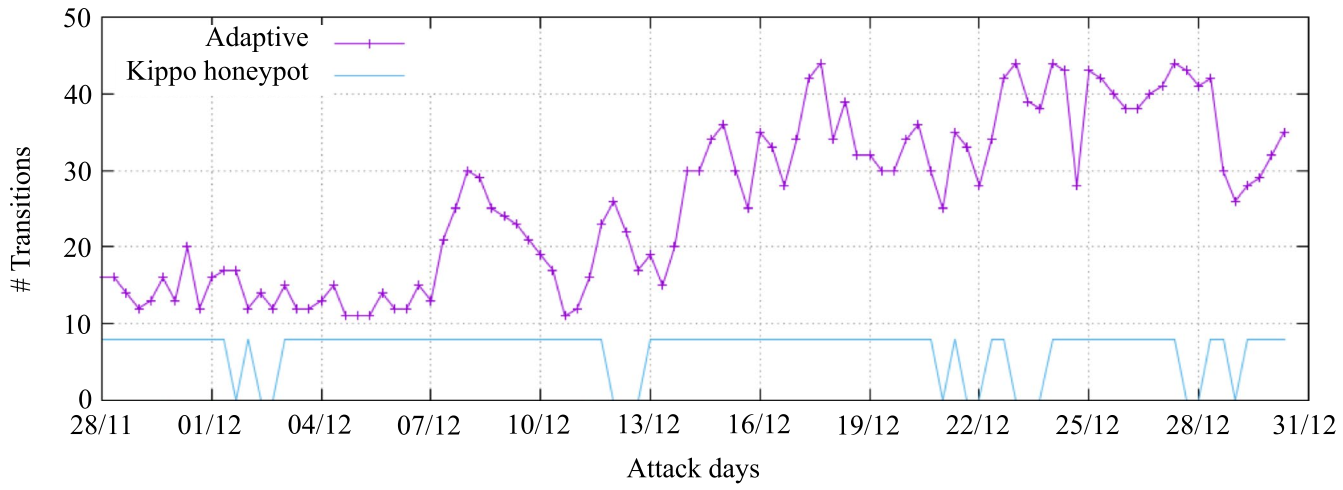

FIGURE 4 Realization of entire attack sequence

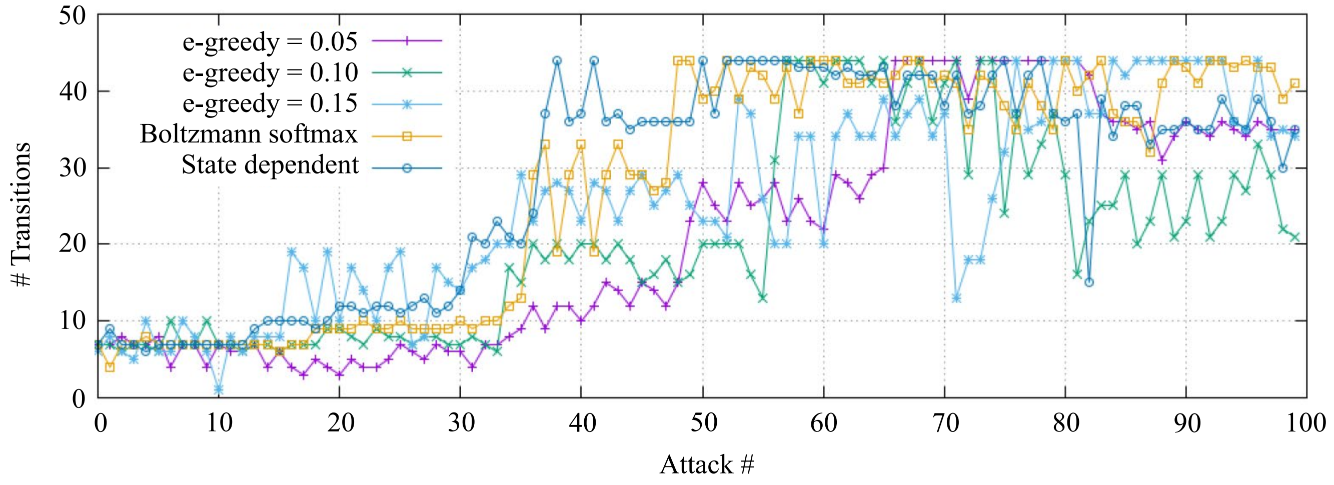

FIGURE 5 Performance of explorer policies using SARSA

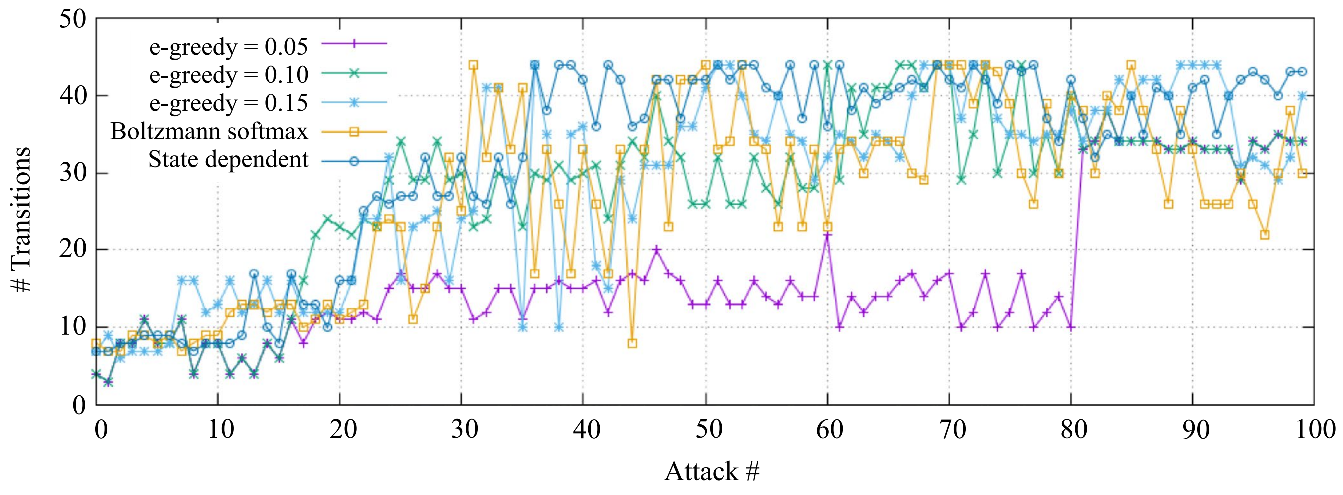

FIGURE 6 Performance of explorer policies using Q-Learning

all commands in the attack sequence were realized. The realization of the entire attack sequence was affected by other malware interactions on HARM. Moreover, the adaptive honeypot completed its task after 19 days (17/12/2017), rendering subsequent attack interactions and data collection redundant. The frequency of the Mirai variant was uniform throughout the duration of the deployments. The repetitive nature of the bot provides the adaptive environment with the opportunity to learn from each attack iteration. Continuing to operate the adaptive IoT honeypot was unnecessary after the realization of the entire attack sequence. For this short 30-day deployment, approximately 45% of the period and dataset were redundant.

5.2 | Optimization

The command sequences from the online deployment can function as an input stream into an implementation of HARM on a local, offline Eclipse environment. The input stream is parsed by a Python action daemon to reflect malware interactions captured by the online deployment. HARM was deployed on the Internet using SARSA (1). Python allows the adaptive honeypot to import and use SARSA or Q-Learning. It can also specify the ϵ-greedy explorer component and the following explorer policies:

- Epsilon greedy: A discrete explorer that mostly executes the original policy but occasionally returns a random action.
- Boltzmann softmax: A discrete explorer that executes actions with a probability that depends on their values.
- State dependent: A continuous explorer that disrupts the resulting action with added, distributed random noise.

The optimization process examines the performance of HARM, configured with varying combinations of learning algorithms and explorer policies against the malware

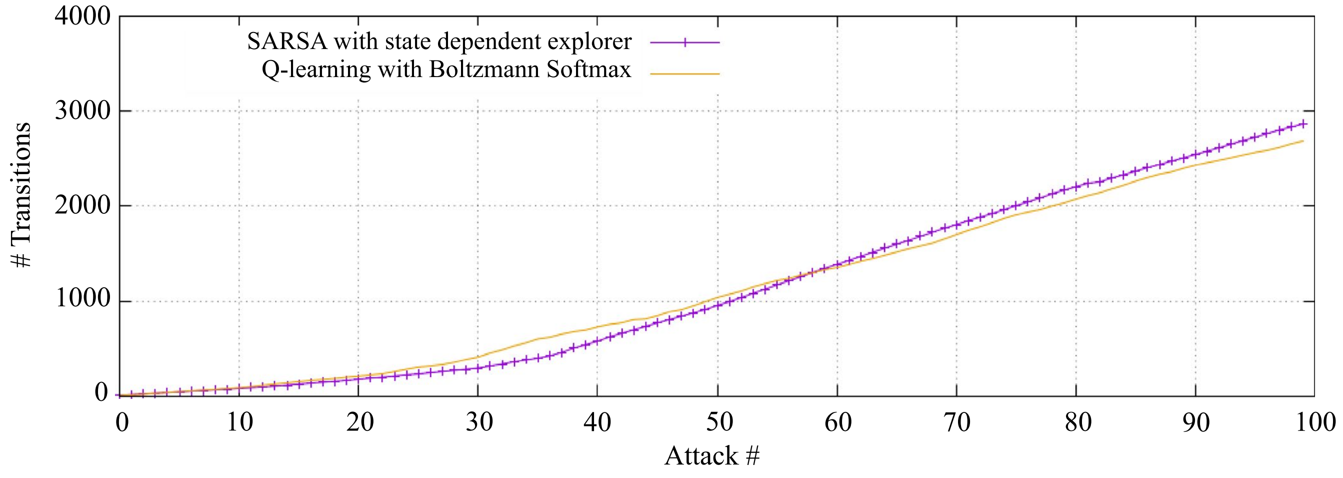

FIGURE 7 Comparison of SARSA/state dependent with Q-Learning/Boltzmann softmax

variant in Table 2. ε-greedy was streamed three times with parameter 0.05, 0.1, and 0.15. Therefore, for SARSA, the dataset was streamed five times: once for Boltzmann softmax and state dependent, and three times for ε-greedy. The process was then repeated for Q-Learning. The dataset from Section 4 was streamed 10 times through the adaptive honeypot in a controlled Eclipse environment. The first sequence (five times) with the learner set to SARSA, and the second sequence with the learner set to Q-Learning. The honeypot captured the interactions, actions, and number of commands in the Miraivariant attack sequence that represented transitions from s to $s_t + 1$. These transitions were collated, and HARM was reset after each streaming. Figures 5 and 6 show the performance of the explorer policies for SARSA and Q-Learning, respectively. It can be seen that SARSA and state dependent realized the entire attack sequence after 38 iterations (Figure 5). Q-Learning combined with Boltzmann softmax required 31 iterations for the same task.

Two controlled experiments were further conducted with the dataset. The best-performing combinations, namely *SARSA/state dependent* and *Q-Learning/Boltzmann-softmax* were configured on separate honeypots. These combinations performed best against one particular malware variant. The entire dataset was streamed through both combinations, and the cumulative number of transitions was computed. Figure 7 shows that the SARSA/state dependent combination performed approximately 6% better than the Q-Learning/Boltzmann softmax combination. This experiment highlights the agility requirement in honeypot deployment. Although Q-Learning/Boltzmann softmax realized the Miraivariant attack sequence faster (31 vs 38 attack iterations), SARSA/state dependent accumulated more transitions on the entire dataset, beginning at attack 59 (Figure 7). Continuous monitoring and analysis of honeypot datasets can provide new combinations for redeployment.

6 | CONCLUSIONS

Cyber forensics should evolve with malware methods. Technological advancements provide new attack vectors, and honeypots should adapt accordingly. In this paper, we proposed a new framework that uses machine-learning to construct adaptive honeypots; moreover, we proposed deployment methods rendering these honeypots agile. Adaptability and agility were demonstrated on a live honeypot dataset captured on an SSH attack vector. A number of surveys have presented honeypot technology targeting malware on other attack vectors. The development and operations of honeypots are based on older taxonomies designed to produce large datasets over longitudinal deployments. This framework can be applied to honeypots on other attack vectors. To be relevant for cyber forensics, honeypot technology requires an updated development framework that (a) is cognizant of malware evolution, (b) can use new technology in honeypot development to adapt to this evolution, and (c) can ensure deployment and optimization agility. Honeypot datasets suffer from repetition. Adaptive, agile honeypots can exploit the characteristics of automation and repetition to expedite the exposition of malware attack methods. The targeted Mirai bot was the dominant attacking tool. Other malware was highly infrequent throughout the deployment duration. Short adaptive deployments coupled with optimization and redeployment can improve the discovery of new malware variants.

CONFLICT OF INTEREST

The authors declare no potential conflict of interests.

REFERENCES

1. C. Kolias et al., *Ddos in the iot: Mirai and other botnets*, Computer **50** (2017), no. 7, 80–84.
2. Kapersky, *New iot-malware grew three-fold in h1 2018*, 2018, Available from: https://www.kaspersky.com/about/press-relea

ses/2018_new-iot-malware-grew-three-fold-in-h1-2018 [last accessed October 2018].

3. L. Spitzner, *Honeypots: Catching the insider threat*, in Proc. Annu. Comput. Security Applicat. Conf. (Las Vegas, NV, USA), Dec. 2003, pp. 170–179.
4. M. Oosterhof, *Not capturing any mirai samples*, 2017, Available from: https://github.com/micheloosterhof/cowrie/issues/411 [last accessed February 2018].
5. Y. M. P. Pa et al., *IoTPOT: Analysing the rise of IoT compromises*, in Proc. USENIX Conf. Offensive Technol. (Berkeley, CA, USA), Aug. 2015, pp. 1–9.
6. S. M. Bellovin, *Packets found on an internet*, ACM SIGCOMM Comput. Commun. Rev. **23** (1993), no. 3, 26–31.
7. N. Provos, *Honeyd-A virtual honeypot daemon*, in Proc. DFN-CERT Workshop (Hamburg, Germany), 2003, p. 4.
8. B. McCarty, *The honeynet arms race*, IEEE Secur. Priv. **99** (2003), no. 6, 79–82.
9. P. Baecher et al., *The nepenthes platform: An efficient approach to collect malware*, in Proc. Int. Workshop Recent Adv. Intrusion Detection (Hamburg, Germany), Sept. 2006, pp. 165–184.
10. G. Portokalidis, A. Slowinska, and H. Bos, *Argos: An emulator for fingerprinting zero-day attacks for advertised honeypots with automatic signature generation*, ACM SIGOPS Operat. Syst. Rev. **40** (2006), no. 4, 15–27.
11. X. Jiang, X. Wang, and X. Dongyan, *Stealthy malware detection and monitoring through VMM-based out-of-the-box semantic view reconstruction*, ACM Trans. Inf. Syst. Security **13** (2010), no. 2, 12:1–28.
12. I. Kuwatly et al., *A dynamic honeypot design for intrusion detection*, in Proc. IEEE/ACS Int. Conf. Pervasive Services (Beirut, Lebanon), July 2004, pp. 95–104.
13. N. Kambow and L. K. Passi, *The need of network security*, Int. J. Comput. Sci. Inform. Technol. **5** (2014), no. 5, 60986101.
14. M. Nawrocki et al., *A survey on honeypot software and data analysis*, 2016, Available from: https://arxiv.org/pdf/1608.06249.pdf [last accessed June 2020].
15. W. Fan et al., *Enabling an anatomic view to investigate honeypot systems: a survey*, IEEE Syst. J. (2018), no. 99, 1–14.
16. ENISA, *Proactive detection of security incidents - Honeypots*, 2012, Available from: https://www.enisa.europa.eu/publications/proactive-detection-of-security-incidents-II-honeypots [last accessed June 2020].
17. Deutsche Telekom, *Dtag community honeypot project*, 2018, Available from: http://dtag-dev-sec.github.io/ [last accessed October 2018].
18. S. Kyung et al., *HoneyProxy: Design and implementation of next-generation honeynet via SDN*, in Proc. IEEE Conf. Commun. Netw. Security (Las Vegas, NV, USA), Oct. 2017, pp. 1–9.
19. W. Han et al., *HoneyMix: Toward SDN-based Intelligent Honeynet*, in Proc. ACM Int. Workshop Security Softw. Defined Netw. Netw. Function Virtualization (New Orleans, LA, USA), Mar. 2016, pp. 1–6.
20. W. Fan and D. Fernández, *A novel SDN based stealthy TCP connection handover mechanism for hybrid honeypot systems*, in Proc. IEEE Conf. Netw. Softwarization (Bologna, Italy), July 2017, pp. 1–9.
21. W. Fan et al., *Honeydoc: An efficient honeypot architecture enabling all-round design*, IEEE J. Sel. Areas Commun. **37** (2019), no. 3, 683–697.
22. F. Zhang et al., *Honeypot: a supplemented active defense system for network security*, in Proc. Int. Conf. Parallel Distrib. Comput., Applicat. Technol. (Chengdu, China), Aug. 2003, pp. 231–235.
23. C. Seifert, I. Welch, and P. Komisarczuk, Taxonomy of honeypots, Victoria University of Wellington, School of Mathematical and Computing Sciences, 2006, pp. 1–19.
24. W. Fan, D. Zhihui, and D. Fernández, *Taxonomy of honeynet solutions*, in Proc. SAI Intell. Syst. Conf. (London, UK), Nov. 2015, pp. 1002–1009.
25. J. F. Shoch and J. A. Hupp, *The "worm" programs—early experience with a distributed computation*, Commun. ACM **25** (1982), no. 3, 172–180.
26. F. Cohen, *Computer viruses*, Comput. Security **6** (1987), no. 1, 22–35.
27. E. Le Malécot and D. Inoue, *The carna botnet through the lens of a network telescope*, in Proc. Foundations Practice Security (La Rochelle, France), Oct. 2014, pp. 426–441.
28. D. Dagon et al., *A taxonomy of botnet structures*, in Proc. Annu. Comput. Security Applicat. Conf. (Miami Beach, FL, USA), Dec. 2007, pp. 325–339.
29. J. Wright, *Killerbee: practical zigbee exploitation framework*, in Proc. ToorCon Conf. (San Diego, CA, USA), Sept. 2009.
30. B. Mphago et al., *Deception in dynamic web application honeypots: Case of Glastopf*, in Proc. Int. Conf. Security Manag., 2015, p. 104.
31. S. Dowling, M. Schukat, and H. Melvin, *A ZigBee honeypot to assess IoT cyberattack behaviour*, in Proc. Irish Signals Syst. Conf. (Killarney, Ireland), June 2017, pp. 1–6.
32. Y.-Z. Chen et al., *Spatiotemporal patterns and predictability of cyberattacks*, PLoS One **10** (2015), no. 5, e0124472.
33. P. Wang et al., *Honeypot detection in advanced botnet attacks*, Int. J. Inf. Comput. Secur. **4** (2010), no. 1, 30–51.
34. N. Krawetz, *Anti-honeypot technology*, IEEE Secur. Priv. **2** (2004), no. 1, 76–79.
35. T. Holz and F. Raynal, *Detecting honeypots and other suspicious environments*, in Proc. Annu. IEEE SMC Inf. Assurance Workshop (West Point, NY, USA), June 2005, pp. 29–36.
36. S. Khattak et al., *A taxonomy of botnet behavior, detection, and defense*, IEEE Commun. Survey Tutorials **16** (2014), no. 2, 898–924.
37. W. Fan, D. Fernández, and Du Zhihui, *Versatile virtual honeynet management framework*, IET Inf. Secur. **11** (2016), no. 1, 38–45.
38. I. You and K. Yim, *Malware obfuscation techniques: A brief survey*, in Proc. Int. Conf Broadband, Wireless Comput., Commun. Applicat. (Fukuoka, Japan), Nov. 2010, pp. 297–300.
39. M. Antonakakis et al., *Understanding the mirai botnet*, in Proc. USENIX Conf. Security Symp. (Berkeley, CA, USA), Aug. 2017, pp. 1093–1110.
40. E. Alata et al., *Collection and analysis of attack data based on honeypots deployed on the internet*, in Quality of Protection, Springer, 2006, pp. 79–91.
41. F. Vanhoenshoven et al., *Detecting malicious URLs using machine learning techniques*, in Proc. IEEE Symp. Series Comput. Intell. (Athens, Greece), Dec. 2016, pp. 1–8.
42. S. Nanda et al., *Predicting network attack patterns in SDN using machine learning approach*, in Proc. IEEE Conf. Netw. Function Virtualization Softw. Defined Netw. (Palo Alto, CA, USA), Nov. 2016, pp. 167–172.
43. P. Owezarski, *Unsupervised classification and characterization of honeypot attacks*, in Proc. Int. Conf. Netw. Service Manag. (Rio de Janeiro, Brazil), Nov. 2014, pp. 10–18.

44. T. Schaul et al., *Pybrain*, J. Mach. Learn. Res. **11** (2010), 743–746.
45. S. Dowling, *An adaptive honeypot using reinforcement learning implementation*, 2017, Available from: https://github.com/sosdow/RLHPot [last accessed December 2018].
46. G. Wagener et al., *Heliza: talking dirty to the attackers*, J. Comput. Virol. **7** (2011), no. 3, 221–232.
47. A. Pauna and I. Bica, *Rassh-Reinforced adaptive ssh honeypot*, in Proc. Int. Conf. Commun. (Bucharest, Romania), May 2014, pp. 1–6.
48. S. Dowling, M. Schukat, and E. Barrett, *Improving adaptive honeypot functionality with efficient reinforcement learning parameters for automated malware*, J. Cyber Security Technol. **2** (2018), no. 2, 75–91.
49. S. Dowling, M. Schukat, and E. Barrett, *Using reinforcement learning to conceal honeypot functionality*, in Proc. Joint Eur. Conf. Mach. Learn. Knowl. Discovery Databases (Dublin, Ireland), Sept. 2018, pp. 341–355.

AUTHOR BIOGRAPHIES

Seamus Dowling Seamus Dowling is a lecturer with Galway Mayo Institute of Technology (www.gmit.ie). He specializes in networking, telecommunications, virtualization, and cyber security. He initiated and manages the Cisco Academy at GMIT, Mayo Campus. He holds a BSc in Computing from Waterford Institute of Technology, an MSc in Computing from Sligo Institute of Technology, and a PhD from NUI Galway. Previous research includes analyzing DDoS attacks and developing intelligent honeypots for IoT malware. He has worked in both the private and public sector in IT support, administration, and management roles since 1993.

Michael Schukat Dr. Michael Schukat is a lecturer and researcher in the School of Computer Science at the National University of Ireland, Galway. His main research interests include security/privacy problems of connected real time/time-aware embedded systems (ie, industrial control, IoT and cyber-physical systems) as well as their communication/time synchronization and cryptographic protocols. Originally from Germany, Dr. Schukat studied computer science and medical informatics at the University of Hildesheim, where he graduated with an MSc (Dipl. Inf.) in 1994 and a PhD (Dr. rer. nat.) in 2000. Between 1994 and 2002 he worked in various industry positions, where he specialized in deeply embedded real-time systems across diverse domains, such as industrial control, medical devices, and automotive and network storage. To date, Michael has published over 80 research articles.

Enda Barrett Dr. Enda Barrett is a lecturer and research scientist at the National University of Ireland, Galway. In 2009, Enda completed an MRes in Computer Science, where he developed a rule-based classifier to detect cardiac arrhythmias from a three-lead ECG signal. In 2013, Enda received his PhD in Computer Science from NUI Galway. His PhD research investigated the application of a subset of machine learning known as reinforcement learning to automate resource allocation and scale applications in infrastructure as cloud computing services. Upon completion of his PhD, Enda joined Schneider Electric as a research engineer on a globally distributed research team. In 2019, he was awarded funding by Science Foundation Ireland to develop machine-learning and computer-vision approaches to detect individuals in noisy aquatic environments for assisting in drowning prevention and search-and-rescue operations. To date, Enda has published over 40 research articles, including 12 journal papers, 4 patents, and 25 conference/workshop papers.