Towards Self-Adaptive Cyber Deception for Defense

Jason Landsborough,[1] Luke Carpenter,[1] Braulio Coronado,[1]
Sunny J. Fugate,[1] Kimberly J. Ferguson-Walter[2] and Dirk C. Van Bruggen[2]

[1]Naval Information Warfare Center Pacific
[2]Laboratory for Advanced Cybersecurity Research
{landsbor, carpentc, bcoronad, fugate, kjfergu}@spawar.navy.mil

Abstract

Cyber attackers currently benefit from an asymmetric advantage, leveraging both features and flaws of networking protocols and software to discover and exploit vulnerabilities with impunity. Although significant work has been done to automate various cyber defenses, novel research remains in combining autonomic reasoning and defensive cyber deception. While many difficulties remain in creating a robust system, we have actively explored the utility of such systems for achieving effective cyber defense. Our current approach applies autonomic reasoning to the task of interfering with an attacker's movement through the Cyber Kill Chain® [1] by employing deceptive countermeasures. In this paper, we explore the integration of autonomic computing with insights from game theory and cognitive and behavioral psychology to create a system for adaptive cyber defense using deception.

1. Introduction

Longstanding trends to create systems with unrelenting reliability and consistency have led to enterprise computer environments and network topologies which are often static—retaining static addressing and system configurations for long durations. Today's systems also consistently and truthfully reveal system state and configuration status, naively informing potential attackers of system and software version information. The static nature and trusting naivety of existing network environments is advantageous to an attacker—they rely on the veracity and relevance of the information they obtain, and can often archive it for later successful use.

A recent trend in network defense has been the exploration of techniques to disrupt the information gathering reconnaissance phase. One method to perform such disruption is through the use of defensive cyber deception such as honeypots, honeynets, or decoys [2]. Our work focuses on the use of lightweight, low-interaction decoys due to the speed and ease at which they can be configured and managed and the broad class of defensive strategies which they enable. Configuring a high-fidelity honeypot often requires a similar amount of effort to configuring a true server. In some cases the costs of configuring and managing such a system may be even higher. The intricacies of creating fake systems, traffic, and user behaviors can be daunting. Configuring a decoy is simpler due to the decoy only being realistic to network scanning and initial network communications, but not requiring its internal configuration to be realistic. The idea of a low-interaction decoy is that its realism is only superficial. However, while significantly less work is required in creating and maintaining decoys compared to honeypots, it is still uncommon for their configurations to be regularly changed after deployment. Fine tuning of decoy configurations on an ongoing basis is seen as cost prohibitive due to the administrative burdens involved in assessing, planning, and implementing configuration changes. As a result, existing decoy systems tend to be a relatively static defensive measure which limits its long term utility against persistent or knowledgeable attackers.

Not only do our defensive systems tend to be statically configured, but generally fail to adapt to an attacker's actions or adapt in ways that are easily predicted and countered. Creating defenses capable of automatically adapting strategies would require a sophisticated understanding of both the system being

URI: https://hdl.handle.net/10125/70857
978-0-9981331-4-0

adapted and the intent and actions of an attacker. It is likely that such a system must be capable of both implementing deceptive tactics and reasoning about an attacker's use of deception and counter-deception. To this end, we investigate the use of autonomic computing to reason about and control defensive cyber systems. In particular, we have most recently applied these techniques to managing an adaptive cyber deception system [3]. By combining autonomic computing and cyber deception, we believe we can enable a defensive system to gain an initial defender advantage and counter attacker actions through automated adaptation. Our system reasons about an attacker's actions and estimates the attacker's beliefs based on their interactions with decoys deployed on a local computer network and interspersed among, and initially indistinguishable from, the real hosts. The intent of the overall system is to deter, delay, or deny attacker access to the real assets with speed and scale that would be impractical to achieve with a human defender.

2. Background

2.1. Cyber Deception

Currently, best-in-class defenses automate the configuration and deployment of technologies such as rule-based intrusion prevention systems which automate the blocking of malicious network communications. Tied to the increasing sophistication of systems is a strong desire to maintain full cognizance of system behaviors, properties, and risks. Fortunately, system owners and defenders exist in an information rich environment, often with comprehensive knowledge of system functions and defenses. However, such knowledge is not complete. There are gaps as well as misconfigurations, faulty logic, and potentially exploitable defects. It is exactly these gaps which are leveraged by malicious actors.

It is widely acknowledged that attackers have an asymmetric advantage against defenders who must defend all parts of a network at all times. Fortunately, attackers are usually working from an information austere vantage point. Prior to interacting with a network or system, an attacker may know very little about its purpose, defenses, or the risks associated with interacting with it. This knowledge asymmetry can be exploited by cyber defenders through the use of defensive cyber deception which adds false or misleading information into the cyber environment, increasing the difficulty of learning a system or network's true configuration, and increasing the odds of detecting and stopping an attacker's actions.

Cyber deception as a whole can provide an entire class of moving target defense (MTD) strategies and techniques which can have a significant impact on attack success and on the overall security of a system. Prior research has demonstrated that adaptive attackers are more challenging to defend against [4], and likewise, our defensive strategies must be strategically unpredictable, rather than randomly changing. The intentional masking of key system information can force attackers to use heavier handed approaches — such as when a system is configured to selectively ignore traffic unless it is from a pre-determined and trusted source, forcing attackers to use side-channel inference techniques to learn a system's port and service configuration [5]. Active injection of false system features can lure an attacker into attempting to exploit a seemingly easy target, resulting in early detection and defensive response actions. Moreover, providing an attacker with verifiable indicators that defensive deception is being used can act as an effective deterrent against further actions, supporting the decision for an attacker to avoid easy targets [6].

2.2. Autonomics

Inspired by the human autonomic nervous system, autonomic computing aims to address increasing complexity in software systems [7, 8]. Autonomic computing adopts the Monitor, Analyze, Plan, Execute (MAPE) control-loop paradigm to enable a set of self-sustaining properties (self-configuration, self-optimization, self-protection, and self-healing) in managed systems. An autonomic manager practicing the MAPE control loop generates a knowledge base from sensors into the system in the Monitor and Analyze stages. Automated reasoning in the Plan and Execute stages select adaptation strategies through decision-making algorithms to effect change in the system. System changes are then fed back into the autonomic manager through sensors in a closed loop to ensure overall system health. Often, tuning systems requires continuous human oversight but self-optimizing systems are able to tune their own parameters to maintain peak operational efficiency. Autonomics provides an introspective view to systems by monitoring internal system processes to identify and address performance issues thereby enabling self-protection and recovery.

One such autonomic manager is the The Rainbow Autonomics Framework [9]; a research tool developed at Carnegie Mellon University to explore autonomic self-management of systems. The Rainbow framework is a non-intrusive approach to adapt the behavior of

systems through heuristic driven activation of various actuators. Rainbow has been used in domains such as industrial control systems [10], robot navigation [11], naval command and control [12, 13], and network security through DoS mitigation [14]. System architects can build their systems with Rainbow in mind or design de-coupled interfaces to enable self-management after the system has been designed with minor or often no changes to the system. Rainbow's *System Layer* defines the interfaces into the system, while an *Architecture Layer* models the system and reasons over adaptation strategies. A *Translation Infrastructure* enables bidirectional communication between the two layers. As a realization of the MAPE paradigm, Rainbow enables self-managing properties on systems through the following components:

- ***Probes*** are sensors into the system to collect system properties reflecting current system state.
- ***Gauges*** receive information from probes through the *Translation Infrastructure* to maintain Rainbow's representation of the system which is typically expressed in the Acme formal description language [15].
- ***Strategies* and *Tactics*** are established at design time in Stitch, a language for expression of adaptation decision trees [16]. Strategies encompass the breadth of adaptation responses available to the autonomic reasoning process while tactics are used by strategies to perform a specified action
- Closing the control-loop are ***Effectors*** which are called by tactics to effect change in the managed system. Effectors, like probes, are tailored to the system and utilize the *Translation Infrastructure* for communication.

System architects establish design assumptions through architectural *constraints* in Acme models of the managed system. The *Architecture Evaluator* detects constraint breaches to signal the *Adaptation Manager* that a response is necessary to address a fault in the system. Latency aware adaptation considers the delay between strategy execution and observation of the desired effect on the system [17] which informs the tracking of historical strategy effectiveness used in utility calculation. Complex multi-step strategies are achieved through decision trees where the outcome of a step influences the selection of the next path in the tree. Strategies reference dynamically updating Acme model properties at every step to ensure decision-making is performed with current knowledge of the system. Inherent to Stitch strategies is the *strategy probability* property which enables the game-theoretic concept of mixed strategies. In Section 3.3 we describe using the Rainbow autonomics framework to create a prototype system for adaptive cyber defense using deception, with strategies that leverage game theory concepts to aid in strategy selection.

2.3. Motivation

Game theory is used to study decision making in situations when individuals have conflicting goals and uncertainty concerning the other player's motivations. Game theoretic modeling and analysis have been applied to an incredibly wide-ranging set of application areas, including cyber defense, and more recently, defensive deception. Of particular relevance to our work are games which model imperfect information, where the sequence of decisions made is not readily available to all players. In cyber environments neither defenders nor attackers have perfect information about the environment [18]. Defensive cyber deception creates a knowledge asymmetry in favor of the defender which can be modeled using extensive-form games where each player has access to only a subset of the parameters, game history, or game structure [19].

An attacker operating with incomplete information may incorrectly presume that a system is not vulnerable and ignore it – the system is protected by looking uninteresting. Such a scenario can be modeled by removing information from an attacker's view of the game environment. Alternatively an attacker unwittingly interacting with a decoy will likely disclose their presence to defenders. This can be represented as a new state in the defender's game model representing a detection event which is enabled by the decoys.

An attacker in a defensive deception environment is operating with incomplete information about the actual utility of their own moves. If the attacker is rational and aware of deception then at worst they will expend additional resources in disentangling real from decoy; a benefit to the defender. If however, the attacker is rational and unaware or boundedly rational, then the game is likely to tip further in favor of a deceiving defender. In game theory, purely rational players are greedy and selfish by definition and will only choose strategies they believe will produce the highest possible payoff. However, if a player lies about their rationality, or plays sub-optimally, this can lure a rational player into taking advantage of the apparent lapse [3].

Of particular relevance to our implementation of cyber deception game within an autonomics framework is the use of apparently poor strategies as a lure or

deterrent. For example, once an attacker is detected we can interfere with their network communications in various ways. In particular, we can slow or degrade communications to specific machines on the network. This technique provides the appearance of poor network performance which can nudge an attacker into moving to a new target—useful if the attacker's original target was a valuable or vulnerable system. Alternatively, when an attacker interacts with a decoy, we might intentionally force the connection to drop, simulating an intrusion prevention technique and potentially indicating to the attacker that their original target was valuable enough to warrant such a sophisticated defense.

Normally, a rational player would never select a sub-optimal strategy. However, observations of poor play by the defender can affect an attacker's perceptions and beliefs of the rationality of the defender. A defender who intentionally plays sub-optimally as a deception or feint can result in defensive advantages. Cyber deception allows a defender to fool an attacker into taking a apparently rational greedy strategy leading the attacker to incur unexpected detectability, deterrence, or delay. Lastly, a deceiving defender, having pre-staged deceptive strategies, has a more comprehensive and more correct history of the actions taken by both players. While both players are operating off of incomplete information, the beliefs of the defender are grounded in more comprehensive knowledge of the cyber environment while the attacker's beliefs are tampered with and manipulated by false information provided by the deception.

Oppositional human factors is the use of human factors theory, technique, and practice to disrupt the usability and utility of computing systems and networks for malicious actors [20]. It involved using research that has been done to make users and defenders more effective and efficient and inverting it in order to disrupt and impede cyber attackers. Examples have been documented from red teaming experiments of attentional and decision-making biases evident in cyber attack behaviors, postulating that defenses can be designed to specifically exacerbate these biases for defensive gain [21]. While more research is needed to formalize which biases can be most strongly induced in cyber attackers and how [22], we believe that adaptive cyber deception techniques will have significant utility in triggering biases in cyber attackers. Our prototype sets the groundwork for an adaptive system that can automatically take these types of actions.

A primary goal of automation applied to cybersecurity is to reduce the workload and monitoring required of human operators overseeing the network. As the complexity and scale of computer networks increases, so does the cognitive demand, fatigue, and stress [23] experienced by human operators conducting cyber threat management. Other benefits of automation include quicker reaction time to observed threats and the exploration of new types of strategies enabled by autonomic reasoning that cannot be achieved through traditional manual means. However, increased automation raises concerns over who is really in control in the decision-making process and whether operator situational awareness can be maintained. Therefore, it is important to consider both sides of the human-machine team and understand how to best utilize both teammates. To best utilize automation in the cyber security domain, it is important to consider several factors including: which information processing stage automation is applied to [24, 25], the ability to adapt to changes in the environment [25, 26], and the level of trust placed on the automation by human operators [27, 28].

When automating actions and responses in an adaptive cyber deception system, it is important to consider how the role of human operators change. Having a human-in-the-loop at a low level of reasoning would not enable the speed needed to effectively react to a cyber attack. Therefore, human operators may provide input at a more strategic level in the decision making process. A defensive cyber deception system may have many different strategic goals which will lead to the selection of very different tactical response options which drive each action selected by the system. There are a range of strategies which include: collection of intelligence on the attacker (the goal of many honeypots), delaying attack progress (perhaps until the defender's team can complete a specific security task such as patching a vulnerable system), and denying access (such as preventing an attacker from accessing specific systems which are known to be particularly vulnerable/valuable). In our prototype, this strategy is an operator selected goal, which allows for human-machine teaming, and provides the system with a subtree of response options available based on the currently selected goal.

3. Adaptive Cyber Deception

3.1. Test Scenario

Our cyber scenario was inspired by the APT29 and APT3 scenario descriptions in the MITRE ATT&CK® framework, but focused primarily on the subsets concerned with theft of credentials, pivoting, and data exfiltration [29]. A successful attack in both of these scenarios consists of leveraging an initial foothold on the network, pivoting to a system with privileged access

to information or additional computing resources (such as a database); exploitation the computing resources using the additional privileges gained via the pivot; and exfiltration of private or confidential data.

In our scenario, we have a small network consisting of two Windows 7 client machines and a server hosting a database. The scripted attacker starts with a foothold in the network, having already gained access through a successful phishing attack, and launches automated attacks from a Kali Linux machine. During this initial compromise the attacker is also able to obtain a list of potentially vulnerable Windows clients on the network. While the test network environment is much smaller than those commonly found in commercial enterprise networks, it presented true to life vulnerabilities and system configurations, running in virtual machines, to the simulated attacker. The network and system services are similar to those commonly found and exploited on commercial networks.

The attacker's goal is to move laterally through the network after establishing persistence on one of the Windows 7 clients, obtain credentials for the database server, and then exfiltrate valuable data from the database. The attacker accomplishes this by automating both the reconnaissance and explotation steps using the Metasploit Framework [30], Metasploit's native scripting engine, and custom Python code. The attacker attempts exploitation of the Windows clients using Metasploit's SMB psexec exploit module [31] with a Meterpreter reverseTCP shell payload. Exfiltration of the database server is accomplished using FTP.

The attacker begins by choosing two clients available on the network and attempts to exploit them to establish persistence. The attacker needs just one of the clients to successfully be exploited. Before exploitation they conduct a port restricted NMAP SYN scan to verify that targeted services are running on the chosen clients. If they are successful in moving laterally into a client system the attacker is able obtain the IP address and credentials for one of the available database servers. They will again conduct the scan to verify service availability. If the FTP service is running they then attempt to exfiltrate roughly five Gigabytes of data using the credentials obtained from the Windows clients.

As a defender, we attempt to disrupt the attacker's forward progress and success through the use of decoys and actions orchestrated by the autonomics framework. Decoys are initially deployed as a mirror of existing systems: two decoy clients and a decoy server.

3.2. Test Conditions

The adaptive deception system was tested with two operator set goals (delay and deny) against two control conditions, (no decoys and static decoys). For each conditions, each run is repeated 100 times:

- Control - No decoys are present. Rainbow is not present.
- Static Decoys - Decoys are present, but Rainbow is not, resulting in *static* decoy defense.
- Delay Goal - Adaptive decoys; adaptation strategies are configured for delay tactics.
- Deny Goal - Adaptive decoys; adaptation strategies are configured for deny tactics.

From the attacker's perspective there are four outcomes:

- Success - The attacker was able to successfully exfiltrate the database.
- Pivot failure - The attacker was unable to maintain persistence on a real machine.
- Failure - The attacker was able to pivot, but unable to exfiltrate the database.
- Timeout - The attacker script timed out.

In the Cyber Kill Chain® [1] , Success is an attacker completing 'actions on objectives' in stage 7, Pivot failure is failing to complete 'installation' in stage 5, Failure is failing to reach stage 7, and Timeout indicates they timed out during stage 7.

The *delay* and *deny* goals were selected for this experiment as they represent aligned but not identical desired outcomes. Categorically, we only see timeout failures in the delay strategy as our current deny tactics have an immediate and obvious effect on the attack techniques used. However, while it should be clear that imposing an increased latency or other form of delay is not the same as kicking an attacker out of the network, a large enough delay may result in an equivalent outcome. Similarly, immediate denial of an attacker to a particular network resource may delay immediate progress by an attacker. However, such an attacker might repeat an action at some point in the future such that the current denial strategy is ineffective. In both conditions, the selected strategies are implemented as a mixture of individual technical methods (tactics, in Rainbow's terminology) which are selected at random according to pre-defined probability distributions.

3.3. Adaptive Cyber Deception System

In this section, we detail the design and implementation of our adaptive cyber deception system [32]. We are using a *decoy system* which is capable of deploying hundreds of lightweight containers. These containers have unique IP and MAC addresses and run real services, but are made much lighter weight than Virtual Machines through the use of shared underlying operating system components and resources. Network packets sent to a decoy will generate alerts, which we use as high-confidence tripwires, as real users would not normally interact with decoys as they do not provide useful services.

The decoy system is configured and managed via a REST API, used to probe and enable effects on the network using Rainbow. Using this API, we can modify decoys, add decoys, remove decoys, increase packet delays to decoys, decrease packet delays to decoys, and spoof TCP reset packets to real or decoy machines on the network to terminate an active TCP session.

We implemented the concept of guiding the automation's decision-making process through explicit human-selected goals. The user can select a goal of 'delay' to attempt to slow down an attacker or 'deny' to attempt to stop the attacker. These high level goals each result in multiple configuration changes to Rainbow for both likelihood of selecting a particular adaptation response and whether a given action is considered at all.

It may be initially unclear why a cyber defender would accept the arguably weaker goal of *delay* over the apparently stronger goal of *deny*. One might ask why we would ever delay an attacker rather than to eject them from our system or networks. However, if, as in our scenario, the attacker already has a foothold on the network, then a deny strategy would potentially remove them from the systems that we are currently aware are compromised, but unfortunately, such a heavy handed strategy both provides a strong indicator to an attacker that their recent activities have been detected and fails to inform the defender concerning what other systems an attacker may already be in control of. The attacker may still have access to the network or may regain this access. Drastic measures may tip off the attacker and result in destruction of important data or access. Delaying the attacker in such situations may enable defenders to better prepare for eventual actions taken to remove the attacker's access altogether. Further, we may not be entirely confident that the actions are representative of an actual attack. Presenting alternative strategies might help to tease out a true attacker from a false positive. For example, if we detect an interaction with a decoy, increasing the delay to this system might cause a normal user to simply stop trying to connect to it. Alternatively, an attacker may also cease their activities due to frustration but is much more likely to connect to other systems based on their similarities to systems they have already interacted with in hope of finding one that is an amenable target. By taking the more subtle approach of delaying traffic we can also increase attacker cost even when they are not fully acknowledged by defenders.

Rainbow asynchronously probes for information about the decoy system using the REST API. These probes provide information on decoy state and configuration, including the services that they run, any alerts generated through interaction with decoys, and information concerning strategies which are being implemented to interfere with attacker actions (such as adding delays to network communications).

Information gathered by probes is fed to gauges which filter the data and update Rainbow's model of the system. Components in the model include real devices, decoy devices, compromised devices, and the operator-specified goal. When alerts are generated (due to packets sent to, or authentication attempts made on, decoys), connectors between compromised devices and decoy devices are made. Constraints are tied to both the number and types of alerts.

To construct strategies for our system, we took inspiration from recent advances in game theory, including prior work on game theory for cyber deception and Stackelberg games, where players interact in a leader/follower fashion over many rounds [33]. Rainbow's adaptation manager selects strategies based on expected utility. This approach excels in traditional control theory applications, but we believe may be sub-optimal in adversarial cyber defense scenarios, as attackers might learn how to use an autonomic system's reasoning against the system. Much like how defenders want to learn of their attacker's behavior and goals to determine how to best counter them, attackers are able to learn characteristics of defensive systems to gain an advantage. Game theory provides a mechanism for defending against attackers knowing which strategy their opponent will use through the use of mixed strategies, where a player makes choices based on a probability distribution across available strategies.

We take this same approach for Rainbow strategies. We first consider all possible branches in a strategy for a given architectural constraint, and have a weight assigned to each branch that contains one or more tactics. Based on operator selected goals — delay or deny in our example — weights are adjusted. When a goal is selected, as in the case of deny, entirely new tactics can be enabled or disabled by the system and

	Attacker Successes	Pivot Failures	Exfiltration Failures	Timeouts
Control	100	0	0	0
Decoys	42	19	39	0
Delay	40	23	27	10
Deny	11	78	11	0

Table 1. Results of 100 runs for each condition. Success or failure is from the attacker's perspective.

overall parameters for the system are modified.

While our decision procedure closely matches what one would commonly find in an extensive form game model, in reality, the defender-attacker interaction may extend over many rounds, including rounds where no action is taken, rounds where unobserved actions are taken, or rounds where multiple actions are taken by one or both players. So, while we generally conceive of the game structure and decision procedure using a game theory framework and theoretical models, our implementation does not clearly distinguish between rounds. This represents a disconnect between theory and practice, but has allowed for a straightforward and scalable implementation. We are currently working to extend the Rainbow adaptation manager and strategy executor to more closely represent our theoretical models which rely on reasoning over the full history of actions for both players in a Stackelberg game [33].

4. Results

In this section we detail the results for each 100 runs per condition, as described in section 3.2

4.1. Control

In the control condition, there are no decoys and Rainbow is not present, so there are no obstacles in the attacker's way. As a result, all 100 runs were a success for the attacker (see Table 1). The average successful run time was 250.05 seconds and is indicated by the dotted line in the following figures for comparison.

4.2. Static Decoys

In the static decoys condition, decoys had been pre-configured and pre-deployed, one for each real machine in the network. As shown in Table 1, in this condition, the attacker was only successful 42 times, was unable to exploit and pivot 19 times, and failed to exfiltrate the database 39 times. Pivot failures can result from selecting only decoys to exploit, or also choosing one decoy and one real machine, but not successfully establishing a persistent session with the real machine due to Metasploit failing.

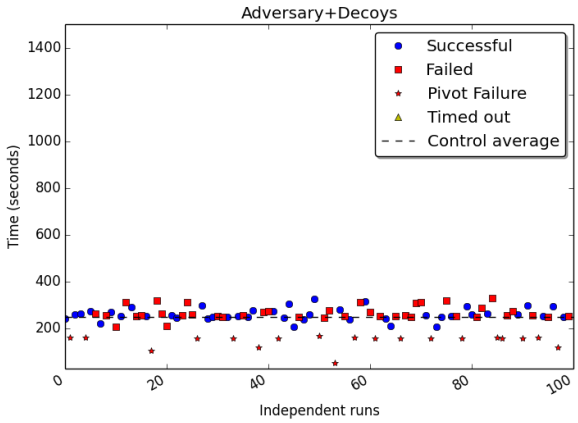

Figure 1. Static Decoy results

Figure 1 shows the results for all runs. The average success time was 261.80 seconds, slightly higher than the control average. The average time for pivot failures was 148.61 seconds, whereas the average time for exfiltration failures was 268.71 seconds. The overall average run time was 243.0 seconds.

4.3. Delay Goal

In the delay condition, decoys had been pre-deployed as in the static decoys condition, but the adaptive deception system with delay as the user-defined goal is then activated. This enabled strategies to be selected with tactics involving: modifying a decoy, adding a new random decoy, adding a decoy matching the decoy that generated the alert (which would possibly add new targets that the attacker is interested in if they are spending time focusing on a particular decoy), and increasing packet latency depending on the severity of the alert.

The attacker in the delay condition was successful 40 times. Because the delay strategies ultimately do not stop the attacker, we expect a similar success rate as the decoy scenario without additional intervention. The attacker had 27 exfiltration failures and 23 pivot failures, each for similar reasons to the static decoy condition. What is new in the delay condition is the presence of 10 timeout conditions. This is largely due to our tactic which increases packet delays for network traffic sent to and from decoys. Even with limited network interaction by the attacker, we do notice benefits of this tactic. For clarification, the timeout condition applies to database exfiltration timeout. A 'timeout' for attacker-client interaction was set to 600 seconds, 10 times the control interaction length so the scenario would not take hours

or days per run, but would not result in a 'timed out' outcome if the attacker could still pivot from a different client. Depending on the amount of delay applied to decoys, even the limited interaction in this scenario could take hours or days without a timeout. The increased time among runs is shown in figure 2. The average successful run was 630.23 seconds, average pivot failure run was 817.88 seconds, average database exfiltration failure was 637.23 seconds, average run resulting in a timeout was 753.60 seconds, and the overall average run time was 687.62 seconds.

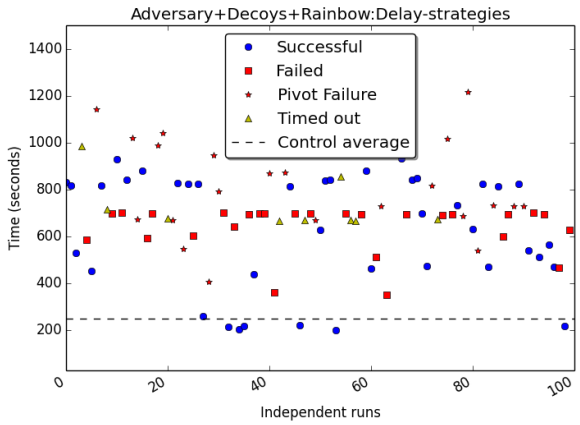

Figure 2. Delay results

Although attacker success isn't ultimately affected, they are slowed down significantly, which was the operator's selected goal. This is still a defender success over the static decoys condition as the attacker incurs additional costs with their other interactions. In addition to the cost of time wasted by the attacker, this could also permit more time for a defender to maneuver, such as adding additional defenses or deploy threat hunters to gather intelligence. There were some cases where even though the attacker initially triggers a delay strategy, but where further interaction is primarily with non-decoy hosts, resulting in limited utility in adding a delay. Further, we did not observe much benefit to adding additional decoys as the attacker did not enumerate the entire network, instead relying on information obtained from a compromised machine. If an attacker interacted with more decoys, we would expect to see a greater impact with these strategies.

It might be noted that an attacker with knowledge of the delay strategy may simply modify their attacks to be resistant to timeouts. This is fair criticism of the strategy if judged from a *denial* perspective. However, for an attacker to be free from the timeouts they must increase the overall latency they are willing to incur. Since this latency is under the control of the defender and can be increased at will, this is a losing condition for the attacker. Counter-intuitively, a better response by an informed attacker would be instead to increase sensitivity to timeouts and redraw a new target (without replacement), minimizing the effect of the delay strategy on their overall progress.

4.4. Deny Goal

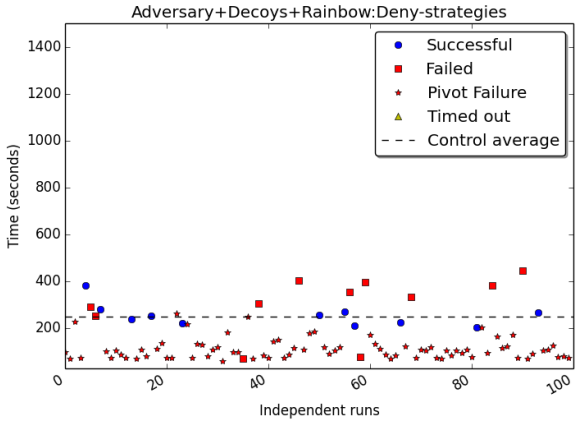

Figure 3. Deny results

In the deny condition, decoys had been pre-deployed as in the static decoys condition, but the adaptive deception system with deny as the user-defined goal is then activated. This uses the strategies in the delay condition, but with the additional tactic of sending TCP reset packets for connections made by the attacker to any machine. Here, we see very few successes, 11, for the attacker, a large amount of pivot failures, 78, and 11 exfiltration failures.

The significant number of pivot failures are due to the new tactic which can cause a TCP session to end. Because the attacker relies on a persistent connection for success, once they hit a decoy and generate an alert, the tactic may be chosen to disrupt any of the attacker's active TCP sessions over several minutes and disrupts their session. In a more realistic setting, an attacker would re-exploit the machine, but at this point, defenders may have already been alerted to begin incident response or threat hunting procedures, or we could have also started enabling additional defensive capabilities, such as modifying the network to block access. This type of TCP session interference tactic could be risky as it could feasibly be used by an informed attacker with the ability to

spoof connection attempts to perform a denial of service attack. In practice, we recommend using stronger indicators than single packet events or network scanning events. Triggering these more risky tactics based only on established TCP connections can better ensure the attacker isn't spoofing session initiations to disrupt legitimate users and services.

Figure 3 for the deny condition shows that the average runtime is lower, largely due to the TCP reset tactic interfering with the attacker's connection that stopped them earlier in the Cyber Kill Chain®. The average successful run was 256.64 seconds, average pivot failure run was 113.19 seconds, average exfiltration failure was 302.98 seconds, and the overall average runtime was 149.75 seconds.

5. Conclusions and Future Work

In this work, we compare the effects of adaptive defensive deception on an automated attacker. We have shown how an autonomic system can be used to perform management of a defensive deception system. Against an automated attacker we demonstrate defender advantage when using static decoys versus no cyber deception. When adding in user-defined goals that guide adaptive decoys, compared to the control scenario with no cyber deception or static defenses we observe an 175% increase in attacker runtime with delay goal strategies, and a 89% reduction in successful runs with deny goal strategies.

In this paper we described how high-level human-selected goals such as *delay* and *deny* can be used to direct an adaptive cyber defense system. These goals were selected as examples of techniques thought to be similarly effective against automated and human attackers. However, our work encompasses many additional defensive goals, some best attained by deception, such as misinforming the attacker, that can only be tested and evaluated through studies with human attackers. There is reason to believe this where the most powerful effects of adaptive cyber deception will become evident; future work will examine if this is the case. Furthermore, additional work is needed to study the effects of adaptive cyber deception and attacker knowledge. Our attacker model is very simplistic, and would benefit from a more sophisticated model which may include attacker knowledge, preference, goal, or level of frustration. We have also only scratched the surface in terms of modeling oppositional human factors effects by incurring usability delays and disruption. There are many more opportunities to leverage cognitive and behavioral psychology to interfere with an attacker for defensive gain. One obvious extension to this is to incorporate a richer library of goals and associated strategies, enabling a human defender to direct the system's behaviors and adaptations in more sophisticated ways. Unsurprisingly, we have noted in our testing that for some tactics, the effectiveness of the defense appears to depend on whether the attack is directly controlled by a human or is automated. Although such automated tools are likely to have been initiated by a human attacker, this difference presents an opportunity to use these tactics to determine whether the system is contending with an automated attack or a human attacker. Feedback of these observations could then be used to guide adaptations which are more optimized against human or automated attackers respectively.

Another area of interest is contending with multiple attackers (or in some cases, determining whether the currently observed attacker has been seen previously). Previous work in this area has focused on forms of passive attribution, attempting to *fingerprint* or otherwise uniquely identify particular attackers through observations made of their actions on a network or system. The combined use of defensive cyber deception and observation-driven adaptations provides new avenues for exploring methods for differentiating between attackers. Developing novel methods for determining the mere presence of multiple attackers may be a reasonable first step in this direction. Interestingly, research questions to address these issues overlap but also present unique perspectives which diverge from the literature on attack attribution. Can we detect when multiple attackers are present? Are the attackers aware of one another? Are the attackers colluding or sharing information? And this leads to new directions for oppositional human factors research. Are there actions we can take to disrupt teams of colluding attackers? What about if they do not know of the presence of the other, can we get them to disrupt each other?

References

[1] E. M. Hutchins, M. J. Cloppert, and R. M. Amin, "Intelligence-driven computer network defense informed by analysis of adversary campaigns and intrusion kill chains," *Leading Issues in Information Warfare & Security Research*, vol. 1, no. 1, p. 80, 2011.

[2] K. E. Heckman, F. J. Stech, R. K. Thomas, B. Schmoker, and A. W. Tsow, *Cyber Denial, Deception and Counter Deception*. Advances in Information Security, Springer International Publishing, 2015.

[3] S. Fugate and K. Ferguson-Walter, "Artificial intelligence and game theory models for defending critical networks with cyber deception," *AI Magazine*, vol. 40, pp. 49–62, Mar 2019.

[4] F. Moisan and C. Gonzalez, "Security under Uncertainty: Adaptive Attackers Are More Challenging to Human

Defenders than Random Attackers," *Frontiers in Psychology*, vol. 8, pp. 1029–10, June 2017.

[5] R. Ensafi, J. Knockel, G. Alexander, and J. r. Crandall, "Detecting intentional packet drops on the internet via tcp/ip side channels: extended version," 2013.

[6] K. Ferguson-Walter, D. LaFon, and T. Shade, "Friend or Faux: Deception for Cyber Defense," *Journal of Information Warfare*, vol. 16, no. 2, pp. 28–42, 2017.

[7] J. O. Kephart and D. M. Chess, "The vision of autonomic computing," *Computer*, vol. 36, no. 1, pp. 41–50, 2003.

[8] A. Computing, "Ibm's perspective on the state of information technology, 2001," *URL www. research. ibm. com/autonomic/manifesto/autonomic computing. pdf*.

[9] D. Garlan, S.-W. Cheng, A.-C. Huang, B. Schmerl, and P. Steenkiste, "Rainbow: Architecture-based self-adaptation with reusable infrastructure," *Computer*, vol. 37, no. 10, pp. 46–54, 2004.

[10] J. Cámara, P. Correia, R. de Lemos, D. Garlan, P. Gomes, B. Schmerl, and R. Ventura, "Incorporating architecture-based self-adaptation into an adaptive industrial software system," *Journal of Systems and Software*, vol. 122, pp. 507–523, 2016.

[11] P. Jamshidi, J. Cámara, B. Schmerl, C. Käestner, and D. Garlan, "Machine learning meets quantitative planning: Enabling self-adaptation in autonomous robots," in *2019 IEEE/ACM 14th International Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS)*, pp. 39–50, 2019.

[12] R. S. Gutzwiller, D. S. Lange, J. Reeder, R. L. Morris, and O. Rodas, "Human-computer collaboration in adaptive supervisory control and function allocation of autonomous system teams," in *International Conference on Virtual, Augmented and Mixed Reality*, pp. 447–456, Springer, 2015.

[13] I. Dzieciuch, J. Reeder, R. Gutzwiller, E. Gustafson, B. Coronado, L. Martinez, B. Croft, and D. S. Lange, "Amplifying human ability through autonomics and machine learning in impact," in *Micro-and Nanotechnology Sensors, Systems, and Applications IX*, vol. 10194, p. 101941Y, International Society for Optics and Photonics, 2017.

[14] E. Yuan, S. Malek, B. Schmerl, D. Garlan, and J. Gennari, "Architecture-based self-protecting software systems," in *Proceedings of the 9th international ACM Sigsoft conference on Quality of software architectures*, pp. 33–42, 2013.

[15] D. Garlan, R. Monroe, and D. Wile, "Acme: An architecture description interchange language," in *CASCON First Decade High Impact Papers*, pp. 159–173, 2010.

[16] S.-W. Cheng and D. Garlan, "Stitch: A language for architecture-based self-adaptation," *Journal of Systems and Software*, vol. 85, no. 12, pp. 2860–2875, 2012.

[17] G. A. Moreno, J. Cámara, D. Garlan, and B. Schmerl, "Proactive self-adaptation under uncertainty: a probabilistic model checking approach," in *Proceedings of the 2015 10th joint meeting on foundations of software engineering*, pp. 1–12, 2015.

[18] H.-M. Chou and L. Zhou, "A game theory approach to deception strategy in computer mediated communication," in *Intelligence and Security Informatics (ISI), 2012 IEEE International Conference on*, pp. 7–11, IEEE, June 2012.

[19] K. Ferguson-Walter, S. Fugate, J. Mauger, and M. Major, "Game theory for adaptive defensive cyber deception," in *Proceedings of the 6th Annual Symposium on Hot Topics in the Science of Security*, HotSoS, ACM, 2019.

[20] R. Gutzwiller, K. J. Ferguson-Walter, S. Fugate, and A. Rogers, "'Oh, Look, A butterfly!' A framework for distracting attackers to improve cyber defense," in *Human Factors and Ergonomics Society (HFES)*, (Philadelphia, Pennsylvania), Oct. 2018.

[21] R. S. Gutzwiller, K. J. Ferguson-Walter, and S. J. Fugate, "Are cyber attackers thinking fast and slow? Evidence for cognitive biases in red teamers reveals a method for disruption," *Proceedings of the Human Factors and Ergonomics Society (HFES) Annual Meeting*, 2019.

[22] C. Johnson, R. Gutzwiller, K. Ferguson-Walter, and S. Fugate, "A cyber-relevant table of decision making biases and their definitions," ResearchGate, 2020.

[23] G. Killcrece, K.-P. Kossakowski, R. M. Ruefle, and M. T. Zajicek, "State of the practice of computer security incident response teams (csirts)," 2003.

[24] L. Onnasch, C. D. Wickens, H. Li, and D. Manzey, "Human performance consequences of stages and levels of automation: An integrated meta-analysis," *Human Factors*, vol. 56, no. 3, pp. 476–488, 2014.

[25] R. Parasuraman, T. B. Sheridan, and C. D. Wickens, "A model for types and levels of human interaction with automation," *IEEE Transactions on systems, man, and cybernetics-Part A: Systems and Humans*, vol. 30, no. 3, pp. 286–297, 2000.

[26] W. B. Rouse, "Adaptive aiding for human/computer control," *Human Factors*, vol. 30, no. 4, pp. 431–443, 1988.

[27] R. Parasuraman and D. H. Manzey, "Complacency and bias in human use of automation: An attentional integration," *Human factors*, vol. 52, no. 3, pp. 381–410, 2010.

[28] B. M. Muir, "Trust in automation: Part i. theoretical issues in the study of trust and human intervention in automated systems," *Ergonomics*, vol. 37, no. 11, pp. 1905–1922, 1994.

[29] B. E. Strom, A. Applebaum, D. P. Miller, K. C. Nickels, A. G. Pennington, and C. B. Thomas, "Mitre att&ck: Design and philosophy," *Technical report*, 2018.

[30] F. Holik, J. Horalek, O. Marik, S. Neradova, and S. Zitta, "Effective penetration testing with metasploit framework and methodologies," in *2014 IEEE 15th International Symposium on Computational Intelligence and Informatics (CINTI)*, pp. 237–242, Nov 2014.

[31] Rapid7, "Microsoft Windows Authenticated User Code Execution." Available online at `https://www.rapid7.com/db/modules/exploit/windows/smb/psexec`. Accessed January 20, 2020., viewed October 2019.

[32] K. J. Ferguson-Walter and S. J. Fugate, "Adaptive cyber deception," US Provisional Patent Application Serial No. 62/874,805 filed July 16, 2019.

[33] X. Feng, Z. Zheng, P. Mohapatra, and D. Cansever, "A stackelberg game and markov modeling of moving target defense," in *International Conference on Decision and Game Theory for Security*, pp. 315–335, Springer, 2017.