Using Bayesian Attack Detection Models to Drive Cyber Deception

James H. Jones, Jr.
Department of Electrical and Computer Engineering
George Mason University
Fairfax, VA 22030

Kathryn B. Laskey
Department of Systems Engineering and Operations Research
George Mason University
Fairfax, VA 22030

Abstract

We present a method to devise, execute, and assess a cyber deception. The aim is to cause an adversary to believe they are under a cyber attack when in fact they are not. Cyber network defense relies on human and computational systems that can reason over multiple individual evidentiary items to detect the presence of meta events, i.e., cyber attacks. Many of these systems aggregate and reason over alerts from Network-based Intrusion Detection Systems (NIDS). Such systems use byte patterns as attack signatures to analyze network traffic and generate corresponding alerts. Current aggregation and reasoning tools use a variety of techniques to model meta-events, among them Bayesian Networks. However, the inputs to these models are based on network traffic which is inherently subject to manipulation. In this work, we demonstrate a capability to remotely and artificially trigger specific meta events in a potentially unknown model. We use an existing and known Bayesian Network based cyber attack detection system to guide construction of deceptive network packets. These network packets are not actual attacks or exploits, but rather contain selected features of attack traffic embedded in benign content. We provide these packets to a different cyber attack detection system to gauge their generalizability and effect. We combine the deception packets' characteristics, the second system's response, and external observables to propose a *deception model* to assess the effectiveness of the manufactured network traffic on our target. We demonstrate the development and execution of a specific deception, and we propose the corresponding deception model.

Key words: Cyber Deception, Cyber Attack, Bayesian Model, Deception Model, Intrusion Detection System

1. INTRODUCTION

Network-based Intrusion Detection Systems (NIDS) are essentially granular sensors. Their measurements consist of computer network traffic, sometimes at the packet level, which matches signatures of known cyber attack activity. For a typical network, the individual data points are numerous and require aggregation, fusion, and context to acquire meaning. This reasoning may be accomplished through the use of cyber attack detection models, where the NIDS data points represent evidence and specific cyber attacks or classes of attacks represent hypotheses. Modeling approaches, including Bayesian Networks, have been applied in the past, are an active research area, and are in use today in deployed systems.

The input to a NIDS sensor is network traffic, which is inherently uncertain and subject to manipulation. Prior research has exploited this fact to create large numbers of false NIDS alerts to overwhelm or disable the backend processing systems. In this work, we leverage knowledge of the backend cyber attack models to craft network traffic which manipulates the inputs and hence the outputs of both known and unknown models. With a small number of packets and no actual cyber attack, we are able to create the false impression of an active attack.

In this work, we describe a general approach to network-based offensive cyber deception, and we demonstrate an implementation of such a deception. We use an existing cyber attack detection model to guide the development of deception traffic, which is then processed by a second and distinct cyber attack detection model. Finally, we propose a deception model to assess the effectiveness of the deception on a target. Future work will expand and automate the generation of deceptive network packets and further develop the deception model.

2. RELATED WORK

Deception has been a staple of military doctrine for thousands of years, and a key element of intelligence agency activities since they took their modern form in World War II. From Sun Tzu 2,500 years ago (Tzu, 2013), to *Operation Mincemeat* in 1943 (Montagu and Joyce, 1954), to the fictional operation in the 2007 book *Body of Lies* (Ignatius, 2007), one side has endeavored to mislead the other through a variety of means and for a variety of purposes. The seminal work of Whaley and Bell (Whaley, 1982; Bell and Whaley, 1982; Bell and Whaley, 1991), formalized and in some ways defended deception as both necessary and possible to execute without "self contamination".

Deception operations have naturally begun to include the cyber domain, although the majority of this work has been on the defensive side. Fred Cohen suggested a role for deception in computer system defense in 1998 (Cohen, 1998) and simultaneously released his honeypot implementation called The Deception Toolkit (Cohen, 1998). Honeypots are systems meant to draw in attackers so they may be distracted and/or studied. The Deception Toolkit was one of the first configurable and dynamic honeypots, as opposed to prior honeypots which were simply static vulnerable systems with additional administrator control and visibility. Other honeypot implementations have followed, and they remain a staple of defensive cyber deception. Neil Rowe (2003)(2007), his colleague Dorothy Denning (Yuill, Denning, and Feer, 2006), and students (Tan, 2003) at the Naval Postgraduate School have been researching defensive cyber deception for several years. Extending their early work identifying key disruption points of an attack, they propose deception by resource denial, where some key element of an active attack vector is deceptively claimed to be unavailable. Such an approach stalls the attacker while the activity can be analyzed and risks mitigated. Other defensive cyber deception approaches include masking a target's operating system (Murphy, McDonald, and Mills, 2010), and actively moving targets within an IP address and TCP port space (Kewley, et al., 2001), later labeled "address shuffling". Crouse's (2012) comparison of the theoretical performance of honeypots and address shuffling remains one of the few rigorous comparisons of techniques. Until recently, most defensive cyber deception involved theoretical work or small proofs of concept. However, in 2011, Ragsdale both legitimized defensive cyber deception and raised the bar when he introduced DARPA's Scalable Cyber Deception program (Ragsdale, 2011). The program aims to automatically redirect potential intruders to tailorable decoy products and infrastructures in real time and at an enterprise scale.

By comparison, offensive cyber deception has been discussed only briefly in the literature, often as a secondary consideration. For example, honeypots are typically a defensive tool but may be used in an offensive sense to provide disinformation to an adversary. Similarly, deliberately triggering an adversary's network defenses to overwhelm or disable equipment, software, or operators was discussed openly in 2001 (Patton, Yurcik, and Doss 2001) but proposed as cover for other attacks rather than to effect a deception. A small number of offensive cyber deception implementations have been presented, such as the D^3 (Decoy Document Distributor) system to lure malicious insiders (Bowen, Hershkop, Keromytis, and Stolfo, 2009) and the ADD (Attention Deficit Disorder) tool to create artificial host-based artifacts in memory to support a deception (Williams and Torres, 2014). While offensive cyber warfare has entered the public awareness with the exposure of activity based on tools such as Stuxnet, Flame, and Shamoon, offensive cyber deception remains the subject of limited open research and discussion.

Our deception work focuses on aggregation and reasoning tools applied to Network Intrusion Detection Systems (NIDS). These reasoning tools emerged from the inundation of alerts when NIDS sensors were first deployed on enterprise networks. Such tools may simply correlate and aggregate alerts or may model cyber attack and attacker behavior to reason over large quantities of individual evidentiary items and provide assessments of attack presence for human operators to review. Such reasoning models are abundant in the literature and in operational environments, having become indispensable to cyber defenders and remaining an active research area. Initial work on correlating and aggregating NIDS alerts appeared in 2001 (Valdes and Skinner, 2001). A few years later, a body of research emerged which correlated NIDS events with vulnerability scans to remove irrelevant alerts, for example (Zhai, et al., 2004). More advanced reasoning models emerged a few years later, attempting to capture attack and attacker behavior using various techniques. For example, Zomlot, Sundaramurthy, Luo, Ou, and Rajagopalan (2011) applied Dempster-Shafer theory to prioritize alerts, and Bayesian approaches remain popular (Tylman, 2009; Hussein, Ali, and Kasiran, 2012; Ismail, Mohd and Marsono, 2014). Jones and Beisel (2014) developed a Bayesian approach to reasoning over custom NIDS alerts for novel attack detection. A prototype of this approach, dubbed Storm, was used as the base model in this work.

Our research builds on this rich body of prior work, merging the basic precepts of deception, manipulation of network traffic, and model-based reasoning into an offensive cyber deception capability. We use existing *detection* models to derive corresponding *deception* models, demonstrating the ability to deceive an adversary on their own turf and causing them to believe they are under attack when in fact they are not. This capability may be used offensively to create an asymmetry between attackers generating small numbers of deception packets and targets investigating multiple false leads, and

defensively to improve the sensitivity, specificity, and deception recognition of existing cyber attack detection tools.

3. BACKGROUND

Signature-based Network Intrusion Detection Systems operate by matching network traffic to a library of patterns derived from past attacks and known techniques. Matches generate alerts, often one per packet, which are saved and sent to a human operator for review. The basic idea of intrusion detection is attributed to Anderson (1980). Todd Heberlein introduced the idea of network-based intrusion detection in 1990 (Heberlein, et al., 1990). Only when processing capabilities caught up to network bandwidth did the market take off in the late 1990s and early 2000s. Unfortunately, early enterprise deployments generated massive numbers of alerts, to the point that human operators could not possibly process them all. Two capabilities came out of this challenge: (1) correlation between NIDS data and other enterprise sources, such as vulnerability scanning data, e.g., see ArcSight[i], and (2) aggregators which model cyber attacks and use NIDS alerts as individual evidentiary items, only alerting a human when multiple aspects of an attack are detected, e.g., see Hofmann and Sick (2011). As noted in Section 2, many modeling approaches have been applied to the aggregation and context problem for more than a decade, and the area remains one of active research, e.g., Boukhtouta, et al (2013).

For the base model in this project, we used an existing cyber attack detection model previously developed by one of the authors. The implementation of this model, called Storm, uses network traffic observables and a Bayesian Network reasoning model to detect a system compromise resulting from known and novel cyber attacks. The system ingests raw network traffic via a live network connection or via traffic capture files in libpcap[ii] format. Individual packets and groups of packets are assessed against signatures associated with cyber attack stages, such as reconnaissance, exploitation, and backdoor access (see Figure 1). Prior to ingest by the model, saturation and time decay functions are applied to packets which match signatures so that model output reflects the quantity and timing of packets. Ingest and model updates occur in real time as packets are received and processed. Packet capture and signature matching is implemented in C++ using libpcap on a Linux (Ubuntu) system. Matching packet processing, model management, and the user interface are implemented in Java, and the Bayesian Network is implemented with Unbbayes[iii]. Packets are passed to the model via a TCP socket so that packet processing and reasoning may be performed on different systems, although we used a single server for our testing.

The Storm system reasons over indirect observables resulting from the necessary and essentially unavoidable steps necessary to effect a system compromise. This underlying cyber attack process is shown in Figure 1 below, where a typical attack progress downward from State 1 (S1) to State 7 (S7). Observables are created at each state and transition. The existing Storm implementation contains one or more observable signatures for each of the six state transitions (T1-T6 in the figure).

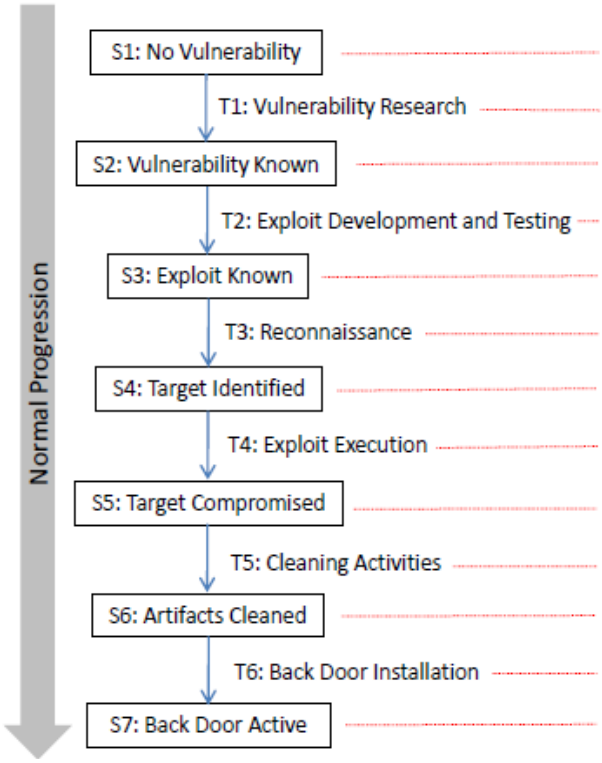

Figure 1: Cyber Attack Model

The reasoning model, shown in Figure 2, was derived from expert knowledge and consists of 20 signature evidence nodes (leaves labeled Tnn), three derived evidence nodes (labeled Mn), two protocol aggregation nodes (labeled Port80 and Port25), six transition aggregation nodes (labeled Tn), and a root node (labeled Compromise). Signature hits are processed and used to set values for the Tnn and Mn nodes. As implemented, one model is instantiated for each cyber attack target (unique target IP address). Model instances are updated whenever new evidence is received or a preconfigured amount of time has passed, and the root node values are returned as Probability of Compromise given Evidence, P(C|E), for each target.

The theory behind the model, further explained by Jones and Beisel (2014), is to recognize observables created when a cyber attack transitions to a new state. For example, when transitioning to the exploit stage, packets with NOP instructions (machine code for "do nothing"

and used in buffer overflow type attacks) or shell code elements (part of many exploit payloads) are often seen. As such, the Storm signature rules for detecting observables are not specific to particular attacks, but rather represent effects common to actions associated with cyber attack stages in general. Taken individually, signature matches do not imply an attack. However, when aggregated and combined in context by the reasoning model, an accurate assessment of attack existence may be produced. The system is able to detect novel attacks, since signatures are based on generic cyber attack state transitions instead of specific attack signatures.

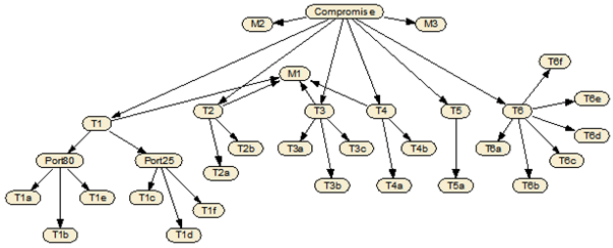

Figure 2: Bayes Net Cyber Attack Detection Model

For most environments, network traffic is insecure and untrusted. Most networks and systems carry and accept traffic that may be both unencrypted and unauthenticated. As such, nearly anyone can introduce traffic on an arbitrary network or at least modify traffic destined for an arbitrary network or system. While full arbitrary packet creation, modification, and introduction is not generally possible, the ability to at least minimally manipulate packets destined for an arbitrary network or system is inherent in the design and implementation of the Internet. Packet manipulation is straightforward using available tools like Scapy[iv], requiring only knowledge of basic object oriented concepts and an understanding of the relevant network protocols.

It is the combination of an ability to manipulate network traffic and models which use network traffic as evidentiary inputs which we exploit in our work.

4. METHODOLOGY

Our goal for this project is to establish the viability of manipulating an adversary's perception that they are the target of a cyber attack when in fact they are not. We begin by conducting a sensitivity analysis of a known cyber attack detection model to identify candidate influence points. We design, construct, and inject network packets to trigger evidence at a subset of these influence points. We construct a corresponding deception model to assess the likelihood that our deception is effective. This derivative deception model combines the impact of our deception packets with other factors, such as the ease with which a target may invalidate the deception packets, the prevalence of alternative explanations for detection system alarms, and external indicators of the target's response activities. The impact of our deception packets is estimated by their effect on an alternative cyber attack detection system, in this case Snort[v]. See Figure 3 for an overview of this process.

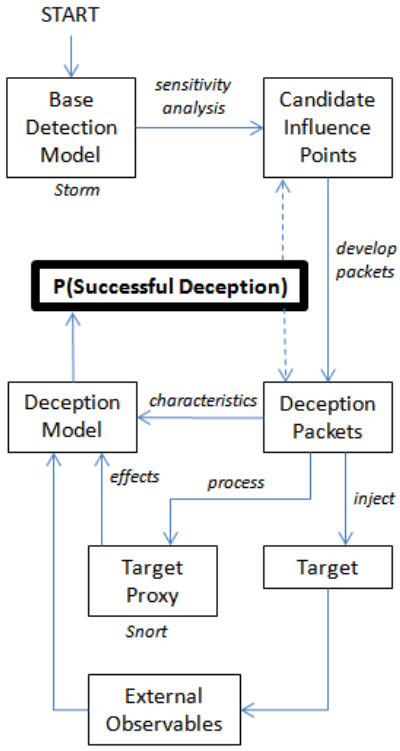

Figure 3: Process Overview

The deception model output, P(Successful Deception) is envisioned to be a dynamic value computed in real time as deception packets are delivered to a target and external observables are collected. As the deception operation unfolds and feedback from external observables is incorporated, additional existing deception packets may be injected, or influence points may be examined for additional deception packet development.

Our base detection model is a Bayesian Network (Figure 2) from a test implementation of the Storm cyber attack detection system. A single node sensitivity to findings analysis (from Netica) is summarized in Table 1 for the 20 evidence input nodes. We reviewed this output and the descriptions of each signature to select those which (a) would have high impact on Storm's probability of compromise based on the sensitivity analysis, (b) could be reasonably developed into a deception packet or packets, and (c) could be general enough to be detected by a target's cyber attack detection system, i.e., not Storm. In Table 1, the eight non-gray rows are those that were selected for deception packet development (signatures T5a, T6a, T6b, T6d, T4a, T4b, T1e, and T1f).

Table 1: Storm model sensitivity analysis

Signature	Mutual Info	Variance of Beliefs
T5a	0.03305	0.0011763
T6e	0.02719	0.0008344
T6a	0.02719	0.0008344
T6b	0.02719	0.0008344
T6d	0.02719	0.0008344
T6c	0.02719	0.0008344
T6f	0.02719	0.0008344
T4a	0.01701	0.0004315
T4b	0.01701	0.0004315
T3b	0.01658	0.0003618
T3c	0.00702	0.0001202
T3a	0.00259	0.0000372
T1a	0.00002	0.0000002
T1b	0.00002	0.0000002
T1e	0.00002	0.0000002
T1c	0.00002	0.0000002
T1d	0.00002	0.0000002
T1f	0.00002	0.0000002
T2a	0.00001	0.0000002
T2b	0.00001	0.0000002

Our test environment consisted of a Storm implementation running on Ubuntu 11.10 and a packet manipulation host running BackTrack5[vi]. We captured normal network traffic in a test environment and processed the traffic through the Storm system to confirm that no attacks were detected. Storm, like most NIDS implementations, has the ability to ingest live network traffic as well as network traffic capture files without loss of accuracy or fidelity. Traffic was captured using the open source Wireshark[vii] tool, saved as a pcap file, then ingested by Storm. We labeled this original packet capture file "clean" and used it as the basis for our subsequent packet manipulations.

We used Scapy on the BackTrack5 instance to craft deception packets. Scapy is an open source Python based packet crafting and editing tool. Packets may be loaded from a pcap file, then manipulated in an environment similar to a Python command shell and written out to a pcap file. Scapy supports creation and modification of any packet field down to the byte level and to include the raw creation and editing of packet data. To create our deception packets, we made minor modifications to packets from the clean set. By minimizing changes, we produced packets that maintained most of the clean session characteristics and so would not be blocked by a firewall or other packet screening device. Also, packets were modified only to the extent necessary to trigger the desired signature, so the modified packets do not contain any actual attacks. The modified packets and associated unmodified session packets, such as session establishment via the TCP 3-way handshake, were exported to a separate pcap file so they could be ingested by the Storm and Snort systems in a controlled manner.

The eight signatures selected for deception and the related deception packets are summarized in Table 2.

Table 2: Signatures and deception packets

Signature T1e
Description: After 3-way handshake, DstPort=80, payload≠<ASCII>
Explanation: Abnormal traffic to web server (usually expect GET or POST with ASCII data)
Deception packet: Inserted non-ASCII (hex > 7F) at beginning of payload for existing HTTP session
Signature T1f
Description: After 3-way handshake, DstPort=25, payload≠<ASCII>
Explanation: Abnormal traffic to a mail server (normally we expect plaintext commands)
Deception packet: Edited HTTP session to use server port 25; inserted non-ASCII (hex > 7F) at beginning of payload
Signature T4a
Description: Client to server traffic containing 20+ repeated ASCII characters
Explanation: Buffer overflows often use a long string of ASCII characters to overflow the input buffer
Deception packet: Inserted 43 "d" (hex 64) characters at the beginning of existing HTTP session payload
Signature T4b
Description: Client payload contains 20+ identical and consecutive NOP instruction byte patterns
Explanation: A "NOP sled" is a common technique used in buffer overflow exploits; the sled consists of multiple NOP (No Operation) instructions to ensure that the real instructions fall in the desired range
Deception packet: Inserted 24 hex 90 (known NOP code) characters at the beginning of existing HTTP session payload

Signature T5a
Description: Client to server traffic if port≠23 and first 100 bytes of payload contains "rm", "rmdir", "rd", "del", "erase"
Explanation: File or directory removal activity
Deception packet: Inserted "rm " (hex 726D20) characters at the beginning of existing HTTP session payload
Signature T6a
Description: First two bytes of client to server payload="MZ"
Explanation: COM, DLL, DRV, EXE, PIF, QTS, QTX, or SYS file transfer for use in a backdoor
Deception packet: Inserted "MZ" (hex 4D5A) and filename "exe" characters at the beginning of existing HTTP session payload
Signature T6b
Description: New Port opened on server; ignore first 500 packets after startup
Explanation: Traffic from a port not previously seen might indicate the opening of a new back door
Deception packet: Edited HTTP session to use server port 25 (ingested after first 500 packets)
Signature T6d
Description: Unencrypted traffic on encrypted port
Explanation: Traffic on encrypted sockets (HTTPS, SMTP with SSL, Secure Shell, etc.) should be encrypted once the session is established.
Deception packet: Inserted ASCII text in an established SSH session

Manual creation of the deception packets required a moderate amount of effort. When crafting deception packets, care must be taken to use carrier traffic which will be passed by a firewall or similar network security gateway while still triggering the desired signature. Flexibility in carrier traffic is signature dependent. For example, some signatures have offset or port dependencies like requiring the traffic to be, or not to be, on port 80 (HTTP), while others are more flexible. Future work will develop an automated deception packet creation capability.

We began by identifying a candidate session for packet modification. For example, we could start with an existing HTTP or HTTPS session and alter packet payload, or we might also alter TCP ports. Payload modification required adjustments to the TCP checksum, IP checksum, and IP length values as well. To create a deception packet set, we loaded the clean pcap file in scapy, made the desired packet modifications, and wrote the resulting packet set to a new pcap file. We then used Wireshark to confirm our modifications and to extract and save only the session of interest as a distinct pcap file. We confirmed our deception packets by processing them with Storm, and later Snort, in a controlled environment.

We tested single occurrences of each signature separately and in a subset of possible combinations. For each individual signature and for selected combinations, we also tested the effects of 10 and 20 signature instances. Finally, for selected signatures, we measured the effect of multiple occurrences for values 1, ..., 25. For all tests, we reset the Storm model, loaded the desired pcap file, and recorded the resulting P(C|E).

We then processed each of the deception pcap files (one per signature) with Snort, separately and in combinations and repetitions.

5. EXPERIMENTAL RESULTS

Each signature pcap file was processed by Storm in quantities of 1, 10, and 20 hits. Storm was reset after each run, that is, reset after a run of one T1e hit, then reset after a run of 10 T1e hits, then reset after a run of 20 T1e hits, etc. Results are recorded in Table 3.

Table 3: Single signature impact on P(C|E) with repetition

Signature		Qty=1	Qty=10	Qty=20
ID	Short Description	P(C|E)	P(C|E)	P(C|E)
T1e	HTTPload!=ASCII	0.00	0.01	0.03
T1f	SMTPload!=ASCII	0.00	0.01	0.03
T4a	Repeated ASCII	0.01	0.05	0.16
T4b	Repeated NOPs	0.01	0.05	0.15
T5a	Cleanup cmds	0.02	0.09	0.27
T6a	Executable load	0.02	0.08	0.22
T6b	New server port	0.02	0.08	0.22
T6d	Unencrypted SSL	0.02	0.08	0.22

The relationships between the quantity of signature hits and P(C|E) in each row indicate that setting the Bayesian Network findings is not a simple True/False assignment. To confirm this behavior, we recorded the effect on P(C|E) of 1, 2, ..., 25 signature hits for signatures T5a and T6a. These results are graphed in Figures 4a and 4b. The curves and apparent inflection points of the graphs indicate that the signature hits are subject to a ramping up requirement at low quantities and a saturation adjustment at high quantities. This is in fact implemented by a pre-processing step in the Storm system and is not actually a part of the Bayesian Network component of Storm.

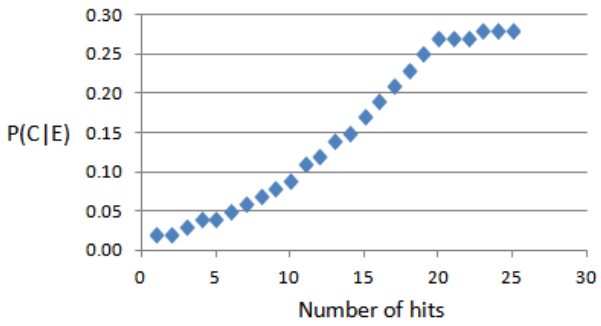

Figure 4a: T5a signature hit effect for n = 1..25

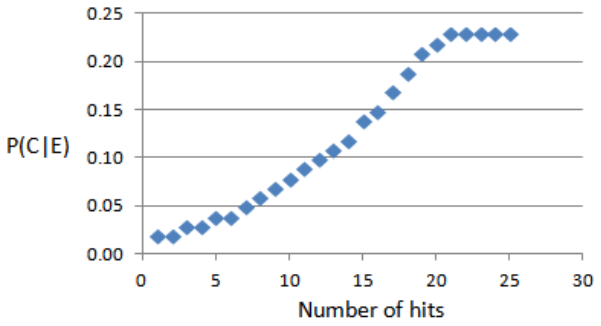

Figure 4b: T6a signature hit effect for n = 1..25

We constructed 11 combinations of signature hits at selected quantities and tested each combination. The results are shown in Tables 4a and 4b below (split for readability).

The results indicate that the desired effect was achieved in two ways: (1) single hits across a wide range of signatures (e.g., tests D, E, and F), and (2) repeated hits on selected signatures (e.g., tests H and K). Assuming a threshold of P(C|E) > 0.75 for alerting, meta-event alerts could be generated with as few as five packets spread across five signatures as in test F, or 40 packets spread across only two different signatures as in test K.

We processed the same eight pcap deception files with Snort. Six of the eight files triggered Snort alerts, as summarized below in Table 5.

Snort Priority 1 are the most severe, Priority 3 the least. The name, classification, and priority of each signature are assigned by the signature author. A base Snort install contains signatures contributed by the Snort developers and the open source community.

Our deception packets produced five Priority 1 alerts, one Priority 2 alert, and one Priority 3 alert. Two deception packets (pcap files for T5a and T6b) did not trigger any Snort alerts.

Table 4a: Effect of signature combinations on P(C|E)

Signature	**Test**					
ID	**A**	**B**	**C**	**D**	**E**	**F**
T1c					1	
T1e					1	
T4a		1	1	1	1	1
T4b		1	1	1	1	
T5a				1	1	1
T6a	1		1	1	1	1
T6b	1		1	1	1	1
T6d	1		1	1	1	1
P(C|E)	**0.10**	**0.02**	**0.46**	**0.86**	**0.89**	**0.79**

Table 4b: Effect of signature combinations on P(C|E)

Signature	**Test**				
ID	**G**	**H**	**I**	**J**	**K**
T1c					
T1e					
T4a	1	10	10		
T4b					
T5a	1	10	10	10	20
T6a	1	10		10	20
T6b					
T6d					
P(C|E)	**0.37**	**0.93**	**0.58**	**0.72**	**0.78**

In a default configuration, and without any subsequent aggregation or alert thresholds, Snort alerts are explicitly linear, meaning that combination and repetition testing produced the obvious results. For example, running any one pcap file N times produces N alerts. Similarly, running the files in combination produced the expected total of alerts, e.g., running the entire set 10 times produced 70 Snort alerts.

Table 5: Snort alerts from deception packets

File	Snort Alerts
T5a	None
T6a	Lotus Notes .exe script source download attempt [Classification: Web Application Attack] [Priority: 1]
T6b	None
T6d	Protocol mismatch [Priority: 3]
	EXPLOIT ssh CRC32 overflow /bin/sh [Classification: Executable code was detected] [Priority: 1]
T4a	MailSecurity Management Host Overflow Attempt [Classification: Attempted Admin Privilege Gain] [Priority: 1]
T4b	SHELLCODE x86 NOOP [Classification: Executable code was detected] [Priority: 1]
T1e	apache chunked enc mem corrupt exploit attempt [Classification: access to potentially vuln web app] [Priority: 2]
T1f	x86 windows MailMax overflow [Classification: Attempted Admin Privilege Gain] [Priority: 1]

6. DECEPTION MODEL

A sample deception model is shown in Figure 5. The model consists of three key parts: (A) the deception packets, (B) external observables indicating a successful deception, and (C) external observables indicating an unsuccessful deception.

Each deception packet, area A in the figure, is assessed for alternative explanations. For example, a byte string that we embed in a JPEG image file may generate a NIDS alert for an unrelated attack, but upon examination will be discounted as a chance occurrence and hence a false alarm. Strong alternative explanations suggest that the target might not interpret the packet as part of an attack and so would weaken the packet node's intended effect on the Successful Deception node. Similarly, each deception packet is assessed for how difficult it will be for a target to invalidate the packet. Again using the example of a byte string embedded in a JPEG image file, if the triggered NIDS alert is an exploit of image viewers, then the packet will be difficult to invalidate. A difficult-to-invalidate packet will have a strong positive influence on the Successful Deception node via the intended effect node.

Processing the original eight deception packets (pcap files) with Snort provides additional parameters for the model. The number, priority, and relevance of Snort alerts are used to build the Conditional Probability Table of the Deception Success node.

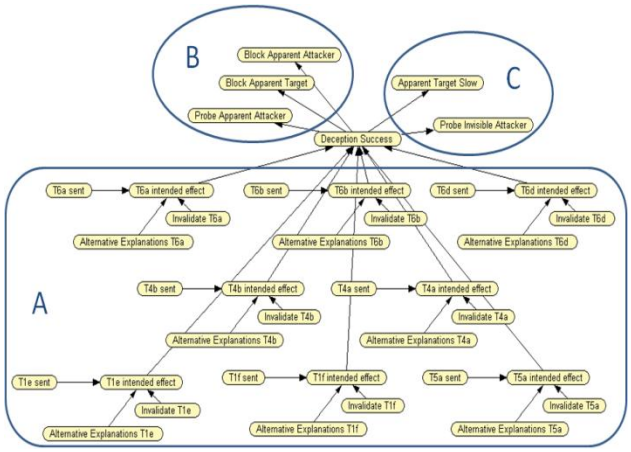

Figure 5: Bayes Net Cyber Deception Model

Area B in the graphic contains three nodes representing external observables which could indicate a successful deception. The "apparent target" and "apparent attacker" are the endpoints of the deception packets. As noted elsewhere, these systems may not send or receive any of the observed traffic, but they will be endpoints from a network monitor's point of view. If a target blocks the apparent target or attacker, or takes the apparent target off-line, then the deception is likely working. Similarly, if the target system operators probe the apparent attacker, then the deception is likely working.

Area C in the graphic contains two nodes for external observables which may indicate that the deception is not working. If the apparent target's response or processing time slows down, this may indicate that the target has added monitoring capabilities in order to trace the source of the deception, although this could also indicate monitoring in response to a perceived successful deception. The other node in area C is the worst case scenario. Although none of the deception packet contents are directly traceable to the actual perpetrators of the deception, probing of the perpetrators systems, especially from the target of the deception, might indicate that the deception has failed and the target suspects the true source of the deception.

The model of Figure 5 is a work in progress at the time of this writing. Preliminary values for the conditional probability tables have been developed but not yet tested or refined. Our work suggests that such a deception model may be developed for other domains where we have some control over the inputs to the base model. Our process in Figure 3 may be generalized by replacing "packets" with "evidence", since packets are simply our mechanism for affecting an evidentiary input node. Generally, a derived

deception model consists of the nodes that we directly influence, their estimated effect on the target's model, and external observables.

7. CONCLUSIONS AND FUTURE WORK

We demonstrated the ability to construct network packets which will look similar to normal network traffic, pass through a typical Firewall, trigger specific attack element signatures, and have a controlled impact on a back-end cyber attack detection reasoning model. Further, we proposed a derived deception model to dynamically assess the effectiveness of the cyber deception activities, and we suggested how such a deception model might be constructed for other domains. In support of cyber defense, our work also supports the testing and development of more accurate reasoning models and research geared towards detecting deception.

An apparent limitation of our work is a requirement to know the signatures which trip alerts and are fed to the back end reasoning model. However, this is not necessarily true. While these signatures may be known, as in the case of systems leveraging open source tools like Snort, it is also true that a system designed to detect specific attacks or attacks of a certain class will use similar signatures. The common requirement to derive a discriminatory signature that is as short as possible results in different entities independently producing similar signatures. We have observed this effect in the NIDS domain, where commercial and open source tools have similar signature sets for many attacks. Similarly, we observe this effect in the antivirus and malware detection industry, where different vendors and open source providers frequently generate similar signatures independently. The implication is that we could develop probable signatures for specific attacks or behaviors, then develop deception packets to trip these signatures with a reasonable expectation of successfully affecting a target system using unknown signatures. We partially demonstrated this by processing our Storm-derived packets with Snort.

As noted above, we assert that the use of pcap files is equivalent for our purposes to live network traffic capture and processing. However, it is true that in most live network scenarios we will not be able to put both sides of a TCP session on the wire as we did in this work. Rather, we will have to establish a live session with a target computer and modify subsequent session packets in real time, or we will have to intercept and modify packets between a target and some other system. This is an implementation issue vs. a question of validity, as the results presented here hold regardless of how the deceptive packets are introduced.

Future work will focus on automated deception packet creation, development of delivery mechanisms, and the derived deception model. We created our packets manually based on a review of the target signature and several iterations of trial and error. Our next step is to create deception packets directly from signature descriptions. For example, given a Snort signature file, we could craft multiple deception packets in an automated fashion. A related effort will explore the automation of delivery mechanisms, for example establishing TCP sessions with an internal host and delivering deception packets and injection of deception material into an existing network traffic stream. Author Jones recently led a project to develop a hardware-based inline packet rewriting tool which could be used for such a purpose. Finally, we will continue the development and generalization of deriving deception models from detection models.

References

Anderson, J. P. (1980). Computer security threat monitoring and surveillance (Vol. 17). Technical report, James P. Anderson Company, Fort Washington, Pennsylvania.

Bell, J. B., & Whaley, B. (1982). Cheating: deception in war & magic, games & sports, sex & religion, business & con games, politics & espionage, art & science. St Martin's Press.

Bell, J. B., & Whaley, B. (1991). Cheating and deception. Transaction Publishers.

Boukhtouta, A., Lakhdari, N. E., Mokhov, S. A., & Debbabi, M. (2013). Towards fingerprinting malicious traffic. Procedia Computer Science, 19, 548-555.

Bowen, B. M., Hershkop, S., Keromytis, A. D., & Stolfo, S. J. (2009). Baiting inside attackers using decoy documents (pp. 51-70). Springer Berlin Heidelberg.

Cohen, F. (1998). A note on the role of deception in information protection. Computers & Security, 17(6), 483-506.

Cohen, F. (1998). The deception toolkit. Risks Digest, 19.

Crouse, M. B. (2012). Performance Analysis of Cyber Deception Using Probabilistic Models (Master's Thesis, Wake Forest University).

Heberlein, L. T., Dias, G. V., Levitt, K. N., Mukherjee, B., Wood, J., & Wolber, D. (1990, May). A network security monitor. In Research in Security and Privacy, 1990. Proceedings., 1990 IEEE Computer Society Symposium on (pp. 296-304). IEEE.

Hofmann, A., & Sick, B. (2011). Online intrusion alert aggregation with generative data stream modeling. Dependable and Secure Computing, IEEE Transactions on, 8(2), 282-294.

Hussein, S. M., Ali, F. H. M., & Kasiran, Z. (2012, May). Evaluation effectiveness of hybrid IDs using snort with naive Bayes to detect attacks. In Digital Information and Communication Technology and it's Applications (DICTAP), 2012 Second International Conference on (pp. 256-260). IEEE.

Ignatius, D. (2007). Body of Lies. WW Norton & Company.

Ismail, I., Mohd Nor, S., & Marsono, M. N. (2014). Stateless Malware Packet Detection by Incorporating Naive Bayes with Known Malware Signatures. Applied Computational Intelligence and Soft Computing, 2014.

Jones, J. and Beisel, C. (2014) Extraction and Reasoning over Network Data to Detect Novel Cyber Attacks. National Cybersecurity Institute Journal. Volume 1, Number 1.

Kewley, D., Fink, R., Lowry, J., & Dean, M. (2001). Dynamic approaches to thwart adversary intelligence gathering. In DARPA Information Survivability Conference & Exposition II, 2001. DISCEX'01. Proceedings (Vol. 1, pp. 176-185). IEEE.

Montagu, E., & Joyce, P. (1954). The man who never was. Lippincott.

Murphy, S. B., McDonald, J. T., & Mills, R. F. (2010). An Application of Deception in Cyberspace: Operating System Obfuscation1. In Proceedings of the 5th International Conference on Information Warfare and Security (ICIW 2010) (pp. 241-249).

Patton, S., Yurcik, W., & Doss, D. (2001). An Achilles' heel in signature-based IDS: Squealing false positives in SNORT. Proceedings of RAID 2001.

Ragsdale, D. (2011). Scalable Cyber Deception. Defense Advanced Research Projects Agency, Arlington, Virginia, Information Innovation Office.

Rowe, N. C. (2003, June). Counterplanning deceptions to foil cyber-attack plans. In Information Assurance Workshop, 2003. IEEE Systems, Man and Cybernetics Society (pp. 203-210). IEEE.

Rowe, N. (2007, March). Planning cost-effective deceptive resource denial in defense to cyber-attacks. In Proceedings of the 2nd International Conference on Information Warfare & Security (p. 177). Academic Conferences Limited.

Tan, K. L. G. (2003). Confronting cyberterrorism with cyber deception (Doctoral dissertation, Monterey, California. Naval Postgraduate School).

Tylman, W. (2009) Detecting Computer Intrusions with Bayesian Networks. Intelligent Data Engineering and Automated Learning - IDEAL 2009. Lecture Notes in Computer Science Volume 5788, 2009, pp 82-91.

Tzu, S. (2013). The art of war. Orange Publishing.

Valdes, A., & Skinner, K. (2001, January). Probabilistic alert correlation. In Recent Advances in Intrusion Detection (pp. 54-68). Springer Berlin Heidelberg.

Whaley, B. (1982). Toward a general theory of deception. The Journal of Strategic Studies, 5(1), 178-192.

Williams, J., & Torres, A. (2014). ADD - Complicating Memory Forensics Through Memory Disarray. Presented at ShmooCon 2014 and archived at https://archive.org/details/ShmooCon2014_ADD_Complicating_Memory_Forensics_Through_Memory_Disarray. Retrieved June 8, 2014.

Yuill, J., Denning, D. E., & Feer, F. (2006). Using deception to hide things from hackers: Processes, principles, and techniques. North Carolina State University at Raleigh, Department of Computer Science.

Zhai, Y., Ning, P., Iyer, P., & Reeves, D. S. (2004, December). Reasoning about complementary intrusion evidence. In Computer Security Applications Conference, 2004. 20th Annual (pp. 39-48). IEEE.

Zomlot, L., Sundaramurthy, S. C., Luo, K., Ou, X., & Rajagopalan, S. R. (2011, October). Prioritizing intrusion analysis using Dempster-Shafer theory. In Proceedings of the 4th ACM workshop on Security and artificial intelligence (pp. 59-70). ACM.

---

[i] http://www.hp.com/go/ArcSight
[ii] http://sourceforge.net/projects/libpcap/
[iii] http://sourceforge.net/projects/unbbayes/
[iv] http://www.secdev.org/projects/scapy/
[v] http://www.snort.org/
[vi] http://www.backtrack-linux.org/downloads/
[vii] http://www.wireshark.org/