Deceptive Self-Attack for Cyber-Defense

Jared Chandler
Tufts University
jared.chandler@tufts.edu

Adam Wick
Fastly
awick@fastly.com

Abstract

The asymmetry between cyber-defense and cyber-offense is well-known; defenders must perfectly protect their systems, while attackers need only find one flaw. Defensive cyber-deception has been proposed as a way to mitigate this problem, by using various techniques designed to require attackers to defend themselves from misdirection, false data, and counter-attack. In this paper, we propose a new cyber-deception technique: deceptive self-attack (DSA). DSA modifies network and systems to give the appearance that an unknown third party is also at work attacking the same systems. It is our contention that the presence of this (deceptive) adversary pressures real adversaries in novel ways useful to cyber-defense; and discuss these effects. As a study in DSA, we present and evaluate SOUNDTHEALARM*, a SMT-solver based system for generating deceptive self-attack network traffic.* SOUNDTHEALARM *uses public attack signatures from the Suricata intrusion detection system to automatically generate network traffic consistent with a particular cyber-attack signature.*

Keywords: Traffic Generation, Intrusion Detection, Cyber Deception, Deceptive Self-Attack, Synthesis

1. Introduction

Cyber-defense is by nature an asymmetric conflict. A defender must do everything correct every time, while an attacker only needs to exploit a weakness once to gain a foothold on a network. Traditional cyber-defense focuses on providing leverage to the defender through tooling and analysis, attempting to extend the abilities of a limited number of security professionals to create out-of-scale benefits. While huge advances have been made in these tools (including improvements in software development methodologies that ensure the proper operation of software, improvements in observability to detect intrusion, the creation of runtime access control to limit the capability of a compromised system, etc.), at the end of the day the attacker need focus on a relatively small set of problems, while the defender needs to manage many more. Cyber-deception addresses this asymmetry by increasing the costs to the adversary through deceit: by delaying, degrading, misleading, and/or discouraging attacks in the hopes of denying their goals [1, 2, 3]. One interesting commonality across many cyber-deceptions is a tendency to try to mask properties of the real system with properties of a real, reasonable, and correct false system. In this paper, we present a new cyber-deception strategy that instead tries to extend the real system with signatures of a system under attack, even when it is actually in a good state. We call our approach *deceptive self-attack*, or DSA. We define deceptive self-attack as an apparent cyber-attack on a defender's network that an adversary misattributes to a group other than the defender, such as some other adversary. The goal of this work is, much like other cyber-deceptions, to trick the adversary into making decisions that are beneficial to the defender through the use of misinformation. In this particular case, if the real attacker believes that there is another attack on the same system, this belief can potentially cause them to reconsider the cost/benefit of attacking the host. Even if they do remain on the host, DSA allows a defender to impose additional burden on the adversary by increasing the factors they must consider while on the network and/or changing the adversary's time horizon, thus increasing opportunities for an adversary to inadvertently reveal themselves.

In this paper, we outline DSA, present case studies of three usage scenarios, and then present a proof-of-concept tool for generating DSA network traffic either on a host or as a network flow using publicly available intrusion detection system (IDS) attack signatures. The goal of our proof-of-concept tool, which we call SOUNDTHEALARM, is to produce traffic that is both realistic and tuned to both the kinds of information or systems that a defender wants to protect and an attacker seeks to exploit.

The paper begins with an overview of the DSA technique in Section 2, along with background on its context, the threat model assumed, and the obstacles to implementation. This section also includes an argument of the benefits of DSA when successful. Next, in Section 3, we introduce SOUNDTHEALARM, our proof-of-concept tool that automatically generates DSA network traffic using well-known IDS attack signatures[1]. Using this proof-of-concept as our starting point, we then present case studies outlining the use of DSA for the purposes of deterrence, detection, and response in Section 4, and evaluate SOUNDTHEALARM's ability to trigger both the Suricata[2] and Snort[3] intrusion detection systems in Section 5. Finally, we then discuss related work to DSA, and conclude with the risks, limitations, and future directions for this research.

2. Deceptive Self-Attack

We define deceptive self-attack (DSA) as actions taken by a defender to create the perception in an adversary that another adversary (a competitor) is attacking resources under the defender's control. Intuitively DSA strives to make the adversary believe that there is another entity, with different goals, at work with malicious intent on the same device or network. To be successful, we believe that DSA must have four key characteristics. First, DSA attempts must be *perceived* by the adversary as something separate from ordinary operation. In addition, the effects of the attack must be perceivable by the adversary from their foothold: it must generate network effects that will be transmitted to the adversary's open sockets or listeners, must generate files that will be discovered by the adversary during its normal operation, etc. Second, the adversary must be able to *recognize* the perceived change in its environment as an attack or other malicious activity, rather than as normal operation of the system. Third, the defender must, at all times, be able to *control* the frequency and nature of actions taken by the DSA system. Fourth, like all effective cyber-deceptions, DSA must be *unrecognizable* as the actions of the defender. However, DSA extends this misdirection by inducing the adversary to misattribute the attack to a third party. The first three of these characteristics are common to many other cyber-deception techniques; it is the fourth, coupled with the deceptive third party, that is the key insight of DSA.

[1]http://rules.emergingthreats.net/open/suricata
[2]https://suricata.io
[3]https://www.snort.org

2.1. Background Context

Do adversaries pay attention to network traffic? For several different categories of malicious actors, there is evidence that they both collect and exploit network traffic as part of system compromise and as part of standard cyber-reconnaissance techniques [4]. Numerous pieces of malware include mechanisms for packet capture on infected systems [5, 6, 7], and instances of large-scale traffic redirection have been observed [8]. Finally, there is a robust ecosystem of tools for the interception[4][5] and exploitation [6][7] of network traffic. We consider this evidence of a channel for DSA to influence adversaries.

Do adversaries care about competition from other groups? For botnet operators and crypto-mining gangs, there is evidence that they care a great deal. For crypto-mining malware, competitor processes running on the system consume resources that decrease the profitability of an adversary's mining. Accordingly, the adversary's malware will terminate competitor processes to ensure the greatest share of resources is available for their mining purposes [9, 10, 11].

Similarly, Botnet operators worry about losing control of the compromised systems to other competing botnets [12]. Such losses diminish the power and profitability of the botnet. To prevent these losses, botnet malware may close off or patch the vulnerabilities used to gain initial access to a system to prevent other botnets from stealing it [13, 14, 15].

These two examples illustrate adversaries acknowledging that they work in a competitive ecosystem, and taking additional steps to detect and degrade their competitors while defending their own access to a resource. Our goal with deceptive self-attack is to both promote further development of these protections – as this development time comes at the "expense" of building new attacks – as well as to trigger as many of these defenses as possible, hopefully making bots more easier to spot.

In addition, for more subtle attackers, such as those seen in industrial or governmental espionage, our goal with deceptive self-attack is to push adversaries to reconsider their timelines. Espionage-focused adversaries are much more likely to have more extended timelines than botnet authors, as they often are much more interested in avoiding attribution. As a result, these attacks typically present fewer observable markers to intrusion detection software, and can be much more

[4]https://mitmproxy.org
[5]https://www.ettercap-project.org
[6]https://github.com/odedshimon/BruteShark
[7]https://github.com/lgandx/PCredz

difficult to spot. However, the existence of another attacker – and, in particular, another less sophisticated attacker – can complicate their cost/benefit calculus. A less subtle operator on the same machine is much more likely to cause a defender to notice something wrong with the system, and potentially trigger a deeper forensic analysis; an analysis that may trigger the discovery of both compromises. As a result, the espionage calculus changes: would it be better to attempt to migrate to a different host, to avoid getting caught in this net? Or should the adversary pursue a more aggressive timeline, to maximize their utility before the system is shut down? The first case is a clear win for the defender, as the intruder is now off the network; the second case is more complex, but we contend is also beneficial, as more aggressive timelines are usually associated with more easily discovered attacks.

2.2. Benefits

We consider the following possible benefits from employing DSA.

First, DSA reduces an adversary's window to take action. The introduction of a competitor onto the network forces the adversary to not only consider the defender's actions, but whether the competitor will cause some change either through their action, or by drawing the defender's attention which prevents the adversary's objectives. In this manner, the adversary has an incentive to either act immediately, or to delay indefinitely adopting a wait-and-see attitude toward the competitor's actions.

Next, DSA adds realism to a network. Real networks are broken and have flaws. The presence of attack traffic and a competitor on the network reinforce what an adversary already knows: the network is vulnerable to those seeking to exploit it. DSA adds believably to a honey-network [16, 17, 18] beyond what simulating legitimate users can provide [19, 20, 21], and accordingly is complementary to such systems.

DSA creates uncertainty in the mind of the adversary. The presence of a competitor forces the adversary to consider that their perception of the network or resources may be subject to manipulation by the competitor as well as the defender. The adversary must consider that any information they extract from the target network could have been already manipulated by the competitor.

DSA creates more work for the adversary, forcing them to consider the potential motives and actions of the competitor in addition to those of the defender. Just as the defender tries to protect their resources on the network, the adversary must consider protecting their tools, techniques, and targets from the competitor.

Trying understand the scope of a competitor's actions, while operating in an environment with a competitor, can cause an adversary to make unforced errors. These errors result in an adversary inadvertently disclosing their presence beyond what they already must do to explore the network and accomplish their goals.

2.3. Threat Model

We assume as our threat model an adversary who has gained access to a network and is able to intercept network traffic [5, 6, 7]. This access may be via a traditional computing platform, such as a server or workstation, or through the compromise of a router, switch or other network device. Once on the network, we presume that our adversary has the capability to exfiltrate data, corrupt/destroy data on the systems they have compromised, and attempt lateral network movement.

Finally, we assume that without intervention on the defender's part, or a decision on the adversary's, the adversary will maintain ongoing persistent access to this network.

2.4. Obstacles

The key obstacle for DSA is providing a sufficiently realistic attack that the adversary is likely to respond, without actually affecting the security of the system. One way to manage this risk is through the use of attacks that are noisy enough to be noticed but also expected to fail. For example, launching an exploit against a server that could realistically have been expected to be patched. This scenario provides the adversary with a view that someone is attempting an attack, without actually compromising the system. Closely related is the issue of positioning attack traffic such that an adversary can encounter it, either as a network flow or potentially local to a host or honeypot expected to be compromised. The application of DSA must be carefully planned by administrators to strike this balance between the severity of the perceived attack and the security of the system.

Beyond this key obstacle, we note two additional obstacles: (1) the realism of the exploits versus the sophistication of the deceptive competition and (2) automation.

With regard to the first, it is important that users of DSA do not accidentally disclose vulnerabilities to adversaries that the adversaries were not previously aware of. As a result, the kinds of attacks pursued by DSA agents are likely to be based on well-known vulnerabilities or botnets.

With regard to automation, we note that one approach to DSA would be to conduct self-attack in the same manner as a penetration test, using a red-team to attempt to gain access to resources on the network. This approach has several disadvantages and risks. First, red-teams are costly and can't be everywhere at once [22, 23]. Two of our case studies describe using DSA in an ongoing and continuous fashion. Utilizing actual red-teams to perform this sort of continuous attack is unrealistic. Second, by observing actual red-team or penetration-testing operations on the network, the adversary could gain insight into what exploits and techniques might actually work, as these teams would need to use real exploit methodologies to discover real vulnerabilities on the network.

Our approach to DSA, which we call SoundTheAlarm, automates the process of DSA to achieve its goals without requiring costly human intervention. It also uses off-the-shelf attack signatures, rather than attacks, to (intuitively) create the smoke of an attack without the accompanying fire. We perform this deception by automatically creating detectable network traffic with the signatures of real-world well-known attacks, rather than running the attacks themselves. The generated traffic can then be introduced to the network as a flow, or generated on and confined to the interface of a single host.

3. DSA Traffic Generation Method

Our approach to DSA starts with the requirement that our end result must be readily and unambiguously identifiable as an attack. In trying to determine how to ensure this, our first key insight is that intrusion detection systems (IDS), which exist to monitor for attacks, leverage public – and continuously updated – signatures of malicious network traffic. As new attacks are uncovered by security researchers, system administrators and cyber-security organizations, corresponding signatures to identify the traffic are created and shared so that other users may identify malicious activity and take action. Our second key insight is that we can use these detection signatures as recipes to generate network traffic that will match the signature and be flagged as malicious.

For our implementation, we use signatures from the open source Suricata threat detection engine.[24] Each signature is composed of a set of properties a packet or packets must satisfy in order to be considered malicious by the IDS. An example Suricata signature is shown in Figure 1. Signatures may include network IP addresses, source and destination ports, the contents of the packet at various locations, the order between packets and other relationships within the data. These signatures are commonly written using a domain specific language [8].

```
alert tcp $EXTERNAL_NET any -> $HOME_NET any
(msg:"ET TROJAN Possible Metasploit Payload Common
Construct Bind_API (from server)";
flow:from_server,established; content:"|60 89 e5
31|"; content:"|64 8b|"; distance:1; within:2;
content:"|30 8b|"; distance:1; within:2;
content:"|0c 8b 52 14 8b 72 28 0f b7 4a 26 31 ff|";
distance:1; within:13; content:"|ac 3c 61 7c 02 2c
20 c1 cf 0d 01 c7 e2|"; within:15; content:"|52 57
8b 52 10|"; distance:1; within:5; metadata:
former_category TROJAN; classtype:trojan-activity;
sid:2025644; rev:1; metadata:affected_product Any,
attack_target Client_and_Server, deployment
Perimeter, deployment Internet, deployment Internal,
deployment Datacenter, tag Metasploit,
signature_severity Critical, created_at 2016_05_16,
updated_at 2018_07_09;)
```

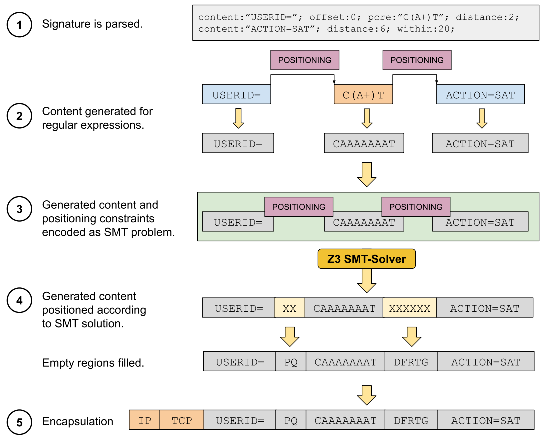
Figure 1. Example Suricata Signature.

For this work, we focused the packet payload portions of signatures drawn from the Suricata IDS. These descriptions specify portions of the packet payload to match, the values required for a match, and positioning relationships between matches. The descriptions of matches use a domain specific language for regular expressions to describe the byte-values matched by the signature, which then describes a finite state-machine [25] which can act as a Boolean function, recognizing some byte-strings and excluding others. Fundamentally, in SoundTheAlarm, we invert these finite state-machines such that they become generating functions, producing only strings which the original state-machine would recognize. We then use an SMT-solver [26] to order and position these generated strings in a manner satisfying the constraints.

In more detail, our approach to creating a DSA message from a Suricata signature is broken into five main steps, as illustrated in Figure 2:

1. **Signature Parsing.** We parse the signature and identify the portions specifying content to be matched. These are either `content` portions specifying an exact byte-string which should appear in the payload, or `pcre` portions specifying a regular expression which should recognize a portion of the payload. These match specifications may also contain positioning constraints describing where the match should be located in terms of absolute position from the start of the payload (`offset`, `depth`), and in position relative to previous matches (`distance`, `within`). We illustrate these position constraints and meanings in Figure 3.

[8]https://suricata.readthedocs.io/en/suricata-6.0.0/rules



```
content:"USERID="; offset:0; pcre:"C(A+)T"; distance:2;
content:"ACTION=SAT"; distance:6; within:20;
```

1 Signature is parsed.
2 Content generated for regular expressions.
3 Generated content and positioning constraints encoded as SMT problem.
4 Generated content positioned according to SMT solution.
Empty regions filled.
5 Encapsulation
POSITIONING
POSITIONING
USERID=
C(A+)T
ACTION=SAT
USERID=
CAAAAAAAT
ACTION=SAT
Z3 SMT-Solver
USERID= XX CAAAAAAAT XXXXXX ACTION=SAT
USERID= PQ CAAAAAAAT DFRTG ACTION=SAT
IP TCP USERID= PQ CAAAAAAAT DFRTG ACTION=SAT

Figure 2. Steps for generating payloads from signatures.

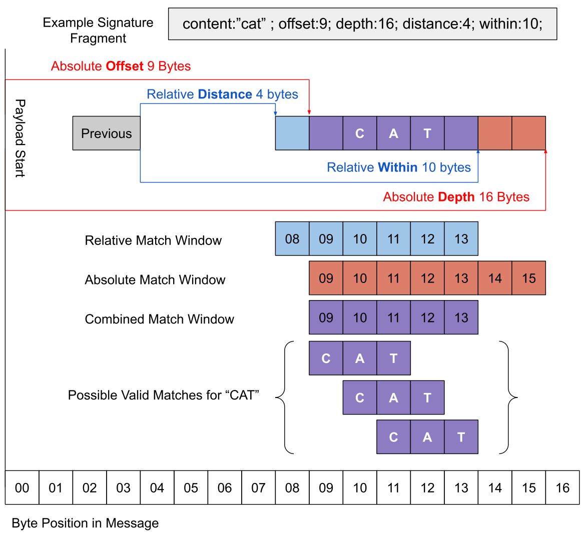


Figure 3. Example of Suricata content position constraint meanings.

2. **Regular Expression Content Generation.** For each `content` and `pcre` entry we generate a matching byte-string. For `content` entries we use the match value directly, while for `pcre` entries we use an open-source library [9] to generate matching byte-strings from the regular expression.

3. **Constraint Solving.** The expressive nature of the Suricata signature domain-specific-language means that the constraints on each byte-string and their positioning can be very complex, due to the ability to express relative position constraints between matches. SoundTheAlarm solves this problem by encoding the constraints for a message as a boolean satisfiability problem using the Z3 SMT-solver [26]. Z3 searches for placements of each byte-string such that all of the position constraints are satisfied. If it finds a set of such positions that meet all of these constraints, Z3 reports these as a solution.

4. **Payload Generation** SoundTheAlarm then creates a payload by positioning each byte-string at a specific position determined by Z3. SoundTheAlarm can then either fill any gaps with random values, or leverage a set of existing packets to transcribe the attack signature portions onto an otherwise innocuous payload for added realism.

5. **Encapsulation.** We then encapsulate the generated packet in the appropriate IP, TCP and UDP wrappers using any port information specified by the signature. We generate these wrappers using an open-source packet manipulation library [10]. In cases where the signature specifies that a 3-way TCP handshake has taken place, we create the appropriate precursor packets to give the appearance of a TCP connection having been established.

If we desire another message from the same signature, we can do so by restarting this process at step 2, to generate new byte-strings for `content` and `pcre`, or at step 3, by repeating the call to Z3 but asking for a different solution that satisfies the signature constraints.

Our proof-of-concept tool is implemented as a Python script, which takes Suricata format signatures as input and produces DSA messages as output. Such messages can be generated and validated in real-time or stored for transmission at a later time.

The design of SoundTheAlarm has several key advantages. First, we can generate attack traffic for new signatures as they are developed, allowing our method to stay current as new attacks are uncovered. Second, we are able to leverage a large baseline set of signatures from which to generate attacks, sufficient to give coverage for many network environments and services. Third, when kept in sync with the IDS or host-based security system, our method does not introduce vulnerabilities or vulnerable systems to the network, nor does it risk exploit tools being uncovered; the IDS will catch any real uses of the attack generated by an adversary. Finally, our method does not disclose

[9]https://github.com/asciimoo/exrex

[10]https://scapy.net

anything which is not already public knowledge, as all signatures used are freely available.

Next, we focus on how our DSA traffic generation technique could be employed to achieve cyber-defense objectives in Section 4 followed by our evaluation of SoundTheAlarm in Section 5.

4. DSA Operation Case Studies

To characterize how we envision DSA being used, we present three scenarios as case studies reflecting the cyber-defense objectives of deterrence, detection, and disruption.

4.1. Case Study 1: Deterrence

In our first case study, the adversary has gained access to the network, and is performing a basic reconnaissance by sniffing network traffic. They do this to understand what systems are active on the network, to understand which services are being used on the systems, and to look for information (such as credentials) that could aid in exploitation.

With DSA enabled, the adversary will see our attack traffic, and potentially recognize it as an attack by some other entity. Whether automatically, or based on a command and control thread back to a human operator, the adversary concludes they have already have competitor on the network. They then weight their goal of exploiting the network against:

- the risk that data may have already been altered,
- the risk that the competitor may detect the adversary's actions and respond, and
- the risk that the defender may have or soon will be alerted to the presence of one or both groups on the network.

In this scenario, the adversary determines that the risk outweighs the benefit and conducts no further exploitation.

4.2. Case Study 2: Detection and Distraction

Our second case, shown in Figure 4 study proceeds in line with the first, however once realizing there is a competitor on the network, the adversary elects to explore further. The attacker spends time considering what the competitor is attacking, as opposed to looking at real systems for exploit. Reasoning that the systems of interest to their competitor might be of interest to themselves, the adversary probes both the target of the competitor's attack (a honeypot) and the competitor's infrastructure for launching the attack. In doing each of these, the defender is alerted to the presence of the adversary. The defender detects the probing, and any exfiltrations of honey-data or honey traffic back to the adversary's infrastructure, gaining insight helpful for later attribution.

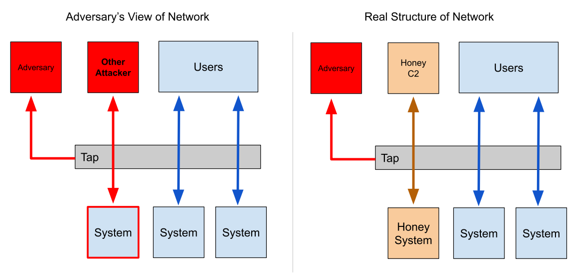


Figure 4. Illustration of the adversary's perception of a network using DSA compared to reality.

Next, the adversary reasons that the competitor gained access either through the same method as the attacker, potentially left unpatched or misconfigured, or through some other vulnerability. As a result, the adversary spends time searching for other "unlocked doors" such that they could gain exclusive control of the network, and disrupt the competitor. This effort requires the adversary to expend time and resources which would otherwise go toward further exploitation. The adversary reasons incorrectly that the attacks being used by their competitor have so far gone unnoticed by the defender, and consequently attempts to use these attacks to exploit other real systems on the network. By leveraging these ineffective attack techniques, the adversary characterizes to the defender the extent and nature of their interests. Having been alerted to the presence of the adversary, having indicators of their interest and the infrastructure under their control, the defender can choose an appropriate response.

4.3. Case Study 3: Disruption

Our third case study assumes an adversary has gained access to a network on which DSA is not being actively conducted, but where the defender has gained awareness of the adversary through some other technique; a technique that they do not wish to disclose as doing so would risk identifying the source of the intelligence. The defender, however, does wish to take action to remove the adversary from the network.

To do so, the defender initiates DSA, creating a believable subtext for their network remediation. The adversary encounters and perceives the DSA to be indicative that a competitor has also gained access. When the defender remediates the network and the adversary loses access, the adversary conclude erroneously that the defender's actions were the result of the noisy competitor, and not through failing of their own security.

5. Evaluation and Discussion

We evaluate our proof-of-concept tool for generating DSA traffic towards the goal of determining how effective our technique is in generating a broad amount of attack traffic. To do so, we split our evaluation into two components: determining what percentage of our input IDS signatures SoundTheAlarm can generate traffic for, and then what percentage of the generated traffic successfully can trigger an IDS. The first result shows how effective our technique is at parsing, processing, and synthesizing traffic given the wide variety of signatures in our corpus; we argue that our strong result shows that our technique is flexible across a wide range of signatures. Once we had done so, we then fed the generated traffic back into the IDS, to determine if our generation techniques were capable of triggering alerts; our results suggest that our approach is promising, as we were able to successfully emulate approximately 96% of the signatures we evaluated.

While our ultimate goal is to assess the impact of DSA on red-team or adversary subjects through a honey environment, in this work we focus on first generating a plausible stimulus for follow-on evaluation described in Section 7.

5.1. Experimental Setup

To perform our evaluation, we selected the state-of-the-art Suricata and Snort intrusion detection systems. To evaluate SoundTheAlarm, we used the Proofpoint Emerging Threats[11] community signature set containing 19988 unique signatures, and then filtered this set down to include only TCP and UDP signatures. We excluded signatures that inspect http traffic, leaving these for future work, as a means of scoping the research for initial evaluation. We also excluded signatures that contained protocol-specific field identifiers, and advanced criteria such as sampled packet frequency and message size thresholds, as again, we wished to identify our technique's utility before expanding to cover all cases. For reference, our TCP evaluation set contained 692 candidates drawn from an initial 3399 signatures, and our UDP evaluation set contained 95 candidates drawn from an initial 369 signatures. All experiments were run on an instance of Google Colaboratory[12] with 2 Intel® Xeon® 2.20Ghz CPUs and 12GB of RAM.

[11]https://rules.emergingthreats.net/open/suricata-5.0

[12]https://colab.research.google.com

5.2. Evaluating Generation

For each signature, we used SoundTheAlarm to generate a triggering message by inverting the signature as described in Section 3. Once the payload was generated, each message was then appropriately embedded in TCP or UDP packets with the source and destination ports pulled from the signature definition, and then into IP packets.

Our initial result is that we were able to generate messages for 685 out of the 692 TCP signatures (99%), and 89 out of the 95 UDP signatures (94%) we provided to SoundTheAlarm. This result suggests that our technique, which uses Z3, can scale reasonably to this class of generation problem. Since SMT-solvers can sometimes face significant performance (or halting) problems unexpectedly, we were encouraged that we saw none of this behavior in our sample set, with no message taking longer than 1 second to generate.

5.3. Evaluating Believability

From the generated packets, we then constructed a candidate network trace (PCAP) for analysis by the IDS. Each PCAP was provided to a Suricata instance configured to use only the candidate signature under examination; this improved startup time and also ensured that we did not generate false positives by generating a message for one signature that was incidentally flagged by another. We also used an automatic process to parse the output of the IDS to determine if the provided PCAP triggered an alert for an attack, to avoid transcription errors.

Of the 685 candidate TCP signatures our tool generated payloads for, 672 were identified as attacks by the IDS, while 13 were not, resulting in a success rate of 98%. Of the 89 candidate UDP signatures our tool generated payloads for, all 89 were identified as attacks by the IDS; a success rate of 100%, resulting in an overall (TCP and UDP) success rate of 98%. We consider this a strong positive result for a proof-of-concept system, as it was able to effectively generate deceptive self-attack messages on demand for almost every candidate signature.

Next, we sought to evaluate how our generated payloads generalized on Snort, a well known IDS alternative to Suricata. Both Snort and Suricata use a common signature syntax. Suricata allows protocol specific extensions in signatures, but rules are otherwise interchangeable, allowing our candidate Suricata signatures to be used directly with Snort. For each of the 672 TCP and 89 UDP signatures which Suricata identified as an attack, we provided the signature rule and PCAP generated by SoundTheAlarm to an instance of Snort. Snort identified all 672 TCP and 89 UDP pairs as attacks, indicating our proof-of-concept approach generalized across IDS systems. Interestingly, some of our message generation failures appear to be due to malformed signatures. We observed 4 signatures which specified that the same portion of the payload should simultaneously have two different values. In those cases, our solver would simply state that it was impossible to generate a valid message, and halt.

A further 8 signatures failed to generate messages due to a non-standard ordering of the `content` portions of the signature. Once corrected, SoundTheAlarm was able to successfully generate messages which triggered an alert. Finally, 1 signature failed to generate a message due to a failure of the regular expression inversion step. For cases where a message was generated correctly, but the IDS failed to trigger on the input we identified several interesting root causes. 7 signatures escaped specific characters which Suricata ignores, but which were included in our generated messages. 5 signatures contained malformed regular expression. Finally, one signature used both `content` and `pcre` elements to reference the same portion of the byte-string. One assumption we had made is that each `content` or `pcre` element in a signature correspond to non-overlapping portions of the payload. We believe that our choice of a SMT-solver based approach will allow us to handle such instances in future work.

6. Related Work

The goal of DSA is to help a defender deter, detect and disrupt the activities of adversaries on their network through deception. We consider three categories of relevant related work: general approaches to cyberdeception, works related to DSA as a strategy, and works related to our generation of network traffic from IDS signatures.

The use of cyberdeception [1] to detect adversaries within a network is not new [27], although much of this work focuses on heavyweight [28], virtual [29] or lightweight [30, 31] honeypots meant as decoys for real systems, rather than SoundTheAlarm's deceptive annotation approach. Honeypots typically serve as ablative detection systems and as an observational platform to expose the adversary techniques. DSA seeks to complement this approach by baiting an adversary into disclosure through the presence of deceptive attack traffic on the network and/or hosts.

Works related to DSA as a strategy include game theoretic approaches with multiple adversaries [32], which led us to consider how to add decoy players to a two-player game. Related research focuses on how cyber-security professionals attack and defend networks [2, 33, 34]. Works which involve compromising systems and then defending from competitors [35, 36] are most relevant to DSA.

Related traffic generation works focus on testing if an IDS is configured correctly [37] or measuring its effectiveness at identifying attacks [38, 39]. While some approaches replay real traffic, others generate simple packets; these include Erlacher et al.'s idsEventGenerator [40], Nidsbench[13], and IDSwakeup[14]. SoundTheAlarm's SMT-solver approach synthesizes messages which adhere to the positioning constraints in the Suricata signature language, unlike other works. SoundTheAlarmis also uniquely able to create stateful sequences of packets wrapping synthesized messages.

The second approach to the generation of network traffic focuses directly on generating realistic, non-adversarial network messages, either for the purposes of direct human deception [41] or to skew traffic flow data [31]. While our immediate aim is to to trigger an IDS, ultimately our goal is deceiving adversary into misidentifying our generated messages as an attack. Dyer et al.'s work on format-transforming encryption [42] focuses on embedding an encrypted communication channel in messages easily mistaken for three innocuous protocols. Our approach attempts to generate deceptive messages for any Suricata attack signature. Yu et al's work [43] leverages a program and corresponding attack-pattern regular expression to generate attack strings. Finally, Chandler et al.'s work [44] on botnet cyber-deception presents a network approach for deceiving a botnet operator that their botnet is functional throughout mitigation and removal. SoundTheAlarm instead focuses on generating traffic capable of indicating another botnet is at work on the same network.

7. Scope and Future Work

Our proof-of-concept tool illustrates a coarse approach to automatically generating attack traffic, and is thus limited in which IDS signatures it can generate traffic for; we have discussed examples in Section 5. We believe that many of the technical limitations of our approach can be overcome with additional engineering

[13]https://packetstormsecurity.com/UNIX/IDS/nidsbench
[14]https://github.com/SavSanta/idswakeup

while maintaining the existing technique. For example, extending our support for DNS and HTTP requires extending our infrastructure to deal with more extensive use of strings. That being said, support for strings already exists within Z3, so this presently appears to primarily be an engineering problem. One possible direction is using our technique to automatically write attack-signatures onto existing network flows at the host level. Similarly, we would like to evaluate our technique's ability to trigger alerts on host-based security systems. We see this approach as having the potential benefit of tailoring DSA traffic in real-time to that of a host using the its existing traffic as a template, and creating attack alerts in host log files for an adversary to discover.

Perhaps more critically, our method of generating attack traffic may be sufficient to trigger an IDS, but whether it would stand up to human scrutiny by a a forensics expert is unknown. Now that we have shown the basic approach is feasible, however, we hope to explore evaluation of our generated traffic by human operators. Along a similar line of investigation, we note that DSA is only useful if an adversary notices it. There is no guarantee that an adversary will look at network traffic or perform analysis such that they conclude an attack is taking place, or even that they will change their behavior should they notice it. Again, we hope to evaluate responses to these deceptive inputs in future experiments using red teams and by placing DSA traffic on honeynets.

8. Conclusion

In this paper, we introduced the concept of deceptive self-attack (DSA), a technique for improving cyber-defense by through deterrence, detection, and disruption. We outlined three case studies describing possible use cases for DSA, and presented our proof-of-concept approach to automatically generating DSA traffic from open-source intrusion detection signatures, successfully generating traffic classified as an attack in 96% of our evaluation cases.

Acknowledgments

This material is based upon work partly supported by the Defense Advanced Research Projects Agency (DARPA) under Contract No. HR0011-19-C-0073. The views, opinions, and/or findings expressed are those of the author(s) and should not be interpreted as representing the official views or policies of the Department of Defense or the U.S. Government.

References

[1] E. Al-Shaer, J. Wei, W. Kevin, and C. Wang, "Autonomous cyber deception," *Springer*, 2019.

[2] K. Ferguson-Walter, T. Shade, A. Rogers, M. C. S. Trumbo, K. S. Nauer, K. M. Divis, A. Jones, A. Combs, and R. G. Abbott, "The Tularosa Study: An Experimental Design and Implementation to Quantify the Effectiveness of Cyber Deception.," tech. rep., Sandia National Lab.(SNL-NM), Albuquerque, NM (United States), 2018.

[3] S. Achleitner, T. La Porta, P. McDaniel, S. Sugrim, S. V. Krishnamurthy, and R. Chadha, "Cyber deception: Virtual networks to defend insider reconnaissance," in *Proceedings of the 8th ACM CCS international workshop on managing insider security threats*, pp. 57–68, 2016.

[4] W. Mazurczyk and L. Caviglione, "Cyber reconnaissance techniques," *Communications of the ACM*, vol. 64, no. 3, pp. 86–95, 2021.

[5] "LightBasin: A Roaming Threat to Telecommunications Companies." https://www.crowdstrike.com/blog/an-analysis-of-lightbasin-telecommunications-attacks. Accessed: 2022-09-03.

[6] "Quick and Simple: BPFDoor Explained." https://thehackernews.com/2022/06/quick-and-simple-bpfdoor-explained.html. Accessed: 2022-09-03.

[7] "ZuoRAT Hijacks SOHO Routers To Silently Stalk Networks." https://blog.lumen.com/zuorat-hijacks-soho-routers-to-silently-stalk-networks. Accessed: 2022-09-03.

[8] C. C. Demchak and Y. Shavitt, "China's maxim–leave no access point unexploited: The hidden story of china telecom's bgp hijacking," *Military Cyber Affairs*, vol. 3, no. 1, p. 7, 2018.

[9] "This is what happens when two ransomware gangs hack the same target - at the same time." https://www.zdnet.com/article/two-ransomware-gangs-hacked-the-same-target-at-the-same-time-heres-what-happened-next/. Accessed: 2022-06-03.

[10] "New Jenkins Campaign Hides Malware, Kills Competing Crypto-Miners." https://www.f5.com/labs/articles/threat-intelligence/new-jenkins-campaign-hides-malware--kills-competing-crypto-miner. Accessed: 2022-06-03.

[11] "Kingminer patches vulnerable servers to lock out competitors." https://www.bleepingcomputer.com/news/security/kingminer-patches-vulnerable-servers-to-lock-out-competitors/. Accessed: 2022-06-03.

[12] H. Griffioen and C. Doerr, "Examining Mirai's Battle Over the Internet of Things," in *Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security*, pp. 743–756, 2020.

[13] J. Choi, A. Abusnaina, A. Anwar, A. Wang, S. Chen, D. Nyang, and A. Mohaisen, "Honor Among Thieves: Towards Understanding the Dynamics and Interdependencies in IoT botnets," in *2019 IEEE Conference on Dependable and Secure Computing (DSC)*, pp. 1–8, IEEE, 2019.

[14] C. Brierley, J. Pont, B. Arief, D. J. Barnes, and J. Hernandez-Castro, "Persistence in Linux-Based IoT Malware," in *Nordic Conference on Secure IT Systems*, pp. 3–19, Springer, 2020.

[15] "Gafgyt Targeting Huawei and Asus Routers and Killing Off Rival IoT Botnets." https://www.f5.com/labs/articles/threat-intelligence/gafgyt-targeting-huawei-and-asus-routers-and-killing-off-rival-iot-botnets. Accessed: 2022-06-03.

[16] I. Kuwatly, M. Sraj, Z. Al Masri, and H. Artail, "A dynamic honeypot design for intrusion detection," in *The IEEE/ACS International Conference on Pervasive Services, 2004. ICPS 2004. Proceedings.*, pp. 95–104, IEEE, 2004.

[17] A. Mairh, D. Barik, K. Verma, and D. Jena, "Honeypot in Network Security: A Survey," in *Proceedings of the 2011 International Conference on Communication, Computing & Security*, pp. 600–605, 2011.

[18] N. Provos, "Honeyd-A virtual Honeypot Daemon," in *10th DFN-CERT Workshop, Hamburg, Germany*, vol. 2, p. 4, 2003.

[19] H. Lin, S. Dunlap, M. Rice, and B. Mullins, "Generating Honeypot Traffic for Industrial Control Systems," in *International Conference on Critical Infrastructure Protection*, pp. 193–223, Springer, 2017.

[20] I. Siniosoglou, G. Efstathopoulos, D. Pliatsios, I. D. Moscholios, A. Sarigiannidis, G. Sakellari, G. Loukas, and P. Sarigiannidis, "NeuralPot: An industrial honeypot implementation based on deep neural networks," in *2020 IEEE Symposium on Computers and Communications (ISCC)*, pp. 1–7, IEEE, 2020.

[21] "Ostinato Traffic Generator for Network Engineers." https://ostinato.org/. Accessed: 2022-06-03.

[22] J. Schab, "Tackling DoD cyber red team deficiencies through systems engineering," *SANS Information Security Reading Room. Sans Institute*, 2019. Accessed: 2022-06-03.

[23] J. J. Li and L. Daugherty, "Training cyber warriors: What can be learned from defense language training?," tech. rep., RAND National Defense Research Institute Santa Monica CA, 2015.

[24] J. S. White, T. Fitzsimmons, and J. N. Matthews, "Quantitative Analysis of Intrusion Detection Systems: Snort and Suricata," in *Cyber sensing 2013*, vol. 8757, pp. 10–21, SPIE, 2013.

[25] M. Sipser, "Regular Languages," *Introduction to the Theory of Computation*, pp. 31–98, 2006.

[26] L. d. Moura and N. Bjørner, "Z3: An efficient SMT solver," in *International conference on Tools and Algorithms for the Construction and Analysis of Systems*, pp. 337–340, Springer, 2008.

[27] F. Cohen, "A note on the role of deception in information protection," *Computers & Security*, vol. 17, pp. 483–506, 12 1998.

[28] L. Spitzner, *Honeypots: Tracking Hackers*. Addison-Wesley Professional, 2002.

[29] N. Provos and T. Holz, *Virtual Honeypots: From Botnet Tracking to Intrusion Detection*. Addison-Wesley Professional, 2007.

[30] N. Provos, "A virtual honeypot framework," in *Proceedings of the 13th Conference on USENIX Security Symposium - Volume 13*, SSYM'04, (Berkeley, CA, USA), pp. 1–1, USENIX Association, 2004.

[31] A. Wick, "I Want Your Flow To Be Lies," in *FloCon*, 2017.

[32] P. Lee, A. Clark, B. Alomair, L. Bushnell, and R. Poovendran, "A Host Takeover Game Model for Competing Malware," in *2015 54th IEEE Conference on Decision and Control (CDC)*, pp. 4523–4530, IEEE, 2015.

[33] K. Ferguson-Walter, M. Major, D. Van Bruggen, S. Fugate, and R. Gutzwiller, "The World (of CTF) is Not Enough Data: Lessons Learned from a Cyber Deception Experiment," in *2019 IEEE 5th International Conference on Collaboration and Internet Computing (CIC)*, pp. 346–353, IEEE, 2019.

[34] T. Shade, A. Rogers, K. Ferguson-Walter, S. B. Elsen, D. Fayette, and K. Heckman, "The Moonraker Study: An Experimental Evaluation of Host-Based Deception," in *Proceedings of the 53rd Hawaii International Conference on System Sciences*, 2020.

[35] K. Bock, G. Hughey, and D. Levin, "King of the hill: A novel cybersecurity competition for teaching penetration testing," in *2018 USENIX Workshop on Advances in Security Education (ASE 18)*, 2018.

[36] K. Leune and S. J. Petrilli Jr, "Using capture-the-flag to enhance the effectiveness of cybersecurity education," in *Proceedings of the 18th Annual Conference on Information Technology Education*, pp. 47–52, 2017.

[37] N. J. Puketza, K. Zhang, M. Chung, B. Mukherjee, and R. A. Olsson, "A Methodology for Testing Intrusion Detection Systems," *IEEE Transactions on Software Engineering*, vol. 22, no. 10, pp. 719–729, 1996.

[38] R. P. Lippmann, D. J. Fried, I. Graf, J. W. Haines, K. R. Kendall, D. McClung, D. Weber, S. E. Webster, D. Wyschogrod, R. K. Cunningham, *et al.*, "Evaluating intrusion detection systems: The 1998 DARPA off-line intrusion detection evaluation," in *Proceedings DARPA Information Survivability Conference and Exposition. DISCEX'00*, vol. 2, pp. 12–26, IEEE, 2000.

[39] J. McHugh, "Testing Intrusion Detection Systems: A Critique of the 1998 and 1999 DARPA Intrusion Detection System Evaluations as performed by Lincoln Laboratory," *ACM Transactions on Information and System Security (TISSEC)*, vol. 3, no. 4, pp. 262–294, 2000.

[40] F. Erlacher and F. Dressler, "How to test an ids? genesids: An automated system for generating attack traffic," in *Proceedings of the 2018 Workshop on Traffic Measurements for Cybersecurity*, pp. 46–51, 2018.

[41] S. Maucione, "Loose lips may better Air Force security with 'Prattle'," *Federal News Network*, 2017.

[42] K. P. Dyer, S. E. Coull, T. Ristenpart, and T. Shrimpton, "Protocol Misidentification Made Easy with Format-Transforming Encryption," in *Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security*, pp. 61–72, 2013.

[43] F. Yu, M. Alkhalaf, and T. Bultan, "Generating Vulnerability Signatures for String Manipulating Programs using Automata-Based Forward and Backward Symbolic Analyses," in *2009 IEEE/ACM International Conference on Automated Software Engineering*, pp. 605–609, IEEE, 2009.

[44] J. Chandler, K. Fisher, E. Chapman, E. Davis, and A. Wick, "Invasion of the Botnet Snatchers: A Case Study in Applied Malware Cyberdeception," in *Proceedings of the 53rd Hawaii International Conference on System Sciences*, 2020.