Evading Network Anomaly Detection Systems: Formal Reasoning and Practical Techniques

Prahlad Fogla and Wenke Lee
College of Computing, Georgia Institute of Technology
Atlanta, Georgia, USA
prahlad@cc.gatech.edu, wenke@cc.gatech.edu

ABSTRACT

Attackers often try to evade an intrusion detection system (IDS) when launching their attacks. There have been several published studies in evasion attacks, some with available tools, in the research community as well as the "hackers" community. Our recent empirical case study showed that some payload-based network anomaly detection systems can be evaded by a polymorphic blending attack (PBA). The main idea of a PBA is to create each polymorphic instance in such a way that the statistics of attack packet(s) match the normal traffic profile. In this paper, we present a formal framework for the open problem: given an anomaly detection system and an attack, can one *automatically* generate its PBA instances? We show that in general, generating a PBA that optimally matches the normal traffic profile is a hard problem (NP-complete). However, the problem of finding a PBA can be reduced to the SAT or ILP problems so that solvers available in those domains can be used to find a near-optimal solution. We also present a heuristic (*hill-climbing*) to find an approximate solution. Our framework can not only expose how the IDS can be exploited by a PBA but also suggest how the IDS can be improved to prevent the PBA. We have experimented with our framework using the PAYL 1-gram and 2-gram anomaly detection system, and the results have validated our framework.

Categories and Subject Descriptors

C.2.0 [**Computer-Communication Networks**]: General—*Security and protection*

General Terms

Security

Keywords

anomaly detection, polymorphic blending attack, mimicry attack

1. INTRODUCTION

As defense techniques such as intrusion detection systems (IDSs) become widely deployed, attackers now have to consider how to defeat these mechanisms when launching their attacks. In particular, polymorphic attacks are designed to evade the detection of misuse detection systems. In such an attack, each polymorphic instance of an attack can look very different. Thus, there may not be an accurate or reliable pattern that an IDS can use as an attack signature. Anomaly detection systems offer a defense against polymorphic attacks because typically each attack instance still looks different from normal data. For example, the payload of an attack packet may contain some non-printable characters or unusual byte structure; whereas the payload of a normal packet predominantly contains `ASCII` characters with predefined structure, as required by the application protocol. Indeed, several network-based anomaly detection systems, e.g., PAYL [31, 32], which model the byte (or n-gram) frequency characteristics of the normal packets, have been shown to be effective against polymorphic attacks.

It is inevitable that attackers will attempt to evade anomaly detection systems. Our previous work [10] showed that a polymorphic blending attack (PBA) can evade PAYL. A PBA is similar to a mimicry attack [30], but is applied to network IDS rather than host-based IDS. The main idea of a PBA is that, after learning the normal profile using some normal packets, the attacker can mutate a given attack instance so that the byte characteristics of the final attack packet(s) match the normal profile. We showed that such mutations can be achieved using a simple byte substitution scheme followed by byte padding. However, these techniques are not general in that they are based on heuristics that work well for PAYL but not necessarily other anomaly detection systems.

Therefore, an important open problem with polymorphic blending attacks is: given an anomaly detection system and an attack, can one *automatically* generate the PBA instances? The motivation for solving this problem is to provide a defender the means to evaluate an IDS and even improve it.

In this paper, we present a formal framework for polymorphic blending attacks. We first show that a wide range of IDS can be represented using either a regular grammar or a stochastic regular grammar. The problem of generating a PBA then becomes finding a mutated attack instance that will be accepted by the IDS regular grammar. We show that in general, generating a PBA is a hard problem (NP-complete), but it can be translated to the SAT (satisfiability) or ILP (integer linear programming) problem. Thus, solvers in these problem domains can be used to find a near optimal solution for generating a PBA. In addition, we present heuristics to find approximate solutions very efficiently. We also show that our framework can not only expose how the IDS can be exploited by a PBA but also suggest how the IDS can be modified to prevent the PBA while maintaining the desired detection performance (e.g., low false positive rate). We have experimented with our framework using the PAYL 1-gram and 2-gram anomaly detection system, and the results have validated our framework.

The rest of the paper is organized as follows. Section 2 discusses the related work. In section 3, we present our framework that includes models for the class of IDS considered in this research, and the basic steps to generate a polymorphic blending attack. In section 4, we analyze the complexity of generating a polymorphic blending attack and present some methods to find approximate solutions for the problem. We show the results of our experiments in section 5. In Section 6, we describe how to use our framework to improve an IDS. Section 7 concludes the paper.

2. RELATED WORK

Various attack mutation techniques have been used by attackers to evade misuse detection systems. [29] presents some mutation techniques commonly used at the different layers of protocols. In [23], Rubin et al. modeled different attack transformations as inference rules. These rules can be used repeatedly to generate different attack instances from a given initial attack instance.

Several attack specification languages have been developed by researchers to represent the attack signatures efficiently and accurately. NETSTAT [6] represents the attack events using a state machine called STATL. Snort [21] represents a signature using regular expressions consisting of network bytes. It also uses other packet attributes. Liang et al. [17] presented extended FSA based attack signatures. GARD [22] is a tool for regular language based generation of attack instances. It is observed that many of the proposed signatures are based on a regular grammar or state machine.

Code polymorphism and metamorphism techniques [27] are used to mutate and obfuscate the shellcodes present in the attack. Tools like tPE, AdMutate, and CLET [5, 16, 33] perform advanced code polymorphism, and are available on the Web. Several approaches have been proposed to detect polymorphic attacks. [3] uses instruction semantics information to detect different instances of malware. Kruegel et al. [13] presented graph coloring based detection of malicious code. In [28], Toth et al. observed that the maximum binary executable section in a normal `http` packet is very small and may be used to detect the presence of shellcode in a packet. STRIDE [1] checks for the presence of `NOP` sleds in a packet to find buffer overflow attacks. Polygraph [20] generates signatures for worms by finding a set of longest substrings shared between different instances of the worm.

Payload-based anomaly IDS are used to detect application-layer attacks, including polymorphic attacks. PAYL [31, 32] records the byte (or n-gram) frequency characteristics of the normal packets. If the frequency characteristics of a packet differs significantly from the normal, the packet is deemed suspicious. NETAD and LERAD [14, 18] models the first few bytes of the application layer headers of a packet using a set of rules. In [15], Kruegel et al. proposed six different models, namely, length, character distribution, probabilistic regular grammar, token set, attribute presence, and attribute order for detection of `http` attacks. They modeled different `http` attributes like `URL` path, `SQL` query string, etc. Sekar et al. [25] presented an anomaly detection system based on network protocol specifications. Specification is defined using extended finite state automaton. Statistical features based on the state transitions are monitored to detect anomalies.

In host based IDS, system call sequence is commonly used to detect intrusions. Sekar et al. [24] proposed to use a program counter along with system calls to model the normal system call sequence as a DFA. [8, 7] proposed to use call stack information to reliably detect carefully crafted intrusions. Mimicry attack was first introduced by Wager et al. [30]. It modifies the system call sequence generated by the attack code so that it matches the normal system call sequence. An advanced mimicry attack was described in [12] that can evade the IDSs that uses call stack information. Polymorphic blending attack is similar to mimicry attacks because both try to evade anomaly IDS by matching the attack characteristics to the normal profile. Mimicry attacks target host-based IDSs, whereas polymorphic blending attacks target network IDSs.

[2] raises doubts on the security of machine learning based IDSs in the face of a determined and resourceful attacker. It discussed the possible manipulation of the IDS training process so that the normal space of an IDS is gradually moved to include attack packets. A polymorphic blending attack takes a different approach and modifies the attack instance to move it closer to the normal space. CLET [5] is a publicly available polymorphism tool which tries to perform preliminary blending. CLET adds extra padding bytes in a given attack packet in an effort to match the byte frequency distribution of the attack to the normal profile. Our recent work [10] explores polymorphic blending attacks and presents basic techniques for generating such attacks. It shows that polymorphic blending attacks are feasible, and presents a case study for PAYL 1-gram and 2-gram. In this paper, we present a methodical approach to polymorphic blending attacks. Our research includes formal models and theoretical results, general algorithms, and efficient heuristics.

3. A FORMAL FRAMEWORK

The basic observation behind a PBA is that a network IDS monitoring high-speed and high-volume traffic typically uses simple statistical measures instead of complex structural or semantic information to model the normal traffic. An attacker can exploit this simplicity or limitation to devise attacks capable of evading the IDS.

In a polymorphic blending attack, it is assumed that the attacker knows the features and the algorithms used in the IDS [10]. Given some normal data packets, the attacker can generate an artificial normal profile that is close to the normal profile being used by the IDS. The attacker can roughly guess the error threshold of the IDS using an estimation of the desired false positive and detection rate of the IDS. After the attacker estimates the normal profile used by the IDS, the attacker decides the various parameters (e.g., attack vector, decryptor, encryption scheme, etc.) and structure (e.g., placement of each attack section in the payload) of the attack data. Then the attacker carefully chooses an encryption key so that the encrypted attack body closely matches the desired normal profile. Lastly the attacker pads the attack data with some normal data to match the attack packet even closer to the normal profile. A detailed description of each of the above steps can be found in [10].

In this work, we study the problem: given an anomaly detection system and an attack, can one automatically generate the PBA instances? Our approach is to develop a formal framework that starts with the models for the IDS. Based on these models, we can then reason about the complexity of the problem of generating a PBA, and develop the general algorithms for solving the problem.

3.1 Modeling Intrusion Detection Systems

In our recent work [10], we considered a class of anomaly detection systems that use only simple byte statistics of the normal traffic. We would like to generalize the concept of polymorphic blending attack to include a wide range of anomaly detection systems that use other structural information of the normal traffic.

3.1.1 Anomaly Detection Systems

Since a polymorphic attack typically mutates only the packet payload, we limit our scope to payload-based anomaly IDS. These systems record the statistics and structure of the bytes present in the normal network traffic packets. Such anomaly IDS proposed by researchers include NETAD[18], LERAD [19], service-specific

IDS [14], PAYL [31], and structure based detection of Web attacks by Kruegel et al. [15]. We observe that these IDSs can be represented as stochastic Finite State Automaton (sFSA) or equivalently stochastic Regular Grammar (sRG). sFSA is similar to FSA and has a probability assigned to all the transitions in the FSA.

3.1.1.1 PAYL.

PAYL records the average frequency of different unique n-grams that appear in normal traffic packets. An n-gram model can be described using an sFSA where each state represents the unique $(n-1)$-gram corresponding to the last $(n-1)$ bytes in the packet. A transition from state $A(a_0a_1 \cdots a_{n-2})$ to state $A'(a_1a_2 \cdots a_{n-1})$ exists if and only if n-gram $(a_0a_1....a_{n-2}a_{n-1})$ is present in the normal traffic. The probability of a transition is equal to the probability of the corresponding n-gram in the normal traffic. Every state is a start state and every state is an accept/end state. For example, 1-gram model can be represented using a single state FSA: for every unique byte in the normal traffic, there exists a transition from the state to itself; and the probability of the transition is the same as the frequency of the byte in normal traffic.

3.1.1.2 NETAD and LERAD.

Mahoney et al. presented a series of anomaly IDSs that use some network level data along with some payload data to detect intrusions. These systems use attributes such as bytes or words present at specific positions in the payload. A LERAD rule is of the form (if $word_1 = x1, \cdots, word_{m-1} = x_{m-1}$ then $word_m \in X = \{x_{1,m}, \cdots, x_{n,m}\}$). Such a rule can be seen as a regular grammar of the form $(x_1\, x_2 \cdots x_{m-1}\, \{x_{1,m}| \cdots |x_{n,m}\})$. Multiple rules can be combined using the '|' term to obtain a single regular grammar.

3.1.1.3 Structure-Based Systems.

Kruegel et al. presented an IDS for Web services. A Web traffic packet is divided into attributes and different attributes are recorded using different byte characteristics, including: attribute length, byte frequency, byte structure using sRG, and token set. As in PAYL, byte frequency can be represented using a sFSA with one state. Token set $(T = \{t_1, \cdots, t_n\})$ can be seen as a regular grammar of the form $(t_1| \cdots |t_n)$. Models of the different attributes can be combined to form a single sFSA. The length constraint on an attribute is handled separately during the blending attack generation.

In summary, the above anomaly detection systems can be represented using a sFSA. An anomaly detection system normally allows some errors. Typically, it uses some distance metric to define the distance of an observed packet from the normal profile. If the distance is smaller than a threshold, the packet is considered normal. Otherwise, the packet is considered anomalous. The distance metric is orthogonal to the IDS models and cannot be directly incorporated in the corresponding sFSA. In later sections, we will show how the distance metric is accounted for when generating a polymorphic blending attack.

3.1.2 An Example

We have shown that a wide range of anomaly IDSs represent the byte statistics and/or structure of a normal packet using either FSA or sFSA. One main reason for using FSA (or equivalently, regular expression) is that determining whether a string is generated by a FSA is very fast, and can thus be used over high speed networks.

In this work, we assume that the attacker is trying to evade an IDS that can be represented using either an FSA or sFSA. Figure 1 shows a simple example of a sFSA IDS. The IDS accepts the strings containing only the following tuples: *ab*, *ba*, and *bb*. We use this simple IDS as a running example throughout the paper.

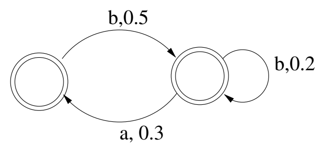


Figure 1: Simple sFSA IDS containing 3 tuples

$$\underset{\text{AV}}{\underline{aaba}}\ \underset{\text{Dec}}{\underline{cbb}}\ \underset{\text{Key}}{\underline{k_1k_2}}\ \underset{\text{Attack Code}}{\underline{00h\ 01h\ 00h\ 00h}}$$

$$\underset{\text{AV}}{\underline{aaba}}\ \underset{\text{Dec}}{\underline{cbb}}\ \underset{\text{Key}}{\underline{k_1k_2}}\ \underset{\text{Encrypted Attack Code}}{\underline{00h \oplus k_1\ 01h \oplus k_2\ 00h \oplus k_1\ 00h \oplus k_2}}$$

Figure 2: Simple attack example. Attack code is 4 byte string with `NUL` and `SOH ASCII` characters.

3.2 Polymorphic Blending Attack

As in [10], we focus on the attacks that allow arbitrary code execution. Thus, the attack packet contains a shellcode that is run on the victim host. A polymorphic attack contains five sections:

- Attack vector: exploits the vulnerability present on the victim host. Polymorphism of attack vector is achieved using different attack mutation techniques as discussed in [10].
- Attack code: the shellcode that the attacker wants to execute at the victim host. The attack code is stored encrypted in the attack packet.
- Polymorphic decryptor: decrypts the encrypted attack code and transfers control to attack code. Various code obfuscation techniques can be used to generate different instances of polymorphic decryptor.
- Decryption table: used to decrypt the attack code.
- Padding: extra (junk) data appended in order to closely match the normal profile.

In [10], we used a simple byte substitution scheme for encryption. During encryption, every attack character in the attack body is substituted by a normal character. To store the reverse substitution (or decoding) table of the simple byte substitution scheme, we use the same technique as in [10]: the index of the decoding table determines the attack character, and the entry at an index is the normal character used to substitute the corresponding attack character.

The polymorphic blending techniques studied in this paper include both *XOR* encryption scheme and byte substitution scheme. It is important to consider *XOR* because there are several existing polymorphism tools that use *XOR* based encryption already, and these tools may be extended to generate PBA. Unlike substitution, the decryption key for *XOR* is the same as the encryption key, and can be stored in a straight forward manner.

Both substitution and *XOR* are very simple schemes and are used in more complex encryption schemes. By studying PBA with these simple schemes, we hope to develop a understanding as well as solutions applicable to more complex schemes.

3.2.1 Generating An Attack Instance

Assume that the attacker has learned a (s)FSA corresponding to the (artificial) normal profile of the targeted IDS, the next step is to design an attack packet that can be accepted by the (s)FSA.

First, the attacker decides on the encryption scheme used for encrypting the attack code. Then a mutated instance of the attack vector and a polymorphic decryptor is generated, and their positions in the attack packet are determined.

The attack packet sections of the attack vector and polymorphic decryptor should preferably be accepted by the (s)FSA already. In some cases, there does not exist a path in the (s)FSA that corresponds to these attack packet sections (e.g., the mutated attack vector still contains characters not seen in normal packets), resulting in errors when the packet is matched with the IDS normal profile. If such error is greater than the IDS threshold, it means that it is not possible to generate a successful polymorphic blending attack.

The next step is to determine the encryption key. The main requirement is that the packet sections of the encrypted attack code and the decryption key should be accepted by the (s)FSA . For a sFSA, the additional requirement is for the whole attack packet to also match the transition probabilities. One approach is to first adjust the sFSA for what have been already matched by the attack packet sections of the attack vector and the polymorphic decryptor, then find a encryption key so that the sections of the encrypted attack code and the decryption key match the remaining parts of the sFSA. More specifically, the path taken by the attack vector and the decryptor in the sFSA is first identified. If there does not exist such a complete path (e.g., some transitions are not in the sFSA), there will be an error matching the attack packet with the normal profile already. If there exists multiple paths, the path with high probability transitions is chosen. Then the probabilities of the transitions present in the path are reduced according to the number of times a transition appears in the path. An encryption key is then chosen so that the sections of the encrypted attack code and the decryption key can match the adjusted sFSA closely.

The final, and sometimes optional, step is to pad the attack packet to have a desired packet length. For a sFSA, padding can be used to make the final attack packet match even more closely with the normal profile. The process works as follows. First, given the original sFSA representing the IDS, adjust the probabilities of the transitions in sFSA for the attack vector, the decryptor, the key table, and the encrypted attack code. That is, similar to the step of determining the encryption key discussed above, the sFSA is adjusted according to what have already been matched by these existing attack packet sections. Then more bytes are padded to the attack packet to match the remaining transitions and probabilities of the sFSA.

We use a simple blending attack (shown in Figure 2) to demonstrate the different concepts presented in the paper. We use an *XOR* encryption scheme with key length 2 (obviously, the encryption key and the decryption key is the same in an *XOR* scheme). The attack vector, decryptor, key, and attack code are concatenated in a given order to produce an attack packet payload. Although we assume an attack structure as shown in Figure 2, the techniques presented in this paper should work for other attack structures.

For convenience, we denote the string corresponding to the decryption key concatenated with the encrypted attack code as S_{key_ac}.

4. FORMAL ANALYSIS

As discussed in Section 3.2.1, one of the steps in generating a PBA is to find an encryption key so that the attack packet sections of the decryption key and the encrypted attack code, or S_{key_ac}, can be accepted by the FSA or the adjusted sFSA (for convenience, in this section we simply call the adjusted sFSA a sFSA). We will show that this is a hard problem even when using very simple encryption schemes, namely, substitution and *XOR*. As a corollary, the problem is hard when using more complex encryption schemes. The most direct and important corollary, however, is that the problem of generating a PBA is a hard problem.

A brute force approach to find the encryption key requires generating every possible key and checking the distance (as defined by the IDS) of the S_{key_ac} to the (s)FSA. For a simple substitution-based (encryption) scheme, this will take at least ${}^{n}P_{m}(n-m)^{n-m}$ iterations, where n is the number of unique normal characters and m is the number of unique attack characters. For *XOR* encryption with key of length l, finding an optimal polymorphic blending attack will take at least n^{l} iterations. These numbers can be very large, and thus a brute force approach is often impractical.

In this section, we first analyze the hardness of finding an encryption key that ensures S_{key_ac} is a valid string accepted by the FSA (without any transition probabilities) corresponding to the IDS. We show that this problem is NP-complete. Thus, it may not be solvable deterministically in polynomial time of the key length l or m. We show this result for both byte substitution based encryption and *XOR* based encryption. This result can be extended to show that even if we allow a solution to contain $\epsilon\|S_{key\_ac}\|$ number of invalid transitions, the problem is still NP-complete. We extend the above results and argue that the problem of finding an encryption key that optimally matches the S_{key_ac} to sFSA or finding a solution within an ϵ range of the optimal solution is also NP-complete.

4.1 NP-Completeness of the Blending Attack Problem

4.1.1 Substitution Based Encryption Scheme

We formally define the problem of finding a substitution key that ensures S_{key_ac} is accepted by the FSA of an IDS.

DEFINITION 4.1. *Given an attack code and the FSA of an IDS, the problem PBA_{sub}^{FSA} is to find a one-to-one mapping from attack characters to normal characters such that S_{key_ac} is accepted by the given FSA.*

THEOREM 4.1. *Problem PBA_{sub}^{FSA} is NP-complete.*

PROOF. For a problem to be NP-complete, the problem should be in NP and should be NP-hard.

A problem is in NP if a given solution can be verified for its correctness in polynomial time. Given a one-to-one mapping, we can easily generate the decryption key (a table) and the encrypted attack code. Since FSA is a decidable language, we can verify in polynomial time whether or not S_{key_ac} string will be accepted by the FSA. Thus, we can efficiently verify if the one-to-one mapping is correct or not.

To prove that the problem is NP-hard, we reduce the well known 3-SAT problem to PBA_{sub}^{FSA}. Consider a 3-SAT problem with q, $q \leq 128$, variables and r clauses. If q is smaller than 128, we add dummy unused variables to make the total number of variables to be 128. Suppose the 3-SAT problem is,

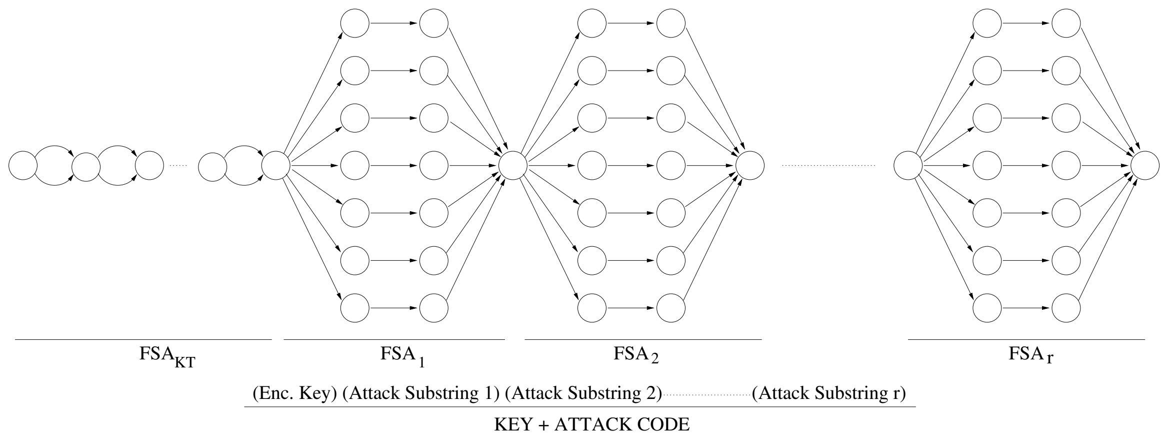
$$SAT = (x_{10} \vee x_{11} \vee x_{12}) \wedge (x_{20} \vee x_{21} \vee x_{22}) \wedge \cdots \wedge (x_{r0} \vee x_{r1} \vee x_{r2}),$$

where $x_{10}, x_{11}, \cdots, x_{r2} \in \{x_0, \overline{x_0}, x_1, \overline{x_1}, \cdots x_{127}, \overline{x_{127}}\}$.

Given the above 3-SAT, we design the PBA_{sub}^{FSA} problem as follows. For every variable x_i in the 3-SAT, we have an attack character att_i, two normal characters $norm_i$ and $norm_{i+128}$, and a corresponding entry e_{att_i} in the decryption table. e_{att_i}, $128 \leq i \leq 255$, is a dummy decryption table entry. The value of the variable x_i is related to e_{att_i} as follows.

$$\begin{aligned} x_i &= 1, \text{ if and only if } e_{att_i} = norm_i \text{ and } e_{att_{i+128}} = norm_{i+128} \\ &= 0, \text{ if and only if } e_{att_i} = norm_{i+128} \text{ and } e_{att_{i+128}} = norm_i \quad (1) \end{aligned}$$

For every clause $clause_\alpha$, $1 \leq \alpha \leq r$ in the 3-SAT, we construct a section of FSA (FSA_α) as shown in Figure 3. First, we construct the truth table of the clause (see Table 1). For each entry in the truth table, we have a path containing three edges where each edge corresponds to the value of a variable in the truth table entry. In addition to FSA_α, $1 \leq \alpha \leq r$, we have a section of FSA (FSA_{KT}) of length 256 corresponding to the key.



FSA$_{KT}$
FSA$_1$
FSA$_2$
FSA$_r$
(Enc. Key) (Attack Substring 1) (Attack Substring 2) (Attack Substring r)
KEY + ATTACK CODE

Figure 4: FSA and S_{key_ac} corresponding to the SAT problem

x_1	x_3	x_8	e_{att_1}	e_{att_3}	e_{att_8}
0	0	0	$norm_{129}$	$norm_{131}$	$norm_{136}$
0	0	1	$norm_{129}$	$norm_{131}$	$norm_8$
0	1	1	$norm_{129}$	$norm_3$	$norm_8$
1	0	0	$norm_1$	$norm_{131}$	$norm_{136}$
1	0	1	$norm_1$	$norm_{131}$	$norm_8$
1	1	0	$norm_1$	$norm_3$	$norm_{136}$
1	1	1	$norm_1$	$norm_3$	$norm_8$

Table 1: Truth table and corresponding key table for clause $x_1 \vee \overline{x_3} \vee x_8$

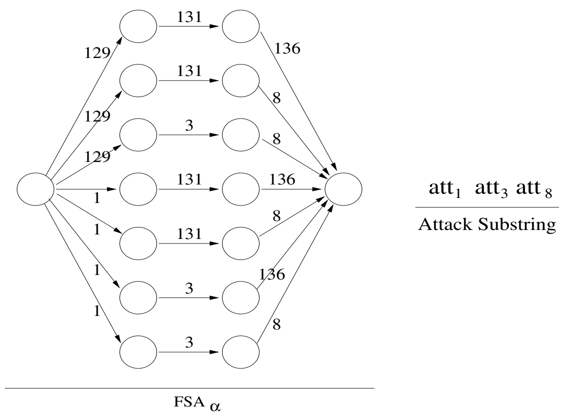


Figure 3: FSA_α and attack substring for clause $x_1 \vee \overline{x_3} \vee x_8$. For convenience, we represent $norm_i$ by just i.

Also, for every variable x_i or $\overline{x_i}$ in a $clause_\alpha$, we have an attack character att_i in the attack code. Thus for every clause $clause_\alpha$, we have a substring (str_α) of length 3 in the attack code. Figure 3 shows an example attack code substring for a hypothetical clause. The encoded attack substring will be $e_{att_1}e_{att_3}e_{att_8}$.

In Figure 3, we can observe that the encrypted str_α is accepted by the FSA_α if and only if the encoding of attack characters are chosen from one of the entries in the given encoding table shown in Table 1. Since every entry in the encoding table corresponds to an entry in the truth table of the $clause_\alpha$, the encrypted str_α is accepted by the FSA_α if and only if $clause_\alpha$ is *true*.

The final FSA, FSA_{SAT}, and the attack code corresponding to the 3-SAT problem are shown in Figure 4. The construction of the above FSA_{SAT} takes polynomial time. Please note that there exists a solution to the given PBA_{sub}^{FSA} problem if and only if the encrypted str_α is accepted by FSA_α for all $1 \leq \alpha \leq r$.

If the above PBA_{sub}^{FSA} problem has a solution mapping e_{att_i}, $0 \leq i \leq m-1$, then one can find assignments for variables $x_i, 0 \leq i \leq 127$ using Equation 1. Since S_{key_ac} is accepted by FSA_{SAT} for mapping $e_{att_i}, 0 \leq i \leq m-1$, the encrypted str_α is accepted by FSA_α for all $1 \leq \alpha \leq r$. However, the encrypted str_α is accepted by FSA_α only if $clause_\alpha$ is *true*. Thus, all clauses of the 3-SAT problem is *true* and the 3-SAT is satisfied.

Also, if there exists an assignment of variables x_i such that the 3-SAT problem is satisfied, then we can compute e_{att_i} using Equation 1. Since 3-SAT is satisfied, all $clause_\alpha$, $1 \leq \alpha \leq r$, are *true*. But $clause_\alpha$ is *true* only if the encrypted str_α is accepted by FSA_α. Thus, all encrypted str_α, $1 \leq \alpha \leq r$, are accepted by FSA_α, and S_{key_ac} is accepted by FSA_{SAT}.

From above, we can conclude that PBA_{sub}^{FSA} is at least as hard as 3-SAT. Since 3-SAT is NP-hard, PBA_{sub}^{FSA} is also NP-hard. Since PBA_{sub}^{FSA} is also in NP, PBA_{sub}^{FSA} is an NP-complete problem. □

4.1.2 XOR Encryption Scheme

We formally define the problem statement of finding a XOR encryption key that ensures S_{key_ac} is accepted by the FSA of an IDS.

DEFINITION 4.2. *Given an attack code and the FSA of an IDS, the problem PBA_{xor}^{FSA} is to find an encryption key, of length l, so that S_{key_ac} is accepted by the given FSA.*

THEOREM 4.2. *Problem PBA_{xor}^{FSA} is NP-complete.*

PROOF. The proof of NP-completeness of PBA_{xor}^{FSA} is similar to the proof of PBA_{sub}^{FSA}. The proof is not provided in the paper due to the space restrictions. □

4.1.3 Corollaries

We have now proved that finding an encryption key that ensures S_{key_ac} is accepted by the FSA of an IDS is NP-complete. Suppose we allow the solution to have $\epsilon\|S_{key\_ac}\|$ number of invalid transitions, the problem still remains NP-hard because of the fact that $(1-\epsilon)$-SAT (or ϵ-UNSAT, $\epsilon < 1$) is an NP-hard problem.

Now consider the problem of finding an encryption key that optimally matches the S_{key_ac} to sFSA. This problem is considered harder than $PBA_{sub/xor}^{FSA}$ because in addition to the requirement of using only valid edges of the sFSA, we need to match the probability of each transition in the sFSA. Thus, finding the suitable encryption key for sFSA should be NP-hard. Following logic similar to above, we can also conclude that finding an encryption key that matches the S_{key_ac} to sFSA within ϵ bound of the optimal solution is also NP-hard.

Substitution and *XOR* are very basic encryption schemes. In fact, the more complex encryption schemes such AES and DES [11] use substitution and *XOR* as basic operations. Since we have shown that the problem is hard when the simpler schemes are in use, we can conclude that the problem is still hard when using the other more complex encryption schemes.

To conclude, we have shown that finding an encryption key that ensures S_{key_ac} is accepted by the (s)FSA of an IDS is a hard problem (NP-complete). The most direct and important corollary is that the problem of generating a polymorphic blending attack is hard. In fact, the problem of determining whether or not a polymorphic blending attack exists is also a hard problem. This follows from the fact that it is NP-hard to verify if a given SAT or ϵ-UNSAT problem has a solution or not.

4.2 Reduction to SAT and ILP

In Section 4.1, we showed that the problem of finding an appropriate encryption key for a polymorphic blending attack is very hard. That is, it may take time exponential to the key length or character size. Although an attacker may not have any time restrictions, a polynomial time solution is clearly more desirable. There are good heuristic solvers available for the SAT problems or Integer Linear Programming (ILP) problems. These solvers provide approximate solutions in very reasonable amount of time. If we have a non-stochastic, or FSA based, IDS, we can reduce the polymorphic blending attack problem to a SAT problem. For a sFSA based IDS, we can map the problem to ILP. Then a heuristic solver can be used to obtain solution for the reduced problem.

Before we show the SAT or ILP reduction, we would reduce the problem of finding an appropriate encryption key to the problem of finding a path from the start state to accept states in a Directed Acyclic Graph (DAG). An edge in the DAG may have constraints of the form $k_i = j$, $j \in U$, where k_i is the ith key and U is the set of all characters. The chosen path should contain a minimal number of conflicting constraints. In addition, we may have some restrictions on the frequency of occurrences of edges corresponding to the frequency matching requirement for sFSA. In the following sections, we use the example shown in Figures 1 and 2 to illustrate the concept.

4.2.1 Construction of DAG

Given a (s)FSA, a key length (l_k), and an attack code ac of length l_{ac}, we construct a DAG of depth $l_k + l_{ac}$ as follows. Suppose s_0 is the end state of the path in (s)FSA as traced by the attack vector and the decryptor. v_0 is the root vertex of the DAG corresponding to state s_0. At every depth d of the DAG, we have a set of vertices $V_d = \{v_{d_i}\}$ such that the state v_i is reachable from state s_0 in exactly d transitions in (s)FSA. The accept vertices of the DAG are the leaf vertices at depth $l_k + l_{ac}$, which correspond to accept states in the (s)FSA. There exists an edge $e_{d_{ij}}$ from v_{d_i} to $v_{(d+1)_j}$ if and only if there exists a transition t_{ij} from state v_i to state v_j in (s)FSA. The weight of the edge is proportional to the probability of transition t_{ij}. The constraint $constr_{d_{ij}}$ associated with an edge $e_{d_{ij}}$ at depth d is shown below.

$$\begin{aligned} constr_{d_{ij}} &= (k_d == char_{ij}), && \text{if } d < l_k, \\ &= (k_{ac[d-l_k]} == char_{ij}), && \text{if } d \geq l_k \text{ and substitution}, \\ &= (k_{(d \bmod l_k)} \oplus ac[d-l_k] == char_{ij}), && \text{if } d \geq l_k \text{ and XOR} \end{aligned}$$

where $char_{ij}$ is the normal character corresponding to transition t_{ij} and $ac[i]$ is the attack character at the i position of attack code. An example construction of DAG is shown in Figure 5. The DAG corresponds to the example FSA and example attack shown in Figure 1 and 2, respectively.

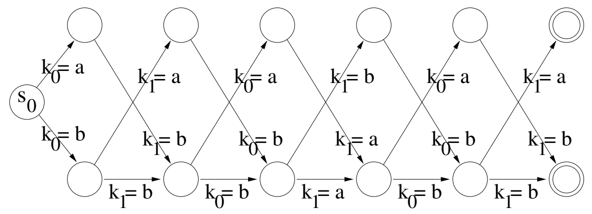


Figure 5: DAG corresponding to example FSA

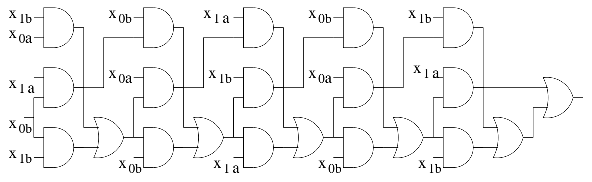


Figure 6: SAT representation of example DAG

The problem of finding an appropriate encryption key for a given attack code and FSA is equivalent to finding a path from the root vertex to an accept vertex in the DAG. Given a path P_{dag} in DAG consisting of edges with no (or minimal) conflicting constraints, we can find the encryption key by setting the constraints of the edges on the path to *true*. The path P_{fsa} followed by S_{key_ac} in the (s)FSA is similar to the path P_{dag}. If $e_{d_{ij}}$ is an edge at depth d in P_{dag} then transition t_{ij} is in P_{fsa} at depth d.

4.2.2 Translation to SAT

If the given IDS is an FSA with no probabilities on edges, the problem of finding a appropriate path in DAG can be translated to SAT. First, we translate the DAG problem to CIRCUIT-SAT [4]. Then we can efficiently translate CIRCUIT-SAT to SAT. For each constraint of the form $k_i = j$ in the DAG, we have a variable x_{ij} that is *true* if and only if the constraint is satisfied, and false otherwise. Now we can directly translate the DAG to CIRCUIT-SAT. A vertex v with input degree deg_{in} in DAG has a corresponding OR gate (OR_v) in CIRCUIT-SAT with deg_{in} inputs. For every outgoing edge (with some constraint $k_i = j$) of a vertex v, we have an AND gate whose input is x_{ij} and OR_v. The final output is OR of all the accept states. Figure 6 shows the conversion of our example DAG to CIRCUIT-SAT. We can then efficiently translate the CIRCUIT-SAT into a SAT problem. In the given SAT problem, we need to add additional requirement that a given key k_i is assigned to only one normal character. This means if x_{ij} is $true$ then $x_{ij'}$ is $false$ for all $j' \neq j$. Also, for substitution based encryption scheme, we need to add additional clauses in SAT to ensure that a normal character is assigned to a single attack character. These cardinality constraints can be efficiently represented in SAT [26]. Furthermore, for a substitution scheme, there exists empty entries in the decryption key table corresponding to the characters not present in the attack code. These characters can be mapped to any unassigned normal character. This requirement can be written as clause $\bigwedge_{j \in N}((\overline{\bigvee_{i \in M} x_{ij}}) \vee (\bigvee_{i \in \overline{M}} x_{ij}))$, where M and N are the set of attack and normal characters, respectively. We can solve the final SAT problem using one of the several available SAT solvers, which are capable of solving huge SAT (or ϵ-UNSAT) problems in reasonable amount of time.

4.2.3 Translation to ILP

For a sFSA IDS, we propose translating the problem of finding a good path in a DAG into an Integer Linear Programming problem. ILP is known to be NP-hard but there exists multiple good heuristics to solve big ILP problems. An ILP tries to find the minimum of a linear function over a set of variables defined by a finite number of linear constraints. All the variables in the solution should be integers. An ILP is called 0-1 ILP if all the variables are required to be either 0 or 1. Now we will show the reduction of finding the optimal path in a DAG to ILP problem.

For every edge $e_{d_{ij}}$ at level d in the DAG, we have a variable $h_{e_{d_{ij}}}$.

$$\begin{aligned} h_{e_{d_{ij}}} &= 1 \text{ if edge } e_{d_{ij}} \text{ is in the } solution\ path \text{ of DAG,} \\ &= 0 \text{ otherwise} \end{aligned}$$

For every constraint $x_i = j$ in the DAG, we have a variable $constr_{ij}$.

$$\begin{aligned} constr_{ij} &= 1 \text{ if constraint } x_i = j \text{ is } true \text{ in the solution of DAG,} \\ &= 0 \text{ otherwise} \end{aligned}$$

There is a valid path from the start vertex to an accept vertex if and only if the number of *selected* outgoing edges at the start state is one, the number of *selected* incoming edges is equal to the number of *selected* outgoing edges for all the intermediate vertices, and the number of *selected* incoming edges is equal to one for one of the end vertices. An edge is *selected* if it is in the solution path. These three conditions can be represented in terms of linear equations as follows:

$$\sum_{e \in OUT(v_0)} h_e + err_{v_0} = 1$$

$$\sum_{e \in IN(V_{accept})} h_e + err_{accept} = 1$$

$$\sum_{e \in IN(v_{di})} h_e + err_{di} = \sum_{e \in OUT(v_{di})} h_e, \forall\ v_{di} \text{ at depth } d, \forall 1 \le d \le l_k + l_{ac} - 1$$

where $IN(v)$ and $OUT(v)$ are the sets of in and out edges of vertex v, respectively. v_{di} is the ith vertex at depth d. The err terms account for invalid paths in the solution and are 0 if the path conditions are satisfied. In case the condition is not satisfied, err_{di} can be either 1 or -1 depending on the difference of the number of *selected* incoming and outgoing edges at the given node.

At any depth $d, 0 \le d \le l_k + l_{ac} - 1$, the number of edges from vertices at depth d to vertices at depth $d+1$ should be one. That is,

$$\sum_{e \in OUT(V_d)} h_e + err_{V_d} = 1, \forall 0 \le d \le l_k + l_{ac} - 1 \tag{2}$$

where V_d is the set of vertices at depth d. Again, the err_{V_d} term accounts for the errors. err_{V_d} can take values 0 or 1 depending on the number of outgoing edges at a given depth.

If an edge in the DAG is chosen in the path, the corresponding constraint should be satisfied. Suppose $constr_e$ represents the constraint associated with edge e. Then the requirement can be satisfied using following equation:

$$constr_e \ge h_e, \forall \text{ edge } e \in \text{ DAG} \tag{3}$$

Further, we can ensure that a given key is assigned to only one character by using following equation:

$$\sum_{j \in U} constr_{ij} = 1, \forall 0 \le i \le l_k \tag{4}$$

where U is the set of all possible characters and l_k is the key length.

For a one-to-one byte substitution scheme, a normal character should not be assigned to multiple attack characters. That is,

$$\sum_{i \in M} constr_{ij} \le 1, \forall j \in N \tag{5}$$

where M and N are the set of attack and normal characters, respectively. The following set of equations ensure that the characters not present in the attack character set are mapped only to normal characters not assigned to attack characters.

$$\begin{aligned} NAC_j \times \|\overline{M}\| &\ge \sum_{i \in \overline{M}} x_{ij}, \forall j \in N, \\ NAC_j + \sum_{i \in M} x_{ij} &\le 1, \forall j \in N \end{aligned}$$

The first equation makes sure that NAC_j is 1 if any non-attack character is mapped to a normal character j. The second equation ensures that if NAC_j is 1, then the normal character j is not assigned to any attack character, and vice-versa.

The above set of equations guarantee that there exists a path from the start vertex to the end vertex with some errors. Now we present the minimization criteria to reduce the errors and the distance of S_{key_ac} from the sFSA. Assume a distance metric of the form:

$$dist = \sum_{transition\ t \in sFSA} \kappa_t \times |p_t - \frac{num_t}{l_k + l_{ac}}| \tag{6}$$

where κ_t is some constant associated with transition t, p_t is the probability of the transition to be taken in sFSA, and num_t is the number of times the transition t is taken by the S_{key_ac}.

The minimization criteria for the ILP problem can be written as:

$$\sum_{transition\ t \in sFSA} const_t \times |p_t - \frac{\sum_d h_{d_t}}{l_k + l_{ac}}| + \sigma \times (\sum_{v \in V_{dag}} |err_v| + \sum_d err_{V_d}) \tag{7}$$

where V_{dag} is the set of vertices in the DAG and σ is the weight of the errors caused by taking a invalid edge in the sFSA. Note that the $|\alpha - \beta|$ term in minimization can be rewritten as abs_{diff} where, $\alpha - \beta \ge -abs_{diff}$ and $\alpha - \beta \le abs_{diff}$.

Solving the above ILP for the given minimization criteria will provide the encryption key. Using this we can generate the encrypted attack code and prepare the polymorphic blending attack packet by concatenating attack vector, decryptor, decryption key, and the encrypted attack code. We can then perform padding to match the final attack packet even closer to the normal, as discussed in Section 3.2.1.

4.3 Heuristic Solutions

Rather than finding the optimal solution, an attacker may simply apply some heuristics that produce an approximate (or good enough) solution using much less resources (time and memory). Here we present a simple heuristic that finds a good approximate solution very efficiently.

The heuristic is based on the *hill climbing* algorithm, which is used widely in artificial intelligence and constraint solving. *Hill climbing* starts with an initial solution and iteratively improves it. At each step, the algorithm looks at the neighboring solutions and chooses one that is better than the current solution. The definition of neighbors depends on the problem domain.

We now present our heuristic. Given an IDS and an attack instance, we choose a random encryption key and calculate the distance between S_{key_ac} and (s)FSA. Now, we randomly choose a k_i to modify in the key. For all the possible character (c) values, we first temporarily assign $k_i = c$. For a substitution scheme, if c is already assigned to some attack character k_j, we temporarily swap the normal characters assigned to k_i and k_j. We then find the new distance to (s)FSA using the temporary key. We choose the character that reduces the distance by the maximum value and assign it to k_i. At this point, a new key position (k_j) is chosen to modify and the process is repeated. This above process is iterated for the desired number of iterations or until a satisfactory solution is produced.

It is possible in the above approach to reach a local maximum that is not very close to the optimal solution. We reach a local maximum if modifying any key increases the distance to (s)FSA. To overcome this problem, whenever we reach local maximum, we choose a small set of key positions and set them to some random values, and restart the above iterative process of finding solution. The idea is by randomly picking another starting point in the solution space, the new solution point may belong to a locale that has better local maximum.

The above heuristic can give us very good solution with sufficient number of iterations.

5. EXPERIMENTS AND RESULTS

The motivation of our research was to address the open problem: given an anomaly detection system and an attack, can one *automatically* generate the PBA instances? Thus, in our experiments, we wanted to directly compare our formal framework with the more ad-hoc approaches developed in our previous work [10]. The key elements of our experimental set-up were the same as in [10]. That is, we used the same anomaly detection systems, namely, PAYL 1-gram and 2-gram, as well as the same attack and same traffic datasets, as in [10]. The results showed that, although our framework is based on an abstract model of IDS and uses general algorithms, it automatically generated PBA instances that were more evasive (i.e. better matched the IDS normal profiles) than or at least as good as the PBA instances from the more PAYL specific algorithms in [10].

Briefly, the experimental dataset contains 15 days of Web traffic with 4.7 million packets. 14 days of traffic were used for training the IDS profile. A part of the remaining 1 day of traffic was used to generate/train an artificial profile used by the attacker. We generated PAYL 1-gram and 2-gram models (using the 14 days of traffic) for three different packet lengths, namely, 418, 730, and 1460. The attack vector is based on the implementation of `firew0rker` [9]. More information of the dataset can be found in [10].

For all the experiments, we divided the attack flow into multiple packets. The attack vector was placed at the start of the first packet. The decryptor was divided into several sections and allotted to different attack packets. The attack body was also divided into multiple chunks. The sFSA corresponding to the artificial profile was adjusted for attack vector and polymorphic decryptor. A separate encryption key for each attack body fragment was generated using our framework to match the adjusted artificial profile. Each attack body fraction was encrypted using the corresponding key and appended to the corresponding attack packet. Then each attack packet was padded to the desired packet length. The final attack packets were then used together to launch an attack.

5.1 PAYL 1-gram Evasion

We applied our framework to generate polymorphic blending attacks to evade 1-gram PAYL, using substitution-based encryption and XOR encryption, respectively. For the XOR scheme, we used a 64 byte key. For each encryption scheme, we translated the problem of finding the optimal encryption key for 1-gram evasion to an ILP problem. We used *ILOG CPLEX* to solve the ILP problems. *CPLEX* is a commercial optimization tool for solving Mixed Integer Programs (MIP). We obtained a near-optimal solution (i.e., encryption key) for the ILP problems. The attack code was then encrypted using this key, and padding was performed. For comparison, we also generated polymorphic blending attacks using the one-to-one local substitution scheme presented in [10].

It took 6.5 seconds on average to solve an ILP problem on a Pentium-M 2GHz machine. The solution provided was within 0.2% of the optimal solution. Figure 7 shows the distance of the attack flow from the artificial profile and the IDS profile. The results for both substitution and XOR encryption schemes, as well as the scheme from [10], are shown in the figure. The x-axis shows the number of packets attack flow was divided into and the y-axis shows distance of the attack flow from the artificial profile and IDS normal profile. This distance is the maximum of the distances of individual attack packets in a flow. A horizontal line corresponding to anomaly error threshold for the 1% IDS false positive is also shown.

The error distance of attacks generated using substitution based encryption with ILP is almost identical to the previous approach from [10]. Thus, the 1-gram blending approach in [10] also provides a near optimal substitution table.

The error distance for attacks generated using the XOR encryption scheme is much higher. This is expected. For substitution, by replacing attack characters with normal characters, we can ensure that only normal characters are present in the mutated attack packet. For *XOR*, it is harder to find an appropriate key such that it contains only normal characters and *XOR*ing it with attack characters also results in only normal characters. For packet length 418 and 730, the error distance of PBA generated using XOR based scheme is twice or more than the substitution-based scheme. For packet length 1460, the error distance for XOR based scheme is comparatively smaller. Also, the difference decreases as the number of attack packets in the attack flow increases. The large amount of padding space available masks the error produced by the attack code in XOR based scheme.

In the plots, all the attack points below the horizontal error threshold line will not be detected by the IDS with a 1% false positive rate. If the false positive rate is decreased, typically the anomaly error threshold is increased. That is, if the horizontal line is moved up, more attack points will be missed by the IDS.

For packet length 730 and 1460, we need only two packets to evade PAYL 1-gram when using substitution. The IDS can be evaded using attack flow of size as low as 1460. For packet length 418, we need 8 packets to evade the IDS. The XOR based scheme can also evade the IDS for packet lengths 730 and 1460, although with bigger attack size. For packet length 418, XOR needs to divide attack packets into many more packets in order to evade the IDS.

5.2 PAYL 2-gram Evasion

We also generated polymorphic blending attacks to evade PAYL 2-gram. We used the heuristic presented in Section 4.3 to generate such attacks. We started with a random solution and iterated the hill climbing steps 25000 times. The best encryption key seen during the process was recorded. The attack code was then encrypted using this key and the attack packet was padded to the desired length. The distance of the attack flow from the normal profiles was recorded. For comparison, we also generated the polymorphic blending attacks using the 2-gram blending algorithm presented in [10].

For substitution based encryption, it took $10min$ on average to perform 25000 iterations on a given problem. For XOR encryption, performing 25000 iterations took little more than an hour on average. The time of each iteration is dependent on the range of keys and the number of terms in the distance calculation. For substitution, the range of keys is all the normal characters; whereas

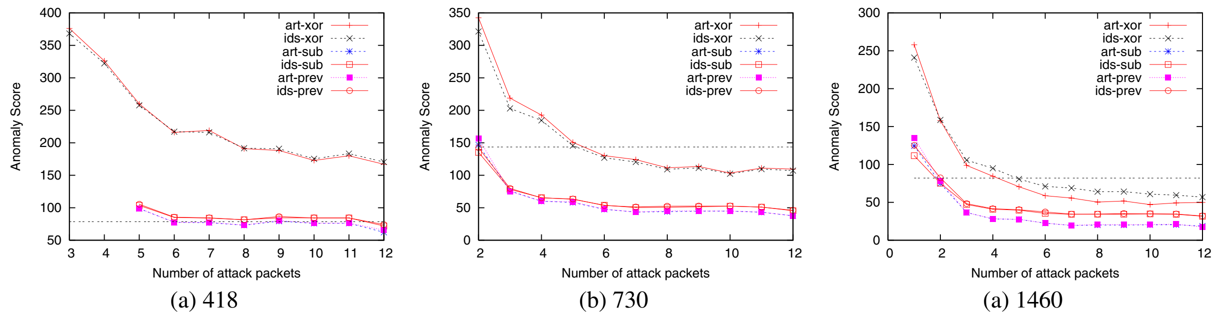


Figure 7: Anomaly score or error distance of 1-gram blending attack. The plots with prefix *art* and *ids* corresponds to distance from the artificial profile and the IDS profile, respectively. *xor* and *sub* corresponds to the PBA generated for XOR and substitution based schemes using our framework. *prev* denotes the algorithm from previous paper.

in XOR, the range of keys can be the set of all the possible characters. Also, since XOR is not able to match the normal profiles closely, there are several new 2-grams in the attack packet and thus the number of terms in the distance calculation may be large. These two reasons account for the long running time for XOR.

The distances of the attack packets from the normal profiles are shown in Figure 8. The results for substitution-based encryption and XOR encryption, as well as the scheme from [10], are shown, along with a horizontal line corresponding to error threshold of the 1% IDS false positive rate.

Using substitution, our heuristic-based approach is able to better match the normal profile than the previous approach [10] for most attack instances. For packet length 418, when the number of attack packets is 8 or 9, the previous approach matches the normal profile better than the heuristic from our framework. Checking the distance of individual attack packets, we observed that for some packets, the heuristic got stuck in a bad local maximum for considerable number of iterations. Thus, the heuristic was not able to find a good solution in the given number of iterations. In such cases, one can restart the heuristic using another random solution or run the heuristic for more number of iterations.

Compared with PAYL 1-gram, we needed a larger number of attack packets to evade PAYL 2-gram. The minimum attack size required to evade the PAYL 2-gram is 2190, or 3 packets of length 730.

6. HARDENING THE IDS AGAINST POLYMORPHIC BLENDING ATTACKS

Our framework can be used to improve an stochastic IDS to make it harder for an attacker to evade the IDS. The basic idea is to find transitions (or edges) in the sFSA that are traversed frequently by multiple polymorphic blending attacks. If an edge contributes only a small error term in computing the distance of PBAs to normal profile but contributes a large error term for normal packets, we can exclude the edge when calculating the distance of a monitored packet from the normal file. The result will be increased detection rate against attacks, including the PBAs, with maybe only slightly increased false alarm rate.

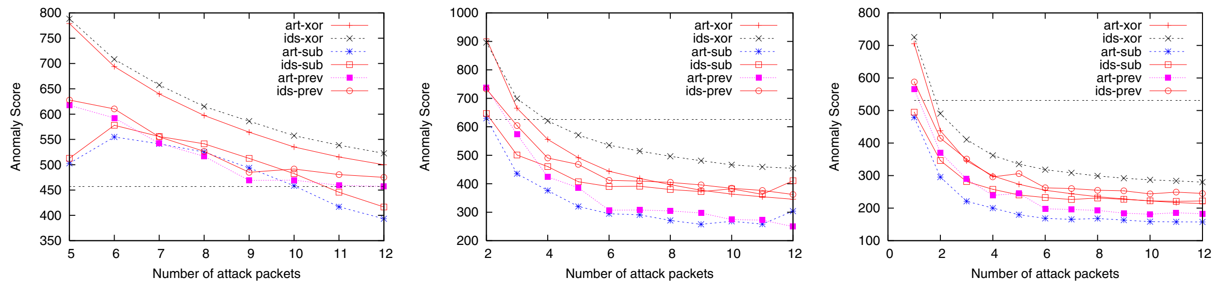
We now explain in more details how to improve the IDS against PBA. First, we collect multiple attack instances for different known attacks on the system. For each of these instances, we generate multiple polymorphic blending attacks using techniques described in Section 4.2. For all of these attacks pba_i, we find paths $path_i$ taken by pba_i in the sFSA. Suppose the number of times an edge e_j is present in the $path_i$ is denoted by n_{ji} and their contributing term in distance calculation is $dist_{ji}$. Then the relative contribution of edge e_j in the distance between polymorphic blending attacks and sFSA is calculated as $pba\_rel\_dist_{e_j} = \frac{\sum_{pba_i} dist_{ji}}{\sum_{e_k \in E}(\sum_{pba_i} dist_{ki})}$, where E is the set of all edges in the sFSA. Similarly, we calculate the relative contribution of edge e_j in the distance between normal packets and sFSA as $nor\_rel\_dist_{e_j} = \frac{\sum_{nor_i \in T} dist_{ji}}{\sum_{e_k \in E}(\sum_{nor_i \in T} dist_{ki})}$, where T is the set of training data packets.

We would like to exclude the edges that have a high value of $\frac{nor\_rel\_dist_{e_j}}{pba\_rel\_dist_{e_j}}$ and that occur with high frequency in PBAs. Let $deg_{e_j} = \frac{nor\_rel\_dist_{e_j}}{pba\_rel\_dist_{e_j}} \times pba\_p_{e_j}$, where $pba\_p_{e_j}$ is the probability of an edge e_j being present in a PBA. deg_{e_j} denotes the degree with which an edge e_j is assisting the attacker in evading the IDS. We exclude the edges that have the higher values of deg_{e_j}. Note that we are not actually removing the edge from the sFSA. Rather, by excluding the distance term corresponding to edge e_j, we are marginally reducing the distance of blending attack packets from the IDS but significantly reducing the distance of normal packets from the IDS. Thus, the error distance threshold of the IDS can be reduced while keeping the false positive rate almost same and increasing the detection rate for a PBA. If an attacker launches a new PBA against the IDS using a new exploit, the IDS will have a high probability of detecting the new attack if it takes the similar path (in sFSA) as the attacks pba_i. We plan to perform more detailed analysis of above proposed technique.

7. CONCLUSION

In this paper, we presented a formal framework for polymorphic blending attacks. We modeled a variety of IDSs as either FSA or sFSA. We then reduced the problem of finding an optimal PBA that matches the IDS normal profile to the problem of finding an encryption key that optimally matches the string representing *the decryption key concatenated with the encrypted attack code* to the (s)FSA of the IDS. We showed that finding such an encryption key is an NP-complete problem. We presented some techniques to reduce this problem to a satisfiability problem and an nteger linear programming problem. Thus, optimization algorithms available for these problem domains can be used to generate a near-optimal encryption key and hence, a near-optimal PBA. We also proposed a heuristic that can be used to find good approximate encryption keys very efficiently. We validated our framework using PAYL 1-gram and 2-gram. We also proposed a technique to improve the performance of an IDS against PBAs.

The results of our experiments showed that our framework can automatically generate PBAs that can evade an IDS. The PBAs



(a) 418 (b) 730 (a) 1460

Figure 8: Anomaly scores of 2-gram blending attacks.

generated by our framework were able to match the normal profile more closely than the PBAs produced by the previously proposed IDS specific algorithms. The time required to solve the ILP problem to generate a PBA to evade PAYL 1-gram was only a few seconds. Generating attack packets to evade PAYL 2-gram took several minutes. A substitution-based encryption scheme was shown to be more effective than *XOR* encryption for evading an IDS.

A polymorphic blending attack is also achievable using other shellcode transformation techniques, for e.g., equivalent instruction substitution and garbage insertions. We plan to further study polymorphic blending attacks by incorporating different attack mutation techniques, metamorphism and code obfuscation. Our current framework focuses on algorithms for generating the (optimal) encryption key. We plan to extend it to automatically determine the best mutation techniques as well the optimal padding bytes.

Acknowledgements

This work is supported in part by NSF grant CCR-0133629 and Office of Naval Research grant N000140410735. The contents of this work are solely the responsibility of the authors and do not necessarily represent the official views of NSF and the U.S. Navy. The authors would like to thank Dr. Emilie Danna for her valuable suggestions.

8. REFERENCES

[1] P. Akritidis, E. P. Markatos, M. Polychronakis, and K. Anagnostakis. Stride: Polymorphic sled detection through instruction sequence analysis. *In 20th IFIP International Information Security Conference*, 2005.

[2] M. Barreno, B. Nelson, R. Sears, A. D. Joseph, and J. D. Tygar. Can machine learning be secure? *In Proceedings of the ACM Symposium on Information, Computer, and Communication Security (ASIACCS)*, 2006.

[3] M. Christodorescu, S. Jha, S. Seshia, D. Song, and R. Bryant. Semantics-aware malware detection. *In Proceedings of the IEEE Symposium on Security and Privacy*, 2005.

[4] T. H. Cormen, C. E. Leiserson, and R. L. Rivest. Introduction to algorithms. *The MIT Press/McGraw-Hill*, 1990.

[5] T. Detristan, T. Ulenspiegel, Y. Malcom, and M. Underduk. Polymorphic shellcode engine using spectrum analysis. *Phrack Issue 0x3d*, 2003.

[6] S. T. Eckmann, G. Vigna, and R. A. Kemmerer. Statl: An attack language for state-based intrusion detection. *JOURNAL OF COMPUTER SECURITY*, 10:71–104, 2002.

[7] H. Feng, J. Giffin, Y. Huang, S. Jha, W. Lee, and B. Miller. Formalizing sensitivity in static analysis for intrusion detection. *In Proceedings of the IEEE Symposium on Security and Privacy*, 2004.

[8] H. Feng, O. Kolesnikov, P. Fogla, W. Lee, and W. Gong. Anomaly detection using call stack information. *In Proceedings of the IEEE Symposium on Security and Privacy*, 2003.

[9] Firew0rker. Windows media services remote command execution exploit. *http://www.k-otik.com/exploits/07.01.nsiilog-titbit.cpp.php*, 2003.

[10] P. Fogla, M. Sharif, R. Perdisci, O. M. Kolesnikov, and W. Lee. Polymorphic blending attacks. *In 15th USENIX Security Symposium*, 2006.

[11] C. Kaufman, R. Perlman, and M. Speciner. Network security: Private communication in a public world. *Prentice Hall*, 2002.

[12] C. Kruegel, E. Kirda, D. Mutz, W. Robertson, and G. Vigna. Automating mimicry attacks using static binary analysis. *In 14th Usenix Security Symposium*, 2005.

[13] C. Kruegel, E. Kirda, D. Mutz, W. Robertson, and G. Vigna. Polymorphic worm detection using structural information of executables. *In Recent Advances in Intrusion Detection (RAID)*, 2005.

[14] C. Kruegel, T. Toth, and E. Kirda. Service specific anomaly detection for network intrusion detection. *In Proceedings of the ACM SIGSAC*, 2002.

[15] C. Kruegel and G. Vigna. Anomaly detection of web-based attacks. *In Proceedings of the ACM Conference on Computer and Communication Security (ACM CCS)*, pages 251–261, 2003.

[16] Ktwo. Admmutate: Shellcode mutation engine. *http://www.ktwo.ca/ADMmutate-0.8.4.tar.gz*, 2001.

[17] Z. Liang and R. Sekar. Fast and automated generation of attack signatures: a basis for building self-protecting servers. *Proceedings of the 12th ACM Conference on Computer and Communications Security (ACM CCS)*, pages 213 – 222, 2005.

[18] M. Mahoney. Network traffic anomaly detection based on packet bytes. *In Proceedings of the ACM SIGSAC*, 2003.

[19] M. Mahoney and P.K. Chan. Learning nonstationary models of normal network traffic for detecting novel attacks. *In Proceedings of the SIGKDD*, 2002.

[20] J. Newsome, B. Karp, and D. Song. Polygraph: Automatically generating signatures for polymorphic worms. *In Proceedings of the IEEE Symposium on Security and Privacy*, 2005.

[21] Martin Roesch. Snort-lightweight intrusion detection for networks. *In Proceedings of the 13th USENIX conference on System administration*, pages 229 – 238, 1999.

[22] S. Rubin, S. Jha, and B. P. Miller. Language-based generation and evaluation of nids signatures. *In Proceedings of the IEEE Symposium on Security and Privacy*, 2005.

[23] S. Rubin, S. Jha, and B.P. Miller. Automatic generation and analysis of nids attacks. *In Annual Computer Security Applications Conference (ACSAC)*, 2004.

[24] R. Sekar, M. Bendre, D. Dhurjati, and P. Bollineni. A fast automaton-based method for detecting anomalous program behaviors. *In Proceedings of the IEEE Symposium on Security and Privacy*, 2001.

[25] R. Sekar, A. Gupta, J. Frullo, T. Shanbhag, A. Tiwari, H. Yang, and S. Zhou. Specification-based anomaly detection: A new approach for detecting network intrusions. *In Proceedings of the ACM conference on Computer and communications security (ACM CCS)*, 2002.

[26] C. Sinz. Towards an optimal cnf encoding of boolean cardinality constraints. *In Principles and Practice of Constraint Programming*, pages 827–831, 2005.

[27] P. Szor. Advanced code evolution techniques and computer virus generator kits. *The Art of Computer Virus Research and Defense*, 2005.

[28] T. Toth and C. Kruegel. Accurate buffer overflow detection via abstract payload execution. *In Recent Advances in Intrusion Detection (RAID)*, 2002.

[29] G. Vigna, W. Robertson, and D. Balzarotti. Testing network-based intrusion detection signatures using mutant exploits. *In Proceedings of the ACM Conference on Computer and Communication Security (ACM CCS)*, pages 21–30, 2004.

[30] D. Wagner and P. Soto. Mimicry attacks on host-based intrusion detection systems. *In Proceedings of the ACM Conference on Computer and Communication Security (ACM CCS)*, 2002.

[31] K. Wang and S. Stolfo. Anomalous payload-based network intrusion detection. *In Recent Advances in Intrusion Detection (RAID)*, 2004.

[32] K. Wang and S. Stolfo. Anomalous payload-based worm detection and signature generation. *In Recent Advances in Intrusion Detection (RAID)*, 2005.

[33] T. Yetiser. Polymorphic viruses: Implementation, detection, and protection. *Technical Report, VDS Advanced Research Group*, 1993.