2009 29th IEEE International Conference on Distributed Computing Systems

Characterization and Solution to A Stateful IDS Evasion

Issam Aib, Tung Tran, and Raouf Boutaba
University of Waterloo, Waterloo, ON, Canada
(iaib, t3tran, rboutaba)@uwaterloo.ca

Abstract

We identify a new type of stateful IDS evasion, named signature evasion. We formalize the signature evasion on those Stateful IDSs whose state can be modeled using Deterministic Finite State Automata (DFAs). We develop an efficient algorithm which operates on rule set DFAs and derives a minimal rectification of evasive paths. Finally, we evaluate our solution on Snort signatures, identify and rectify existing vulnerable flowbit rule sets [1]

1. Introduction

Stateful Intrusion Detection Systems (IDSs) use stateful signatures to simulate the behavior of the application protocol they are protecting and to identify malicious behavior. However, due to complexity and overhead reasons, it is difficult to fully simulate every session state in an IDS. This leads to a class of signature-based IDS evasion that exploits possible differences between the IDS session state and actual (application) session state making it susceptible to false negatives. This paper identifies this type of evasion, named *signature evasions*. The proposed evasion assumes knowledge of IDS rules. However, if rules are not available, reverse engineering of network signatures techniques [1] can always be used.

To illustrate how a signature evasion is carried out, we provide an example of such an evasion on the Snort [2] rule set in Table 1. The rule set follows an FTP session. It raises an alert if a non-admin user tries to do anything related to an important file which should only be accessed by the Admin. Rules r_1 and r_2 follow the login process of a non-admin user, r_3 signals a denied login, r_4 detects a successful login, r_5 indicates that the user has logged out of the FTP session, and r_6 raises an alert when a logged-in non-admin user tries to do anything with the restricted file. Rules r_7, r_8 and r_9 handle the case of an admin user log in. In the right most column we give a symbol to each different signature used in the rule set.

Snort *flowbits* offer the stateful property to the rules and allow the detection engine to track state across multiple packets in a single session. A flowbit is a boolean flag that can be set (1) or unset (0) by a rule. Rules without the "flowbits: noalert" option are alert rules (r_6).

Deriving a DFA representation of a Snort flowbits rule set is relatively straightforward. Fig.1 shows the DFA of the FTP rule set of Table 1. The signature alphabet of the FTP rule set is $\Sigma = \{a, b, c, d, e, f, g\}$.

We define an *evasive signature* as a packet which triggers an IDS rule (Snort in our example) while in reality it has no effect on the actual session state at the server side. For each signature s^r of a rule r, it is ideally possible to divide it into two subsignatures $s^r = s^r_+ + s^r_*$, where s^r_* represents all evasive signatures and s^r_+ the set of correct signatures.

Let p^s denote a packet that matches signature s. $abde_*f$ is an example of a successful signature evasion sequence to the FTP rule set. The attacker can apply this evasion sequence to perform a real attack as follows: First, he logs in as a normal user (non admin) with a correct username and password. This action requires the sending of p^a and p^b from the attacker side and leads to the sending of p^d from the server to indicate that the user is successfully authorized. At this phase, the Snort DFA reaches state 3 (Fig.1). In the next step, the attacker tries to fabricate a p^e_* packet that triggers r_5 at Snort taking it to state 0 but does not do so at the server side (which remains at state 3). In order to do this, the attacker sends a packet that matches r_5 but actually does something else rather than exiting the session as Snort thinks. The attacker can do that, for example, by creating a directory named "QUIT". After that, he can perform the last step of the attack of accessing the restricted file at the server, *i.e.*, issues a p^f packet. This last action will not trigger the target rule r_6 and hence successfully evades Snort. It can also be noticed that many other signature evasions are also possible, such as g_*abdf (evasion through state 5), abc_*g_*abdf, and ab_*cbde_*f.

We formalize the signature evasion problem in Sec.2 and develop a formal characterization of evasive paths in Sec. 3. Sec. 4 provides an optimal rectification of signature evasions based on the direct manipulation of a rule set DFA. Signature evasion vulnerabilities are detected in a number of existing Snort flowbit rules and the results are presented in Sec.5. We then summarize related work and conclude.

1. This research was partially supported by NSERC STPG 322235_05 and WCU Project R31-2008-000-10100-0.

1063-6927/09 $25.00 © 2009 IEEE
DOI 10.1109/ICDCS.2009.65

Table 1. FTP login rule set

r_1	msg: “FTP Normal User Login Attempt - Send username”; flow: established, to_server; content: “USER”; depth:5; nocase; content:!“admin”; within:5; content: “|0D0A|”; flowbits: set, ULA; flowbits: noalert;	a
r_2	msg: “FTP Normal User Login Attempt - Send password”; flow: established, to_server; content: “PASS”; depth:5; flowbits: isset, ULA; flowbits: set, ULA2; flowbits: noalert;	b
r_3	msg: “FTP login denied”; flow: established, to_client; flags:A; flowbits: isnotset, UL; flowbits: isset, ULA; flowbits: isset, ULA2; flowbits: unset, ULA; flowbits: unset, Normal UserLoginAttempt2; flowbits: noalert;	c
r_4	msg: “FTP login granted”; flow: established, to_client; content: “230 Login successful.|0D0A|“; nocase; flags: AP; flowbits: isset, NormalUserLoginAttept2; flowbits: set, UL; flowbits: noalert;	d
r_5	msg: “FTP user exits”; flow: established, to_server; content: “|QUIT|0D0A|”; nocase; flowbits: isset, UL; flowbits: unset, UL; flowbits: unset, ULA; flowbits: unset, ULA2; flowbits: noalert;	e
r_6	msg: “Normal User accesses important file”; flow:established, to_client; content: “Windows”; content: “system32”; content: ”sam”; nocase; flowbits: isset, UL;	f
r_7	msg: “FTP Admin Login Attempt - Send username”; flow: established, to_server; content: “USER”; depth:5; nocase; content:‘ ‘admin”; within:5; content: “|0D0A|”; flowbits: set, ALA; flowbits: noalert	g
r_8	msg: “FTP login denied”; flow:established, to_client; flags:A; flowbits: isset, ALA; flowbits: unset, ALA; flowbits: noalert;	c
r_9	msg: “FTP Admin exits”; flow:established, to_server; content: “|QUIT|0D0A|“; nocase; flowbits: isset, ALA; flowbits: unset, ALA; flowbits:noalert;	e

2. Problem Definition

Let $\mathcal{R}$ be a signature rule set. A rule of $\mathcal{R}$ that generates an alert is referred to as a *target rule*. A rule is *evadable* if its signature can be triggered by a fake (fabricated) packet. The set of all target rules in a rule set is called the *target rules set*. A *target rule group* is a group of target rules that generate the same type of alert (these rules may possibly identify different signatures).

Let g_a be a target rule group of $\mathcal{R}$ which detects attack a. An a-attack is a flow of packets which trigger an alert rule of g_a.

Definition 2.1 (Target Rule Group Evasion): A target rule evasion on g_a, or a-evasion, is an a-attack camouflaged by sequences of evasive signatures resulting in the non triggering of any of the rules of g_a.

A target rule evasion may be an evasion of the entire rule set if it does not trigger any other alert rule. However, this is not a mandatory requirement. Some target rule evasions fall within the general category of evasions which trigger a different set of alerts than those which the attack actually does. This type of evasion is tricky as it diverts the attention of the security administrator by providing a wrong diagnosis of the problem, hence giving more time to the attacker to accomplish his goal. The algorithm we provide in this paper is also able to stop these evasions.

Let $D = (Q, \Sigma, \delta, n^0, F)$ be a DFA of $\mathcal{R}$, such as the one in Fig.1. Crossing a transition in D is equivalent to the triggering of a rule of $\mathcal{R}$ which may change the DFA state as well as generate output (log, warning, alert). More accurately, D is a Mealey FSM (Finite State Automaton) since every rule is allowed to generate output. In addition, a flow is accepted by D only if at least one target rule of $\mathcal{R}$ is triggered, *i.e.*, one *target transition*, and it is not important after that in which state the last packet of the flow will bring the DFA in. We use a simple algorithm to convert each signature rule set DFA to a DFA where each target state accepts only transitions that generate alerts related to exactly one target rule group.

Definition 2.2 (A_*/A_+ Notation): Let A =$\{ s^1, \ldots, s^n\}$ be a set of signatures. A_* denotes the subset of A formed only by evasive signatures, i.e. $A_* = \{s^i_*|s^i \in A \wedge s^i_* \neq \phi\}$. Similarly, $A_+ = \{s^i_+|s^i \in A\}$. This notation naturally extends to DFA paths.

Definition 2.3 ($\oplus$ operator): Let n be a node of DFA D and p a string of signatures. $n' = n \oplus_D p$ denotes the node of D reachable from n using p. The D subscript is omitted if it is known from the context.

Definition 2.4 ($\ominus$ operator): Let p be a string of signatures belonging to signature alphabet $\Sigma = A \cup B$, where $A \cap B = \phi$. The string $p' = p \ominus B$ is formed by systematically deleting every element of B appearing in p. We also write $p = \ominus_B(p')$ or $p' \in \ominus_B^{-1}(p)$.

Corollary 1 (Characterization of Evasive attacks): Let p_+ be an r-attack detectable by final states F_r of DFA D, i.e. $n^0 \oplus p \in F_r$. An evasion of p is an attack with path p' such that $n^0 \oplus p' \notin F_r$ and $p' \ominus \Sigma_* = p_+$, where Σ is the alphabet of signatures of D.

Definition 2.5 (D^+ Notation): The false positive closure D^+ of a rule set DFA D is a DFA with the same detection capability of D without the negative effect of generating false positives.

Fig.2 shows the D^+ of the FTP rule set (Table 1). For exam-

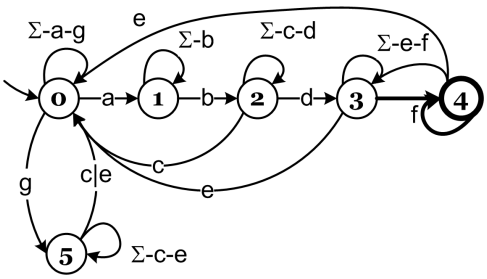

Figure 1. DFA of the FTP rule set

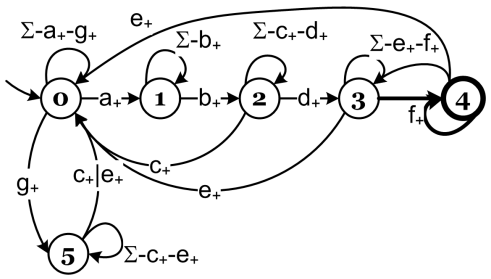

Figure 2. False Positive Closure of the FTP DFA

ple, this DFA does not consider the evasive path $a_*b_*d_*f_*$ an attack while D does (Fig.1). D^+ never generates false alarms due to evasive packets. Hence, it can be used to model the actual session state of the service that the rule set protects, so we will also refer to it as the actual session state DFA.

Definition 2.6 (D° Notation): D° denotes the NFA obtained from D by making its final states the new set of start states and all the states final ($F = Q$).

Let $\mathcal{L}_{std}$ denote the usual function which returns the language recognized by a DFA (all input which ends at a final state).

Lemma 1 (Language of a rule set DFA): Let $\mathcal{L}(D)$ denote the language of a rule set DFA D: *i.e.*, the set of all flows that trigger a target rule of $\mathcal{R}$. We have:

$$\mathcal{L}(D) = \mathcal{L}_{std}(DD^\circ) \tag{1}$$

Next, we identify the language of all signature evasions that can be applied on D.

Theorem 1 (DFA of all signature evasions): Let F_r be the set of target states of D related to target rule group $g_r \subset \mathcal{R}$. Let $D_r = (Q, \Sigma, \delta, n^0, F_r)$ be the DFA obtained from D by keeping only F_r as target states. Let D_r^+ be the false positive closure of D_r. Let $D_r^\circ = (Q, \Sigma, \delta, F_r, Q)$ be the NFA obtained by making F_r the new start states and all states in Q target states. Let $\neg D_{-r} = (Q - F_r, \Sigma - r, \delta - r, n^0, Q - F_r)$ be the DFA obtained from D_r by removing all states $\in F_r$ and making all the other states final. The DFA D_r^e which generates all successful r-evasions is then:

$$D_r^e = \neg D_{-r} \cap (D_r^+ D_r^{+,\circ}) \tag{2}$$

3. Characterization of Evasion Paths

This section characterizes the form of an evasion path. It also identifies a theorem which will be used in the next section to derive a polytime solution to the signature evasion problem.

Definition 3.1 (Trail DFA): It is a DFA of m nodes $\{n_i | 0 \leq i \leq m\}$ ordered from n_0 to n_m where: *(1)* There is a trail which spans all the nodes of the DFA. That is, $\forall i, n_{i+1} \in n_i^+$ (n_{i+1} reachable from n_i in one hop), *(2)* All paths of D respect the initial ordering of the nodes. That is, $\forall i,\ n_i^+ \subseteq \{n_j | m \geq j \geq i\}$, and *(3)* All final nodes are articulations of the DFA (cut nodes). That is, if $n_k \in F$, then D has no transition (n_i, s, n_j) such that $i < k < j$.

Definition 3.2 (Disjoint-Hop Trail DFA): is a trail DFA with the additional condition that if $\exists i, j | j > i + 1 \wedge (n_i, s, n_j) \in \delta$ then the DFA has no transition (n_k, s, n_l) where $i \leq k < l < j$.

Theorem 2: Disjoint-Hop Trails are unevadable.

Theorem 3 (Minimal Evadable Paths): If p is an evadable attack path of D then the attack simple path p_{min} derived from p is also evadable.

Corollary 2: To determine if an alert rule r is evadable, it is sufficient to consider only the simple paths to the set of its final nodes F_r.

This result tells us that if we manage to prove that no simple path to an alert node is evadable then any evasion to that node is impossible.

Lemma 2 (Fork Property of Evasion Paths): Let r be an evadable alert rule of $\mathcal{R}$ and F_r the set of final states of D corresponding to r. Let p be a simple attack path which triggers r. If q is an evasion of p (hence of r) then q forks with p at a node a where the prefix sub-paths of p and q from the start state to a form a disjoint hop trail (*e.g.*, Fig.5).

Theorem 4 (Characteristic Evasion Path): Let p be an evadable attack simple path. Let q be an evasion of p which forks with it at a node a with a transition t as specified by Lem.2, such that (Fig.3): $p_+ = p_+^0 s_+ p_+^1$, $q = q^0 t q^1$, $t \neq n_0$, $n_0 \oplus p_+^0 = n_0 \oplus q^0 = a$, $a \oplus s = b$, $a \neq b$, $a \oplus t = c$, $a \neq c$, $b \neq c$, $q \ominus \Sigma_* = p_+$, then:

1 – The transition in $(d, s, e) \in q$ which corresponds to the s_+ in (a, s, b) has to occur after node c, i.e. $\in q^1$ (Fig.3, 4) where $e \neq b$.

2 – There exists another evasion q' of p, which starts by prefix p_+^0, crosses transition (a, t, c) with evasion t_* and proceeds with an all evasive input $q_{1,*}^1$ until it reaches d, *i.e.*, $q' = p_+^0 t_* q_{1,*}^1 s_+ q_2^1$ (Fig.5).

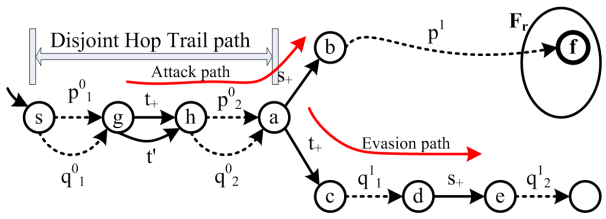

Figure 3. Possible evasion path

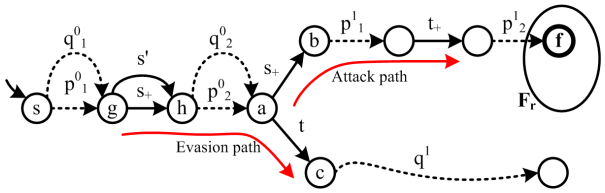

Figure 4. Impossible evasion path

Applied to the DFA of Fig.6 where a corresponds to state 2 and s to transition b, it can be noticed that an evasion is possible through transition e since $(5, b, 5) \in \delta$. An example of such an evasion is ae_*bc.

In what follows the use of a, b, c, d, e, s will be based on their meaning as used in Thm.4.

The strength of the characteristic evasion path theorem lies in three aspects. First, it indicates that, to check for evadability, only fork nodes need investigation. Second, the prefix to a fork node can be ignored and the only concern is whether the transition at the fork node leading to a final state is also reachable from the other side of the fork. What is finally important to look at is whether from e it is possible to generate a path which evades the path from b to a target state. We will use this observation to identify a minimal rectification of a rule set DFA against signature evasions.

4. Resolution of Signature Evasions

Based on Thm.4, it can be determined that rule r which corresponds to the final nodes F_r of Fig.5 is evadable. To rectify this, a *patch* can be applied by adding transition (d, s, b) to D as shown in the figure. We name this procedure *stitching based on the characteristic evasion path theorem*. Using this stitching, any evasion q' (ref.Thm.4) becomes detectable as it now leads the stitched automaton to a state in F_r. Since any evasion path q implies the existence of a q' as indicated by the theorem, then the non existence of any successful q' in the stitched automaton implies that all the corresponding q evasions are no longer possible. By applying this on all cases where the stitching is feasible, it can be determined that all possible evasions are no longer possible.

As an example, the application of this stitching method on the automaton of Fig.6(a) results in the signature evasion safe automaton of Fig.6(c). It can be noticed that the $(5, b, 3)$ stitching transition (Fig.6(b)) is able to stop evasion ae_*bc.

The stitching we employ is *minimal* in the number of stitching transitions to add to an automaton. This is because if the stitchings were to be carried beyond node e then usually more than one transition (and at least one) will be required.

When an alert is triggered by the stitched DFA, there is still a possibility of it being a false alarm. To explain that, let's consider the notation:

Definition 4.1 ($\mathcal{E}(p)$ Notation): If $p = s^1 s^2 \dots s^n$ is a signature string then

$$\mathcal{E}(p) = \{q^1 q^2 \dots q^n | q^i \in \{s^i_+, s^i_*\}\}$$

Note that, by definition, all flows in $\mathcal{E}(p)$ are considered by the corresponding IDS rule set as the same flow since by definition it does not differentiate an s^i_+ from an s^i_*. In this regard, if q is an evasion path detected by the stitched DFA, then any flow in $\mathcal{E}(q)$ will also raise an alert even it is not actually an evasion. This in fact is also true for normal attacks that are detectable by the original rule set. This type of false positives is always inevitable.

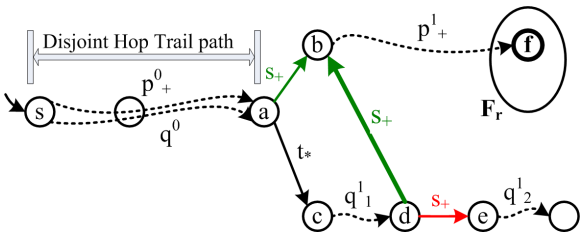

Figure 5. Characteristic Evasion Path

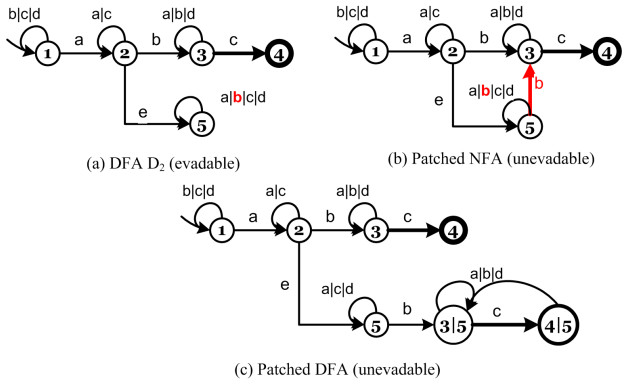

Figure 6. Application of Minimal Stitching

Algorithm 1 edgeSet minStitch (DFA D, NFA safeD)

Let $D = (Q, \Sigma, \delta, n^0, F = \cup_{r \in R} F_r)$, and $D^+ = (Q, \Sigma, \delta^+, n^0, F)$ its positive closure.
safeD = D.clone();
edgeSet minStitch = ϕ;
for all target rule group F_r **do**
  for all fork node $a \notin F_r$ **do**
    // $suc^*(b)$ =set of all nodes reachable from b
    for all $b \in suc(a) \mid suc^*(b) \cap F_r \neq \phi$ **do**
      Let (a, S, b) be the set of transitions from a to b
      for all fork tuple (a, b, c) **do**
        // nodes(s) =$\{n | s$ is outgoing from $n\}$
        for all $(d, s, e) \in \delta | d \in ((nodes(s) - F_r - \{a\}) \cap suc^*(c)) \wedge (e \notin F_r \cap b)$ **do**
          // $\mathcal{E}_r^{b,e}$ defined in Thm.5
          if $\mathcal{E}_r^{b,e} \neq \Phi$ **then**
            // there is an evasion suffix from e
            minStitch $+= \{(d, s, e)\}$
            safeD += {(d, s, e)} ;
          end if
        end for
      end for
    end for
  end for
end for
return minStitch

The next step would be to find an efficient way to identify the states d to apply the stitching on. To do so, we will use Thm.5, which provides a straightforward procedure to identify if an evasion can successfully proceed from state e (Thm.4).

Theorem 5: Let $\mathcal{E}_r^{b,e}$ be the set of all suffixes q_2^1 of r-evasions q (ref. Thm.4). Let D_r^{+b} be the DFA formed from D^+ by making b a start state and only $F_r \subset F$ as final states. Let $\neg D_{-r}^e$ be the DFA formed from D_r by making e a start state, then removing all final states F_r, then making all the other states final. We have:

$$\mathcal{E}_r^{b,e} = \neg D_{-r}^e \cap (D_r^{+b} D_r^{+\circ}) \quad (3)$$

At this point, we can present Alg.1 which applies minimal stitching using the result of Thm.5. For each target rule group F_r it does the following: For each fork tuple (a, b, c) where F_r is reachable from b, it checks whether there is a d reachable from c ($d \in suc^*(c)$) where $(d, s, e) \in \delta$ and $\mathcal{E}_r^{b,e} \neq \Phi$. If this is the case then a stitch is required at node d.

Lemma 3: Alg.1 generates a minimal rectification of signature evasions and has a polytime complexity.

5. Evaluation

We collected 400 publicly available flowbit rule sets (most of them from BleedingEdge [3] and SourceFire [4]). We implemented a parser that generates a rule set DFA and runs Alg.1. In case necessary stitches are found, a safe stitched NFA is generated as well as the corresponding safe (non vulnerable) DFA. 4.5% of the flowbit rule sets were found vulnerable to the proposed attack when all rules are considered evadable (for all signature s, $s_* \neq \phi$). Because of its polytime complexity, the code runs in few minutes over all rule sets except for rule set 6 which took about two hours.

It can be noticed from Table 2 that most of the vulnerable rule sets require a digit number of stitches to become safe for the exception of four of them. This first indicates that most of the times our stitching method generates DFAs which can be easily converted to a safe rule set. For example, the DFA of the 10^{th} flowbit rule set is shown in Fig.8. It has five states, an alphabet of three different signatures, one fork state, and only requires one stitch. The resulting NFA is given in Fig.8 and the corresponding safe DFA in Fig.9. The tool we developed automatically generates DFAs/NFAs in the standard DOT format [5] for easy visualization.

There are however four flowbit rule sets for which more than 10 stitches are required. For rule set 12, this is rather good as the overall stitching results in a reduced DFA size. Hence, only three rule sets pose problem: 6, 7, and 18. Rule set 7 has 12 fork nodes and requires 719 stitches resulting in a safe DFA of 2095 states. For rule set 18, the safe DFA has 21448 states and 671609 transitions. Finally, rule set 6 has a large DFA of 192 nodes, of which 142 are fork nodes. The number of stitches alone is 30142 and the resulting safe DFA could not fit within the 1.5 GB allocated by our code. This shows that there is a need for using a different resolution method for these three and similar rule sets. When the safe DFA size is reasonable, a new rule set can be readily generated by considering each state as a separate binary number. This implies $\lceil log_2(|Q|) \rceil$ binary digits, then map

Table 2. Vulnerable Rule Sets

	Flowbit Rule Set DFA				SafeNFA	SafeDFA			
ID	Σ	Q	δ	Forks	Stitches	Q_1	$\delta 1$	$\frac{Q_1}{Q}$	$\frac{\delta_1}{\delta}$
1	6	5	30	2	3	7	20	1.40	0.67
2	5	17	85	13	7	37	160	2.18	1.88
3	6	2	12	1	4	5	21	2.50	1.75
4	3	5	15	2	1	6	17	1.20	1.13
5	3	3	9	1	2	4	10	1.33	1.11
6	34	194	6596	142	30142	Huge	Huge	Huge	Huge
7	13	21	273	12	719	2095	$2x10^4$	99.76	88.84
8	3	5	15	1	1	7	18	1.40	1.20
9	5	17	85	13	7	37	155	2.18	1.82
10	3	5	15	1	1	7	20	1.40	1.33
11	3	3	9	1	1	5	14	1.67	1.56
12	39	3	117	1	35	5	49	1.67	0.42
13	16	3	48	1	10	5	35	1.67	0.73
14	12	3	36	1	6	5	31	1.67	0.86
15	2	3	6	1	1	4	8	1.33	1.33
16	2	3	6	1	1	4	8	1.33	1.33
17	8	3	24	1	6	5	22	1.67	0.92
18	56	18	1008	8	1328	$2x10^4$	$7x10^5$	1191	666

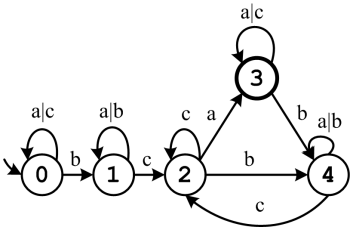

Figure 7. DFA of rule set 10

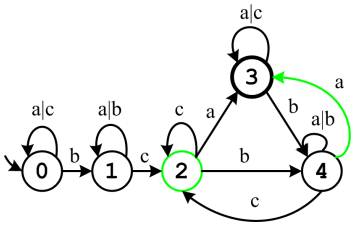

Figure 8. Its Stitched NFA

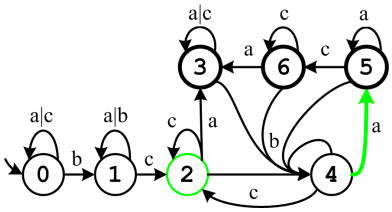

Figure 9. Its Safe DFA

each digit to a flowbit label and use that in the condition and output of each transition, resulting in a number of δ rules.

6. Related Work

Ptacek and Newsham [6] were the first to bring up a way to evade a NIDS by using TCP Segmentation and IP Fragmentation. FragRoute was the tool created to carry out these evasion techniques. A NIDS needs to carry out TCP segments and IP fragments reassembly to counter these evasion techniques. Handley and Paxson [7] [8] discussed evasion techniques based on inherent ambiguities of the TCP/IP protocol which lead to the difference between a NIDS and its protected system in performing TCP segments and IP fragments reassembly. Traffic normalization suggested by Handley et al. [7] tries to remove these ambiguities by patching the packet stream. *Active Mapping* is another solution proposed by Shankar and Paxson [9], which eliminates TCP/IP-based ambiguities in a NIDS analysis with minimal runtime cost. It is implemented in the Snort Stream5 preprocessor [10].

Besides NIDS evasion techniques, several attacks on IDSs have been identified in the literature. Wagner and Soto [11] revealed mimicry attacks on a HIDS. Snot [12], Stick [13], IDSWakeup [14] and Mucus [15] are stimulation tools that cause DOS attacks on Snort by overloading it with alerts from mutated packets constructed from Snort rules. The algorithmic complexity DOS is another NIDS attack identified in [16] [17].

Rubin et al. [18] created the AGNET tool which uses an inference engine to generate as well as detect attacks on a NIDS starting from an exemplary attack instance. Our signature attack is not an instance generated by AGENT, assuming that the rule set represents the original exemplary attack instance. Our attack is neither a TCP nor an application-level transformation. Existing evasion techniques can be used by our evasion. However, these techniques only apply to injected packets which are not part of the actual session. Automatically generated semantic-aware signatures [19] or session signatures [20] can also be vulnerable to our evasion. In order to avoid false positives, these generated signatures consider "innocent"paths (or sequences) which are not attack instances. Our signature evasion exploits these "innocent" paths and tries to convince the IDS that the session is following one of the "innocent" paths.

7. Conclusion

In this paper, we identified and formally specified a new type of evasion on IDS signature rule sets, named signature evasions. We provided an algorithm to patch a rule set against this type of evasion based on DFA manipulation. The algorithm runs in polytime and derives a minimal rectification to the rule set by identifying the smallest number of transitions (stitchings) to apply on the rule set DFA to render it safe.

We will develop a tool to assist in the design of signature-evasion safe rule sets as well as the patching of existing ones. We also intend to investigate the problem of optimal mapping from a stitched NFA to a signature rule set in order to solve the scalability problem inherent to using signature-evasion safe DFAs.

References

[1] D. Mutz, C. Kruegel, W. Robertson, G. Vigna, and R. A. Kemmerer, "Reverse engineering of network signatures," in *AusCERT*, 2005.

[2] I. Sourcefire, *Snort Users Manual*, 2003-2008.

[3] BleedingEdge Inc., http://www.bleedingthreats.net.

[4] SourceFire, Inc., http://www.sourcefire.com.

[5] E. Gansner, E. Koutsofios, and S. North, "Drawing graphs with dot," Jan 2006.

[6] T. Ptacek and T. Newsham, "Insertion, Evasion, and Denial of Service: Eluding Network Intrusion Detection," 1998.

[7] M. Handley, V. Paxson, and C. Kreibich, "Network intrusion detection: evasion, traffic normalization, and end-to-end protocol semantics," in *USENIX Security*, 2001.

[8] V. Paxson, "Bro: a system for detecting network intruders in real-time," in *USENIX Security*, 1998.

[9] U. Shankar and V. Paxson, "Active mapping: resisting NIDS evasion without altering traffic," in *IEEE Symposium on Security and Privacy*, 2003.

[10] J. Novak and S. Sturges, "Target-Based TCP Stream Reassembly," Sourcefire, Inc., 2007.

[11] D. Wagner and P. Soto, "Mimicry attacks on host-based intrusion detection systems," in *ACM CCS*, 2002.

[12] SoftPedia, "Snot description," Mar 2009.

[13] C. Giovanni, "Fun with Packets: Designing a Stick," White Paper, Tech. Rep., 2001.

[14] HSC, "Idswakeup collection of tools," Jan 2002.

[15] D. Mutz, G. Vigna, and R. Kemmerer, "An experience developing an IDS stimulator for the black-box testing of network intrusion detection systems," in *IEEE ACSAC*, 2003.

[16] S. Crosby and D. Wallach, "Denial of Service via Algorithmic Complexity Attacks," in *USENIX Security*, 2003.

[17] R. Smith, C. Estan, and S. Jha, "Backtracking algorithmic complexity attacks against a NIDS," in *IEEE ACSAC*, 2006.

[18] S. Rubin, S. Jha, and B. P. Miller, "Automatic generation and analysis of NIDS attacks," in *IEEE ACSAC*, 2004.

[19] V. Yegneswaran, J. T. Giffin, P. Barford, and S. Jha, "An architecture for generating semantics-aware signatures," in *USENIX Security*, 2003.

[20] S. Rubin, S. Jha, and B. Miller, "Language-based generation and evaluation of NIDS signatures," in *IEEE Symposium on Security and Privacy*, 2005.

Proofs

Proof of Lem.1 (page 3)

By definition, any flow $p \in \mathcal{L}(D)$ triggers at least one alert rule of $\mathcal{R}$. p can be decomposed into two sub flows $p_1 p_2$ where the last packet of p_1 triggers the first alert. Hence, $p_1 \in \mathcal{L}_{std} D$ as it brings it to a target state by definition of how we construct a rule set DFA. The second sub-flow p_2 is only required to continue taking D from one state to another (*i.e.*, no unexpected input). Since it starts doing so starting from a final state of D, hence $p_2 \in \mathcal{L}_{std}(D^\circ)$. We conclude that:

$$\mathcal{L}(D) = \mathcal{L}_{std}(D)\mathcal{L}_{std}(D^\circ) = \mathcal{L}_{std}(DD^\circ)$$

Proof of Thm.1 (page 3)

By definition, an r-evasion q is a sequence of signatures which never crosses a target state of D_r, *i.e.*, $q \in \neg D_{-r}$. In addition, an r-evasion is detectable by the actual session DFA. By Lem.1 this implies that $q \in D_r^+ D_r^{+\circ}$.

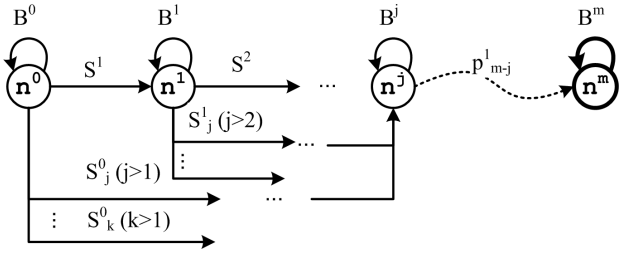

Figure 10. Disjoint-Hop Trail DFA

Proof of Thm.2 (page 3)

Let $D = (Q, \Sigma, \delta, n^0, F)$ be a trail DFA where $Q = \{n^0, n^1, \ldots, n^m\}$ and D^+ its false positive closure. Without limiting the generality of the proof we assume that $n^m \in F$ (because of the articulation property of target states (Def.3.2)). Let S^i the disjunction of signature that link node n^{i-1} to n^i, *i.e.*, $(n^{i-1}, S^i, n^i) \subset \delta$. Transitions between nodes $n^i, n^j | j > i+1$ are denoted by S^i_j. In addition, since buckles are allowed in a trail DFA, we denote by B^i the buckle at state n^i (disjunction of all individual buckles at that state).

Let's assume that target state n^m is evadable. This implies that there exists a path p^1_m such that $n^0 \oplus p^1_{m,+} = n^m$ and another path p^2_m such that: $p^1_{m,+} = p^2_m \ominus \Sigma_*$, $n^0 \oplus_D p^2_m \neq n^m$, $n^0 \oplus_{D^+} p^2_m = n^m$.

We will prove by recurrence on m:

1) Case where $m = 1$: Since p^2_1 cannot reach n^1, it can only be of the form $(B^0)^*$, which in turn cannot end with s^1 because $s^1 \notin B^0$ by definition of a DFA. Hence, p^2_1 cannot exist $\Rightarrow$ D is not evadable.

2) Case where $m = N+1, N \geq 1$ (Fig.10): Let's assume that disjoint-hop trail DFAs where $1 \leq m \leq N$ are not evadable. Let D be a disjoint-hop trail DFA where $m = N+1$. p^1_m is of the form:

$$p^1_m = (B^0_+)^a s^0_{j,+} p^1_{m-j,+} \tag{4}$$

where $a \geq 0$, $s^0_1 \in S^1$, $s^0_j \in S^0_j$, and $n^j \oplus p^1_{m-j,+} = n^m$. n^j cannot be node n^0 by definition of the DFA.
If p^2_m exists, it has then to be of the form:

$$p^2_m = q_k p^2_{m-k} \tag{5}$$

where, $n^0 \oplus q_k = n^k$, $q_k \ominus \Sigma_* = (B^0_+)^a s^0_{j,+}$, $n^k \oplus p^2_{m-k} \neq n^m$. Since $s^0_{j,+} \in q$, k cannot be less than j because of the dijoint-hop property (Def.3.2) and the trail property of the DFA (once n^0 is left it cannot be re-visited). Hence, $k \geq j$. We have $n^0 \oplus p^2_m = n^0 \oplus (q_k p^2_{m-k}) = (n^0 \oplus q_k) \oplus p^2_{m-k} = n^k \oplus p^2_{m-k}$. That p^2_m is an evasion of p^1_m implies that we need to have: $n^j \oplus_D p^1_{m-j,+} = n^k \oplus_D p^2_{m-k}$, and $n^j \oplus_{D^+} p^1_{m-j,+} = n^j \oplus_{D^+} p^2_{m-k}$. The solution to this latter problem

if it exists implies a solution to the problem where $n^j \oplus p^1_{m-j,+} = n^j \oplus q' \oplus p^2_{m-l}$ in D. Since by hypothesis of recursion this latter condition is impossible, it implies that p^2_{m-k} cannot exist, which in turn concludes the proof.

Proof of Thm.3 (page 3)

Let $p_{min} = s^1 s^2 \dots s^m$ be the simple path derived from p. By definition of a simple path, $n^0 \oplus p = n^0 \oplus p_{min}$, p_{min} has no loops or cycles, and p is of the form:
$p = p^1 s^1 p^2 s^2 \dots p^m s^m p^{m+1}$
where each $p^i \in \Sigma^*$. Since q evades p, then q is of the form:
$q = q^1 s^1_+ q^2 s^2_+ \dots q^m s^m_+ q^{m+1}$
where $q^i \ominus \Sigma_* = p^i_+$. It implies that the path:
$q' = q^1_* s^1_+ q^2_* s^2_+ \dots q^m_* s^m_+ q^{m+1}_*$
has the properties:
$q' \ominus \Sigma_* = p_{min,+} \Rightarrow n^0 \oplus q' = n^0 \oplus p_{min,+}$, $n^0 \oplus q' = n^0 \oplus q$
which by definition evades p_{min}.

Proof of Lem.2 (page 3)

By definition of a simple path, p and q cannot be on the same trail by virtue of Thm.2. Hence, q has to fork at least once with p. In addition, Thm.2 indicates that a disjoint-hop fork (Def.3.2) on a trail cannot lead to any successful evasions. Hence, q has at least one non disjoint-hop fork with p.

Proof of Thm.4 (page 3)

Case the s_+ of q belongs to q^1.

1) Case where transition $(a, t_*, c) \in q$: This case fits perfectly with the condition of the theorem (Fig.5).

2) Case where transition $(a, t_+, c) \in q$ (Fig.3): We look if it is possible to derive a new evasion path q' that evades p but uses transition (a, t_*, c) instead of (a, t_+, c). Let g be the node crossed by p^0 such that:
$p^0 = p^0_1 t p^0_2$, $n^0 \oplus p^0_1 = g$,
$q^0 = q^0_1 t' q^0_2$, $n^0 \oplus q^0_1 = g$,
t' is null ($g = h$) or any transition of Σ including t,
$g \oplus t = g \oplus t' = h$,
$t_+ q^1_1 s_+ \ominus \Sigma_* = t_+ p^0_2 s_+$.
It derives from $t_+ q^1_1 s_+ \ominus \Sigma_* = t_+ p^0_2 s_+$ that $q^1_1 \ominus \Sigma_* = p^0_2$. This implies that the path $q' = (p^0_{1,+} t_+ p^0_{2,+}) t_* (q^1_{1,*} s_+ q^1_2)$ evades p, which also fulfills the conditions of the theorem.

Case the s_+ is traversed by p^0 (Fig.4).

Let g be the node crossed by p^0 such that:
$p^0 = p^0_1 t p^0_2$, $n^0 \oplus p^0_1 = g$,
$q^0 = q^0_1 s' q^0_2$, $n^0 \oplus q^0_1 = g$,
s' may be null ($g = h$) or any signature $\in \Sigma$
$g \oplus s = g \oplus s' = h$, and
$g \oplus s_+ p^0_2 s_+ = b$,
We then have: $p_+ = p^0_{1,+} s_+ p^0_{2,+} s_+ p^1_{1,+} t_+ p^1_{2,+}$ and $q = q^0_1 s_+ q^0_2 t_+ q^1$. It derives that $n^0 \oplus q^0_1 = n^0 \oplus p^0_{1,+} s_+ p^0_{2,+}$, which by Thm.2 implies that $g = h = \dots = a$. However, the latter condition is impossible by DFA definition because a has already a transition to $b \neq a$. Hence q cannot exist.

Proof of Thm.5 (page 5)

Let q be an evasion path as identified in Thm.4. For q to be successful (1) q^1_2 has to make D^+_r pass through a target state of F_r starting from state b and (2) from state e in D_r no target state in F_r is traversed through q^1_2. The first constraint implies that from state b in D^+_r a target state $\in F_r$ is first reached then any other walk in D^+_r is acceptable $\Leftrightarrow q^1_2 \in D^{+b}_r D^{b+\circ}_r$. By definition it can be easily noticed that $D^{b+\circ}_r = D^{+\circ}_r \Leftrightarrow q^1_2 \in D^{+b}_r D^{+\circ}_r$. The second constraint implies that $q^1_2 \in \neg D^e_{-r}$. Hence $\mathcal{E}^{b,e}_r = \neg D^e_{-r} \cap (D^{+b}_r D^{+\circ}_r)$.

Proof of Lem.3 (page 5)

Since finding the set of all fork tuples (a, b, c) takes $O(|Q||\delta|)$, identifying $suc^*(c)$ for all nodes is $\simeq O(|Q|)$, identifying the function $nodes(s)$ giving the set of nodes with an outgoing transition s is $O(|\delta|)$ we conclude that algorithm Alg.1 runs in polytime if only the test $\mathcal{E}^{b,e}_r \neq \Phi$ is doable in polytime. In fact not only $\mathcal{E}^{b,e}_r \neq \Phi$ can be done in polytime but also the construction of $\mathcal{E}^{b,e}_r$. The intersection needs to be made by traversing states of $\neg D^e_{-r}$ and using a marker m initially unset. The traversal of $\neg D^e_{-r}$ is done by making sure the corresponding transition is also feasible in D^{+b}_r. If in the process of doing so a target state of D^{+b}_r is encountered the marker m is set. At that point, all states of D^{+b}_r are set as final by the traversing process and the process of computing the intersection continues in the standard way. Hence, in essence constructing $\mathcal{E}^{b,e}_r$ will take no longer than the time it takes to constructing $\neg D^e_{-r} \cap D^{+b}_r$ twice.