

FireCracker: A Framework for Inferring Firewall Policy using Smart Probing

Taghrid Samak, Adel El-Atawy, and Ehab Al-Shaer
DePaul University
Chicago, IL, USA 60604
{taghrid,aelatawy,ehab}@cs.depaul.edu

Abstract—A firewall policy that is correct and complete is crucial to the safety of a computer network. An adversary will benefit a lot from knowing the policy or its semantics. In this paper, we propose a framework that could be used to blindly discover a firewall policy without prior knowledge. We show how an attacker can reconstruct a firewall’s policy by probing the firewall with tailored packets into a network and forming an idea of what the policy looks like. The proposed methodology shows how to discover a policy that is semantically equivalent to the original one used in the deployed firewall. Three techniques are proposed for reconstructing the policy as well as to intelligently choose the probing packets adaptively based on the firewall response. We show the possibility of obtaining the deployed policy in a feasible time with acceptable accuracy.

I. INTRODUCTION

Identifying the anatomy of the victim’s defenses is always an asset to any attacker. In computer networks, attackers are always keen to know every possible detail about the target networks/corporates. The information can be the topology of the network, the IP’s used for servers and workstations inside the network, and the defense mechanisms available. Firewalls are currently considered one of the cornerstones of any network defense mechanism. The main purpose of firewalls is to filter out incoming/outgoing packets to/from the network based on packet headers using a given policy defined by the network administrator.

The firewall policy consists of a list of rules, with each rule representing a set of conditions and an action. If an incoming packet matches all of these conditions, then the action is taken: allow the packet to pass or drop this packet immediately. A packet can match the conditions of more than one rule, in such a case, the first rule will have priority and its action (*i.e.*, allow/reject) will be applied to the packet.

In this paper, we study the possibility that an attacker can actively probe the firewall in order to deduce the policy used to protect the target network. In general, this discovery process involves three main properties of the target policy: 1) policy structure, 2) the default deny rule and 3) the partial order relation between the rules. Disclosing each of those properties, alone or joint, has a major effect on the safety/security of the network.

1) *Policy Structure*: Policy structure refers to the acceptance region of the policy address space. The discovered structure represents a semantically equivalent policy to the one configured on the firewall. As a result, the

attacker can target the protected network via erroneously opened protocols/ports and address ranges.

- 2) *Default rule*: The default rule is usually a deny rule and it may not be explicitly mentioned in the policy. Packets matching this rule only are the most expensive ones on performance, because they undergo filtering against all rules in the policy before being rejected [10]. Knowing the default rule enables an attacker to launch a denial of service (DoS) on the firewall itself.
- 3) *Rules Relation*: The filtering algorithm within the firewall can be detected as well. The partial order relation between the rules can be discovered by taking timing and packets’ delays into account. Such an analysis will work directly against the actual implementation of the firewall filtering algorithm. Thus, discovering the filtering technique of the firewall to some extent. A DoS attack can be designed to target rules having maximum matching time.

The main focus of the paper will be the discovery of the policy structure. By intelligent selection of test packets, the firewall could be probed. The response of the firewall to those packets will reveal the policy structure. The learning of firewall policy structure is similar to Boolean function learning from examples, except in this case the samples are not randomly selected or provided.

Running an exhaustive sampling of all possible packets in order to reconstruct the policy will have a 100% accuracy, but is impossible due to the huge space needed (*i.e.*, $28 * 2^{32} * 2^{32} * 2^{16} * 2^{16} = 2^{104}$). Therefore, an intelligent sampling is essential to exploit the nature and structure by which we specify firewall rules. Almost all firewalls restrict rule syntax to be a conjunctive of conditions on packet header fields, and each field is checked against a range of consecutive values. For example, a very short firewall policy might look like:

```
R1: deny icmp any 150.160.170.*
R2: deny tcp any:any 150.160.170.*:1024-65535
R3: allow tcp any:any 150.160.170.*:80
R4: deny any any:any any:any
```

As we can see, on each dimension the condition is stated as a single value or a range of consecutive values (including the “any”). Thus, the overall space that falls within a rule is a hyper rectangle, and our task is to identify its boundaries. Learning Boolean functions can be used as well, but the nature of firewall policy rules makes our proposed methods more suited than general purpose function discovery techniques [4],

[14]. The fact that policy rules have hyper-rectangular shape and mostly in powers of 2 (due to the way IP address masks are specified) intuitively suggest that the use of specific algorithm instead of general learning can be highly beneficial.

The samples used to learn the policy are used based on the firewall response to previous samples. In this Boolean function, we are only interested in the positive samples that correspond to the acceptance space. In this work, we propose three main techniques to discover the policy space. The first method uses space division and region growing. The second method uses a variant of the split-and-merge technique. The third technique is a genetic algorithm with a simplistic objective function. We introduced the idea behind the first two techniques in [19]. A general attack will look like this:

- 1) Generate packets
- 2) Probe firewall
- 3) Generate next sample according to the firewall response
- 4) Combine the information obtained to deduce policy rules

The preliminary testing results on the framework show that by using a reasonable number of packets, the detected policies have satisfying accuracy compared to the original ones deployed in the firewall.

The rest of the paper will be organized to show the system design, describe the techniques and provide the results of the experiments performed on the implemented prototype of the system. In section II, we give a quick overview of some of the related work. Next, in section IV, the structure of the system is presented with a description of each of its modules. The following section will contain a detailed description of the techniques and their implementation. Then, sections VI, and VII describe the Random packet analysis and the injection/reporting modules respectively. In section VIII, the system is evaluated, and the results obtained are presented. Finally, ideas about defenses from this attack are presented.

II. RELATED WORK

To our knowledge, very little work was directed to attacking the firewall actively in order to obtain the policy it implements. Most of the available work addresses the problem on a host-by-host basis, and without using any knowledge about the nature of firewall ACL rules, making such techniques less likely to be scalable or automated. In [7], a method similar to traceroute is introduced to scan a firewall for open ports in order to discover both available hosts behind the device and ACL filtering rules. The procedure starts by identifying the path to the firewall using increasing TTL values for IP header. It then uses this TTL to probe individual hosts and ports in incremental fashion. No intelligent packet selection is used in this case. This makes the approach easy to detect by simple scan checking algorithm.

However, some of firewall analysis tools are available but their main goal is to help administrators understand and optimize policies. Information about the firewall and network topology are provided to the tool. This is different from our work here since our main focus instead is to discover the whole policy without knowing anything about the internal network structure. In [16] a firewall analysis tool (FANG)

is concerned with policy discovery using specific queries to help administrators understand and modify configurations. It works well given extra information about network topology and configuration. An extension to this work was introduced in [21], (Lumeta) where the query selection is automated. The Lumeta architecture is designed to limit user interaction as opposed to FANG. However, it still needs topology and routing information.

Other tools aim at discovering conflicts in policies due to misconfiguration [2], [3] and [22]. In [2], conflicts are classified and analyzed for single firewalls as well as distributed environment. Another approach to design firewall policies has been proposed in [9]. This work provide a policy model to verify consistency, completeness and compactness of the policy.

Other firewall analysis tools focus on the performance of firewall in terms of implementation and filtering delays [15] and [11].

Some work has been done where the analysis was performed to test firewalls for the vulnerability to traffic-specific attacks, as IP spoofing attacks which was addressed in [20]. In [13], performance metrics for vulnerabilities resulting from firewall operations are presented and analyzed.

Blind discovery of firewall policies was introduced in [19]. Two approaches were used to select probing packets, along with preliminary results. This work aimed to be a proof of concept on the feasibility of such discovery. We present here an extensive evaluation to those approaches and improve the packet selection methods. We also introduce a new technique to overcome some of the their limitations.

As will be discussed in section III, firewall policy can be formulated as a boolean function, specifically a decision list. Learning general boolean functions has been studied in the literature widely, [14], [4] and [18]. Most of the work done in learning boolean formulae is based on the assumption that the samples are given or randomly chosen from a known distribution. In our work here, we make use of the properties of firewall policies to dynamically choose samples to learn the function. Efficient algorithms for online learning of decision lists are proposed in [17]. However, the samples used are also known, but being processed one by one.

III. THEORETICAL BACKGROUND

In this section we will investigate the learnability of firewall policies. The theoretical foundation for the learning techniques will give estimates to the number of samples needed to discover a certain policy as well as put upper bounds on the performance of such discovery techniques. To be able to calculate a theoretical bound for the number of packets needed to learn the policy given a certain level of accuracy, we will model the firewall policy as a decision list.

For a boolean function with n variables, where each term consists of at most k variables; a k -decision list (k -DL), as defined in [18], is a list L of pairs

$$(f_1, v_1), (f_2, v_2), \dots, (f_r, v_r)$$

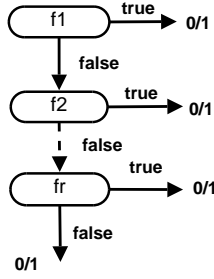


Fig. 1. Decision List Operation

where each f_i is a term in C_k^n , each v_i is a value in $\{0, 1\}$, and the last function f_r is the constant function: **true**.

The boolean function is defined, over the domain of all possible assignments X_n , as follows. For any assignment $x \in X_n$, $L(x)$ is defined to be equal to v_j where j is the *least* index such that $f_j(x) = 1$. Such an item always exists since the last function is always **true**.

From this definition, a decision list can be expressed as a sequence of “if - then - elseif ... else-” rules.

Figure 1 shows the operation of a decision list. The value of the function is calculated by sequential matching of the variables against f_i . If f_i is true, the function will take the corresponding value. If it is false, the *false* branch will be followed and the next condition will be evaluated. The operation ends when all conditions are tested, and none of them was matched. In this case, the default value is applied.

Firewall policies can be modeled as k-Decision Lists, and hence the learnability of the policy can be proven.

Considering the formal definition of a firewall policy from [3]: A firewall policy is a set of filtering rules that control the action of the firewall in response to incoming traffic. An *access policy*, $P = R_1, R_2, \dots, R_n$, is a sequence of n filtering rules that determine the appropriate action performed on any incoming packet.

A *filtering rule*, R_i , consists of a set of constraints on a set of k filtering fields, $X = x_1, x_2, \dots, x_n$, together with an action, act_i , from the set of all possible actions, A .

Each rule can be written in the form: $R_i : C_i \Rightarrow act_i$ where C_i is the constraints on the filtering fields that must be satisfied for the action act_i to be triggered. In firewalls, the set of possible actions, A , has only two actions $\langle permit, deny \rangle$, that can be encoded as one boolean variable or, in our case, be considered the output of the Boolean function itself. The rules of a firewall policy are matched in order against incoming packets. The response of the firewall is the action of the first rule that has field values satisfying the current packet. If the packet does not match with any rule, the packet is denied. The mapping from firewall policy to decision list is performed by mapping each rule condition C_i to function f_i in the decision list. The action of the firewall in response to satisfying C_i is mapped to v_i , the value paired with the function. The available actions are mapped to the set $\{0, 1\}$, where *permit* $\equiv 1$ and *deny* $\equiv 0$.

The fields in our case, X_n space, consist of packet header fields. Each numeric field (source IP, destination IP, source port, destination port, and protocol) is encoded as its binary bit values. The space of the packet header fields becomes the boolean function sample space. The function value is the action of the firewall corresponding to a specific combination of field values.

The goal of our approach is to learn the value of this function for all packet space.

The learnability of decision lists is discussed also in [18], under the PAC model [4]. The minimum number of samples, m , needed to learn k-DL follows the relation:

$$m > \frac{1}{\epsilon} (\ln(|H|) + \ln(\frac{1}{\delta})) \quad (1)$$

where,

$|H|$ is the cardinality of the hypothesis space (the total number of possible functions).

ϵ is the accuracy parameter, the maximum error between the original policy and the discovered one,

and δ is the confidence parameter, the probability that the learning algorithm produces an accurate policy must be at least $1 - \delta$

This relation is valid only for $1 < k < n$. The number of variable defining each term is constrained to be strictly less than the total number of variables. This constraints prevent any rule from having in its definition all bit header fields. In other words, to be able to learn the policy, there have to be some “*don’t care bits*” in the policy rules. An actual policy might contain exact/specific rules, having all the bits mentioned. Those rules are not of interest to the attacking approach, since the major parts of the policy will be in *range* form. Besides, very specific rules leave nothing for the attacker to play around with as fully specified rules includes the source address and port as exact values.

Thus, this equation puts a limit on the performance of any algorithm. In other words, no algorithm can guarantee to always provide results with better accuracy within this limit on number of packets. For a learning algorithm, A , to be able to discover the policy, at least m packets should be used. In the following section, we present such algorithms. The previous discussion also assumes that with random sampling, the policy can still be learned given a certain accuracy (we can not improve over $(1 - \epsilon)$ accuracy using m packets).

In this paper, we show that it is possible to get close to this limit in many cases. In VIII, we see that in most cases the discrepancy between the upper bound on accuracy and the one actually obtained by our algorithms is sufficiently small. In some few cases, our algorithms even surpassed the accuracy stated by the formula using the same number of packets.

IV. SYSTEM ARCHITECTURE

The purpose of our proposed system is to discover the closest possible logical structure of a firewall policy with minimum number of packets sent to the firewall. To achieve this goal, we need to perform two main functions: 1) probing

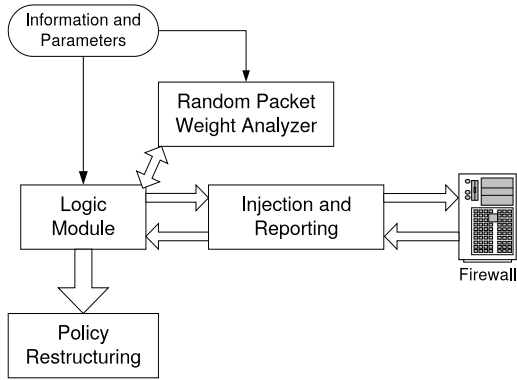


Fig. 2. Framework Structure

packets into the firewall with a feasible reporting mechanism to know which packets were accepted and which have been denied and 2) selecting intelligently those packets considering some basic properties of any firewall policy.

Another design goal is to have such system practically implementable, and expandable with future packet selection and policy rebuilding techniques. Thus, the system is built (almost all parts are implemented) as a layered system where different versions of a layer can be selected and/or replaced flexibly.

Our framework has three main layers corresponding to those main functions; a logical module, injection/reporting module, and policy restructuring module. Figure 2 shows these components and their interconnections. The injection/reporting module is responsible for communicating with the firewall, sending packets and reporting back the action of the firewall. The logic layer (higher level) performs the actual discovery of collective firewall response to sent packets, and performs the packet selection process, to guarantee that only the most useful packets have been sent to discover the policy. This process involves two main operations; candidate selections and space analysis. The candidate selection uses some elementary logic to choose the basic set of packets to be injected. According to the action of the firewall responding to those kernel candidates, the space analysis module performs a more advanced selection to discover the sub regions of the policy depending on previous actions. In many cases, both operations are intertwined together without clear separation of duties. Thus, they are both kept as a single module. However, another critical task is separated in the module of Random Packet Weight Analysis. This module is referenced by the Logic module whenever the later needs random packets. The Random Packet module generates random packets that are more likely to be popular in practical policies. Examples of common selection guidelines by which this module selects its random samples are discussed in section VI.

In each module we investigate multiple techniques for performing the operations. For each proposed technique, the communication between the two modules is transparent. Using

whatever method at each layer does not affect the performance of the other layer. The logical module generates packets and sends them to injection. The injection/reporting module reports the action of each packet back to the logic. For the logical module, we investigate the two techniques introduced in [19] for the space analysis part. We also introduce a new technique based on genetic algorithms navigate the space. The candidate selection process uses packets properties along with information about general firewall policies to optimize the number of packets to be sent.

The injection/reporting step uses a set of techniques to obtain the firewall response. In section VII, a description of these techniques is shown. They include using ICMP messages, compromised or worm infected hosts.

The Policy Restructuring Module runs once at the final stage to clean up the policy structure. After the discovery algorithm terminates, the policy will be simplified by any of the already available rule simplification procedures as the ones mentioned in [9]. Also, redundant and shadowed rules [3] will be eliminated as part of this process.

In the following sections, we will describe in more detail the operation of each module.

V. SPACE NAVIGATION AND POLICY LEARNING TECHNIQUES

In this section we describe the techniques used to discover policy structure. By navigating the space of all field values, the action of the firewall against generated packets helps understand the policy. The packets are generated adaptively and intelligently with guided learning algorithms that make use of the firewall response to previous packets such that attack time and number of packets are minimized.

The only primitive operations that the attacker uses is sending a packet and waiting to see whether it is going to be allowed or filtered by the firewall, and generating a random packet. These operation are implemented by the Injection/Reporting Module, and the Random Packet Analyzer module respectively.

Every packet sent is a point in the traffic space, which is the space over which our target function is to be discovered. The space is composed of all possible values of the dimensions being investigated (*i.e.*, the space composed of different values for source/destination IP address and port). Every packet (*i.e.*, point/sample) has one of two values (or signs), either allowed or denied (positive or negative). Thus a single test packet sent will evaluate the filtering policy at a single point in order to identify the hyper-rectangles covered by each one of the policy rules. We will use the notion of points and signs to simplify the discussion and analysis.

First, we describe two methods that are based on space division. The first method starts from points in space with known action and keeps extending the region to find boundaries for the rule. The second method goes in the opposite direction; starting from the whole available space, it divides the space as long as there are different actions within the area under testing.

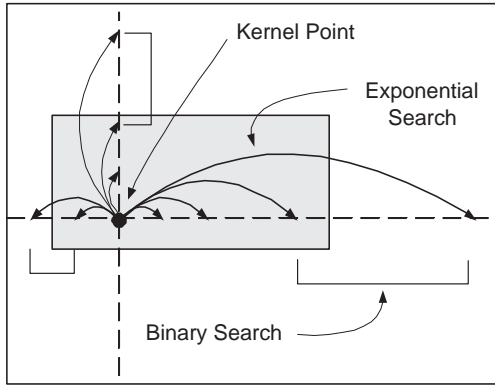


Fig. 3. Searching for rule boundaries starting from a kernel

A. Region Growing Approach

In this method, we start assuming the default “deny all” rule. By sampling from the space, we wait till a packet passes through the firewall indicating a “permit” rule, and this packet will be called the rule “kernel”. We then perform an exponential search in each of the d dimensions (e.g., $d = 5$ for the case of a firewall that uses the protocol, source and destination address and port fields to specify rules), to find a change in the sign that indicate the end of the rule in that dimension. Following the exponential search comes a binary search to pinpoint the exact boundary of the rule’s space (See Figure 3). These two steps takes only $O(\lg n)$, where $\lg n$ is the number of bits in each one of the fields (i.e., IP=32, port=16, protocol=8 bits).

Once the boundaries of the rule are identified, we, recursively, start sampling inside the rule searching for any rules that are exceptions to this rule (i.e., searching for a new kernel). This is the same operation performed previously as we searched for this rule as an exception of the “default deny” rule, and so on.

The algorithm executes as follows:

- 1) Search for a point in the space with a different action than the default.
- 2) Grow the region surrounding this point till a reverse action is found in all directions.
- 3) Recursively run the procedure on the outside regions and the discovered one, till you reach your stopping limit.

In the algorithm, we simply start with the network-wide limits as the default sampling space, which represents the “default deny” rule. The algorithm then samples to find a kernel point with opposite action, then it identifies the boundaries for the current discovered rule. Once the rectangle of the newly identified rule is obtained, the space around the rule is partitioned as shown in figure 4(b) for simple two-dimensional space ($d = 2$). This partitioning pattern is used rather than the more natural split-all-dimensions partitioning (figure 4(a)), because this will yield $1+2d$ partitions instead of 3^d partitions. Each subspace (including the current rule) will be recursively analyzed. The stopping criterion of the algorithm depends on the maximum number of packets the attacker has to send, or

the time allowed. For details of the algorithm and its stopping criteria, refer to [19].

B. Split-and-Merge Approach

In this method, we use the split-and-merge algorithm that is used for image segmentation [8]. The main objective is to partition the space into n non-overlapping regions based on a partitioning criterion. In our case the regions will represent individual policy rules, and the criterion will be the action of the rule and its shape. To formalize our problem we consider the whole space as the initial region R which will be the default “deny” rule that covers all the space. We wish to partition R into n sub-regions (sub-rules) R_1, R_2, \dots, R_n such that:

- 1) $\bigcup_{i=1}^n R_i = R$
- 2) R_i is a connected region
- 3) R_i has a rectangular shape (policy rule restriction)
- 4) $R_i \cap R_j = \Phi$ for all i and j where $i \neq j$
- 5) $P(R_i) = TRUE$ for $i = 1, 2, \dots, n$
- 6) $P(R_i \cup R_j) = FALSE$ for $i \neq j$

where $P(R_i)$ is a logical predicate applied to region R_i . In the case of rule identification, the predicate value will be whether all points (i.e., packets belonging to this sub-space) in this region have the same action (i.e., receive the same treatment from the firewall: permit/deny). The third condition is specific to our application, so each detected rule will have a rectangular shape.

The following is a simple description for the split-and-merge algorithm:

- 1) Split into four disjoint quadrant any region R_i for which $P(R_i) = FALSE$.
- 2) Merge any adjacent regions R_i and R_j for which $P(R_i \cup R_j) = TRUE$.
- 3) Stop when no further merging or splitting is possible.

This algorithm restricts the splitting to equal quadrant. The splitting can also be performed in a more general way depending on the space of the problem. For testing the predicate $P(\dots)$, sampling is preformed to select which points in space to consider for evaluating the predicate. First the whole space is divided into four quadrants then recursively each quadrant will be investigated and split. The predicate is evaluated on the lower level, then the merging is performed on the way up to the whole space. The sampled points for our algorithm will be the packets sent by the attacker.

So far, our discussion assumes that the data are two dimensional. For the general case of d dimensions (multiple packet header fields), slight change is required to the algorithm. The rule will have a d -dimensional hyper-rectangular shape. The splitting also will be performed considering the higher dimensionality. Instead of splitting each region into four quadrants, the region will be partitioned into 2^d partitions. However, the basic operation of the algorithm stays essentially the same. The algorithm details can be found in [19].

Figure 5 illustrates three iterations of the algorithm on a two-dimensional space.

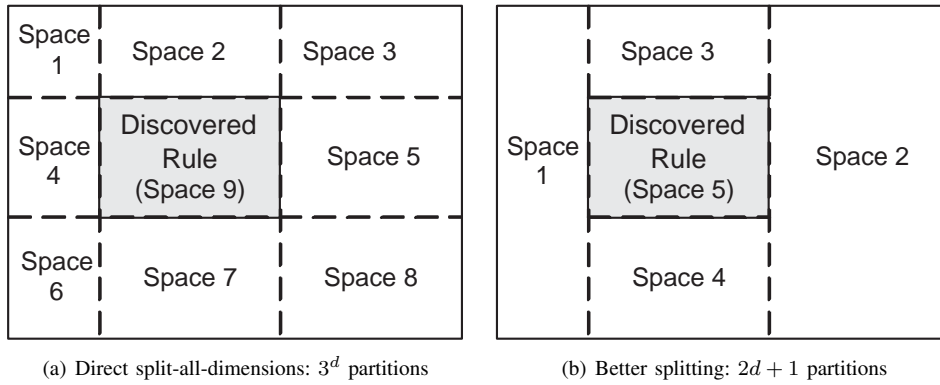


Fig. 4. Partitioning Space for Region Growing

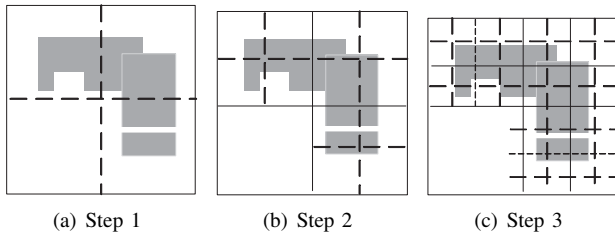


Fig. 5. Split-and-merge example.

C. Genetic Algorithm

In this section, we present a genetic algorithm (GA) approach for discovering policy structure. We start by briefly introducing the concepts of GA, then we show how that can be applied in the case of policy discovery. In general, a genetic algorithm investigates the search space of a problem to find the best solution. By an iterative procedure starting from an arbitrary solution, this solution evolves to reach the optimal. The measure of improvement is called the solution fitness.

As described in [6], GA encodes the decision variables of the problem into strings of alphabets in certain cardinality. Strings that are candidate solutions to the problem are called chromosomes, the alphabets are referred to as genes. To evolve a good solution, an objective function is selected to be the relative fitness of the candidates. Best candidates will keep improving the solution while eliminating lower fit candidates. The main steps of this operation include: (1) *Initialization* The initial population of the candidates is usually random selection from the solution space. (2) *Evaluation* The fitness values of the candidates are evaluated. (3) *Selection* This step generates more copies of the best solution (high fitness). (4) *Recombination (Cross-over)* Parts of selected candidates are combined in pairs or more to create new solution. Since the best candidates are used, some of the new generation will eventually improve over the parents. (5) *Mutation* After recombination, parts of the created solution is randomly altered. (6) *Replacement* The new generation replaces the parents. (7) Steps 2-6 are repeated until some stopping criterion.

To map those steps to the problem of policy discovery, we need to define first the search space. A high level look at

the problem will intuitively search for the best policy that is closest to the original one. In other words, a policy that will match most of the samples generated. In this sense, the steps of the algorithm will be:

- 1) Start with an initial set of random policies.
- 2) Generate packets covering rules for each policy.
- 3) Probe the firewall and get the response.
- 4) Select the best policies (results conform to policy structure), and rerun the algorithm.

This is a straight forward application for GA to solve our problem. Unfortunately, this is hard to implement and deploy. The first limitation for this approach is the complexity for the candidate policies; policy size, rules structure and policy overlap. The number of possible policies to try is exponential in the number of variables; this makes it even harder to select the population size. Step 3 in this solution is the most expensive one, that makes the whole approach infeasible. Even if the policy complexity is known, and the population size can be approximated, step 3 will have to probe the firewall for each of those candidate policies. This increases the number of packets needed to discover the policy, which may trigger some defense mechanism at the network being probed.

A better use of GA in this domain is to improve the packet generation process. We need to intelligently generate packets depending on the previous response of the firewall. The packet is considered useful if it was accepted by the firewall. This should be reflected in the fitness value of the packet.

- 1) Start by generating an initial set of candidate packets.
- 2) Probe the set of candidates to the firewall and get response.
- 3) Evaluate current generation.
- 4) Cross-over.
- 5) Mutation.
- 6) Update the current reconstructed policy.
- 7) Repeat.

In our case each packet will represent a sample. The header values are the encoded gene string. The crossover is performed by exchanging bit values between candidates, to generate more fit samples. A random change in some of the bit corresponds to the mutation process. In discovering the acceptance region of a

policy, priority in candidate selection is given to the acceptance space. The fitness value of a sample will be high if the sample has a permit action.

Algorithm 1 PolicyDiscover_Genetic()

```

while Cost < MaxCost do
  Create_Generation(GENERATION_SIZE)
  EvaluatePackets();
  Cost += GENERATION_SIZE
  for all PKT ∈ Generation:PKT.sign==accept do
    RL = Create_Rule(PKT ±δ)
    for all i = 1; i < K; i ++ do
      pk = Generate_Packet(RL)
      if pk.sign == accept then
        RL = RL ∪ Create_Rule(pk ±δ)
      end if
    end for
    COST += K
    Discovered_Policy ← RL
  end for
end while

```

VI. RANDOM PACKET ANALYSIS

This module is responsible for generating random packets that are probable to have a chance being allowed by the policy to be discovered. Requests for random packets are sent to this module from the logic module, independent from the specific technique used.

Network Administrators have some tendencies and patterns in writing their firewall policies. However, there is no established study about these behaviors except that policies are always not perfect. Some of the policies might lack required rules for preventing well known vulnerabilities from being exploited inside their networks [5], [12]. Other problems arise from the policy structure, and rule-rule interaction [3]. In this work, however, we are focused on obtaining the firewall policy even if it does not contain any of the above mentioned problems; emphasizing on the fact that even perfect policies are not truly perfect if they are known to the attacker. Some of the administrators patterns and common practices will come in handy when selecting the ideal sampling strategy.

From observing many firewall policies as well as our experience in the field, we believe that the following are common practices that can be exploited to state our sampling guidelines:

- IP addresses are allocated for servers at first, and using the lowest IP values for the network management servers (*e.g.*, Gateways, DNS servers, etc). Following this small range comes the business/functionality servers (*e.g.*, web servers, DB servers, etc). Client IPs are either statically allocated from a medium to high range, or dynamically allocated in more of a high range.
- Ports used on all servers follow some common practices and well known values (aside from the standard port constants).

- The protocol field and the destination port, almost always come in pairs. In other words, mentioning a port value in the rule (*i.e.*, not an “any” value) means the protocol field will be specified as well.

Although this list is very far from complete, it can help us design better-than-random strategies to select our test packets. The following is the strategy used in kernel selection for each algorithm:

- A percentage is allocated to popular values. For example, standard port values, and 0, 1 and 255 values for bytes of the IP addresses. Of course these values are only selected when they fall within the area under consideration.
- The rest is distributed over the space, given a slight bias towards smaller values. For the Split-and-Merge algorithm, this also affects the splitting boundaries (*i.e.*, non-equal quadrants).
- Generating some of the samples to be in streaks, having consecutive values in one or more the dimensions. For example, generating consecutive packets with address and port equals (10.11.12.13:5430), and (10.11.12.14:5431). These lines in space help captures these rules that cover a simple line.
- With a lower ratio, some packets should be assigned to cover parts of the boundary of the space investigated.

VII. INJECTION AND REPORTING

To obtain the firewall response, the attacker should have an arsenal of tools (*i.e.*, firewall probing techniques) in order to have this hard-to-get piece of information independent of the firewall and target network configuration. These techniques are to be selected at run-time based on the type of entry we need to discover. In other words, discovering TCP entries can use different techniques than general packets. Also address types can affect our choice of probing technique; uni-cast, multi-cast or broadcast. A few techniques are known and many more are used by hackers for this purpose. For example:

- *Using ICMP Time exceeded (type 11, code 0)*. By sending packets with the required header info (*i.e.*, proto, src IP/proto, dst IP/proto), but with TTL value that is barely enough to go beyond the firewall. Either the firewall will drop the packet or a node behind it will send back a time-exceeded message showing that this entry is open.
- *Using ICMP Time exceeded (type 11, code 1)*. In many firewall configurations, fragments can pass more easily into the protected network based on the good-will assumption that the missing layer-4 information (in non-initial fragments) might cause the packet to be accepted. By sending just the non-initial fragment through the firewall, we can expect a node behind the firewall to send back a reassembly time exceeded message, indicating that fragments for this entry are allowed.
- *Using ICMP Unreachable error (type 3)*. This ICMP type contains many sub-types that indicate different reasons for the dropped packets. By trying to trigger each of the reasons, we will be able to identify whether the packet

went through the firewall filtering layer or not. In one version of the technique, the packet MTU is adjusted to be large enough to require fragmentation by the first node behind the firewall, with the “don’t fragment” bit set. Unlike other ICMP errors, most networks allow for these specific errors to flow out of the networks for diagnosis purposes. Another version is to try triggering the “source route failed” error code. This error code can be triggered if we included an invalid source route in the datagram options. However, many firewalls and administrators select to consider packets with options dangerous and drop them immediately. If a response is received in the form of ICMP unreachable message, this indicates the packet has passed the filtering phase (based on the policy or the use of ip-options).

- *Using an insider responder.* Another approach is to assume having a compromised machine directly behind the firewall that will perform sniffing over all packets passing through the firewall. The injection process will take place directly to the firewall itself. This pinger(injector)/ponger(sniffer) scheme is useful in reducing time taken to discover the policy, but it is the hardest configuration to deploy over the network. This technique might also allow for more accurate analysis of the probes, but is much less likely to use in practice because of the difficulty of sniffing backbone traffic in a switched network.

It is clear that the ICMP-based approaches are the least invasive, but they require that the firewall administrator enable (or forget to disable) ICMP responses in the firewall. However, it is very common to have sufficient ICMP types enabled that will enable one of the ICMP-based reporting techniques to work correctly. In the same time, they pose a limitation in the discovery process where the source IP of the sent probes must be the real attacker’s address. To have a better view of the policy that might use the source address in its filtering criteria, the attacker can utilize different machines (*i.e.*, zombies) in this process. In contrast with the pinger/ponger layout that can spoof addresses for sent packets and provide a much better coverage for discovering.

In the evaluation section VIII, we show that a small number of probes are needed to build a considerable part of the firewall policy. This result enables us to use multiple probes for every check. Some probes have a probability of having a false result (*e.g.*, timeout selected for response was not enough), but combining the result of multiple techniques can give much higher credibility in the final probe result. However, these multiple probes are used at wide intervals not to raise any suspicions at the attacked network.

VIII. ATTACK EVALUATION

The policies used to evaluate the attack were generated as described in [1]. This policy generation scheme is flexible, allowing the use of different policy structures. Policies are generated in different sizes, ranging in 10 – 1500 rules. For each policy the discovery process is applied using the three

space navigation methods, section V. The accuracy bound has been changed in the range 0.95 – 0.99. At each accuracy value, the number of packets/samples used for the discovery is calculated from equation 1 with confidence level of 95%. For each run of the experiment, the independent variables are policy size, and desired accuracy that was used to calculate the number of packets. The achieved accuracy is the actual accuracy resulting from using those packets in the navigation. Since we generate the policies, the achieved accuracy can be easily calculated from the difference between the original generated policy and the discovered one.

The implementation used Binary Decision Diagram to represent policies as boolean expressions. The operations are performed efficiently for each packet/sample. We used only the main 4 fields of the IP header (IP source/destination address, source/destination port). The protocol field was limited to TCP and UDP subspaces. This setting is more appropriate for attackers as the information about open TCP and UDP ports and machines is considered more valuable. In all the experiments, the attack accuracy is calculated by evaluating the intersection between the original policy and the discovered one, and normalize over the original policy. The number of satisfying assignments of the policy expression is used for the calculations, which can be efficiently obtained using BDDs.

A. Desired vs. Achieved Accuracy

In figure 6, achieved accuracies correspondent to the number of packets are shown for each technique. The theoretical upper bound on the accuracy as discussed in section III is plotted in bold for each chart. For each policy size 10 – 1500, the accuracy is evaluated at different maximum number of packets. As shown in the three figures, the lower number of rules per policy results in the least accuracy. This can be attributed to the fact that smaller policies have fewer number of patches in their accept space making it harder to discover using the initial random sampling.

B. Policy Size vs. Accuracy

Figure 7 compares the three techniques achieved accuracy versus the number of rules in the policy. The accuracy level increases for large policies with the three techniques performing almost consistently. For policies with less than 500 rules, the performance of the techniques varies. Thus the attacker should select the best technique based on his estimation of the policy complexity at the firewall.

C. Policy Area vs. Accuracy

The structure of the policy and the accept space area affect the accuracy as well. Policy area refers to the portion of the total space that the policy occupies; the accept space of the policy relative to the total space. This measurement is useful when the attacker has an estimate of the portions of the addresses used by the network. The area is calculated by dividing the number of satisfying assignments of the original policy over the total number of satisfying assignment of the whole space. The operation is performed on *log* scale for

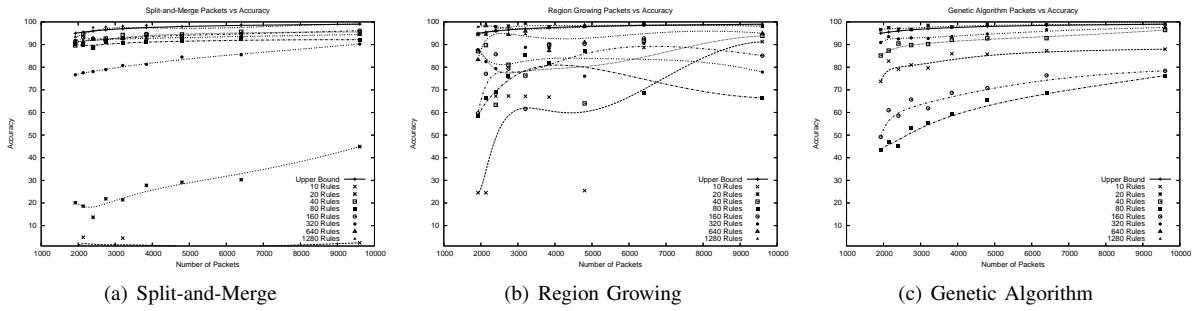


Fig. 6. Number of packets versus the achieved accuracy with different policy sizes.

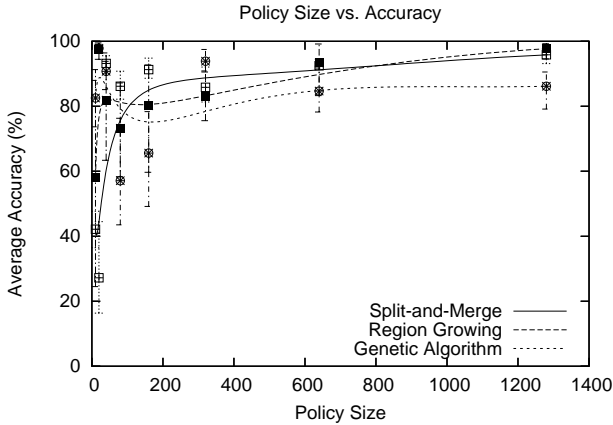


Fig. 7. Number of rules in the original policy versus the achieved accuracy for the techniques.

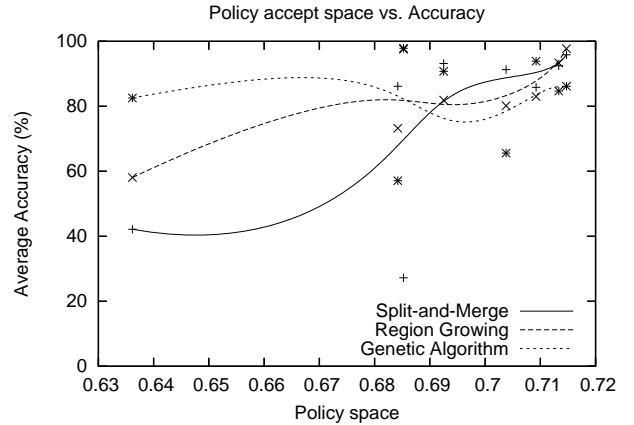


Fig. 8. The acceptance area of the original policy versus the achieved accuracy.

efficiency. In figure 8, the techniques are compared in the light of the selectivity of the policy in question. An overall trend shows that the larger the policy, the more accurate the discovery will be. The most sensitive technique to the policy accept area was the split-and-merge due to its homogeneous distribution of the discovery samples/packets.

In all of our experiments, the false negative errors were practically negligible (never exceeded 0.001%). On the other hand, the false positive errors were significant yet do not affect the applicability of the techniques. The false positive ratio is calculated as the part the acceptance space of the discovered policy that was not defined in the original one, using also the satisfying assignments count. In figure 9, we can see that false positive errors varied amongst the three techniques (1%–40%). However, for medium to large policies (> 20 rules) the false positive errors were almost never higher than 15%.

D. Results Summary

From the evaluation results, we can decide the best strategy to apply based on some expectations about the policy. For large policies, the split-and-merge algorithm outperforms both region growing and the genetic approach. This observation is attributed to the fact that all the space is used in testing the policy (splitting and merging). The other two methods depends on the initial set of packets, and are localized in space. The

genetic approach performs better for small policies, since it gives more priority to already discovered acceptance regions. Region growing algorithm has the worst average performance, however it is very efficient in learning exact rules since it can identify boundaries accurately. As a result, region growing can be used to discover the default deny rule better than the other techniques.

IX. CONCLUSION

This paper introduces a new kind of attack on network security devices especially firewalls. The attacker is shown to be able to probe the firewall by sending intelligently selected tailored packets in order to get an idea about the policy on the victim firewall. A simple framework structure is presented that should support future added techniques for policy analysis, packet selection and policy clarification. Three alternative techniques are presented for the packet selection/policy discovery module. The three techniques are based on the region growing, split-and-merge, and genetic algorithms. Evaluation of these techniques was performed and very promising results have been achieved with respect to accuracy as well as precision. In most cases, most of the accept space of the victim policy is discovered with accuracy more than 80-90%. Also, the discovered policy rarely included any false-positive area, with a precision of more than 90% in most cases. Taking into

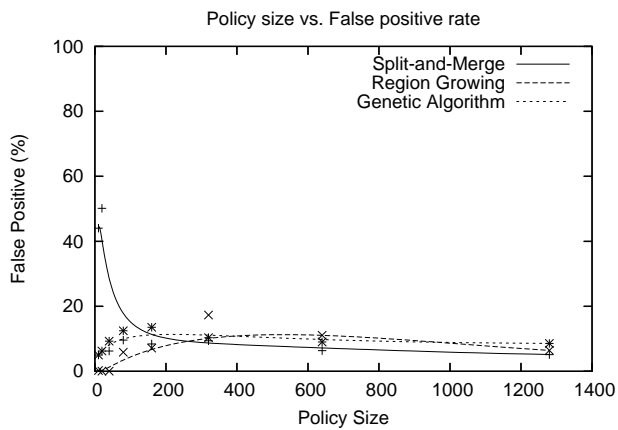


Fig. 9. The number of rules in the original policy versus the false positive ratio for the techniques.

consideration that these experiments were performed with less than 10,000 packets, this attack proves to be very promising or, in other words, very dangerous.

Limitations and Future Work

Knowing the firewall response needs more experimenting with stand-alone firewall boxes as well as host-based, software firewalls. Fingerprinting techniques need to be used to identify the best technique to be used for this specific firewall. An important alternative for ICMP messages for discovering the response is worm-based pinger-ponger pairs. Worms injected into the victim network can sniff packets coming out of the firewall and report back very accurate results. But the placement inside the network and implementation of such worms needs further work and planning.

Another important issue is discovering the implementation of the packet classification algorithm. Using accurate timing of packet delays can give a great deal of information into the internals of the target firewalls. At the least they can give us information about the actual rule order inside the policy. However, statistical matching and constant-time matching algorithms are getting more popular and they can make our task for measuring the timing harder. Also, they might call for even better placement of agents for injecting and reporting test packets.

REFERENCES

- [1] *An Automated Framework for Validating Firewall Policy Enforcement*. IEEE Computer Society, 2007.
- [2] Ehab S. Al-Shaer and Hazem H. Hamed. Firewall policy advisor for anomaly discovery and rule editing. In *IFIP/IEEE Eighth International Symposium on Integrated Network Management (IM 2003)*, pages 17–30, 2003.
- [3] Ehab S. Al-Shaer and Hazem H. Hamed. Discovery of policy anomalies in distributed firewalls. In *The 23rd Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOM 2004)*, 2004.
- [4] Martin Anthony and Norman Biggs. *Computational learning theory: an introduction*. Cambridge University Press, New York, NY, USA, 1992.
- [5] CERT/CC. Cert/cc, overview incident and vulnerability trends. April 2006.
- [6] David E. Goldberg. *Genetic Algorithms in Search, Optimization & Machine Learning*. Addison-Wesley, Reading, MA, 1989.

- [7] David Goldsmith and Michael Schiffman. Firewalking: A traceroute-like analysis of ip packet responses to determine gateway access control lists, <http://www.packetfactory.net/firewalk/firewalk-final.html>, October 1998.
- [8] Rafael C. Gonzalez and Richard E. Woods. *Digital Image Processing*, chapter 10, pages 615–617. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2nd edition.
- [9] Mohamed G. Gouda and Alex X. Liu. Firewall design: Consistency, completeness, and compactness. In *The 24th International Conference on Distributed Computing Systems (ICDCS 2004)*, pages 320–327, 2004.
- [10] Hazem Hamed, Adel El-Atawy, and Ehab Al-Shaer. Adaptive statistical optimization techniques for firewall packet filtering. In *The 25th Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOM 2006)*, 2006.
- [11] B. Hickman, D. Newman, S. Tadjudin, and T. Martin. Benchmarking methodology for firewall performance. RFC 3511 (Informational), 2003.
- [12] Ken Cutler John Wack and Jamie Pole. Guidelines on firewalls and firewall policy. NIST(National Institute of Standards and Technology), January 2002. Special Publication 800-41.
- [13] Seny Kamara, Sonia Fahmy, Eugene Schultz, Florian Kerschbaum, and Michael Frantzen. Analysis of vulnerabilities in internet firewalls. *Computers and Security*, 22(3):214232, 2003.
- [14] M. Kearns, M. Li, L. Pitt, and L. G. Valiant. On the learnability of Boolean formulae. In A. V. Aho, editor, *Proceedings of the nineteenth annual ACM Symposium on Theory of Computing, New York City, May 25–27, 1987*, pages 285–295, New York, NY 10036, USA, 1987. ACM Press.
- [15] Michael R. Lyu and Lorrien K.Y. Lau. Firewall security: Policies, testing and performance evaluation. *COMSAC. IEEE Computer Society*, 00:116, 2000.
- [16] Alain Mayer, Avishai Wool, and Elisha Ziskind. Fang: A firewall analysis engine. *SP '00: Proceedings of the 2000 IEEE Symposium on Security and Privacy*, 00:0177, 2000.
- [17] Ziv Nevo and Ran El-Yaniv. On online learning of decision lists. *J. Mach. Learn. Res.*, 3:271–301, 2003.
- [18] Ronald L. Rivest. Learning decision lists. *Mach. Learn.*, 2(3):229–246, 1987.
- [19] Taghrid Samak, Adel El-Atawy, Ehab Al-Shaer, and Hong Li. Firewall policy reconstruction by active probing: An attacker's view. In *The Second Workshop on Secure Network Protocols (NPSec 2006)*, 2006.
- [20] Voravud Santiraveewan and Yongyuth Permpoonanalarp. A graph-based methodology for analyzing ip spoofing attack. *AINA '04: Proceedings of the 18th International Conference on Advanced Information Networking and Applications*, 02:227, 2004.
- [21] Avishai Wool. Architecting the Lumeta Firewall Analyzer. In *Proceedings of the Tenth USENIX Security Symposium, August 13–17, 2001, Washington, DC, USA*, 2001.
- [22] Lihua Yuan, Jianning Mai, Zhendong Su, Hao Chen, Chen-Nee Chuah, and Prasant Mohapatra. Fireman: A toolkit for firewall modeling and analysis. In *2006 IEEE Symposium on Security and Privacy (S&P 2006)*, pages 199–213, 2006.