Abstract—In the past decade scanning has been widely used as a reconnaissance technique to gather critical network information to launch a follow up attack. To combat, numerous intrusion detectors have been proposed. However, scanning methodologies have shifted to the next-generation paradigm to be evasive. The next-generation reconnaissance techniques are intelligent and stealthy. These techniques use a low volume packet sequence and intelligent calculation for the victim selection to be more evasive. Previously, we proposed models for firewall policy reconnaissance that are used to set bound for learning accuracy as well as to put minimum requirements on the number of probes. We presented techniques for reconstructing the firewall policy by intelligently choosing the probing packets based on the responses of previous probes. In this work, we show the statistical analysis of these techniques and discuss their evasiveness along-with the improvement. First we present the previously proposed two techniques followed by the statistical analysis and their evasiveness to current detectors. Based on the statistical analysis, we show that these techniques still exhibit a pattern and thus can be detected. We then develop a hybrid approach to maximize the benefit by combining the two heuristics.

I. INTRODUCTION

Cyber threats have caused huge losses which are in billions of dollars [32]. Network reconnaissance is commonly used as a precursor step to gather the network information and plan attacks on network services and infrastructures. Scanning is a commonly used network reconnaissance technique for discovering potential vulnerabilities which then can be exploited to launch an attack. Different scanning techniques and strategies have been widely adopted by different threats to infect and propagate the network. Some threats use topological scanning technique in which the scanning is done according to the topology. However, some use vertical or horizontal scanning strategy in which either the range of ports or IPs is scanned. These scanning techniques probe each port or IP in the given range. Thus, they are not intelligent and scan each target in the given range or list. All of these scanning techniques and strategies have been widely discussed in the research community [7]. To cater for these scanning techniques, significant effort has been put in developing intrusion detection systems. These intrusion detectors have shown acceptable accuracy for detecting scanning probes [6]. However, new techniques were developed over the past recent years that are sophisticated; thus reconnaissance techniques have shifted its paradigm to the next generation. These next generation reconnaissance techniques trade accuracy for evasiveness and they discover the firewall policy structure.

The next-generation reconnaissance techniques have the following properties: (1) they are more intelligent as they do not scan the network randomly and select the next scan target based on the responses received, (2) they discover ranges or blocks such as firewall rules instead of each individual host, (3) they are stealthy since total volume of scan traffic being generated is considerably low as compared to previously used techniques because they do cater for denied packets and put an upper bound on the number of denied packets in a given range and (4) they scan the network in an intelligent fashion that even if the scan does not reach the destination, they still infer some information. Since the scan generation is done intelligently, they do not deviate much from the normal behavior thus avoiding detection. They put an upper bound on the number of probes being generated for a particular target space unlike traditional techniques which use to probe each host in the target space. They intelligently chose the next scan target based on the response from previous probe, thus reducing the total number of probes in order to avoid considerable deviation from the normal traffic behavior.

Main theme of next generation reconnaissance strategies is to learn by inference rather than discover by probing. Examples of next generation reconnaissance strategies are Split-and-Merge, Region Growing, Genetic Algorithm [27] etc. In this work, we investigate Split-and-Merge and Region Growing since they showed promising results for firewall policy discovery. Split-and-merge works in a fashion where it considers the address space as a two dimension plane. It splits the space into equal sized blocks for further investigation. It may split further and then it merges them back to identify the boundary or range implied in the network policy. Similarly, region growing picks a random point on a two dimension plane to find an accept space. Then it starts growing exponentially to identify the edge of the accept space/rule. The contribution of the work is twofold 1) we show that these next generation reconnaissance techniques are evade against the well-known intrusion detectors. Therefore, we conduct an analysis to discuss some features which can be exploited in designing a state-of-the-art detector for these techniques 2) Moreover, we propose a hybrid reconnaissance approach which leverages the two existing techniques and addresses the individual limitations of the constituent techniques.

Section II discusses the related work. Problem formalization
and theoretical foundation for the work is discussed in Section III. Existing next-generation reconnaissance techniques, which are under consideration, are discussed in Section IV. However, Section V discusses the easiveness of these techniques. Statistical analysis of these techniques and a simple countermeasure based on the analysis performed is shown in Section VI. A hybrid approach and its evaluation is presented in Section VII which leverage the existing techniques. In Section VIII we conclude our work.

II. RELATED WORK

Since the under consideration techniques discover firewall policy, we discuss its related work. To our knowledge, very little work was directed to attacking the firewall actively in order to obtain the employed policy. 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, which make such techniques less likely to be scalable or automated. In [9], a method similar to traceroute is introduced to scan a firewall for open ports in order to discover available hosts both behind the device and ACL filtering rules. It increases TTL value to probe individual hosts and ports in an incremental fashion. No intelligent packet selection is used in this case. This makes the approach easy to detect by simple scan detection algorithms. To help network administrators, [23] proposes a firewall analysis tool (FANG) which is concerned with the policy discovery using specific queries but it requires extra information about network topology and configuration. An extension to this work was introduced in [35] (Lumeta), where the query selection is automated but it still needs topology and routing information.

Since this work presents the blind discovery of policies using modeling techniques, we discuss the existing policy modeling work. Techniques proposed in [1], [2] and [36] aim at discovering conflicts in policies due to misconfiguration e.g., rules that contradicts or shadow each other. In [1], conflicts are classified and analyzed for a single firewall as well as distributed environment. Another approach to design firewall policy has been proposed in [11]. This work provides 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 [22], [13]. Some work has been done where the analysis was performed to test the firewall for vulnerability to traffic specific attacks such as IP spoofing attacks which were addressed in [30]. In [16], performance metric for vulnerabilities resulting from firewall operations are presented and analyzed.

Blind discovery of firewall policies was introduced in [29]. Two approaches were proposed to select probing packets and preliminary results were shown. This work aimed to be a proof of concept on the feasibility of such discovery. A more elaborate discussion of the approach is presented in [27], where three heuristics were proposed and evaluated against multiple policies.

Firewall(s) policy can be formulated as a boolean function specifically a decision list, as discussed in Section III. Boolean function learning has been studied widely in the literature [5] and [26]. Most of the work in this area is based on the assumption that the samples are given or randomly chosen from a known distribution. In this work, we make use of the policy properties to dynamically choose samples in order to learn the function. Efficient algorithms for online learning of the decision list are proposed in [24]. However, the samples used are also known but processed one by one. An approach for the guided learning of the boolean function was proposed in [33] where examples were generated according to the current hypothesis to improve the learning. The algorithm maintains two hypotheses, one is the learned function (so far) consistent with the current set of examples, and other is consistent with the negative of the examples. The idea is to generate the next example to satisfy both hypotheses and update according to the correct classification. This approach is not efficient since it mainly relies on generating a satisfying assignment to both the hypotheses.

Decision list learning was first discussed and proved to be Probably Approximately Correct (PAC) in [26]. Other techniques were proposed addressing the attribute efficient learning for Decision List, where the number of attributes contributing to the example classification are limited to a constant [19], [21]. The run time of these approaches is exponential in the number of relevant attributes limiting the applicability of the algorithm for large cases.

III. PROBLEMORMALIZATION

In this section, we investigate the reconnaissance of firewall policies. The theoretical foundation will give estimates to the number of samples needed to discover a certain firewall policy as well as put an upper bound on the performance. A theoretical bound for the number of packets needed to discover the policy given a certain accuracy can be estimated by modeling the 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 [26] is a list \( L \) of pairs: \((f_1, v_1), (f_2, v_2), \ldots, (f_r, v_r)\), where each \( f_i \) is a term in \( C_k \), each \( v_i \) is a value in \( \{0, 1\} \), and the last function \( f_r \) is the constant function: true. \( C_k \) is the conjunction of \( k \) literals from the total \( n \) literals. 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 \( f_j \) 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. Network policies employed in firewall(s) can be modeled as k-Decision Lists, and hence the learning of the policy can be proved. Considering the formal definition of a
policy from [2]; a policy is a set of filtering rules that control the action in response to incoming traffic. An access policy, $P = R_1, R_2, \ldots, R_n$, 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, $X = x_1, x_2, \ldots, x_k$, 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 constraint on the filtering fields that must be satisfied for the action $act_i$ to be triggered. In the network policy employed in firewalls, the set of possible actions, $A$, has only two actions: $<permit, deny>$, that can be encoded as one boolean variable $\{0, 1\}$, where $permit \equiv 1$ and $deny \equiv 0$. The rules of a policy are matched in order against the incoming packets. The response of the policy 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 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 policy in response to satisfying $C_i$ is mapped to $v_i$, the value paired with the function.

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 corresponding binary bit value. The space of the packet header becomes the boolean function sample space. The function value is the action of the firewall filtering a specific combination of field values. The goal of our approach is to learn the value of this function for all packet space. The following two sections will describe the learning models for boolean functions that have been adopted in the procedure of learning decision lists.

\[\begin{array}{c|c|c|c|c} & R_1 & R_2 & R_3 & \hline f_1 & \text{true} & \text{false} & \text{true} & 0/1 \\
& \text{true} & \text{false} & \text{true} & 0/1 \\
& \text{true} & \text{true} & \text{true} & 0/1 \\
& \text{false} & \text{false} & \text{false} & 0/1 \end{array}\]

Fig. 1. Decision List operations

A. PAC Model

The Probably Approximately Correct (PAC) model deals with the offline learning of boolean functions from examples. In this model, the aim is to find an approximately correct hypothesis after seeing a random sample of classified instances. Let $A$ be the learning algorithm, $H$ is the hypothesis space, and $f \in F$ is the target function. $A$ is said to PAC-learn $F$ by $H$, if for any distribution, and any error parameters $\varepsilon$ and $\delta$, $A$ runs at time at most 

$$m > \frac{1}{\varepsilon} \left( \ln(|H|) + \ln\left(\frac{1}{\delta}\right) \right)$$

(1)

where, $|H|$ is the cardinality of the hypothesis space (the total number of possible functions), $\varepsilon$ is the accuracy parameter (the maximum error between the original policy and the discovered one), and $\delta$ is the confidence parameter (the probability that the learning algorithm produces an accurate policy must be at least $1 - \delta$). For example, if a policy has 2 two rules, each allowing 2 ip addresses and 2 ports, then the cardinality is 8. Given the accuracy and confidence parameter, the bound can be calculated. For a k-decision list, this relation is valid only for $1 < k < n$. The number of variables defining each term is constrained to be strictly less than the total number of variables. An algorithm proposed by Rivest in [26] is tailored to the online learning of k-DL. The assumption made by this algorithm is that the number of levels of the list is fixed.

Policy Learning under the PAC model: The $1 < k < n$ constraint prevents any policy 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 e.g., network address ranges instead of single IP. An actual policy might contain exact/specific rules, having all the bits mentioned. Since the major parts of the policy will be in the range form, our heuristic (approximation) will ignore the exact rules. Another problem with using the PAC directly is the large number of examples needed to satisfy those error bounds. The minimum number of examples required to learn the decision list according to the PAC model is very large in the case of policies. It is exponential in the number of attributes, which in this case the total number of bits in each packet header. This number can be reduced significantly given the fact that we are trying to generate samples adaptively based on the classification of the decision list for previous samples. Using optimal encoding of the policy results in the reduction of the total hypothesis space, $|H|$, to a logarithmic bound.

B. Mistake Bound Model

This model is better suited for the interactive learning of boolean functions. The online nature of the learning under this model makes it directly applicable to our case of adaptive learning. As in the PAC model, the mistake bound model is used to measure the performance of a learning algorithm, $A$. The learning procedure can be viewed as a sequence of trials performed by the algorithm against an adversary, [24]. The learner keeps on enhancing its current hypothesis, $h$ according to the adversary responses. The learning process terminates when, $A$ makes a specific number of mistakes. The steps involved in trial $t$ of the algorithm are:

1) Adversary picks an instance $x^{(t)} \in X_n$
2) $A$ outputs $h(x^{(t)})$, where $h : X_n \rightarrow \{0, 1\}$
3) The adversary reveals the correct classification of the instance, $c(x^{(t)})$
4) If $h(x^{(t)}) \neq c(x^{(t)})$ then a single mistake is incurred by $A$
5) $A$ can update its hypothesis

The mistake bound, $M(c)$, for $A$ on $c$ is defined as: if for any sequence of instances (possibly infinite), $A$ never makes more
than $M(c)$ mistakes while learning $c$. A learns $c$ within the mistake bound model if, $M(c) = \text{poly}(n, \text{size}(c))$, and if its running time during each trial is also $\text{poly}(n, \text{size}(c))$, where $n$ is the number of attributes and $\text{size}(c)$ is the size of $c$ in bits.

**Policy Learning under the Mistake Bound model:** This model provides a framework for our learning strategy, where the adversary corresponds to the firewall response to our generated packets. Algorithms developed for online learning of decision lists within the mistake bound model (as in [24]) provide a low mistake bound in terms of relevant attributes. The bound is $O(r^{2D}\log(n))$, where $r$ is the number of relevant attributes, and $D$ is the length of the decision list. For policies employed in firewall(s), those parameters have high values (maximum of 104), which will explode the running time. A slight modification of this model using adaptive selection of examples for step (1) will limit the number of mistakes encountered by the algorithm.

**C. Combining Models**

For our application, we combine both the models together as shown in [27]. We will use the number of examples calculated in the PAC model (equation 1) to learn the policy using the online approach of the mistake bound model, with adaptive selection of examples. The total number of packets, $m$, calculated by equation 1 guarantees that the resulting policy will be within a bounded error from the original one, $\varepsilon$. The use of this value, $m$, provides a stopping criteria for our heuristics in order to not to run indefinitely. The mistake bound analysis comes in the design of our approaches, where we aim to choose the samples that will limit the number of mistake encountered during the learning process.

**IV. EXISTING NEXT GENERATION RECONNAISSANCE TECHNIQUES**

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

The only primitive operation that the attacker uses is sending a packet and waiting to see whether it is going to be allowed or not by the employed policy, and generating a random packet. 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 (header fields such as IP/port) being investigated. 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. There are different ways to infer the feedback. First it can be a firewall response, however, it is mostly abandoned in practice. Second, if the probe reaches the destination, in case of TCP, it will get a feedback. Third, there can be a machine behind the firewall which will observe which probes have been passed through and notify. This will be able to provide feedback for UDP and other protocols as well. Third scenario has been widely adopted in firewall fingerprinting for different purposes [18].

In Algorithm 1 we simply start with the network-wide limits as the default sampling space, which represents the “default deny” rule. The value of $-1$ assigned to the sign parameter indicates the deny action as mentioned before. The maxCost parameter indicates the maximum number of packets the algorithm should use. After calling the recursive function, the policy will be simplified by any of the already available rule simplification procedures as the ones mentioned in [11]. This algorithm summarizes the functionality of the logic module. The recursive function can be replaced by any of the proposed discovery approaches. maxCost will be calculated based on the theoretical bound estimates discussed before.

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. Details of these techniques can be found in [27].

**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 employed policy 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 indicates the end of the rule in that dimension. Following the exponential search comes a binary search to pinpoint the exact boundary of the rules space (See Figure 2). These two steps take only $O(\log n)$, where $\log n$ is the number of bits in each field (i.e., IP=32, port=16, protocol=8 bits).

Once the boundaries of the rule are identified, it recursively start sampling inside the rule searching for any rules that are exceptions to this rule (i.e., searching for a new kernel). This

---

**Algorithm 1: Extract_Policy(maxCost)**

1. $\text{sign} = -1$, default is deny;
2. $P \leftarrow \text{Extract_Recursive}(\text{global bounds}, -1, \text{maxCost});$
3. Apply Policy Simplification Techniques on $P$;
4. return $P$;

Fig. 2. Searching for rule boundaries starting from a kernel
is the same operation performed previously as we searched for this rule as an exception of the “default deny” rule, and so on.

B. Split-and-Merge Approach

In this method, we use the split-and-merge algorithm that is used for image segmentation [10]. 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) \( R_1, R_2, \ldots, R_n \), 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 network policy employed in 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 of the split-and-merge algorithm in the 2-dimensional case:

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. For testing the predicate \( P(\ldots) \), 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 further 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. Figure 3 illustrates three iterations of the algorithm on a two-dimensional space.

V. ARE THESE TECHNIQUES REALLY EVASIVE?

A great effort was invested in the area of intrusion detection specially addressing scanning detection [4]–[12]. However, next-generation reconnaissarce can be viewed from a network administrator perspective as a stealthy scanning. According to [31], scanning activities are divided into three categories; horizontal, vertical, and block scans. These techniques fall in the block scan type, where multiple fields are considered in generating scanning probes. These type of scans are generally harder to detect.

Statistical scan detectors depend mainly on the assumption that the normal traffic can be characterized by a known probability distribution. The Spice framework [31], also known as SPADE detector uses information measure to detect anomalous traffic, then a correlation analysis for detected events is performed to group similar scanning activities for future reference. Following is the list of heuristics used for the correlation and how these techniques might overcome the measures:

- **Feature Equality**: If the features are equal, the relation between two detected events will evaluate to full connection. It could overcome by not sending similar field values unless previous ones have timed out.
- **Feature Proximity**: The proximity could be in time, or the value itself. It can be avoided by sending closely located packets separated by big time interval.
- **Feature Covariance**: If the rate of change of field values is the same between two events, then they belong to the same scanning group. In region growing scheme, it increase field values exponentially, which might link packets at the same steps in the search together.

Most of these approaches depend on the assumption that the time interval between packets is small enough to be detected before the time out of the event. A large time interval compared to the time out of SPADE can be used.

Another variation of statistical portscan detection is based on the work in [15], where sequential hypothesis testing is used. Their developed algorithm (TRW, Threshold Random Walk) test the two hypotheses of a remote host to be scanner or benign. The independence assumption in the model analysis limits its applicability in our case. Our approach has the flexibility to generate alternating packets from different partitions resulting in the same discovery. This oscillation corresponds to the worst case limitations for the TRW. Other measures have been proposed in [17] that characterize the detectability of scanning mechanisms. The main two parameters are \( (\alpha, T) \), the number of source IPs used in the scan and the time to cover the targeted address space. For any scanning approach those measures need to be estimated with respect to the total address space as well as the number of active hosts. The analysis is performed in the context of TRW parameters, where hypothesis are defined depending on a source being benign or malicious.

Similarly, Maximum Entropy [12] is also a well known detector in the research community. To detect anomaly, it divides the observed traffic into multiple classes based on protocol and destination ports. Each class contains a range of destination ports and protocol. It works in a sliding window fashion, so in-order to detect an anomaly, any class should perturb more than a threshold value \( \nu \) for \( x \) number of times in a given time window length \( t \). It can be intuitively argued that maximum entropy will not be able to detect it since these techniques has an upper bound to generate number of scan packets in a given patch i.e., class, thus avoiding detection.

We show the experimental results of TRW and Maximum Entropy detectors on these techniques i.e., split-and-merge
and region growing. Both the detectors failed to detect the reconnaissance probes. TRW calculates the likelihood ratio for each connection attempt and classify them as anomalous if the likelihood ratio for any host increases more than a threshold. Since the hosts were also generating benign traffic, the likelihood ratio did not cross the threshold set. Similarly, we used the basic principle of Maximum Entropy and trained it on the benign traffic to build a normal/benign distribution which is then used to compare with the run-time distribution using KL-Divergence measure. If the deviation is more than a particular threshold value, it raises an alarm. Please note that the thresholds used for the experimentation purposes are the same which provided acceptable accuracy in the recent literature [3], [6].

Figure 4(a) shows that TRW failed to detect the split-and-merge traffic in a malicious time window. It can be clearly observed that the likelihood ratio did not exceed more than the threshold value. However, it perturbed but not enough to exceed the threshold value. Similarly, Figure 4(b) shows the threshold value observed in a 60 sec time window for maximum entropy. Here the limit on exceeding the threshold value was 30 times in a 60 second window and threshold value used was 15 which is shown by a solid red line. However, it can be observed that the run-time distribution deviated more than the threshold only 10 times. The main reason for deviating only 10 times was the slow scan rate introduced by the split-and-merge strategy. It did not deviate the run-time distribution for 30 times in a 60 second time window, thus avoiding detection. Therefore, just by slowing down or the affect of averaging out in the normal traffic, split-and-merge strategy was able to go evasive and undetected in-case of maximum entropy. Moreover, it generates traffic which falls into different classes of maximum entropy. Thus, the split-and-merge was evasive to maximum entropy detector. Similar results were observed for Region Growing reconnaissance. It is clear from Figure 5 that both the detectors failed to detect the reconnaissance. Figure 5(a) shows the reconnaissance time window for TRW on Region Growing probes. Perturbations in score are very similar as compared to TRW for Split-and-Merge in Figure 4(a). However, maximum entropy on region growing reconnaissance observed higher scores which can be observed in Figure 5(b). We found out that few hosts were generating traffic for classes which were not present in the benign profile, thus causing a higher deviation. However, Region Growing still go undetected since it did not perturb more than 30 times in a 60 sec time window.

VI. STATISTICAL ANALYSIS OF EXISTING TECHNIQUES

In this section, we show the statistical analysis of the traffic generated by these techniques i.e., Split-and-Merge and Region Growing. We show some trends observed which gives us interesting insight and can be used to further improve the techniques or build a detector for the techniques. We assume that the policies have the following properties 1) policies have a default deny rule 2) policies have rules in the form of ranges 3) policies have an accept space. Based on the analysis, we show that the generated traffic exhibits some pattern, though it is random in some sense but it can be predicted using information theoretic measures for the next time window, thus resulting in its detection.

A. Packet Count Statistics

Next generation scanners are smart enough and don’t perform scanning randomly. However, they have a mechanism to decide intelligently, with some randomization factor, for the next scan target. Decision is based on the response from the previous scan. Their ultimate goal is to generate as less traffic as possible with a low scan rate to stay undetected. However, at the same time they try to get the maximum information about the firewall policy. Due to lesser number of probes, each single scan reveals some important information and plays an important role in decision making for selecting the next scan target.

Table I shows the packet counts generated by split-and-merge in multiple experiments. We do not report packet count statistics for Region Growing since they are lower than split-and-merge due to the fact that they select scan targets exponentially in target space in order to identify the boundary of accept space. However, split-and-merge does not grow the region exponentially and tends to divide patches into smaller patches for precise identification of the boundary. First column represents the number of IPs in the target space followed by the number of ports in the second column. Therefore, combination of space IPs and space ports results in the total

<table>
<thead>
<tr>
<th>Space IPs</th>
<th>Space Ports</th>
<th>Scan Space</th>
<th>Pkts Gen</th>
<th>Pkts Denied</th>
</tr>
</thead>
<tbody>
<tr>
<td>256</td>
<td>1000</td>
<td>256000</td>
<td>686</td>
<td>4</td>
</tr>
<tr>
<td>512</td>
<td>1000</td>
<td>512000</td>
<td>181</td>
<td>10</td>
</tr>
<tr>
<td>768</td>
<td>1000</td>
<td>768000</td>
<td>256</td>
<td>56</td>
</tr>
<tr>
<td>1024</td>
<td>1000</td>
<td>1024000</td>
<td>107</td>
<td>20</td>
</tr>
</tbody>
</table>
scan space available, presented in the third column, to the split-and-merge strategy to explore. Last two columns show the packets generated by the approach and denied by the policy, respectively. Please note that the maximum depth (levels to divide the space) allowed for splitting the available space used was 5. However, accept space (network policy) size was 128000, which is a combination of 128 IPs having 1000 ports open on each of them. It can be intuitively thought of as two dimension space having ip addresses on one dimension and ports on the other dimension. Please note that the accept space is a subset of the total scan space available to split-and-merge strategy to explore in a network.

It can be observed that the packets generated for each scan space is very less as compared to the total space. For all of these experiments the accept space of a policy was fixed. It can be observed that bigger the total scan space, the lesser the number of packets generated. It is because bigger available space will have bigger blocks which will have the deny space in most of the blocks and these blocks will not be explored upto the maximum depth rather they will be marked as denied space after reaching maximum bound limit for each block at the first depth. Moreover, the accept space will tend to fall in lesser blocks, thus resulting in lesser packet generation for scan. It can be stated, since the accept space size is fixed and scan space is increasing, the percentage of accept space is decreasing overall with respect to the total scan space. The small available space with the same sized accept space will have accept space distributed in multiple blocks since the space is always divided into 4 blocks and each block will be further divided into 4 blocks. Therefore, split-and-merge will reach maximum depth in more blocks, thus resulting in higher number of packets generation to reveal the policy.

### B. Space Dependency

Another relevant property which gave us the interesting insight is the Euclidean distance. In case of split-and-merge approach, it is intuitive that as long as the scan packets fall in the accept space of a policy, the split will cause a shrink in the distance between the subsequent scans. However, in case of scan falling in the deny space, it will keep trying until an accept space is found in the same or other block until an upper bound is reached. However, in case of region growing, distance of subsequent scans will increase since it exponentially traverses the space to identify the boundary when the probes are falling in the accept space. If the probe falls into the deny space, it will probe in the other dimension of the same kernel, thus reducing the distance since it starts from kernel. Therefore, a pattern can be observed in case of region growing which shows exponential increase in distance and then reduction as well.

Euclidean distance measures the distance between the two points in a metric space. Since in our case its a two dimension space, destination ip and port for the target, the distance between the two given points $p$ and $q$ in a two dimensional space can be calculated using formula:

$$d(p, q) = \sqrt{(p_1 - q_1)^2 + (p_2 - q_2)^2}$$

However, for the purpose of space dependency over a period of time for all scan targets, we are interested in the average distance for all the points at a given time. Therefore, we calculate the average distance for all the subsequent scans at a given time. Figure 6 shows the average distance over time for each scan at a given time. We run the experiment for three different parameters for split-and-merge. The scan rate was kept fixed i.e., 50 scans per minute since the traffic is not mixed with any dataset, therefore, it does not affect the analysis. For figure 6(a), maximum split depth of 5 was allowed. However, the accept space size with reference to the total scan space was set to 20%. Similarly, maximum split depth of 3 and accept space size of 20% was used for Figure 6(b). Third sample was generated using depth limit of 7 and accept space of 30%. It can be observed that all the three experiments exhibit a decaying trend over time, thus the average distance between subsequent points is getting shorter. This gives us the insight that split-and-merge ultimately converges to a small block to reveal the boundary precisely, up-till a maximum bound on the depth i.e., how small an address range can be. However, few perturbations can be observed in the average distance. They were caused since it hits the deny space blocks as well. Therefore, average distance may increase in a small time window, but overall it shows a decreasing trend upon finding the accept space. On the other hand, region growing shows an exponentially increasing trend upon falling into the accept space as shown in Figure 6(d). For three different samples, different accept space size were used. However, sample space was kept the same. Higher the accept space to sample space ratio, the distance increases exponentially. On the contrary, distance is decreased when the probe falls in the deny space since it restarts the exponential increase in other dimension, thus increasing the distance again, as shown in Figure 6(d).
C. Temporal Dependence

Temporal dependence is another relevant property which we analyzed. It can be intuitively argued that subsequent scan packets will exhibit certain level of temporal dependence since the next scan target is decided based on the response of the previous scan. Therefore, subsequent packets would be somehow correlated to each other. It can serve as an important metric in statistically analyzing the behavior.

Autocorrelation measures the on-average temporal dependence between the random variables in a stochastic process at different points in time. For a given lag \( k \), the autocorrelation function of a stochastic process \( X_n \) (where \( n \) is the time index) is defined as:

\[
\rho[k] = \frac{E\{X_0X_k\} - E\{X_0\}E\{X_k\}}{\sigma_{X_0}\sigma_{X_k}},
\]

where \( E\{\cdot\} \) represents the expectation operation and \( \sigma_{X_k} \) is the standard deviation of the random variable at time lag \( k \). The value of the autocorrelation function lies in the range \([-1, 1]\], where \( \rho[k] = 1 \) means perfect correlation at lag \( k \) and \( \rho[k] = 0 \) means no correlation at all at lag \( k \).

We used the same samples generated for average distance analysis in the previous section. Figure 7 show the autocorrelation function versus the lag for scanning probes. For all the three samples of split-and-merge shown in Figure 7(a), (b) and (c), it can be seen that initially they were anti-correlated. However, in general the correlation exhibit a growing trend for all the lags, which is of our interest. However, in two cases, i.e., Figure 7(a) and (c), the correlation drops after certain lags and starts growing again, though the overall linear trend (shown by solid line) shows the growing trend over all. Upon discovering one accept space, split-and-merge splits it further to identify the boundary or edges, thus causing autocorrelation to grow. However, when the accept space in this region is completely identified, it moves to the other blocks to look for any other accept space, thus resulting in the drop of autocorrelation coefficient. However, as soon as it finds the block having accept space, it again starts discovering the edges or boundary which again results in the growing trend of autocorrelation.

Similarly, it can be observed in Figure 7(b) that split-and-merge does not exhibit any drops since the accept space was not multiple in the overall scan range. Therefore, upon discovering the particular range, it tries to identify the edges which exhibit a growth in the autocorrelation trend. Figure 7(d) shows the auto-correlation trend for region growing on three samples. It can be observed that all the three samples show no correlation or anti-correlation (close to 0 or \(-1\), respectively) initially and then the correlation tends to increase. However, after the few lags it decreases again due to the fact that region growing starts probing in a different dimension. The probes belonging to the same dimension and falling into the accept space exhibit the increasing trend in auto-correlation. Therefore, the oscillatory trend is observed for region growing samples.

D. Modeling Dependence

Since the subsequent scans show some dependency on each other, which is shown by the decreasing trend of average distance and an increase in their correlation. It can be intuitively argued that subsequent scan traffic shares some information with the already seen scan traffic. Therefore, in-order to verify that after how many scans most of the information about a subsequent scan is already there, we calculate the conditional entropy over different markov chain order. It can be rephrased as: what is the order of markov chain model that should be used to predict the subsequent scan. We use the conditional entropy measure [34] to calculate the amount of information given at each markov chain model.

To identify the order of correlation presence in the reconnaissance probe generation random process, we define a markov chain based stochastic model as follows: Let the scans at discrete time instance \( n \) represent the realization of a random variable derived from a stochastic process \( X_n \). This process is a markov chain if it satisfies the markov property, which is defined as:

\[
\Pr \{X_n = j \mid X_{n-1} = i, X_{n-2} = i_{n-2}, \ldots, X_0 = i_0\} = \Pr \{X_n = j \mid X_{n-1} = i\} = p_{ji}.
\]

Equation 4 shows that the probability of a next state is only dependent on the current state. Therefore, we can define a markov chain model \( X_n \) for scans by combining their destination ip and ports, thus revealing a unique number for each destination ip and port pair. We name this pair as ‘score pair’ for the ease of discussion. We divide all possible values of the score pair in multiple non-overlapping bins. Each bin then represents a state of the markov chain, while the set of all bin indices \( \psi \) is its state space. Based on this state representation, we can define a 1-st order markov chain \( X_n^{(1)} \), in which each bin represents a state of the random process. The transition probability matrix of the 1-st order markov chain \( P^{(1)} \) can be computed by counting the number of times state
i is followed by state j. The resulting $|\psi(i)|$ histogram can be normalized to obtain the state-wise transition probability mass functions as the rows of $P(1)$.

We can find the conditional probability of the 1-st order markov chain as:

$$H(1) = \sum_{i \in \psi(1)} \pi_i^{(1)} \sum_{j \in \psi(1)} p_{j|i}^{(1)} \log_2 \left( p_{j|i}^{(1)} \right),$$

(5)

where $\pi_i^{(1)}$ is the average probability of being in state $i$, which is computed by counting the total number of times each state is visited and then normalizing this frequency histogram.

Here $H(1)$ defines the amount of average information which is not provided for score pair $X_n$ when it is predicted using the score pair $X_{n-1}$ which was already known. Therefore, if score pairs are not correlated, $H(1)$ will assume a larger value. It means $X_{n-1}$ did not provide much information about $X_n$. Therefore, if $H(1)$ assumes a larger value, we can use multiple score pairs to predict the next score pair. It can be modeled by increasing the order of markov chain i.e., $l = 1, 2, \ldots$. Then the random variable will become $X_n^{(l)}$, in which each state is an $l$-tuple $< i_0, i_1, \ldots, i_{l-1} >$. This $l$-tuple will be used to predict the next score pair. Hence a transition probability matrix $P^{(l)}$ can be computed by counting the number of times $< i_0, i_1, \ldots, i_{l-1} >$ is followed by state $< i_1, \ldots, i_{l-1}, i_l >$.

The conditional entropy of $X_n^{(l)}$ defined on $\psi^{(l)}$ can then be computed using the same method as Equation 5. Therefore, if $H(1) \geq H(2) \geq \ldots \geq H(l)$ trend is observed, it reveals the order at which most of the information about next score pair is already given by the previous score pair.

Figure 8 shows the conditional entropy over different markov chain order. It can be clearly observed that all the three samples of split-and-merge in Figure 8(a), (b) and (c) and region growing in Figure 8(d) exhibit a decaying trend. The decrease of conditional entropy over a higher order of markov chain model reveals that most of the information about the next traffic is already given by the previously seen traffic. Therefore, it can be stated as, markov chain order reveals the next traffic is already given by the previously seen traffic. The conditional entropy has dropped to a very low value i.e., less than 0.1 when the order of markov chain is 11.

E. Countermeasure

Based on the statistical analysis done in the last section, we now propose a simple markovian based detector which takes into account the temporal and spacial dependency. The markovian algorithm proposed is basically a variant of the stochastic target tracking algorithm [14].

We model the address space into two dimensions. Where one dimension represents the destination IP address and the other represents the destination port number. Therefore, intuitively it can be said that each probe can be represented as a tuple of destination ip address and port number. The total maximum space can be $2^{32}$. However, the individual network to secure would be a small subset of it, since the address range of a network would be known. We then divide the network address space into $k$ equal sized bins. The number of bins can be calculated as a bi-product of conditional entropy over markov chain model order where it goes minimum. However, the size of each bin can be calculated by dividing the address space by $k$. The size of first and last bin is kept flexible in-order to accommodate the unexpected scan traffic.

Let $\varepsilon(n)$ denote the average euclidean distance and $P(n)$ represents the transitional probability matrix of $k \times k$ dimension at time $n$, where $P_{ij}^{(n)}$ represents the entry at $i^{th}$ row and $j^{th}$ column. Also let $r(n)$ denote the observed value and $\hat{r}(n)$ be the predicted value. Then the algorithm operates as follows:

$$\varepsilon(n) = \frac{\sum_{i=1}^{n} \sqrt{(\sigma_i - \sigma_{i-1})^2 + (\phi_i - \phi_{i-1})^2}}{n}$$

(6)

$$\hat{r}(n+1)_{j|r(n-1)} = \frac{\hat{p}_{j|r(n-1)}^{(n+1)}}{\sum_{i=1}^{k} \hat{p}_{i|r(n-1)}^{(n+1)}}, \forall j = 1, \ldots, k,$$

(7)

where $\beta \propto \frac{1}{\varepsilon(n)}$ for Split-and-Merge and $\beta \propto \varepsilon(n)$ for region growing

$$\hat{p}_{j|r(n-1)}^{(n+1)} = \max_{j=1, \ldots, k} P_{j|r(n)}^{(n+1)}$$

(9)

Equation 6 calculates the average euclidean distance $\varepsilon(n)$ for the traffic observed till time interval $n$, where $\sigma_i$ represents the destination ip and $\phi_i$ represents the destination port for the interval $i$. However, if the average euclidean distance decreases or remains the same, equation 7 proportionally increases the weight of the probability of seeing $r(n)$ after $r(n-1)$ in a transition probability matrix $P(n)$. On the other hand, if average euclidean distance increases, the probability of seeing $r(n)$ after $r(n-1)$ in a transition probability matrix $P(n)$ decreases because $\beta$ is inversely proportional to the average distance in case of split-and-merge. However, for region growing $\beta$ is directly proportional to the average distance since it increases.
Therefore, equation 7 proportionally increases the weight of seeing $r(n)$ after $r(n-1)$ when the average euclidean distance increases. Since each row of transition probability matrix is a probability mass function $pmf$, i.e., the next traffic may belong to any possible states based on the probabilities, equation 8 updates the transition probability matrix to $p^{(n+1)}$ for the next time interval. Lastly, equation 9 is used to predict the next bin/range to which traffic/scan would belong for time interval $n+1$.

For the evaluation of the simple countermeasure shown above, we used the network traces of Lawrence Berkeley National Laboratory (LBNL) dataset [20], [25]. This dataset was collected at two international network locations at the Lawrence Berkeley National Laboratory (LBNL), USA. The main applications in internal and external traffic were Web (HTTP), Email and Name Services. More than 5000 hosts were communication with more than 10,000 remote hosts. Background traffic rate varies from 3.5pkts/sec to 243pkts/sec. We generated traces of both the reconnaissance techniques i.e., split-and-merge and region growing for different accept space sizes i.e., 10%, 20% and 30%. Multiple instances were generated for each accept space size and mixed in the benign trace of LBNL dataset. Detection results achieved using the simple countermeasure algorithm are shown in the Figure 9. It can be observed from Figure 9(a) that higher the accept space, the easier it is to detect the split-and-merge reconnaissance. Similarly, Figure 9(b) shows the detection trend of region growing reconnaissance. However, detection for 10% accept space was higher than 20% initially but it changed later on. The reason is that region growing randomly selects the kernel in the initial phase and later on it identifies the correct boundary. Therefore, the later phase is more intuitive and initially it can be a little random.

VII. THE HYBRID APPROACH AND ITS EVALUATION

In this section, we show how a hybrid approach can leverage the existing reconnaissance techniques. We combine both the split-and-merge and region growing approach and evaluates the proposed approach over the existing techniques. We show its reconnaissance improvement achieved whilst staying evasive. The hybrid approach has two stages; first the split-and-merge recursively splits the space to ensure the maximum exploration; secondly, upon reaching the maximum depth/smallest possible block size, region growing is used in each block to identify the precise boundary of the accept space in the block. However, both split-and-merge and region growing are used in the similar way and nothing is changed. The only difference is that they are used in two stages. Similarly, the maximum number of packets $m$ for probing are calculated from equation 1 and the size of the smallest patch $n_0$ will indicate the level on which the recursive split operation will stop. To test a patch for further split, random sampling is done and if recursive split is stopped, region growing is used to identify the precise boundary.

In the subsequent sections, we show the comparative evaluation and evasiveness of the proposed hybrid approach with the underlying existing techniques.

A. Comparative Evaluation

For the purpose of comparative evaluation, we conducted the experiments on real and synthetic policies both. Please note that the techniques under consideration fall into the block scan techniques i.e., they scan the ranges. Therefore, the synthetic policies employed had the rules which represents ranges instead of sparse individual hosts. The application of these techniques is to discover the network ranges of enterprise networks which usually have rules in the form of address range. However, we also evaluated the techniques on real enterprise policies having shorter ranges, or even individual host rules, which were obtained from Cisco and were anonymized whilst maintaining the ip and port structure of the network policy. Synthetic policies were generated as described in [8] and [28]. We used different policy structures to generate the synthetic policies. The size range used was 10−2000 rules. To maintain the realistic structure, we provided the probability distribution for each header field value. These distributions reflect common practices e.g., port 80 had higher probability since it is widely used.

For each policy, the reconnaissance process was applied using the aforementioned techniques. The reconnaissance can be done in three ways: 1) probe the firewall itself, 2) probe the end hosts behind firewall, 3) place an observatory node behind the firewall which confirms whether the probe has been filtered or not by the firewall. Since most of the firewalls block ICMP and are stateful, we opted to probe the end hosts using TCP SYN since the policies under consideration were consisted of TCP rules. For each reconnaissance process, 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 generated employed policy and the discovered one.

The comparative evaluation of the hybrid technique with both split-and-merge and region growing is investigated. Figure 10 show the accuracy comparison of all the three techniques for different policy size and space. Accuracy tells the percentage of accept space correctly identified from the total accept space. Patch sizes revealing the best accuracy were used in the comparison. However, the same set of policies are used for all the three techniques. Figure 10(a) shows the
three techniques accuracy for different policy sizes. It is clear that larger policies had better accuracy. Reason behind this is that larger policies have larger accept spaces. Therefore, once a probe is accepted, probability of subsequent accept probes increases and error bound limit is not reached quickly. It can be observed that for smaller policies the hybrid technique provided higher accuracy as compared to the other techniques. Although all the techniques provide higher accuracy, it can be noticed that hybrid technique consistently provides better accuracy overall.

Policy size does not necessarily reflect the percentage of the accept space i.e., accept space to total space ratio since the rule fields were randomly generated, we conducted accuracy comparison for different policy space (i.e., accept space to total space ratio). Figure 10(b) show the accuracy comparison on different policy space. It can be seen that for higher policy space, all the techniques discover the accept space with high accuracy. However, for smaller accept space hybrid technique outperforms the other techniques.

Figure 11 show the false positive comparison for the same policies and experiments used in Figure 10. False positive is the space which has been identified as accept space however it was actually a deny space. It can be clearly seen in both the Figures 11(a) and (b) that split-and-merge has the highest false positives. The reason is that split-and-merge can not split the patch further upon reaching the smallest allowed patch size. If the patch has both accept and deny space, and it was marked as accept space, then it will also raise a false positive. On the contrary, region growing is not bounded by the limitation factor for smallest possible patch size. However, it tries to find the boundary as precisely as possible by dividing the space. Therefore, false alarms observed were less in this case. Lastly, hybrid approach takes advantage of both and uses splitting and once the smallest patch is reached, it uses region growing with in the patch to identify the boundary, therefore resulting in the lower false positives.

B. Discovery of Real Policies

Comparative evaluation on real policies is shown in Figure 12. Policies having strict criteria (shorter accept ranges or even single point i.e., one IP and one port rules instead of range) were chosen for the evaluation. This makes it harder for the reconnaissance technique to discover the entire space with higher accuracy. Average port range was 17 and IP range was 138. Most of the rules, i.e., > 90%, were single point rules. If you compare the overall performance of reconnaissance techniques on synthetic policies having larger accept space as shown in Figure 10(a) and real policies having strict rules in Figure 12(a), it can be noticed that the performance degraded overall from 90% to 80% approximately due to the policy structure. However, accepted packets in real policy for split-and-merge was 82, region growing was 71 and hybrid were 73. These policies were anonymized; however, the structure of the policy space was kept same.

Accuracy evaluation on real policies is shown in Figure 12(a). Similar trends can be observed that hybrid approach is consistently providing higher accuracy. Moreover, for larger policy size the accuracy is higher than 80% but for smaller policy size it is under 80%. Similarly, the false positives are reported in Figure 12(b). Considerable performance improvement, in terms of lower false alarms, is achieved by hybrid technique over split-and-merge and region growing.

C. Evasiveness

To investigate the detection of reconnaissance traffic generated by hybrid approach, we mixed multiple samples of it in the benign traces of LBNL dataset. TRW and Maximum Entropy were used to classify anomalous events from the mixed traffic. In previous sections we have explained the working methodology of both the detectors. It can be clearly observed from Figure 13(a) and (b) that TRW and Maximum Entropy (respectively) were unable to detect the reconnaissance time
window of hybrid technique. Since the figure shows a single instance of reconnaissance time window, we mixed multiple instances to observe the overall detection and false alarm rate. Both the detectors incur high false alarms rate i.e., greater than 40% in order to achieve acceptable detection rate. Therefore, we argue that existing well known detectors are unable to detect the reconnaissance techniques under consideration.

We do not show the statistical analysis of hybrid approach since the approach uses split-and-merge in first stage and region growing in second stage. Reconnaissance traffic of first stage shows the trends of split-and-merge. However, second stage follows the trend of region growing. The proposed detector can not be applied since it either works for increasing distance or decreasing distance. However, the average distance for subsequent probes decreases during the first stage and increases during the second stage.

VIII. CONCLUSION

Existing reconnaissance techniques were discussed and it was shown that these techniques are evasive to existing well known intrusion detectors. Their analysis was shown to provide some helpful insights for providing a countermeasure. Based on the analysis and patterns observed, a simple prototype of countermeasure (detector) was shown along with its detection accuracy results. However, it is shown that how the combination of these existing approaches can yield a new approach which may achieve higher performance improvement in terms of reconnaissance whilst staying undetected. We also discussed why the proposed prototype of detector can not detect the hybrid approach. We emphasize that there is a need of further investigation on next generation reconnaissance techniques so that intrusion detectors can detect them by leveraging the patterns observed.

ACKNOWLEDGEMENT

This work is supported by the National Science Foundation under Grant Number CNS-1023868. Any opinions, findings, and conclusions or recommendations expressed in this work are those of the author(s) and do not necessarily reflect the views of the National Science Foundation.

REFERENCES

Muhammad Qasim Ali is working in Cyber Defense and Network Assurability (CyberDNA) Center in the Department of Software and Information Systems as Research Assistant since 2010. He is currently a final year PhD student. His area of expertise is intrusion detection. His PhD work focuses on intrusion defense mechanisms for critical infrastructures. Before joining the PhD program, he has worked in the industry for multiple years in network security domain. He has received his masters degree from National University of Sciences and Technology (NUST) Islamabad, Pakistan.

Ehab Al-Shaer is a Professor and the Director of the Cyber Defense and Network Assurability (CyberDNA) and CCAA Center in the College of Computing and Informatics at University of North Carolina Charlotte. His primary research areas are network security, security management, fault diagnosis, and network assurability. Prof. Al-Shaer edited/co-edited more than 10 books and book chapters, and published about 100 refereed journals and conferences papers in his area. Prof. Al-Shaer is the General Chair of ACM Computer and Communication 2009-2010 and NSF Workshop in Assurable and Usable Security Configuration, August 2008. Prof. Al-Shaer also served as a Workshop Chair and Program Co-chair for number of well-established conferences/workshops in his area including POLICY 2008, IM 2007, ANM-INFocom 2008, CCS-SafeConfig 09, MMNS 2001, and E2EMON 04-05. He also served as a member in the technical program and organization committees for many IEEE and ACM conferences. He was awarded many Best Paper Awards. Prof. Al-Shaer received his MSc and Ph.D. in Computer Science from the Northeastern University (Boston, MA) and Old Dominion University (Norfolk, VA) in 1998 and 1994 respectively.

Taghrid Samak joined the Advanced Computing for Science, Lawrence Berkeley National Lab as a Postdoctoral Researcher in 2010. She is working on data mining and anomaly detection for workflow scientific applications. Her PhD work was in developing novel algorithms for distributed network policy configuration. Her emphasis is on “data science” at the intersection of statistical algorithms and computing, in the context of important scientific problems. She has developed workflow failure prediction algorithms, fingerprinting HPC codes, and two projects with the LBNL Joint Genome Institute: predicting protein function and genome assembly errors. She received her PhD in Computer Science from DePaul University, Chicago.