Modeling modern network attacks and countermeasures using attack graphs

Modeling Modern Network Attacks and Countermeasures Using Attack Graphs

Kyle Ingols, Matthew Chu, Richard Lippmann, Seth Webster, Stephen Boyer

MIT Lincoln Laboratory
244 Wood Street
Lexington, Massachusetts 02420-9108
Email: kwi@ll.mit.edu

***Abstract*—By accurately measuring risk for enterprise networks, attack graphs allow network defenders to understand the most critical threats and select the most effective countermeasures. This paper describes substantial enhancements to the NetSPA attack graph system required to model additional present-day threats (zero-day exploits and client-side attacks) and countermeasures (intrusion prevention systems, proxy firewalls, personal firewalls, and host-based vulnerability scans). Point-to-point reachability algorithms and structures were extensively redesigned to support "reverse" reachability computations and personal firewalls. Host-based vulnerability scans are imported and analyzed. Analysis of an operational network with 85 hosts demonstrates that client-side attacks pose a serious threat. Experiments on larger simulated networks demonstrated that NetSPA's previous excellent scaling is maintained. Less than two minutes are required to completely analyze a four-enclave simulated network with more than 40,000 hosts protected by personal firewalls.**

I. Introduction

Enterprise networks are constantly under attack from a continuous stream of threats. Some are widespread, such as the "conficker" worm [1]; others are highly targeted, such as the recently analyzed "GhostNet" activities [2]. Currently, no accurate automated approach exists to predict the risk these and other threats pose to specific networks. This is one of the primary goals of our research. Accurate automated risk assessments would make it possible to compare the risk of multiple threats, select effective countermeasures, predict the threat posed by hypothesized attacks, and determine the effectiveness of planned and past security expenditures. This requires accurate models of adversaries, of networks, of vulnerabilities, and of a network's mission or purpose.

Over the past few years we have developed a tool named "NetSPA" that uses attack graphs to model adversaries and the effect of simple countermeasures [3]–[5]. It creates a network model using firewall rules and network vulnerability scans. It then uses the model to compute network reachability and attack graphs representing potential attack paths for adversaries exploiting known vulnerabilities in server software ("server-side" vulnerabilities). This discovers all hosts that can be compromised by an attacker starting from one or more initial locations. Asset values assigned to each host measure the utility of hosts to a network's purpose or mission. Degradation in mission capability is then represented by the percentage of the total asset value threatened by the attacker.

NetSPA scales roughly as $O(n \log n)$ as the number of hosts in a typical network increases. It has been used to analyze actual networks with more than 3,000 hosts and simulated networks with nearly 50,000 hosts. Risk is assessed for different adversaries and countermeasures by measuring the total assets that can be captured by an adversary and also by measuring the attacker effort as suggested in [6]. We currently compute simple measures of attacker effort, such as the number of hops or stepping-stone hosts, the effort required to exploit given vulnerabilities as defined by CVSS [7], and the number of unique exploits required to compromise all exploitable hosts in the network.

Past research on using automated attack graphs to model adversaries (e.g., [5], [8]) – including our own – has used threat and countermeasure models that are now out of date. The most common threat model used in the past is of an attacker compromising hosts via server-side vulnerabilities, and the primary countermeasure explored is patching these vulnerabilities. This paper describes how we extended the NetSPA tool to model additional modern threats and countermeasures.

In the remainder of this paper we first describe the most important new attacks and defenses and our modeling approach. We then present the extensive changes in reachability computation required by these models, followed by evaluations on a small real network and large simulated networks. The paper ends with a discussion of related work, limitations, and future work.

II. Modeling Client-Side Attacks

Some of the most important modern threats are vulnerabilities in web browsers, e-mail clients, document viewers, and multimedia applications running on victim machines,

This work is sponsored by the United States Air Force under Air Force Contract FA8721-05-C-0002. Opinions, interpretations, conclusions and recommendations are those of the authors and are not necessarily endorsed by the United States Government.

1063-9527/09 $26.00 © 2009 IEEE
DOI 10.1109/ACSAC.2009.21

or "client-side" vulnerabilities (e.g., [9], [10]). There are many potential vectors for exploiting such vulnerabilities. We model client-side attacks in which an attacker adds malicious content to a server that a victim then downloads unknowingly. Additional vectors such as malicious email and encrypted tunnels are considered as future work.

Client-side vulnerabilities must be found to be modeled. Prior attack graph systems (e.g., [8], [11]) identified only server-side vulnerabilities using network-based scanners such as Nessus [12]. These scanners are unsuitable for detecting most client-side vulnerabilities, even when login credentials are provided, because they do not examine many details of installed client software. Recent advances in the Security Content Automation Protocol (SCAP) program [13] make it possible to obtain details necessary to model client-side vulnerabilities in a consistent manner across different operating systems. In section VI-B we describe how we use an Open Vulnerability Assessment Language (OVAL)-based scanner, running on each host, to detect and collect client-side vulnerabilities.

Client-side attacks require a host-to-host reachability computation that can be considered *backwards* from that required for server-side attacks. For a server-side attack, we must determine if an attacker can reach the victim host's vulnerable server. For a client-side attack, we must instead determine if the malicious server can be reached by a vulnerable client. Section V-D describes NetSPA's reverse-reachability system.

III. Modeling Zero-Day Attacks

Another important modern threat is an attacker with knowledge of an unpublished, or "zero-day," vulnerability [10]. Although it is impossible to predict the existence of any specific zero-day vulnerability, it is possible to hypothesize a zero-day vulnerability in specific software applications to ensure that the impact of an eventual zero-day can be understood and minimized.

We evaluate zero-day risk by hypothesizing a zero-day vulnerability in each application on a network, one at a time, and building an attack graph for each new vulnerability to assess attacker effort. We then order applications by the benefit provided to an attacker, were a zero-day available. This allows defenders to focus on those application instances that provide the most attacker benefit. They can uninstall, move, limit ingress or egress, or monitor these installations more closely with an anomaly-based intrusion detection system (IDS).

Proper modeling of zero-day vulnerabilities requires an inventory of server and client software on each computer. Ideally such an inventory would use the new SCAP-based Common Process Enumeration (CPE) standard [13], [14], as it provides a consistent name for all applications. CPE entries typically provide an OVAL plugin designed to detect the software; thus a host-based OVAL scanner could gather application information.

Until CPE matures, we model only server-side zero-day vulnerabilities by assuming that each protocol and port represents a different application. For example, NetSPA posits the existence of a piece of software called "25/tcp," running on every device with TCP port 25 open. Although not ideal, our current approach is automatable and provides useful results on large networks without requiring additional host-based scanning.

IV. Modeling Modern Countermeasures

Many countermeasures can be deployed to defend enterprise networks [15]. In this section we describe how some of the most common countermeasures have been included in NetSPA. These include personal firewalls that filter traffic at the host level, intrusion prevention systems (IPSs) that block disallowed types of web content such as ActiveX controls, and proxy firewalls that provide a common protected outgoing client connection to the Internet for many hosts in a network.

Personal firewalls, also called endpoint or host-based firewalls, are installed on individual hosts to control incoming and outgoing traffic. In an enterprise, these firewalls are often centrally managed and use identical rulesets. NetSPA's previous reachability system [4] was designed only for inline firewalls typically used at network borders. The prior system's performance rapidly deteriorated with many personal firewalls because each personal firewall was modeled separately as an inline firewall. Section V-F describes our new system, which restores performance by grouping personal firewalls with identical rulesets together. This required developing new software to import rules used by personal firewalls, redesigning our reachability model, and rewriting substantial parts of NetSPA's reachability computation system.

Intrusion prevention systems (IPSs) are used to protect vulnerable hosts by blocking incoming malicious content. Two aspects of IPSs need to be modeled: their effect on vulnerabilities, and their impact on reachability. Because small changes in malicious code can often elude IPS signatures, many IPSs block entire classes of potential attack vectors – ActiveX controls or Java applets, for example. Modeling this behavior requires a mapping between the blocked attack vector and the vulnerabilities that depend on that vector for exploitation. This mapping is currently created by automated keyword searches on NVD vulnerability descriptions followed by manual confirmation. Creating a mapping in this manner is far from ideal; future SCAP standards such as CPE may provide more automatable means of making this connection.

IPS systems can be deployed as inline proxies, much like inline firewalls. They can also be deployed as "non-transparent" proxies, to which client machines must explicitly connect. The proxy then connects to the desired server. Using a mapping of attack vector to vulnerabilities, NetSPA

can model the effect of both types of IPS systems via simple adjustments to its reachability model. These changes are detailed in Section V-E.

V. Reachability

Attack graph generation tools require, at a minimum, answers to two core questions: where are the vulnerabilities, and what are the restrictions on attackers' ability to exploit them? Vulnerability scanners answer the first question. The answer to the second must be inferred from the network's underlying topology and the firewalls and routers that dictate traffic flow. We are thus required to model the network's topology and filtering devices in an efficient manner.

NetSPA's network model supposes that an individual *host* possesses one or more *interfaces*, each of which may have an IP address. These interfaces have zero or more open *ports*, accepting connections from other hosts. A port has a *port number* and *protocol*. Each host and port may have zero or more *vulnerability instances*, particular flaws or configuration choices which may be exploitable by an attacker.

A straightforward means of representing a network's reachability is an $I - K$ matrix, where I represents the number of interfaces and K represents the number of server ports. A given cell in the matrix indicates whether traffic from the source interface to the target port is permitted. Although straightforward, it is often large and redundant.

The remainder of this section explores the methods used to make reachability computation tractable in large networks. Our previous work [3], [4] handled filtering and NAT rules as discussed in Sections V-A and V-B, though the ordering of rules was far less flexible than the system now described. Section V-C discusses new rule-based branching, used to handle devices with multiple potential egress paths for traffic. Section V-D and V-E cover additions made to accommodate client-side attacks, non-transparent proxies, and IPS systems. Finally, Section V-F covers NetSPA's aggressive grouping strategies, designed to identify and leverage redundancy in the $I - K$ matrix.

A. Firewall Model

NetSPA models reachability using tuples of the form `[source IP -> target IP:portnum/protocol]`. Much like the FIREMAN [16] system, these sets are represented as binary decision diagrams (BDDs). A BDD is an efficient way to represent a Boolean equation like $x \wedge (z \vee y)$, such that evaluating the equation on a set of variable assignments can be evaluated in time proportional to the number of Boolean variables, rather than to the length of the Boolean expression. The variables are the bits of the reachability tuple – for example, the source IP in an IPv4 network can be represented as 32 bits, or 32 Boolean variables. Because the number of variables is fixed, we can traverse the BDD in constant time. Like FIREMAN, NetSPA implements BDDs via the BuDDy library [17].

NetSPA models firewalls via rules, rulegroups, and chains. A *rule* matches a subset of reachability and acts upon it; these actions include allowing and denying the traffic. The specific dispositions are discussed later. Every rule belongs to a *rulegroup*. A rulegroup consumes a set of reachability as input and produces three output sets $< A, D, R >$. The A, D, R notation, adapted from FIREMAN, refers to the set of allowed traffic (A), denied traffic (D), and traffic that was not acted on by an allow or deny rule (R). A *chain* points to a rulegroup and dictates the next step for traffic in each of the rulegroup's three output sets.

Each interface in the network is assigned an inbound and an outbound chain to adjudicate traffic passing through it. Inbound chains are used to explicitly designate how traffic moves *through* a host. Traffic that is given the default disposition, but refers to the inbound interface's listening address, is assumed to go *to* the host, i.e., to a port on the firewall itself. Outbound chains apply only to traffic *from* the host itself, not to traffic passing *through* the host but originating elsewhere.

To traverse a rulegroup, we take the input *source set* $\mathcal{S}$ of reachability and traverse the rules, in order, one at a time. For rule number i, we consider $< A, D, R_i >$, where A is the traffic that has already been accepted, D is the traffic that has already been denied, and R_i is the traffic that was not accepted or denied by a rule prior to rule i. For a set of n rules, we begin with $< A, D, R_1 >=< \emptyset, \emptyset, \mathcal{S} >$; the final result is $< A, D, R_{n+1} >$. We write R_{n+1} as simply R.

Filtering rules are the easiest to model, as they simply accept or deny traffic. For example, the rule `[ * -> 10.0.0.1:25/tcp ]: ALLOW` permits any available traffic targeting port number 25, protocol tcp, address 10.0.0.1 to pass (i.e., go to the A output). To evaluate an allow rule i, we use the FIREMAN method of assigning it a *match set* P_i, a BDD representing what the rule itself matches. For this example, P_i is `[ * -> 10.0.0.1:25/tcp ]`. Of the reachability not yet acted upon (R_i), we must take what the rule matches and move it to A:

$$\begin{aligned} R_{i+1} &= R_i - P_i \\ A &= A \cup (R_i \cap P_i) \end{aligned}$$

A deny rule is handled similarly.

A rulegroup with only filtering rules need only be evaluated once; the result can be reused for multiple potential source sets. It is sufficient to start with $< \emptyset, \emptyset, * >$, compute $< A, D, R >$, and then, for any source set $\mathcal{S}$, compute $< A \cap \mathcal{S}, D \cap \mathcal{S}, R \cap \mathcal{S} >$ as the output.

B. Network Address Translation

The FIREMAN system is very powerful and handles filtering rules very well, but it does not consider network address translation (NAT) rules. NAT rules modify the contents of R_i, usually by manipulating either the source address (source NAT, or SNAT) or the target address or port (destination NAT, or DNAT). NetSPA's reachability system models both.

NetSPA's handling of NAT rules via BDDs uses BuDDy's $exist$ function. $exist(X, Y)$ takes all variables used in Y and removes them from X, essentially resetting those variables to a "do not care" state. For example, $exist(y \wedge x \wedge (z \vee y), y) = x \wedge z$. This can be used to remove restrictions on sections of the reachability tuple.

For example, consider the NAT rule `[ 192.168.0.0/24 -> 10.0.0.1:25/tcp ] : SNAT to 10.5.17.36`. This rule changes the source IP address of matched reachability to 10.5.17.36, while the destination address, protocol, and port remain unchanged.

We model a NAT rule using three BDDs. Recall that a filtering rule has only P_i, the traffic matched by the rule. For a NAT rule, we build P_i as well. We also require M_i, the *mask set* representing the information that the rule *changes* – every bit changed is set to one, and all other bits are left unset. Lastly, we require T_i, the *transform set*, representing the translated value imposed by the rule. For the example rule, P_i is `[ 192.168.0.0/24 -> 10.0.0.1:25/tcp ]`, M_i is `[ 255.255.255.255 -> * ]`, and T_i is `[ 10.5.17.36 -> * ]`.

To process a NAT rule, we first isolate traffic that the rule matches. This, like a filtering rule, is computed as $(R_i \cap P_i)$. Second, we compute the modified reachability by using the $exist$ function to remove the values to be changed (M_i). Finally, we use T_i to insert the changed values, taking care to not manipulate traffic that the rule does not match $(R_i - P_i)$.

$$R_{i+1} = (R_i - P_i) \cup (exist(R_i \cap P_i, M_i) \cap T_i)$$

NAT rules, by themselves, do not permit or deny traffic; A and D are unaltered.

Consider as an example the rulegroup shown in the left-hand side of Figure 1. For this example we drastically simplify reachability by ignoring destination port and protocol, and restricting the source and destination addresses to the sets $(1, 2, 3, 4)$ and (a, b, c, d), respectively – doing so allows us to visually depict the resulting $I \rightarrow K$ reachability matrix, shown in the upper left of the figure. In the matrix, a gray cell represents true, and white represents false.

For illustrative purposes, we begin with $\mathcal{S} = (1, 2, 3) \rightarrow *$. The first rule accepts only traffic from 1 to a (shown as $1 \rightarrow a$), thus adding $P_i \cap R_i = 1 \rightarrow a$ to the accept disposition A and passing the remainder $R_i - P_i$ to the next rule.

The second rule is a NAT rule, translating reachability from source address 1 to the source address 4. As a result,

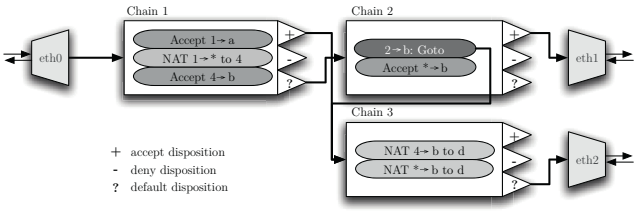

Figure 2. Example Arrangement of Three Chains with Three Interfaces

$1 \rightarrow (b, c, d)$ is removed from R_{i+1}, as it's been translated to something else, and $4 \rightarrow (b, c, d)$ is added; this value proceeds to the next rule.

The final rule is another filtering rule, accepting traffic from 4 to b. It adds $R_i \cap P_i = 4 \rightarrow b$ to the accept disposition A and removes P_i from R_{i+1}. The final result is $A = (1 \rightarrow a, 4 \rightarrow b), D = \emptyset, R = ((2, 3) \rightarrow *, 4 \rightarrow (c, d))$.

C. Branching between chains

Transitioning between chains is accomplished via chain dispositions, which determine the destination of traffic from each of the three answers $< A, D, R >$. For example, consider the three chains shown in Figure 2. Here, the accept set of chain 1 becomes the input to chain 3, and the accept set of chain 2 exits the firewall via interface `eth1`.

In addition to chain-to-chain branching, it is sometimes desirable to branch to another chain from within a rule. NetSPA supports two rule dispositions for this purpose: *goto* and *plusgoto*. In both cases, the traffic matched by the rules $(R_i \cap P_i)$ is sent to another chain as its input $\mathcal{S}$. A goto rule prevents the matched traffic from continuing in the original rulegroup $(R_{i+1} = R_i - P_i)$; a plusgoto rule does not $(R_{i+1} = R_i)$. NetSPA uses plusgoto rules extensively to model routing where multiple valid routes exist; via plusgoto rules, NetSPA can explore all of the possible routes in order to perform a worst-case evaluation.

The combination of chains and the edges between them creates a directed acyclic graph, e.g., the graph shown in Figure 2. We refer to a chain that directly receives traffic from a given interface as that interface's *entry node*, and any chain that directly sends traffic to an interface as one of the interface's *exit nodes*.

D. Reverse Reachability

To support client-side attacks, NetSPA's reachability system has been extended to efficiently compute reachability *backwards*, starting at the malicious server and working backwards to the vulnerable clients. This could be accomplished by computing forward reachability from every host on the network. However, on networks with solid restrictions on outbound traffic, it is more efficient to traverse reachability chains, and their rules, backwards.

To compute reverse reachability through a firewall, we must posit a set of reachability $\mathcal{T}$ that we wish to run

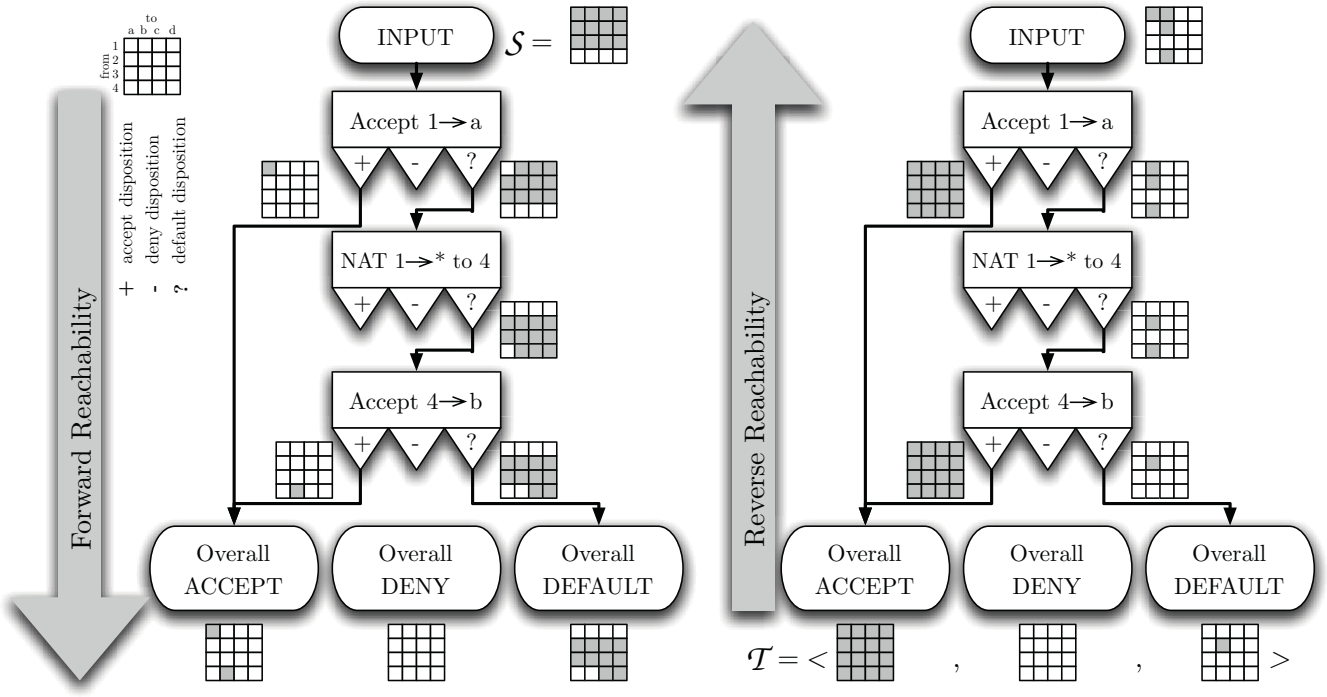

Figure 1. Simplified Example Reachability Flow Through a Rulegroup: Forward and Reverse

backwards from a given subnet, through all interfaces connected to that subnet. For a given interface, we identify the interface's exit nodes and walk the chain graph backwards from them to all other interfaces' entry nodes. Within a rulegroup, we walk the rules in reverse as well.

Each rule disposition is handled differently. An allow rule must remove from R_{i+1} everything that the rule matches, because such traffic would never have been sent to R_{i+1} by the rule. It must also add to R_i everything in A that the rule matches, as it would have flowed through the rule to A:

$$R_i = (R_{i+1} - P_i) \cup (A \cap P_i)$$

Deny rules are similar.

NAT rules are more complicated. M_i is unchanged, but we must use a new P_i', the reachability *after* the NAT rule was applied, and T_i', the altered reachability *before* the NAT rule makes its changes. For example, consider the NAT rule we evaluated in Section V-B: `192.168.0.0/24 -> 10.0.0.1:25/tcp : SNAT to 10.5.17.36`. M_i remains the same, P_i' becomes `10.5.17.36 -> 10.0.0.1:25/tcp`, and T_i' becomes `192.168.0.0/24 -> *`. We do not remove the NAT rule's output from R_i, as it could have occurred on its own; a NAT rule that translates `X` to `Y` does not prevent `Y` from occurring in the absence of `X`. We process the NAT rule as follows:

$$R_i = R_{i+1} \cup (exist(R_{i+1} \cap P_i', M_i) \cap T_i')$$

Goto rules remove what they match, because that reachability would have been sent to some other chain: $R_i = R_{i+1} - P_i$. Plusgoto rules have no effect: $R_i = R_{i+1}$.

However, goto and plusgoto rules are treated differently when we branch back to them from another chain: in this case, evaluation of the rulegroup begins at the goto or plusgoto rule, not at the end of the rulegroup. We therefore begin at the goto/plusgoto rule i with everything that the rule could have sent to the chain that we're now coming from: $< A, D, R_i >=< \emptyset, \emptyset, P_i \cap \mathcal{S} >$.

For example, consider again the chain graph of Figure 2 and hypothesize traffic traveling backwards through `eth2`. We begin on Chain 3 with $\mathcal{T} =< \emptyset, \emptyset, * >$. Processing backwards, we would arrive at the start of Chain 3 with $\mathcal{S} = *$. This propagates to the accept disposition of Chain 1, and also to the goto rule in the middle of chain 2's rulegroup. Upon arrival at the goto rule, we compute $P_i \cap *$, yielding $R_i = 2 \rightarrow b$; the only traffic that could have been sent to Chain 3 by this goto rule is $P_i = 2 \rightarrow b$, so that is all that could be propagated backwards. From chain 2, the result $2 \rightarrow b$ becomes the input to the default disposition of Chain 1.

We can now traverse Chain 1's rulegroup backwards, as shown in the right-hand side of Figure 1. The input is $\mathcal{T} =< *, \emptyset, 2 \rightarrow b >$.

The last rule in the rulegroup, "Accept $4 \rightarrow b$," computes $R_i = (R_{i+1} - P_i) \cup (A \cap P_i)$; as a result, $R_i = ((2,4) \rightarrow b)$.

The NAT rule manipulates the contents of R_i in reverse,

translating $4 \rightarrow *$ back to $1 \rightarrow *$ but not removing $4 \rightarrow *$. The resulting $R_i = ((1,2,4) \rightarrow b)$.

Finally, the first filtering rule acts on $< *, \emptyset, (1,2,4) \rightarrow b >$. From this, $P_i \cap R_i = (1 \rightarrow a)$, so it's added to the final answer, $R_1 = ((1,2,4) \rightarrow b, 1 \rightarrow a)$. This is the sum total of reachability that could enter `eth0` and emerge from `eth2` as part of our originally hypothesized set $\mathcal{T} =< *, \emptyset, \emptyset >$.

The discussion above focuses on traffic *through* firewalls. Addressing traffic *from* a victim host that has its own rules, or *to* a malicious server that has not disabled its own firewall, uses similar principles. If the traffic is *from* a victim host with address `X`, we start at the interface's outbound chain with $\mathcal{T} =< \mathtt{X}, \emptyset, \emptyset >$ and travel backwards as before.

Traffic *to* a firewalled malicious server, however, is more complicated, because there are no explicit exit nodes – any chain reachable from the inbound chain could provide reachability to the host itself. The NetSPA system therefore computes *forward* reachability via $\mathcal{S} =< \emptyset, \emptyset, * >$, keeping track of all chains reached, and then performs reverse reachability from all of those chains as potential exit nodes.

E. Non-Transparent Proxies and IPS Systems

Transparent proxies look just like inline firewalls. To use a non-transparent proxy, however, a client machine must be configured to explicitly connect to the proxy host, which then makes the connection on the client's behalf. From a reachability perspective, the client only connects to the proxy, but the proxy can then connect to anywhere else on behalf of the client.

We model this situation in NetSPA via destination NAT. We add a NAT rule of the form `[ * -> X : DNAT to * ]`, where `X` is the proxy server's own IP and port number. Other target values, such as protocol and port number, could be wildcarded as desired. IPS systems that then restrict the servers that can be contacted could be modeled by adding deny rules after the DNAT rule.

NetSPA uses a similar trick to model the effect of an IPS. Assuming a mapping from attack vector to vulnerabilities exists, NetSPA models the IPS by treating it like a firewall. NetSPA's filtering model has been extended to permit blocking traffic based on vulnerability, adding vulnerability to the normal tuple of source IP, destination IP, destination port, and protocol.

F. Grouping Hosts and Firewalls

An $I \quad K$ matrix for representing reachability is typically highly redundant; on typical networks, groups of interfaces are treated identically by the firewalls and other filtering devices. Interfaces within such a group will have identical rows in the matrix, as they will all be able to reach the same ports. NetSPA identifies these *forward reachability groups* and computes only one row for each, saving both time and space.

To do this, NetSPA collects the set $\mathcal{N}$ of all IP address singletons and ranges used in all firewall rules. Two interfaces on the same subnet with identical rules can be grouped together if their listening addresses are in the same subset of $\mathcal{N}$; reachability for one is identical to reachability for the other. The grouping operation on a network with I interfaces and L rules is $O(IL + I \log I)$, but the savings obtained by running the algorithm is substantial. In the ideal case of a network with one interconnected internal subnet and a firewall that does not differentiate between the network's hosts, the factor of I in the $I \quad K$ matrix size can effectively drop out and be replaced by a constant. In the pathological case of a network where each interface is treated differently, then no savings are achieved. In practice, we've found that real networks typically receive a substantial benefit from this approach. In the first case explored in Section VI-B2, we dropped from 65,025 cells to 3,825 – a factor of 17 savings in time and memory.

NetSPA can use a similar approach to create *reverse reachability groups*, combining redundant *columns* in the $I \quad K$ matrix.

In addition, NetSPA groups hosts with identical personal firewall rulesets by forming *target reachability groups*. Two single-homed hosts are in such a group if they are on the same subnet and their inbound chains are isomorphic; that is, they use the same rulegroup and all of the chains' dispositions are isomorphic. NetSPA's reachability engine can then traverse a target reachability group's common rules once to attempt reachability to every host in the group. However, the extra context means that the system also knows to traverse their chains, outbound and inbound, when traffic moves between members of the group.

VI. Performance

To evaluate the scalability of the new system, we conducted a number of measured tests on various synthetic networks. All tests were executed on a 2.4GHz Pentium 4 computer running Linux with 1GB of RAM. Peak memory measurements were done via Linux's `libmemusage.so` library. We measured runtime via wallclock. For timing tests, the system was run five times, discarding the first result and averaging the remaining four. In all cases the system posited an external attacker, computed reachability as needed, constructed the attack graph, and computed recommended remediation steps.

A. Full- and Partially-Synthetic Networks

We begin by comparing the new system's ability to aggregate personal firewalls to the previous system's costly workaround. We use a set of test data from a small 251-host network. We placed identical synthetic personal firewalls on 52 of the hosts and varied the number of rules in the common ruleset from 0 to 10,000; these results are shown in the top of Figure 3. We then varied the number of hosts

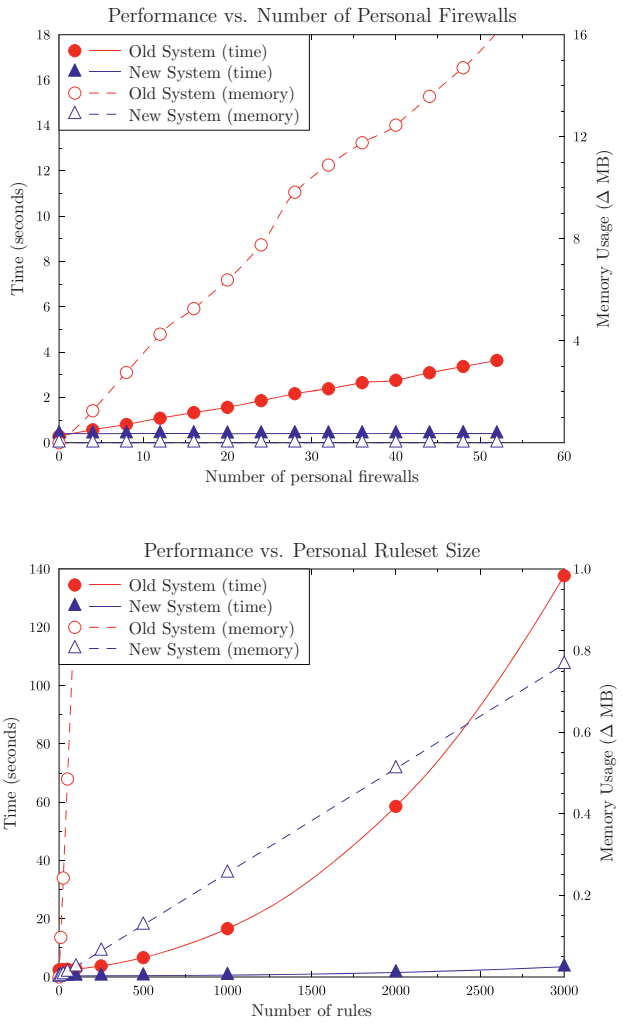

Figure 3. NetSPA Performance with Personal Firewalls; Old System of [4] vs. New System

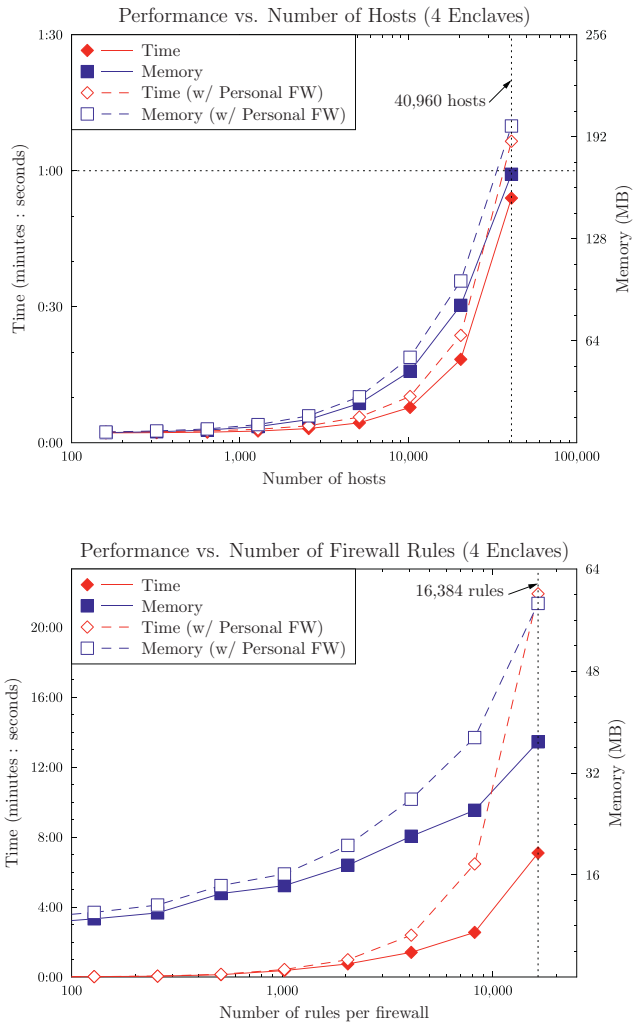

Figure 4. NetSPA Performance with Enclave, Rule, and Host Scaling; New System

with personal firewalls from 0 to 52, fixing the common ruleset at 250 rules. These results are shown on the bottom of Figure 3. In both cases we show the total time consumed, as well as the change in memory consumption from the zero point of the X axis. We show only the change in memory so scaling effects can clearly be seen between the two systems, as the differences in baseline memory requirements are comparatively unimportant.

The benefit of the new system is clear: adding additional personal firewalls with a shared ruleset causes almost no impact. Adding rules to the ruleset causes a roughly linear increase, though this is difficult to see because the system remains very fast throughout the test. In practice, we would not expect personal firewall rulesets to exceed 2,000 rules.

We conducted additional scalability experiments on the synthetic network shown in Figure 5. As shown, the network

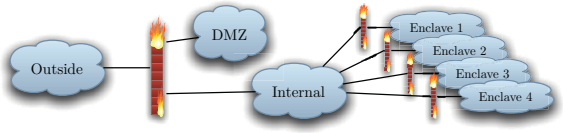

Figure 5. Synthetic Network Structure Used for Scaling Experiments

includes a border firewall, a DMZ, an internal network, and four additional enclaves, each with its own border firewall. An attack path from the outside, to the DMZ, to the inside, and finally to each enclave was established. Ten ports were assigned to each host, half of which were assigned one vulnerability each.

In the first case, shown on the top of Figure 4, we kept the number of rules constant at 250 per firewall and varied only the total number of hosts in the network. We show results

with and without personal firewalls, where the personal firewalls used common per-enclave rulesets of 250 rules. The graph's scale is log-linear and shows the tool's ability to handle large numbers of hosts in seconds.

In the second case, shown on the bottom of Figure 4, we leave the number of hosts constant at 1,250 and vary the number of rules on each firewall. We again show results with and without personal firewalls; here, the number of rules on the personal firewalls equals the number on the infrastructure firewalls. The graph's scale is log-linear and shows that running time remains under two minutes in the typical 100 to 2,000 rules per firewall expected in practice.

B. Evaluation on a Real Network

We also evaluated the system using a real network of 85 hosts behind a Juniper Netscreen firewall. We began by collecting typical data, consisting of a Nessus scan and the Juniper firewall ruleset. We then collected additional host-based data, discussed in Section VI-B1, and then evaluated several scenarios on the network in Section VI-B2.

1) OVAL Scanning: We deployed the OVAL scanner on the target network's Windows-based hosts and collected vulnerability scan data every time a user logged into the Windows domain.

Over the course of a month, we obtained 155 scans of 43 hosts, logging a total of 402 unique vulnerabilities over that time. The NVD database categorized 233 of them as client-side vulnerabilities (the NVD entries have the `user_init` flag set). The remainder were classified as locally exploitable (26) or remotely (server-side) exploitable (143); of those 143, 25 are said to yield root access when exploited.

We randomly chose ten of the 25 remote-to-root vulnerabilities identified as remotely exploitable and evaluated them by hand, examining the NVD entries and additional references. The ten CVE IDs and their evaluations appear in Table I. Based on our evaluation, eight of the ten were incorrectly classified as server-side vulnerabilities.

This result makes us hesitant to trust NVD's otherwise solid data when evaluating a vulnerability that is suspected to be client-side. Until this is addressed, we suggest mitigating the risk of false negatives in the attack surface by considering all vulnerabilities discovered by OVAL to be client-side vulnerabilities and additionally using a network-based scanner to discover server-side vulnerabilities.

2) Experiments: Data in hand, we began evaluating the results with NetSPA. In all cases we hypothesized an adversary on the outside of the firewall.

The results of this and all other experiments on the real network are shown in Figure 6.

Figure 6a clearly shows that the situation is dire; all 169 of the server-side vulnerabilities and 51 of the 84 hosts can be exploited directly, through the perimeter firewall. We investigated the reachability traces provided by NetSPA and identified three rules which were permitting the majority of the inbound traffic. All three were intentional, allowing traffic from an asset management server and a handful of machines that belonged behind the enclave but weren't physically located there. As a hypothetical hardening measure, we removed these rules. The adversary could now only directly compromise 69 vulnerabilities as shown in Figure 6b. Once past the firewall, of course, the other 100 can still be exploited from an initially compromised host, as shown by the light gray squares. All 51 vulnerable hosts are exploited, although only 22 are exploited in the first step.

We then discovered an omission in the Juniper ruleset: a server called "MS-WINS" was used in an allow rule, but the port number was not defined. NetSPA models the worst case, so it assumed the rule allowed traffic to any destination port number. When redefined with the appropriate WINS port number of 1512, as shown in Figure 6c, the network became impervious to attack – as shown, the vulnerabilities are known but cannot be exploited, and no hosts are compromised.

At this point we introduced the OVAL scans from Section VI-B1, adding knowledge of 514 client-side vulnerabilities to the network. As shown in Figure 6d, the adversary could immediately take advantage, compromising even more hosts than before – 65 instead of 51 – as we are now aware of client-side vulnerabilities on hosts that had no server-side vulnerabilities. First, the client-side vulnerabilities are compromised, and then the server-side vulnerabilities can be attacked from any of those initially compromised hosts.

We next added available data on personal/endpoint firewalls on the network, but they did not impede the adversary, as they did not restrict traffic from the adversary's starting location or from hosts within the enclave itself. As a final step, we hypothesized an inline IPS capable of blocking exploits against Microsoft Office. We assumed, for example, that an adversary relying on a vulnerable client downloading a corrupt Word document would be thwarted. Note that we are neglecting simple attacker workarounds, such as convincing the client to use an encrypted connection, but for the sake of example we assume the adversary is using unobfuscated transmission methods. Figure 6e shows the impact such an IPS could have; 267 of the client-side vulnerabilities are no longer immediately compromisable from the outside, though the same number of hosts (65) are still eventually compromised. What-if experiments such as these could help defenders decide if potential changes are worthwhile investments.

VII. Related Work

To the best of our knowledge, no other commercial or research tools that use attack graphs to assess network risk include explicit models of personal firewalls, intrusion prevention systems, or modern client-side attacks. A comprehensive review of past attack graph research is presented in [8]. More recent approaches include [11], [18], and [19]. The

CVE ID	Description	Client-side?
CVE-2004-0963	Buffer overflow in Microsoft Word 2002 ... in a `.doc` file	**yes**
CVE-2004-1153	vulnerability in Adobe Acrobat Reader 6.0.0 ... `.ETD` document	**yes**
CVE-2006-2372	overflow in the DHCP Client service for Microsoft Windows 2000	**yes**
CVE-2006-4691	Workstation service (`wkssvc.dll`) ... via `NetrJoinDomain2` RPC	no
CVE-2007-0065	(OLE) Automation in Microsoft Windows ... remote attackers ... via a crafted script request (from Microsoft: "could allow remote code execution if a user viewed a specially crafted Web page."	**yes**
CVE-2007-1204	Universal Plug and Play (UPnP) service ... crafted HTTP headers	no
CVE-2008-0082	An ActiveX control ... is marked as safe for scripting	**yes**
CVE-2008-0102	Microsoft Office Publisher ... via crafted `.pub` file	**yes**
CVE-2008-3009	Windows Media Player 6.4 ... replies to authentication requests	**yes**
CVE-2008-3010	Windows Media Player 6.4 ... credential-reflection attacks, by sending an authentication request	**yes**

Table I
EVALUATION OF TEN ALLEGEDLY REMOTE-TO-ROOT VULNERABILITIES FOUND BY OVAL

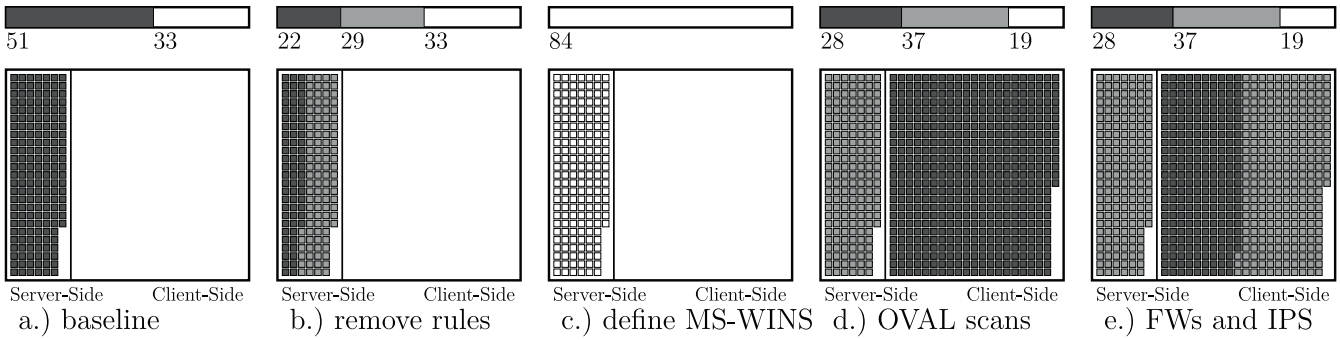

Figure 6. Order of Vulnerability Instance Compromise in Six Scenarios. The upper bar represents compromised hosts: white is uncompromised; light gray is compromised via a "stepping-stone" host; dark gray is compromised immediately. The lower graphic represents vulnerability instances: white squares indicate known but uncompromisable vulnerabilities; light gray squares indicate vulnerabilities that can only be compromised via a "stepping-stone" host; dark gray squares indicate an immediately exploitable vulnerability.

Topological Vulnerability Analysis (TVA) system [20] used in [11] does not explicitly model any type of firewall or analyze firewall rules. Instead, a subset of the network's host-to-host reachability is estimated using exhaustive host-to-host vulnerability scans. This introduces quadratic complexity, scales poorly to large networks, and complicates firewall modeling. This system also relies on a network vulnerability scanner and does not gather information about client-side attacks. The MulVAL system [21] used in [18] also does not explicitly model firewalls but assumes reachability is provided. Research in [19] uses our prior NetSPA system and thus scales well, but like our prior research, modern attacks and countermeasures are not modeled. Commercial attack graph products such as RedSeal [22] and Skybox [23] model firewalls and compute reachability from firewall rules but neither product's website mentions the ability to model personal firewalls, IPSs, or client side attacks.

There are many papers on firewall ruleset analysis and a few companies that analyze rulesets. Wool et al. [24] provides typical sizes for rulesets and confirms that firewall configuration errors are very common. Systems like FIREMAN [16], FANG [25] and a few companies (e.g., AlgoSec [26], RedSeal [22], Skybox [23]) detect different types of firewall misconfigurations such as overlapping or contradictory rules. Some also compute reachability and determine if firewalls enforce an overall policy. We use BDDs from [16] for efficient rule application which are similar in purpose to Michigan State's Firewall Decision Diagrams (FDDs) [27]–[29].

VIII. LIMITATIONS AND FUTURE WORK

The current NetSPA tool includes many countermeasures and attacks, but these need to be supplemented and improved. Some additional countermeasures we plan to add include the Federal Desktop Core Configuration (FDCC) for Windows, application white listing on hosts, and filtering provided by IPSs for exploits related to specific vulnerabilities. We also plan to model additional threats such as conficker [1] that use restricted exploit sets and propagation vectors. We additionally plan to model adversaries exploiting trust relationships between hosts, for example by using open Windows shares or shares protected by weak or common passwords. We also plan to explore more complex client-side attacks and model attacks specific to web sites and database

servers such as SQL injection and cross-site scripting. Virtual machines offer another exploit vector that should be considered, as access to the host OS allows compromise of all guest OS images. Improved modeling of zero-day attacks will involve capturing the application inventory on clients using OVAL and adding the ability to insert hypothetical zero-day vulnerabilities into the applications discovered.

Field tests of the tool are always highly instructive; they generally involve importing rules from additional types of firewalls and performing further tests on actual networks. The tool's usability will also be improved via further refinements to the attack graph GUI [5].

REFERENCES

[1] P. Porras, H. Saidi, and V. Vegneswaran, "An analysis of conficker's logic and rondezvous points," SRI International, Tech. Rep., February 2009.

[2] S. Nagaraja and R. Anderson, "The snooping dragon: social-malware surveillance of the tibetan movement," University of Cambridge, Computer Laboratory, Tech. Rep. UCAM-CL-TR-746, Mar. 2009.

[3] R. Lippmann *et al.*, "Validating and restoring defense in depth using attack graphs," in *IEEE Military Communications Conference (MILCOM)*, 2006.

[4] K. Ingols, R. Lippmann, and K. Piwowarski, "Practical attack graph generation for network defense," in *ACSAC*. IEEE Computer Society, 2006, pp. 121–130.

[5] L. Williams, R. Lippmann, and K. Ingols, "GARNET: A graphical attack graph and reachability network evaluation tool," in *Visualization for Computer Security (VizSEC)*, ser. Lecture Notes in Computer Science, J. R. Goodall, G. J. Conti, and K.-L. Ma, Eds., vol. 5210. Springer, 2008, pp. 44–59.

[6] D. L. Buckshaw *et al.*, "Mission oriented risk and design analysis of critical information systems," *Military Operations Research*, vol. 10, pp. 19–38, 2005.

[7] P. Mell, K. Scarfone, and S. Raomanosky, "A complete guide to the common vulnerability scoring system version 2.0," in *Forum of Incident Response and Security Teams (FIRST)*, 2007.

[8] R. P. Lippmann and K. Ingols, "An annotated review of past papers on attack graphs," MIT Lincoln Laboratory, Project Report IA-1, 2005.

[9] "IBM Internet Security Systems X-Force 2008 trend and risk report," IBM Global Technology Services, Tech. Rep., 2009.

[10] D. Turner *et al.*, "Symantec internet security threat report, trends for July - December 07," Symantec, Tech. Rep., 2008.

[11] S. Noel and S. Jajodia, "Optimal IDS sensor placement and alert prioritization using attack graphs," *Journal of Network and Systems Management*, vol. 16, no. 3, pp. 259–275, 2008.

[12] Tenable. Nessus security scanner. [Online]. Available: http://www.nessus.org

[13] R. Martin, "Making security measurable and manageable," in *IEEE Military Communications Conference (MILCOM)*, 2008.

[14] Common platform enumeration. MITRE. [Online]. Available: http://cpe.mitre.org

[15] R. Ross *et al.*, "Recommended security controls for federal information systems and organizations," National Institute of Standards and Technology, NIST Special Publication 800-53, Revision 3, February 2009.

[16] L. Yuan *et al.*, "FIREMAN: A toolkit for FIREwall modeling and ANalysis," in *IEEE Symposium on Security and Privacy*. IEEE Computer Society, 2006, pp. 199–213.

[17] J. Lind-Nielsen *et al.* BuDDy, a binary decision diagram library. [Online]. Available: http://buddy.sourceforge.net/

[18] R. E. Sawilla and X. Ou, "Identifying critical attack assets in dependency attack graphs," in *13th European Symposium on Research in Computer Security (ESORICS)*, S. Jajodia and J. López, Eds., vol. 5283. Springer, 2008, pp. 18–34.

[19] N. Pham, L. Baud, P. Bellot, and M. Riguidel, "A near real-time system for security assurance assessment," in *3rd International Conference on Internet Monitoring and Protection (ICIMP)*, 2008, pp. 152–160.

[20] S. Jajodia, S. Noel, and B. O'Berry, *Topological Analysis of Network Attack Vulnerability*. Kluwer Academic Publisher, 2003, ch. 5.

[21] X. Ou, S. Govindavajhala, and A. Appel, "MulVAL: A logic-based network security analyzer," in *Proceedings of the 14th USENIX Security Symposium*, 2005, pp. 113–128.

[22] RedSeal. (2009, April) Redseal systems. [Online]. Available: http://www.redseal.net

[23] Skybox. (2009, April) Skybox security, inc. [Online]. Available: http://www.skyboxsecurity.com/

[24] A. Wool, "A quantitative study of firewall configuration errors," *Computer*, vol. 37, no. 6, pp. 62–67, June 2004.

[25] A. Mayer, A. Wool, and E. Ziskind, "Fang: A firewall analysis engine," in *IEEE Symposium on Security and Privacy*, 2000, pp. 177–187.

[26] Algosec. [Online]. Available: http://www.algosec.com

[27] A. Liu, E. Torng, and C. Meiners, "Firewall compressor: An algorithm for minimizing firewall policies," in *27th Conference on Computer Communications (INFOCOM)*. IEEE, 2008, pp. 176–180.

[28] C. Meiners, A. Liu, and E. Torng, "TCAM Razor: A systematic approach towards minimizing packet classifiers in TCAMs," in *15th IEEE International Conference on Network Protocols (ICNP)*, 2007, pp. 226–275.

[29] A. Khakpour and A. Liu, "Quarnet: A tool for quantifying static network reachability," Michigan State University, East Lansing, Michigan, Tech. Rep. MSU-CSE-09-2, January 2009.