A Behavioral Approach to Worm Detection

Daniel R. Ellis, John G. Aiken, Kira S. Attwood, Scott D.Tenaglia
The MITRE Corporation
{ellisd, jaiken, kattwood, stenaglia}@mitre.org

ABSTRACT

This paper presents a new approach to the automatic detection of worms using behavioral signatures. A behavioral signature describes aspects of any particular worm's behavior that are common across the manifestations of a given worm and that span its nodes in temporal order. Characteristic patterns of worm behaviors in network traffic include 1) sending similar data from one machine to the next, 2) tree-like propagation and reconnaissance, and 3) changing a server into a client. These behavioral signatures are presented within the context of a general worm propagation model. Taken together, they have the potential to detect entire classes of worms including those which have yet to be observed.

This paper introduces the concept of an network application architecture (NAA) as a way to distribute network applications. An analysis shows that the choice of NAA impacts the sensitivity of behavioral signatures. An NAA that satisfies certain constraints significantly improves worm detection sensitivity. Mathematical models of traffic flow, NAAs, worm propagation, and worm detection provide a context for the entire discussion.

Categories and Subject Descriptors

C.2.0 [**Computer-Communication Networks**]: Security and protection; C.2.1 [**Network Architecture and Design**]: Network communications

General Terms

Security

Keywords

Network Worms, Intrusion Detection, Behavioral Signatures, Descendant Relation, Network Application Architecture

WORM'04, October 29, 2004, Washington, DC, USA.

1. INTRODUCTION

Worms—malicious processes that spread autonomously from one host to another—cause major problems in today's networks. Current worm defense begins with manual worm detection followed by damage repair. Automatic detection is particularly challenging because it is difficult to predict what form the next worm will take. However, automatic detection and response is fast becoming an imperative because a newly released (flash or topological) worm can infect millions of hosts in a matter of seconds [11, 12].

The approach to the worm detection problem introduced in this paper relies on behavioral patterns of worms that reflect network communications typical of worms. Behavioral signatures can be used to detect classes of worms and common worm implementations and designs, even if a worm has never been seen before.

The approach presented here differs from those used in contemporary enterprise postures in two ways. The first characteristic of contemporary postures is the reliance on a particular type of *signature-based* intrusion detection. In the contemporary case, a *signature* is a regular expression known *a priori* (e.g., [9]). Most signatures deployed in current intrusion detection systems (IDSs) focus on detecting specific regular expressions in network packets. The use of a previously unknown version of an exploit will evade detection.

The behavioral detection approach contrasts from this form of *signature-based* detection. Instead of looking for fixed regular expressions in payloads, the behavioral approach focuses on detecting patterns at a higher level of abstraction. Ideally, the patterns are inherent behaviors of worm spread and distinct from normal network traffic. The frequency of and interrelationships between behaviors improve detection accuracy. To evade a behavioral signature requires a change in fundamental behavior, not just its network footprint. Modifying behaviors to evade detection may be much more challenging.

For example, most classes of worms propagate by exploiting a vulnerability in a remote network service. A worm acts as a client and infects the host with the network service. The newly infected host then acts like a client as it attempts to infect other hosts. Active unimodal worms cannot easily mask the behavior of turning servers into clients and still spread.

Similarly, for a worm to spread, the worm code has to be transmitted from an infected host to the target host. Also, the infecting worm must send an exploit to a target host to infect it. In the case that the infecting host sends the

exploit and worm code from itself, there is a network traffic trace from the infecting host to every host it infects, and thence to all the hosts they infected, and so forth. This trace, when seen in aggregate preserves a tree-like structure (it is not necessarily a perfect tree structure, see Section 3 for details). An infecting host must contact a target host in order to spread. The inductive application of this rule unavoidably leads to a tree-like structure. No pure network worm can evade this signature. Section 4 discussed these and other behavioral signatures.

The second characteristic of contemporary postures is that the sensors are located around the circumference of the enterprise. This approach only monitors traffic that crosses the circumference through the gateway; it does not monitor traffic that enters through alternate ways. It also does not monitor any internal communications. Weaver et al. discuss the weaknesses of a circumference-focused perimeter and promote weaving defenses throughout the LAN [15].

This paper focuses on detection within an enterprise's network. The behaviors of interest are of greatest value when the sensors are distributed such that all internal traffic is monitored. As the data is gathered with high resolution, more meaningful data can be gathered and more precise responses executed. This supports layered defenses throughout an enterprise.

This paper considers three behavioral signatures. The first is that the inputs and outputs of a host are related for all non-discriminating worms that do not have a polymorphic network footprint. The second is that non-discriminating worms turn servers of a service into clients of the service. Together, these two signatures identify behavior, which, on a per-host basis indicates a change in logic. The third signature is identifying a tree-like structure in communication patterns emerging from infected nodes. As the worm spreads, infected hosts contact other hosts. The resulting tree-like communications have features in common, possibly including the previous two signatures.

NAA impacts the sensitivity of this behavioral approach. That is, the distribution of hosts and network applications across those hosts impacts the normal traffic patterns on an enterprise network. Under certain NAAs, constraints are placed on traffic patterns, which worm traffic patterns violate. Violations of these constraints are straightforward to detect. This paper identifies architectural designs which, if implemented, will improve worm detection sensitivity.

1.1 Related Work

The worm model presented by Ellis [3] is assumed for the basis of this work. Weaver et al. [14] discuss the known classes of worms and argue that defenses against classes of worms is a reasonable proposition. This paper presents some defenses that are relevant to several classes of worms. Section 4 articulates behavioral signatures that can be used to detect classes of worms. Section 6 evaluates the sensitivity of the signatures with respect to network construction.

Stuart Staniford's work in worm detection is the most relevant work. His research included the development of a graph-based intrusion detection system (GrIDS) [10]. While the approach was clearly relevant to the problem of worm detection, the accuracy, sensitivity, and performance of the system are not described. Staniford was the first to propose using communication patterns for worm detection. His approach used an internal graphical representation of communication patterns and demonstrated how to combine two graphs together for distributed detection. GrIDS, unfortunately, was ahead of its time and not appreciated and has not been maintained for several years. In a discussion with the former stewards of the GrIDS project, it was determined that a comparison of the approach proposed in this paper to the GrIDS implementation is not feasible because GrIDS is several years out of date and no longer supported.

Staniford also developed CounterMalice. The theoretical foundations of the technology were described in [12]. This approach is best described as scanning throttling and can be shown to be an effective mechanism to quarantine random scanning worms. Staniford's work in this area advances the work of Williamson's throttling work [17] in that it takes into consideration not just frequency of egress communications, but the frequency of failed communications, which are expected to be more common among random scanning worms. Further, Staniford's approach is network based, while Williamson's approach is host-based. Both of these approaches can be applied to an enterprise's internal networks. The demonstrated effectiveness of these two approaches is limited to the class of random scanning worms. There is no claim that these approaches will be effective at detecting other classes of worms.

Kim et al. [5] and Sing et al. [8] recently presented ways to automatically develop signatures that match on common content in network packets. This paper generalizes the approach of evaluating inputs and outputs to infected hosts as a detection signature.

Joshua Guttman presented a graphical network model [4] to describe routing policies designed to enhance network security. The model in this paper is a variation and extension Guttman's model and the activity graphs presented in [10]. The nodes in the Guttman model are subnetworks and routers. The model presented in this paper is derived from the Guttman model by first specializing to the case when each subnetwork contains a single host. Sensors are then added yielding three types of nodes: hosts, routers, and sensors. Then the model is enhanced by adding features to describe worm propagation and detection in a TCP/IP network. The data flow networks used in this model is a richer representation than the activity graphs described in [10].

The Poly^2 project at CERIAS [2] has proposed the concept of a *network service architecture* (NSA), which includes construction principles that should be applied to improve the security of a suite of enterprise network services. The concept of a network application architecture *NAA* presented in this paper expands the scope of the *NSA* to include the distribution of client and or peers in the infrastructure. This expanded scope encourages verifiability and improves the ability to detect and respond to malicious behavior.

1.2 Graph and Network Theory

The reader will notice that the definitions are consistent with graph and network theory [1], with one exception. The definition of a network is modified to allow for multiple links between nodes.

A *directed network* is a set of nodes, N, and a collection (not set) of links, $L \subseteq N \times N$. There may be more than one link between any two nodes. A network is denoted by the ordered pair, (N, L), or simply by L if the set of nodes is understood. A network, $(N1, L1)$, is a subnetwork of a network, $(N2, L2)$, if $N1 \subseteq N2$ and $L1 \subseteq L2$. The $\subseteq$ symbol

is overloaded in the expression $L1 \subseteq L2$ to indicate the subnetwork relationship as well as the subset relationship. The notation xLy is often used for $(x, y) \in L$.

Various features are attached to the nodes, links, and network itself. A node or link feature is a function whose domain is some subset of the set of nodes or links, respectively, in a given network. The domain of a network feature is a set of subnetworks of the universal network. (The graph of the network is the underlying set pair, (N, L), stripped of the features.)

If xLy, then x is the source node and y is the destination node. The link is represented graphically by an arrow going from x to y. If xLy and yLx, then the link is represented by a single bidirectional link. If the network is inherently symmetric, then the links are represented by single lines connecting the nodes.

1.3 Paper Organization

Section 2 presents the abstract communications network model. The abstract network model includes a model of the (relatively) static infrastructure and the dynamic communications and traffic flow. Section 2 also presents the concept of an NAA within the context of the abstract communications network model. Section 3 presents a model of worm propagation and presents the behavioral patterns that are codified into behavioral signatures. Section 4 derives behavioral worm signatures from the communication patterns articulated in Section 3. Section 5 presents the concept of an NAA. Section 6 presents an analysis of the impact of NAAs on worm detection sensitivity. Section 7 restates the contributions in this paper. Section 8 describes ongoing work that will advance the ideas presented in this paper.

2. ABSTRACT COMMUNICATIONS NETWORK MODEL

This section defines the abstract communications network (ACN) model. The ACN is a network-theoretic model of computer networks and the data flows that cross them. The worm propagation network (Section 3), behavioral signatures (Section 4), and NAAs (Section 5) are expressed within the framework of the ACN. There are many aspects of a realistic network (e.g., repeaters, firewalls) that are not included in this model because they are not directly relevant to the worm detection mechanisms presented in this paper.

The ACN consists of nodes that represent three elements (hosts, routers, and sensors) and four types of links (hardware, routing, data flow, and spans). The hosts include both user workstations and servers. The routers compute the routing tables, enforce routing policy, and perform the actual switching of the message flow. The sensors collect a subset of the traffic on the network. The (symmetric) *hardware* links represent a physical communications medium whereby data can flow from one computational element to another. The *routing* links represent the direction that data gets routed from one host to another. The *data flow* links represent conversations between computational elements. The primary focus in this paper is on the *data flow* network, F.

F captures the semantics of end-to-end communication between two hosts. For each conversation between two hosts in the computer network there exists one link between those two hosts in F. It is possible for there to be multiple links between two nodes just as there may be multiple conversations between two hosts in a computer network. Each link in F preserves the features of the respective conversation, including the headers, content, and timing information. For example, the source and destination IP addresses, source and destination ports, TCP flags, payload content, start and end times (assume a master clock), etc., are included as features of the respective links.

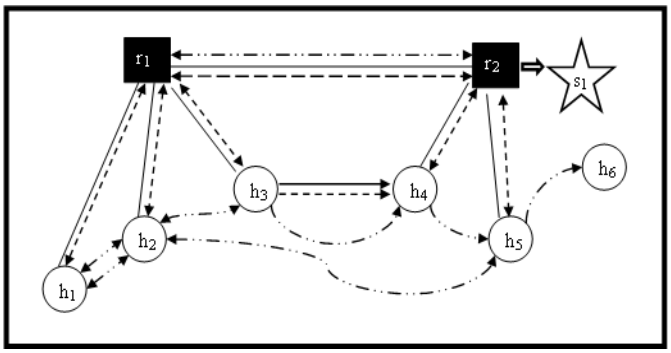

Figure 1: A sample ACN

A simple ACN with six hosts, two routers, and three types of links is pictured in Figure 1. The black boxes represent the routers, the open circles represent the hosts and the star represents a sensor. Each link type represents a separate component subnetwork. The most significant difference between the subnetworks is in the semantics of their links.

The link in Figure 1 between s_1 and r_2 represents a spanning link. A sensor that sees all packets that pass through a router *spans* the router. Any router that is spanned by a sensor is a monitored router. A sensor has access to all features of a data flow link that passes through the monitored router it spans. Throughout this paper, sensor deployment is assumed to allow visibility to all data flows associated with any host in the enterprise network. Practically, every access-layer switch in the enterprise is monitored by a sensor.

3. WORM PROPAGATION MODEL

The worm propagation model extends the ACN model to include worm spread across a network. This section presents the descendant relation, which captures communication patterns that emerge as a consequence of worm spread and provides the foundation for the detection model described in Section 4.

This section is outlined as follows. Section 3.1 shows how to extend the ACN model of Section 2 to include the links representing worm propagation. Section 3.2 presents the descendant relation, a formal model of communication patterns evidenced by worm spread.

3.1 The Worm Propagation Network

A worm must communicate over the network in order to spread. These network communications form links in the ACN (see Section 2). The worm propagation network is constructed to represent the worm spread across a network from infected host to infected host.

Definitions Let (N, L) be the data flow network (F) of the ACN, $(N, N \times N)$. The worm propagation network,

$W = (I, Q)$, is a related to (N, L). I is the set of infected nodes in N. Q is a set of links, (a, b), where a worm on a infected b. For each link in Q, there is a sequence of links in L with the appropriate features to constitute the infection vector (IV); there may be multiple IVs in W.

The worm propagation network, $W = (I, Q)$, is constructed from the communication network, (N, L). W is assumed to start with a set of initially infected nodes, I_{t_0}, and an empty set of links, Q_{t_0}, at time $t = t_0$. A node b and link (a, b) are added to I and Q, respectively, when an infected node a first propagates to it via an IV and successfully infects it. The infection time of a node is the time at which the first infection process completes or t_0, if it is in I_{t_0}. Features of the links in L that constitute the IV preserve the links in Q. The start time of an infection link (a, b) in Q is the time when the communications that cause b to be infected are initiated by a and its end time is the infection time.

W does not capture subsequent reinfections. As events with respect to a single node in the communication network are totally ordered, the identification of the first infection is deterministic. If a infects b, then a's infection time is strictly before the start time of the infection link (a, b), which is strictly before b's infection time. Consequently, a spanning tree proceeds from each node in I_{t_0} where the infection time of a node is strictly greater than that of its parent.

Another attribute of worms is evident in their communication patterns. In order to infect a target host an infected host must exploit it in some way. That means that the worm must send some data across the network that is engineered to force the target host to enable infection. The worm code itself also has to be placed on the target host (assuming it's not already present). Both of these events have a network footprint. The exploit must be sent from the infecting host to the target host, and the target host gets the worm code either from the infecting host or some other host on the network. Since worms have a finite (usually small) number of exploits, the payloads communicated from infected hosts to target hosts is finite. For unimodal worms (worms with only one exploit and infection mechanism), all infections require the exploit. The property that the exploit (and possibly the worm code) is communicated from the infecting host to the target host is expressed as a link feature.

Figure 2 shows a simplified data flow network as a worm spreads from node h_1 to other nodes. The network is simplified in that the *hardware* and *routing* subnetworks are not represented. Also, in Figure 2 only those data flow links that are associated with an ICMP echo request or echo reply (represented as dashed links) or TCP traffic destined to port 80 (represented as solid links) are represented. The direction in which the link was initiated is depicted with the larger arrow (e.g., h_1 sent h_2 an ICMP echo and h_2 replied). The links are numbered to show their relative timing; these links happen to be totally ordered for clarity. TCP links in which an exploit and worm code is sent from client to server are thickened. Other link features are not shown in Figure 2.

Figure 2 shows a worm with an IV that includes an ICMP echo request (followed by a reply) followed by a TCP connection to port 80. If either transaction in the IV is not completed or if the transactions are not in order, the infection does not spread. This worm has one infection vector, IV, which is necessary for propagation. It consists of:

- an ICMP echo request from a to b, followed by

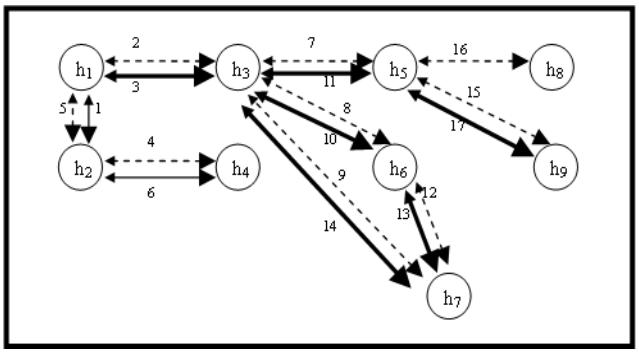

Figure 2: Simplified ACN showing worm spread

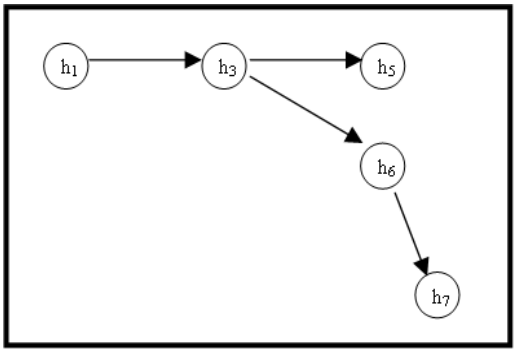

Figure 3: Worm propagation network of Figure 2

- an ICMP echo reply from b to a, followed by
- a TCP SYN from a to b, followed by
- a TCP SYN/ACK from a to b, followed by
- an exploit and worm code from a to b.

Figure 3 shows W constructed from the ACN shown in Figure 2 and the IV described above. Direction is the only feature depicted.

A few of the nodes in Figure2 are not in W. h_2 is not in W because the IV was not completed: the TCP port 80 data flow precedes the ICMP data flow, which does not constitute an IV. This indicates that non-worm traffic in the network may resemble worm traffic. h_4 is not in W because h_2 was not infected when they communicated, even though there were ICMP and TCP data flows. h_8 is not in W because the IV from h_5 is incomplete. The link (h_3, h_7) is not in W because h_7 was already infected ($h6$ infected it first). Although the links between h_5 and h_9 constitute a valid IV, h_9 is still not added to W. The reason in this case is due to the system conditions not being satisfied (e.g., h_9 is not vulnerable). Otherwise, the construction of W from the communications network in Figure 2 is straightforward.

Three observations about the worm propagation network are relevant to the remainder of this paper. First, the construction of the worm propagation network guarantees that it consists of only spanning trees, each rooted at a node

in the initial infection set. Historically, some worms have reinfected hosts multiple times, which, with a different construction may lead to a worm propagation network that has cycles in it.

Second, the ordering of transactions in the communications network is significant and the resulting worm propagation network only contains infected hosts with a link indicating the node that infected it (unless the node is in the initial infection set).

Third, construction of the worm propagation network requires more than just knowledge of the ACN. It requires knowing ground truth about the other system conditions (i.e., visibility, vulnerability, and executability) described in [3]. The fact that an infected host employed communications in a way that satisfies the worm's link predicate with a target host does not imply the target host is infected. The target host must also satisfy the system conditions, which are not represented in the ACN model or captured in the worm propagation network.

3.2 The Descendant Relation

The worm propagation network consists of a set of spanning trees, where each tree's root is in the initial infection set. It is instantiated with the hosts in the initial infection set. The construction continues by adding links to the worm propagation network when events defining worm propagation occur in the data flow network. A link is added to the worm propagation network when the IV and the system conditions hold true in the data flow network. This section generalizes the process of developing spanning trees using an arbitrary predicate, the link predicate, over the links between two hosts in the data flow network.

The link predicate specifies conditions that must be true for a node to *descend* another. The descendant relation covers the set of nodes reachable from a node by following links that satisfy the link predicate. The conjunction of the IV and system conditions is an example of a link predicate.

Definitions Let (N, L) be the data flow network of the ACN, $(N, N \times N)$. A *link predicate*, $P_t(a, b)$, is true with respect to two nodes in N, a and b, at time t if there are links between a and b in L that satisfy P at time t. A link predicate can be arbitrarily defined with respect to N and L. The following are sample link predicates. $P_t(a, b)$ is true iff at or before t:

- a sends b a UDP packet to port 139
- a sends a TCP SYN packet to port 80 on b then b replies with a SYN ACK packet
- a establishes a TCP connection with b, a sends a TCP FIN packet, but b never sends a TCP RST or TCP FIN packet
- a sends an ICMP echo request to b, then b sends and ICMP echo reply to a, then a establishes a TCP connection with b on port 22

Definitions Node a *descends* b (at time t) with respect to link predicate P if $P_t(a, b)$ is true. The descendant relation, D_a, is the set of all descendants of a. The notation for the descendant relation is:

$$b \in D_a(P_t, L) = \bigcup_i D_a^i(P_t, L)$$

This reads "b is a descendant of a at depth i with respect to link predicate P over the set of links L at time t." The time, depth, and link set are omitted when they are clear from context or insignificant.

Given network (N, L), $a, b \in N$, the descendant relation, $D_a(P, L)$ is recursively defined with respect P and L as:

Base Cases:

$a \in D_a^0 P_t \quad \forall P_t$

$b \in D_a^1 P_u \quad$ if $\exists t \mid (a, b) \in L \wedge P_t(a, b) \wedge u > t$

Induction Case:

$c \in D_a^i P_u \quad$ if $\exists b, t \mid b \in D_a^{i-1}(P_t) \wedge (b, c) \in L$
$\wedge P_u(b, c) \wedge u > t$
$\wedge \neg \exists j \mid c \in D_a^j$

The condition $u > t$ implies that a descends b strictly before b descends c.

A descendant tree is the result of applying a descendant relation (with a specified link predicate) with respect to some parent node to a data flow network.

Figure 3 contains the descendant network with respect to h_1 of Figure 2 when the link predicate is the logical conjunction of the IV and the system conditions. The worm propagation network is the special case when the link predicate is the conjunction of the IV and the system conditions.

Using a different link predicate would produce a different descendant network with respect to h_1 of Figure 2. If the link predicate is

$$P(a, b) = \{ \begin{array}{ll} \text{True} & \text{if } a \text{ sends ICMP echo request to } b \\ \text{False} & \text{otherwise} \} \end{array},$$

then all other nodes except for h_4 are descendants. Because h_2 sent the ICMP echo request to h_4 before h_1 descended h_2, h_4 is not a descendant of h_1.

Using the link predicate,

$$P(a, b) = \{ \begin{array}{ll} \text{True} & \text{if } (a, b) \text{ satisfies the } IV \text{ and} \\ & b \text{ satisfies the system conditions} \\ \text{False} & \text{otherwise} \} \end{array},$$

on the communication network of Figure 2, the worm propagation network of Figure 3 is obtained.

The descendant relation is a significant contribution of this paper. It advances the concept of using communication patterns to detect worm spread introduced in [10] by defining and formalizing the descendant relation.

4. BEHAVIORAL SIGNATURES

This section describes behavioral signatures for worm detection and the supporting mathematical framework. Behavioral signatures are valuable because they describe classes of worms without needing to know the specifics of a worm *a priori*. The behavioral signatures presented in this section reflect the discussion on the worm propagation model (Section 3). Section 6 evaluates the sensitivity of these signatures with respect to different worm classes and NAAs.

The worm signatures presented here are behavioral. Three of them (changes from server to client, alpha-in-alpha-out, and fanout) capture behaviors that can be monitored by watching all ingress and egress data flows for a single node and are called *base signatures*. The other (descendant relations) is an *inductive signature* because it applies to a behavior that occurs across nodes. All of the behavioral signatures here are designed to answer the question "Is host

x infected?" Although a descendant relation is an inductive signature, it is still measured with respect to a specific host.

The server-to-client and $\alpha - in - \alpha - out$ signatures are presented before the descendant relation. The fanout signature is shown to be a special case of the descendant relation. An analysis of the sensitivity of these signatures is presented in Section 6.

4.1 Server Changes to Client

When a server becomes infected with an active worm, the worm attempts to infect other hosts. To do this, it initiates connections to other services that the worm knows how to exploit. As it knows how to exploit the service whereby it infected the current host, it may use the same exploit to infect other hosts. When this happens the infected host changes in behavior; it acts as a server of a service until it is infected, after which it acts as a client of the service. The way this signature is formulated with respect to the ACN follows.

To introduce a mathematical notation for this behavior, let $< a, b, c, d, t >$ denote the event that an IP packet was transmitted from port b on host a to port d on host c at time t . That is, (a, b) represents the client address and port and (c, d) the server address and port (note: this differs from the tcpdump format). Then, the signature matches if there exists two times $t1$ and $t2$ such that $t1 < t2$, and packets of the form $< a, ?, c, d, t_1 >$ and $< c, ?, ?, d, t_2 >$ are observed, where ? is the wild card. A hollow arrow can be used to indicate time sequencing, in which case the time parameter is omitted. The shorthand notation for the signature is $< a, ?, c, d > \rightarrow < c, ?, ?, d >$.

A more restrictive version of the signature, which is only meaningful when a node is defined by policy to be only a server of any service, detects when the node acts like a client of any service (of the service bound to port d). The restricted server-to-client signature is $< a, ?, ?, d >$, where a is a server-only node and d is a well-known service port. Notice, a single packet is adequate for detection when a is defined by policy to be only a server of service d and never a client of d.

This signature reflects the behavior that worms, containing their own logic, change the behavior of infected hosts to execute that logic. The accuracy of this signature depends on the NAA (see Section 5).

4.2 $\alpha - in - \alpha - out$

For a worm to spread, an infected host needs to complete a well-defined behavior—one of its (IVs)—with respect to a target host. For each IVs there is a link predicate, PIV, that is true over the respective data flow links. For unimodal worms and worms that use all IVs it has indiscriminately, there will be a pair of matching ingress and egress data flow links. That is, if for any such worm,

$$\exists P : P_i(a, b) \wedge P_j(b, c) \wedge i > j$$

must hold true for some P at times i and j. Although this is the inductive case for worm propagation and intuitively indicates that some behavior that transpired between a and b later transpired between b and c, it is a *base signature* because it can be applied to any single host without any recursion. The $\alpha - in - \alpha - out$ signature is "a sends content to b that b later sends to c". Because at least an exploit must cross the network from an infecting host and the exploit must satisfy some structural properties to be effective, there is some relationship between the contents of payloads of the infecting ingress and egress data flow links. The sending of code in the same data flow links strengthens the signature.

This signature reflects the behavior that worms relay similar content across infected hosts. The accuracy of this signature will be limited for services, like file servers or SMTP servers, where the input (content payload) is very similar to or identical to the outputs.

4.3 Inductive Signatures

The descendant relation is the most significant contribution of this worm detection approach. The idea is that worms are fixed in the logic they execute and the number of infection vectors they employ. Consequently, as a worm spreads from an initially infected node, the number of descendants grows: the number of descendants at each depth grows as does the depth of the most distant descendant. For a worm to spread rapidly, each infection must infect more than one other host on average; that is, each worm must infect more than one other host, or it will fail to spread exponentially [12]. The consequence of this rapid spread is an increasing number of descendants at each level in the descendant network.

There is an inductive relationship between infected hosts. Every new infection (excluding those in I_{t_0}) has a network communication leading to it from an infected host. The infection set is the base case and the infection vectors are the inductive steps (reflected as link predicates in the ACN). Applying this definition recursively yields all hosts infected by the initial infection set.

A descendant relation, $D_a(P, L)$, is the set of hosts that descend from a node a via some link predicate P within the context of some set of links L. When L or P is clear from context, it is omitted. All worms spread such that there exists some P such that all infected nodes are in $D_a(P)$ for some $a \in I_{t_0}$.

For a descendant relation to be captured as a signature it requires a threshold and a link predicate. For example, thresholds can be established for each of the following with respect to some link predicate, P:

1. $FD_t = max(i)$ such that $D_i(P) > 0$: the depth of the furthest descendant at time t.
2. $\sum |Di(P)|$: the number of descendants.
3. $\sum(|D_{i+1}(P)|/|D_i(P)|)/FD$: the average branching factor, if this is greater than one, then the epidemic threshold is reached.
4. $(time(FD_n) - time(FD_1))/(n)$: the average propagation time from one generation to the next. $time(FD_i)$ is the time the first descendant at depth i is added to the descendant relation, $time(FD_i) = min\{t|FDt = n\}$.

The rationale for the signatures above is that as a worm spreads, the depth and breadth of the subsequent descendant tree grows from the root, which is an infected host. 1) is a measure of the depth of the tree. 2) is a measure of the number of descendants in the tree. 3) measures the average branching factor of the descendant network. If greater than one, the average branching factor indicates a descendant network that is growing exponentially. 4) measures the time it takes to get to a specific depth. The time to depth

1 is not meaningful, however, as generations are added in rapid succession, it grows increasingly likely that an automated process, like a worm, is controlling the spread.

Each of the above signatures can be calculated for all a in F. Any time a threshold is breached there is an indication that a worm is spreading from a. Ideally, the signature is calculated for the most discriminating link predicate, which may vary from worm to worm. Clearly a link predicate of "a sends IP packets to b" will be sensitive, but may not be adequately accurate.

In the case where there is non-worm communication on the network, the link predicate needs to have discriminating features. Otherwise, uninfected hosts will also be in the descendant relation (i.e., accuracy will be poor). Each worm has its own propagation mechanism, which may be distinct from other traffic. The more prolific a node is with respect to a discriminating link predicate, the more accurate the detection mechanism.

A few sample link predicates for descendant relations are provided in Section 3.2. Some link predicates that capture some historical worms are:

- $P_{LionWorm}(a, b) =$
  - a establishes and tears down a TCP connection with port 80 on b
  - a establishes another TCP connection with b and over the connection
    - a sends an exploit for bind to b
    - a sends a bootstrap script to b
    - b does not close its side of the connection
  - b contacts c (URI = 207.181.140.2:27374)
  - c sends Lion Worm source code to b
- $P_{Slammer}(a, b) =$
  - a sends a UDP packet to port 1434 on b with an exploit and code
- $P_{Blaster}(a, b) =$
  - a connects to TCP port 135 on b and sends an exploit
    - b does not close its side of the connection
  - b sends a UDP packet to port 69 on a
  - a replies with UDP packet with worm code
- $P_{Welchia}(a, b) =$
  - a sends an echo request to b
  - b sends an echo reply to a
  - a connects to TCP port 135 on b and sends an exploit
    - b does not close its side of the connection
  - b sends a UDP packet to a port between 666 and 765 on a
  - a replies with UDP packet with worm code
- $P_{Sasser}(a, b) =$
  - a establishes a TCP connection with port 445 on b and sends an exploit
    - b does not close its side of the connection
  - a establishes a TCP connection with port 9996 on b and sends instructions to download the worm code
  - b establishes a connection with port 5554 on a and downloads the worm code

Half-open connections are a common occurrence in worm traffic. The reason is that recovering a socket after a buffer overflow is difficult. If a socket does not get *shutdown* by the application, its side of the connection remains open. The utility of this observation is that link predicates can be generalized to capture classes of worms. In this case, the class is defined by an implementation that fails to terminate a TCP connection.

Fanout has been shown to be an effective detector of indiscriminate scanning worms [17, 12]. Fanout is special case of the descendant relation where only the descendants at a depth of one are considered. That is, the approach is to calculate $D_0(P)$ for all hosts and block any host where it is greater than some threshold. The link predicates for each are:

- $P_{Williamson}(a, b) = a$ sends an IP packet to b
- $P_{Staniford}(a, b) =$
  - a sends an IP packet to b
    - Either
      - b replies with a RST
      - An ICMP host b unreachable message is returned
      - no response is seen before a timeout occurs

The descendant relation is more sensitive than fanout. Fanout, like the previous two behavioral signatures can be evaluated on a host-by-host basis by monitoring the traffic ingress and egress for each host. The descendant relation looks at behaviors across hosts. It can detect behaviors that drop below the thresholds for detection at a single host because the same behaviors are witnessed across hosts. For example, the descendant relation would be sensitive to a worm that contacted no more than 2 hosts per second. With a reasonable targeting algorithm, such a worm could saturate an enterprise in minutes and still not be detected by the fanout signatures.

To summarize this section, the descendant relation is most useful when defined by link predicates that describe discriminating communication patterns that are common to worm implementations (e.g., the base signatures, half-open connections). Patterns in reconnaissance, protocol usage, and payload content all propagate from infection to infection and make for sensitive link predicates. For every worm there exists at least a trivial descendant relation that is perfectly sensitive. However, as the number of a worm's infection vectors increases, so does its inconspicuousness. The accuracy of trivial descendant relations is a weakness. Also, if a worm can spread passively (i.e., a contagion worm), it is less distinct from normal communication patterns between clients and servers.

5. NETWORK CONSTRUCTION

An NAA defines how an enterprise will distribute the functionality of its network applications across its network. For each NAA there are different properties that are satisfied over the subsequent data flow networks. This section describes the network service models, the concept of an NAA, and the impact of an NAA on the properties of data flow networks. Section 6 analyzes the impact of an NAA on the behavioral signatures described in Section 4.

5.1 Network Service Models

There are two network service models that have been influential in partitioning the functionality of network applications across hosts: the client-server model and the peer-to-peer model.

5.1.1 Client-Server Model

The client-server model distributes functionality across a network in two pieces. Each participant in this network application is one of two pieces: a client or a server. The client is the piece that desires data and functionality; the server is the piece that provides them. Both the client and the server are programs running on hosts and are bound to layer four ports on their respective hosts. The client initiates communications with the server and the server replies accordingly. A typical transaction consists of a request followed by a reply. Additional traffic may flow over this same channel in both directions as well. The parameters of the data flow are preserved as link features in F. The direction of initiation (i.e., from client to server) is depicted with a larger arrow than the return flow (e.g., h_1 iniated a connection with h_8 in Figure 4).

5.1.2 Peer-to-Peer Model

The peer-to-peer (P2P) model decomposes into only one type of network application piece: the peer. A peer is equivalent in functionality to both a client and a server bundled into one atomic component. Each peer can initiate and respond to communications from other peers.

5.2 Network Application Architectures

An NAA is a description of how a specific enterprise distributes network applications across its network. This includes a description of where clients, servers, and peers of the various network applications are to be located across a network. This section presents three NAAs. Section 6 contains an analysis of the impact of the choice of NAA on worm detection.

This section assumes the distinctness of server and client ports in accordance with conventions [6].

5.2.1 Pure Client-Server NAA

A pure client-server NAA (PCSNAA) is an architectural pattern that prohibits any host from being both a client of a network service and a server of any other. That is a network follows the PCSNAA pattern if:

> $\forall a, a \in N$ { a | a has only services on it or a has only clients on it}.

An analysis of the resulting data flow network for such an architected network results in the following conclusions. Each server node is the root of a tree of depth one with possibly large breadth. Each node that has one or more a clients on it initiates a communication with possibly multiple servers. Figure 4 gives an example of a network with several client nodes ($h_1 - h_7$) and two server nodes (h_8, h_9).

Figure 4 depicts a sample network constructed to satisfy a PCSNAA. There are hosts that are clients only and hosts that are servers only. Two of the hosts (h_3 and h_4) initiated communications with both servers. Data flow trees of depth one and possibly large breadth are characteristic of all F that satisfy the PCSNAA. This pattern is characteristic of all PCSNAA, independent of what network applications are used and what layer four ports the servers and clients use (so long as the ports are unique).

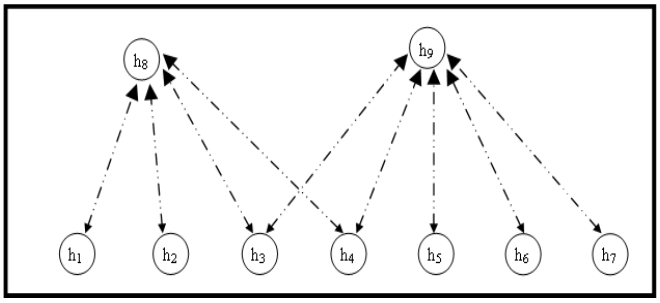

Figure 4: PCSNAA example

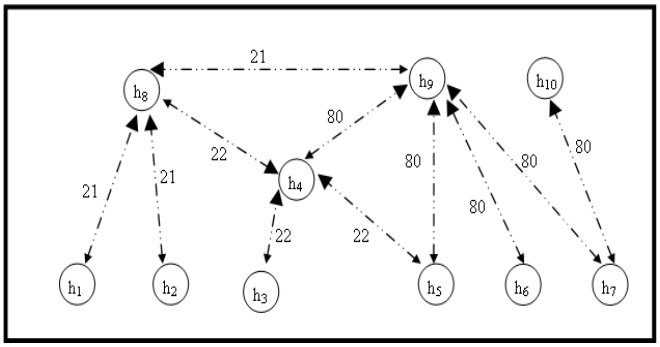

Figure 5: CSNAA example

5.2.2 Client-Server NAA

A client-server NAA (CSNAA) is an architectural pattern that prohibits any host from being a client of a network service and also being a server of the same network service. A network follows the CSNAA pattern if:

> $\forall a \in N$ { a | for any service S, a is not both a server and client of S}.

Thus, a host may be a server of one service and a client of another. A DNS server in recursive mode does not satisfy CSNAA. However, if it does not operate in recursive mode and receives updates out-of-band (e.g., using FTP), then it does.

Figure 5 provides a sample depiction of a data flow network that satisfies CSNAA. To more clearly depict a significant pattern in Figure 5, each link is labeled with the destination port associated with the server node. It follows from the definition of CSNAA (and the assumption of unique service ports) that no node will both reply to traffic destined to a port and also initiate communication destined to the same port on another host. However, a node may reply to traffic destined to one of its service ports and initiate communication destined to another.

There are two patterns that are properties of networks that satisfy CSNAA. The first is that cycles in F are pos-

sible. To demonstrate the second pattern, a new graph $F_p = (N, L_p)$ is constructed such that it only has links with destination port equal to p. For each p, F_p has no cycles and each node is a server or a client or an isolated node. Each client in F_p may have links to more than one server. In Figure 5, F_{80} has two clients (h_6, h_7) and two servers (h_9, h_{10}) and F_{22} has three clients (h_3, h_5, h_8) and one server (h_4).

5.2.3 Ad-hoc NAA

An ad-hoc NAA (ANAA) is a pattern that makes no prohibitions against the relationships between any client, server, or peer of a network application. Contemporary university host laboratories typically satisfy ANAA. Each workstation typically has the common network services (e.g., FTP, HTTP, Telnet, SSH, Rlogin, Finger) available and students access the services of neighboring hosts on the network.

It is not clear if there are patterns in ANAA networks that are useful in worm detection. One value that an NAA offers is the ability to reason about communication patterns. Whereas, the two previous NAAs offer patterns, the violations of which can be easily detected, for ANAA, there are no such patterns. The following section, shows that worm propagation stands out more clearly in data flow networks that satisfy one of the previous two NAAs. The lack of such constraints on data flow networks that satisfy ANAA is disturbing, because they may be more difficult to protect from worms.

6. NAA IMPACTS ON DETECTION

This section focuses on the analysis of architectural impacts on the sensitivity of the proposed signatures. However, some initial (incomplete) analysis of the detection performance and the feasibility of constructing such networks are briefly presented. No analysis on the accuracy is presented.

Definitions The *modality* of the worm is defined by the number of infection vectors the worm has; a unimodal worm has one infection vector and a multimodal worm has multiple. A *context-dependent* worm is one that uses a proper subset of its infection vectors to spread from any particular host; a non-discriminating worm uses all infection vectors available to it. A *network polymorphic* worm is one that changes its network footprint so that it is never the same across infections. A worm is *active* if it initiates connections to target hosts in order to spread. Conversely, a *passive* worm waits for a target to connect to it (i.e., a contagion worm [11].

6.1 Sensitivity Analysis

Two factors affect the sensitivity of the behavioral signatures discussed in Section 4: the NAA and the type of worm being detected. This section analyzes the sensitivity of the signatures proposed in Section 4. The expected accuracy of each signature is also discussed, although inadequately due to a lack of data. The conclusion of this section is that behavioral signatures can be used sensitively to detect classes of worms.

Weaver et al. [14] describe classes of worms based on their target acquisition functions (TAF). The TAF is not an adequate discriminator for the sensitivity of the behavioral signatures. There are three new features of worms that affect the sensitivity of these behavioral signatures that have not been adequately described in the literature. These are particularly significant for evaluating the sensitivity of the host-based signatures.

Table 1 presents the results of the sensitivity analysis of two base behavioral signatures (the server-to-client ($S \rightarrow C$) and $\alpha - in - \alpha - out$ (α) signatures) for different NAAs. The restricted server-to-client signature (Section 4 is used in the Pure Client-Server NAA. A "Mod." of 0 means a unimodal worm; a 1 means a multimodal worm. A "Cont.-Dep." of 0 means the worm uses all exploits from an infected node; a 1 means the worm uses a proper subset of exploits from any given node. A "Poly." of 0 means the *network* footprint for each infection vector is constant across infections; a 1 means that the same infection vector has a different network footprint for each use. An "Act." of 0 means that the worm initiates data flows to targets; a 1 means the worm is a contagion worm and waits for connections from targets passively. The four rows (4-7) indicating some type of unimodal, context-dependent worm have been removed because such a worm does not exist by definition.

For each <worm-type, signature , NAA> tuple, an evaluation of "absolute" (Abs), "statistical" (Stat), or "inconclusive/ineffective" (In) is given for a class of worm. An "absolute" is given for those signatures where detection has provably perfect sensitivity (the signature will always be triggered for the class of worm and NAA). A "statistical" is given for those signatures where there is expected to be a discriminator between worm and non-worm traffic, but where thresholds may be difficult to determine and vary from environment to environment. An "inconclusive/ineffective" is given in cases where no reasonable argument indicates the signature will be better than random discriminators.

Table 1 highlights the contrast in sensitivity based on NAA. Any enterprise that is willing to architect their networks according to PCSNAA or CSNAA can achieve a significant improvement in detection capability using the signatures presented in this paper. Unfortunately, many networks are architected according to ANAA.

A server-to-client signature ($S \rightarrow C$) is perfectly sensitive (Abs) in a PCSNAA by an active worm (rows 1, 3, 5, 7, 9, 11, 13, 15). If it ever initiates communications, which it must do if infected by an active worm, the signature is triggered. In the case of a CSNAA, the signature is triggered in all cases unless the worm is discriminating, in which case it may choose not to use the exploit egress with which it was infected ingress. The statistical sensitivity for multimodal, discriminating worms (rows 12-13) is determined based on the likelihood that the worm will use the same exploit on ingress to and on egress from a node. A contagion worm bypasses the server-to-client signature altogether (rows 0, 2, 8, 10, 12, 14). A unimodal contagion worm (rows 0, 2) requires either an exploit that can infect a peer or an exploit that can infect both a server and a client. If the former case, the peer is required to behave only like a client or only like a server but not like a peer in the PCSNAA and CSNAA.

The $\alpha - in - \alpha - out$ signature is sensitive when the worm is neither discriminating nor network polymorphic (rows 0, 8, 9). The statistical sensitivity for multimodal, discriminating worms that are not network polymorphic (rows 12-15) is determined based on the likelihood that the worm will ever use the same exploit on ingress to and egress from a node.

Table 1 only includes base behavioral signatures and does not discuss inductive behavioral signatures (e.g., signatures

Row	Worm Properties				Signature Sensitivity by (NAA x Signature) PCSNAA		CSNAA		ANAA	
	Mod.	Cont.-Dep.	Poly.	Act.	$S \rightarrow C$	α	$S \rightarrow C$	α	$S \rightarrow C$	α
0	0	0	0	0	In	Abs	In	Abs	In	Abs
1	0	0	0	1	Abs	Abs	Abs	Abs	In	Abs
2	0	0	1	0	In	In	In	In	In	In
3	0	0	1	1	Abs	In	Abs	In	In	In
8	1	0	0	0	In	Abs	In	In	In	Abs
9	1	0	0	1	Abs	Abs	Abs	Abs	In	Abs
10	1	0	1	0	In	In	In	Abs	In	In
11	1	0	1	1	Abs	In	Abs	In	In	In
12	1	1	0	0	In	Stat	In	Stat	In	Stat
13	1	1	0	1	Abs	Stat	Stat	Stat	In	Stat
14	1	1	1	0	In	In	In	In	In	In
15	1	1	1	1	Abs	In	Stat	In	In	In

Table 1: Base behavioral signature sensitivity analysis by NAA

based on a descendant relation). The sensitivity would depend on the link predicate and thresholds chosen. Clearly for the link predicate "a sends IP packets to b", a threshold of 1 would be perfectly sensitive. It would, however, have a large false alarm rate, as it would trigger on all communications.

Several factors affect the sensitivity of inductive signatures: the target acquisition function, the sensor topology, and the implementation of the worm. The more infections that happen across monitored routers, the more accurate the descendant relation will be. Also, if the worm uses reconnaissance or unique protocol implementations (e.g., not closing a TCP connection), a highly discriminating signature emerges.

Although most contemporary networks do not satisfy PCSNAA or CSNAA and the base behavioral signatures may have limited detection sensitivity, the inductive signatures do. Inductive signatures may prove to be adequately sensitive even in ANAA networks. A descendant relation can be sensitive. The question is what link predicates and thresholds to use. The values for both can be empirically determined (and are the focus of ongoing research).

6.1.1 Performance Analysis

Performance of the $S \rightarrow C$ signature can be very fast. If the servers and clients are known *a priori* then detection can be made based on a single packet using a look-up chart. If the servers and clients are discovered dynamically, then two packets are necessary, but detection on the second is trivial.

Performance of the α depends on the mechanism used to identify the common signal. Two recent papers describe the performance of their mechanisms [5, 8].

Performance of the inductive signatures is exponential in the worst case scenario, which poses a problem to real-time performance. However, network construction impacts performance as it does sensitivity. The more constrained (that is, like PCSNAA) a network is the less state is created and maintained. It is also not yet clear to what depth data structures need to be maintained. Initial analysis is that a depth of 3-5 will be adequate for most enterprise networks.

6.1.2 Feasibility Analysis

Although the limited detection sensitivity of the base signatures in ANAA networks appears to be a negative result, the advantages of PCSNAA and CSNAA can be reaped incrementally. Every node that satisfies PCSNAA or CSNAA can be monitored sensitively with the base signatures. Many networks existing today, in varying degrees, may have nodes or subnets that satisfy PCSNAA or CSNAA. Each such host or subnet can already benefit from the advantages. This greatly decreases the barrier to entry and can provide immediate benefit.

Consider designing an enterprise network to optimize worm detection while preserving the functionality necessary to get meaningful work done. This example illustrates a simple construction intended to show some improvement over the status quo, but which may be suboptimal for some design constraints.

Such a network would have all of the network applications necessary. The network would be partitioned into three logical parts: PCSNAA, CNSAA, and ANAA. All workstations would be put into PCSNAA. Users need only be clients. P2P applications would be forbidden in the PCSNAA. The users could still use FTP, SMTP, HTTP, SSH, and other network clients. Windows hosts would be put in a domain (and not a workgroup) and forbid local fire sharing.

The hosts in CNSAA would be primarily servers. Where other resource constraints require a minimum number of hosts, server processes can be aggregated onto hosts. (Minimizing the number of network servers on a host improves other aspects of security, but not necessarily worm detection.) The hosts in this logical part of the network interface with the rest of the network as servers only. For example, a host may be both a file server and a web server. Also, a host may have a back-end system that only it interfaces (e.g., a web server with a database back-end). Regardless, the rest of the network sees only the web server.

The final logical partition, the ANAA partition, contains those hosts that, for business purposes, must act as peers. There are a handful of protocols in wide use across the Internet that are implemented using peering relationships whether by functional necessity or only by *de facto* implementation. DNS and SMTP are two applications that fit into this category. (Routing protocols are not considered, as this partition is for end-hosts only.) General computing resource labs (e.g., clusters of high-power workstations) that allow users to SSH from host to host may act like peers as

well. Application proxies (e.g., web proxies) would also be in the ANAA partition.

The partitions can be distributed over various LANs, however, a LAN should consist of only one partition. Each LAN could then be independently spanned and covered with a minimal amount of logic.

The enterprise network as a whole may not be PCSNAA, however, the number of hosts in the more constrained partitions is maximized. The advantages of such partitions can be gained incrementally. Worm detection is improved as the number of hosts increases in more constrained partitions and decreases in less constrained partitions. This is a significant advantage. Implementing such a strategy requires little initial effort and the payoffs are immediate.

7. CONCLUSION

The most significant contributions of this paper are the presentation of the behavioral worm detection approach and the analysis of its sensitivity. The behavioral detection approach has the ability to detect classes of worms without *a priori* information on any specific worm. The behaviors signatures were developed using the definition of a worm as a guide. The base behavioral signatures reflect the propagation of logic into and out of and the change in logic on an infected host. The inductive behavioral signatures reflect the propagation aspects of worms.

Another contribution is the notion of a network application architecture. Interestingly, the NAA significantly affects the sensitivity of behavioral signatures. The science of constructing secure networks is weaker than the analogous science in constructing secure software. This paper interjects an initial set of constraints, phrased as architectural patterns, that can be used to improve worm detection sensitivity, an important element of network security.

A negative result is presented that ad-hoc network application architectures negatively impacts worm detection sensitivity. However, an encouraging result is that NAAs can be applied incrementally and derive incremental improvements in detection sensitivity. Inductive signatures, expressed as descendant relations, can be used to detect worms as they spread through a network, independent of the NAA and may provide significant benefits in unconstrained environments.

8. FUTURE WORK

The relative accuracy of the signatures presented needs to be analyzed. MITRE's corporate network is being instrumented with sensors to evaluate the accuracy of the signature empirically. An analysis of the sensitivity, accuracy, and performance characteristics of the approach will be reported hereafter.

9. ACKNOWLEDGMENTS

The MITRE Technology Program is funding the Automated Worm Detection and Response research project, the auspices under which this work was performed.

Paul Ammann, Sushil Jajodia, and Nicholas Weaver provided useful feedback on the approach at various phases in its development.

10. REFERENCES

[1] Joan M. Aldous and Robin J. Wilson, Graphs and Applications: An Introductory Approach, Springer-Verlag, 2000.

[2] Eric Bryant et al, Poly^2 Paradigm: A Secure Network Service Architecture, 19th Annual Computer Security Applications Conference (ACSAC), December, 2003.

[3] Dan Ellis, Worm Anatomy and Model, Proceedings of ACM CCS WORM Workshop 2003, October, 2003.

[4] Joshua D. Guttman, Filtering Postures: Local Enforcement for Global Policies, Proceedings, 1997 IEEE Symposium on Security and Privacy pages 120-129. IEEE Computer Society Press. May 1997.

[5] Hyang-Ah Kim, Brad Karp, Autograph: Toward Automated, DistributedWorm Signature Detection, USENIX Security Symposium, to appear, 2004.

[6] Internet Engineering Task Force RFC 1700. http://www.ietf.org/rfc/rfc1700.txt.

[7] Jose Nazario, Defense and Detection Strategies against Internet Worms, Artech House, 2004.

[8] Sumeet Singh, Cristian Estan, George Varghese, Stefan Savage, The EarlyBird System for Real-time Detection of Unknown Worms, to be presented at the Sixth Symposium on Operating System Design and Implementation (OSDI), 2004.

[9] http://www.snort.org/.

[10] Stuart Staniford et al., The Design of GrIDS: A Graph-Based Intrusion Detection System. UCD Technical Report CSE-99-2, January, 1999.

[11] Stuart Staniford, Nicholas Weaver, Vern Paxson, How to 0wn the Internet in your Spare Time, USENIX Security Symposium, 2002.

[12] Stuart Staniford. Containment of Scanning Worms in Enterprise Networks. Journal of Computer Security, to appear, 2004.

[13] Richard W. Stevens, TCP/IP Illustrated, vol. 1, Addison Wesley Longman, Inc., 1994.

[14] Nicholas Weaver, Vern Paxson, Stuart Staniford, Robert Cunningham. A Taxonomy of Computer Worms. Proceedings of ACM CCS WORM Workshop 2003, October 2003.

[15] Nicholas Weaver, Dan Ellis, Vern Paxson, Stuart Staniford, Worms vs. Perimeters: The Case for HardLANs, To appear, Hot Interconnects 2004, Stanford University, August, 2004.

[16] Nicholas Weaver, Stuart Staniford, Vern Paxson, Very Fast Containment of Scanning Worms, USENIX Security Symposium, 2004.

[17] Matthew Williamson. Throttling Viruses: Restricting Propagation to Defeat Mobile Malicious Code. In ACSAC, 2002.