

An Architecture for Generating Semantic-Aware Signatures

Vinod Yegneswaran, Jonathon T. Giffin, Paul Barford, Somesh Jha

Abstract

Identifying new intrusion exploits and developing effective detection signatures for them is essential for protecting computer networks. We present *Nemean*, a system for automatic generation of intrusion signatures from honeynet packet traces. Our architecture is distinguished by its emphasis on a *modular design framework* that encourages independent development and modification of system components and *protocol semantic awareness* which allows construction of signatures that greatly reduce false alarms. The building blocks of our architecture include transport and service normalization, intrusion profile clustering and automata learning that generates connection and session aware signatures. We evaluate our architecture through a prototype implementation that demonstrates the potential of semantic-aware, resilient signatures. For example, signatures generated by Nemean for NetBIOS exploits had a 0% false-positive rate and a 0.04% false-negative rate.

1 Introduction

Computer network security is a multidimensional activity that continues to grow in importance. The prevalence of attacks in the Internet and the ability of self-propagating worms to infect millions of Internet hosts has been well documented [33, 37]. Developing techniques and tools that enable more precise and more rapid detection of such attacks presents significant challenges to both the research and operational communities.

Network security architectures often include network intrusion detection systems (NIDS) that monitor packet traffic between networks and raise alarms when malicious activity is observed. NIDS that employ *misuse-detection* compare traffic against a hand-built database of signatures or patterns that identify previously documented attack profiles [3, 18]. While the effectiveness of a misuse-detector is linked tightly to the quality of its signature database, competing requirements make generating and maintaining NIDS signatures difficult. On one hand, signatures should be *specific*: they should only identify the characteristics of specific attack profiles. The lack of specificity leads to false alarms—one of the major problems for NIDS today. For example, Sommer and Paxson argue that including context, such as the victim’s response, in NIDS signatures may reduce false alarm rates [30]. On the other hand, signatures should be *general* so that they match variants of specific attack profiles. For example, a signature that does not account for transport or application level semantics can lead to false alarms [7, 22, 36]. Thus, a balance between specificity and generality is an important objective for signatures.

We present the design and implementation of an architecture called *Nemean*¹ for automatic generation of signatures for misuse-detection. Nemean aims to create signatures that result in lower false alarm rates by balancing specificity and generality. We achieve this balance by including *semantic awareness*, or the ability to understand session-layer and application-layer protocol semantics. Examples of session layer protocols include NetBIOS and RPC, and application layer protocols include SMB, TELNET, NCP and HTTP. Increasingly, pre-processors for these protocols have become integral parts of NIDS. We argue that these capabilities are essential for automatic signature generation systems for the following reasons:

1. Semantic awareness enables signatures to be generated for attacks in which the exploit is a small part of the entire payload.
2. Semantic awareness enables signatures to be generated for multi-step attacks in which the exploit does not occur until the last step.
3. Semantic awareness allows weights to be assigned to different portions of the payload (*e.g.*, timestamps, sequence numbers, or proxy-cache headers) based upon their significance.
4. Semantic awareness helps produce generalized signatures from a small number of input samples.
5. Semantic awareness results in signatures that are easy to understand and validate.

¹The first labor of the Greek hero Heracles was to rid the Nemean plain of a fierce creature known as the Nemean Lion. After slaying the beast, Heracles wore its pelt as impenetrable armor in his future labors.

Our architecture contains two components: a *data abstraction component* that normalizes packets from individual sessions and renders semantic context and a *signature generation component* that groups similar sessions and uses machine-learning techniques to generate signatures for each cluster. The signatures produced are suitable for deployment in a NIDS [3, 18, 35]. We address specificity by producing both connection-level and session-level signatures. We address generality by learning signatures from transport-normalized data and consideration of application-level semantics that enables variants of attacks to be detected. Therefore, we argue that Nemean generates *balanced* signatures.

The input to Nemean is a set of packet traces collected from a honeynet deployed on unused IP address space. Any data observed at a honeynet [8]² is anomalous, thus eliminating both the problem of privacy and the problem of separating malicious and normal traffic.³ We assume that the honeynet is subject to the same attack traffic as standard hosts and discuss the ramifications of this assumption in Section 8.

To evaluate Nemean’s architecture, we developed a prototype implementation of each component. This implementation enables automated generation of signatures from honeynet packet traces. We also developed a simple alert generation tool for off-line analysis which compares packet traces against signatures. While we demonstrate that our current implementation is extremely effective, the modular design of the architecture enables any of the individual components to be easily replaced. We expect that further developments will tune and expand individual components resulting in more timely, precise and effective signatures. From a broader perspective, we believe that our results demonstrate the importance of Nemean’s capability in a comprehensive security architecture. Section 3 describes the architecture and Sections 4 and 5 present our prototype implementation of Nemean.

We performed two evaluations of our prototype. First, we calculated detection and misdiagnosis counts using packet traces collected at two unused /19 address ranges (32K total IP addresses) from two distinct Class B networks allocated to our campus. We collected session-level data for exploits targeting ports 80 (HTTP), 139 and 445 (NetBIOS/SMB). Section 6 describes the data collection environment. We use this packet trace data as input to Nemean to produce a comprehensive signature set for the three target ports. In Section 7, we describe the major clusters and the signatures produced from this data set. Leave-out testing results indicate that our system generates accurate signatures for most common intrusions, including Code Red, Nimda, and other popular exploits. We detected 100% of the HTTP exploits and 99.96% of the NetBIOS exploits with 0 misdiagnoses. Next, we validated our signatures by testing for false alarms using packet traces of all HTTP traffic collected from our department’s border router. Nemean produced 0 false alarms for this data set. By comparison, Snort [3] generated over 80,000 false alarms on the same data set. These results suggest that even with a much smaller signature set, Nemean achieves detectability rates on par with Snort while identifying attacks with superior precision and far fewer false alarms.

2 Related Work

Sommer and Paxson [30] propose adding connection-level context to signatures to reduce false positives in misuse-detection. In [7], Handley *et al.* describe transport-level evasion techniques designed to elude a NIDS as well as normalization methods that disambiguate data before comparison against a signature. Similar work describes common HTTP evasion techniques and standard URL morphing attacks [22]. Vigna *et al.* [36] describe several mutations and demonstrate that two widely deployed misuse-detectors are susceptible to such mutations. The work of Handley *et al.* and Vigna *et al.* highlights the importance of incorporating semantics into the signature-generation process.

Honeypots are an excellent source of data for intrusion and attack analysis. Levin *et al.* describe how honeypots extract details of worm exploits that can be analyzed to generate detection signatures [14]. Their signatures must be generated manually.

Several automated signature generation systems have been proposed. Table 1 summarizes the differences between Nemean and the other signature-generation systems. One of the first systems proposed was Honeycomb developed by Kreibich and Crowcroft [12]. Like Nemean, Honeycomb generates signatures from traffic observed at a honeypot and is implemented as a Honeyd [20]⁴ plugin. At the heart of Honeycomb is the

²A honeynet is a network of high-interaction honeypots.

³A negligible amount of non-malicious traffic on our honeynet is caused by misconfigurations and is easily separated from the malicious traffic.

⁴Honeyd is a popular open-source low-interaction honeypot tool that simulates virtual machines over unused IP address space.

	Traffic source	Generates Contextual Signatures	Semantic Aware	Signature Generation Algorithm	Target Attack Class
Nemean	Honeypots	<i>Yes</i> (Generates connection- and session- level signatures)	<i>Yes</i>	(MSG) Clustering and automata learning	<i>General</i>
Autograph	DMZ	<i>No</i> (Generates byte-level signatures)	<i>No</i>	(COPP) partitioning content blocks	<i>Worm</i>
EarlyBird	DMZ	<i>No</i> (Generates byte-level signatures)	<i>No</i>	Measuring packet-content prevalence	<i>Worm</i>
Honeycomb	Honeypots	<i>No</i> (Generates byte-level signatures)	<i>No</i>	Pairwise LCS across connections	<i>General</i>

Figure 1: Comparison of Nemean to other signature-generation systems.

longest common substring (LCS) algorithm that looks for the longest shared byte sequences across pairs of connections. However, since Honeycomb does not consider protocol semantics, its pairwise LCS algorithm outputs a large number of signatures. It is also frequently distracted by long irrelevant byte sequences in packet payloads, thus reducing its capability for identifying attacks with small exploit strings, exemplified in protocols such as NetBIOS. We discuss this in greater detail in Section 7.3.

Kim and Karp [11] describe the Autograph system for automated generation of signatures to detect worms. Unlike Honeycomb and Nemean, Autograph’s input are packet traces from a DMZ that includes benign traffic. Content blocks that match “enough” suspicious flows are used as input to COPP, an algorithm based on Rabin fingerprints that searches for repeated byte sequences by partitioning the payload into content blocks. Like Honeycomb, Autograph does not consider protocol semantics. We argue that such approaches, while attractive in principle, seem viable for a rather limited spectrum of observed attacks and are prone to false positives. This also makes Autograph more susceptible to mutation attacks [7, 22, 36]. Finally, unlike byte-level signatures produced by Autograph, Nemean can produce both connection-level and session-level signatures.

Another system developed to generate signatures for worms, Earlybird [29], measures packet-content prevalence at a single monitoring point such as a network DMZ. By counting the number of distinct sources and destinations associated with strings that repeat often in the payload, Earlybird distinguishes benign repetitions from epidemic content. Like Autograph, Earlybird also produces byte-level signatures and is not aware of protocol semantics. Hence Earlybird has the same disadvantages compared to Nemean as Autograph.

Pouget and Dacier [19] analyze honeypot traffic to identify root causes of frequent processes observed in a honeypot environment. They first organize the observed traffic based on the port sequence. Then, the data is clustered using association-rules mining [1]. The resulting clusters are further refined using “phrase distance” (which is similar to the hierarchical edit distance metric described in Section B) between attack payloads. Pouget and Dacier’s technique is not semantically aware. Julisch [9] also clusters alarms for the purpose of discovering the root-cause of an alarm. After clustering the alarms, Julisch’s technique generates a *generalized alarm* for each cluster. Intuitively, generation of generalized alarms is similar to the automata-learning step of our algorithm. However, the goals and techniques used in our work are different than the ones used by Julisch.

Anomaly detection is an alternative approach for malicious traffic identification in a NIDS. Anomaly detectors construct a model of acceptable behavior and then flag any deviations from the model as suspicious. Anomaly-detection techniques for detecting port scans have been explored in [10, 32]. Balancing specificity and generality has proven extraordinarily difficult in anomaly detection, and such systems often produce high rates of false alarms. This paper focuses on misuse-detection, and we will not investigate anomaly-detecting techniques further.

3 Nemean Architecture

As shown in Figure 2, Nemean’s architecture is divided into two components: the data abstraction component and the signature generation component. The input to Nemean is a packet trace collected from a honeynet. Even when deployed on a small address space (*e.g.*, a /24 containing 256 IP addresses), a honeynet can provide a large volume of data without significant privacy or false positives concerns.

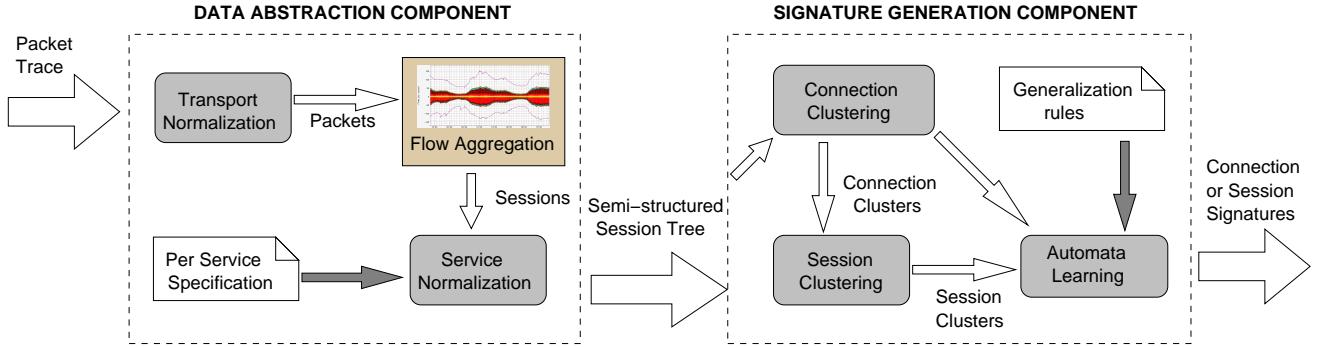


Figure 2: Components and data flow description of the Nemean architecture

3.1 Data Abstraction Component

The Data Abstraction Component (DAC) aggregates and transforms the packet trace into a well-defined data structure suitable for clustering by a generic clustering module without specific knowledge of the transport protocol or application-level semantics. We call these aggregation units *semi-structured session trees (SSTs)*. The components of the DAC can then be thought of in terms of the data flow through the module as shown in Figure 2. While we built our own DAC module, in principle it could be implemented as an extension to a standard NIDS, such as a Bro policy script [18].

Transport normalization disambiguates obfuscations at the network and transport layers of the protocol stack. Our DAC reads packet traces through the `libpcap` library. This can either be run online or offline on `tcpdump` traces. This step considers transport-specific obfuscations like fragmentation reassembly, duplicate suppression, and checksums. We describe these in greater detail in Section 4.

The *aggregation* step groups packet data between two hosts into sessions. The normalized packet data is first composed and stored as *flows*. Periodically, the DAC expires flows and converts them into *connections*. A flow might be expired for two reasons: a new connection is initiated between the same pair of hosts and ports or the flow has been inactive for a time period greater than a user defined timeout (1 hour in our experimental setup). Flows are composed of packets, but connections are composed of request-response elements. Each connection is stored as part of a *session*. A session is a sequence of connections between the same host pairs.

Service-specific information in sessions must be normalized before clustering for two reasons. First, classification of sessions becomes more robust and clustering algorithms can be independent of the type of service. Second, the space of ambiguities is too large to produce a signature for every possible encoding of attacks. By decoding service-specific information into a canonical form, normalization enables generation of a more compact signature set. A detection system must then first decode attack payloads before signature matching. This strategy is consistent with that employed by popular NIDS [3]. We describe the particular normalizations performed in greater detail in Section 4.

The DAC finally transforms the normalized sessions into XML-encoded SSTs suitable for input to the clustering module. This step also assigns weights to the elements of the SST to highlight the most important attributes, like the URL in an HTTP request, and deemphasize the less important attributes, such as encrypted fields and proxy-cache headers in HTTP packets. The clustering module may use the weights to construct more accurate session classifications.

3.2 Signature Generation Component

The clustering module groups sessions and connections with similar attack profiles according to a similarity metric. We assume that sessions grouped together will correspond to a single attack type or variants of a well-known attack while disparate clusters represent distinct attacks or attack variants that differ significantly from some original attack. Effective clustering requires two properties of the attack data. First, data that correspond to an attack and its variants should be measurably similar. A clustering algorithm can then classify such data as likely belonging to the same attack. Second, data corresponding to different attacks must be measurably dissimilar so that a clustering algorithm can separate such data. We believe that the two required properties are unlikely to hold for data sets that include significant quantities of non-malicious or normal traffic. Properties

of normal traffic vary so greatly as to make effective clustering difficult without additional discrimination metrics. Conversely, malicious data contains identifiable structure even in the presence of obfuscation and limited polymorphism. Nemean’s use of honeynet data enables a reasonable number of meaningful clusters to be produced. While each cluster ideally contains the set of sessions or connections for some attack, we also presume that this data will contain minor obfuscations, particularly in the sequential structure of the data, that correspond to an attacker’s attempts to evade detection. These variations provide the basis for our signature generation step.

The automata learning module constructs an attack signature from a cluster of sessions. A generator is implemented for a target intrusion detection system and produces signatures suitable for use in that system. This component has the ability to generate highly expressive signatures for advanced systems, such as regular expression signatures with session-level context that are suitable for Bro [18,30]. Clusters that contain many non-uniform sessions are of particular interest. These differences may indicate either the use of obfuscation transformations to modify an attack or a change made to an existing attack to produce a new variant. Our signature generation component generalizes these transformations to produce a signature that is resilient to evasion attempts. Generalizations enable signatures to match malicious sequences that were not observed in the training set.

4 Data Abstraction Component Implementation

We have implemented prototypes of each Nemean component. While the Nemean design provides flexibility to handle any protocol, we focus our discussion on two specific protocol implementations, HTTP (port 80) and NetBIOS/SMB (ports 139 and 445), since these two services exhibit great diversity in the number and types of exploits.

4.1 Transport-Level Normalization

Transport-level normalization resolves ambiguities introduced at the network (IP) and transport (TCP) layers of the protocol stack. We check message integrity, reorder packets as needed, and discard invalid or duplicate packets. The importance of transport layer normalizers has been addressed in the literature [7,21]. Building a normalizer that *perfectly* resolves all ambiguities is a complicated endeavor, especially since many ambiguities are operating system dependent. We can constrain the set of normalization functions for two reasons. First, we only consider traffic sent to honeynets, so we have perfect knowledge of the host environment. This environment remains relatively constant. We do not need to worry about ambiguities introduced due to DHCP or network address translation (NAT). Second, Nemean’s current implementation analyzes network traces off-line which relaxes its state holding requirements and makes it less vulnerable to resource-consumption attacks.

Attacks that attempt to evade a NIDS by introducing ambiguities to IP packets are well known. Examples of such attacks include simple *insertion attacks* that would be dropped by real systems but are evaluated by NIDS, and *evasion attacks* that are the reverse [21]. Since Nemean obtains traffic promiscuously via a packet sniffer (just like real a NIDS), these ambiguities must be resolved. We focus on three common techniques used by attackers to elude detection.

First, an invalid field in a protocol header may cause a NIDS to handle the packet differently than the destination machine. Handling invalid protocol fields in IP packets involves two steps: recognizing the presence of the invalid fields and understanding how a particular operating system would handle them. Our implementation performs some of these validations. For example, we drop packets with an invalid IP checksum or length field.

Second, an attacker can use IP fragmentation to present different data to the NIDS than to the destination. Fragmentation introduces two problems: correctly reordering shuffled packets and resolving overlapping segments. Various operating systems address these problems in different ways. We adopt the *always-favor-old-data* method used by Microsoft Windows. A live deployment must either periodically perform *active-mapping* [28] or match rules with passive operating system fingerprinting. The same logic applies for fragmented or overlapping TCP segments.

Third, incorrect understanding of the TCP Control Block (TCB) tear-down timer can cause a NIDS to improperly maintain state. If it closes a connection too early it will lose state. Likewise, retaining connections too long can prevent detection of legitimate later connections. Our implementation maintains connection state for an hour after session has been closed. However, sessions that have been closed or reset are replaced earlier

1. Build the multiset C of all normalized connections.
2. Cluster C into exclusive partitions $\mathcal{CL} = \{\xi_i\}$.
3. Produce a connection-level signature ϕ_ξ for each cluster by generalizing cluster data.
4. Build the multiset S' of all sessions. Each session $s' \in S'$ is a sequence of identifiers denoting the connection clusters that contain each connection in the session.
5. Cluster S' into partitions $\Psi = \{\psi_i\}$.
6. Produce a session-level signature L_ψ for each cluster, generalizing the observed connection orderings.
7. Produce a NIDS signature. The signature is a hierarchical automaton where each transition in the session-level signature requires that the connection-level signature for the identified connection cluster accepts.

Figure 3: Multi-level Signature Generalization (MSG) algorithm. Section 5 provides more complete details.

if a new connection setup is observed between the same host/port pairs.

4.2 Service-Level Normalization

We provide a brief discussion of the implementation of service normalizers for two popular protocols HTTP and NetBIOS/SMB.

Ambiguities in HTTP sessions are primarily introduced due to invalid protocol parsing or invalid decoding of protocol fields. In particular, improper URL decoding is a point of vulnerability in many intrusion detection systems. Modern web servers allow substitution of encoded characters for ASCII characters in the URL and are often exploited as means for evasion of common NIDS signatures. Our DAC correctly decodes several observed encodings such as hex encoding and its variants, UTF-8 encoding, bare-byte encoding, and Microsoft Unicode encoding. Regardless of its encoding, the DAC presents a canonical URL in ASCII format to the clustering module. We provide details on some commonly observed encoding schemes in Appendix A.1. Currently, our implementation does not handle all obvious HTTP obfuscations. For example, we do not process pipelined HTTP/1.1 requests. Such requests need to be broken into multiple connections for analysis. We plan to incorporate this functionality into our system in the future.

NetBIOS is a session-layer service that enables machines to exchange messages using names rather than IP addresses and port numbers. SMB (Server Message Block) is a transport-independent protocol that provides file and directory services. Microsoft Windows machines use NetBIOS to exchange SMB file requests. NetBIOS/SMB signature evasion techniques have not been well studied, possibly due to the lack of good NIDS rules for their detection. A full treatment of possible NetBIOS/SMB ambiguities is outside the scope of this paper. We describe certain ambiguities handled by our normalizer in Appendix A.2.

5 Multi-level Signature Generalization

We designed the *Multi-level Signature Generalization (MSG)* algorithm to automatically produce signatures for normalized session data. The signatures must balance specificity to the exploits observed in the data with generality, the ability to detect attack variants not previously observed. We use machine-learning algorithms, including clustering and finite state machine generalization, to produce signatures that are well-balanced.

Due to the hierarchical nature of the session data, we construct signatures for connections and sessions separately. First, we cluster all connections irrespective of the sessions that contain them and generalize each cluster to produce a signature for each connection cluster. Second, we cluster sessions based upon their constituent connections and then generalize the clusters. Finally, we combine the session and connection signatures to produce a hierarchical automaton signature, where each connection in a session signature must match the corresponding connection signature. Figure 3 presents a high-level overview of the algorithm.

Steps 1 and 2: Generating connection clusters. Let S be the multiset of normalized sessions produced by the data abstraction component. Denote each session $s \in S$ as an ordered list of connections: $s = c_1.c_2.\dots.c_{n_s}$. Let $Conn(s) = \{c_i\}_{i=1\dots n_s}$ be the multiset of connections in s and $C = \bigsqcup_{s \in S} Conn(s)$ be the multiset of all connections in the normalized data. Note that \bigsqcup denotes multiset union. Let $\mathcal{CL} = \{\xi_i\}_{i=1\dots m}$ be an exclusive clustering of C into m clusters ξ_i . Section 5.1 presents the implementation of the clustering algorithm. Clustering inserts every element into a partition, so $\bigsqcup_{i=1}^m \xi_i = C$. Exclusive clustering requires that no partitions overlap, so $\xi_i \cap \xi_j = \emptyset$ for $i \neq j$. It immediately follows that there exists a well-defined

function $\Gamma : C \rightarrow \mathcal{CL}$ defined as $\Gamma(c) = \xi$ if $c \in \xi$ that returns the cluster containing c .

Step 3: Building connection-level signatures. Learning algorithms generalize the data in each cluster to produce signatures that match previously unseen connections. Let Σ be the alphabet of network events comprising connection data. A learning algorithm is a function $Learn : \mathcal{P}(\Sigma^*) \rightarrow \mathcal{P}(\Sigma^*)$ that takes a set of strings $\widehat{\phi}_\xi = \bigcup_{c \in \xi} c$ and returns a regular language $\phi_\xi \supseteq \widehat{\phi}_\xi$. Section 5.2 presents the generalization algorithms used in our work. We recognize ϕ_ξ with a regular automaton that is the connection-level signature for cluster ξ .

Steps 4 and 5: Generating session clusters. Rewrite the existing sessions to produce a new set S' .

$$S' = \biguplus_{s=c_1 \dots c_{n_s} \in S} [s' = \Gamma(c_1) \dots \Gamma(c_{n_s})]$$

From an implementation perspective, each $\Gamma(c_i)$ in a rewritten session is simply an integer index indicating which connection cluster contains the original connection. Intuitively, we allow any connection c_i comprising part of session s to be replaced with any connection $c'_i \in \Gamma(c_1)$ identified by clustering as similar. Let Ψ be a clustering of S' .

Steps 6 and 7: Building session-level signatures. As with connection-level generalization, construct a regular language L_ψ for each cluster $\psi \in \Psi$ that accepts the sessions in ψ and variants of those sessions. Again, we recognize the language with a finite automaton. The connection cluster identifiers $\Gamma(c)$ label transitions in the session-level automata. The resulting signature is thus hierarchical: traversing a transition in the session signature requires connection data matching the signature for the connection cluster.

5.1 Star Clustering Implementation

We cluster connections and sessions using the same algorithm. We implemented the on-line star clustering algorithm, which clusters documents based upon a similarity metric [2]. This algorithm has advantages over more commonly-known techniques, such as the k -means family of algorithms [15]. For example, star clustering is robust to data ordering. K -means, conversely, produces different clusters depending upon the order in which data is read. Moreover, we need not know *a priori* how many clusters are expected. Although it seems suitable, we make no claims that star is the optimal clustering algorithm for our purposes, and we expect to consider other algorithms in future work.

Star clustering builds a *star cover* over a partially-connected graph. Nodes in the graph each represent one or more items with semantically equivalent data. We arbitrarily choose one item at each node to be the *representative item*. A link exists between two nodes if the similarity between the corresponding representative items is above a designated threshold. A *star cluster* is a collection of nodes in the graph such that each node connects to the cluster center node with an edge. A star cover is a collection of star clusters covering the graph so that no two cluster centers have a connecting edge. In the original algorithm, a non-center node may have edges to multiple center nodes and thus appear in multiple clusters. We implemented a modified algorithm that inserts a node only into the cluster with which it has strongest similarity to produce an exclusive clustering.

Item similarity determines how edges are placed in the graph. We implemented two different similarity metrics to test sensitivity: *cosine similarity* [2] and *hierarchical edit distance* (Appendix B). The cosine similarity metric has lower computational complexity than hierarchical edit distance and was used for our experiments in Section 7.

Cosine similarity computes the angle between two vectors representing the two items under comparison. For each connection A , we build a vector D_A giving the distribution of bytes, request types, and response codes that appeared in the network data. For sessions, the vector contains the distribution of connection cluster identifiers. If θ is the angle between vectors D_A and D_B representing items A and B , then:

$$\cos \theta = \frac{D_A \cdot D_B}{\|D_A\| \|D_B\|}$$

where ‘ \cdot ’ represents inner product and $\|v\|$ is the vector norm. All vector values are non-negative, so $0 \leq \theta \leq \pi/2$ and $1 \geq \cos \theta \geq 0$. The similarity between items is the value $\cos \theta$, with $\cos \theta = 1$ indicating equality.

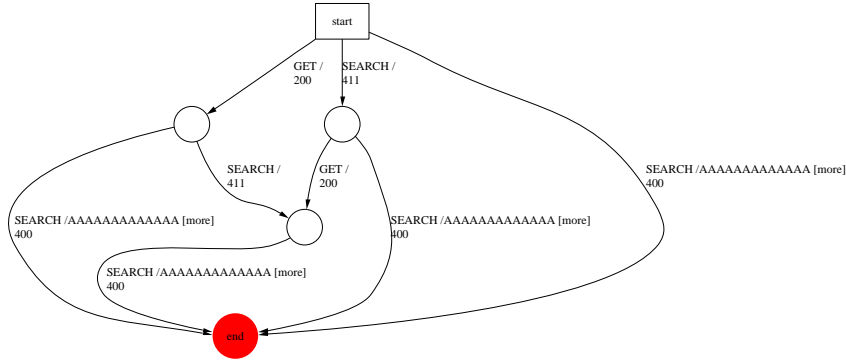


Figure 4: Welchia session level signature. For brevity, we label a single transition with both a request and a reply.

We initially believed hierarchical edit distance, though costly, would be the better similarity metric. It preserves connection ordering information within each session and differentiates between the various data fields within each connection. We believed these properties would produce better clusters than the cosine metric. Our experiments revealed that while both distance metrics work quite well, cosine is less sensitive to the threshold parameters used in partitioning clusters. Hence, we use cosine distance in this paper’s experiments and describe the hierarchical edit distance metric in Appendix B.

Using a similarity metric, we construct the partially-connected similarity graph. An edge connects a pair of nodes if the similarity of the representative sessions is above a threshold, here 0.8. We then build a star cover over the similarity graph. Each star cluster is a group of similar sessions that presumably are variants of the same exploit. The cluster set is then passed to the generalization module to produce the automaton signature.

5.2 Cluster Generalization and Signature Generation

Signature generation devises a NIDS signature from a cluster of similar connections or sessions. We generalize variations observed in a cluster’s data. Assuming effective clustering, these variations correspond to obfuscation attempts or differences among variants of the same attack. By generalizing the differences, we produce a resilient signature that accepts data not necessarily observed during the training period.

The signature is a finite state automaton. We first construct a probabilistic finite state automaton (PFSA) accepting exactly the event sequences contained in a cluster, with edge weights corresponding to the number of times an edge is traversed when accepting all cluster data exactly once. PFSA learning algorithms [24] then use stochastic measures to generalize the data variations observed in a cluster. In this work, we generalized HTTP connection-level signatures with the *sk-strings* method [24], an algorithm that merges states when they are probabilistically indistinguishable. Session-level clusters were generalized with *beam search* [17]. Our algorithm uses both *sk-strings* and *simulated beam annealing* [23] to generalize NetBIOS signatures. These generalizations add transitions into the state machine to accommodate such variations as data reordering and alteration of characters in an attack string. Likewise, repeated strings may be generalized to allow any number of repeats.

We implemented a second generalization that allows signatures to accept any behavior at points of high variability. *Subsequence creation* converts a signature defining a sequence of session data into a signature that is a subsequence of that data. This is a signature with “gaps” that accept arbitrary sequences of arbitrary symbols. We insert gaps whenever observing four or more patterns with a common prefix, common suffix, and one dissimilar data element. For example, let $A, B \in \Sigma^*$ and $v, w, x, y \in \Sigma$. If the signature accepts AvB , AwB , AxB , and AyB , then we replace those four sequences with the regular expression $A[.*]B$. Intuitively, we have identified a portion of the signature exhibiting large variation and allow it vary arbitrarily in our final signature. Nemean signatures are thus not limited only to data observed during training but can detect previously unknown attacks.

Figure 4 shows a session-level signature for Welchia, a worm that exploits a buffer overflow. Generalization produced a general signature that matches a wide class of Welchia scans without losing the essential

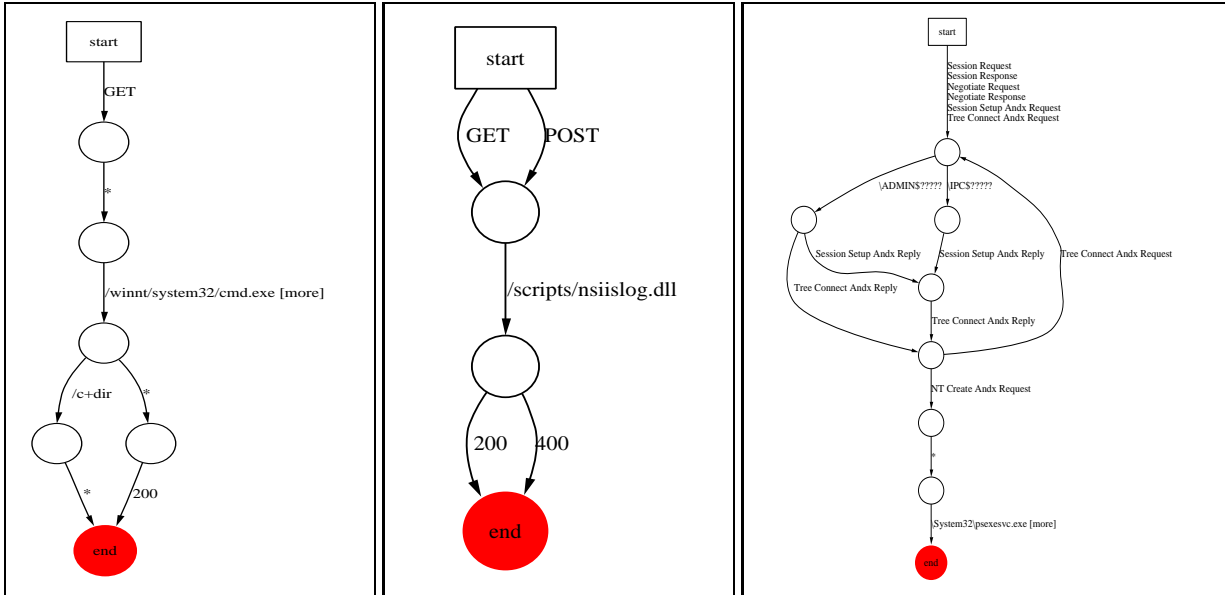


Figure 5: Nimda, Windows Media Player Exploit, and Deloder connection level signatures. The “*” transitions in the Nimda signature match any $\sigma \in \Sigma^*$.

buffer overflow information characteristic to the worm. Figure 5 shows connection-level signatures for Nimda, a Windows Media Player exploit, and the Deloder NetBIOS worm. The connection-level Nimda signature is an example of a signature for an exploit with high diversity. In particular, note that the subsequence creation generalization allows this signature to match a wide class of Nimda attacks. The Windows Media Player exploit is representative of an HTTP exploit where the size of the exploit URL is small. Previous signature generation techniques, such as Honeycomb, fail for small URLs. The Deloder signature demonstrates the capability of Nemean to generate signatures for exploits using more complex protocols like NetBIOS/SMB.

Our current implementation of the signature generation engine works best with some expert supervision. In our experience, different types of attacks require different types of signatures to be built. For example, we constructed connection-level signatures for Nimda because these attacks are independent of connection ordering and are identified by a string contained in a single connection. Conversely, Welchia employs a multi-stage attack that typically involves three ordered connections, so session-level signatures may be more appropriate for this worm (see Section 5.2). Clustering algorithms are also imperfect and can sometimes produce clusters that contain a small number of irrelevant sessions. Simple sanity checks can easily discount such irrelevant clusters, leading to a more robust signature set.

6 Data Collection

The data used for our evaluation comes from two sources: (i) honeypot packet traces collected from unused address space that we used to build signatures and evaluate the detection capability of Nemean and (ii) packet traces collected from our departmental border router that we used to test the resilience of our signatures to false positives.

6.1 Data Collection: Honeypot Traffic

Traffic from two unused /19 IP address blocks totaling 16K addresses from address ranges allocated to our university was routed to our honeynet monitoring environment. To normalize the traffic received by our infrastructure a simple source-filtering rule was employed: one destination IP address per source. Connections to additional destination IP addresses were dropped by the filter.

These filtered packets were subsequently routed to one of two systems based upon type-of-service. HTTP requests were forwarded to a fully patched Windows 2000 Server running on VMware. The NetBIOS/SMB traffic was routed to a virtual honeypot system similar to Honeyd. We routed NetBIOS/SMB packets to an active responder masquerading as an end host offering NetBIOS services rather than to the Windows 2000 Server for two reasons. First, the fully patched Windows 2000 Server often rejected or disconnected the

Table 1: Honeypot Data Summary

Port	Learning Data (2 days)				Test data (7 days)			
	Packets	Sources	Connections	Sessions	Packets	Sources	Connections	Sessions
80	278,218	10,859	25,587	12,545	100,291	12,925	12,903	5,172
139	192,192	1,434	3,415	1,657	6,764,876	539,334	1,662,571	24,747
445	1,763,276	14,974	35,307	19,763	6,661,276	383,358	1,171,309	37,165

Table 2: Productive Data Summary (HTTP: 8 hours, 16GB)

Data Flow	No. Clients	No. Servers	No. Sessions	No. Connections
Internal clients -> External servers	380	4,422	16,826	106,456
External clients -> Internal servers	18,634	24	28,491	87,545

session before we had enough information to classify the attack vector accurately. This could be due to invalid NetBIOS names or user/password combinations. Our active responder accepted all NetBIOS names and user/password combinations. Second, Windows 2000 servers limit the number of simultaneous network share accesses which also inhibit connection requests from succeeding.

We collected two sets of traces, a short term training set (2 days) and a longer testing set (7 days) to evaluate Nemean detection capability as summarized in Table 1.

6.2 Data Collection: Productive Traffic

Obtaining packet traces for live network traffic is a challenge due to privacy concerns. While network operators are amenable to sharing flow level summaries, anonymizing payloads remains an unsolvable problem and as such its hard to obtain packet traces with application payloads.

We were able to obtain access to such data from our department’s border. The network is a sparsely allocated, well managed /16 network with approximately 24 web servers and around 400 clients. We were able to passively monitor all outgoing and incoming HTTP packets on this network for an 8 hour period. Table 2 provides a summary of this dataset.

7 Evaluation

We tested the effectiveness of Nemean’s HTTP and NetBIOS signatures and examined the session clusters used to produce these signatures. Section 7.1 reveals the major classes of attacks in our recorded data and quantitatively measures the clusters produced by the clustering module. We perform an evaluation of the detection and false positive rates of Nemean’s signatures and compare our results with Snort’s HTTP capabilities. Finally, we provide a qualitative discussion of our experience with Honeycomb.

7.1 Evaluating the Clusters

- **HTTP Clusters:** Figure 6 provides an overview of the major HTTP clusters in our learning data set. WebDAV scans account for the majority of the attacks in our data set. WebDAV is a collection of HTTP extensions that allow users to collaboratively edit and manage documents in remote web servers. Popular WebDAV methods used in exploits include OPTIONS, SEARCH, and PROPFIND and are supported by Microsoft IIS web servers. Scans for exploits of WebDAV vulnerabilities are gaining in popularity and are also used by worms like Welchia. Nimda attacks provide great diversity in the number of attack variants and HTTP URL obfuscation techniques. These attacks exploit directory traversal vulnerabilities on IIS servers to access `cmd.exe` or `root.exe`. Figure 5 contains a connection-level Nemean Nimda signature. Appendix C provides more details of other observed exploits such as Frontpage, web crawlers and open-proxy.

- **NetBIOS Clusters:** Worms that are typically better known as email viruses dominate the NetBIOS clusters. Many of these viruses scan for open network shares and this behavior dominated the observed traffic. They can be broadly classified into three types:

1. *Hidden and open share exploits:* This includes viruses, including LovGate [4], NAVSVC, and Delder [13], use brute force password attacks to look for open folders and then deposit virus binaries in startup folders.

2. *MS-RPC query exploits*: Microsoft Windows provides the ability to remotely access MSRPC services through named pipes such as `epmapper` (RPC Endpoint Mapper)⁵, `srvsvc` (Windows Server Service), and `samr` (Windows Security Account Manager). Viruses often connect to the MSRPC services as guest users and then proceed to query the system for additional information that could lead to privileged user access. For example, connecting to the `samr` service allows the attacker to obtain an enumeration of domain users,

3. *MS-RPC service buffer overflow exploits*: The most well-known of these exploits are the `epmapper` service which allows access to the RPC-DCOM exploit [16] used by Blaster and the more recent `lsarpc` exploit used by Sasser [26]. We provide more details in Appendix D.

• **Cluster Quality**: We quantitatively evaluated the quality of clusters produced by the star clustering algorithm using two common metrics: *precision* and *recall*. Precision is the proportion of positive matches among all the elements in each cluster. Recall is the fraction of positive matches in the cluster among all possible positive matches in the data set. Intuitively, precision measures the relevance of each cluster while recall penalizes redundant clusters.

We first manually tagged each session with conjectures as shown in Figure 6. Conjectures identified sessions with known attack types and it is possible for a session to be marked with multiple conjectures. It is important to note that these conjectures were not used in clustering and served simply as evaluation aids to estimate the quality of our clusters.

The conjectures allow us to compute *weighted precision* (wp) and *weighted recall* (wr) for our clustering. As sessions can be tagged with multiple conjectures, we weight the measurements based upon the total number of conjectures at a given cluster of sessions. We compute the values wp and wr as follows: Let C be the set of all clusters, J be the set of all possible conjectures, and c_j be the set of elements in cluster c labeled with conjecture j . Then $|c_j|$ is the count of the number of elements in cluster c with conjecture j .

$$\begin{aligned} wp &= \sum_{c \in C} \left(\frac{|c|}{|C|} \sum_{j \in J} \left(\frac{|c_j|}{\sum_{k \in J} |c_k|} \frac{|c_j|}{|c|} \right) \right) & wr &= \sum_{c \in C} \left(\frac{|c|}{|C|} \sum_{j \in J} \left(\frac{|c_j|}{\sum_{k \in J} |c_k|} \frac{|c_j|}{|C_j|} \right) \right) \\ &= \frac{1}{|C|} \sum_{c \in C} \frac{\sum_{j \in J} |c_j|^2}{\sum_{k \in J} |c_k|} & &= \frac{1}{|C|} \sum_{c \in C} \left(\frac{|c|}{\sum_{k \in J} |c_k|} \sum_{j \in J} \frac{|c_j|^2}{|C_j|} \right) \end{aligned}$$

In the formulas above, $\sum_{k \in J} |c_k| \geq |c|$ and $\sum_{k \in J} |C_k| \geq |C|$ as sessions may have multiple conjectures. Figure 7 presents graphs indicating how precision and recall vary with the clustering similarity threshold. Recall that in the star clustering algorithm, an edge is added between two sessions in the graph of all sessions only if their similarity is above the threshold. Although less true for NetBIOS data, the similarity threshold does not have a significant impact on the quality of the resulting clustering. Clustering precision drops as the threshold nears 0 because the star graph becomes nearly fully connected and the algorithm cannot select suitable cluster centers. Recall that no cluster centers can share an edge, so many different clusters merge together at low threshold values. At the clustering threshold used in our experiments (0.8), precision scores were perfect or nearly perfect.

7.2 Signature Effectiveness

Intrusion detection signatures should satisfy two basic properties. First, they should have a high detection rate; *i.e.*, they should not miss real attacks. Second, they should generate few false alarms. Our results will show that Nemean has a 99.9% detection rate with 0 false alarms. Two additional metrics evaluate the quality of the alarms raised by an IDS. Precision empirically evaluates alarms by their specificity to the attack producing the alarm. Noise level counts the number of alarms per incident and penalizes redundant alarms. In these tests, we use Snort as a baseline for comparison simply because that is the most widely adopted intrusion detection system. We used the latest version of Snort available at the time, Snort-2.1.0 with the HTTP pre-processor enabled, and its complete associated ruleset. In some sense, Snort is the strawman because of its well-known susceptibility to false-positives. We use this because of our inability to compare with Honeycomb (see section 7.3) and because there is no source code publicly available for Earlybird or Autograph [11, 29].

⁵a Windows RPC Management Service

CLUSTER 1:	9175 Unique client IPs,	10515 Sessions			
	Identified as Options	: 10515 (100%)			
CLUSTER 2:	597 Unique client IPs,	735 Sessions			
	Identified as Nimda	: 735 (100%)			
	Identified as Code Blue	: 15 (2%)			
CLUSTER 4:	742 Unique client IPs,	808 Sessions			
	Identified as Welchia	: 808 (100%)			
	Identified as Search	: 794 (98%)			
CLUSTER 3:	201 Unique client IPs,	226 Sessions			
	Identified as Search	: 99 (44%)			
	Identified as Web Crawler	: 5 (2%)			
CLUSTER 5:	51 Unique client IPs,	52 Sessions			
	Identified as Nimda	: 52 (100%)			
CLUSTER 17:	47 Unique client IPs,	102 Sessions			
	Identified as Propfind	: 102 (100%)			
	Identified as Options	: 102 (100%)			
CLUSTER 8:	20 Unique client IPs,	20 Sessions			
	Identified as Nimda	: 20 (100%)			
CLUSTER 7:	11 Unique client IPs,	11 Sessions			
	Identified as Windows Media Exploit:	11 (100%)			
CLUSTER 6:	10 Unique client IPs,	10 Sessions			
	Identified as Search	: 10 (100%)			
CLUSTER 9:	8 Unique client IPs,	8 Sessions			
	Identified as Code Red Retina	: 8 (100%)			
	Identified as Search	: 5 (63%)			
CLUSTER 11:	6 Unique client IPs,	6 Sessions			
	Identified as Propfind	: 6 (100%)			
	Identified as Options	: 6 (100%)			
CLUSTER 19:	5 Unique client IPs,	5 Sessions			
	Identified as Propfind	: 5 (100%)			
	Identified as Options	: 5 (100%)			
CLUSTER 12:	3 Unique client IPs,	3 Sessions			
	Identified as Propfind	: 3 (100%)			
	Identified as Options	: 3 (100%)			
CLUSTER 10:	2 Unique client IPs,	2 Sessions			
	Identified as FrontPage Exploit	: 2 (100%)			
CLUSTER 16:	2 Unique client IPs,	3 Sessions			
	Identified as Kazaa	: 3 (100%)			
CLUSTER 13:	1 Unique client IPs,	2 Sessions			
	Identified as Web Crawler	: 1 (50%)			
CLUSTER 14:	1 Unique client IPs,	1 Session			
	Identified as Real Media Player	: 1 (100%)			
CLUSTER 15:	1 Unique client IPs,	1 Session			
	Identified as Propfind	: 1 (100%)			
	Identified as Options	: 1 (100%)			
CLUSTER 18:	1 Unique client IPs,	1 Session			
	Identified as Open Proxy	: 1 (100%)			

Figure 6: HTTP Port 80 cluster report

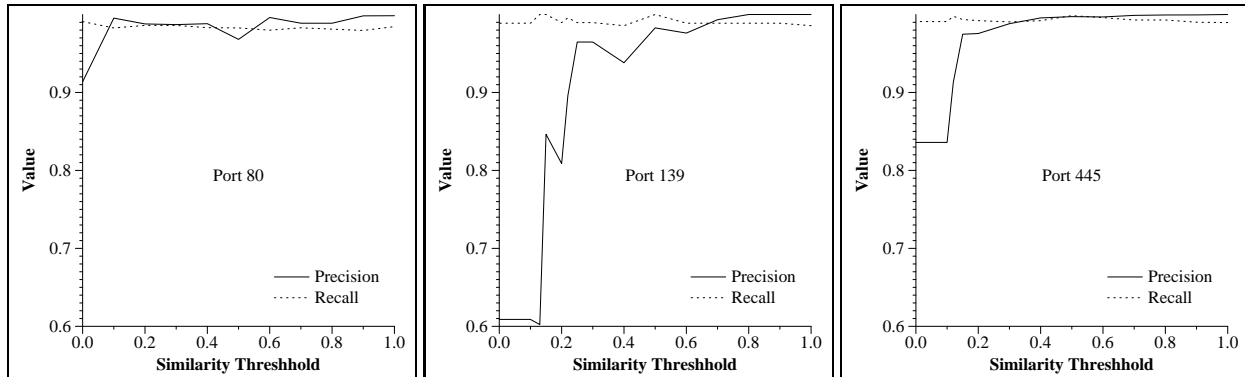


Figure 7: Effect of clustering similarity threshold upon weighted precision and weighted recall. Note that the y-axis begins at 0.6.

- 99.9% Detection Rate:** We evaluated the detection rate of Nemean signatures using *leave-out testing*, a common technique in machine learning. We used the honeynet data set described in Table 1 to automatically create connection-level and session-level signatures for the clusters identified in a training data set. We measured the detection rate of the signatures by running signature matching against data in a different trace collected from the same network (see Table 1).

Connection-level HTTP signatures detected 100.0% of the attacks present, and the somewhat more restrictive session-level signatures detected 97.7%. We did not evaluate session-level signatures for Nimda because the extreme variability of Nimda attacks made such signatures inappropriate. Table 3 shows the number of occurrences of the HTTP attacks and the number detected by Nemean signatures. For comparison, we provide detection counts for Snort running with an up-to-date signature set. Snort detected 99.7% of the attacks.

The detection rate of NetBIOS attacks is similarly very high: we detected 100.0% of the attacks present. Table 4 contains the detection rates for NetBIOS/SMB signatures. Snort provides only limited detection capability for NetBIOS attacks, so a comparison was infeasible. All signatures were connection-level because the defining characteristic of each attack is a string contained in a single connection. The structure of connections within a session is irrelevant for such attacks.

- Zero Misdiagnoses or False Alarms:** We qualify incorrect alerts on the honeynet data as misdiagnoses. Although not shown in Table 3, all Nemean HTTP signatures generated 0 misdiagnoses on the honeynet trace. Misdiagnosis counts for NetBIOS/SMB on the honeynet data were also 0, as shown in Table 4. We also measured false alarm counts of Nemean HTTP signatures against 16GB of packet-level traces collected from our department’s border router over an 8 hour time period. The traces contained both inbound and outbound HTTP traffic, most of which was legitimate. We evaluated both Nemean and Snort against the dataset. In all fairness, it must be noted that Snort has a larger signature set which makes it more prone to false positives. Nemean results are highly encouraging: 0 false alarms. Snort generated 88,000 alarms on this dataset, almost

Table 3: Session-level HTTP signature detection counts for Nemean signatures and Snort Signatures. We show only exploits occurring at least once in the training and test data.

Signature	Present	Nemean		
		Conn	Sess	Snort
Options	1172	1172	1160	1171
Nimda	496	496	N/A	495
Propfind	229	229	205	229
Welchia	90	90	90	90
Windows Media Player	89	89	89	89
Code Red Retina	4	4	4	0
Kazaa	2	2	2	2

all of which were false alarms. The Snort false alarms were produced by a collection of overly general signatures. Appendix D.1 contains a table showing the top seven Snort false alarm categories by volume. Our ability to evaluate for false negatives through manual inspection was encumbered by privacy concerns.

Our university filters NetBIOS traffic at the campus border, so we were unable to obtain NetBIOS data for this experiment.

• **Highly Specific Alarms:** Although the decision is ultimately subjective, we believe our signatures generate alerts that are empirically better than alerts produced by packet-level systems such as Snort. Typical Snort alerts, such as “Bare Byte Unicode Encoding” and “Non-RFC HTTP Delimiter”, are not highly revealing. They report the underlying symptom that triggered an alert but not the high-level reason that the symptom was present. This is particularly a problem for NetBIOS alerts because all popular worms and viruses fire virtually the same set of alerts. We call these *weak alerts* and describe them in Appendix D. Nemean, via connection-level or session-level signatures, has a larger perspective of a host’s intentions. As a result, we generate alerts specific to particular worms or known exploits.

• **Low Noise due to Session-Level Signatures:** Moreover, Nemean provides better control over the level of noise in its alarms. Packet-level detection systems such as Snort often raise alerts for each of multiple packets comprising an attack. A security administrator will see a flurry of alerts all corresponding to the same incident. For example, a Nimda attack containing an encoded URL will generate URL decoding alarms in Snort and alerts for WEB-IIS cmd.exe access. Sophisticated URL decoding attacks could later get misdiagnosed as Nimda alerts and be filtered by administrators. Our normalizer converts the URL to a canonical form to accurately detect Nimda attacks. Since Nemean aggregates information into connections or sessions and generates alerts only on the aggregated data, the number of alerts per incident is reduced.

In summation, we believe these results demonstrate the strength of Nemean. It achieves detection rates similar to Snort with dramatically fewer false alarms. The alerts produced by Nemean exhibit high quality, specifying the particular attack detected and keeping detection noise small.

7.3 Honeycomb Evaluation

One of the first efforts to address the problem of automatic signature generation from honeypot traces. We performed a comparison between Nemean and Honeycomb on identical traces as a means for further understanding the benefits of semantic awareness in automated signature generators. This evaluation was complicated by two issues: first, we transformed Honeycomb’s Honeyd plug-in implementation into a standalone application by feeding it the input traffic from a pcap loop. Second, since Honeycomb was developed as a proof-of-concept tool, it turned out to be incapable of processing large traces⁶. In our experience, Honeycomb’s processing time grows almost exponentially with each connection since it performs pairwise comparison across all connections, and running it on a relatively small trace of 3000 packets took over 15 hours on a high performance workstation. As a result, our evaluation is a qualitative comparison of Honeycomb signatures and its performance on a small trace with 126 HTTP connections.

Honeycomb produced 202 signatures from the input trace. While there were several perfectly functional

Table 4: Detection and misdiagnosis counts for connection-level Nemean NetBIOS signatures. This data includes both port 139 and port 445 traffic.

Signature	Present	Detected	Misdiagnoses
Srvsvc	19934	19930	0
Samr	8743	8741	0
Epmapper	1263	1258	0
NvcplDmn	62	61	0
Deloder	30	30	0
LoveGate	1	0	0

⁶An observation which was confirmed through personal communication with the author.

Table 5: Example signatures produced by Honeycomb on an HTTP trace with 126 connections

Exploit	Honeycomb Signature	Deficiency
Nimda	<ol style="list-style-type: none"> 1. /MSADC/root.exe?/c+dir HTTP/1.0 0D 0A Host: www 0D 0A Connection: close 0D 0A 0D 2. /root.exe?/c+dir HTTP/1.0 0D 0A Host: www 0D 0A Connection: close 0D 0A 0D 	Redundant ^a
WebDAV	<ol style="list-style-type: none"> 1. SEARCH / HTTP/1.1 0D 0A Host: 128.1 	Restrictive ^b
N/A	<ol style="list-style-type: none"> 1. 0D 0A Connection: Keep-Alive 0D 0A 0D 2. HTTP /1 	Benign

^aSignature 2 is a more general version of signature 1 which is redundant.

^bThe Host field should be ignored. As presented would lead to missed attacks from sources with prefixes other than 128.1

signatures, there were also a surprisingly large number of benign strings that were identified by the LCS algorithm. Some of these were small strings such as “GET” or “HTTP” that are clearly impractical and just happened to be the longest common substring between unrelated sessions. Communication with the Honeycomb author revealed these were part of normal operation and the typical way to suppress these are to whitelist signatures smaller than a certain length. There were also much longer strings in the signature set such as proxy-headers that also do not represent real attack signatures. It seems that the only way to avoid these kinds of problems is through manual grooming of signatures by an expert with protocol knowledge.

The summary of the comparison of signatures produced by Honeycomb versus those produced by Nemean is as follows:

1. Honeycomb produces a large number of signatures that lack specificity due to pairwise connection comparison. Nemean’s MSG algorithm generalizes from a cluster that includes several connections resulting in a smaller, balanced signature set.
2. Pairwise LCS employed by Honeycomb often leads to redundant (non-identical) signatures which would multiple alarms for the same attack. Again, Nemean’s MSG algorithm generalizes from clusters and its semantic-awareness makes it far less prone to redundant signature production.
3. Honeycomb signatures are often too restrictive. As a result, we require several restrictive signatures to capture all instances of a particular attack and this could lead to false negatives. Nemean’s generation of balanced signatures make them less susceptible to false negatives.
4. Honeycomb’s lack of semantic awareness leads to signatures consisting of benign substrings. These lead to false positives and is also the reason why Honeycomb is unable to produce precise signatures for protocols such as NetBIOS, MS-SQL and HTTP attacks like Nimda where the exploit content is a small portion of the entire attack string. Nemean’s semantic awareness addresses the issue of benign substrings.

We present examples of signatures that we obtained from Honeycomb that demonstrate these weaknesses in Table 5.

8 Discussion

A potential vulnerability of Nemean is its use of honeynets as a data source. If attackers become aware of this, they could either attempt to evade the monitor or to pollute it with irrelevant traffic resulting in many unnecessary signatures. Evasion can be complicated by periodic rotation of the monitored address space. Intentional pollution is a problem for any automated signature generation method and we intend to address it in future work.

Three issues may arise when deploying Nemean on a live network. First, live networks have real traffic, so we cannot assume that all observed sessions are malicious. To produce signatures from live traffic traces containing mixed malicious and normal traffic, we must first separate the normal traffic from the malicious. Flow-level anomaly detection or packet prevalence techniques [29] could help to identify anomalous flows in the complete traffic traces. Simple techniques that flag sources that horizontally sweep the address space, vertically scan several ports on a machine, and count the number of rejected connection attempts could also be used.

```

alert tcp any any -> 10.0.0.0/8
(msg: "(msg:WEB-IIS nsiislog.dll access";
 flow:to_server,established;
 uricontent:"/scripts/nsiislog.dll")
 nocase; reference:...)
```

Figure 8: Snort rule for Windows Media Player Exploit

```

signature nsiislog {
  ip-proto == tcp
  dst-port == 80
  http /*./scripts/nsiislog.dll
  requires-signature-opposite ! http_200_ok
  tcp-state-established
}

signature http_200_ok {
  ip-proto == tcp
  src-port == 80
  payload /*.HTTP\1\.. 200/
  event ``HTTP 200 OK``
  tcp-state-established
}
```

Figure 9: Bro Request/Reply Signature for Windows Media Player Exploit

Second, Nemean must generate meaningful signatures for Snort, Bro, or other NIDS. Snort utilizes an HTTP preprocessor to detect HTTP attacks and does not provide support for regular expressions.

Figure 8 shows the transformed Snort signature generated for the Windows Media Player exploit shown in Figure 5. Snort does not have the ability to associate both a request and a response in a signature, so our Snort signature ignores the response. Converting Nemean signatures to Bro signatures (see Figure 9) is straightforward since Bro allows for creation of policy scripts that support the use of regular expressions.

Third, while it is not the focus of this paper, Nemean may be run online. This makes Nemean attractive as a means to defend against new worms that propagate rapidly. The data abstraction component's modules work without any changes on live traces. The star clustering algorithm is also designed to perform incremental clustering and work in an online fashion. Anomaly detection techniques could be employed in parallel with Nemean to flag compelling clusters for worm outbreaks. Automatically generated Nemean signatures for these clusters could then be rapidly propagated to NIDS to defend against emergent worms. The resilience of Nemean signatures to false positive makes such a deployment practical.

9 Conclusions

We have described the design and implementation of Nemean, a system for automated generation of balanced NIDS signatures. One of the primary objectives of this system is to reduce false alarm rates by creating signatures that are semantically aware. Nemean's architecture is comprised of two major components: the data abstraction component and the signature generation component. This modular design supports and encourages independent enhancement of each piece of the architecture. Nemean uses packet traces collected at honeynets as input since they provide an unfettered view of a wide range of attack traffic.

We evaluated a prototype implementation of Nemean using data collected at two unused /19 subnets. We collected packet traces for two services for which we developed service normalizers (HTTP and NetBIOS/SMB). Running Nemean over this data resulted in clusters for a wide variety of worms and other exploits. Our evaluation suggests that simple similarity metrics like the cosine metric can provide clusters with a high degree of precision. We demonstrated the signature generation capability of our system and discussed optimizations used our automata learning module such as structure abstraction and subsequence creation. We showed that Nemean generated accurate signatures with extremely low false alarm rates for a wide range of attack types, including buffer overflows (Welchia), attacks with large diversity (Nimda), and attacks for complicated protocols like NetBIOS/SMB.

In future work, we intend to hone the on-line capabilities of Nemean and to assess its performance over longer periods of time in live deployments. We will also continue to evaluate methods for clustering and learning with the objective of fine tuning the resulting signature sets.

References

- [1] R. Agrawal, T. Imielinski, and A. Swami. Mining association rules between sets of items in large databases. In *Proceedings of the 1993 ACM SIGMOD International Conference on Management of Data*, 1993.

- [2] Javed Aslam, Katya Pelekhov, and Daniela Rus. A practical clustering algorithm for static and dynamic information organization. In *ACM-SIAM Symposium on Discrete Algorithms (SODA)*, Baltimore, Maryland, January 1999.
- [3] Brian Caswell and Marty Roesch. The SNORT network intrusion detection system. <http://www.snort.org>, April 2004.
- [4] Tony Conneff. W32.HLLW.lovgate removal tool. <http://securityresponse.symantec.com/avcenter/venc/data/w32.hllw.lovgate.removal.tool.html>, April 2004.
- [5] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to Algorithms*. MIT Press, 2nd edition, 2001.
- [6] Timothy Devans. SMB command codes. <http://timothydevans.me.uk/nbf2cifs/smb-smbcommandcode.html>, April 2004.
- [7] Mark Handley, Vern Paxson, and Christian Kreibich. Network intrusion detection: Evasion, traffic normalization and end-to-end protocol semantics. In *10th USENIX Security Symposium*, Washington, DC, August 2001.
- [8] The HoneyNet project. <http://project.honeynet.org>, April 2004.
- [9] K. Julisch. Clustering intrusion detection alarms to support root cause analysis. *ACM Transactions on Information and System Security (TISSEC)*, 6(4):443–471, November 2003.
- [10] J. Jung, V. Paxson, A. W. Berger, and H. Balakrishnan. Fast portscan detection using sequential hypothesis testing. In *IEEE Symposium on Security and Privacy*, pages 211–225, 2004.
- [11] Hyang-Ah Kim and Brad Karp. Autograph: Toward automated, distributed worm signature detection. In *13th USENIX Security Symposium*, San Diego, California, August 2004.
- [12] Christian Kreibich and John Crowcroft. Honeycomb—creating intrusion detection signatures using honeypots. In *2nd Workshop on Hot Topics in Networks (Hotnets-II)*, Cambridge, Massachusetts, November 2003.
- [13] Kyle Lai. Deloder worm/trojan analysis. http://www.klcconsulting.net/deloder_worm.htm, April 2004.
- [14] John Levine, Richard LaBella, Henry Owen, Didier Contis, and Brian Culver. The use of honeynets to detect exploited systems across large enterprise networks. In *2003 IEEE Workshop on Information Assurance*, West Point, New York, June 2003.
- [15] Stuart P. Lloyd. Least squares quantization in PCM. *IEEE Transactions on Information Theory*, IT-2:129–137, 1982.
- [16] Microsoft security bulletin MS03-007. <http://www.microsoft.com/technet/security/bulletin/MS03-007.asp>, April 2004.
- [17] Jon Patrick, Anand Raman, and P. Andreae. *A Beam Search Algorithm for PFSA Inference*, pages 121–129. Springer-Verlag London Ltd, 1 edition, 1998.
- [18] Vern Paxson. BRO: A system for detecting network intruders in real time. In *7th USENIX Security Symposium*, San Antonio, Texas, January 1998.
- [19] F. Pouget and M. Dacier. Honeypot-based forensics. In *Proceedings Of AusCERT Asia Pacific Information technology Security Conference 2004(AusCERT2004)*, Brisbane, Australia, May 2004.
- [20] Neils Provos. Honeyd: Network rhapsody for you. <http://www.citi.umich.edu/u/provos/honeyd>, April 2004.
- [21] Thomas Ptacek and Timothy Newsham. Insertion, evasion and denial of service: Eluding network intrusion detection. Technical report, Secure Networks, January 1998.
- [22] Rain Forest Puppy. A look at Whisker’s anti-IDS tactics. <http://www.wiretrip.net/rfp/txt/whiskerids.html>, April 2004.
- [23] Anand Raman and Jon Patrick. Beam search and simulated beam annealing. Technical Report 2/97, Department of Information Systems, Massey University, Palmerston North, New Zealand, 1997.
- [24] Anand V. Raman and Jon D. Patrick. The sk-strings method for inferring PFSA. In *14th International Conference on Machine Learning (ICML97)*, Nashville, Tennessee, July 1997.
- [25] Daniel J. Roelker. HTTP IDS evasions revisited. In *DEFCON 11*, Las Vegas, Nevada, August 2003.
- [26] W32 Sasser.Worm. <http://securityresponse.symantec.com/avcenter/venc/data/w32.sasser.worm.html>.
- [27] Security Focus. Microsoft IIS 5.0 “translate: f” source disclosure vulnerability. <http://www.securityfocus.com/bid/1578/discussion/>, April 2004.
- [28] Umesh Shankar and Vern Paxson. Active mapping: Resisting NIDS evasion without altering traffic. In *IEEE Symposium on Security and Privacy*, Oakland, California, May 2003.

- [29] Sumeet Singh, Christian Estan, George Varghese, and Stefan Savage. The Earlybird system for real-time detection of unknown worms. Technical Report CS2003-0761, University of California, San Diego, August 2003.
- [30] Robin Sommer and Vern Paxson. Enhancing byte-level network intrusion detection signatures with context. In *10th ACM Conference on Computer and Communication Security (CCS)*, Washington, DC, October 2003.
- [31] Sophos. W32.agobot-bt. <http://www.sophos.com/virusinfo/analyses/w32agobotbt.html>, April 2004.
- [32] S. Staniford, J. A. Hoagland, and J. M. McAlerney. Practical automated detection of stealthy portscans. *Journal of Computer Security*, 10(1/2):105–136, 2002.
- [33] Stuart Staniford, Vern Paxson, and Nicholas Weaver. How to Own the internet in your spare time. In *11th USENIX Security Symposium*, August 2002.
- [34] Symantec. W32.welchia.worm. <http://securityresponse.symantec.com/avcenter/venc/data/w32.welchia.worm.html>, April 2003.
- [35] G. Vigna and R. Kemmerer. NetSTAT: A network-based intrusion detection system. *Journal of Computer Security*, 7(1):37–71, 1999.
- [36] G. Vigna, W. Robertson, and D. Balzarotti. Testing network-based intrusion detection signatures using mutant exploits. In *Proceedings of the ACM Conference on Computer and Communication Security (ACM CCS)*, Washington, DC, October 2004.
- [37] Vinod Yegneswaran, Paul Barford, and Johannes Ullrich. Internet intrusions: Global characteristics and prevalence. In *ACM SIGMETRICS*, San Diego, California, June 2003.

A Protocol Normalization

This section provides more specifics on the HTTP and NetBIOS normalizations that were implemented in Nemean.

A.1 HTTP URL Encoding

Hex encoding refers to substitution of hexadecimal characters for ASCII characters in the URL. For example, the hexadecimal value of ASCII character ‘.’ is 0x2E. The URL `..\scripts\winnt\cmd.exe` could alternatively be expressed as `%2E%2E\scripts\winnt\cmd.exe`. There are other variations of hexadecimal encoding such as *double-percent hex encoding*, *double-nibble hex encoding*, *first-nibble hex encoding* and *second-nibble hex encoding* [25]. All such hexadecimal encodings are decoded correctly by the DAC HTTP normalizer.

UTF-8 encoding is used to represent unicode characters that are outside of the traditional 0-127 ASCII range. For example, the character ‘A’ can be encoded as `%C1%81`. Existing NIDS systems either disregard UTF-8 encoding or perform conversions for certain standard code pages. Likewise, our implementation has rules for converting commonly observed UTF-8 ASCII codes.

Bare-byte encoding is similar to UTF-8 encoding except that there is no ‘%’ character preceding the hexadecimal bytes. For example, `0xC10x81 = ‘A’`. Microsoft Windows IIS servers provide an additional format for encoding unicode characters that is commonly known as *unicode encoding*. In this format, unicode values are expressed as `%U` followed by 4 hexadecimal characters. For example, the character ‘A’ is `%U0041`.

Directory traversal is a common URL obfuscation technique. Attackers add `noop` directory traversals like `../../../../system32/../../../../` in system paths. This method effectively evades NIDS signatures that do not account for directory traversal attacks. Our DAC normalizes `noop` directory traversals.

A.2 NetBIOS/SMB Normalization

Invalid NetBIOS/SMB protocol fields may confuse intrusion detection systems. These could include, for example, NetBIOS packets with incorrect length fields. We identify such incidents from the response of our honeynet server. The ambiguities of unicode and UTF-8 encoding previously described for HTTP traffic apply for NetBIOS as well. If the server and clients both possess unicode capabilities, it is very common for NetBIOS clients to negotiate unicode. This implies that data fields such as filenames will be expressed as 16 bit unicode characters. All unicode data fields are converted to ASCII fields in the parse tree if unicode has been negotiated.

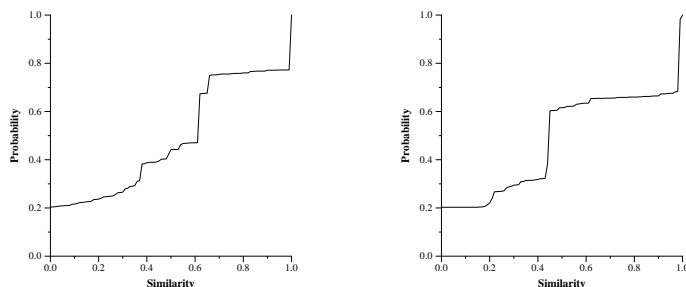


Figure 10: Port 445 edit distance (left) and cosine similarity (right) cumulative distribution functions.

NIDS systems and Nemean should also recognize and account for violations of session semantics. Examples of such violations include sending ASCII character bytes to a unicode negotiated session or sending messages in an invalid order. We rely upon the server’s response to recognize these situations.

Our normalizer additionally removes meaningless information from certain resource identifiers. Universal Naming Convention (UNC) is a standard method for naming files and resources in a network. This is supported by Microsoft Windows and can be used to refer to file requests in SMB. UNC names appear as `\\servername\sharename\path\filename`, where the server name can be either a domain name or an IP address. If it refers to the local IP address, it provides no meaningful information and is removed.

B Hierarchical Edit Distance Similarity Metric

Hierarchical edit distance computes the similarity between sessions A and B as a function of the number of modifications needed to convert A into B (or equivalently, B into A). This metric extends the well-known edit distance algorithm [5] for strings of characters to hierarchical vectors. In a hierarchical vector, elements may themselves be vectors. Terminal elements must have an equality test. The hierarchical edit distance similarity between sessions A and B is:

$$1 - \frac{\text{HIEREDITDIST}(A, B)}{\text{COST}(A) + \text{COST}(B)}$$

where $\text{HIEREDITDIST}(\cdot, \cdot)$ and $\text{COST}(\cdot)$ are given in Algorithm 1. This computation preserves connection ordering within a session. Equivalent connections appearing in different orders in different sessions reduce session similarity because edits would be required to produce session equivalence. As a result, hierarchical edit distance generally rates sessions as less similar than does the cosine metric.

Qualitatively, the edit distance metric produced a larger number of clusters than the cosine metric and the individual clusters were very accurate. The cosine metric produced fewer clusters, although the accuracy was surprisingly comparable to that of the edit distance. The reason for this is apparent from the cumulative distribution functions given in Figure 10. The graphs show the probability that any two sessions will have a similarity measure above the threshold on the x -axis. The stable region of the hierarchical edit distance metric, from 0.75 to nearly 1.0, is higher than the corresponding stable region for the cosine metric. This indicates that a greater number of sessions are similar, so the Star cluster graph contains more edges connecting sessions. The number of clusters produced depends upon the edge count, so edit distance produces more clusters. Note also that the cosine metric identifies a larger number of sessions as identical than does the hierarchical edit distance metric, as evidenced by the sharp increase at 1.0. The larger size of the stable region from 0.5 to almost 1.0 in for the cosine metric also implies that the cosine metric is less tightly coupled to a good threshold choice.

C HTTP Clusters

Clusters for port 80 traffic represent all of the most widely reported worms, some lesser-known exploits, and benign web-crawler traffic. Table 6 evaluates the Snort alerts for this malicious traffic. Figure 6 provides a summary of our clustering results. We see three significant clusters. Cluster 24 corresponds to sources that try to send an OPTIONS request to see the list of publicly supported HTTP commands. Typically these are sources looking for various WebDAV vulnerabilities. The other significant clusters include Nimda sources

```

HIEREDITDIST( $x, y$ ):
Input:  $x$  and  $y$  are either both terminals or both hierarchical vectors of identical height
Result: The edit distance between  $x$  and  $y$ 
begin
  if  $x$  and  $y$  are terminals then
    if  $x = y$  then
      return 0
    else
      return 1
    endif
  else
     $m \leftarrow \text{DEGREE}(x)$ 
     $n \leftarrow \text{DEGREE}(y)$ 
     $A \leftarrow (m + 1) \times (n + 1)$  matrix indexed at  $[0, 0]$ 

    for  $k \in [0, m]$  do  $A[k, 0] \leftarrow \sum_{i=1}^k \text{COST}(x_i)$ 

    for  $k \in [0, n]$  do  $A[0, k] \leftarrow \sum_{i=1}^k \text{COST}(y_i)$ 

    for  $i = 1$  to  $m$  do
      for  $j = 1$  to  $n$  do
         $\text{InsertCost} \leftarrow A[i - 1, j] + \text{COST}(x_i)$ 
         $\text{DeleteCost} \leftarrow A[i, j - 1] + \text{COST}(y_j)$ 
         $\text{ReplaceCost} \leftarrow A[i - 1, j - 1] + \text{HIEREDITDIST}(x_i, y_j)$ 
         $A[i, j] \leftarrow \text{MINIMUM}(\text{InsertCost}, \text{DeleteCost}, \text{ReplaceCost})$ 
      endfor
    endfor
    return  $A[m, n]$ 
  endif
end

```

```

COST( $s$ ):
Input: A vector or a terminal
Result: Cost of insertion or deletion of  $s$ 
begin
  if  $s$  is terminal then
    return 1
  else
     $k \leftarrow \text{DEGREE}(s)$ 
    return  $\sum_{i=0}^k \text{COST}(s_i)$ 
  endif
end

```

Algorithm 1: HIEREDITDIST computes the hierarchical edit distance between two sessions. $A[i, j]$ is the minimum number of insertions, deletions, or replacements required to convert the subvector $x_1 \dots x_i$ to the subvector $y_1 \dots y_j$. COST calculates the cost to insert or remove a hierarchical vector from an existing hierarchy.

Table 6: Assessment of Snort port 80 alerts

Scan	Snort alarms	Assessment
OPTIONS Request	Simple translate	Weak alert
PROPFIND Request	Simple translate	Weak alert
Welchia	Safe scan attempt WebDav search access U encoding	Correct
Frontpage	Chunked encoding	Incorrect alert
Web Crawler	none	Correct
Open-Proxy Scan	none	False negative
Nimda	WEB-IIS cmd.exe access Unicode directory traversal	Correct
Code-Blue	WEB-IIS cmd.exe access	Correct

Table 7: Assessment of Snort NetBIOS/SMB alerts

Scan	Snort alarms	Assessment
MS-RPC	Port 445: None Port 139: IPC\$ share access	False negative Weak alert
Deloder	Port 445: None Port 139: IPC\$/ADMIN\$ share access	False negative Weak alert
NAVSVC	Port 445: None Port 139: IPC\$ share access C\$ ADMIN\$ share access	False negative Weak alert Missing unicode
LoVGate	Port 445: None Port 139: IPC\$,ADMIN\$ share access	False negative Weak alert

(Cluster 1) and Welchia sources (Cluster 12). Scans for exploits of WebDAV vulnerabilities account for the majority of the scans in our dataset. The list below describes the major types of WebDAV scans.

OPTIONS request: This is the dominant port 80 request observed in our logs. The client sends an HTTP OPTIONS request of the form shown in Figure 11. The server returns the list of supported options. Scanners are trying to obtain a list of scriptable files by sending “translate: f” in the options header of the HTTP request [27].

PROPFIND exploit: PROPFIND requests are frequently associated with OPTIONS requests. Many sources first send an OPTIONS / request to see if PROPFIND is supported before attempting the PROPFIND exploit. PROPFIND is a WebDAV feature that returns lists of data. Sources use PROPFIND to attempt to view listings and content of cgi files on the target machine. As with malicious OPTIONS requests, the PROPFIND requests include the special “translate: f” in the header. As a result, the Snort alert is essentially equivalent to that for the simple translate exploit.

WebDAV buffer overflow exploit: All of the sources in cluster 12 are Welchia sources using a unique 3 step scanning process. A source first sends a GET / request, then a SEARCH / request. Finally, if it receives a 411 length required error message from the server for the SEARCH request, it sends a WebDAV SEARCH request containing data that overflows a buffer. Snort produces three alerts for this exploit: the first for to the

```

OPTIONS / HTTP/1.1
translate: f
User-Agent: Microsoft-WebDAV-MiniRedir/5.1.2600
Host: 10.104.138.47
Content-Length: 0
Connection: Keep-Alive

```

```

[**] WEB-IIS view source via translate header [**]

[Classification: access to a potentially vulnerable
web application] [Priority: 2]

```

Figure 11: Translate exploit HTTP request and Snort alert

```

POST /_vti_bin/_vti_aut/fp30reg.dll HTTP/1.1
Host: %s
Transfer-Encoding: chunked

```

```

alert tcp \$EXTERNAL_NET any -> \$HTTP_SERVERS \$HTTP_PORTS
(msg:'WEB-FRONTPAGE rad fp30reg.dll access';
uricontent: '/fp30reg.dll'; ...)

```

Figure 13: HTTP request and Snort rule for Frontpage exploit

first SEARCH request and two more alerts for the second SEARCH request [34].

Our cosine-metric clustering algorithm effectively aggregated WebDAV exploits. Nimda sessions were divided into multiple clusters due to the variants of Nimda and varying directory prefixes used in these scans. Due to these differences, our clustering algorithm separates the common scanning episodes of the well known variants from the isolated scans. This is important because the isolated scans might also be associated with other less common or unknown exploits. For example, we found an exploit for the Code Blue worm being clustered along with the Nimda sources as shown in Figure 12.

A Frontpage exploit is among the port 80 clusters. Figure 13 shows the Snort rule for the Frontpage exploit. Interestingly, Snort did not generate an alert for this exploit when run against our trace despite the presence of the rule in its dataset. The trace was misdiagnosed as a chunked encoding attack directed against vulnerable Apache servers. It seems that the presence of chunked encoding in the HTTP header prevents other rules from executing correctly.

Figure 6 also shows a cluster (Cluster 2) of scans from web crawlers. It is important to note that the scans from web crawlers did indeed get clustered together. Normally, we would not expect scans from web crawlers to be seen at at honeynet because these IP addresses neither have DNS entries nor host any content. Hence, they should not be linked by any other web pages. Our analysis revealed that these scans were, in fact, due to obsolete DNS entries.

The CONNECT request is used for tunneling requests via proxy servers. Open-proxy servers are popular in some countries as a means to obfuscate surfing activity. They are also often used by spammers to forward mail. Figure 14 shows two different instances of CONNECT requests. *Snort did not fire an alert for either one of these scans.*

D NetBIOS/SMB Clusters

NetBIOS/SMB scanners that probe ports 139 and 445 (with the possible exception of the MS RPC scanners such as Blaster) are predominantly email viruses which also have a network share propagation component. The major clusters include sources accessing the Security Account Manager samr pipe such as the Lioten (iraq_oil) worm, sources accessing the MS-RPC epmapper pipe such as the Agobot (Sophos) worm [31], the Deloder worm [13], and NvcplDmn. Table 7 provides a summary of the Snort alerts for NetBIOS/SMB scans.

The RPC Endpoint Mapper (epmapper) maintains the connection information for the RPC processes in a Windows machine. Scanners use the NetBIOS/SMB service to connect to the epmapper service and indirectly exploit the same vulnerability as worms like Blaster and Welchia [16]. Besides Blaster, scanners connecting to this share include machines infected with variants of the Agobot worm [31]. These machines

```

GET /scripts/../../../../winnt/system32/cmd.exe?/c+dir

GET /_vti_bin/..tftp+-i+%s+get+httpext.dll..tftp+-i+%s+get+httpext.dll..tftp+-i+%s+get+httpext.dll..tftp+-i+%s+get+httpext.dll..winnt/system32/cmd.exe?/c+dir

GET /iisadmin/../../../../winnt/system32/cmd.exe?/c+dir

```

Figure 12: Code Blue and Nimda attacks. Requests 1 and 3 are Nimda, while request 2 is Code Blue.

```

CONNECT 1.3.3.7:1337 HTTP/1.0
CONNECT ns.watson.ibm.com:25 HTTP/1.0

```

Figure 14: Two instances of CONNECT (Open-proxy exploit)

```

alert tcp $EXTERNAL_NET any -> $HOME_NET 139
(msg:"NETBIOS SMB IPC$ share access";
 flow:to_server,established; content:"|00|";
 offset:0; depth:1; content:"|FF|SMB|75|";
 offset:4; depth:5; content:"\\IPC$|00|";
 ...)

alert tcp $EXTERNAL_NET any -> $HOME_NET 139
(msg:"NETBIOS SMB IPC$ share access (unicode)";
 flow:to_server,established; content:"|00|";
 offset:0; depth:1; content:"|FF|SMB|75|";
 offset:4; depth:5;
 content:"|5c00|I|00|P|00|C|00|$|00|";
 ...)

```

Figure 15: Two Snort rules for IPC share access

```

alert tcp $EXTERNAL_NET any -> $HOME_NET 139
(msg:"NETBIOS SMB ADMIN$access";
 flow:to_server,established;
 content:"\\ADMIN$|00 41 3a 00|";
 reference:arachnids,340; classtype:attempted-admin;
 sid:532; rev:4;)

```

Figure 16: Snort rule for ADMIN share access

Table 8: Snort false alarm summary for over 45,000 HTTP sessions collected from our department’s border router.

<i>Alert</i>	<i>Volume</i>
Non-RFC HTTP Delimiter	32246
Bare Byte Unicode Encoding	28012
Apache Whitespace (TAB)	9950
WEB-MISC /doc/ Access	9121
Non-RFC Defined Character	857
Double-Decoding Attack	365
IIS Unicode Codepoint Encoding	351

then create the file `Nvscv32.exe`.

Snort signatures are particularly weak in detecting SMB exploits, especially against port 445. The Snort rules contain no references to `Nvscv32.exe`. However, rules to detect connections to the IPC\$ share exist, as shown in Figure 15. The second rule is for unicode negotiated clients. These are very general rules that lack specificity and encompass virtually every NetBIOS worm in the wild. Surprisingly, the two rules did not fire alerts on the SMB exploits because they were written only for port 139. As mentioned earlier, most NetBIOS worms attack both 139 and 445 simultaneously and tend to prefer port 445 (raw SMB). Deeper evaluation of this signature showed that it was unnecessarily restrictive. It checked for content matching: `FF|SMB|75|`, where 75 is the SMB command code. The code 0x75 corresponds to the `SMBTreeConnectAndX` command. However, this command is interchangeable with either `SMBtcon` (0x70) or `SessionSetupAndX` (0x73) [6]. We observed several instances of missed attacks where the sources used `SessionSetupAndX`.

Deloder Worm/Virus: The Deloder Worm targets port 445 and connects to the IPC and ADMIN shares. The worm uses simple password attacks to spread to Windows 2000 and Windows XP machines [13]. The worm attempts to create the file `psexecsvc.exe` in the `System32` folder. Snort contains no signatures to specifically detect Deloder, although there are general signatures that detect connections to the IPC and ADMIN shares. As with the Agobot worm, these signatures are present only for port 139 and not port 445. Figure 16 shows the Snort rule detecting connections to ADMIN shares on port 139. The rule set is missing a corresponding rule as in Figure 15 for unicode negotiated clients.

NAVSVCS.exe We have identified what seems to to be a trojan (`NvcplDmn`) or an adware with a network scanning component. The worm connects to the IPC share and C share and attempts to copy the file `Navsvc.exe` to the startup folders. Snort does not have a special rule to detect this worm. On port 445 this worm would not fire any alarms. On port 139, the closest rules are those that detect IPC and C share accesses. Again, the rule to detect C share access would need a separate rule for unicode negotiated sessions.

LovGate LovGate is an email virus that spreads via network shares and was primarily observed on port 139. It tries to connect to the ADMIN and IPC shares and then attempts to access the `svctl` named pipe (service control manager). The worm drops the file `Netservices.exe` using the `NTCreateX` command and once infected tries to send emails through the SMTP server `www.163.com`. There are no specific rules in Snort to detect connections to the service control manager pipe or creation of any of the virus files.

D.1 False Alarm Details: Productive Data Department Border Router

It is well known that overly general Snort signatures produce a large number of false alarms on benign network traffic. Table 8 provides a summary of the Snort alarms generated on an 8 hour trace of overwhelmingly benign HTTP traffic collected at our department's border router. Reducing Snort alarm rate would require reengineering of many signatures. Additionally the overly general signature provides little specific information about the type of exploit that may be occurring.