Internet Sieve: An Architecture for Generating Resilient Signatures

Vinod Yegneswaran
Jonathon Giffin
Paul Barford
Somesh Jha

Technical Report #1507

May 2004

Internet Sieve: An Architecture for Generating Resilient Signatures

Vinod Yegneswaran, Jonathon T. Giffin, Paul Barford, Somesh Jha
{vinod, giffin, pb, jha}@cs.wisc.edu
University of Wisconsin Technical Report

Abstract

We present *iSieve*, a modular architecture for identifying intrusion profiles in packet trace data and automatically constructing resilient signatures for the profiles. The first component of the architecture organizes and normalizes packet trace data collected from honeynets. The second component classifies this data into attack profiles based upon data similarity measures. The final component uses machine learning methods to generate an automaton for each attack profile. These automata can then be used as signatures by network intrusion detection systems. We show how a large, diverse data set is effectively summarized by each component of our system and use these results to highlight implementation considerations in the architecture. Evaluation demonstrates iSieve's ability to generate resilient signatures for many different intrusion profiles. For example, our learned signatures detect 99.98% of the intrusive sessions in NetBIOS data and generate no false alarms.

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 [14, 16, 30]. 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. In their recent analysis of the Witty worm, Moore *et al.* [15] state that zero-day worms such as Witty have effectively demonstrated how the current "patch model for Internet security has failed spectacularly".

Network security architectures often include network intrusion detection systems (NIDS) that monitor packet traffic between networks for malicious activity and raise alarms upon intrusions. Commonly used NIDS compare traffic against a hand-built signature database that contains previously documented attack profiles [2, 18]. A NIDS generates false alarms if a signature identifies features in traffic streams that are common to both malicious and benign packets. The standard objective in creating signatures is to make them as specific to a given attack profile as possible to minimize false alarms. If a signature is too specific, however, an attack profile can be easily morphed to render it ineffective. Ideally, a *resilient signature* would incorporate common obfuscation transformations used by attackers so that it could accurately identify variants of known attacks.

This paper introduces a modular architecture called *Internet Sieve (iSieve)* that identifies intrusions and automatically generates such resilient signatures. We use packet traces from a honeynet[1] [7] deployed on unused IP address space as input to our system. Any data observed at a honeynet is anomalous[2], thus eliminating both the problem of privacy and the problem of separating malicious and normal traffic. We assume that the honeynet receives attack traffic similar to that observed on standard hosts and discuss the ramifications of this assumption in Section 7. Our architecture includes three components. The *data abstraction component* normalizes packets from individual sessions in the collected data and converts them into a standard format for evaluation. The *clustering component* uses machine learning to group sessions that have similar charac-

[1] A honeynet is defined as a network of high-interaction honeypots.

[2] A negligible amount of non-malicious traffic observed on our honeynet is due to misconfiguration and is easily separated from the malicious traffic.

teristics into clusters. Intuitively, each cluster corresponds to a specific attack and its variants observed at the honeynet. The *signature generation component* produces a connection-level or session-level finite-state automaton (FSA) signature for each cluster, suitable for deployment in a NIDS. As discussed in Section 2, this architecture differs significantly from prior work. Section 3 describes our modular system architecture.

To evaluate iSieve's architecture, we developed a prototype implementation of each component. We have also developed a simple alert generation tool for off-line analysis which compares packet traces against FSA 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 substituted. 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 iSieve's capability in a comprehensive security architecture. Section 4 presents our prototype implementation of iSieve.

We performed two evaluations of our implementation. First, we calculated detection and misdiagnosis counts using packet traces collected at two unused /19 address ranges totaling 16K 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) using the data collection environment described in Section 5. We then used this packet trace data as input to iSieve to produce a comprehensive signature set for the three ports. In Section 6, we describe the major clusters and the signatures produced. 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 99.1% of the HTTP exploits and 99.98% of the NetBIOS exploits with 0 misdiagnoses. Second, we validated our signatures by testing for false alarms using packet traces of all HTTP traffic collected from our department's border router. iSieve produced 0 false alarms on this dataset. For comparison, Snort [2] generated over 88,000 false alarms on the same data set. These results suggest that even with a much smaller signature set, iSieve achieves detectability rates on par with Snort while identifying attacks with superior precision and far fewer false alarms.

2 Related Work

Most network intrusion detection systems including Snort [2], Bro [18], and a collection of commercial tools use misuse detection to identify attacks. Misuse detection scans for specific malicious patterns or signatures in monitored traffic. Sommer and Paxson proposed extending such systems to use session-level signatures [26]. Signatures generated by iSieve may be added to any misuse detector, although our signatures are particularly suited to systems that use regular expressions and provide session-level context, such as Bro.

Anomaly detection systems such as EMERALD [17] and STAT [29] use statistical profiles of normal traffic rather than signatures to identify attacks. Thus, our techniques do not apply to these systems. However, anomaly detectors may enable signature learning in environments with mixed malicious and normal traffic. Our clustering component expects its input traffic stream to be largely devoid of benign traffic, as a large volume of normal traffic would potentially exhibit great variation and skew the clusters. If the anomaly detector can cull out normal traffic, clustering and learning can then occur as described in this paper.

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 [6]. Similar work describes common HTTP evasion techniques and standard URL morphing attacks [21]. They inform the development of our data abstraction component which normalizes session data at both transport and application levels for HTTP and NetBIOS data.

Honeypots provide data suitable for intrusion and attack analysis. Levin *et al.* describe how honeypots extract details of worm exploits that can be analyzed to generate detection signatures [11]. Unlike our architecture, their signatures are generated by hand.

Honeycomb is a host-based intrusion detection system that includes signature generation capabilities with design goals similar to our own [9]. Honeycomb extends Honeyd [19], an open source honeypot tool that

```
01:48:18.076540 213.189.83.103.47606 > 128.104.134.106.80:
P 1:150(149) ack 1 win 17520 <nop,nop,timestamp 878172804 0> (DF)

SEARCH./.HTTP/1.1..Host:.128.104.134.106..
Connection:.keep-alive.
 X-Forwarded-For:.62.150.189.187..
 Via:.1.1.supercache2.(NetCache.NetApp/5.3.1R2)....

01:48:15.751137 213.189.83.103.47276 > 128.104.134.106.80: P 1:273(272)
ack 1 win 17520 <nop,nop,timestamp 878172572 0> (DF)

GEI./.HTTP/1.1..Host:.128.104.134.106..
Connection:.keep-alive..
Accept:.image/gif,.image/x-xbitmap,.image/jpeg,.image/pjpeg,.*/*..
User-Agent:.Mozilla/4.0.(compatible;.MSIE.5.5;.Windows.98).
 X-Forwarded-For:.62.150.189.187..
 Via:.1.1.supercache2.(NetCache.NetApp/5.3.1R2)....
```

Figure 1: Two HTTP payloads with proxy cache headers. Bold text marks strings identified by the LCS algorithm.

simulates virtual machines over unused IP address space. Honeycomb uses the *longest common substring* (LCS) algorithm on packet-level data recorded by Honeyd to automatically generate signatures. We consider Honeycomb to be the canonical example of an LCS tool. More recent systems that extend the basic LCS ideas include Earlybird and Autograph [8, 25].

Honeycomb is inherently limited because it does not consider higher-level protocol structure and semantics. Its LCS algorithm is frequently "distracted" by long strings in the packet payload that are irrelevant for attack identification. Figure 1 shows two HTTP requests scanning for WebDAV exploits. The URL is the significant discriminating field but is only a small portion of the entire payload. The LCS algorithm identifies the strings in bold as the signature although these strings have no direct bearing on the exploit. Hence, Honeycomb is poorly suited to attacks with small exploit strings. It has been demonstrated to produce relevant signatures only for buffer-overflow exploits where the attack string is very long [9]. Systems like Autograph that ignore higher level protocol semantics are also susceptible to similar problems.

Another deficiency of Honeycomb's LCS algorithm is that it outputs a single string for each signature. The consequence is that Honeycomb cannot produce signatures for protocols like NetBIOS/SMB [9]. In such data, the relevant fields are a small fraction of the entire payload and may not be contiguous. Hence, signatures seen in real-world systems like Snort are often composed of multiple substrings of varying lengths that are all equally valuable. We argue that a more pragmatic approach is to build a system that incorporates higher-level protocol semantics into the signature-generation algorithm. Our data abstraction component marks the pertinent data in each session so that clustering can focus only upon the relevant data. We also use a clustering algorithm that is less dependent upon the size of the exploit string. As a result, our automaton-based signatures can easily describe such patterns and enable iSieve to produce very accurate NetBIOS signatures.

3 Internet Sieve Architecture

We have designed a framework for automatically generating robust signatures via efficient monitoring of honeynets. As shown in Figure 2, iSieve's architecture is divided into three components: the data abstraction layer, the clustering module, and the signature generator. The input to iSieve could be packet traces collected at any network. However, in this paper, we consider the particular instance of data collected on honeynets. This traffic is attractive because it primarily consists of malicious traffic. 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.

3.1 Data Abstraction Component

The Data Abstraction Layer (DAL) 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 com-

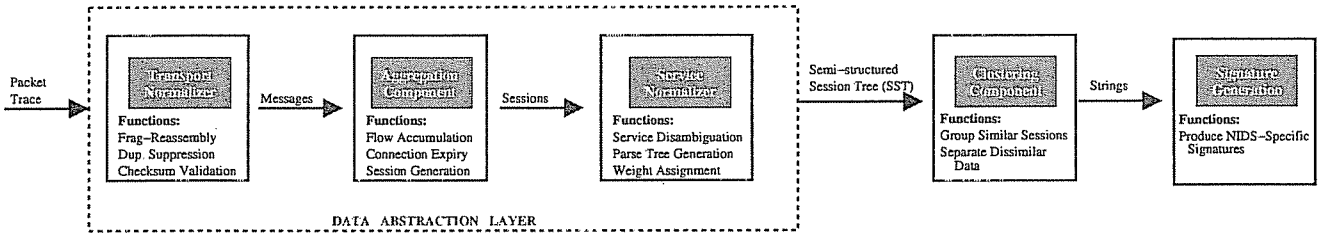

Figure 2: Components and data flow description of the Internet Sieve architecture

ponents of the DAL can then be thought of in terms of the data flow through the module as shown in Figure 2.

Transport normalization disambiguates obfuscations at the network and transport layers of the protocol stack. The DAL 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 DAL 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 [2]. We describe the particular normalizations performed in greater detail in Section 4.

The DAL finally transforms the normalized sessions into XML-encoded SSTs suitable for input to the clustering module. This step 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 Clustering Component

The clustering component identifies sessions belonging to the same attack. It classifies sessions according to a similarity or distance metric. Sessions grouped together correspond to a single attack or variants of the attack. Distinct clusters correspond to distinct attacks or attack variants that differ significantly from some original attack. Effective clustering requires two properties of the attack data. First, sessions that correspond to an attack and its variants should be measurably similar. A clustering algorithm can then classify such sessions as likely belonging to the same attack. Second, sessions corresponding to different attacks must be measurably dissimilar so that a clustering algorithm can separate such sessions. We believe that the two required properties of session data 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. Internet Sieve's use of honeynet data enables a reasonable number of meaningful clusters to be produced.

Each cluster ideally contains the set of sessions for some attack. We presume that these sessions 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 resilient signature generation step.

3.3 Signature Generation

The signature generation component 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, 26].

Clusters that contain 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 algorithm 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.

Our current implementation of the signature generation engine works best with minimal 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 4.5). 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.

4 Design and Implementation

We have implemented prototypes of each iSieve component. We focus on two specific protocols, 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 [6, 20]. 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, iSieve's current implementation analyzes network traces offline which relaxes its state holding requirements and makes it less vulnerable to resource-consumption attacks.

An adversary could also attempt to evade a NIDS by introducing ambiguities to IP packets. Examples include simple *insertion attacks* that would be dropped by real systems but are evaluated by NIDS, and *evasion attacks* that are the reverse [20]. Since iSieve 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 overlap-

ping 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* [24] 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 if a new connection setup is observed between the same host/port pairs.

4.2 Service-Level Normalization: HTTP

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. An attacker has available a large variety of HTTP URL encoding techniques. Our DAL correctly decodes hex encoding and its variants, UTF-8 encoding, bare-byte encoding, and Microsoft unicode encoding. Regardless of its encoding, the DAL presents a canonical URL in ASCII format to the clustering module. We provide details on some commonly observed encoding schemes.

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* [22]. All such hexadecimal encodings are decoded correctly by the DAL 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 DAL normalizes `noop` directory traversals.

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.

4.3 Service-Level Normalization: NetBIOS/SMB

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.

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.

NIDS systems and iSieve 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.

4.4 Clustering

The iSieve clustering component classifies normalized sessions into groups containing similar data. We implemented the on-line star clustering algorithm, which clusters documents based upon a similarity metric [1]. This algorithm has advantages over the more common k-means family of clustering algorithms [12]. For example, it 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 intend 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 sessions with semantically equivalent data. We arbitrarily choose one session at each node to be the *representative session.* A link exists between two nodes if the similarity between the corresponding representative sessions 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.

Session similarity determines how edges are placed in the graph. We implemented two different similarity metrics to test sensitivity: *cosine similarity* [1] and *hierarchical edit distance* (Appendix A). The cosine similarity metric has lower computational complexity than hierarchical edit distance and was used for our experiments in Section 6.

Cosine similarity computes the angle between two vectors representing the two sessions under comparison. For each session A, we build a 2^8-ary vector D_A giving the distribution of bytes that appeared in the session data. If θ is the angle between vectors D_A and D_B representing sessions 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 sessions is the value $cos\,\theta$, with $cos\,\theta = 1$ indicating session equality.

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 accuracy of 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 A.

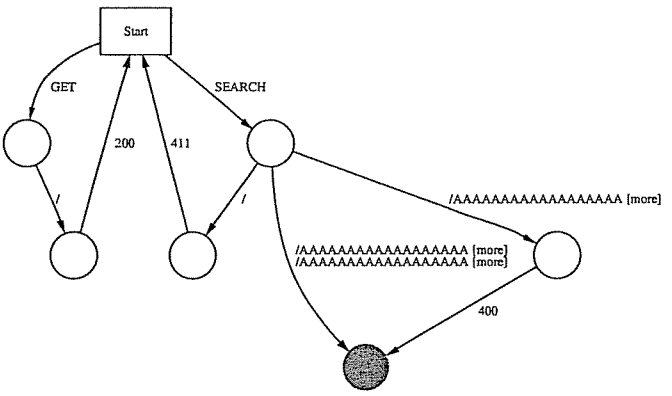

Figure 3: Welchia session level signature

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 finally passed to the signature generation module.

4.5 Signature Generation

Signature generation devises a NIDS signature from a cluster of similar attack sessions. We construct a finite state automaton (FSA) signature for each cluster. These signatures are suitable for use in a detection system that can use regular expressions as signatures. If a FSM contains no loops, then enumeration of all accepted data streams can produce a family of signatures for use in systems requiring simple string signatures.

The process of generating signatures from clusters is complicated by our desire to generate resilient signatures for attack variants that have not been seen by the cluster module. We would like the signatures to be sufficiently general so that they can detect all attacks that match the class of attacks in the cluster, yet generate few false alarms. 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. We implemented two generalizations: *structure abstraction* and *subsequence creation.*

Structure abstraction adds new sequences to the automaton based upon the previously existing patterns in the signature. This can occur in two ways. Let A, B, X, and Y be sequences of observed session data (*e.g.* each is a contiguous sequence of transitions in the automaton). Let AB denote concatenation of symbols. First, if the signature includes the patterns AB, AY, and XY, then we add the sequence XB. Intuitively, the pre-existing patterns AB and AY indicate that B may be interchanged with Y, and hence XB is added to the signature. Second, if the signature accepts the sequences AC and ABC, then add the signature AB^*C accepting an arbitrary number of the sequence B. This generalization introduces loops into the signature. The cycles in the Welchia signature (see Figure 3) accepting any number of arbitrarily interleaved `GET / 200` and `SEARCH / 411` occur due to structure abstraction.

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 and B be as above. Let v, w, x, and y be single data elements. If the signature contains 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. The wild-card transitions in the optimized Nimda signature shown in Figure 4 correspond to subsequences in the observed dataset that exhibited significant variability.

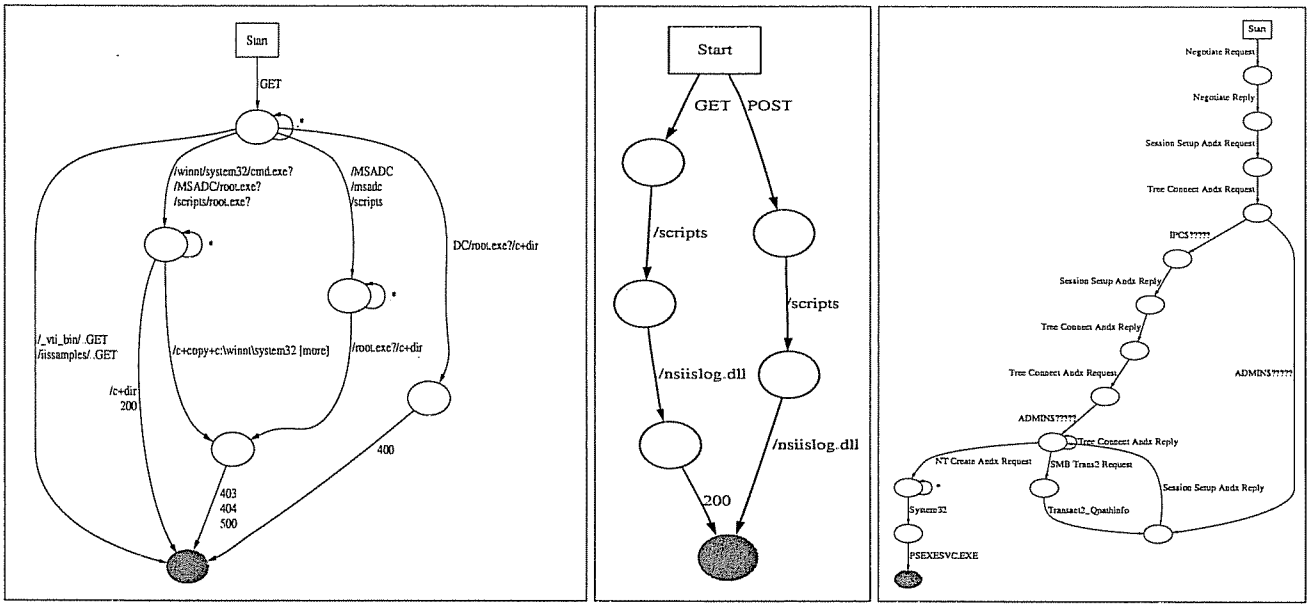

Figure 4: Nimda, Windows Media Player Exploit, and Deloder connection level signatures

The resulting FSM may thus accept a greater collection of sessions than those present in the clusters. Each accepted string has characteristics in common with the attack strings observed in the cluster and is likely an attack variant not previously observed. Our resulting signatures are thus not beholden only to data observed during training but can detect previously unknown attacks.

Our signature generation engine can produce both session-level and connection-level signatures. Session-level signatures are more restrictive as they impose ordering upon the constituent connections and must match a collection of connections. As a result, they are less likely to generate false alarms, although they may miss attack variants. Connection-level signatures are less restrictive and better generalize to previously unseen attacks, although this freedom may lead to a higher false alarm rate. For attacks like Nimda that exhibit great diversity, session semantics are not meaningful and connection-level signatures are more appropriate. For worms like Welchia, session-level signatures are more accurate.

Figure 3 shows a session-level signature for Welchia, a worm that exploits a buffer overflow. Structure abstraction produced a general signature that matches a wide class of Welchia scans without losing the essential buffer overflow information characteristic to the worm. Figure 4 shows the signatures generated 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 iSieve to generate signatures for exploits using more complex protocols like NetBIOS/SMB.

5 Data Collection

Traffic from two unused /19 IP address blocks totaling 16K addresses from address ranges allocated to our university was routed to our monitoring environment. This environment contained three primary components: a NAT filter, a VMware honeynet, and an active sink (see Figure 5).

The NAT filter responded to the intra-campus border router via ARP (Address Resolution Protocol) as the next hop for the specified address blocks. The NAT filter used a packet filter to capture the traffic addressed to these blocks. The packets received were first filtered using a simple filtering rule: one destination IP address per source. Connections to additional destination IP addresses were dropped by the filter. This filtering rule

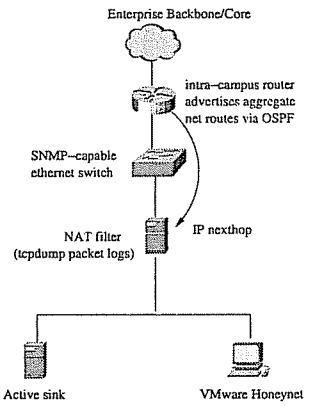

Figure 5: Data collection environment

Table 1: Data Summary

	Learning Data				*Test data*			
Port	**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

reduced the volume of traffic by over two orders of magnitude and provided a consistent view of the network to the adversaries. The filtered packets were then forwarded to either the active sink system or the VMware honeynet, based upon the service.

We routed HTTP requests to a fully patched Windows 2000 Server running on VMware. We routed NetBIOS/SMB packets to the 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 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, NetBIOS connection requests were occasionally rejected due to too many simultaneous network share accesses.

We collected the data over a two day period. We collected three sets of traces, one for each port. Table 1 provides a high level description of the traces. The number of sessions is roughly the same as the number of unique sources with the discrepancy due to sessions that expired after a timeout. The number of port 139 connections and sources is much smaller than that of port 445. Although the typical worm scans ports 139 and 445 simultaneously, we did not increment the connection count and the source count until we received data packets. Most sources tended to prefer port 445 and did not send data packets to port 139 if the former connection was successful.

6 Evaluation

We tested the effectiveness of our generated HTTP and NetBIOS signatures and examined the session clusters used to produce these signatures. Sections 6.1–6.3 reveal the major classes of attacks in our recorded data and quantitatively measure the clusters produced by the clustering module. We conclude with an evaluation of the detection and false positive rates of iSieve's signatures and compare our results with Snort's HTTP capabilities.

6.1 HTTP Clusters

We begin with 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 4 contains a connection-level iSieve Nimda signature.

The remaining clusters observed in our data include clusters for the Welchia worm, a Frontpage exploit, web crawlers, and an open-proxy exploit. Appendix B provides more details of these exploits.

6.2 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. The first set of viruses, including LovGate [3], NAVSVC, and Deloder [10], use brute force password attacks to look for open folders and then deposit virus binaries in startup folders.

The second set of viruses are more sophisticated. Windows provides the ability to access system services such as `epmapper` (Windows RPC Service), `srvsvc` (Windows Server Service), and `samr` (Windows Security Account Manager). These viruses connect to default hidden shares on Windows such as IPC or ADMIN to access these services. This enables them to launch more intelligent attacks. For example, connecting to the `samr` service allows the attacker to obtain an enumeration of domain users, and accessing the RPC service allows access to the well known RPC-DCOM exploit [13]. We provide more details in Appendix C.

6.3 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. They were not used in clustering and served only as evaluation aids. It is possible for a session to be marked with multiple conjectures. We used these conjectures to measure 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.

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

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

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 particularly true for NetBIOS data, the similarity threshold has a significant impact on the quality of the resultant clustering in general. However, there are

```
CLUSTER   0:    598 Unique Client IPs,     739 Sessions
       Identified as NIMDA                :   739 (100%)
       Identified as Code Blue            :    21   (3%)
       Identified as Code Blue Variant    :    14   (2%)
CLUSTER   1:     70 Unique Client IPs,      71 Sessions
       Identified as NIMDA                :    71 (100%)
CLUSTER   2:      3 Unique Client IPs,       7 Sessions
       Identified as web crawler          :     6  (86%)
       Identified as Unknown              :     1  (14%)
CLUSTER   3:      1 Unique Client IPs,       1 Sessions
       Identified as Unknown              :     1 (100%)
CLUSTER   4:      1 Unique Client IPs,       1 Sessions
       Identified as real media player    :     1 (100%)
CLUSTER   5:     89 Unique Client IPs,      93 Sessions
       Identified as SEARCH               :    93 (100%)
CLUSTER   6:     10 Unique Client IPs,      10 Sessions
       Identified as SEARCH               :    10 (100%)
CLUSTER   7:    116 Unique Client IPs,     131 Sessions
       Identified as Unknown              :   125  (95%)
       Identified as NIMDA                :     6   (5%)
CLUSTER   8:      8 Unique Client IPs,       8 Sessions
       Identified as Code Red Retina      :     8 (100%)
       Identified as SEARCH               :     5  (63%)
CLUSTER   9:      1 Unique Client IPs,       1 Sessions
       Identified as Unknown              :     1 (100%)
CLUSTER  10:      1 Unique Client IPs,       1 Sessions
       Identified as Unknown              :     1 (100%)
CLUSTER  11:      1 Unique Client IPs,       1 Sessions
       Identified as Code Blue            :     1 (100%)
```

```
CLUSTER  12:    751 Unique Client IPs,     817 Sessions
       Identified as Welchia              :   817 (100%)
       Identified as SEARCH               :   801  (98%)
CLUSTER  13:      6 Unique Client IPs,       7 Sessions
       Identified as Unknown              :     7 (100%)
CLUSTER  14:     14 Unique Client IPs,      14 Sessions
       Identified as Windows Media Exploit:    14 (100%)
CLUSTER  15:      1 Unique Client IPs,       1 Sessions
       Identified as Unknown              :     1 (100%)
CLUSTER  16:      2 Unique Client IPs,       2 Sessions
       Identified as FrontPage Exploit    :     2 (100%)
CLUSTER  17:      7 Unique Client IPs,       7 Sessions
       Identified as PROPFIND             :     7 (100%)
       Identified as OPTIONS              :     7 (100%)
CLUSTER  18:     14 Unique Client IPs,      17 Sessions
       Identified as PROPFIND             :    17 (100%)
       Identified as OPTIONS              :    17 (100%)
CLUSTER  19:     41 Unique Client IPs,      93 Sessions
       Identified as PROPFIND             :    93 (100%)
       Identified as OPTIONS              :    93 (100%)
CLUSTER  20:      1 Unique Client IPs,       2 Sessions
       Identified as Kazaa                :     2 (100%)
CLUSTER  21:      1 Unique Client IPs,       1 Sessions
       Identified as Kazaa                :     1 (100%)
CLUSTER  22:      1 Unique Client IPs,       1 Sessions
CLUSTER  23:      1 Unique Client IPs,       1 Sessions
       Identified as Open Proxy           :     1 (100%)
CLUSTER  24:   9184 Unique Client IPs,   10525 Sessions
       Identified as OPTIONS              : 10525 (100%)
```

Figure 6: HTTP Port 80 cluster report

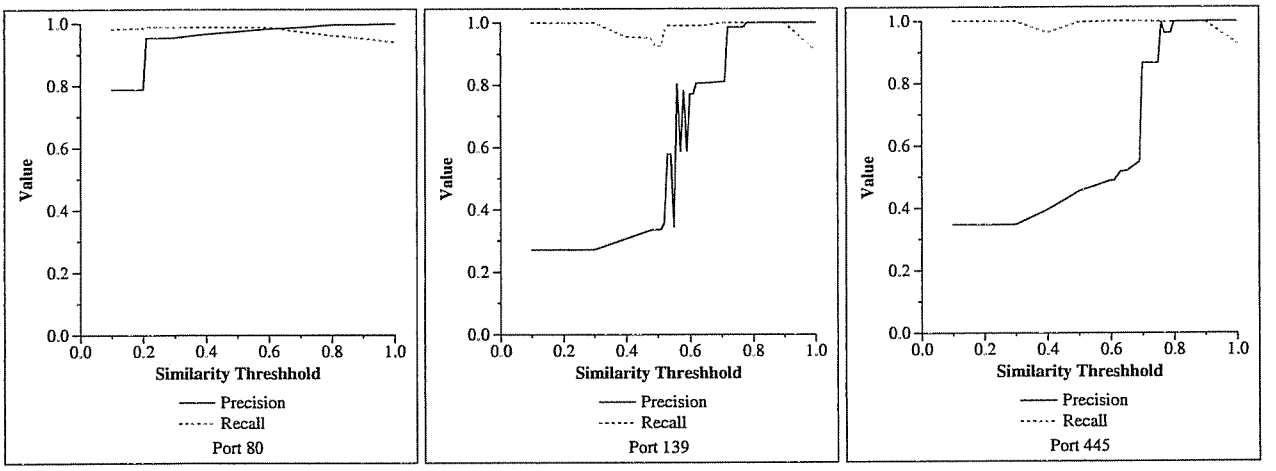

Figure 7: Effect of clustering similarity threshold upon weighted precision and weighted recall. Points of high volatility in precision scores correspond to significant shifts in the resulting clusterings.

clearly good values to use. The range from 0.75 to 0.9 maximizes both precision and recall scores for NetBIOS data and is effective for HTTP data. At the clustering threshold used in our experiments (0.8), precision scores were perfect or nearly perfect.

We favor clusterings with high precision even if recall scores diminish. In a non-hierarchical clustering scheme like star clustering, it is easier to merge two clusters of similar data than it is to split a cluster of differing data. We thus allow similar data to potentially be divided into two clusters, dropping the recall, knowing that we may later manually combine the clusters. Conversely, if dissimilar data is clustered together, improving recall but decreasing precision, then a single cluster may contain sessions from different exploits and it may be difficult to generate reasonable signatures.

6.4 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 iSieve 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.

Table 2: Session-level HTTP signature detection counts for iSieve signatures and Snort Signatures. We show only exploits occurring at least once in the training and test data.

Signature	*Present*	*iSieve*	*Snort*
Options	1174	1162	1173
Nimda	500	497	499
Propfind	229	225	229
Welchia	92	92	92
Windows Media Player	89	89	89
Code Red Retina	4	4	0
Kazaa	2	2	2
Open Proxy	1	0	0

Table 3: Detection and misdiagnosis counts for connection-level iSieve NetBIOS signatures. This data includes both port 139 and port 445 traffic.

Signature	*Present*	*Detected*	*Misdiagnoses*
Srvsvc	19973	19967	0
Samr	8743	8742	0
Epmapper	1224	1223	0
NvcplDmn	62	61	0
Deloder	30	30	0
LoveGate	1	1	0

• **99.9% Detection Rate:** We evaluated the detection rate of iSieve 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).

We detected 99.0% of the attacks present in the HTTP data. We used connection-level signatures for PROPFIND and Nimda and session-level signatures for all other attacks. Table 2 shows the number of occurrences of the HTTP attacks and the number detected by iSieve 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 3 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 2, all iSieve 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 3.

We also measured false alarm counts of 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 iSieve and Snort against the dataset. Table 4 shows the encouraging results for iSieve signatures: 0 false alarms.

Our automatically-generated iSieve signatures substantially improve upon the false alarm rate of Snort signatures. Snort generated 88,000 alarms on this dataset, almost all of which were false alarms. Table 5 lists the top seven Snort false alarm categories by volume. 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. Table 5 shows that the types of alerts produced by Snort 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 C. iSieve, 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.

Table 4: HTTP false alarm counts for iSieve signatures. The test data set contained 47,473 sessions collected from our department's border router.

Signature	*iSieve*
Welchia	0
Nimda	0
Code Red	0
Options	0
Propfind	0
Windows Media Player	0
Kazaa	0
Web Crawler	0
Open Proxy	0

Table 5: Snort false alarm summary for 47,473 HTTP sessions collected from our department's border router.

Alert	*Volume*
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

• **Low Noise due to Session-Level Signatures:** Moreover, iSieve 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 iSieve 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 iSieve. It achieves detection rates similar to Snort with dramatically fewer false alarms. The alerts produced by iSieve exhibit high quality, specifying the particular attack detected and keeping detection noise small.

7 Discussion

A potential vulnerability of iSieve 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 iSieve 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 [25] 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.

Second, iSieve 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 4. 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 iSieve 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, iSieve may be run online. This makes iSieve 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

```
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

iSieve to flag compelling clusters for worm outbreaks. Automatically generated iSieve signatures for these clusters could then be rapidly propagated to NIDS to defend against emergent worms. The resilience of iSieve signatures to false positive makes such a deployment practical.

8 Conclusions

We have described the design and implementation of iSieve, an architecture for automated generation of resilient NIDS signatures. iSieve is composed of three integral components: the data abstraction component, the cluster component, and the signature generation component. This modular design supports and encourages independent enhancement of each piece of the architecture. We evaluated our system's capability using data collected at two unused /19 subnets. We collected data for two services for which we developed service normalizers (HTTP and NetBIOS/SMB). Running iSieve with this data as input resulted in clusters for a wide variety of worms and exploits. Our evaluation suggests that simple similarity metrics like the cosine metric can provide clusters with a high degree of precision. We also demonstrated the signature generation capability of our system and discussed optimizations used in signature generation such as structure abstraction and subsequence creation. We showed that iSieve 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. Our analysis of the Snort rules for these clusters revealed that Snort performs poorly across both services.

References

[1] 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.

[2] Brian Caswell and Marty Roesch. The SNORT network intrusion detection system. http://www.snort.org, April 2004.

[3] Tony Conneff. W32.HLLW.lovgate removal tool. http://securityresponse.symantec.com/avcenter/venc/data/-w32.hllw.lovgate.removal.tool.html, April 2004.

[4] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to Algorithms*. MIT Press, 2^{nd} edition, 2001.

[5] Timothy Devans. SMB command codes. http://timothydevans.me.uk/nbf2cifs/smb-smbcommandcode.html, April 2004.

[6] Mark Handley, Vern Paxson, and Christian Kreibich. Network intrusion detection: Evasion, traffic normalization and end-to-end protocol semantics. In *10^{th} USENIX Security Symposium*, Washington, DC, August 2001.

[7] The Honeynet project. http://project.honeynet.org, April 2004.

[8] Hyang-Ah Kim and Brad Karp. Autograph: Toward automated, distributed worm signature detection. In *13th USENIX Security Symposium*, San Diego, California, August 2004.

[9] 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.

[10] Kyle Lai. Deloder worm/trojan analysis. http://www.klcconsulting.net/deloder_worm.htm, April 2004.

[11] 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.

[12] Stuart P. Lloyd. Least squares quantization in PCM. *IEEE Transactions on Information Theory*, IT-2:129–137, 1982.

[13] Microsoft security bulletin MS03-007. http://www.microsoft.com/technet/security/bulletin/MS03-007.asp, April 2004.

[14] David Moore, Vern Paxson, Stefan Savage, Colleen Shannon, Stuart Staniford, and Nicholas Weaver. Inside the slammer worm. *IEEE Security and Privacy*, 1(4):33–39, July/August 2003.

[15] David Moore and Colleen Shannon. The spread of the Witty worm. http://www.caida.org/analysis/security/witty/, April 2004.

[16] David Moore, Colleen Shannon, and Jeffrey Brown. Code-red: A case study on the spread and victims of an Internet worm. In *2nd ACM Internet Measurement Workshop (IMW)*, Marseille, France, November 2002.

[17] Peter G. Neumann and Phillip A. Porras. Experience with EMERALD to date. In *1st USENIX Workshop on Intrusion Detection and Network Monitoring*, Santa Clara, California, April 1999.

[18] Vern Paxson. BRO: A system for detecting network intruders in real time. In *7th USENIX Security Symposium*, San Antonio, Texas, January 1998.

[19] Neils Provos. Honeyd: Network rhapsody for you. http://www.citi.umich.edu/u/provos/honeyd, April 2004.

[20] Thomas Ptacek and Timothy Newsham. Insertion, evasion and denial of service: Eluding network intrusion detection. Technical report, Secure Networks, January 1998.

[21] Rain Forest Puppy. A look at Whisker's anti-IDS tactics. http://www.wiretrip.net/rfp/txt/whiskerids.html, April 2004.

[22] Daniel J. Roelker. HTTP IDS evasions revisited. In *DEFCON 11*, Las Vegas, Nevada, August 2003.

[23] Security Focus. Microsoft IIS 5.0 "translate: f" source disclosure vulnerability. http://www.securityfocus.com/bid/1578/discussion/, April 2004.

[24] Umesh Shankar and Vern Paxson. Active mapping: Resisting NIDS evasion without altering traffic. In *IEEE Symposium on Security and Privacy*, Oakland, California, May 2003.

[25] 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.

[26] 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.

[27] Sophos. W32.agobot-bt. http://www.sophos.com/virusinfo/analyses/w32agobotbt.html, April 2004.

[28] Symantec. W32.welchia.worm. http://securityresponse.symantec.com/avcenter/venc/data/w32.welchia.worm.html, April 2003.

[29] Giovanni Vigna and Richard A. Kemmerer. NetSTAT: A network-based intrusion detection system. *Journal of Computer Security*, 7(1):37–71, 1999.

[30] Vinod Yegneswaran, Paul Barford, and Johannes Ullrich. Internet intrusions: Global characteristics and prevalence. In *ACM SIGMETRICS*, San Diego, California, June 2003.

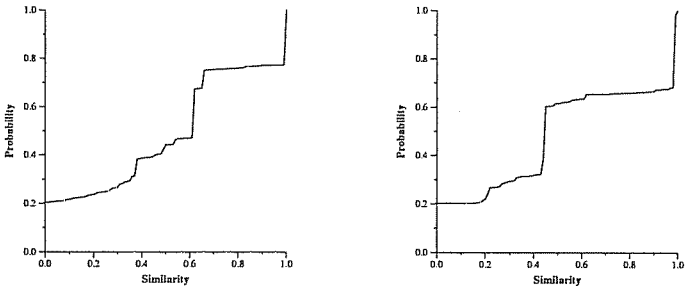

Figure 10: Port 445 edit distance (left) and cosine similarity (right) cumulative distribution functions.

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

A 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 [4] 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.

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
    $m \leftarrow$ DEGREE(x)
    $n \leftarrow$ DEGREE(y)
    $A \leftarrow (m+1) \times (n+1)$ matrix indexed at $[0, 0]$
    for $k \in [0, m]$ **do** $A[k, 0] \leftarrow \sum_{i=1}^{k}$ COST(x_i)
    for $k \in [0, n]$ **do** $A[0, k] \leftarrow \sum_{i=1}^{k}$ COST(y_i)
    for $i = 1$ *to* m **do**
      for $j = 1$ *to* n **do**
        $InsertCost \leftarrow A[i-1, j]$+COST($x_i$)
        $DeleteCost \leftarrow A[i, j-1]$+COST($y_j$)
        $ReplaceCost \leftarrow A[i-1, j-1]$+HIEREDITDIST($x_i$, y_j)
        $A[i, j] \leftarrow$MINIMUM($InsertCost$, $DeleteCost$, $ReplaceCost$)
      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
    $k \leftarrow$ DEGREE(s)
    return $\sum_{i=0}^{k}$ 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 $x_1 \ldots x_i$ to the subvector $y_1 \ldots y_j$. COST calculates the cost to insert or remove a hierarchical vector from an existing hierarchy.

```
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

```
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..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.

B 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 (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 [23].

```
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

```
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)

- **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 first SEARCH request and two more alerts for the second SEARCH request [28].

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.*

C 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)

```
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

worm [27], the Deloder worm [10], 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 [13]. Besides Blaster, scanners connecting to this share include machines infected with variants of the Agobot worm [27]. These machines 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 `SMBTreeConnectAndX`. However, this could also possibly be `SMBtcon` (0x70) or `SessionSetupAndX` (0x73) [5]. We observed several instances of `SessionSetupAndX` in the wild.

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 [10]. 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.

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 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 `svcctl` 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.