The Kerf toolkit for intrusion analysis

Javed Aslam, Sergey Bratus, David Kotz, Ron Peterson, Daniela Rus, and Brett Tofel

Department of Computer Science, and the Institute for Security Technology Studies

Dartmouth College

{jaa, sergey, dfk, rapjr, rus, btofel}@cs.dartmouth.edu

Dartmouth College Computer Science Technical Report TR2004-493

March 31, 2004

Abstract

We consider the problem of *intrusion analysis* and present the *Kerf Toolkit*, whose purpose is to provide an efficient and flexible infrastructure for the analysis of attacks. The Kerf Toolkit includes a mechanism for securely recording host and network logging information for a network of workstations, a domain-specific language for querying this stored data, and an interface for viewing the results of such a query, providing feedback on these results, and generating new queries in an iterative fashion. We describe the architecture of Kerf, present examples to demonstrate the power of our query language, and discuss the performance of our implementation of this system.

1 Introduction

Network-based intrusions have become a significant security concern for system administrators everywhere. Existing intrusion-detection systems (IDS), whether based on signatures or statistics, give too many false positives, miss intrusion incidents, and are difficult to keep current with all known attack signatures. Although high-level correlation tools have recently been developed, and they improve the quality of the alerts given to system administrators [HRTT03], they have a limited success rate, they tend to detect only known types of attacks, and ultimately they result only in an alert message to a human administrator. Human experts are still necessary to analyze the alert (and related data) to determine the nature of the attack. Human experts are also the key tool for identifying, tracking, and disabling new forms of attack. Often this involves experts from many organizations working together to share their observations, hypotheses, and attack signatures. Unfortunately, today these experts have few tools that help them to automate this process.

The goal of the *Kerf* project[1] is to provide an integrated set of tools that aid system administrators in analyzing the nature and extent of an attack and in communicating the results to other administrators or law-enforcement agencies. The premise of Kerf is the recognition that human experts do, and will, play a critical role in the process of identifying, tracking and disabling computer attacks. We also recognize that an important part of the discovery, analysis, and defense against new distributed attacks is the cooperation that occurs between experts across different organizations. Thus, we are building semi-automated tools that help system administrators to (1) identify the characteristics of an attack given data from network and host-based sensors, (2) develop a hypothesis about the nature and origin of the attack, (3) share that hypothesis with security managers from other sites, (4) test that hypothesis at those other sites and coordinate the results of testing, and (5) archive the data necessary for use as evidence. In this report we lay out the vision for the Kerf project, describe the system architecture, and present the prototype of tools supporting this vision.

1.1 Kerf: Vision

Imagine the typical system administrator, responsible for a collection of hosts on one or a few subnets within an organization. Each host logs its activity, using the Unix syslog facility or Windows' Event Logging service. An intrusion detection system (IDS) monitors some or all hosts, and possibly the network, generating and logging alerts about potential attackers. An attack discovered by a system administrator (whether alerted by the IDS, by a security bulletin posted on the web, or as a hunch), must be investigated.

Kerf is intended to assist in this investigation, the *intrusion analysis*, after the attack has been detected. We

© Copyright 2003, Trustees of Dartmouth College. All rights reserved. This project was supported under Award No. 2000-DT-CX-K001 from the Office for Domestic Preparedness, U.S. Department of Homeland Security. Points of view in this document are those of the author(s) and do not necessarily represent the official position of the U.S. Department of Homeland Security.

[1] A "kerf" is the slit made by a saw as it cuts through a log.

assume that correct and complete host and network logs are available, up to a point, because Kerf uses a secure off-host logging facility. The analyst's goal, then, is to reconstruct the evidence of the attack from records of individual events in the available logs.

The analysis process is inherently *interactive*: the sysadmin begins with a vague mental *hypothesis* about what happened, and uses Kerf tools to test and revise that hypothesis. The process is also inherently *iterative*: each new piece of information allows the sysadmin to revise the hypothesis and to explore further. The hypothesis is alternately *refined* as information that partially confirms the hypothesis is discovered, and *expanded* as the sysadmin tries new avenues that broaden the investigation. The result is a specific hypothesis about the source and nature of the attack, with specific evidence to support it.

Our goal is to aid the process of intrusion analysis, by allowing the sysadmin to express, refine, and expand hypotheses, and to collect the evidence to support them. We also provide tools to *archive* the hypothesis and the evidence for later study or presentation to law enforcement. Furthermore, many attacks originate outside the current administrative domain, and some attackers proceed to use local computers to launch further attacks outside the current domain. Thus, tools are needed to *extrapolate* the hypothesis, that is, to produce a hypothesis about what might be seen in the logs at other sites, and to easily communicate that hypothesis to the sysadmins at those other sites. Finally, since the same attack may be underway at other sites, or may occur at another time or at another host within the current site, the sysadmin needs tools to *generalize* the hypothesis into an attack signature.

The current approach to intrusion analysis is shown in Figure 1. Using traditional tools, such as `grep`, `awk`, and `more`, or their equivalent, the sysadmin browses each log file on each host and examines the resulting text output. This approach is difficult, because it requires (1) constructing complex regular expressions or scripts for searching the logs, (2) manually correlating events from different logs or different hosts, and (3) systematically recording actions and the results for later study or action by law enforcement. Because this process is difficult and tedious, most sysadmins can not fully explore and understand an attack or document it for study by others.

The Kerf approach contributes five key components to the process, as shown in Figure 2. We introduce and motivate each piece here, and describe them in more detail in the following section. First, Kerf's logging facility securely records log entries away from the client hosts that may be attacked. Second, the logs are stored in an indexed database for quick and sophisticated retrieval. Third, we designed a query language, called SawQL, for intrusion analysis; it allows the sysadmin to express analysis hypotheses to the Kerf tools. Fourth, the graphical user interface includes data organization modules that help to display the results in a compact, meaningful view. Finally, the *hypothesis engine* helps to automate the process of generating, refining, expanding, extrapolating, and generalizing hypotheses.

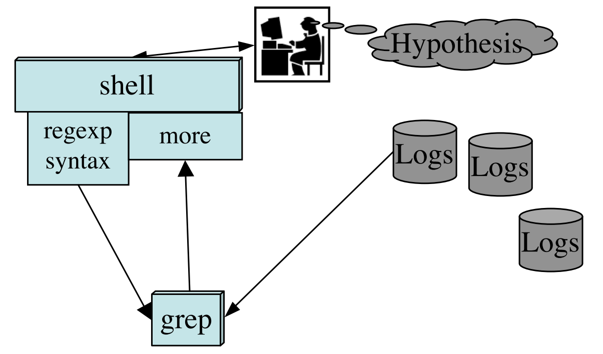


Figure 1: Hypothesis refinement: Today's typical process

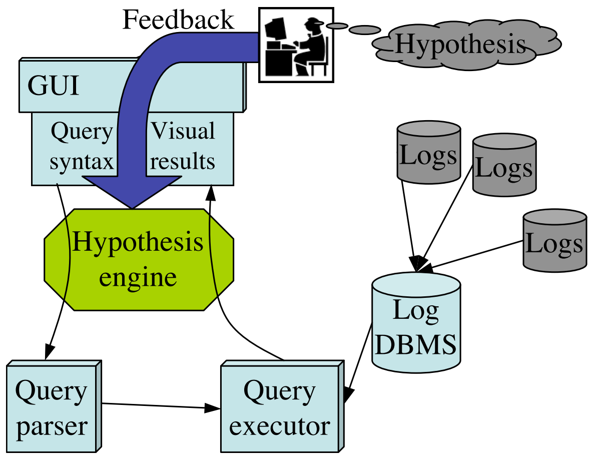


Figure 2: Hypothesis refinement: The Kerf approach

Secure logging. Most hackers who have successfully compromised a system proceed to remove traces of their intrusion from the system's logs. Thus, it is important to securely store host and network logging information off-host. There are many approaches and existing software for secure real-time transfer of log data from a collection of hosts to a secure log server. Kerf can take advantage of any such mechanism. For the purposes of our prototype we implemented a secure logging host that can receive, decode and store logging information from multiple sources.

Our approach is similar to that used by the HoneyNet project [Spi03]. The key difference is that while the HoneyNet system captures more data, using kernel modifications, the Kerf system captures only event logs and needs only a user-level forwarding daemon. Also, obfuscation of both the origin and the character of their logging traffic is essential to the HoneyNet approach, whereas we encrypt and do not disguise our traffic. Kerf's logging host receives encrypted UDP datagrams from its networked clients, but the logging host itself does not have an IP address. Thus, the logging host is relatively secure from conventional Internet attacks. The current implementation accepts Unix syslogs and Windows Event Logs.[2] It is possible to add support for httpd logs, IDS events in IDMEF format, and network logs [MV02], or adapt our logging host to accept log data from other secure remote-logging tools.

Database. Many intrusions involve multiple hosts, and the evidence for many intrusions may be seen in multiple types of logs. To support fast retrieval of relevant records, the logging host stores incoming log records in a database, indexing on important fields (such as host, facility, any IP address mentioned in the record, and any user name mentioned in the record). This approach also serves to isolate the log collection mechanism from the analysis mechanism, and to limit the amount of parsing, indexing, and searching that must be done within our analysis tool. The current implementation uses MySQL.

Domain-specific query language (Section 2.2). Given the database of log records, the analyst could use SQL queries to search for relevant records. SawQL (pronounced SAW-quill) is our extension to SQL designed specifically to express a sysadmin's hypothesis about an attack with maximum flexibility, abstracting the schema and join semantics of the underlying database. SawQL is oriented towards extracting *sequences* of logged event records correlated either temporally or on variables corresponding to common record fields such as hostnames, IP addresses, ports and user names.

By building these features into the language of Kerf we hope to speed the discovery of interesting links in the data and avoid the problems inherent in using traditional tools. These traditional tools offer little help with organizing search results, correlating results, suggesting new queries that organize or refine the data set, and thus make relatively simple analysis tasks error-prone and time consuming.

Data organization and presentation (Sections 2.3–2.4). The centerpiece of the Kerf toolset is the *Landing*[3] application, which provides a graphical interface to the sysadmin. Landing allows the user to enter SawQL queries, displays the results of the queries, and allows the user to provide feedback to the hypothesis engine.

Given the amount of log data collected from an organization's hosts, many queries will retrieve a large number of matching sequences. It is critical to help the sysadmin to organize and visualize these sequences. Our current implementation presents the set of sequences as a set of trees, and uses semantic compression to reduce the matching sequences to a set of patterns that describe those sequences. We intend to explore other approaches.

Hypothesis engine. Given a SawQL query from the sysadmin, Kerf extracts and displays the matching sequences. The GUI allows the sysadmin to mark each sequence "suspicious" or "innocuous", and to indicate the interesting elements of each suspicious sequence. Using any feedback provided (not all sequences need be marked), the engine uses algorithms drawn from the machine-learning community to suggest new queries that better fit the suspicious data, aiding with hypothesis refinement.

This component is under development, and the details are beyond the scope of this report. When complete, the hypothesis engine will also support extrapolation and generalization.

Contributions. The Kerf project makes three specific contributions: the domain-specific query language SawQL, the query engine that supports correlation of records by time and by feature (using variables), and a compression-based approach for data organization. Perhaps the most significant contribution, however, is that Kerf provides an integrated front end and powerful correlation and data-representation tools to aid the analyst, all in one package.

In the remainder of this report we describe the architecture of the Kerf system, present some examples of the use of Kerf for intrusion analysis, discuss the performance of our initial prototype, and compare our work with related research.

2 Architecture

In this section we describe in detail the Kerf modules introduced above.

2.1 Secure Logging

After an attack, the logs on a compromised host can not be trusted. Any modern rootkit modifies some components of a compromised system to exclude logging of intrusion-related events while maintaining logging of normal activities. Some kits also include log editors and

[2] Although we have tested the Windows version of the Kerf transponder and an EventLog to syslog format conversion utility, we have not yet experimented with Windows logs, due to the lack of an appropriate corpus of examples.

[3] A "landing" is the place where logs are gathered for sorting, loading, and redistribution.

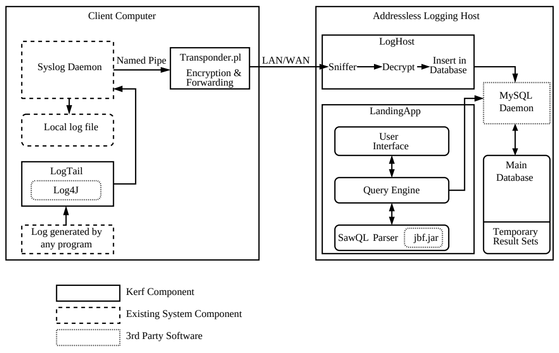


Figure 3: Kerf's secure logging architecture

”cleaners”. Most attention is given to removing traces of the initial penetration, which are the most valuable source of information for intrusion analysis.

For credible intrusion analysis, then, we must preserve the log records of the attacked host, at least until the point when the intruder achieved full control over the host. In this section we describe a remote logging system, in which a sysadmin configures all hosts to log all events directly to a secure central host. We take an approach similar to that used by the HoneyNet project [Spi03], although they capture more data, using kernel modifications, whereas we capture only event logs and need only a user-level forwarding daemon. Also, obfuscation of both the origin and the character of their logging traffic is essential to their approach, whereas we encrypt and do not disguise our traffic.

We configure each monitored client with a logging daemon, as shown in Figure 3. The logging daemon encrypts and forwards each log message as a UDP packet. The logging host sniffs the network for packets, extracts and decrypts the log data, and stores them in its database.

While it is possible to configure many standard logging tools, such as syslog, for remote logging, unencrypted logging traffic would be an invaluable source of information for any attacker capable of sniffing the network. Encrypted traffic is harder to intercept and more difficult to fake. In our Linux implementation, we configure each client's syslogd to forward log records to a named pipe. At the other end of the named pipe is a perl program that reads records from the pipe, encrypts them, and then sends the data onto the local network as UDP packets destined for the logging host. While other systems push log data through secure tunnels, we are unaware of any that encrypt each packet individually.

Although we assign an IP address to the logging host, for use in the UDP packets, the logging host is configured with no IP address. At boot time, the client installs a static ARP entry that maps the bogus IP address to the MAC address of the logging host. When there is a router or switch between the client and the logging host, we insert a manual route so that the log packets get forwarded properly onto the network segment where the logging host resides. We also configure intervening firewalls so that they do not block UDP traffic on port 514 (syslog).

Attacks on the logging host. The central logging host is a natural target for an attack. Our approach is to dedicate a host entirely to the logging task, to run no applications or other services on that host, and to configure the host without an IP address.

An attacker may compromise a client and discover the MAC and IP addresses of the logging host, but these are not useful for any conventional attack on the logging host. Since the logging host does not respond to that IP address, no conventional attack can succeed. The logging host discards any frames that do not contain an encrypted log message. The only service is the logging service, so the only attack is through the logging service.

Since the logging service only accepts encrypted log entries, it is difficult to attack the logic of the logging daemon with carefully crafted messages, or to insert bogus log entries [PN98]. An attacker must first compromise one of the logger's clients, or otherwise obtain the encryption key.

It is possible to overwhelm the logging host by flooding it with frames. With proper configuration of border routers or firewalls, all external traffic destined for the logging host is discarded, so these attacks must come from an internal host. Unless the frames contain a properly encrypted log message, however, they will be quickly discarded. Such attacks are likely to raise alarms on a regular network IDS.

It is also possible to overwhelm the logging host by flooding it with log messages. For example, an attacker could compromise client A and use it to rapidly generate log messages, then use this “smoke screen” to compromise client B. If logging packets from B are dropped by the network or by the logging host, or the disk on the logging host fills, the smoke screen succeeds. While it is not possible to avoid a smoke screen, its effects can be reduced by using a fast logging host with a large disk, or by using multiple logging hosts to distribute the load.

Our secure logging architecture was developed to meet the needs of our prototype implementation, at a time when no pre-packaged secure logging solutions were available. While this architecture was adequate for our testing purposes, an actual deployment of Kerf can be based on a different central logging solution, such as those surveyed in 6.

2.2 The SawQL Language

We designed and implemented a domain-specific query language called *SawQL*. SawQL, an extension of SQL, combines the power of relational data representation with the expressive power of a domain-specific syntax and semantics. In this section we describe the language and its implementation based on a standard SQL engine, and we devote Section 3 to examples of SawQL queries matching diverse attack traces in real system logs.

SawQL provides four critical extensions to SQL.

- SawQL includes keywords to describe common features of log records, such as hostnames, IP addresses, and user names. In our implementation, the logging host parses each incoming log record to extract these fields for the database record, so later queries can quickly extract matching records.
- SawQL provides special syntax to express and retrieve *sequences* of logged events, which in raw SQL would require unwieldy join constructs to describe. This improvement is essential, since most attacks involve a sequence of actions that are visible as a sequence of records in one or more log files. The goal of intrusion analysis is to piece together this sequence of actions.
- SawQL can express connections between records in a sequence, using variable names. The query execution engine correlates log records into sequences with consistent variable bindings. For example, `service 'adduser' AND user %newuser` and `service 'login' AND user %newuser` match two records that refer to the same user.
- SawQL can also express the temporal relationship between records in a sequence; for example, `RELTIME +/- 5 minutes`. In some attacks, the temporal proximity of two events is a critical feature in identifying the attack. This feature also serves to constrain the search.

In contrast, when using traditional tools (such as `grep`) an analyst must take the results of one search, extract interesting elements (such as time, user name, or an IP address), and run new searches on each one of them. Manual use of the command line, or writing ad-hoc scripts, can be error prone, time consuming, and difficult to manage. We expect that significant gains in analysts' productivity will come from alleviating these problems.

Temporal correlation is particularly important for analyzing modern network attacks, in which a sophisticated attacker is likely to conduct reconnaissance, penetration, and control (removal of penetration traces, installation of backdoors, etc.) stages from different hosts. Any hypothesis about such an attack necessarily involves an expression of the temporal proximity of these events and thus temporal correlation of the relevant log records. Moreover, in a distributed system with many components, a certain amount of clock skew is inevitable, and conceptually simultaneous events will have slightly varying timestamps.[4] With SawQL, the user can conveniently mask a known small clock skew in his logs by using the RELTIME clause with a longer time interval to account for the skew.

2.2.1 SawQL syntax overview

SawQL queries. A SawQL *query* is a sequence of one or more *subqueries*. The sequence of subqueries describes the desired sequence of log records; a sequence of records matches the whole query if each record matches the corresponding subquery in the query, and the specified temporal relationships between matching records are satisfied.

Each subquery is an expression, with the following syntax:

- An expression is enclosed in parentheses.
- Parentheses can be used to set the precedence of expression evaluation and to resolve ambiguity.
- Parentheses can be used to nest expressions to any depth.
- Atomic expressions have the form

  (keyword parameter).
- Expressions can be logically combined using AND and OR.

SawQL keywords. Within a subquery, each keyword describes one feature of the log record (such as a hostname or an IP address), and requires one or more parameters. Each parameter can be a word or phrase (in single quotes), a regular expression, a variable name, or a comma-separated list[5] of allowed parameter values (e.g., a list of hostnames, services, PID's, users or IP addresses). Keywords are not case sensitive.

HOSTS: source host of the record

SERVICE: such as ftp, kernel, ssh.

PID: process id.

LOGMSG: Regular expression search for any word or phrase in the text message part of log entries.

IPADDRESS: An IP address (in dotted quads).

ABSTIME: an absolute time range; for example,

```
ABSTIME '2002-12-20 00:00:00'
        '2003-01-05 00:00:00'
```

[4]Under high system load, context switches between logging processes on a single machine can also cause small differences between timestamps of events that we would expect to occur "almost simultaneously."

[5] Lists will be supported in the near future; the OR operator can be used to achieve the same effect currently.

SawQL sequences. In queries with more than one subquery, the RELTIME operator separates subqueries, expressing the required temporal correlation of the records in a sequence. The parameter may be preceded by

- a + to indicate forward in time,
- a – to indicate backward in time, or
- a +/– to indicate backward and forward in time[6].

For example,

```
(HOSTS 'atlantic' AND SERVICE 'ftp')
  RELTIME '+1 day' '-5 hours'
(HOSTS 'pacific' AND SERVICE 'ftp')
```

describes all pairs of log records from the ftp service on host 'atlantic' and on host 'pacific', where atlantic's record occurs up to 1 day prior, or 5 hours after, the record on pacific.

A special variant, ANYTIME,[7] can be used in place of the RELTIME operator to indicate that no temporal correlation is desired.

SawQL variables. Variables may be used in place of any parameter, with a name denoted by percent sign (%). The type of the variable is defined by the keyword that precedes it (e.g., `IPADDRESS %addr`). Variables express correlation between subquery expressions. For example, when looking for all FTP accesses on any hosts that were accessed from the *same* remote IP address,

```
(HOSTS '.*'
      AND  SERVICE 'ftp'
      AND  IPADDRESS %addr)
RELTIME '+/- 3 hours'
(HOSTS '.*'
      AND  SERVICE 'ftp'
      AND  IPADDRESS %addr);
```

2.2.2 SawQL Parser

The Kerf parser converts statements in the SawQL language into a set of SQL queries to run against the database. Kerf uses a Java parser class; the parser source code is generated using Bison and Flex and then run through an interpreter that translates any C code into Java. It takes as input a multiline SawQL query and parses it into a structure we refer to as a "CommandList," in which each Command represents one subquery. The life cycle of a query, from parsing to actual database operations, including correlation, is illustrated in Figure 4, and explained in implementation-level detail below.

A Command contains the following elements:

- String SawQLquery
- String SQLquery
- String SQLqueryWithVariables
- ArrayList variables
- long plusTime
- long minusTime

The first element, SawQLquery, is an exact reconstruction of the original subquery. This provides the user interface with a copy of the original query for display to the user. The second element, SQLquery, is the SawQL subquery parsed into an SQL equivalent with all references to SawQL variables removed. This SQL is used to query the log database before time and variable correlation are performed. The third element, SQLqueryWithVariables, is the SawQL subquery parsed into an SQL equivalent with the SawQL variables left intact. This form preserves precedence order during variable correlation on the query results from SQLquery.

The next element is an array that contains all the variables used in the subquery. For each variable we create a list of all its unique values seen in the query results.

The plusTime and minusTime elements contain the RELTIME part of the subquery, converted into a number of seconds, which is used for time correlation between two subqueries.

Each assembled CommandList is processed by the Kerf query engine. Each Command's SQLquery produces intermediate results that are combined and correlated to obtain the final results of the whole query.

The parser also includes error handling code for exceptions if a parse is incomplete or an error occurs.

2.2.3 SawQL correlation engine

SawQL allows the user to correlate log entries by time (using RELTIME) or by keyword values.

Temporal correlation occurs whenever a query contains multiple subqueries. The RELTIME operator separates subqueries, expressing the maximum time between records in the sequence.

Variable correlation occurs whenever the query contains a variable name in place of a parameter's value. The variable name begins with a percent sign (%), and the type of the variable is defined by the keyword that precedes it (e.g., `IPADDRESS %addr`). Variables express correlation between subquery expressions. For example, when looking for all FTP accesses on any hosts that were accessed from the *same* remote IP address,

```
(HOSTS '.*'
      AND  SERVICE 'ftp'
      AND  IPADDRESS %addr)
RELTIME '+/- 3 hours'
(HOSTS '.*'
      AND  SERVICE 'ftp'
      AND  IPADDRESS %addr);
```

[6]The current parser requires time and date ranges in SQL's verbose format; we plan to extend the parser to translate more readable time ranges (such as "5 minutes") as shown in the examples in this paper.

[7]The current parser accepts RELTIME '*' but will soon accept ANYTIME.

Kerf implements both forms of correlation in the same manner, as shown in the flow chart in Figure 4, ultimately realizing them as *inner joins* on temporary tables holding intermediate results.

Kerf begins the correlation process by executing each SQL statement, and stores the results of each statement as a temporary table in the database. We call these result sets the *intermediate results*, prior to the correlation step. Kerf then traverses the CommandList to determine what time and variable correlations need to be performed. For each needed correlation Kerf executes an SQL statement using an INNER JOIN[8].

For example, a typical variable correlation is performed by the following SQL statement:

```
SELECT DISTINCT
  resultdata4.rserial,resultdata3.rserial
  FROM
  resultdata4 INNER JOIN resultdata3
  USING (variable1) WHERE
  resultdata4.variable1 IS NOT NULL;
```

If the SawQL query implies that the results of a correlation are used later on, intermediate results are stored, substitutions of correlated results are made, and the process of executing SQL joins continues.

When the correlations complete, the results are displayed in the form of a *correlation tree* with extracted log records as nodes, their placement in the tree showing their position in the correlated sequences. The first level of the tree corresponds to initial events of extracted sequences, the second and deeper levels are made up by their respective correlated following events. The display tree and our plans to augment it are discussed in Section 2.3.

2.3 The Landing App

We implemented the Kerf user interface, called *Landing*, as a stand-alone Java/Swing application. Currently Landing allows the following functionality, depicted in Figure 5:

- Entry of SawQL queries.
- Control of execution of the query on the database.
- Display of result sets in tree fashion with branch node labels showing the correlation and leaf nodes showing actual log lines.

[8] This INNER JOIN approach is a refinement of our original approach. Our initial performance testing showed a more than linear growth in search time as the database size increased. Where the SELECT DISTINCT example has a JOIN that returns distinct serial numbers to refer to each line, our original approach returns the actual values for each element in the log line. Some investigations revealed that one source of the slow performance was the DISTINCT keyword, which caused the database to take a long time to calculate unique elements of the message field itself. The INNER JOIN approach, in contrast, just returns the serial number of the log line and then looks up all the elements of the log line. The new approach is faster and this portion of processing time now grows linearly with database size.

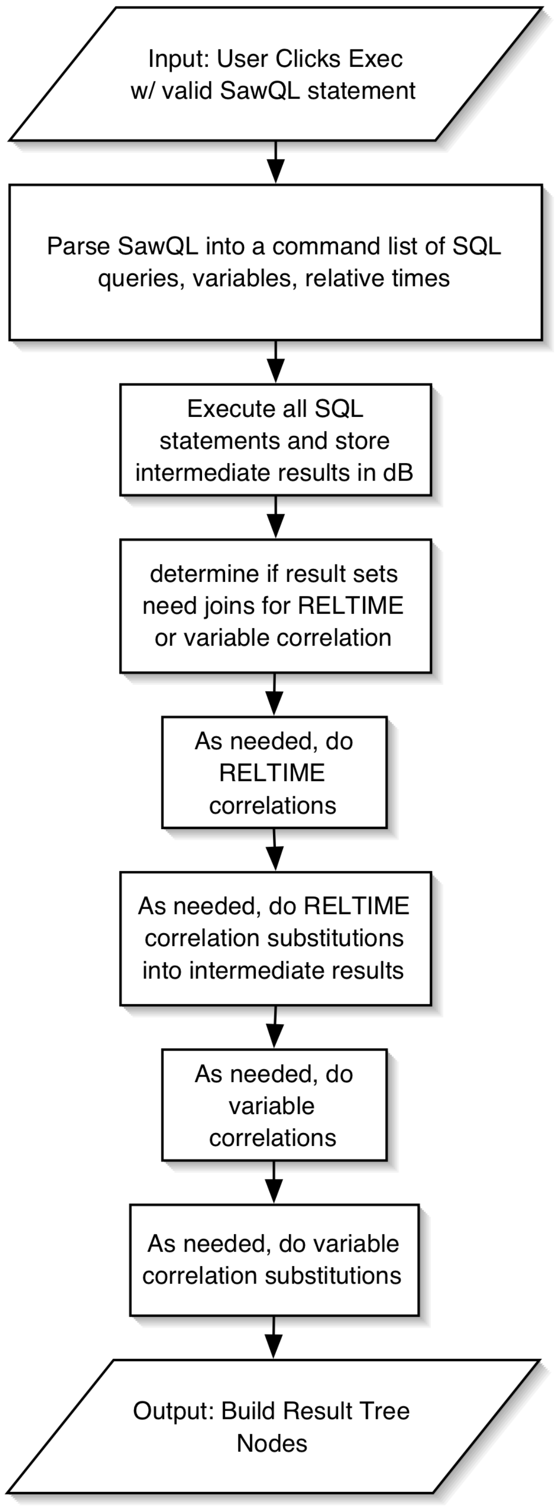
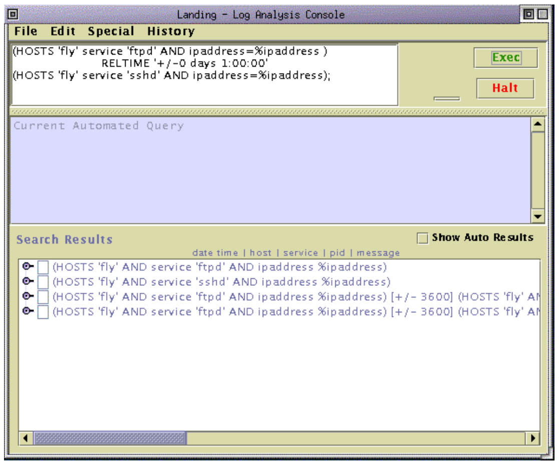

Input: User Clicks Exec w/ valid SawQL statement
Parse SawQL into a command list of SQL queries, variables, relative times
Execute all SQL statements and store intermediate results in dB
determine if result sets need joins for RELTIME or variable correlation
As needed, do RELTIME correlations
As needed, do RELTIME correlation substitutions into intermediate results
As needed, do variable correlations
As needed, do variable correlation substitutions
Output: Build Result Tree Nodes

Figure 4: SawQL query flow

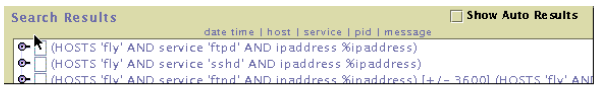
- Multi-threaded operation to allow the [future] automated hypothesis engine to operate independently in the background while the user works.



```
(HOSTS 'fly' service 'ftpd' AND ipaddress=%ipaddress )
                    RELTIME '+/-0 days 1:00:00'
(HOSTS 'fly' service 'sshd' AND ipaddress=%ipaddress);
```

```
date time | host | service | pid | message
(HOSTS 'fly' AND service 'ftpd' AND ipaddress %ipaddress)
(HOSTS 'fly' AND service 'sshd' AND ipaddress %ipaddress)
(HOSTS 'fly' AND service 'ftpd' AND ipaddress %ipaddress) [+/- 3600] (HOSTS 'fly' AN
(HOSTS 'fly' AND service 'ftpd' AND ipaddress %ipaddress) [+/- 3600] (HOSTS 'fly' AN
```

Figure 5: Landing screenshot



```
date time | host | service | pid | message
(HOSTS 'fly' AND service 'ftpd' AND ipaddress %ipaddress)
(HOSTS 'fly' AND service 'sshd' AND ipaddress %ipaddress)
(HOSTS 'fly' AND service 'ftpd' AND ipaddress %ipaddress) [+/- 3600] (HOSTS 'fly' AN
```

Figure 6: Search results in Landing

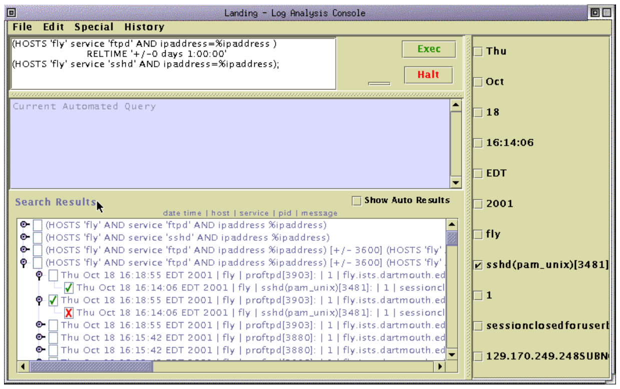
- User feedback input, to help drive the automated hypothesis engine.
- Toggle controls to view the automated query or its results.
- Browser-style history list of previously tried queries.
- Cut, copy, paste of SawQL queries.
- Text output of result set.

When Landing executes queries, it stores all of its intermediate results as tables in the database. The database can join tables quickly, which we use extensively for correlation of results (see Section 2.2.3).

After executing a query, Landing displays the results as a tree, with expansion buttons similar to those in many file browsers (Figure 6). The current implementation of the tree display builds a branch node for each correlation. Nodes are labeled using pieces of the user's original query.

We are currently developing an intelligent mechanism for displaying large result sets with repeating patterns of records, based on semantic compression. The results of compression will help the human user recognize patterns and anomalies in the data, as well as give the hypothesis engine more data to evolve its own queries by letting the user work with larger sets. Our approach is described in Section 2.4.

Within the tree display, we embed the user interface elements needed for relevance feedback on a particular log line. Each tree node can optionally be marked *relevant* (suspicious) or *not relevant* (not suspicious), indicated by a check or an X. If the user marks a line relevant, we follow up by displaying a tokenized list of all the elements of the log line so that they might also indicate what information makes the line relevant. We expect that most result lines would be left in the neutral (unchecked) state in normal usage.

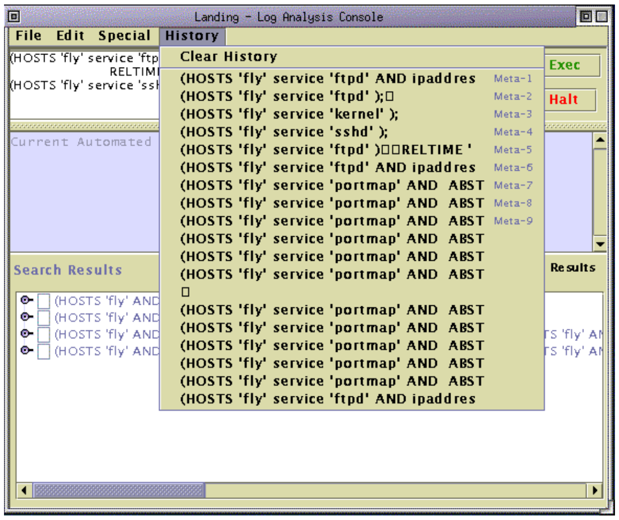


```
(HOSTS 'fly' service 'ftpd' AND ipaddress=%ipaddress )
                    RELTIME '+/-0 days 1:00:00'
(HOSTS 'fly' service 'sshd' AND ipaddress=%ipaddress);
```

```
date time | host | service | pid | message
(HOSTS 'fly' AND service 'ftpd' AND ipaddress %ipaddress)
(HOSTS 'fly' AND service 'sshd' AND ipaddress %ipaddress)
(HOSTS 'fly' AND service 'ftpd' AND ipaddress %ipaddress) [+/- 3600] (HOSTS 'fly'
(HOSTS 'fly' AND service 'ftpd' AND ipaddress %ipaddress) [+/- 3600] (HOSTS 'fly'
    Thu Oct 18 16:18:55 EDT 2001 | fly | proftpd[3903]: | 1 | fly.ists.dartmouth.ed
      ✓ Thu Oct 18 16:14:06 EDT 2001 | fly | sshd(pam_unix)[3481]: | 1 | sessioncl
    ✓ Thu Oct 18 16:18:55 EDT 2001 | fly | proftpd[3903]: | 1 | fly.ists.dartmouth.ed
      ✗ Thu Oct 18 16:14:06 EDT 2001 | fly | sshd(pam_unix)[3481]: | 1 | sessioncl
    Thu Oct 18 16:18:55 EDT 2001 | fly | proftpd[3903]: | 1 | fly.ists.dartmouth.ed
    Thu Oct 18 16:15:42 EDT 2001 | fly | proftpd[3880]: | 1 | fly.ists.dartmouth.ed
    Thu Oct 18 16:15:42 EDT 2001 | fly | proftpd[3880]: | 1 | fly.ists.dartmouth.ed
```

```
☐ Thu
☐ Oct
☐ 18
☐ 16:14:06
☐ EDT
☐ 2001
☐ fly
✓ sshd(pam_unix)[3481]
☐ 1
☐ sessionclosedforuserb
☐ 129.170.249.248SUBN
```

Figure 7: Feedback panel in Landing



```
Clear History
(HOSTS 'fly' service 'ftpd' AND ipaddres      Meta-1
(HOSTS 'fly' service 'ftpd' );□               Meta-2
(HOSTS 'fly' service 'kernel' );              Meta-3
(HOSTS 'fly' service 'sshd' );                Meta-4
(HOSTS 'fly' service 'ftpd' )□□RELTIME '      Meta-5
(HOSTS 'fly' service 'ftpd' AND ipaddres      Meta-6
(HOSTS 'fly' service 'portmap' AND  ABST      Meta-7
(HOSTS 'fly' service 'portmap' AND  ABST      Meta-8
(HOSTS 'fly' service 'portmap' AND  ABST      Meta-9
(HOSTS 'fly' service 'portmap' AND  ABST
(HOSTS 'fly' service 'portmap' AND  ABST
(HOSTS 'fly' service 'portmap' AND  ABST
□
(HOSTS 'fly' service 'portmap' AND  ABST
(HOSTS 'fly' service 'portmap' AND  ABST
(HOSTS 'fly' service 'portmap' AND  ABST
(HOSTS 'fly' service 'portmap' AND  ABST
(HOSTS 'fly' service 'portmap' AND  ABST
(HOSTS 'fly' service 'ftpd' AND ipaddres
```

Figure 8: History in Landing

The feedback is optional, but if the user chooses to give it, it is used as an input for the hypothesis engine. The hypothesis engine is under development.

All previous SawQL queries are stored in the database for future use (Figure 8). In the future, we plan to expand this functionality by adding query templates and grouping queries in the history according to the investigation.

2.4 Data Organization and Presentation

Since the typical data set used with Kerf will be large, some queries may return extremely large sets of matching record sequences. When the result set of a user query is too large, the next step in formulating a more restrictive refined query is via anomalies in the current result set, by cutting away the bulk of “normal” events and their correlates. Spotting such anomalies is hard without additional tools for analyzing distributions of various record parameters across the result set.

To fulfill the need for organizing and managing the large number of records retrieved in response to a query, we developed algorithms to automatically structure the data display, by analyzing distributions of record parameters in a result set. For a large result set, we use a recursive entropy-based algorithm to split the set’s records or record sequences into groups either directly (using values of their fields such as IP address or user group) or indirectly (using features computed from these fields, such as the likely IP segment of origin). This structure is superimposed on the existing Landing “correlation tree.” The resulting tree has non-uniform depth and branching factor, and it reflects the original correlations. The goal of the algorithm is to achieve a low maximum branching factor at each level of the refined tree. The latter should simplify the following tasks, familiar to any analyst faced with a large result set:

- discovering the actual composition of the result set,
- understanding the distribution and ranges of selected field values in event records, finding subsets of anomalous records,
- navigating to the subsets of interest, and
- extracting the subsets of interest for use with another query.

An important side effect of the grouping algorithm is that it is likely to separate the main bulk of results (“normal” events) from the statistically anomalous rest of the distribution, which is where leads for intrusion hypothesis refinement are often found.

The user may add additional levels of grouping, either by choosing from a list of standard features or by defining custom ones, or directly specify how the tree should be rearranged from the top down, bypassing the grouping algorithm. The tree is rebuilt incrementally without re-running the query, by a module separate from the Kerf query engine. Figure 9 illustrates the intended architecture of the viewing modules.

We record all of the user’s grouping and feature choices, and save them as a classification template that can be applied to other result sets, instantiating grouping nodes to refine them for easier handling. This recording is transparent to the user, although he may choose to view the templates and edit them. The template language resembles XSLT and has features of both decision lists and classification trees.

In the following sections we describe user operations on the classification tree representation of the result set

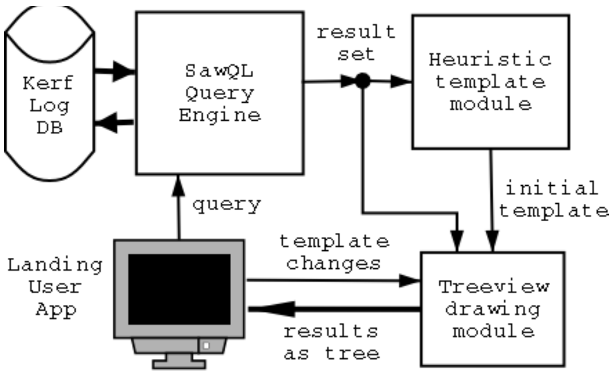


Figure 9: Visualization modules

and the corresponding XML-based template language and its programmatic extensions.

The users of our prototype will notice that the simplest operations on the group nodes of the tree (i.e., subsets of the result set) have effects similar to those of “`grep ... | sort | uniq -c | sort -n`” or “`select distinct ... order by`” statements of shell and SQL environments respectively, but give the user much more flexibility in defining and connecting the filters and in keeping all the records within a common classification framework.

2.4.1 Operations for tree refinement

Starting with the original Landing tree, the user or the splitting algorithm can, for any chosen non-leaf node (including the root)

- split the descendant records of the node into groups based on the distinct values of a feature (a field in the the record, a tuple of fields or a more complicated expression). This replaces the current node with a so-called *distribution node*, which has as many children as there are distinct values of the feature on the current set of descendants, each child node marked with the corresponding value. These child nodes can be sorted by their weight (the total number of event records under them), by their feature value, or by another expression of user’s choice. For viewing convenience, the distribution node itself can be hidden from view.

  These nodes play the role of “`sort | uniq -c | sort -n`” or “`select distinct ... order by`” statements of shell and SQL environments respectively, but give the user a lot more flexibility in defining and connecting her filters together and keeping all the records within a common classification framework.
- set a test expression on the node and rebuild the node,

including as children only those former leaves (event records) that pass the test.

Then redistribute the excluded records among the current node's siblings (which may also have their own tests), and if any records are rejected by all sibling tests, a special default catch-all sibling node is created to collect them as their new parent. Several sibling nodes with tests can be used to build if-then-else structures.

Nodes with tests play the same role as `grep` and `grep -v` filters. We take special care to preserve the complement of any test for inspection and, possibly, further distribution analysis, since failure to inspect the complement of a set of interest (i.e., the records rejected by a filter) is how clues are often lost.

- insert a new grouping node with an arbitrary label as the only child of the current node, and transfer all the children of the current node to the new node. This operation is merely decorative by itself, but is useful when combined with one or both of the above.

These operations can be used to construct complicated trees and reusable tree templates. In particular, the user may choose a library template for routine viewing of her logs, altogether skipping the initial algorithm, if she has a good idea which features are likely to produce the best separation of events into normal and anomalous. Using a fixed template from the start could make generation of a tree somewhat similar to instantiation of an XSLT template, with records one by one traversing the tree built so far and causing new nodes to be instantiated before they reach a leaf position.

2.4.2 A simple example

Here is an example of reducing a flat list of records (from a simple query without correlation, on an actual Unix system log) to a manageable tree. The user was a system administrator concerned with logins from a certain ISP's network and wanted a brief summary of failed and successful logins. A query for login events from `*.isp.net` returned some 600 login records. It turned out that all the logins were from two legitimate users who happened to inhabit distinct dynamic IP ranges, one of whom was prone to typos. The feature pair `(user, host)` was found by the entropy-based organization algorithm to give the best split. The user was thus presented with a 12 line summarization of the 600 line result set. It also became clear that most logins came from one of these users, and his login records were further grouped by month for better presentation (Figure 10).

2.4.3 Template language and tree generation

The templates have three types of nodes: leaves (they become actual log records), list nodes, which serve for classification and grouping, and distribution (or hash[9]) nodes, each with its own feature function. Feature functions compute an integer or string value from one or more record fields. Distribution nodes expand in the view tree into as many nodes as there are distinct values of the feature function[10] on the subset of result set records reaching that node in the process of the tree generating described below.

View tree generation is similar to generation of a result tree from an XSLT template. Starting at a root node (the first one instantiated), each record from the result set traverses the tree built so far, and may cause new nodes to be instantiated. The existing tree thus functions as a dynamically growing decision tree with tests specified by the template nodes. A template node looks as follows:[11]

```
<node name="id"     <!-- ID -->
      test="exp"    <!-- optional -->
      hashkey="exp" <!-- for distribution
                         nodes only -->
      label="exp"
      sortkey="exp" <!-- optional -->
      next="idrefs" <!-- IDREFS -->
/>
```

The meaning of the attributes is as follows. `Name` is a unique ID of a template node. `Test` is an optional expression: if it is present, the template node is instantiated only if this expression evaluates to true. `Hashkey` specifies the classification feature function of a distribution node. `Label` is a required expression which for leaf and list nodes labels the entire node, while for distribution nodes one label (and one node labeled with it) is instantiated per each distinct value of the hashkey. `Sortkey` specifies the order in which child nodes will be sorted under their parent node in the view. The user is expected to often change sorting order while looking at the data. If `sortkey` is absent, records and nodes are sorted on time. Finally, `next` is a comma-separated list of templates to be recursively applied to create child nodes of this node.

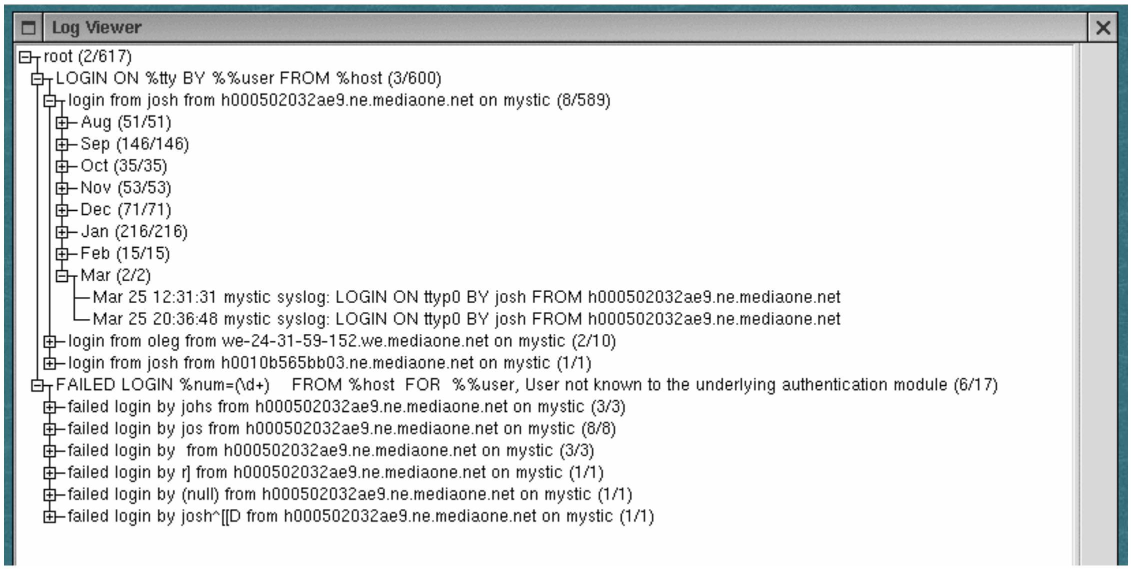
2.4.4 Building trees from templates

A program for the tree builder module is a flat list of nodes, contained in the root node, which only has a `next` attribute, and from which the tree is built. Expressions in attributes marked above with `exp` contains macros for record fields (e.g. %ip or %date). An expression can be evaluated in two ways: by straightforward substitution of

[9]The implementation is based on a hashtable, the value of the node's feature function being used as a hash key.

[10]Most often the feature function is chosen to be the value of one selected field, or an ordered pair of values of two fields, but it can be more domain-specific, such as an application of a fixed netmask to an IP field. Users can define arbitrary feature functions.

[11]Attributes' DTD types are given as comments.



```
Log Viewer
root (2/617)
 LOGIN ON %tty BY %%user FROM %host (3/600)
  login from josh from h000502032ae9.ne.mediaone.net on mystic (8/589)
   Aug (51/51)
   Sep (146/146)
   Oct (35/35)
   Nov (53/53)
   Dec (71/71)
   Jan (216/216)
   Feb (15/15)
   Mar (2/2)
    Mar 25 12:31:31 mystic syslog: LOGIN ON ttyp0 BY josh FROM h000502032ae9.ne.mediaone.net
    Mar 25 20:36:48 mystic syslog: LOGIN ON ttyp0 BY josh FROM h000502032ae9.ne.mediaone.net
  login from oleg from we-24-31-59-152.we.mediaone.net on mystic (2/10)
  login from josh from h0010b565bb03.ne.mediaone.net on mystic (1/1)
 FAILED LOGIN %num=(\d+)    FROM %host  FOR  %%user, User not known to the underlying authentication module (6/17)
  failed login by johs from h000502032ae9.ne.mediaone.net on mystic (3/3)
  failed login by jos from h000502032ae9.ne.mediaone.net on mystic (8/8)
  failed login by  from h000502032ae9.ne.mediaone.net on mystic (3/3)
  failed login by r] from h000502032ae9.ne.mediaone.net on mystic (1/1)
  failed login by (null) from h000502032ae9.ne.mediaone.net on mystic (1/1)
  failed login by josh^[[D from h000502032ae9.ne.mediaone.net on mystic (1/1)
```

Figure 10: Tree view, some nodes expanded

a field value instead of a macro, or by evaluating the result of such substitution according to perl-like syntax (simple string operation and regular expression operations).

The tree is built as follows. The root node is instantiated first, then each log record is tested against the test attributes of its `next` nodes. If the test is passed, a node is instantiated, or, if it already was instantiated by an earlier record, it is entered. If it is a list node, this operation proceeds recursively with its `next` attribute. If it is a leaf node, the label is computed, if present, or the log record itself becomes the leaf label by default, and the leaf is added to the parent node whose `next` attribute was followed to invoke the leaf template. Thus list and leaf nodes with non-empty tests, mutually exclusive for direct children of each node, would represent a classical decision tree. List nodes and non-default leaf nodes are expected to be defined mostly by the user.

The main work is done by distribution templates, that create an a-priory unknown number of child nodes of the template's parent node. The hashkey is evaluated for each record, and a subnode previously created for the hashkey value is entered, or a new subnode is created, if this value was not observed earlier. If the final distribution turns out to be trivial, the resulting nodes are pruned to simplify the tree, and their children are reparented to the node produced by the parent template of the distribution node. This allows for complex stock templates to create simple (or even entirely flat) trees and subtrees when applied to small result sets.

Distribution nodes are expected to be produced both by the initial splitting algorithm (their features chosen based on entropy of the distributions of candidate features), or by the user during Kerf interactions. Since the program to the tree builder is essentially an XML tree internally (although it may be represented more succinctly and appealingly by the GUI, not completely finalized at the time of this writing), its manipulation (insertion of nodes, changing of attributes) is straightforward.

The initial succession of features for the first presentation of the tree is chosen as follows. A set of candidate features is made from the fields in the resulting set of records (e.g. values of single fields, pairs and possibly triples of fields, and special features such as class mask based ones for IPs.) For each candidate feature, the entropy of its distribution is computed during the pass through the records in the set. Then either the non-trivial distribution with the lowest entropy is chosen greedily, or the distribution such that 2^H (where H is its entropy) is closest to the threshold value of lines that we want to simultaneously see on the screen, i.e. the maximum desirable branching factor of a distribution node. The rationale for the latter is that 2^H can be interpreted as a number of distinct values in the distribution where its main weight is concentrated.

3 Examples

In this section we provide several examples that demonstrate the expressive power of the SawQL query language.

3.1 An intrusion step by step

The following example shows how a system administrator can start with a suspicious event and in several steps interactively derive a query that describes the traces of an intrusion and that can be run against logs on other sites or hosts.

Consider the syslog from www.counterpane.com/log-hacked.html. This log corresponds to a real intrusion, posted without analysis. Although this example is very simple, it illustrates how the details observed in the result set lead to the next step ("expansion" of the record set of interest).

1. This syslog contains records of activities by a non-root user with uid 0, a sign of trouble. A log-watching component of an IDS can raise an alert about them. This role could be played on our systems by a periodically scheduled set of SawQL queries that includes the following one.

```
(HOSTS 'www' AND service 'PAM_pwdb' AND   user
     '.*/0' AND NOT user   'root/0');
```

   This query will flag two matching records:

```
Sep 23 17:55:34 www PAM_pwdb[28610]: password for
  (jogja/506) changed by ((null)/0)
Sep 23 18:02:48 www PAM_pwdb[30102]: password for
  (D/507) changed by (jogja/0)
```

2. From these two records, the administrator would immediately notice two suspicious users, jogja and D, and check for their creation and activities, say, with the following query:

```
(HOSTS 'www' AND (user 'D' OR user   'jogja'));
```

   which would return over 50 lines:

```
Sep 23 17:52:38 www useradd[28609]: new user:
  name=jogja, uid=506, gid=10, home=/etc/jogja,
  shell=/bin/bash
Sep 23 17:55:34 www PAM_pwdb[28610]: password for
  (jogja/506) changed by ((null)/0)
Sep 23 17:58:11 www PAM_pwdb[28612]:
  authentication failure; (uid=0) -> jogja for
  login service
Sep 23 17:58:12 www login[28612]: FAILED LOGIN 2
  FROM 202.155.35.132 FOR jogja, Authentication
  failure
Sep 23 17:58:16 www PAM_pwdb[28612]: (login)
  session opened for user jogja by (uid=0)
Sep 23 18:00:05 www login[28632]: FAILED LOGIN 1
  FROM 202.155.35.132 FOR D, User not known to the
  underlying authentication module
Sep 23 18:00:12 www PAM_pwdb[28632]: (login)
  session opened for user jogja by (uid=0)
Sep 23 18:02:32 www adduser[30101]: new user:
  name=D, uid=507, gid=507, home=/home/D,
  shell=/bin/bash
Sep 23 18:02:48 www PAM_pwdb[30102]: password for
  (D/507) changed by (jogja/0)
...
```

3. The administrator can choose to match connections from, say, a class B network corresponding to the IPs in the above log (class C patterns happen to yield no matches):

```
(HOSTS 'www' service 'useradd' AND   LOGMSG
     'new user' AND user   'jogja')
   RELTIME '-1 hour'
(HOSTS 'www' IPADDRESS   '203\.55\..*\..*');
```

   Alternatively, the administrator, having been advised of a recent `ftp` vulnerability, can try `ftp` messages within, say, an hour before the first suspicious user creation:

```
(HOSTS 'www' service 'useradd' AND   LOGMSG
     'new user' AND user   'jogja')
   RELTIME '-1 hour'
(HOSTS 'www' service 'ftpd');
```

   Either one of these queries will find the following records:

```
Sep 23 17:33:20 www ftpd[28594]: FTP LOGIN REFUSED
  (ftp in /etc/ftpusers) FROM 203.55.23.150
  [203.55.23.150], ftp
Sep 23 17:33:47 www ftpd[28595]: FTP LOGIN REFUSED
  (ftp in /etc/ftpusers) FROM 203.55.23.150
  [203.55.23.150], ftp
```

   which likely give the IP address of the machine used in penetration.

   Notice that the administrator may want to pursue one line of queries for a while, and then return to an earlier query and extend it differently. Keeping several branches of investigation open at the same time is an important requirement for the GUI.

   At this point the administrator can already write a query describing the attack:

```
(HOSTS 'www' service 'ftpd' AND LOGMSG
     'FTP LOGIN REFUSED')
   RELTIME '+1 hour'
(HOSTS 'www' service 'useradd' AND   LOGMSG
     'new user' AND USER   %newuser)
   RELTIME '+10 minutes'
(HOSTS 'www' service 'PAM_pwdb' AND   LOGMSG
     'password for .* changed by' AND   USER
     '(null)');
```

4. The administrator might want to check if any users from this machine had connected to other machines while it was compromised, that is starting from the first `ftpd` attack and lasting 3 days.

```
(HOSTS 'www' service 'ftpd' AND LOGMSG
     'FTP LOGIN REFUSED')
   RELTIME '+1 hour'
(HOSTS 'www' service 'useradd' AND   LOGMSG
     'new user' AND USER   %newuser)
   RELTIME '+3 days'
(HOSTS '*' service 'login' AND LOGMSG
     'FROM www');
```

3.2 More examples of attack queries

We have collected a number of real intrusion logs posted on the web or sent to security mailing lists by system administrators, and wrote SawQL queries that matched the traces left by the intruders (see Table 1). Together these queries illustrate most features of the SawQL query language. They are also, in a sense, "intrusion descriptions" that, once formulated, can be applied to other logs on same or sufficiently similar architecture machines across the organization.

4 Performance

We investigated the scalability of Kerf's performance for log-message reception and processing and log line retrieval when doing correlation.

.

Kind of attack	Description	Final SawQL query
URL: http://www.counterpane.com/log-hacked.html		
Attack on `ftpd`	Failed login of one user followed by a login of another, non-root user and addition of the first user	`(HOSTS 'www' service 'PAM_pwdb' AND LOGMSG 'FAILED LOGIN' AND user %newuser) RELTIME '+5 minutes' (HOSTS 'www' service 'PAM_pwdb' AND LOGMSG 'login' AND user %olduser AND NOT user 'root') RELTIME '+5 minutes' (HOSTS 'www' service 'adduser' AND LOGMSG 'new user' AND user %newuser);`
URL: http://project.honeynet.org/challenge/results/submissions/peter/files/messages		
Attack on `get-hostbyname`	`Useradd` shortly after a `get-hostbyname` error	`(HOSTS 'www' service 'rpc.statd' AND LOGMSG 'gethostbyname error for') RELTIME '+1 hour' (HOSTS 'www' service 'useradd' AND LOGMSG 'new (user|group)');`
URL: http://cert.uni-stuttgart.de/archive/bugtraq/2000/05/msg00142.html		
Attack on `sshd`	Failed login instantly followed by a successful login	`(HOSTS 'pigpen' service 'PAM_pwdb' AND LOGMSG 'authentication failure' AND user %someuser) RELTIME '+5 seconds' (HOSTS 'pigpen' service 'PAM_pwdb' AND LOGMSG 'session opened' AND user %someuser);`
URL: http://cert.uni-stuttgart.de/archive/incidents/2000/01/msg00056.html		
Buffer overflow attack on `amd`	`Amd` requested mount instantly followed by root logout and soon followed by a password change	`(HOSTS 'zenith','happy' service 'amd' AND LOGMSG 'amq requested mount') RELTIME '+5 seconds' (HOSTS 'zenith','happy' service 'PAM_pwdb' AND LOGMSG 'session closed' AND user 'root') RELTIME '+10 minutes' (HOSTS 'zenith','happy' service 'PAM_pwdb' AND LOGMSG 'password for .* changed');`
URL: http://cert.uni-stuttgart.de/archive/incidents/2000/04/msg00100.html		
Attack unknown	Opened session and instant `su` to another user, followed by start of `named` and logout	`(HOSTS '192.168.1.254' service 'PAM_pwdb' AND LOGMSG '(login) session opened' AND user %someuser) RELTIME '+5 seconds' (HOSTS '192.168.1.254' service 'PAM_pwdb' AND LOGMSG '(su) session opened' AND user %anotheruser) RELTIME '+5 minutes' (HOSTS '192.168.1.254' service 'named' AND LOGMSG 'starting.') RELTIME '+5 minutes' (HOSTS '192.168.1.254' service 'named' AND LOGMSG 'Ready to answer queries.') RELTIME '+1 minute' (HOSTS '192.168.1.254' service 'PAM_pwdb' AND LOGMSG '(login) session closed' AND user %someuser);`

Table 1: Sample SawQL queries

4.1 Load scalability of log message processing

We measured the performance of the log-collection component of Kerf (which is comprised of the syslog-receiving application and the MySQL database) to discover the maximum amount of message traffic our system can handle. A computing cluster with eleven cluster nodes[12] acted as clients sending syslog data to the log host. The nodes were connected via gigabit Ethernet through a switch to a gateway machine[13] which linked them to our 100Mbps building LAN.

[12] Dual AMD Athlon XP 2000+ (1.67GHz) CPU's, 1GB RAM, 40GB 7200RPM IDE disks, running RedHat Linux 2.4.18-mosix

[13] Dual AMD Athlon CP 2000+ (1.67GHz) CPU's, 4GB RAM, 36GB 15K RPM U160 SCSI disk running RedHat Linux 2.4.18-mosix

URL: http://cert.uni-stuttgart.de/archive/incidents/2000/03/threads.html#00296		

Attack on `telnetd`	Scan of running services followed by a telnet login from the same IP	`(HOSTS '7of9' service 'in.ftpd' AND LOGMSG 'refused connect' AND IPADDRESS %someip)`
		`RELTIME '+/-5 seconds'`
		`(HOSTS '7of9' service 'in.telnetd' AND LOGMSG 'refused connect' AND IPADDRESS %someip)`
		`RELTIME '+/-5 seconds'`
		`(HOSTS '7of9' service 'in.fingerd' AND LOGMSG 'refused connect' AND IPADDRESS %someip)`
		`RELTIME '+/-5 seconds'`
		`(HOSTS '7of9' service 'sshd' AND LOGMSG 'refused connect' AND IPADDRESS %someip)`
		`RELTIME '+5 minutes'`
		`(HOSTS '7of9' service 'in.telnetd' AND LOGMSG 'connect from' AND IPADDRESS %someip)`
		`RELTIME '+1 minute'`
		`(HOSTS '7of9' service 'login' AND LOGMSG 'LOGIN ON' AND IPADDRESS %someip);`

URL: http://www.cnns.net/samples/samples.asp?samples_id=8		

Attack on sshd	Repeated attempts to connect through `ssh` followed by `adduser`	`(HOSTS 'dnscache' service 'sshd' AND LOGMSG 'Connection from' AND IPADDRESS %someip)`
		`RELTIME '+30 seconds'`
		`(HOSTS 'dnscache' service 'sshd' AND LOGMSG 'Connection from' AND IPADDRESS %someip)`
		`RELTIME '+5 minutes'`
		`(HOSTS 'dnscache' service 'adduser' AND LOGMSG 'new user');`

URL: http://terakoya.hp.infoseek.co.jp/linux/secure.html		

Attack unknown	Creation of a user, the user's login and deletion within a short period of time	`(HOSTS 'pc6' service 'adduser' AND LOGMSG 'new user' AND USER %newuser)`
		`RELTIME '+/-5 minutes'`
		`(HOSTS 'pc6' service 'login' AND LOGMSG 'LOGIN ON' AND USER %newuser)`
		`RELTIME '+/-1 hour'`
		`(HOSTS 'pc6' service 'userdel' AND LOGMSG 'delete user' AND USER %newuser);`

URL: http://lists.insecure.org/lists/incidents/2000/Oct/att-0115/01-redbull.generic.notes		

Most likely, attack through `ftpd` buffer overflow	Sniffer run by a (recently password-changed) user	`(HOSTS 'host1' service 'PAM_pwdb' AND LOGMSG 'session opened')`
		`RELTIME '+/-1 minute'`
		`(HOSTS 'host1' service 'kernel' AND LOGMSG 'device .* entered promiscuous mode')`
		`RELTIME '+/-1 minute'`
		`(HOSTS 'host1' service 'kernel' AND LOGMSG 'device .* left promiscuous mode');`

Table 2: Sample SawQL queries, continued

The LAN path to our log host goes through two colocated 100Mbps switches, thus there is a fairly direct route from the cluster to the log host, although the log host does see some of the broadcast traffic on the building LAN. We chose an IBM Netfinity eServer [14] (representative of a small computer typically used as a server in e-commerce) for use as a log host.

We wrote a Java application that reads syslog entries from a file and sends out encrypted log messages via UDP datagrams to the log host at a user selectable rate. We used real logs from a user's personal Linux workstation. The average size of a log message was 74 bytes. We ran the Java application on 1–11 cluster nodes and adjusted the message sending rate until we discovered the maximum message rate the log host could handle. This also led to a better understanding of the system's sensitivity to CPU, network bandwidth, and disk utilization.

Figure 11 shows the average percent idle CPU usage on the log host as a function of message rate. We plot the average CPU usage, computed over the duration of the test and across both CPU's (the two CPU's had almost identical average usage levels in all cases). The plot labeled "baseline" was computed using a starting database size of zero, with database index entries built and written to disk as each message arrived.

At 539,352 messages/hour a plateau was reached in

[14]Dual Intel 1GHz PIII CPU's, 512MB RAM, 18GB 15K RPM Ultra160 SCSI disk running RedHat Linux 2.4.3-6enterprise

the amount of CPU utilization, indicating a performance limit. There was still about 48% idle CPU (for both CPU's) hence the log host was not CPU bound. Measurements of network bandwidth utilization averaged about 80,000 bytes per second, which corresponds to 1/100th of the total network bandwidth available, thus network bandwidth was not the limiting factor.

Figure 12 shows the rate of disk writes in blocks per second. The "baseline" plot reached a peak at 539,352 messages/hour after which a plateau appears. Thus we see that overall log host performance was limited by disk I/O. While this rate corresponds to a relatively low throughput to the disk, a study of the design of MySQL shows that it is likely that disk seek time is the cause of the limitation.

Returning to Figure 11, the "270MB Start Size" plot shows the effect of starting from a larger database, 270MB rather than zero. The peak log message rate that can be supported is reduced from 539,352 message/hour to 179,784 messages/hour. In Figure 12, the "270MB Start Size" plot shows that again, disk I/O is the limiting factor in performance.

The "Index Cached" plot in Figure 11 shows the effect of allowing MySQL to cache new index entries in memory rather than writing them to disk. In practical use, MySQL would flush the index to disk during a period of low activity. We ran this test with a starting database size of 270MB. It shows an improvement in performance from 179,784 messages/hour to 539,352 message/hour when compared to the "270MB Start Size" plot. Again, Figure 12 shows that disk I/O is the reason for the difference. The fact that the 0MB plot with index caching off and the 270MB plot with index caching on are about identical tells us that almost all of our performance loss with a larger database size is due to index writes to disk, which is a typical limiting factor in database performance in many applications.

Thus the current implementation of the Kerf system has a peak performance of around half a million log messages per hour, with an average log message size of 74 bytes, while using very little network bandwidth and a little more than half of the log host's CPU resources. While more tests would be needed to adequately map the performance of a larger range of message sizes and log host PC configurations, it can be seen that with a server of the size and capabilities we have chosen that a fairly large collection of client machines could be supported by a single log host doing log collection.

4.2 SawQL parser performance

We studied the performance of SawQL by measuring the time for the SawQL language parser to parse a mid-sized query. To measure the performance of the SawQL language parser the following query was run through it 3000 times:

```
(HOSTS 'agent1' service 'portmap' OR   ipaddress
     %Address AND user   %person AND ABSTIME
     '07/10/96 4:5 PM, PDT'
     '01/04/02 3:16 PM, PDT')
   RELTIME '+/-1 hour'
(HOSTS 'solitaire' service 'portmap'   AND
     (ipaddress %Address OR user   %person))
   RELTIME '+1 hour' '-3 hours'
(HOSTS 'oddjob' service %Service AND   ipaddress
     %Address AND user   %person)
   RELTIME '-1 hour' '+5 hours'
(HOSTS 'oddjob' service %Service AND   ipaddress
     %Address AND user   %person)
   RELTIME '*'
(HOSTS 'shasta' pid '1234' AND logmsg   'serious'
     AND (user 'fred' OR   ABSTIME
     '07/10/96 4:5 PM'   '01/04/02 3:16 PM'));
```

The average time to parse this query was 3.7 milliseconds, which means that we can examine about 300 queries of similar complexity per second, or more for simpler queries.

This result gives a sense of the user-interface delays the parser will contribute to Kerf's overall performance when we use the parser to measure the complexity of candidate queries suggested by the hypothesis engine, currently under development.

4.2.1 Query scalability

When the Landing application is used to input SawQL, each atomic SawQL statement generates a single SQL query. Each SQL query returns some result set of the actual log lines matching those parameters. When time or variable correlation is included as part of the SawQL statement it is necessary to store these "intermediate result sets" as temporary database tables and perform what is normally an inner join to get the final results.

We set out to explore the aspects of Kerf's database performance that would cause Kerf to become unusable, that is, where an analyst might wait an unreasonable amount of time for what would seem to be a reasonable query to complete.

There are many elements that might cause a given query to complete in an "unreasonable" amount of time. Database size affects the speed of SawQL query resolution in the same manner it would for a standard SQL query as shown in Figure 13.

We studied SawQL queries involving variable correlation. In the variable correlation case, the number of found results that then need to be correlated on is the most important factor in determining how long a query will take to resolve; we call this number the "intermediate result set size." The intermediate result set size is the number of results returned by the previous, non-joining, SQL queries.

For the following form of query

```
(HOSTS 'tahoe' service 'ftpd' AND ipaddress=%ipaddress
AND ABSTIME '11/10/02 04:00 AM, EDT',
'11/30/02 06:33 AM, EDT')
     RELTIME '+/-0 days 1:00:00'
(HOSTS 'tahoe' service sshd' AND ipaddress=%ipaddress
AND ABSTIME '11/10/02 04:00 AM, EDT',
'11/30/02 06:33 AM, EDT');
```

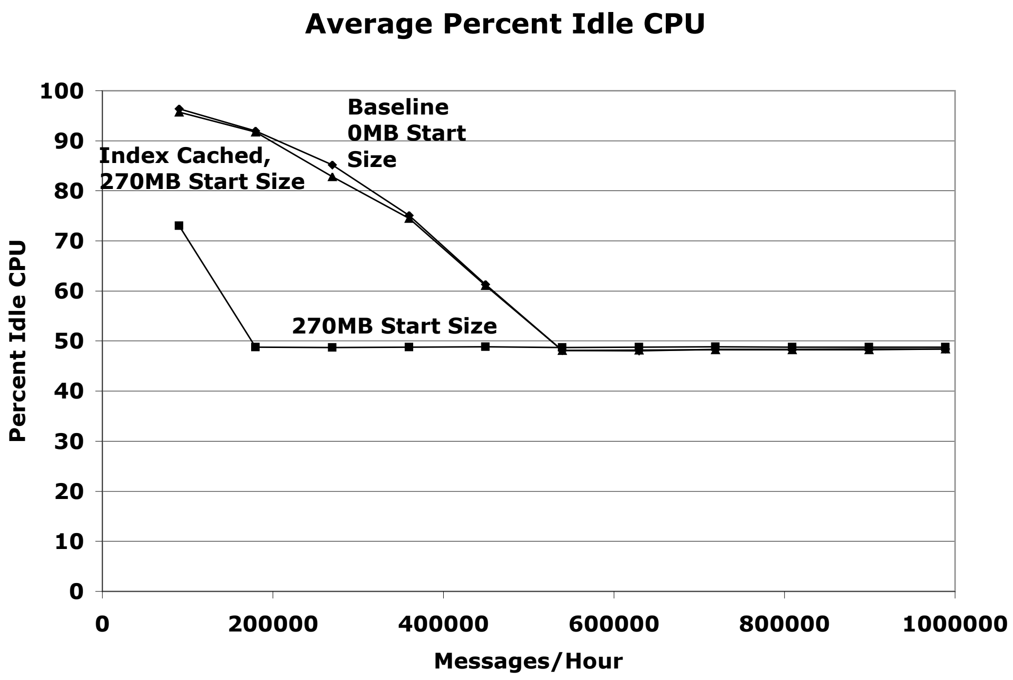


Figure 11: Average percent idle CPU

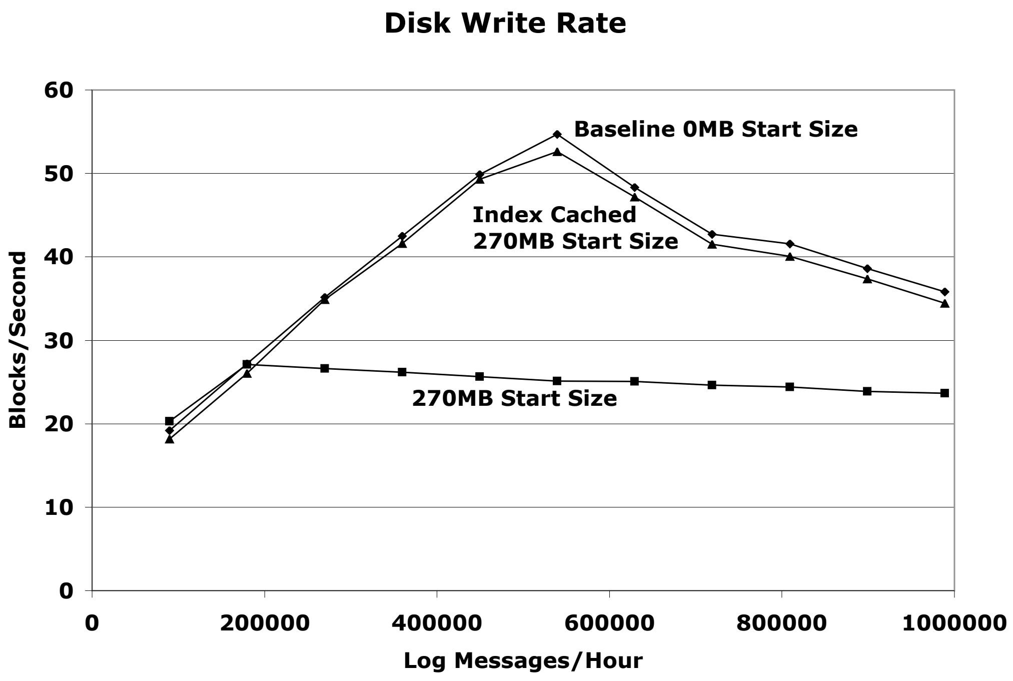


Figure 12: Average disk write rate

Figure 14 plots the total query time as intermediate result set size grows, as well as two key components of the total query time: the time spent correlating records based on one variable, and the time spent correlating the results on time. While the time for a query does grow in a super-linear fashion, we believe it grows in a limited enough fashion to show Kerf being useful on large log databases even when the analyst wants to perform the most demand-

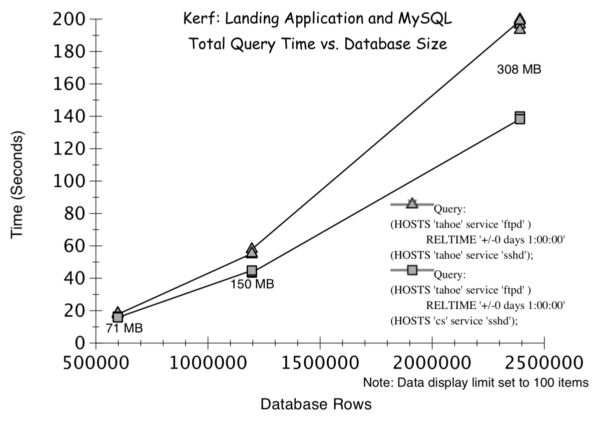


Figure 13: Correlation Time vs. Database Size (size of database marked in megabytes)

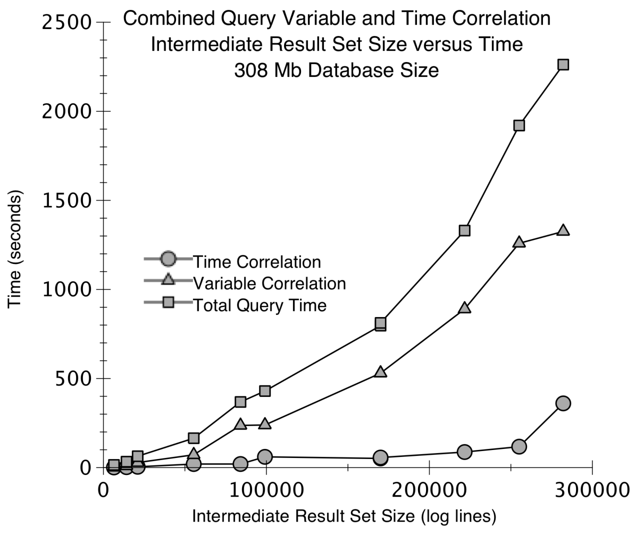


Figure 14: Intermediate Result Set Size vs. Time for Queries Involving Correlation

ing task of variable correlation on items that appear frequently in the database. The spike in the lower curve (time correlation) at the far right of Figure 14 is due to filling real memory and forcing the kernel to swap out to disk. We may alert the user that their query will use more than the available physical memory in future versions of the application.

With tuning, we believe we can reduce the total query time and flatten the curves of the preceding figures. Several improvements are possible in the code that manages the correlation process Figure 15. We also plan to use a newly released feature of MySQL, which allows for subselects. Subselects should allow us to remove the code for managing the storage of intermediate result set tables and reduce much of the associated interprocess communication time. We may also use a "RAIDb" configuration, in

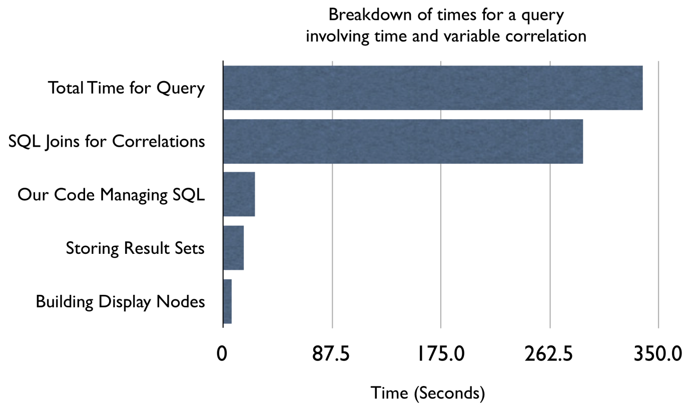


Figure 15: Time Breakdown of a Query Involving Time and Variable Correlation

which the database query can be spread across multiple database engines, hosts, and disks.

Our goal is to support open-ended queries on databases that may contain many matching instances, because an analyst's initial hypothesis (query) may be quite broad. To support our goal we are working to increase the performance of the system and provide more support in the user interface to allow an analyst to pre-judge a query's effectiveness and time to complete.

4.3 Statistics from a deployment

As a part of an ongoing experimental deployment of Kerf, we have installed transponders on 31 Linux desktop machines in our lab, used primarily by the students, a general ssh/login server used by both students and faculty and a firewall–sensor machine protecting a separate subnet and open to the external traffic. The statistics in Table 3 have been observed during a week of normal use.

The highest peak rate of messages from the user machines was observed to be approximately 100 messages per second.

Notice that the firewall–sensor machine is the most active source of messages. The iptables firewall we used was set up with a number of user chains to classify the allowed kinds of traffic, and to produce log messages for packets falling off of these chains, i.e. not matching our intended classification. Thus the high number of messages from the firewall was to be expected. Also, the granularity of these events is much lower, most of these being caused by single packets.

This non-uniformity of message sources raises a question of whether packets from some sources must be given a lower priority as compared to others, or directed to a different logger on a separate network segment, to avoid situations when the attacker may try to hide a penetration attempt by conducting an obviously harmless but noisy scan, eliciting enough messages from the sensor to cause

	Messages/day			Messages/hour			Messages/sec		
Type of host	min	max	avg	min	max	avg	min	max	avg
workstations, combined	625	29165	4656	17	2581	208	1	94	3.5
server	440	4440	2409	28	542	123	1	98	2.3
firewall–sensor	1000	42620	28573	19	6566	1281	1	60	4.7

Table 3: Syslog message rates

a network DOS condition and loss of more valuable log messages from the real target. Limiting the sensor output rate is an obvious remedy, but only a partial one. We are currently measuring the quantitative side of such attacks.

5 Possible vulnerabilities

The Kerf tools are only useful to an analyst if log data can be reliably collected and stored for later use. Thus, our presentation would not be complete without discussing the weaknesses of Kerf. Most of these are characteristic of any remote logging scheme.

There are several possible approaches to attacking a Kerf installation. An attacker can try to prevent log messages from reaching the central logger by creating a DOS condition on the network; in particular, he can try to cause another machine with a Kerf transponder to emit a lot of messages, with which the message relevant to ongoing penetration of the current target will have to contend, with the hope that the important trace message will be dropped (and then deleted from the local log when penetration succeeds). Additionally, once user-level privileges on a Unix machine with standard syslogd are achieved, the attacker can forge messages from any daemon.[15] The above two methods are "deletion" and "insertion" in the IDS terminology, and can be used respectively to deny the analyst vital information for analysis and to confuse him with false intrusion traces.[16] The prospective user of Kerf should be aware of these issues, and take into account the possibility that messages from a compromised machine may not only be masked, but also forged, and watch out for high levels of noise.

Theoretically, the attacker can also try to seize control over the Kerf logger machine, by exploiting some vulnerability in the *libpcap* library (which the logger uses to sniff log messages off the wire), or by exploiting some flaw in Landing's interface to MySQL or in Kerf's message parser. Considering that the logger is IP-less, establishing a control channel would be hard, but not altogether impossible.

[15]Using the logger(1) utility or syslog(3), for example.

[16]For the *Snort* IDS, the *stick* and *snot* tools implement this approach.

6 Related work

6.1 Kerf and Intrusion Detection Systems

Kerf supports incident analysis and recovery, rather than intrusion detection. As a result, our work is complementary to Intrusion Detection Systems (IDSs). Our project uses output of intrusion-detection systems, such as SRI's Emerald[17] and UCSB's STAT[18], in two ways: (1) as a provider of events that will start an analysis; and (2) as data to be used in the analysis process. In many cases, an IDS provides the first alert that spawns a "backward" analysis aimed at identifying the sequence of actions (and associated evidence) that brought the system to the current state. During the incident analysis, IDS alerts will be used as supporting evidence, along with data from host and network logs.

The key difference between Kerf and Intrusion Detection Systems is that an IDS is expected to detect and report attacks in real time, so that an appropriate reaction can be taken by the administrators. An IDS is neither designed for nor expected to recognize all traces of an attack: a timely warning is considered more important than a "full picture". Accordingly, while IDS alerts are very valuable for reconstructing a successful intrusion, an IDS by itself cannot replace analysis tools. While a lot of research went into designing better IDSs (e.g. research systems and prototypes [VK99, Pax98, CCD$^+$99, LS98, Sma88, Lun90, Roe99, CDE$^+$96, SBD$^+$91, PN97], and surveys [ACF$^+$00, BM01, MHL94, Axe00, MM01] and many others), comparatively little effort has been focused on intusion analysis tools.

The IDWG[19] of IETF is standardizing an alert format (IDMEF) and an associated transfer protocol (IDXP). We plan to extend Kerf to understand IDMEF and interoperate with IDXP. Kerf is not closely related to MIT's Lincoln Labs IDS evaluation effort[20]. The goal of the LL project, a three-year effort funded by DARPA, was to provide a means to evaluate IDSs, especially those funded by DARPA. For the Kerf project we may use the test and training data produced as a byproduct of the LL project,

[17]http://www.sdl.sri.com/projects/emerald/

[18]http://www.cs.ucsb.edu/˜rsg/STAT/

[19]http://www.ietf.org/html.charters/idwg-charter.html

[20]http://www.ll.mit.edu/IST/ideval/

to generate test cases for our analysis tools. Although the 1998 and 1999 data will be helpful, the data from the 2000 LL evaluation will be the most valuable because it contains data pertinent to multi-step attacks. _

NIST's Computer Security Resource Center[21] has an intrusion–detection system based on mobile-agent technology. We do plan to use mobile code and possibly mobile agents to aid in distributed data collection, but not intrusion detection. The http://niap.nist.gov/cc-scheme/[22] is another security-tools evaluation and certification project, but to the best of our knowledge they have not developed any tools like ours. The Federal Computer Incident Response Center[23] (FedCIRC), hosted at NIST, suggests tools for intrusion detection and is a forum for reporting attacks. This web site allows limited human-to-human collaboration about attacks, but to our knowledge there is no specific software for forming, communicating, and automatic testing of hypotheses about attacks.

At CERT, the AirCERT[24] project is another attempt to collect alerts from many sites around the country and to organize them into a knowledge base for broader analysis. AirCERT aims to collect intrusion information in real time, and to organize the information for CERT and others to analyze the data. AirCERT focuses on attacks that are known, or at least detectable by existing IDS technology. CERT's Analysis Console for Intrusion Databases (ACID[25]) is a tool to analyze a database of alerts, log data, and packet data. This tool is perhaps closest to our project of any that we have seen, as it is a tool for exploring and analyzing intrusion data, and it can export information to email for informal collaboration. It does not, however, have any capability for hypothesis generation, refinement, or sharing, which are the core of our project.

6.2 Kerf and other research projects

While Kerf's planned functionality in hypothesis generation, hypothesis sharing for human collaboration, and compact data display intersects with that of several other projects that incorporate or are related to intrusion analysis, the focus and methods of these efforts are substantially different. At MITRE, a prototype system for automated diagnosis for computer forensics [ET01] automatically generates attack hypotheses, but it does so based on a detailed up-to-date description of the protected system (not assumed by Kerf) and uses AI abductive reasoning methods. Also, it is not intended to serve as an interactive analysis tool with user feedback. USC's prototype PAID [GFV01] and the analysis components of SRI's EMERALD system also base their hypothesis generation on a different method, Bayesian networks, and are not interactive and iterative. The Cooperative Intrusion Traceback and Response Architecture (CITRA) [SHR$^+$01] (a DARPA-funded effort at Boeing, NAI Labs and UC Davis) aims to develop an infrastructure for sharing intrusion alerts and the results of intrusion analysis between sites (as well as their response policy decisions for coordinated reaction to an attack). However, its focus lies mainly in developing a secure authenticated communication infrastructure and languages for modeling, policy description and implementation across heterogeneous network environments, not in interactive intrusion analysis or hypothesis generation tools. The Willow project[26] (sponsored by DARPA and Rome Laboratory, and developed at the University of Colorado Boulder) aims to develop a similar communication infrastructure and policies for automatic reconfiguration in response to intrusions.

Another related project is titled Plan Recognition in Intrusion Detection Systems, funded under the DARPA Cyber Panel program, located at Honeywell Labs. This work proposes a system using the AI techniques of plan recognition with IDS alert data. The project differs from Kerf significantly in that it tries to actively predict the intentions of an attacker, while Kerf would be used to determine the details of an attack. Some degree of human-supervised correlation is needed for their effort, and in that respect the proposed system would have to make Kerf-style correlations to do its work.

Several other related projects use correlation engines for analysis. The AAFID project [BGFI$^+$98a, SZ00] has a goal similar to ours in the area of log collection and processing. AAFID implements more of the logic at the monitor end. They mention correlation only in passing, indicating that some component of the architecture will do it. The Simple Event Correlator project[27]'s goal is to develop tools for network management, logfile monitoring, security management, and other tasks which involve event correlation. Their correlation engine is different than ours in that it seeks to identify different reactions to events in the stream based on ”state”, defined by observed history within a specified time window. The MACE project[28], the Meta-Alert Correlation Engine, builds upon the CLIPS expert system, creating a distributed application that is capable of automatically filtering and correlating Intrusion Detection alerts into Meta-alerts from multiple sources. Correlation is used in the sense of cross-connecting signatures and vulnerability descriptions from different sources with each other and with the system/network services description. As a direct result, alerts about attacks against absent services can be discarded, while attacks against

[21]http://csrc.nist.gov/focus_areas.html
[22]NIST Common Criteria Evaluation Scheme
[23]http://www.fedcirc.gov/
[24]http://www.cert.org/kb/aircert/
[25]http://www.cert.org/kb/acid/
[26]http://www.cs.colorado.edu/serl/willow/
[27]http://simple-evcorr.sourceforge.net/
[28]http://mace-project.sourceforge.net/

known services can be ranked and highlighted. The AWStats project[29], the Advanced Web Statistics (AWStats) is a free web server logfile analyzer that shows you all your Web (but also FTP or Mail) statistics including visits, unique visitors, pages, hits, hours, search engines, keywords, robots, etc. Correlation is used in the sense of aggregating logged events into sessions and groups.

We also draw from previous work in languages [Gro00]. There are other domain-specific languages for intrusion detection (such as STATL [EVK00, GK02, NCR02, MMDD02]), but none to our knowledge focus on intrusion analysis. Analyst console tools do not usually provide a language that could be used to export and share a description of an attack. Although we believe our SawQL is novel and useful, we are hardly the first to suggest domain-specific languages. For us, SawQL is an integrated part of the hypothesis generation, refinement, and export model, because it provides a convenient, compact way to express a hypothesis as a query.

6.3 Commercial products

The *Internet Scanner* product from Internet Security Systems[30] has *Security Fusion* and *SiteProtector* modules that "provides superior threat prioritization through automated correlation of large amounts of security data," although it is not clear that they use the term "correlation" as we do. Their approach is somewhat similar to Kerf's approach, storing events of all kinds into a database in a common format and then running analyses on the database. The user cannot type in a query, nor can they search for complex sequences of events They can click on an event and get a menu that has questions like: What are the details? What are the target objects of this event? What sensors detected this event? What are the sources of this event? What hosts have this vulnerability? Who attacked this target? What attacks come from this target? What events were against this target? What are the vulnerabilities on this target? There is little information available on their web site, but there is no indication that this product has anything like our hypothesis generation, refinement, extrapolation, or sharing.

There are other indications that industry is beginning to develop tools that help with intrusion analysis. In a recent IDG article[31] there is an interesting line: "Engle declined to identify the software Lehman is using, but vendors of such products include NetForensics[32] Inc. in Edison, N.J., ArcSight Inc.[33] in Sunnyvale, Calif., and Intellitactics Inc.[34] in Bethesda, Md." None of those projects make any mention of technology like our hypothesis generation and refinement, or about the sort of variable correlation supported by Kerf.

[29]http://sourceforge.net/projects/awstats/

[30]http://www.iss.net

[31]http://www.idg.net/ic_1322520_9720_1-5072.html

[32]http://www.netforensics.com

[33]http://www.arcsight.com

6.4 Secure remote logging

We also draw from previous work in remote logging [Rom00, Pre99, SK98]. Although we have no intention of a contribution to the technology of remote logging, any Kerf-like system requires a secure, efficient remote-logging facility. Efforts to protect remote logging mechanisms from a sophisticated attacker have been applied in three different directions: encrypting transmitted information, making the central log host harder to attack by operating it in sniffing-only mode without an IP address, and concealing the very existence of the logging mechanism.

Several free software and commercial tools offer some form of automation for analysis of system logs from multiple sources; Tina Bird compiled an excellent survey of these, together with a collection of links to remote logging tutorials and system-specific information [Bir02]. The practical issues of log processing and database storage have also been discussed in many technical publications [All02, Chu02].

The Unix syslog facility has long had the capability to send log records to other hosts for processing or storage [Rom00], but provided no built-in mechanism for securing this data. Adding security to remote logging proved to be a non-trivial engineering problem in the real world, even with IPSec and SSH available for constructing secure tunnels [Pre99].

A detailed tutorial on using SSL to deliver syslog events to a central host running Snort is provided by Hines [Hin]. This approach requires the logging host to have an IP address, however. We are not the first to suggest the use of an IP-less host for remote logging; indeed, Bauer presented the details at DefCon [Bau02]. Mnemosyne uses a similar technique to send control messages to an IP-less network monitor [MV02]. The Honeynet project combines remote logging, with encrypted packets, with an IP-less gateway host much as in Kerf, but their implementation modifies the Linux kernel, and stresses obfuscation of the very existence of a logging mechanism [Spi03].

6.5 Other sources

There is some work involving the application of visualization techniques to intrusion detection [ES02, She02]. However, these visualization tools do not, as far as we know, support automated generation of intrusion hypotheses, nor provide for sharing them.

Finally, we build on existing work that uses AI techniques for intrusion detection such as [Fra94, BGFI$^+$98b,

[34]http://www.intellitactics.com

SCCC$^+$96]. The key difference between these previous projects and our work is that we do not use machine learning on captured history data to learn about normal system behavior and intrusion anomalies. Instead, we use machine learning for post-hoc data analysis; we aim to automate the process of defining hypotheses for what might have happened, and then to interactively obtain human user feedback to aid in refining those hypotheses.

6.6 Kerf and commonly used toolkits

On one level, the architecture of Kerf reflects the basic necessities of log processing: parsing event records for useful features, and storing these, along with the original form of the record, in such a way that efficient searches are possible. However, Kerf, while making the choices discussed below, adds another level of abstraction to save the analysts' time and effort.

Kerf attempts to automate the process of loading logs in a syslog-like unknown custom format as much as possible. The difficulties of parsing a significant number of log records are often overlooked, whereas in practice this can be, and generally is, an arduous task. Indeed, most log messages come from applications, and follow whatever format discipline can be enforced on their developers (almost none in the case of the standard Unix logging, and only a little better in the Windows world). A parser may be preloaded with the sets of rules for parsing messages from popular applications (e.g., Microsoft's LogParser[35]and various web server reporting tools), but our experience suggests that support for importing logs in a custom format[36] is crucial. To this end, Kerf provides a tool for guessing patterns in batches of syslog-like records, and an interface for user-supplied parser plugins for binary formats, following the example of Snort,[37] Ethereal,[38] and other tools.

While storage solutions other than the relational database have been tried (e.g., ASAX [HCMM92] is optimized to operate on sequential log files, suitably reformatted, in one pass), it is still the simplest choice for implementations, and almost immediately puts the power of SQL at the users' disposal. In this Kerf is not different from various other log processing tools (e.g., ACID[39] and other tools surveyed by Bird [Bir02]).

What separates Kerf from other tools using relational databases and SQL as a back-end is its approach to expressing correlations between events. The main point of its query language design is to allow describing a sequence of events, correlated on time or use of a particular system or network resource, in the most natural and concise form. In other words, SawQL is targeted to describe *processes*, not *signatures*. This is significantly different from the design principles of pattern matching languages targeting intrusion signatures (e.g., Snort and ASAX [HCMM92])[40] and is closer to the approach taken by STATL [EVK00] (which provides more limited support for time correlations).

Additionally, Kerf attempts to provide a single framework for handling log data. With Kerf, an initial investment into defining the features of log records will help to save the effort that is spent on continually re-formatting data during analysis with Unix command line tools. Furthermore, Kerf offers a more flexible environment for statistical analysis[41] of query results than the standard sortable table displays of Ethereal and SQL-based tools (e.g., ACID, LogParser).

7 Summary

Kerf is a tool that aids system administrators in intrusion analysis. We constructed Kerf with the realization that intrusion analysis is inherently an iterative, interactive process. Kerf allows the sysadmin to express her hypothesis about the attack as a query that can be run against the set of logs collected from the network of client hosts, and stored in a central SQL database. We designed the SawQL query language, an extension of SQL, to allow a domain-specific description of sequences of log records, with correlation on time or on key fields. Our current implementation has reasonable performance, but needs tuning.

This paper demonstrates three significant contributions:

- a new domain-specific query language that can express sequences of log records,
- a query engine that can correlate records according to time or common features, using variables, and
- compressed presentation of results for easier visualization.

Perhaps the most significant contribution, however, is that Kerf provides an integrated front end and powerful

[35]http://www.microsoft.com/windows2000/downloads/tools/logparser/

[36]Sorenson in his tutorial http://www.securityfocus.com/infocus/1679 discusses a number of tools that output system information in text format. Although his focus is primarily on obtaining the data in a forensically clean way, we can also think of this data as input to an intrusion analysis tool, for correlation or statistical analysis.

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

[38]http://www.ethereal.com

[39]http://acidlab.sourceforge.net

[40] It is worth noting that pattern matching languages tend to expose their underlying matching mechanisms of doing correlation, a variant of a finite-state machine, whereas SawQL hides the complexity of its own mechanism, leaving space for back-end optimization of resulting multiple joins in the underlying SQL queries.

[41] Burnett gives an excellent tutorial of using statistical heuristics to look for traces of an intrusion in http://www.securityfocus.com/infocus/1712, and Barish also mentions them in http://www.securityfocus.com/infocus/1653,1672. The Kerf data presentation is meant to automate such analysis steps and make them as effortless as possible.

correlation and data-representation tools to aid the analyst, all in one package.

8 Future work

In the near term, we plan to extend our system to handle other kinds of logs, in particular kernel logs of the kind produced by Sun's Solaris BSM[42] and Linux syscall loggers such as Snare[43] or Syscalltrack.[44]

In the long term, a major goal of the Kerf project is to provide semi-automated tools to aid the analyst in hypothesis generation, refinement, archival, generalization, and extrapolation. To this end we are currently developing a *hypothesis engine*. The hypothesis engine consists of three components: (1) a hypothesis generation module, (2) a hypothesis refinement module, and (3) a hypothesis sharing module.

The *hypothesis generation* module assists the user in formulating the initial hypothesis. For example, one possible form of this module would consist of a categorized database of known attack signatures (and their corresponding SawQL queries) which the user could browse to find (or help formulate) an initial query. Given a set of keywords or a small number of suspicious records, the module could also return and rank a collection of candidate initial hypotheses from its database using standard query retrieval techniques. We propose to explore such assistive technologies, though for most "new" attacks, we would expect that the user would probably formulate an initial hypothesis as a SawQL query from the alert data or known attack result.

The *hypothesis refinement* module assists the user in modifying the initial hypothesis to better target suspicious behavior. Given an initial hypothesis, the user can enter the corresponding SawQL query and obtain a set of matching log records from the database. While the initial query might filter the log data to some degree, it is almost certainly the case that the initial hypothesis will need to be refined to better target suspicious behavior and ascertain the nature and extent of the attack. It is the purpose of the hypothesis refinement module to assist in the iterative and interactive refinement of hypotheses. Hypothesis refinement consists of a feedback loop: In each iteration, the user is presented with records matching the current query; these records can be marked by the user as suspicious, normal, or left unmarked; and a learning module then generates a new candidate query (or queries) which can be used in the next iteration. We are currently developing a learning module based on the *minimum description length principle* which is widely used and studied in the machine learning community. In effect, we attempt to find a "succinct" hypothesis (as described by a SawQL query) which describes the given data "well" (i.e., effectively differentiates suspicious and normal data, as defined by the user). Simple hypotheses which effectively describe given data are known to generalize well to unseen (in our case, "unmarked") data, and we expect this phenomenon to hold in our setting as well.

Finally, the *hypothesis sharing* module assists the user in taking the final hypothesis and archiving it for later use, extrapolating it for other specific users and domains, and generalizing it for wider applicability.

We expect our new algorithms and tools for automated hypothesis generation, refinement, and sharing to be a unique contribution to the current state of intrusion-analysis tools.

[42]http://wwws.sun.com/software/security/audit/
[43]http://www.intersectalliance.com/projects/Snare/
[44]http://syscalltrack.sourceforge.net/

Acknowledgments

Marco Cremonini and Andrea Schiavoni designed and implemented an earlier prototype of a similar log-collection and analysis tool. We thank Giovanni Vigna and George Bakos for their helpful discussions, and for help locating host and network log data. We also appreciate the input of other members of the Institute for Security Technology Studies who commented on the current state of the art and needs of intrusion analysis tools.

References

[ACF$^+$00] Julia Allen, Alan Christie, William Fithenand, John McHugh, Jed Pickel, and Ed Stoner. State of the practice of intrusion detection technologies. Technical Report CMU/SEI-99-TR-028, Software Engineering Institute, Carnegie Mellon University, January 2000.

[All02] Jared Allison. Automated log processing. *;login:*, 27(6):17–20, 2002.

[Axe00] Stefan Axelsson. Intrusion detection systems: A survey and taxonomy. Technical Report 99-15, Chalmers Univ., March 2000.

[Bau02] Mick Bauer. Stealthful sniffing, logging, and intrusion detection: Useful and fun things you can do without an IP address. Presentation at DefCon X, August 2002.

[BGFI$^+$98a] Jai Sundar Balasubramaniyan, Jose Omar Garcia-Fernandez, David Isacoff, Eugene Spafford, and Diego Zamboni. An architecture for intrusion detection using autonomous agents. Technical Report CERIAS TR 98/05, COAST Laboratory, Purdue University, June 1998.

[BGFI+98b] Jai Sundar Balasubramaniyan, Jose Omar Garcia-Fernandez, David Isacoff, Eugene Spafford, and Diego Zamboni. An architecture for intrusion detection using autonomous agents. In *Proceedings of the 14th Annual Computer Security Applications Conference*, pages 13–24, December 1998.

[Bir02] Tina Bird. Log analysis resources. www.counterpane.com/log-analysis.html, 2002.

[BM01] Rebecca Bace and Peter Mell. Intrusion detection systems. Technical Report SP 800-31, National Institute of Standards and Technology, August 2001.

[CCD+99] Steven Cheung, Rick Crawford, Mark Dilger, Jeremy Frank, Jim Hoagland, Karl Levitt, Jeff Rowe, Stuart Staniford-Chen, Raymond Yip, and Dan Zerkle. The design of GrIDS: A graph-based intrusion detection system. Technical Report CSE–99–2, Department of Computer Science, University of California at Davis, January 1999.

[CDE+96] Mark Crosbie, Bryn Dole, Todd Ellis, Ivan Krsul, and Eugene Spafford. IDIOT - user guide. Technical Report TR-96-050, Purdue University, West Lafayette, IN, US, September 1996.

[Chu02] Anton Chuvakin. Advanced log processing. online.securityfocus.com/infocus/1613, August 2002.

[ES02] Robert F. Erbacher and Karl Sobylak. Improving intrusion analysis effectiveness. Technical report, University at Albany - SUNY, 2002.

[ET01] Christopher Elsaesser and Michael C. Tanner. Automated diagnosis for computer forensics. Technical report, The MITRE Corporation, August 2001.

[EVK00] S. Eckmann, G. Vigna, and R. Kemmerer. STATL: An attack language for state-based intrusion detection. In *Proceedings of the ACM Workshop on Intrusion Detection Systems*, Athens, Greece, November 2000.

[Fra94] J. Frank. Artificial intelligence and intrusion detection: Current and future directions. In *Proceedings of the 17th National Computer Security Conference*, pages 21–33, October 1994.

[GFV01] V. Gowadia, Csilla Farkas, and Marco Valtorta. Intrusion analysis with soft evidential updates. Technical Report TR2001-0005, University of South Carolina, 2001.

[GK02] Vladimir Gorodetski and Igor Kotenko. Attacks against computer network: Formal grammar-based framework and simulation tool. In *Proceedings of the Fifth International Symposium on Recent Advances in Intrusion Detection (RAID 2002)*, volume 2516 of *Lecture Notes in Computer Science*, pages 219–238. Springer-Verlag, October 2002.

[Gro00] Common Intrusion Detection Framework Working Group. A CISL Tutorial. www.gidos.org/tutorial.html, 2000.

[HCMM92] Naji Habra, Baudouin Le Charlier, Abdelaziz Mounji, and Isabelle Mathieu. Asax: Software architecture and rule-based language for universal audit trail analysis. In *Proceedings of ESORICS'92 European Symposium on Research in Computer Science*. Springer-Verlag, November 1992.

[Hin] Eric Hines. Flying pigs: Snorting next generation secure remote log servers over TCP. http://www.fatelabs.com/flyingpigs.pdf. Version of Wed May 22 10:23:15 EDT 2002.

[HRTT03] Joshua Haines, Dorene Kewley Ryder, Laura Tinnel, and Stephen Taylor. Validation of sensor alert correlators. *IEEE Security & Privacy*, 1(1):46–56, January/February 2003.

[LS98] Wenke Lee and Salvatore J. Stolfo. Data mining approaches for intrusion detection. In *Proceedings of the Seventh USENIX Security Symposium*, pages 79–94. USENIX Association, January 1998.

[Lun90] T. F. Lunt. Ides: An intelligent system for detecting intruders. In *Proceedings of the Symposium: Computer Security, Threat and Countermeasures, Rome, Italy*, 1990.

[MHL94] Biswanath Mukherjee, L. Todd Heberlein, and Karl N. Levitt. Network intrusion detection. *IEEE Network*, pages 26–41, May–June 1994.

[MM01] Ludovic M'e and C'edric Michel. Intrusion detection: A bibliography. Technical

Report SSIR-2001-01, Sup'elec, Rennes, France, September 2001.

[MMDD02] Benjamin Morin, Ludovic Mé, Hervé Debar, and Mireille Ducassé. M2D2: A formal data model for IDS correlation. In *Proceedings of the Fifth International Symposium on Recent Advances in Intrusion Detection (RAID 2002)*, volume 2516 of *Lecture Notes in Computer Science*, pages 115–137. Springer-Verlag, October 2002.

[MV02] Andrew Mitchell and Giovanni Vigna. Mnemosyne: Designing and implementing network short-term memory. In *Proceedings of ICECCS '02*, Greenbelt, MD, December 2002. IEEE Computer Society Press.

[NCR02] Peng Ning, Yun Cui, and Douglas S. Reeves. Analyzing intensive intrusion alerts via correlation. In *Proceedings of the Fifth International Symposium on Recent Advances in Intrusion Detection (RAID 2002)*, volume 2516 of *Lecture Notes in Computer Science*, pages 74–94. Springer-Verlag, October 2002.

[Pax98] Vern Paxson. Bro: A system for detecting network intruders in real-time. In *Proceedings of the Seventh USENIX Security Symposium*, pages 31–52. USENIX Association, January 1998.

[PN97] P. A. Porras and P. G. Neumann. EMERALD: Event monitoring enabling responses to anomalous live disturbances. In *Proc. 20th NIST-NCSC National Information Systems Security Conference*, pages 353–365, October 1997.

[PN98] Thomas H. Ptacek and Timothy N. Newsham. Insertion, evasion, and denial of service: Eluding network intrusion detection. Secure Networks, Inc., January 1998.

[Pre99] Vassilis Prevelakis. A secure station for network monitoring and control. In *Proceedings of the 8th USENIX Security Symposium*, pages 115–122. USENIX Association, August 1999.

[Roe99] M. Roesch. Snort - lightweight intrusion detection for networks. In *13th Administration Conference, LISA'99*, Seattle, WA, November 1999.

[Rom00] Steve Romig. Correlating log file entries. *;login:*, pages 38–44, November 2000.

[SBD+91] Steven R. Snapp, James Brentano, Gihan V. Dias, Terrance L. Goan, L. Todd Heberlein, Che lin Ho, Karl N. Levitt, Biswanath Mukherjee, Stephen E. Smaha, Tim Grance, Daniel M. Teal, and Doug Mansur. DIDS (distributed intrusion detection system) - motivation, architecture, and an early prototype. In *Proceedings of the 14th National Computer Security Conference*, pages 167–176, Washington, DC, October 1991.

[SCCC+96] S. Staniford-Chen, S. Cheung, R. Crawford, M. Dilger, J. Frank, J. Hoagland, K. Levitt, C. Wee, R. Yip, and D. Zerkle. GrIDS—a graph-based intrusion detection system for large networks. In *Proceedings of the 19th National Information Systems Security Conference*, pages 361–370, October 1996.

[She02] Brian K. Sheffler. The design and theory of data visualization tools and techniques, March 2002.

[SHR+01] Dan Schnackenberg, Harley Holliday, Randall, Smith Kelly Djahandari, and Dan Sterne. Cooperative intrusion traceback and response architecture (citra). In *Proceedings of the 2nd DARPA Information Survivability Conference and Exposition (DISCEXII)*. IEEE, June 2001.

[SK98] Bruce Schneier and John Kelsey. Cryptographic support for secure logs on untrusted machines. In *Proceedings of the Seventh USENIX Security Symposium*, pages 53–62. USENIX Association, January 1998.

[Sma88] Stephen Smaha. Haystack: An intrusion detection system. In *Proceedings of the 4th Aerospace Computer Security Applications Conference*, pages 37–44, December 1988.

[Spi03] Lance Spitzner. The Honeynet project: Trapping the hackers. *IEEE Security & Privacy*, 1(2), March/April 2003.

[SZ00] Eugene H. Spafford and Diego Zamboni. Intrusion detection using autonomous agents. *Computer Networks*, 34(4):547–570, October 2000.

[VK99] Giovanni Vigna and Richard A. Kemmerer. NetSTAT: A network-based intrusion detection system. *Journal of Computer Security*, 7(1):37–71, 1999.