Author(s)
Javed A. Aslam, Sergey Bratus, David Kotz, Ron Peterson, Brett Tofel, and Daniela Rus

The Kerf Toolkit for Intrusion Analysis

To aid system administrators with post-attack intrusion analysis, the Kerf toolkit provides an integrated front end and powerful correlation and data-representation tools, all in one package.

JAVED ASLAM
Northeastern University

SERGEY BRATUS, DAVID KOTZ, RON PETERSON, AND BRETT TOFEL
Dartmouth College

DANIELA RUS
Massachusetts Institute of Technology

N etwork-based intrusions have become a significant security concern for system administrators everywhere. Existing intrusion-detection systems (IDSs), whether based on signatures or statistical learning of normal behavior, give too many false positives, miss intrusion incidents, and are difficult to keep current with all known attacks. Although recent high-level correlation tools improve the quality of alerts to system administrators,[1] they have a limited success rate, tend to detect only known attack types, and ultimately generate nothing but alert messages to human administrators. As such, human experts still need to analyze each alert (and related data) to determine the attack's exact nature. Human experts are also the key tool for identifying, tracking, and disabling new attack forms. This work often involves experts from several organizations working together to share their observations, hypotheses, and attack signatures. Unfortunately, few tools help these experts with the process of analyzing log data.

To alleviate this situation, we developed the Kerf toolkit (so-named because a "kerf" is the slit made by a saw as it cuts through a log, and Kerf is our project for processing computer logs.). Its goal is to provide an integrated set of tools that aid system administrators in analyzing the nature and extent of an attack and then communicating the results to other administrators or law enforcement agencies.

An important part of the discovery, analysis, and defense against new distributed attacks is the cooperation that occurs between experts in different organizations. Thus, Kerf contains semiautomated tools that help system administrators identify an attack's characteristics based on data from network and host-based sensors, develop a hypothesis about the attack's nature and origin, share that hypothesis with security managers from other sites, and then test the hypothesis at other sites and coordinate the testing results.

Kerf and intrusion analysis

Picture the typical system administrator, responsible for a collection of hosts on one or more subnets in an organization. Each host logs its activity using the Unix `syslog` facility or the Windows' Event Logging service. An IDS monitors some or all the hosts—possibly even the entire network—and generates and logs alerts about potential attacks. Once a system administrator discovers an attack, he or she must investigate it further.

Kerf is intended to assist in this investigation, called an intrusion analysis, after the attack is detected. We assume that correct and complete host and network logs are available, up to a point, because Kerf forwards encrypted log records to a secure off-host logging server. The analyst's goal is to reconstruct evidence of the attack from individual event records in the available logs.

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

Remote logging in practice

Several free software and commercial tools offer some form of automation for analyzing 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 (see www.loganalysis.org). Others have also discussed the practical issues of log processing and database storage elsewhere.[1,2]

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 logging mechanism's very existence.

Eric Hines provided a detailed tutorial on using SSL to deliver syslog events to a central host running Snort (www.securityfocus.com/guest/3159 and www.securityfocus.com/guest/13283). However, this approach requires the logging host to have an IP address. We're not the first to suggest the use of an IP-less host for remote logging; indeed, Mick Bauer presented the details at DefCon.[3] Andrew Mitchell and Giovanni Vigna's Mnemosyne uses a similar technique to send control messages to an IP-less network monitor.[4] 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 logging mechanism's presence.[5]

References

1. J. Allison. "Automated Log Processing," *;login:*, vol. 27, no. 6, 2002, pp. 17–20.
2. A. Chuvakin, "Advanced Log Processing," 2002, http://online.securityfocus.com/infocus/1613.
3. M. Bauer, "Stealthful Sniffing, Logging, and Intrusion Detection: Useful and Fun Things You Can Do Without an IP Address," presentation at DefCon X, Aug. 2002; http://defconx.wiremonkeys.org.
4. A. Mitchell and G. Vigna, "Mnemosyne: Designing and Implementing Network Short-term Memory," *Proc. IEEE Int'l Conf. Eng. of Complex Computer Systems* (ICECCS 02), IEEE CS Press, 2002, pp. 91–100.
5. L. Spitzner, "The Honeynet Project: Trapping the Hackers," *IEEE Security & Privacy*, vol. 1, no. 2, 2003, pp. 15–23.

Using traditional tools, such as `grep` and `awk` (or their equivalent), the sysadmin browses each host's log file and examines the resulting text output. This approach is difficult for several reasons: it requires the construction of complex regular expressions or scripts for searching the logs, manual correlation of events from different logs or hosts, and systematic recording of actions and results for later study or action. Because this process is difficult and tedious, most sysadmins can't fully explore and understand an attack or document it so that others can study it.

Components

The Kerf approach contributes five key components to the intrusion-analysis process.

Secure logging. After successfully compromising a system, most hackers remove traces of their intrusion from the system's logs. Thus, it is important to securely forward and store logging information off the host. (For more information on remote logging, see the "Remote logging in practice" sidebar.) Many approaches and existing software exist 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.[2] The key difference is that the Kerf system employs only a user-level daemon to forward ordinary syslog events, whereas Honeynet uses kernel modifications to collect and forward more in-depth information, which involves system performance and management costs that are undesirable on production systems. Also, obfuscation of both the origin and character of the logged traffic is essential to the Honeynet Project's approach, whereas we encrypt but don't disguise our traffic. Kerf's logging host receives encrypted User Data Protocol (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 EventLogs (still under development). Users can add support for HTTPd logs, IDS events in intrusion-detection message-exchange format (IDMEF) format, and network logs,[3] or adapt our logging host to accept log data from other secure remote-logging tools.

Database. Many intrusions involve multiple hosts, and evidence of an intrusion might be spread across several logs of different types. 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, and any IP address or username mentioned in the record). This approach also isolates the log-collection mechanism from the analysis mechanism and limits the amount of parsing, indexing, and searching that must be done within our analysis tool. The current implementation uses MySQL.

We've mentioned how Kerf can aid analysts in exam-

ining and extracting data for reporting an incident to law enforcement authorities. Although the specifics are beyond this article's scope, any such tools must be developed carefully if their results are to be admissible as evidence. An analyst can use Kerf to explore the data and then return to the original files to obtain an "original" copy as needed. To assist in this process, the logging host can record the incoming syslog records in flat files as well as the database, storing each record's file offset in the database so that records extracted from the database can be tied back to the flat file as needed.

Domain-specific query language. 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 by abstracting the underlying database's schema and join semantics. SawQL is oriented toward extracting sequences of logged event records correlated either temporally or on variables corresponding to common record fields such as hostnames, IP addresses, ports, and usernames.

By building these features into Kerf's language, we hope to speed up discovery of interesting links in the data and avoid the problems inherent in using traditional tools. Such tools offer little help with organizing search results, correlating results, and suggesting new queries that organize or refine data sets.

Data organization and presentation. The centerpiece of the Kerf toolset is the Landing application, which provides the sysadmin with a graphical interface. Landing lets the user enter SawQL queries, displays their results, and lets the user 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. Our current implementation presents the sequence set as a set of trees and uses semantic compression to reduce the matching sequences to a set of patterns that describe those sequences. We also intend to explore other approaches.

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

This component is under development, and the full details are beyond this article's scope. When complete, the hypothesis engine will also support extrapolation and generalization.

SawQL

SawQL combines the power of relational data representation with the expressive power of a domain-specific syntax and semantics.

SawQL provides four critical extensions to SQL:

- It includes keywords to describe common features of log records, such as hostnames, IP addresses, and usernames. 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.
- It provides special syntax to express and retrieve sequences of correlated logged events, which in raw SQL would require unwieldy join constructs to describe. This improvement is essential because 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.
- It can express connections between records in a sequence, using variable names. The query-execution engine correlates log records into sequences with consistent variable bindings. The two expressions `service 'adduser'` AND `user '%newuser'` and `service 'login'` AND `user '%newuser'`, when used in conjunction in a query, would match pairs of records that referred to the same user.
- It can also express the temporal relationship between records in a sequence; for example, `RELTIME +/- 5 minutes`. The temporal proximity of two events is a critical feature in identifying some attacks. This feature also constrains the search.

When using traditional tools such as `grep`, the analyst must take the results of one search, extract interesting elements (such as time, username, or IP address), and run new searches on each. 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, and so on) 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. With SawQL, the user can conveniently mask a known small clock skew in logs by using the REL-

`TIME` clause with a longer time interval to account for the skew.

SawQL syntax overview

A SawQL query is a sequence of one or more subqueries that 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.

Within a subquery, each keyword describes one feature of the log record (such as a hostname or 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 of allowed values (lists will be supported soon).

The Kerf parser converts statements in SawQL into a set of SQL queries to run against the database.

SawQL correlation engine

SawQL lets the user 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 variable type is defined by the keyword that precedes it (such as `IPADDRESS %addr`). Variables express correlations between subquery expressions. For example, when looking for all FTP accesses on any hosts accessed from the same remote IP address,

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

Kerf implements both forms of correlation in the same manner, as shown in Figure 1's flowchart, ultimately realizing them as inner joins on temporary tables holding intermediate results.

Once the correlations are complete, the results are displayed in the form of a correlation tree with extracted log records as nodes; their placement in the tree shows their positions in the correlated sequences. We delve further into the display tree and our plans to augment it in the "Landing application" section.

SawQL examples

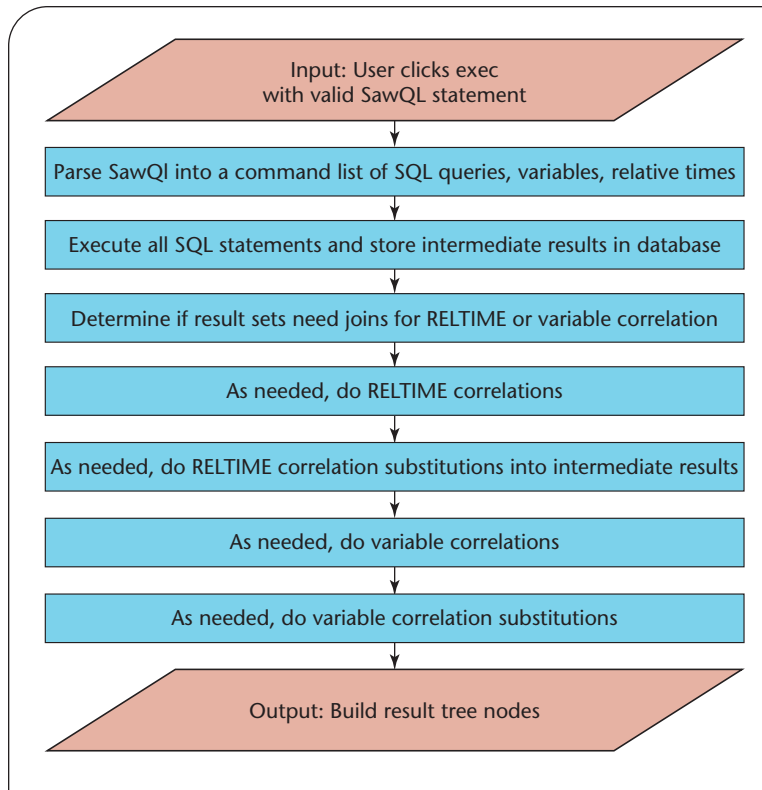The following query examples demonstrate SawQL's expressive power.

Figure 1. SawQL query flow. These are the steps involved in running a SawQL query on the database.

Finding intrusion traces step by step. A system administrator can start with a suspicious event and, in several steps, interactively derive a query that not only describes an intrusion but can be run against logs on other sites or hosts.

Consider the syslog from www.loganalysis.org/sections/signatures/log-hacked.html. This log corresponds to a real intrusion, and was posted without analysis. Although this example is very simple, it illustrates how the details observed in the result set lead to the next step.

First, this syslog contains records of activities by a non-root user with user ID 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:

```
(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]:
```

Related work

We list some work in the field that's related to Kerf.

Remote logging

The Unix syslog facility had long been capable of sending log records to other hosts for processing or storage,[1] but it doesn't provide a built-in mechanism for securing this data. Adding security to remote logging proved to be a nontrivial engineering problem in the real world, even with IPSec and SSH available for constructing secure tunnels.[2]

Languages

Domain-specific languages for intrusion detection (such as STATL, a state transition-based attack description language;[3] Common Intrusion Language, CISL, http://ieeexplore.ieee.org/xpl/abs_free.jsp?arNumber=821507; and the RUSSEL query language of ASAX, (Language for Universal Audit Trail Analysis[4]) as well as correlation of event traces are powerful methods of analysis.[5,6]

These works concentrate on either a systems or theory aspect of the problem. An intrusion specification language, for example, can't usually be directly translated into a query that can be run against a centralized database of log records. Analyst console tools don't usually provide a language that could be used to export and share an attack's descriptions.

Kerf combines the strengths of these approaches, providing at the same time a distributed infrastructure for secure logging, an analyst's GUI with a tree-based presentation of query results, and a domain-specific correlation-oriented language that is both general enough to express and share information about intrusions, and can be efficiently used directly in the console application as a query language for its correlation engine.

Kerf compared to other tools

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

Parsing

Kerf attempts to automate the process of loading logs with a syslog-like format (which is highly variable and not standardized in the message part of the records)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 can be preloaded with the rules sets for parsing messages from popular applications (such as Microsoft's LogParser (www.microsoft.com/windows2000/downloads/tools/logparser/ and various Web server reporting tools), but our experience suggests that support for importing logs in a custom format is crucial. (Sorenson, in his tutorial at www.securityfocus.com/infocus/1679, discusses a number of tools that output system information in text format. Although his focus is primarily

```
      password for (D/507) changed by
      (jogja/0)
```

Next, from these two records, the administrator would immediately notice two suspicious users, jogja and D, and check for their creation and activities 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 203.155.35.132
    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 203.55.35.132
    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)
...
```

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.) To this end, Kerf provides PatternHelper, a tool for guessing patterns in batches of syslog-like records, and an interface for user-supplied parser plug-ins for binary formats, following the example of tools such as Snort (www.snort.org), Ethereal (www.ethereal.com), and others.

Storage

Although many have tried storage solutions other than the relational database (ASAX is optimized to operate on sequential log files, suitably reformatted, in one pass), it's still the simplest choice for implementations, and almost immediately puts the power of SQL at the user's disposal. In this, Kerf is no different from other log-processing tools (such as ACID; http://acidlab.sourceforge.net) and other tools that Tina Bird surveyed (www.loganalysis.org).

Expressing correlation

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 significantly differs from the design principles of pattern-matching languages targeting intrusion signatures (such as Snort and ASAX). It's 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. SawQL is closer to the STATL approach,[3] which provides more limited support for time correlations.

Framework

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

References

1. S. Romig, "Correlating Log File Entries," *;login:*, vol. 25, no. 7, 2000, pp. 38–44; www.usenix.org/publications/login/2000-11/pdfs/log.pdf
2. V. Prevelakis, "A Secure Station for Network Monitoring and Control," *Proc. 8th Usenix Security Symp.*, Usenix Assoc., 1999, pp. 115–122.
3. S.T. Eckmann, G. Vigna and R.A. Kemmerer, "STATL: An Attack Language for State-Based Intrusion Detection," *J. Computer Security*, vol. 10, no. 1, 2002, pp. 71–104; www.cs.ucsb.edu/~vigna/publications.html
4. N. Habra et al., "Asax: Software Architecture and Rule-based Language for Universal Audit Trail Analysis," *Proc. European Symp. Research in Computer Science* (ESORICS 92), Springer-Verlag, 1992, pp. 435–450.
5. P. Ning, Y. Cui, and D.S. Reeves, "Analyzing Intensive Intrusion Alerts via Correlation," *Proc. 5th Int'l Symp. Recent Advances in Intrusion Detection* (RAID 02), LNCS 2516, Springer-Verlag, 2002, pp. 74–94.
6. B. Morin et al., "M2D2: A Formal Data Model for IDS Correlation," *Proc. 5th Int'l Symp. Recent Advances in Intrusion Detection* (RAID'02), LNCS 2516, Springer-Verlag, 2002, pp. 115–137.

The administrator can then choose to match connections from a class B network corresponding to the IPs in the log in step 2 (class C patterns 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 to ftp messages within 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 could pursue one line of queries for a time and then return to an earlier query and extend it differently. Keeping several branches of investigation open simultaneously is an important requirement for the GUI.

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

```
(HOSTS 'www' service 'ftpd' AND LOGMSG
    'FTP LOGIN REFUSED')
    RELTIME '+1 hour'
```

Table 1. Sample SawQL queries.

ATTACK TYPE	DESCRIPTION	FINAL SAWQL QUERY
URL: www.loganalysis.org		

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 `gethostbyname`	`Useradd` shortly after a `gethost byname` 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);`

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');`

```
(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)');
```

An alternative way of analyzing this attack, if the network had also contained a sensor machine running Snort, would involve the administrator being warned about the attack by a Snort alert like this:

```
Sep 23 17:22:00 ids snort: FTP EXPLOIT
    x86 linux overflow [Classification:
    Attempted Administrator Privilege
    Gain] [Priority: 1]: {TCP}
    203.55.23.150:2095 ->
    129.170.213.121:21
```

(129.170.213.121 is our own Web server's address, which we substituted for that of the attacked machine "www.")

In this case, rather than the query in the first step, the analysis' starting point could have been

```
(HOSTS 'ids' service 'snort' AND
    LOGMSG 'FTP EXPLOIT')
RELTIME '+1 hour'
    (HOSTS 'www' service 'login');
```

At this point, the administrator might want to check whether any users from this machine had connected to other machines while it was compromised (starting from the first ftpd attack and lasting three 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');
```

Other examples. Along with this example, we've collected several real intrusion logs that were either 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 of SawQL's features.

Landing application

We implemented the Kerf user interface, called Landing, as a stand-alone Java/Swing application. Currently, Landing enables the following functionality (also depicted in Figure 2):

- entry of SawQL queries;
- control of query execution on the database;
- display of result sets in tree fashion with branch node labels showing the correlation and leaf nodes showing actual log lines;
- automated hypothesis engine operating independently in the background while the user works, thanks to its multithreaded operation;
- 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, and paste of SawQL queries; and
- text output of result set.

After executing a query, Landing displays the results as a tree, with expansion buttons similar to those in many file managers. 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. The tree's first level corresponds to initial events of extracted sequences, and the second and deeper levels are made up by their respective correlated following events.

Within the tree display, we embed user interface elements needed for the user to give feedback about a particular log line's relevance to their current inquiry. As noted earlier, each tree node can optionally be marked relevant (suspicious) or not relevant (not suspicious), as indicated by a check or X. If the user chooses to give feedback, it is used as an input for the hypothesis engine.
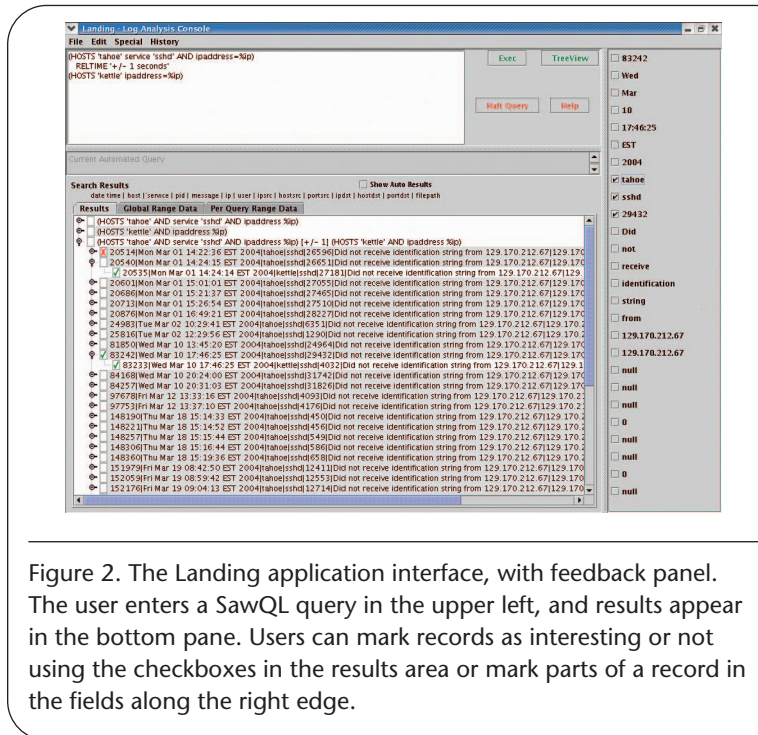

Figure 2. The Landing application interface, with feedback panel. The user enters a SawQL query in the upper left, and results appear in the bottom pane. Users can mark records as interesting or not using the checkboxes in the results area or mark parts of a record in the fields along the right edge.

Data organization and presentation

Because the typical data set used with Kerf is large, we need management tools for organizing the large number of records retrieved in response to queries. One of Kerf's display tools automatically structures 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). As mentioned earlier, this structure is superimposed on the existing Landing correlation tree. The resulting tree has a nonuniform depth and branching factor, and reflects the original correlations. The algorithm's goal is to achieve a low maximum branching factor at each level of the refined tree. This property of the tree should simplify the following tasks, familiar to any analyst faced with a large result set:

- discovering the result set's actual composition,
- understanding the distribution and ranges of selected field values in event records and 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 will likely separate the main bulk of results ("nor-
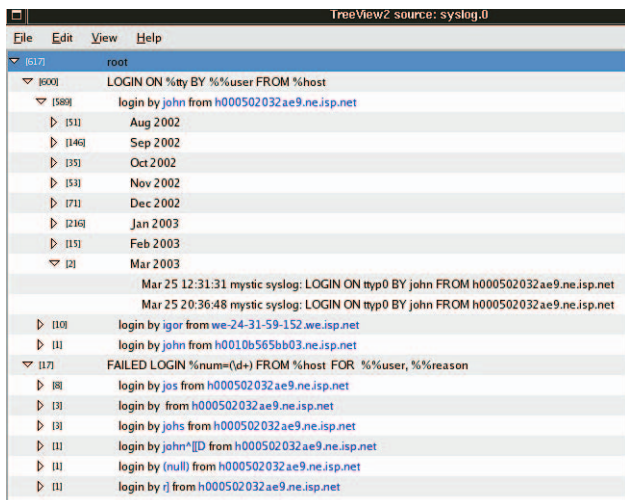
Figure 3. Tree view, some nodes expanded. The Kerf module for adaptive display of query results chose this summarization of the 600+ login events from *.isp.net.

mal" events) from the statistically anomalous rest of the distribution, which is where leads for intrusion-hypothesis refinement are often found.

The user can add additional levels of grouping by choosing from a list of standard features or defining custom ones, or they can directly specify how the tree should be rearranged from the top down, bypassing the grouping algorithm. In either case, the tree is rebuilt without rerunning the query by a module separate from the Kerf query engine.

We record all the user's grouping records 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 or she could choose to view the templates and edit them. The template language resembles XSLT, a language for transforming XML documents into other XML documents, and combines features of decision lists and classification trees.

Kerf users will notice that the simplest operations on the tree's group nodes (that is, subsets of the result set) have effects similar to those of grep `... | sort | uniq -c | sort -n` or `select distinct ... group by ... 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.

A simple example

Let's look at an example of reducing a flat list of records (from a simple query without correlation, on an actual

Unix system log) into a manageable tree. The user is a system administrator concerned with logins from a certain ISP's network and wants a brief summary of failed and successful logins. A query for login events from *.isp.net returns some 600 records. It turned out that all 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 data organization algorithm to produce the best tree-form. 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 (shown in Figure 3).

Performance

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

Load scalability of log message processing

We measured the performance of Kerf's log-collection component (which comprises syslog-receiving application and the MySQL database) to discover the maximum amount of message traffic our system can handle. A computing cluster with 11 cluster nodes acted as clients, sending syslog data to the log host. The nodes were connected via gigabit Ethernet through a switch to a gateway machine, which linked them to our 100-Mbps building LAN.

CPU utilization reached a plateau at 539,352 messages per hour, with an average log message size of 74 bytes, indicating a performance limit. Both CPUs were still about 48 percent idle, proving that 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. Measurements of disk I/O showed write speed to the disk to be the limiting factor in performance. In particular, disk-seek times for database index writes to the disk were the primary limit on disk-write speed, even though there was unused disk-bus capacity.

Although more tests are necessary to adequately map the performance of a larger range of message sizes and log host configurations, it's clear that with a server of the size and capabilities we chose, a single host performing log collection could support a fairly large collection of client machines.

Query scalability

When the Landing application is used to input SawQL,

each atomic SawQL statement generates a single SQL query. Each SQL query returns a result set of the actual log lines matching those parameters. When time or variable correlation is included as part of the SawQL statement, it's 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 studied SawQL queries involving variable correlation. In the variable correlation case, the number of found results to correlate is the most important factor in determining how long a query will take to resolve. This number is the *intermediate result set size*; it is the number of results returned by the previous, non-joining SQL queries.

For the following query form,

```
(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 4 plots the total query time as the intermediate result set size grows, as well 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 superlinear fashion, we believe it grows in a limited enough fashion to show that Kerf would be useful on large log databases—even when the analyst wants to perform the most demanding 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 4 is due to filling real memory and forcing the kernel to swap out to disk. We can alert the user that his or her query will use more than the available physical memory in future versions.

With tuning, we believe we can reduce the total query time and flatten the curves in Figure 4. Several improvements are possible in the code that manages the correlation process (see Figure 5). We also plan to use a newly released feature of MySQL that allows for subselects, which should let us remove the code for managing the storage of intermediate result set tables and reduce much of the associated interprocess communication time. We might also use a "RAIDb" configuration, in 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 might contain many matching instances, because an analyst's initial hypothesis (query) could be quite
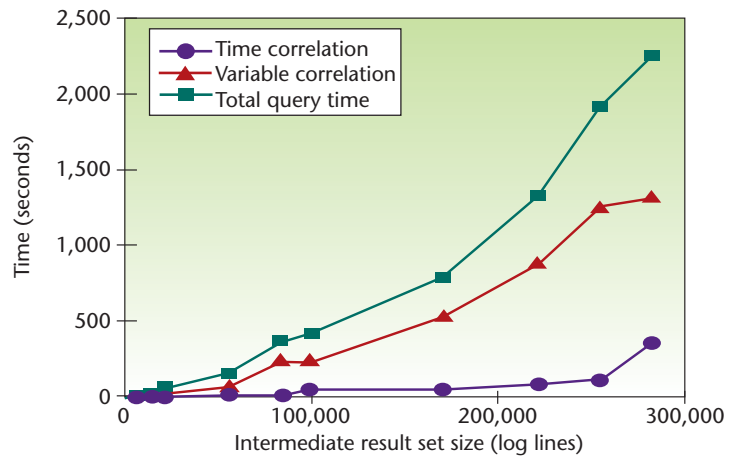

Figure 4. Intermediate result set size versus time for queries involving correlation. The graph shows how query time (and the time-correlation and variable-correlation components of the query time) vary as the size of the results returned by the query varies.
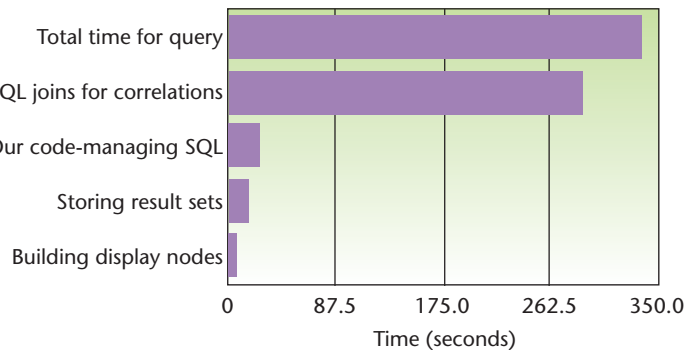

Figure 5. Time breakdown of a query involving time and variable correlation. This graph shows the contributions each step of the query process contributes to the overall query time.

broad. To work toward this goal, we're trying to increase system performance and provide more support in the user interface to let analysts prejudge queries' effectiveness and time to complete.

Possible vulnerabilities

This discussion wouldn't be complete without examining Kerf's weaknesses. Most are characteristic of any remote-logging scheme.

Attackers can use several possible approaches to attack a Kerf installation. They can try to prevent log messages from reaching the central logger by creating a denial-of-service condition on the network; in particular, they can attempt 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. The hope here is that the important trace message will be dropped (and then deleted from the local log when penetration succeeds). Additionally, once attackers achieve user-level privileges on a Unix machine with standard syslogd, they can forge messages from any daemon using the logger utility or syslog. These two methods can be used, respectively, to deny the analyst vital information for analysis and to confuse him or her with false intrusion traces. The prospective Kerf user should be aware of these issues and account for the possibility that messages from a compromised machine might not only be masked, but also forged, and watch out for high levels of noise.

In the near term, we plan to extend our system to handle other kinds of logs—in particular, kernel logs such as those produced by Sun's Solaris Basic Security Module (www.sun.com/software/security/audit/) and Linux syscall loggers such as Snare (www.intersect alliance.com/projects/Snare/) or Syscalltrack (http://syscalltrack.sourceforge.net/).

In the long term, a major goal of the Kerf project is to provide semiautomated tools to aid the analyst in hypothesis generation, refinement, archival, generalization, and extrapolation. To this end, we're developing a hypothesis engine, consisting of a hypothesis-generation module that assists the user in formulating the initial hypothesis; a hypothesis-refinement module, which assists the user in modifying the initial hypothesis to better target suspicious behavior; and a hypothesis-sharing module, which assists the user in taking the final hypothesis and archiving it for later use, extrapolating for other specific users and domains, and generalizing it for wider applicability.

We expect our new algorithms and tools to be a unique contribution to the current state of intrusion-analysis tools. □

Acknowledgments

References

1. J. Haines et al., "Validation of Sensor Alert Correlators," *IEEE Security & Privacy*, vol. 1, no. 1, 2003, pp. 46–56.
2. L. Spitzner, "The Honeynet Project: Trapping the Hackers," *IEEE Security & Privacy*, vol. 1, no. 2, 2003, pp. 15–23.
3. A. Mitchell and G. Vigna, "Mnemosyne: Designing and Implementing Network Short-term Memory," *Proc. IEEE Int'l Conf. Eng. Complex Computer Systems* (ICECCS 02), 2002, IEEE CS Press, pp. 91–100.

Javed Aslam *is an associate professor in the College of Computer and Information Science at Northeastern University. His research interests include machine learning, information retrieval, computer security, and the design and analysis of algorithms. Aslam received a BS in electrical engineering and mathematics from the University of Notre Dame, and a PhD in computer science from MIT. He is a member of the ACM and the IEEE. Contact him at jaa@ccs.neu.edu.*

Sergey Bratus *is a postdoctoral research associate at Dartmouth College's Computer Science Department. His research focuses on applications of machine learning and AI techniques to intrusion analysis. He received his undergraduate education at the Moscow Institute of Physics and Technology and his PhD from Northeastern University. He is a member of Usenix. Contact him at sergey@cs.dartmouth.edu.*

David Kotz *is a professor of computer science at Dartmouth College, director of the Center for Mobile Computing, and executive director of the Institute for Security Technology Studies. His research interests include context-aware mobile computing, pervasive computing, wireless networks, and intrusion detection. He received an AB in computer science and physics from Dartmouth and a PhD in computer science from Duke University. He is a member of the ACM, the IEEE Computer Society, Usenix, and Computer Professionals for Social Responsibility. Contact him at dfk@cs.dartmouth.edu.*

Ronald Peterson *is a senior programmer in Dartmouth College's Computer Science Department, and owner of Peterson Enterprises, which develops PC-based MIDI musical instruments and graphics software. His research interests include cybersecurity, wireless sensor systems, cattle herding, mobile agents, and machine-vision interfaces for novel musical instruments. He received a BA in physics from Lawrence University. Contact him at rapjr@cs.dartmouth.edu.*

Daniela Rus *is an associate professor in the Electrical Engineering and Computer Science department at MIT. Her research interests include distributed robotics, mobile computing, and self-organization. She has a PhD in computer science from Cornell University. She was the recipient of an NSF Career award, is an Alfred P. Sloan Foundation Fellow, and a class of 2002 MacArthur Fellow. Contact her at rus@csail.mit.edu.*

Brett Tofel *is a research associate at Dartmouth College's Institute for Security Technology Studies. His research focuses on the fast analysis of logs from numerous hosts looking for intrusion detection. He has founded and sold a publicly-traded software company; been a senior network engineer for five years for ValleyNet, a nonprofit ISP; and climbed Mount Shasta. He has a BS in engineering from Rensselaer Polytechnic Institute. Contact him at btofel@valley.net.*