Brooery: A Graphical Environment for Analysis of Security-Relevant Network Activity

Christian Kreibich
University of Cambridge Computer Laboratory
15 JJ Thomson Avenue, Cambridge CB3 0FD, UK
christian.kreibich@cl.cam.ac.uk

Abstract

We present the design and implementation of the Brooery, a system for graphical analysis of network activity reported by instances of the Bro intrusion detection system. It supports multiple input streams and provides a web-based graphical user interface to allow the user to analyze the reported activity. The Brooery understands activity at different abstraction levels, allows for quick drill-down searches by focusing on contextuality when moving through the history of events, and provides user-friendly and semantically strong hierarchical filtering to reduce the amount of information presented.

1 Introduction

In recent years, network monitoring has become a widely adopted practice for practically every organization interested in understanding the activity on its networks. Besides other reasons, this is mostly done to improve security: intrusion detection systems (IDSs) are nowadays widely deployed in order to help analysts focus their attention on critical events that otherwise might have been missed in the vast amount of activity.

While these systems have matured a great deal, it has also become clear that the technology is no silver bullet: in the foreseeable future, the human element is going to remain an essential component in the analysis process, largely because our ability to evaluate the relevance of events *in context* of other activity is far superior to the one implemented in present-day technology. This evaluation is rendered more difficult by the fact that the technology currently does an insufficient job at distilling the amount of reported activity into a form whose volume is still comprehensible to humans and at the same time provides all relevant information necessary to understand the reported event in the full context of its occurrence.

We believe that much work remains to be done in helping the network analyst in that task. In this paper we present the *Brooery*,[1] a system to support the analysis of events reported by the Bro IDS [1]. We base our system on Bro because from the outset, Bro has taken a more differentiated approach to the detection problem than other IDSs, by separating *policy* (i.e., what events to report) from *mechanism* (i.e., how to extract the basic building blocks of events from the network). This separation turns out to be crucial in the analysis process, because the difference between relevant events and noise is often entirely defined by a site's policy. By deploying a monitoring policy in line with our understanding of relevance, we can reduce the volume of reported events from the outset. The Brooery presents events to the analyst through a graphical user interface. It allows quick drill-down to relevant details by allowing the analyst to switch between different log types, by the use of contextual navigation techniques, and by employing semantically strong hierarchical filtering that does not require external skills (such as SQL proficiency) from the analyst.

We first recapture Bro's current features in Section 2 to give the reader an intuition of the system the Brooery interfaces with. We present our requirements for the system in Section 3 before describing in detail our resulting architecture along with implementation details in Section 4. We then exemplify the application of our system in Section 5 and review related work in Section 6. The current state of the system and avenues for future work are discussed in Section 7 before we summarize the paper in Section 8.

2 Bro: A Distributed Event-Based Intrusion Detection System

Bro's architecture has remained faithful to the philosophy developed in the original paper [1]. A significant recent improvement has been the introduction of a communications framework as the basis of a more powerful event model suitable for distributed event communication [2, 3]. Figure 1 illustrates Bro's architecture.

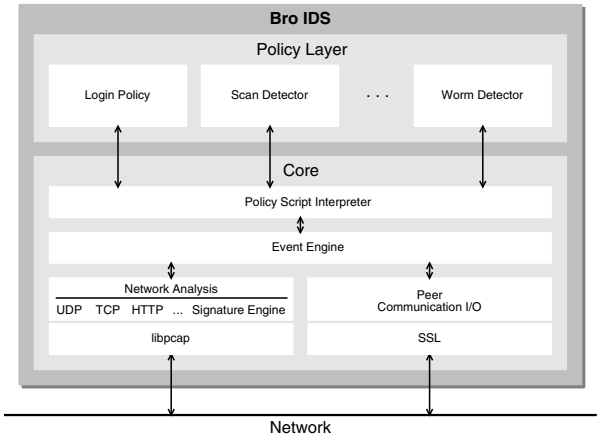

Figure 1: Architecture of the Bro IDS.

2.1 Separation of Mechanism from Policy

A core idea of Bro is to split event detection mechanisms from event processing policies. Event generation is performed by *analyzers* in Bro's core: these analyzers operate continuously and trigger events asynchronously when relevant activity is observed. Examples include the establishment of a new TCP connection, or the request for a URL in an HTTP request. Bro's core contains analyzers for a wide range of network protocols such as TCP, UDP, FTP, HTTP, ICMP, SMTP, RPC, and others. Care is taken to minimize CPU load: only analyzers responsible for triggering the events used at the policy layer are actually enabled. Bro also provides a bidirectional *signature engine* for typical misuse-based intrusion detection: it matches byte string signatures against traffic flows and triggers an event whenever a signature matches [4].

Once an event is triggered, the engine passes it to the *policy layer*. Each Bro peer runs a policy configuration in its policy layer. This policy embodies the site's security policy, expressed in scripts containing statements in the special-purpose Bro scripting language. The language is strongly typed, procedural in style, and provides a wide range of elementary data types to facilitate the analysis of activity on a network. The policy layer maintains a large variety of state information about the activity currently observed on the network. For each event type, one or more *event handlers* are triggered that process events, possibly triggering new ones. Event types are defined by a *name* and a set of typed parameters that characterize individual events.

2.2 Event Logging

Event handlers may decide to *log* an event to persistent storage in a suitable machine- or human-readable format for later analysis. Bro's log management facility currently comprises two stages. First, at the policy level, Bro supports a basic notion of log files. These logs can be opened, closed, and printed to using `printf`-inspired functions. For notices and alarms, separate Bro policies take care of formatting the events appropriately before writing them out. The format of these entries is human-readable, but sufficiently structured for easy parsing. Second, log rotation is configured within the policy layer as well. This takes care of archiving the logs: rotation intervals, monitoring log file size and duration, labelling with start- and end timestamps, and file compression are all taken care of from within the policy layer.

2.3 Communication Framework & State Management

Bro's communication framework supports the serialization and transmission of arbitrary kinds of state between Bro instances. The driving idea behind its design is to allow the realization of *independent state* [2]: we should no longer think of state accumulated at the policy layer as a local concept, but rather as information dispersed and stored throughout the network. The communication model imposes no hierarchical structure. Examples of exchangeable state include triggered events, state kept in policy data structures, and the policy definitions themselves. For the purpose of this paper it is suf-

ficient to think of the entities exchanged between peers as events, though that ignores a large part of its flexibility.

To interface other applications with Bro, we have implemented a lightweight, highly portable library supporting Bro's communication protocol called *Broccoli*,[2] that allows nodes which are not instances of the Bro IDS to partake in its event communication. Broccoli nodes can request, send, and receive Bro events just like Bro itself, but cannot be configured using Bro's policy language. A Broccoli node's policy has to be implemented in the client's code or through mechanisms such as configuration files.

3 The Brooery's Requirements

3.1 Usability Requirements

We have identified the following set of requirements for our system:

- Interoperability with Bro: The system should not require significant changes to Bro itself. Existing communication mechanisms should be leveraged as much as possible.
- Focus on investigation: The primary goal of the system is to enable the analysis of log archive content using a graphical interface, not to provide a real-time alert notification system. A highly interactive user interface, while clearly desirable, is thus not a primary requirement.
- Experimental prototyping: Support for rapid prototyping and experimentation with visualization techniques is more important at this stage than performance optimizations and long-term maintainability.
- Flexible filtering: The predominant problem in the analysis of network activity is the total volume of information. For this reason, effective filtering is essential. The analysis of Bro's log files has so far mostly happened at the shell prompt, and the effectiveness was essentially defined by the analyst's command of the typical text processing toolset: grep, awk, sed, and Perl, just to name few. Skilled use of these tools, while often unintuitive to other analysts, can be quite effective. The system should therefore aim at supporting this mindset in its filter management.
- Contextuality: The richness and diversity of events in Bro requires great flexibility from an analysis environment. The visual navigation should naturally guide the user at all times depending on the context of the currently inspected events, and provide mechanisms for quick drill-down to allow the analyst to focus on the relevant activity.
- Easy accessibility: At present, Bro is in day-to-day use throughout several large organizations around the planet. Such deployments require analyst access from multiple locations and using different platforms. We therefore prefer a standardized and widely available rendering mechanism that requires as little preconfiguration on the analyst's machine as possible.
- Spatial source independence: We would like to be able to select individual Bro nodes as data sources because we do not want to require that all data logging happen at a single place in the network.
- Representational source independence: Bro has traditionally created a set of text-based log files in order to record events for long-term storage. Two other forms of data storage are standard database back-ends and live state contained in running Bro nodes. We would like the system to support access to these uniformly.
- Different user sophistication levels: While Bro's design allows for much flexibility in its configuration, this freedom also means that its users need to spend more time to familiarize themselves with the system before they can use it efficiently and effectively. The user interface should support users of a wide range of sophistication levels, ranging from the occasional log inspection to operators who are intimately familiar with Bro policy development and day-to-day Bro maintenance.

3.2 Threat Model

The Brooery's main purpose is to present highly security-relevant information to the analyst, whose conclusions may have severe consequences for the operation of the network. We therefore need to be

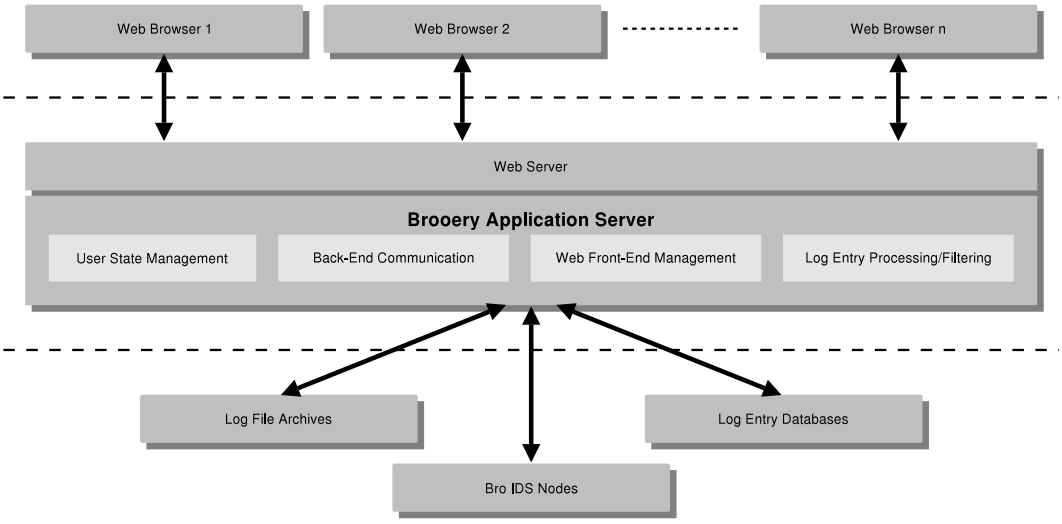

Figure 2: The Brooery's three-tiered Architecture.

aware of avenues attackers can follow in order to fool the analyst with false positives and negatives: an attacker might try to hide successful break-ins from the system, or equally dangerously, have the system report that a break-in did succeed where in fact there was none. The former causes the analyst to miss crucial information, while the latter would constitute a denial of service attack if for example machines had to be taken off line for investigation and recovery. The main attack surfaces are listed in the following and need to be protected carefully by the system:

- At the source. The log entry archives could be manipulated directly, introducing fake events or removing existing ones. Note that this is different from the typical case where IDSs are tricked into false positives or negatives; here, the storage system itself is subverted.
- In transit. Log entries need to be transferred and potentially filtered on their way from the archive to the analyst's console. An attacker with full control over the involved network flows could drop and introduce events at will, if the flows are not protected from tampering.
- During filtering & rendering. The system needs to process and filter log entries for presentation to the analyst. Similar to transit, an attacker who can modify the way the system itself operates can drop and introduce information or just cause the system in general to fail.
- At the destination. Since the application is mainly intended to read, process, and visualize existing information, the main benefit an attacker would gain from having access to the analyst's console is insight into what activity Bro nodes have been monitoring. When the system also permits the analyst to take administrative measures by updating running Bro nodes from the console, attackers with sufficient privileges to assume the role of an administrator could disable or attempt to crash individual Bro nodes.

4 The Brooery's Architecture

Given the requirements just outlined, we decided to implement our system in the three-tiered architecture illustrated in Figure 2. Analysts access the system through web browsers. The web server's back-end implements the core of the application, taking care of user interface rendering, user management, data source communication, and most importantly, the actual log entry processing. We will now discuss each of these components in more detail.

4.1 Web-based user interface

By using a web-based interface, we do away with the need to deploy stand-alone client applications,

while at the same time avoiding any porting overhead that such an application might entail. Web-based interfaces fall short of the interactivity of full-blown client-side applications unless they employ heavy-weight Java Applets or typically unrobust JavaScripts. We want to avoid the use of such features to keep the list of requirements on the client side as small as possible. However, as outlined in the our requirements, the system is primarily meant as an analysis tool for *investigation* of past activity and not per se as a real-time alert *notification* tool. Furthermore, the web-based interface has the obvious advantage that external web-based services can be leveraged immediately through HTML linkage, for example to provide vulnerability information,[3] common TCP/UDP port usage,[4] or reports of scanning activity.[5] We have implemented the web front-end using the Open Source web site development framework Mason[6] and the Apache web server. This allowed us to (*i*) stay within the same language in which we have already accumulated a considerable amount of log analysis code and experience (see Section 4.4 below), and (*ii*) employ more advanced language features and mechanisms for structuring the components of the generated web pages than provided by other popular web site development solutions like for example PHP.[7]

4.2 Multiple communication back-ends

The Brooery is designed to support multiple communication back-ends for communicating with log archives in the form of text file repositories, databases, or live Bro agents. Log archives can reside on remote machines or locally. While log entry archives are clearly only useful for mining past activity, the third mechanism is useful for more general purposes. For example, it could be used to request resource usage summaries from running Bro instances (suitable policies are already part of the Bro distribution), or to adjust their current policies dynamically. All of this would happen within Bro's existing communications framework, making this approach potentially very powerful.

4.3 Management of user sessions

We support access to the system by multiple users at all times. For each user, the system stores his or her current analysis context, including log entries, filtering combinations, inspection time frames, and general user preferences. We currently maintain user identities in the form of user name and perform authentication using passphrases. User identities are used to present returning users with the environment they left earlier.

4.4 Log entry processing & filtering

This component is the core of the system and provides the domain knowledge necessary to manipulate Bro events and log entries. It is implemented in Perl, for four main reasons: first, a large body of well-maintained Perl modules has already been developed in the general context of the Bro project, so we can instantly leverage these efforts. Second, Perl is well suited for rapid prototyping, which we feel is very much the correct development philosophy at this point in time. Third, the CPAN Perl archive offers a vast set of modules for any kind of extension we are likely to need in the future. Fourth, it is easy to create bindings from Perl to native C, should the need arise. This could happen for purposes such as performance optimization, or integration of other components.

The API for log entry retrieval hides the details of the underlying mechanism. For the log processing engine, a log archive is structured into different *log types* representing the different logging domains and abstraction levels at which Bro reports events (e.g., connection summaries, notices, signature matches, or alarms). Each log type's archive is comprised of a set of *log slices*, each labelled with a start- and end timestamp identifying the timeframe it covers. Within a log slice, log entries can be obtained subject to a filtering condition (see Section 4.4.3). Each back-end implementation maps these abstractions to the actual API available for accessing a particular log archive. For accessing text file repositories, we have developed a simple log entry server that the corresponding Brooery back-end communicates with. This resembles in many ways a poor man's database implementation; we stress again that our focus at this point is not on obtaining optimal performance but getting a good feeling for the problem setting.

4.4.1 Timeframe & Log Type Selection and Log Navigation

Our current interaction model requires the user to start the investigation by selecting a timeframe of

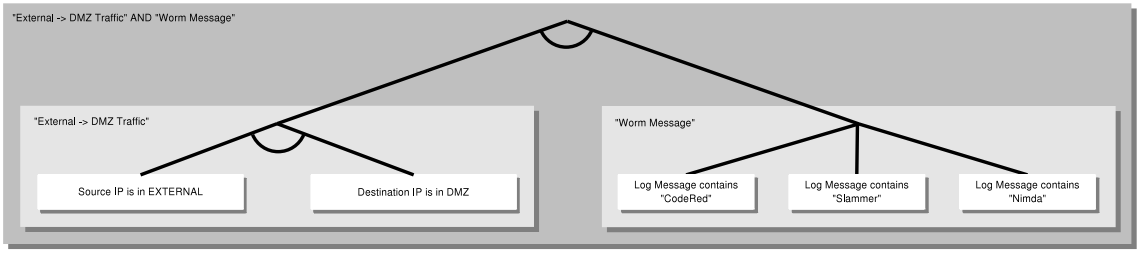

Figure 3: Recursive filter combination using AND/OR-trees and types log entry components: the main filter matches if both the left and right subfilters match, the left subfilter matches if the source IP of an entry belongs to external networks and the destination IP is inside the DMZ, and the right subfilter matches when the message string of a log entry contains "CodeRed", "Slammer", or "Nimda".

activity on which to focus investigation. This timeframe defines a lower and upper bound on temporal relevance of log entries, across different log entry types. After selecting a particular log type, the system then obtains a list of log slices that contain log entries during the configured timeframe. From these log slices, up to a given maximum number of entries are then requested starting from a given timestamp. Within the configured timeframe, the user can then step forward and backward through the log entries, manipulating a configurable maximum number of log entries at any one time.

4.4.2 A Type System for Log Entry Components

In our log entry model, every component of a log entry has a *type*, inspired by the types provided by the Bro scripting language. The Brooery's type structure is richer than Bro's and more hierarchical: at the very least, every log entry component is of the root type "data" which provides textual operators such as "contains" and "does not contain". A large number of different types derive from this root, for example timestamps, flow sizes, port numbers, IP addresses, and protocol names. Further specializations exist for example for source and destination IP addresses.

The benefits we gain from adhering to such a type model throughout all of our log types are *associativity*, *extensibility*, and *semantic processing*: regardless of the type of log we are currently investigating, a source IP address in one log file type will semantically represent the same as a source IP address in a different log type. This allows easy integration of future log types because only novel component types need to be integrated in the type hierarchy. The only information required to support a new log type is the sequence of the components' types. Furthermore, knowing that a log entry component represents for example a timestamp allows us to perform according operations on the component, in this case for example operations such as "earlier-than", "after", or "between".

4.4.3 Recursively Reusable AND/OR-Trees for Log Entry Filtering

The Brooery supports an elaborate concept of log entry matching based on AND/OR-trees, known from other applications such as attack trees [5]. The Brooery combines the expressiveness of conditions using AND/OR-trees with the strengths of the typed log entry components. For example, timestamps can be matched depending on whether they represent time earlier or later than a given timestamp, and IP addresses can be tested for (not) matching an address prefix.

A filter always consists of one ore more *filter parts*, each of which can either contain a filtering criterion as just described, or refer to another existing filter. The filtering results of all filter parts are then combined using Boolean conjunctions or disjunctions and lazy evaluation. Cyclic dependency detection prevents the user from configuring self-referring constructs. This approach to filter management allows the creation of arbitrarily nested filtering *hierarchies*, while ensuring easy re-use of existing filters. Figure 3 illustrates the concepts.

Note that a filter so far only represents a focusing mechanism; "filtering" is not meant to imply

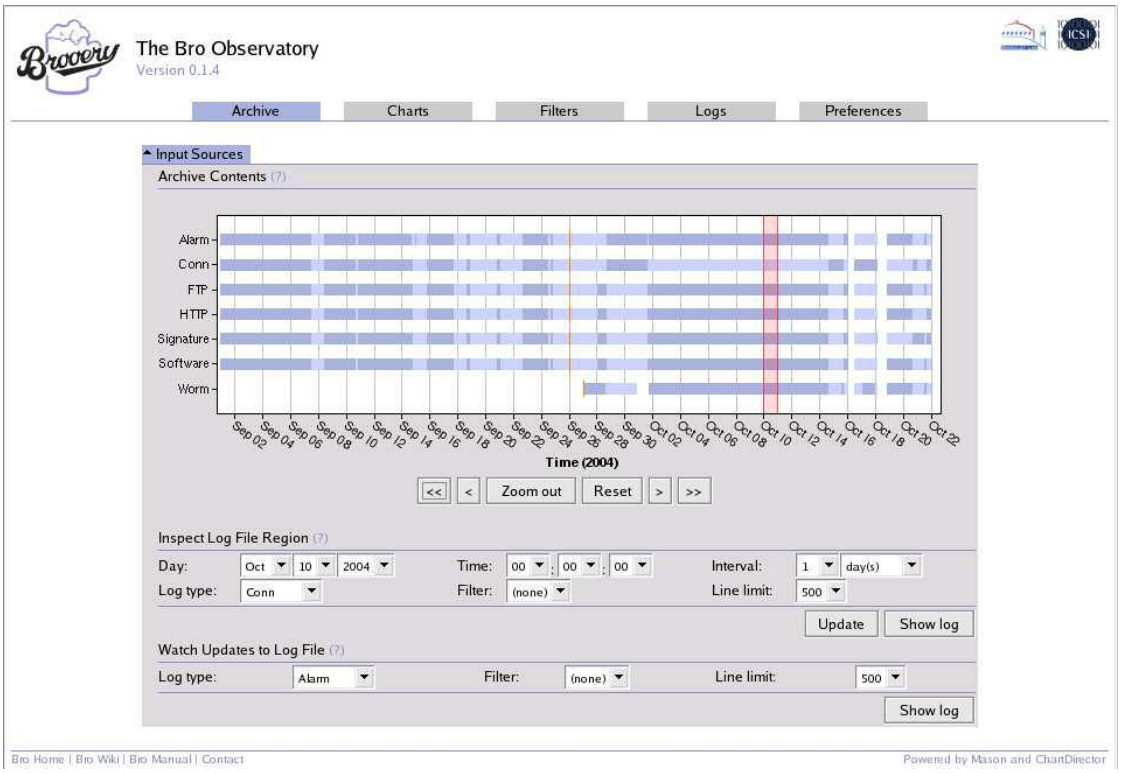

Figure 4: View of a Bro node's log archive.

"dropping" at this point. Dropping a log entry due to successful filter evaluation is merely one of a range of conceivable *filter actions*; other examples include keeping a log entry, labelling it, and *aggregating* log entries by compressing all entries that are matched by a filter into a single abstract entry. We can summarize the aggregation by reporting in an abstract entry the number of entries it represents, while maintaining the common parts of those log entries visible.

The fact that multiple filters can trigger actions of different semantic meaning does imply that multiple filters may need to be active at the same time. For example, one filter could throw out unwanted entries, another one could aggregate the remaining ones. Integrating the output of multiple filters brings the potential of *conflict*: for example, one filter could declare that an entry is to be dropped while a second one asks for it to be kept. The way we solve this problem is through ordering: while the user may have multiple filters active at the same time, those filters have to be put in a sequence, and the first decision made in a conflict domain is decisive.

To allow the user to quickly weed out unwanted information and focus on the interesting entries, the Brooery also supports *incremental* filtering, i.e., the addition of new filter parts to a selected filter. At no time does the nature of the log entry storage shine through; for example, the user never has to resort to entering raw database queries.

4.5 Security Considerations

As outlined by our threat model, care must be taken to restrict the user base of the system to the intended individuals while preventing others from eavesdropping on or even tampering with the information flows. First of all, we assume that when an attacker manages to break into one of the hosts running Bro, or one of the machines storing a log archive, it is unlikely that we can prevent a determined intruder from causing serious damage to the system. Therefore, the security precautions of Bro nodes and log archives remain unchanged, regardless of whether these systems are interfaced with the Brooery or not.

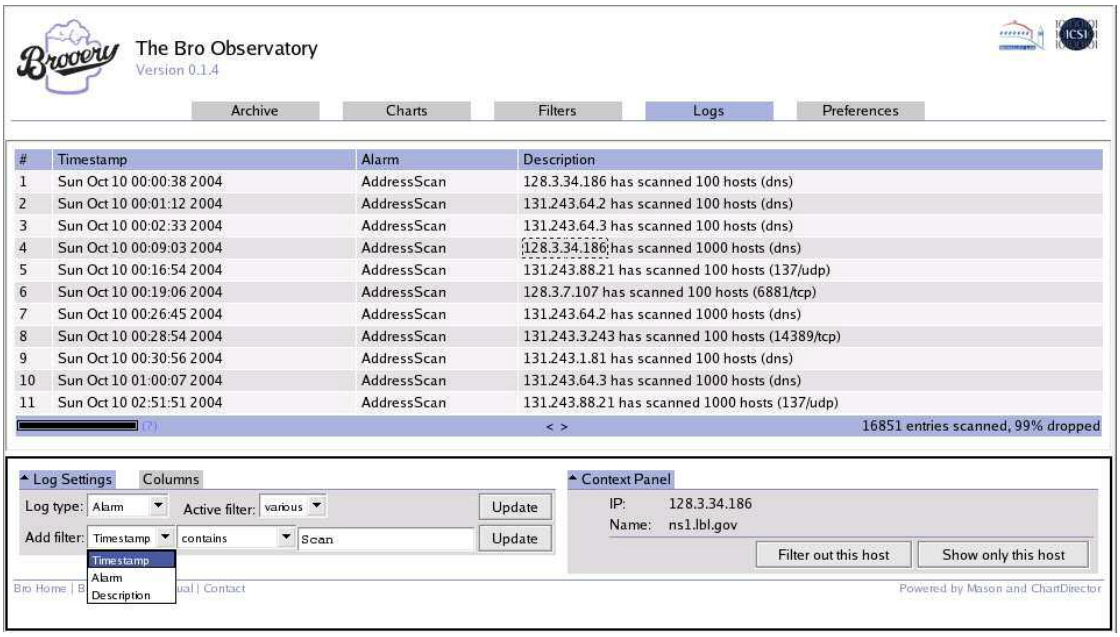

Figure 5: Alarm log entries, filtering the "Alarm" column for entries containing "Scan".

Regarding the communication between the various nodes interacting with the Brooery, we use two different levels of restrictiveness: inside the Bro network, we can assume that the communicating entities know each other's identities. Besides employing SSL encryption, we can therefore require mutual authentication of the communicating peers through certificates. The web browsers accessing the system are still required to use encrypted connections via HTTPS, but as mentioned in Section 4.3 we drop the requirement for client-side certificates and resort to weaker authentication in the form of user names & passphrases. Furthermore, we aim to restrict the reachability of the involved hosts to a minimum whenever possible, for example by only making the ports on which Bro data sources can be tapped available on a separate network.

5 Usage Example

We will now give a quick but illustrative example of user interaction with the Brooery. Let us a assume that we have been informed that on October 10, several users have noticed unusual connection attempts to their machines. Our goal now is to find out whether any relevant scanning activity was detected that day. Figure 4 shows a Bro node's log archive, and we can see that the node has plenty of information for October 10: there are 7 different log types ranging from alarms over connection summaries to worm events, reaching in time from the beginning of September to the present at the time the screenshot was taken. Horizontal blue bars indicate the timeframes during which the Bro node was logging events of a particular type. The blue color is differentiated into two different shades, with each color switch indicating the start of a new log slice. Note that the bars cover the timeframe when the IDS was *ready to log*, not the timeframe from the first logged event to the last one — an important semantical difference. For example, we can see that the system was not monitoring at all for several hours on October 16 and 18, and that worm event logging was introduced on September 27. The small vertical orange lines delineate compressed from uncompressed archival, i.e., log entries residing on the left side of an orange line are stored in compressed fashion.

We specify a time interval covering that day, and look at the contents of the alarms log. We then add a filter on the alarm names that matches all entries containing "Scan", and obtain the result shown in Figure 5. As can be seen, several hosts have scanned a large number of machines, and we can now continue our analysis by looking at the connection summaries for each of those hosts during the given time

period. To do so, we click on any of the shown IP addresses, add a filter part for that address using the option the context panel for IP addresses provides for this purpose, and switch the log type to the connection logs.

6 Related Work

In [6], Hoagland and Staniford proposed SnortSnarf, a web-based console for analyzing Snort alerts [7]. ACID[8] is similar and additionally allows the generation of charts and statistics. Our system is superficially similar to these, however our system is more comprehensive in that it (*i*) can manage a wider variety log types and (*ii*) structures log entries more thoroughly due to typed column entries for stronger semantic filtering across different log types. Sguil[9] is close to our system in the sense that it acknowledges the need for providing contextual information for alerts such as connection logs and packet content. Our system differs from theirs in that filter management in Sguil is less intuitive for the user (who has to resort to SQL statements); also, Sguil is implemented in Tcl/Tk and therefore not as readily accessible as our web-based interface. A number of other user interfaces for Snort exist; they are typically geared towards support for signature management and do not provide the flexibility to deal with the wide range of log information provided by Bro. In the commercial space, user interfaces are often bundled with IDS products directly or offered in the general Security Incident Management domain. As our focus is on open-sourced solutions, we do not review the commercial domain thoroughly in the scope of this paper.

7 Discussion & Future Work

The Brooery is work in progress and a prototyping testbed. We use it for experimentation with different models for analyzing log information and therefore feel it is important to point out that we are currently primarily interested in different metaphors for manipulating the log information; aspects such as performance optimization remain secondary. So far we have found the graphical instruments realizable using HTML sufficient for our needs; it will be interesting to see if this observation will apply to future extensions to the system as well.

We currently see two main avenues for future work. First, we need to augment Bro with a database logging component that does not require fundamental modifications of Bro's logging component. Database-driven archival is very much a necessity for robust log entry storage & retrieval, and, of course, performance. Text-file based storage, while familiar and to a certain degree manageable at the command line, restricts performance and can sometimes pose technical difficulties. One avenue we are considering for achieving this is to turn the act of logging an event into an event itself. That way, the implementation of the logging mechanism would remain up to individual event handlers, could happen in multiple ways in parallel, and other event logging systems (including the Brooery) could tap into the stream of logged events using e.g. Broccoli and the existing event communications framework. This approach would thus fit very nicely into Bro's model. Depending on the implementation of the event handlers responsible for processing such logging events, the events would then be stored in a text file, a database, or processed in some other way. The Brooery's log entry model is geared towards easy mapping onto relational structures; the main difference her are the semantically stronger types used at within the log entry engine.

Second, we intend to investigate the requirements for effective analysis of distributed events. We are currently correlating events originating on multiple Bro nodes using Bro's event communication framework; however, it is not yet clear to us what will turn out to be the best visual metaphor for controlling this correlation and visualizing the results.

8 Summary

We have presented the Brooery, a three-tiered experimental prototyping platform for graphical analysis of network activity reported by instances of the Bro IDS. The system provides contextually relevant drill-down features and supports different Bro log archival back-ends; semantically strong and reusable log entry matching based on AND/OR trees; filtering, labelling, and aggregation of log entries; and hierarchically typed log entry components. The Brooery's development is fully open sourced under a BSD license. More details can be found at `http://www.icir.org/twiki/bin/view/Bro/BrooeryGUI`.

Acknowledgements

This work is carried out in collaboration with Intel Research Cambridge, ICIR, and Lawrence Berkeley National Laboratory. It was supported in part by the U.S. National Science Foundation grant STI-0334088, and the U.S. Department of Energy. We would like to thank Vern Paxson, Jon Crowcroft, and the other Bro developers for helpful discussion and feedback, and the occasional brainstorming in real-world "brooery" environments.

References

[1] Vern Paxson. Bro: A System for Detecting Network Intruders in Real-Time. *Computer Networks (Amsterdam, Netherlands: 1999)*, 31(23-24):2435–2463, 1998.

[2] Robin Sommer and Vern Paxson. Exploiting Independent State For Network Intrusion Detection. Technical Report TUM-I0420, TU München, 2004.

[3] Christian Kreibich and Robin Sommer. Policy-controlled Event Management for Distributed Intrusion Detection. In *Proceedings of the 4th International Workshop on Distributed Event-Based Systems (DEBS'05)*, June 2005.

[4] Robin Sommer and Vern Paxson. Enhancing Byte-Level Network Intrusion Detection Signatures with Context. In *Proc. 10th ACM Conference on Computer and Communications Security*, 2003.

[5] *Secrets and Lies*, pages 318–333. John Wiley and Sons, New York, 2000.

[6] James A. Hoagland and Stuart Staniford. Viewing IDS alerts: Lessons from SnortSnarf. Technical report, Silicon Defense, Nov 2000.

[7] Martin Roesch. Snort: Lightweight Intrusion Detection for Networks. In *Proceedings of the 13th Conference on Systems Administration*, pages 229–238, 1999.