NTNU
Norwegian University of
Science and Technology

LogWheels: A Security Log Visualizer

Vegard Egeland

Master of Science in Computer Science
Submission date: June 2011
Supervisor: Øivind Kure, ITEM
Co-supervisor: Audun Jøsang, UNIK

Norwegian University of Science and Technology
Department of Telematics

Problem Description

The use of information systems has become a given in any modern organization. Security is a fundamental requirement for most of these systems, and the default behavior when encountering potential security incidents is to document them in textual logs.

Experience from the industry indicates that these incident logs are poorly monitored. Perhaps the most important reason is the poor usability: each organization is faced with a variety of different log formats and unclear responsibilities. Poor security usability leads directly to vulnerabilities that can be exploited by hackers and criminals.

The student is tasked with examining how the security usability of incident logs can be improved. One or more of these techniques can be demonstrated experimentally by analyzing and visualizing real or simulated security data.

Assignment given:	15. January 2011
Supervisor:	Øivind Kure, NTNU and UNIK
	Audun Jøsang, UiO and UNIK

Abstract

Logging security incidents is a required security measure in every moderately complex computer system. But while most systems produce large quantities of textual logs, these logs are often neglected or infrequently monitored by untrained personnel.

One of the reasons for this neglect is the poor usability offered by distributed repositories of plain text log data, using different log formats and contradictory terminology.

The use of security visualization has established itself as a promising research area, aiming to improve the usability of security logs by utilizing the visual perception system's abilities to absorb large data quantities.

This thesis examines the state of the art in security log usability, and proposes two ideas to the areas of security log usability and security visualization:

First, we introduce *LogWheels*, an interactive dashboard offering remote monitoring of security incident logs, through a user friendly visualization interface. By offering three levels of granularity, LogWheels provides both an overview of the entire system, and the opportunity to request details on demand.

Second, we introduce *the incident wheel*, the core visualization component of LogWheels. The incident wheel presents three key dimensions of security incidents – 'what', 'when', and 'where' – all within a single screen.

In addition to a specification of LogWheels architecture and visualization scheme, the thesis is accompanied by a functional proof-of-concept, which allows demonstrations of the system on real or simulated security data.

Acknowledgements

This report is the final delivery of the master's program in computer science at The Norwegian University of Science and Technology (NTNU).

The project was undertaken at the University Graduate Center at Kjeller (UNIK), under the guidance of professor Audun Jøsang and professor Øivind Kure. I would like to thank both supervisors for their feedback and support throughout the course of the project.

A special gratitude goes to Andreas Hegna at Watchcom Security Group, for the inspiring discussions leading up to this thesis. The ideas put forth during our discussions were instrumental in carving out the final problem statement, and provided valuable insight into the pragmatics of security visualization.

I would also like to thank Geir Mork at Norman for our meeting and discussion about the security industry, his helpful suggestions towards the prototype and potential applications of the thesis.

While I greatly appreciate all the help and suggestions, any errors, oversights or omissions are mine and mine alone.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

The first chapter provides an introduction to the report.

Divided into five sections, this chapter explains the motivation for choosing security log usability as a research area, specifies the project goals, and describes the scope of the report. Insight into the chosen research method is given, and finally a roadmap for the thesis is provided.

«Read them,» said the King.

The White Rabbit put on his spectacles. «Where shall I begin, please your Majesty?» he asked.

«Begin at the beginning,» the King said gravely, «and go on till you come to the end; then stop.»

– from “Alice’s Adventures in Wonderland” by Lewis Carroll

1.1 Motivation

Logging – the act of recording events occurring in a system – is a mandatory security measure in any moderately complex digital environment. Logs can be used to detect and investigate incidents after they have occurred, but also to assist in the prevention of harmful incidents, by revealing use and abuse patterns and increasing situational awareness [7].

While large organizations can hire dedicated security personnel to monitor the logs, the strategy in smaller organizations is often reminiscent of the conversation between the King and the White Rabbit in the introductory quote. Logs are stored as chronological lists of textual data, and are infrequently skimmed through using basic command line tools [35]. As a consequence, the organizations are at risk of missing both specific security incidents and the big picture [41].

The challenge for these organizations is to balance limited time and resources with the steadily increasing amounts of textual log data being produced [46]. In order to face this challenge, there is a need for user friendly log monitoring tools. Security measures do not provide their intended protection unless the end users know how to use them correctly [44], and in order to utilize the potential of security logs, monitoring them ought to be intuitive and pleasurable, without sacrificing any detail.

A promising perspective on security log usability is the emerging research field of security visualization. Security visualization systems utilize the pattern seeking abilities of the visual perception system, allowing rapid discovery of deviating behavior [50]. By including a well-designed interaction interface, users are encouraged to monitor logs both holistically and in detail [69].

The motivational driver for this thesis was to design a new visualization scheme for security incident logs, firmly rooted both in the worlds of information security and usability. The idea appeared during the author's own period as a system administrator in a small startup company, and was motivated by the desire to get an overall view of the system's security state, rather than occasionally investigating bits and pieces in detail.

1.2 Project Goals

Three high-level goals served as the basis for this project:

PG1 Examine the state of the art in security log usability

PG2 Propose a mechanism to improve the usability of security logs

PG3 Develop a proof-of-concept, illustrating the ideas proposed

In addition to the academic goals, the report was written as a documentation of the final project of the master's program in computer science at The Norwegian University of Science and Technology (NTNU). The university encourages students to take on scientific and technological challenges, showing originality while building on state of the art research, and presenting the project in a structurally clear fashion.

1.3 Project Scope

The project was executed during a 20 week period in the spring of 2011. Due to the limited time frame, a narrow focus was necessary, and some interesting examinations of the ideas proposed had to be left out.

First, the idea has not been empirically tested, nor exposed to a proper real-world feasibility study. These activities are required before taking the concept any further. The focus of the report is on the ideas, and although they were based on state of the art research, and exposed to critical thoughts (chapter 6), this is not a replacement for an empirically driven exploration.

Second, while one of the goals was to develop a proof-of-concept, it was beyond the scope of this project to develop a software tool of production quality. The prototype attached to the report should not be expected to satisfy all the demands being made in a professional setting, but rather to serve as a basis for further investigation on the ideas presented in the text.

Section 7.2 suggests a roadmap for future development of the solution.

1.4 Research Method

As indicated by the high-level goals, this project is primarily one of theoretical studies and construction work.

The main driver of the theoretical study was an examination of state of the art literature, and two meetings with representatives from the security industry. Before the project started, Andreas Hegna from Watchcom Security Group was consulted about the security visualization field [35], and after the completion of the preliminary study, Geir Mork from Norman contributed with industry experience on logging and security usability [58].

While the focus of the report is on ideas and implementations, we hope to see the project move into an empirical phase in the future. The report outlines some of the necessary considerations in chapter 6 and 7.

1.5 Report Outline

The report is divided into seven chapters, including this introduction:

Ch 2 Presents the theoretical foundation of logging, log usability, and visualization.

Ch 3 Examines specific implementations of the ideas outlined in ch 2.

Ch 4 Proposes a new visualization scheme, intended to increase usability of security incident logs in small organizations.

Ch 5 Describes the proof-of-concept implementation of the proposition from ch 4, with specific use cases.

Ch 6 Discusses the experiences learned from the proof-of-concept implementation and execution.

Ch 7 Concludes the report and provides a roadmap for the further work required to turn the idea into successful software.

Chapter 2

Background

This chapter provides the theoretical foundation required to understand security logs and log usability, and introduces the principles used to design the system proposed in chapter 4.

Section 2.1 presents security incident logs, and how they are produced and consumed in a computer system.

Section 2.2 proceeds by introducing the usability challenges caused by standard log implementations, and suggests a framework for techniques which can improve log usability.

Section 2.3 examines how visualization can be used to improve usability, and presents some key principles used in LogWheels' visualization and user interaction schemes.

2.1 Security Logs

The following sections provide an introduction to the concept of security logs, why they are being used, and how a security log infrastructure can be implemented.

2.1.1 Terminology

In its most general sense, a **log** is the written record of events occurring in a system [75]. An **event** can be any observable situation, ranging from expected state changes to suspected intrusion attempts or unexpected internal failures [22].

Information about a single event is stored in a **log record**. The record is structurally divided into **fields**, each field storing a single property of the event [22]. Examples of fields include timestamps, user agent identifiers, and a descriptive message indicating what happened.

The record and field concepts intuitively correspond to database rows and columns, but there are no general standard as to how they should be stored. Logs must therefore be associated with a **log format**, which specifies how the records and fields are encoded.

An example of a log with a common log format is presented in figure 2.1.

This thesis will focus exclusively on security incident logs. In this context, a **security incident** is an event which was determined by a non-human entity, such as an intrusion detection system or a firewall, to be a potential threat to the security of a computing system.

The formal definition can be derived from the National Institute of Standards and Technology's work on security log management [46]:

> **Security incident log** A record of security related events occurring within an organization's systems and networks

The hardware and software involved in the log generation and analysis process is referred to as the **log management infrastructure** [46]. The entities of the infrastructure can be subdivided into two main classes: producers and consumers.

Producers are hardware and software generating log data. They can be hardware (e.g. routers), operating systems, designated security software (e.g. firewalls), or regular applications (e.g. web servers) [22, 56].

Consumers are software tools perform work on existing log data [22]. They serve several purposes, such as consolidating different log formats, performing automatic analyses, or presenting log data visually.

The default behavior of log producers is to continuously append log data to a plain text file on the local file system. The consumers read the files in batch or real time, parse them according to the given log format, and analyze or present the data.

```
Apr 23 08:12:49 server sshd[535]: PW-ATTEMPT: root
Apr 23 08:12:49 server sshd[535]: Failed password for root from
  10.0.160.14  port 39529 ssh2
Apr 23 08:13:02 server sshd[535]: Illegal user administrator from
  10.0.160.14
Apr 23 08:13:02 server sshd[535]: PW-ATTEMPT: administrator
Apr 23 08:13:02 server sshd[535]: Failed password for illegal
  user administrator from 10.0.160.14 port 40444 ssh2
Apr 23 08:13:57 server sshd[535]: PW-ATTEMPT: root
Apr 23 08:13:57 server sshd[535]: Failed password for root from
  10.0.160.14  port 39529 ssh2
Apr 23 08:14:04 server sshd[535]: Illegal user administrator from
  10.0.160.14
Apr 23 08:14:04 server sshd[535]: PW-ATTEMPT: administrator
Apr 23 08:14:04 server sshd[535]: Failed password for illegal
  user administrator from 10.0.160.14 port 40444 ssh2
Apr 23 08:14:04 server sshd[535]: PW-ATTEMPT: administrator
Apr 23 08:14:04 server sshd[535]: Failed password for illegal
  user administrator from 10.0.160.14 port 40444 ssh2
```

Figure 2.1: Example of a log generated by sshd

2.1.2 Technical Infrastructure

Log production is rarely the problem [7]. Most applications have sensible default settings, which can be customized if required. Unless precautions are taken to *prevent* logging, both server software and hardware will produce most of the security data required, and store it on the local file system. Adding a new producer is usually as simple as installing a new piece of software and configuring it according to the documentation.

Consuming logs is harder. Any reasonably advanced computing system generate several logs, but they are stored in different locations, using inconsistent log formats and different interpretations of similarly named fields [46].

These challenges can be solved by a good log management infrastructure. NIST suggests a three-tiered infrastructure [46], an approach which was described in detail by Phillip Q. Maier [54]. As illustrated in figure 2.2, this model divides the log management infrastructure into three abstract tiers:

1. The first tier contains all log producers, which are observing events and generating raw log data.
2. The second tier contains log consumers responsible for data consolidation, normalization, and early filtering techniques.
3. The third tier contains another set of log consumers, responsible for storing the normalized log data, and potentially perform automatic analysis of the logs, or presenting them to an end user.

More complex architectures are available for high-volume systems or widely distributed networks, but the fundamental principles remain the same [54].

Ideally, different physical machines are used at each tier. This prevents the log consumers from interfering with the performance of the system. Communication between the different devices can be handled by standard TCP/IP connections.

In smaller systems, the entire process can be replicated by a software stack. As long as the underlying hardware is able to handle the load, the three-tiered architecture can be executed on a single machine.

The log management infrastructure is implemented through a set of software tools. Three popular approaches can be identified:

- In simple systems, the entire infrastructure can be developed in-house, e.g. using shell scripts and a relational database.
- More complex systems can benefit from commercial off-the-shelf software solutions, commonly referred to as *SIEM solutions* [46].
- A popular compromise is to use a tool chain based on *syslog*, a communication protocol describing a high-level log format and how to transport log data between different tiers [60]. Syslog data is produced by most popular log producers, and compatible log consumers are widely available [1, 32]. With good library support, it's easy to develop new syslog tools.

This thesis will be neutral with regards to the implementation, but will assume the availability of a single repository of similarly formed log records. A more comprehensive overview can be found in [46] and [54].

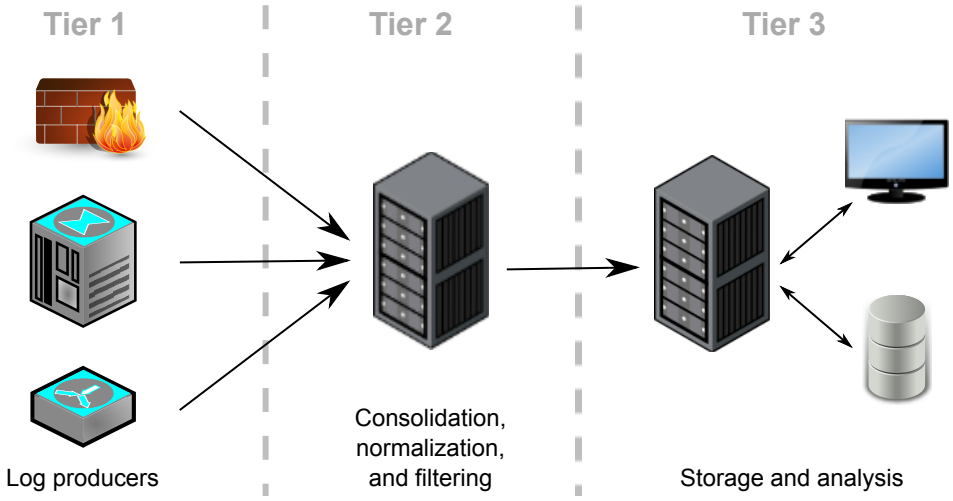

Figure 2.2: Three-tiered log consolidation architecture

2.2 Log Usability

A properly implemented log management infrastructure ensures that log data is available on a common format in a single repository, but still leaves large quantities of textual log data to be interpreted.

This section presents the main challenge of log monitoring, how the principles of security usability can be applied to face these challenges, and a framework of specific techniques used to increase the usability of log consumers.

2.2.1 Challenge

Security incident logs serve several purposes, even in small organizations. Some organizations may be exclusively motivated by legal requirements, but the main objective should be to increase the security of the system.

While large organizations can hire dedicated IT security personnel, industry experience indicate that smaller organizations lack the knowledge and resources to handle the continuous supply of log data sufficiently [35, 46, 58]. As a consequence, no long term trends are identified, resulting in a scheme where the understanding of the system is low, discovery of incidents is subject to the forces of luck, and investigation performed ad hoc [56].

In the words of Mica Endsley: "the problem is no longer lack of information, but finding what is needed when it is needed" [21]. This applies directly to security logs. While there is no lack of high quality log producers, the industry suffers from a lack of user friendly log consumers targeted towards small organizations.

One of the most important tasks for a log management infrastructure is ensuring that the security operator has a high *situational awareness* [50]. Log consumers targeted towards small organizations must therefore be designed to increase the situational awareness and the security usability, without sacrificing the details needed to make informed decisions.

The concepts of situational awareness and security usability will be presented in the following sections.

2.2.2 Situational Awareness

Livnat et al. defines situational awareness as "the ability to identify, process, and comprehend the critical elements of information about what is happening" [50]. In layman terms, situational awareness is a measurement of an operator's knowledge about what's going on in their system.

According to the Endsley model, there are three levels of situational awareness, growing from low to high understanding of the system [20]:

Level 1 - Perception Important information about the environment must be *observed* by the end user.

Level 2 - Comprehension The end user must be able to *interpret and understand* the perceived information.

Level 3 - Projection A practioneer should be able to *project* future statuses of the system, guiding decision making.

In their research on situational awareness failures of pilots, Jones and Endsley discovered that 76.3 % of all errors were caused by failure to perceive the situation correctly, 20.3 % by failure to interpret the perceived information correctly, and 3.4 % by failure to project future situations [42]. In total, it's been shown that 88 % of all accidents among air carriers involve situational awareness problems, and reviews indicate that these importance of situational awareness is not unique to pilots, but that it applies any operator of complex systems [21].

Lakkaraju et al. [48], Yin et al. [79], and Livnat et al. [50] are among the growing body of security researchers looking for techniques to improve situational awareness among security operators. By improving the situational awareness, the operators are more likely to discover security incidents early, defend the system against incidents before any damage occurs, and understand which parts of the system are being threatened [79].

Security logs provide the data needed to provide situational awareness, but the usability of this data does not appear to be adequate.

2.2.3 Security Usability

In order to properly address the usability of security logs, it's necessary to take a step back and investigate the fundamental principles of *security usability*.

The term "user-centered security" was introduced by Zurko and Simon in 1996, in an effort to include user needs as a primary design goal during the development of security systems [81]. Their claim was that security systems have a tradition for ignoring the user, leading the industry to accept that security and usability were somehow contradictory terms.

Ka-Ping Yee is among the researchers who followed up on the challenge, claiming that security and usability can be aligned [78]. Both factors must be ingrained in the development process of a tool, but if done properly, the added usability will increase the security, rather than obstructing it.

A more concrete effort was put forth by Jøsang et al [44]. They take the alignment idea a step further, claiming that security systems are unable to provide intended protection unless people understand how to use them correctly, and that significant risks are caused by poor security usability. They provide specific case studies, demonstrating the principles in the cases of identity management [44, 45] and online bank security systems [5].

After accepting the idea that usability often is the weakest link in the security chain, Jøsang et al. suggest a set of principles for security usability. First, they define *security actions* and *security conclusions* [44]:

Security action When users are required to produce information and security tokens, or to trigger some security relevant mechanism. For example, typing and submitting a password is a security action.

Security conclusion When users observe and assess security relevant evidence in order to derive the security state of systems. For example, observing a closed padlock on a browser, and concluding that the communication is protected by TLS is a security conclusion.

Based on Kerckhoffs' security principles, dating more than 100 years back, Jøsang et al. proceed to introduce eight principles for security usability [44]:

Security Action Usability Principles

A1. Users must understand which security actions are required of them.

A2. Users must have sufficient knowledge and the ability to take the correct security action.

A3. The mental and physical load of a security action must be tolerable.

A4. The mental and physical load of making repeated security actions for any practical number of instances must be tolerable.

Security Conclusion Usability Principles

C1. Users must understand the security conclusion that is required for making an informed decision.

C2. The system must provide the user with sufficient information for deriving the security conclusion.

C3. The mental load of deriving the security conclusion must be tolerable.

C4. The mental load of deriving security conclusions for any practical number of instances must be tolerable.

Examining a security log is an example of drawing a security conclusion. The four security conclusion usability principles all apply to security log consumers, which must provide sufficient detail, while simultaneously being simple enough to encourage frequent observation.

Principle C1 is primarily directed at educating security personnel, which is outside the scope of this thesis. Principles C2, C3 and C4 should be instrumental in the development of any log consumer aiming to improve log usability, and will serve as the basis for the solution proposed in this thesis.

2.2.4 Specific Techniques

The principles of security usability are important, but still abstract. By studying their practical applications, a framework of specific techniques can be created. These techniques are added to the tool chain described in section 2.1.2, and improves the usability of the log scrutinizing process.

The framework used in this thesis is listed in table 2.1, and is based on similar frameworks by Phillip Q. Maier [54], Raffael Marty [56], and Kent et al. [46]. The techniques are implemented through log consumers, located in tier 2 or 3 of the log management infrastructure.

Group	Technique	Tier
Add data	Using third-party sources Internal correlation	2
Reduce data	Aggregation Filtering Normalization	2
Present data	Alerts Reports Historical analysis Dashboard	3

Table 2.1: Techniques to increase usability

Note that these techniques both can and should be combined.

Add Data

The first group of techniques results in new log records being added, or existing log records being expanded with new fields.

Records can be expanded by adding data from *third-party sources*. As an example, using IP address geolocation, the system can add a city and country name in addition to the numerical IP address [38].

New records can be made by using *internal correlation*. Correlation is used to find relationships between events, preferably from different security logs [54]. If one IP address is registered with illegal requests to several services, it's valuable to be informed of this directly, rather than having to inspect every log and manually notice these trends.

Reduce Data

Data reduction is often used in combination with data addition, but the fundamental idea is the polar opposite. By removing irrelevant log records or fields, these techniques intend to increase the signal-to-noise ratio of logs.

The *filter* suppresses low-risk log records [46]. As an example, the majority of requests to a web server is likely to be for static resources in public folders, requests which are highly unlikely to be malicious. Requests with special characters directed at dynamic resources are more important to report.

Another approach to data reduction is *aggregation*, where a collection of events are merged into a single record [46]. For instance, the number of failed logins per IP address is more helpful than a list of every single failed request, surrounded by irrelevant events [56].

Finally, *normalization* removes systematic variation from a data set [77]. This can be used to eliminate mistakes by underlying log producers or threats the system has proved to handle, and instead focus on highlighting statistically unusual incidents.

Data reduction is challenging, as it depends on the system's ability to predict patterns of events. Great care should be taken before permanently removing data [41].

Present Data

In contrast to the other two groups, presentation occurs on the third tier in the consolidation architecture. Presentation tools can be appended to an already existing log management infrastructure by accessing the log repository directly.

The four presentation techniques in this group can be classified in a two-dimensional matrix, as illustrated by table 2.2.

First, a distinction is made between *batch* and *real-time* processing. The former parses new log data on a time periodic basis. For instance, data can be processed every 24^{th} hour, during a time of day when the system is under light load [54]. Real-time analysis is more demanding on the system, but in return it provides immediate feedback.

Second, presentations can be *static* or *interactive*. The former provides a single view, e.g. a generated e-mail or PDF file, while the latter allows the user to interact with the data, using techniques such as hypertext, zooming or animation. The latter increases the design and implementation effort, but is richer in functionality and improves usability [69].

	Real-time	**Batch**
Static	Alerts	Reports
Interactive	Dashboard	Historical analysis

Table 2.2: Presentation matrix for security log data

(a) Alerts The simplest presentation addition to security logs are alerts. This scheme constantly parses the logs looking for incidents, and warns the user immediately if it detects a security breach. Alerts can be sent through e-mail, SMS, or similar communication devices. Alert systems are often incorporated in dedicated security software, such as intrusion detection systems, and reduce the need to constantly monitor the logs [54].

While alerts are helpful in systems that are not constantly monitored, they are not a replacement for a comprehensive usability solution. Defining the thresholds is a continuing challenge [48]. If too many alerts are reported, they will eventually be ignored. If too few alerts are sent, the system administrator will not only risk missing bigger trends, but complete attacks may go unnoticed because they never cross the threshold for sending a warning.

(b) Reports A slightly more advanced way to improve the log scrutinizing process, is by generating reports in batch. A report solution can use addition and reduction techniques, and summarize a days worth of data using a graphical or textual summary [56].

Reports increases situational awareness, but less so than real-time systems. They provide an overview of critical system information, but by the time a report is received, the only viable strategies are investigation and recovery. Reports are a good supplement to real-time dashboards, as they can cover trends over larger time frames.

(c) Historical Analysis Historical analysis expands the static reports by introducing interactivity. Historical analysis systems parse logs in batch, and present them in a dynamic user interface, allowing the user to interact with the graphical components and further investigate anomalous values without leaving the report.

A good log analysis scheme should make comparisons with historical data, clearly presenting deviations from the norm [56]. The report should be kept small enough to encourage frequent usage, while simultaneously containing all necessary data

(d) Dashboard The most advanced log presentation technique is the dashboard. It expands the interactive historical analysis system by introducing real-time data parsing, which provides the user with continuous situational awareness through a constantly moving interface [56]. The interface can be monitored at a glance, and events can be examined the moment they occur.

While dashboards are the presentation tactic best suited for situational awareness applications, they are harder to implement, and may cause a significant performance penalty on the system. Large amounts of log data must be parsed in real time, memory must be available to store temporary data, and the machine must be able to present data to the user while simultaneously handling interface requests. For this reason, it's not recommended to run dashboards on production servers, but on a designated piece of hardware.

2.3 Visualization

While the usability techniques suggested at tier 2 (addition and reduction) are helpful, their end result is still a repository of plain text data. The presentation techniques at tier 3 can use visualization to improve usability.

This section presents the fundamentals of data visualization, and how it can be used to increase security log usability.

2.3.1 Foundations

Plain text logs are not ideal when aiming for high situational awareness. While plain text is easy to parse computationally, the human brain is poorly equipped for processing large text quantities [56].

Fortunately, absorbing large amounts of data is a task almost perfectly adapted to another part of the brain: the visual perception system [50]. With an appropriately designed data visualization scheme, the average human can sift through gigabytes of log data, while discovering anomalous trends, all with a single glance on a computer screen [48, 79].

Visualization can be informally defined as the presentation of abstract data using computer graphics [73], or more formally as "the use of interactive visual representations of abstract data to amplify cognition" [69]. The study of visualization is well established, with statistical data being represented visually as early as in the 1750s, and the growth of the computing business have provided a wide array of digital applications of the original research [8, 74].

Security log visualization is an exciting application of the visualization research field, showing promising results. By taking advantage of the brain's ability to rapidly perceive and comprehend information, visualization increases the situational awareness in a security system [50].

Raffael Marty, one of the pioneers in the field of security visualization, developed the *information visualization process* as a formalized method to present data graphically [56]. The process is illustrated in figure 2.3.

The process follows a conceptual path similar to the previously presented three-tiered log management infrastructure. A separation is made between the data itself, its processing, and finally the visualization.

As the process was designed for specific visualization challenges, rather than the generic approach of this thesis, the first two steps have been generalized in the following description.

Step 1, the problem definition, is to increase the situational awareness of the operator by improving the security usability (see section 2.2.2). Step 2, assessing available data, is covered by the first tier (log producers) in section 2.1.2. The visualization scheme is neutral with regards to the nature of the underlying data, and will only assume a minimal set of expected fields.

The information processing described in step 3 is covered in detail by section 2.2.4. Data addition and reduction techniques occur on tier 2, together with normalization.

The two remaining steps, visual transformation and the view transformation, are located in tier 3, and was party covered by section 2.2.4. With the foundations of visualization in place, the two are covered in more detail later in the following sections.

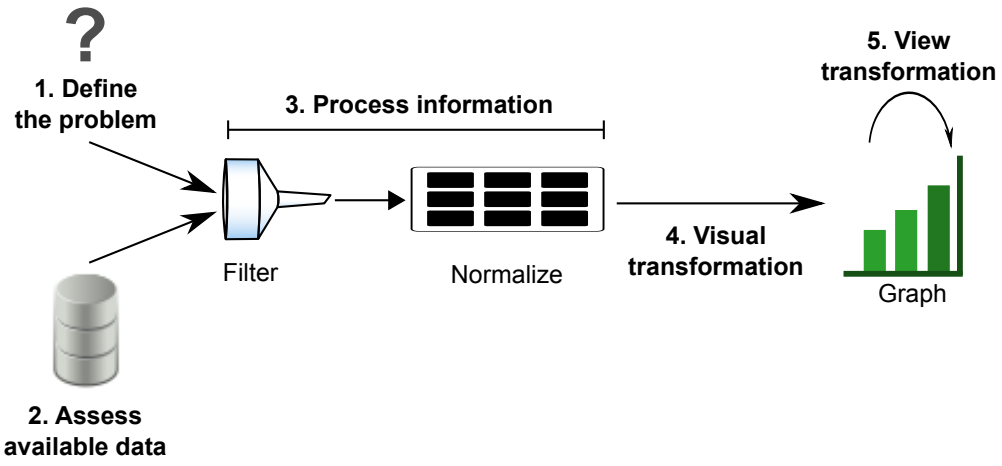

Figure 2.3: The information visualization process

2.3.2 Visual Transformation

The visual transformation is a mapping between raw data and a graphical representation [56]. The data is transformed into a set of shapes or images in a two dimensional plane, using spacial and temporal properties [38].

In the spacial domain, properties such as position, color, size and shape can be used to separate events. In the temporal domain, animation, data replay and association highlighting are potential tactics [41, 79]. In the context of this thesis, the goal of the transformation is to increase the value of the available screen estate, maximizing situational awareness for the operator.

A challenge when visualizing complex data sets is the *dimensional reduction*. Security incidents have dozens of interesting properties, but it's necessary for the visualization tool to determine which properties are needed to provide the necessary comprehension [21].

In their work on situational awareness for network intrusion detection, Livnat et al. presented *the w^3 premise*. This principle suggests three data dimensions considered to be required in a visualization scheme aiming to provide a satisfying level of situational awareness [50]:

> **The w^3 premise**
> Each event must have at least the *What*, *When* and *Where* attributes

'What' is a high level event classification, usually relating to the severity or category of the incident. 'When' indicates the time of the event occurrence, while 'where' describes the physical or topological location of the event. The user can see what's going on, where it is happening and when it happened, all within the context of a single visualization component.

If properly applied, this will provide the user with high situational awareness, and utilize the pattern seeking abilities of the human brain, allowing them to quickly discover anomalous behavior in the system [50, 69].

2.3.3 View Transformation

Static reports can be made to include both overviews and detailed components, but the key to balancing situational awareness and a detailed understanding of the system's inner state is *interactivity*.

The underlying data model of any visualization is a data element or data collection. When working with view transformations, it's common to envision the interface as a set of tasks, each task allowing the user to modify which parts of the underlying data collection to observe [69]. Different visualization paradigms can be used for different zoom levels.

Usability pioneer Ben Shneiderman proposed a taxonomy of important tasks in information visualization [68]:

Task	**Description**
Overview	Gain an overview of the entire collection
Zoom	Zoom in on items of interest
Filter	Filter out uninteresting items
Details-on-demand	Select an item or group and get details when needed
Relate	View relationships among items
History	Keep a history of actions to support undo, replay, and progressive refinement
Extract	Allow extraction of subcollections and of the query parameters

Table 2.3: Shneiderman's task taxonomy

Based on the task taxonomy, Shneiderman illustrated the main flow of a visualization interface by defining the visual information seeking mantra [68]:

Visual information seeking mantra
Overview first, zoom and filter, then details-on-demand

Chapter 3

State of the Art

This chapter introduces some of the practical applications of security log management, usability and visualization, in the form of techniques researched or tools implemented.

Section 3.1 presents the two main classes of log management infrastructures – syslog and SIEM – and refers to more comprehensive sources on how to implement them.

Section 3.2 introduces visualization components used in the security business, and outlines a few dashboard solutions developed academically or commercially.

3.1 Log Infrastructure

If more than one log requires monitoring, it's necessary to have a working log management infrastructure. As mentioned in section 2.1.2, three implementation approaches stand out: in-house development, SIEM tools and a syslog based tool chain. This section will examine some of the existing solutions available for the latter two approaches.

3.1.1 Syslog

To reiterate, *syslog* is a communication protocol describing a high-level log format and how to transport log data between different tiers [60]. In daily speech, syslog refers both to the protocol and the set of tools supporting it.

The first tier is already covered by the log producers. Most popular log producers can be configured to output syslog data, either locally or directly to a remote server [31].

On the second tier, the *syslogd* daemon is used to receive and consolidate the output from the log producers [3]. Syslog data is already normalized by definition, and can be filtered using configuration files. The syslogd tool is installed by default on most UNIX compatible operating systems, and fully functional ports are available for Windows based systems as well (e.g. syslog-win32 [2]).

The received log data is stored in flat files by default, but external tools can be used to store it in a database. Another alternative is to install one of the syslogd replacements. Due to the open standards, improved versions of syslogd are available, supporting enhanced features commercially or for free.

The most prominent syslogd replacements are *syslog-ng* and *rsyslog*. They both adhere to the standard, while providing features such as reliable TCP transfer (rather than unreliable UDP), TLS support for secure transfers (as specified by RFC 5425 [57]), database integration, content-based filtering, and support for proprietary formats [64, 66]. These tools can therefore cover both tier two and three in a single installation.

Setting up a log management infrastructure using syslog consists of modifying configuration files, and using open source tools likely to be preinstalled on the server in question. A detailed account of setting up a syslog based log management infrastructure can be found in [27].

Once the infrastructure is ready, adding new syslog tools is done by simple software installation, or in-house development using well supported libraries, such as Syslog4j [71] or the Python syslog library [23]. At the time of writing, the Syslog Wiki contained an overview of 58 commercial and open source tools, including event notifiers, log analyzers, correlation engines, and web logs, all ready to "plug-and-play" [1].

3.1.2 SIEM Tools

Security Information and Event Management (SIEM) solutions are centralized logging software, implementing the entire stack required to analyze real-time log data [46, 53]. As opposed to the open syslog toolkit, SIEM tools tend to be commercially wrapped, and often depend on proprietary standards.

SIEM has been embraced by several large security companies. ArcSight, a leading SIEM producer, was acquired by Hewlett-Packard for $1.5 billion in 2010 [39], allowing the computer giant to enter a market already inhabited by companies such as CISCO (Security MARS [72]) and RSA (enVision [63]).

While most SIEM tools are commercial, there are open source options, such as Cyberoam iView [15] and OSSIM Snare [59].

SIEM tools are usually complex, putting a great emphasis on rich presentation techniques and the ability to collect log data from as many log producers as possible, including official support for proprietary log formats. This is a benefit for large organizations, but may be perceived as an overkill for a simple Internet startup company. In any case, the end result should be equivalent to a syslog based solution, albeit easier to configure, with better support, and, in most cases, a high initial cost.

Due to the maturity of the log management infrastructure field, the rest of this thesis will assume a pre-existing infrastructure.

3.2 Visualization

Regardless of the underlying log management infrastructure, the information visualization process remains essentially the same. Its fundamental building block is the visualization component - a spatially confined representation of a specific data set. Components are usually combined into a presentation solution, as demonstrated in this section.

3.2.1 Standard Components

The most basic visualization components are **charts**. Charts use properties such as size, shape, and color to represent one or two dimensions of data, and have the benefit of being well known and easy to understand [56]. Common charts include line charts, bar charts, pie charts, and histograms.

While charts are appropriate for representing simple data (e.g. registered incidents per day), they are unable to cope with more complex problems, such as properly addressing the w^3 premise. Adjustments may be made, for instance by expanding the charts to **stacked charts**, but the benefits gained must be weighted against the disadvantage of a more complex presentation.

Charts, therefore, are recommended to present simple, categorical data, but should be supplemented by more advanced components when dealing with larger amounts of multi-dimensional data [56].

Several visualization components support multi-dimensional data. When working with relationships, as is common in network security visualization, the **link graph** is a handy tool. The link graph draws nodes connected by edges on a 2D canvas, and, if properly executed, provides an easy-to-understand illustration of communication patterns or data flow.

Another way to add dimensions is to use a **geographical map**. The map can illustrate spatially distributed data, and is a candidate for the 'where' attribute of the w^3 premise. The disadvantage of maps is their tendency to cause clutter in high traffic areas while leaving other areas empty, thus poorly utilizing the canvas [56].

A map can be combined with a link graph to create a spacial and temporal visualization of data, adhering to the 'what' and 'where' attributes of the w^3 premise in a single screen [38]. Using a two dimensional canvas allows significantly larger data amounts to be presented, and the result is often more aesthetically pleasing than lists of charts [56].

Examples of successful utilizations of the map/link graph combination for security visualization purposes can be seen in [36], [38], [49] and [50].

An alternative when working with multi-dimensional data is a **treemap**. Treemaps visualize hierarchical structures in data, by dividing tree nodes into blocks, and subdividing according to need [56]. In the security field, an example can be to use IP addresses for top blocks, and ports for subblocks, then fill each port block with color coded requests, e.g. green representing safe and red unsafe.

The toolkit of standard components is large, and will not be examined in further detail. For a comprehensive overview of the components mentioned in this section and more, Raffael Marty's "Applied Security Visualization" [56] is recommended.

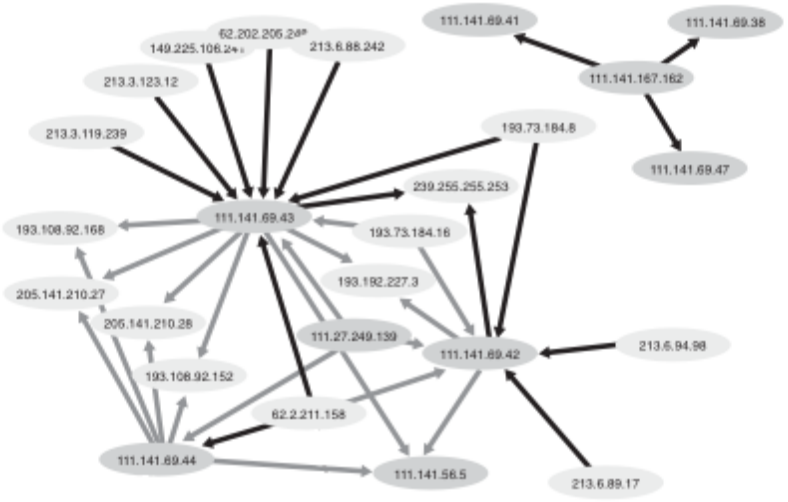

Figure 3.1: Example of a link graph

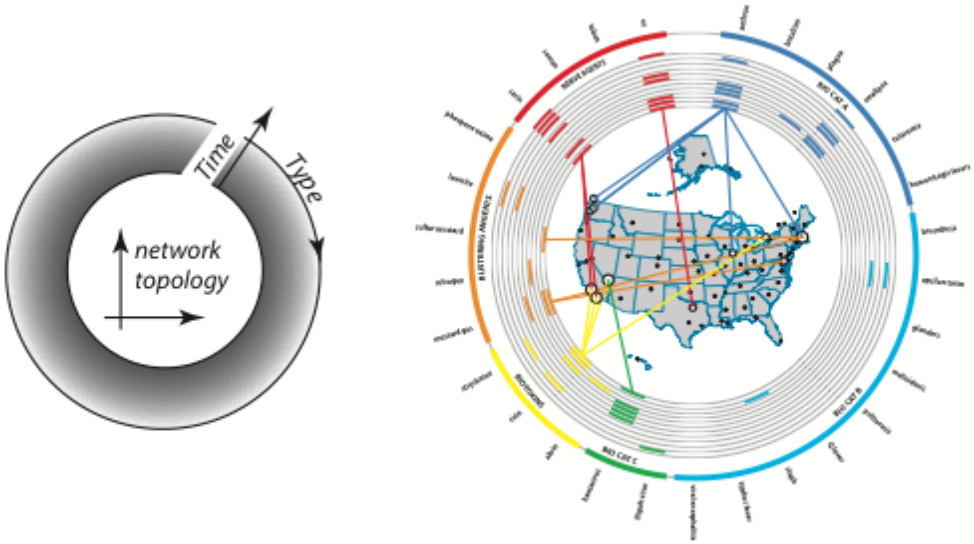

Figure 3.2: Livnat et al.'s visualization paradigm

3.2.2 Custom Components

As the security visualization field has progressed, practitioners have moved from the common toolkit and onto the more advanced practice of designing their own visualization components.

An example of this is Livnat et al.'s work on situational awareness [50]. Their paper introduced the w^3 premise (see section 2.3.2), and presented VisAware: a visualization component presenting the 'what', 'when', and 'where' of incidents discovered by network intrusion detection systems.

VisAware uses a circular visualization component, as illustrated in figure 3.2. The inner ring is dedicated to a map of geographical locations or network topology, while the outer ring contains a classification scheme of potential attacks on the network. Each incident is represented by two nodes, one in the inner and one in the outer ring, and a line connecting the two. The 'what', 'when', and 'where' attributes are mapped to line angle, line radius, and inner ring coordinates respectively.

Custom components are usually made to serve a specific purpose. More than 100 of these components are available at [55]. Inspecting them in detail is outside the scope of this thesis, but several components have been used in the design process of the incident wheel, which is presented in chapter 4.

3.2.3 Solutions

Components are useful when present log data, but it's unlikely that a single component can satisfy all the aspects of security usability. In order to satisfy the information-seeking mantra, a *solution* is introduced. A solution is a set of components, presented in a unified interface.

As previously mentioned, most SIEM tools are delivered with integrated dashboards. As an example, RSA enVision uses a web based dashboard, mostly depending on hypertext and standard charts [63]. Using the browser is a popular approach among commercial SIEM vendors, as it uses well known usability paradigms and make the logs universally accessible.

Similar tools exist for the syslog toolkit. An example is Kiwi Log Viewer, a Windows application using standard GDI components to visualize text based log files in a tabular, color coded format [47].

Specific dashboard solutions are also made independently of underlying log management infrastructures. A classical academic approach is James A. Hoagland's *Visual Audit Browser Toolkit*, which discusses and experiments with graphing and hypertext browsing in detail [41].

Architecturally, one of the most interesting approaches developed academically is Takada and Koike's *Tudumi*. Their proposed system is a client-server based data collector and visualizer, gathering data from several different sources and aggregating them in a single client visualization [73].

One of the architectural choices of Tudumi was to re-implement parts of the log management infrastructure in the tool, making it more holistic, but also more redundant. More specifically, the client fetches data from several servers, thus doing part of the work assigned to tier 2 in the log management infrastructure. It also employs a technique called "data summarization",

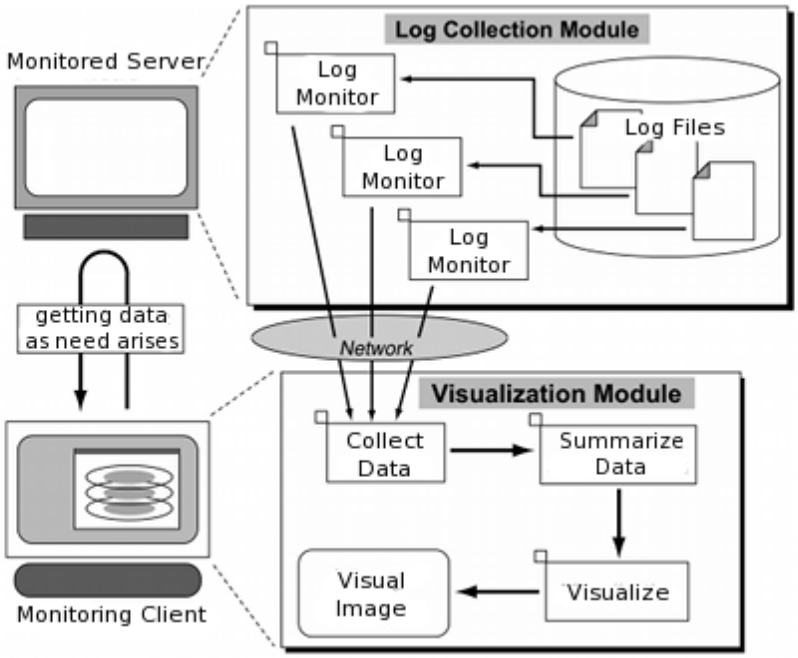

Figure 3.3: Architecture of Takada and Koike's Tudumi

which closely resembles the reduction technique presented as normalization in section 2.2.4 of this report.

The visualization of Tudumi focus on three activities: "accessing the server from other computers, logging in to the server and substituting a user to another user" [73].

After gathering all data, a visual image is created in the form of a three-dimensional spiral, illustrating the relationship between hosts and logged in users with lines and icons. The end result looks interesting, but is not likely to be intuitively understood by untrained users. In addition, the system scope is quite narrow, as opposed to the more general visualizers presented earlier in this section.

The architecture of Tudumi, which inspired the technical aspect of this report, is illustrated in figure 3.3.

More solutions are presented in [55] and [56]. These solutions have been used for inspiration, but will not be further examined in this thesis.

Chapter 4

LogWheels

This chapter contains a proposal for a new security log visualization system called LogWheels.

LogWheels is an interactive dashboard for remote observation of security incident logs. It lends itself to the theoretical foundations given in chapter 2, and is inspired by the state of the art research presented in chapter 3.

Section 4.1 presents the conceptual drivers considered in the development of the architecture and visualization scheme.

Section 4.2 outlines the system from a non-technical point of view, emphasizing the practical applications of the system and how it differs from existing solutions.

Section 4.3, 4.4, and 4.5 proceeds to walk through the details of the system, presenting the technical software architecture, the visualization scheme, and the user interface respectively.

4.1 Conceptual Drivers

Much effort has been put into the log management infrastructure, providing several well documented approaches to create a consolidated and normalized database of security incidents [54]. Several solutions are available for the visualization of data as well, but this fast growing field is still in need of new and creative approaches, guiding its practitioners towards a rich set of empirically tested, high quality visualization solutions [56].

The development of LogWheels was based on a purely theoretical foundation, borrowing principles from the fields of security usability and security visualization, but also human-computer interaction and user interface design.

A set of high level conceptual drivers can be extracted from the theoretical investigations in chapter 2 and 3:

- Increase the situational awareness of end users
- Address security usability principles C2-C4
- Present data in a real-time, interactive dashboard using visualization
- *Extend*, not *re-implement*, the log management infrastructure
- Visual transformation should adhere to the w^3 premise
- View transformation should adhere to the information-seeking mantra
- Navigation should be based on Shneiderman's task taxonomy

In addition to the theoretically based drivers, the system was adapted to small organizations, with simplicity and security in mind.

The intention was not to make a stand-alone product ready for commercial use, but to propose an architecture and a visualization scheme which can be embedded in other applications, or made part of a stand-alone toolkit (see section 7.2).

4.2 System Description

LogWheels is an interactive dashboard, allowing remote monitoring of security incident logs in near real-time. The system is appended to a pre-existing log management infrastructure, and works independently of the nature of the incidents.

A driving force of the system design was to make security *interesting* and *understandable* for organizations without dedicated IT security personnel. It was not designed to handle large and complex computer networks or huge amounts of log data, but challenges commercial SIEM tools on simplicity and usability.

The interface uses well-known usability paradigms, and includes a special purpose visualization component designed to correspond with the conceptual drivers listed in section 4.1. A deliberate choice was made to avoid an enterprise look-and-feel, at the expense of a fresh and intuitively appealing approach. If security monitoring stops being a chore, the situational awareness should be drastically improved.

Due to its use of standard Internet protocols and file formats, the system interoperates with existing tools created for the World Wide Web, including web browsers. Users can connect to a LogWheels server using any Internet enabled device. Data is transferred in real-time, but it's possible to review former states. All connections used between the client and the server are secured, preventing eavesdropping and data manipulation.

While the server does not intend to re-implement the consolidation efforts of the underlying tiers, it provides basic configuration opportunities, allowing rough filtering based on severity levels and log sources. Most of the configuration is done client side, supporting the creation of different "nodes" serving different needs. For instance, a home computer can be setup to display only emergencies, while an office monitor can be used to display all warnings occurring in the system.

A high-level conceptual model is illustrated in figure 4.1.

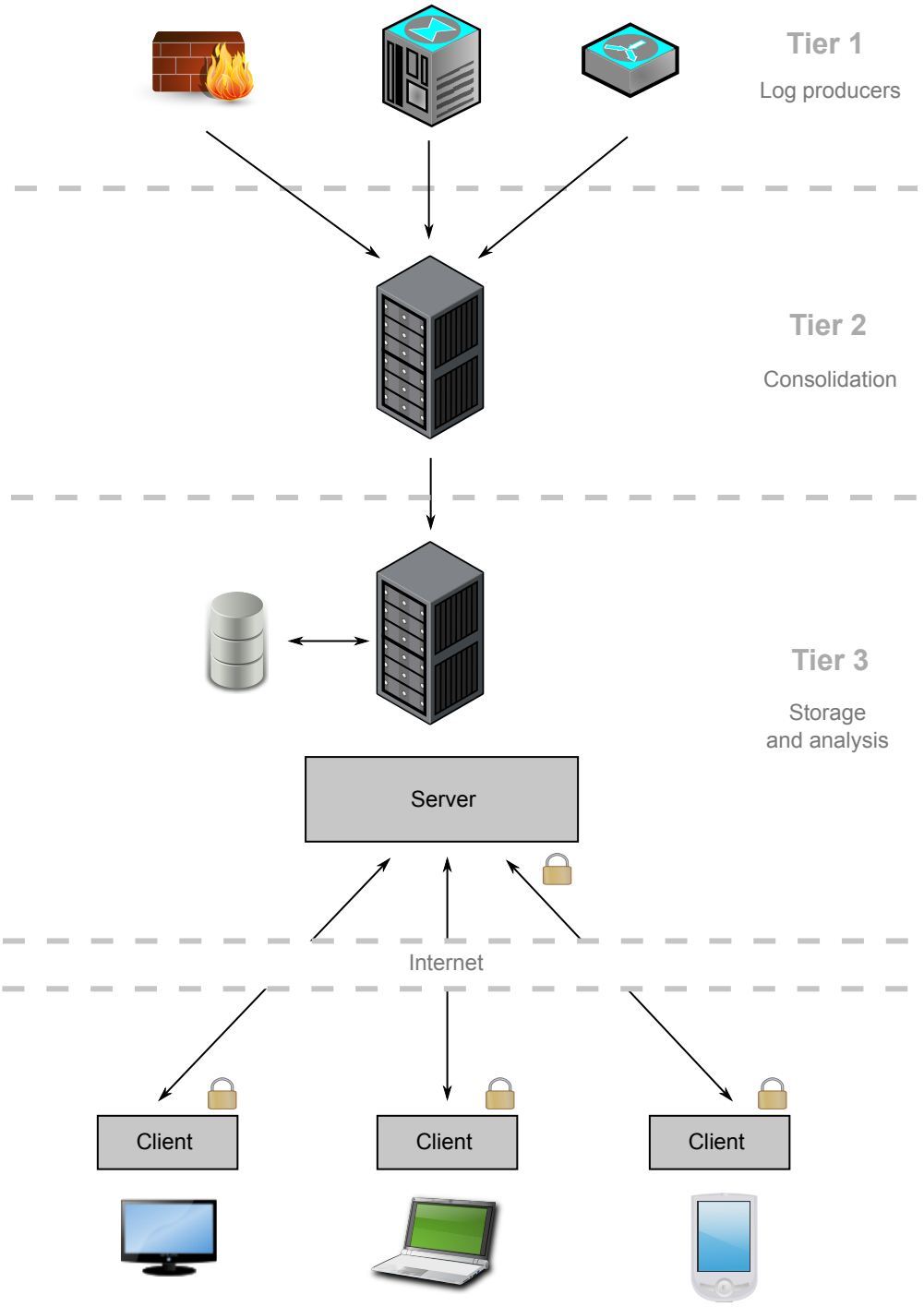

Figure 4.1: Conceptual model of LogWheels

4.3 Architecture

Before introducing the visualization scheme, the suggested architecture for LogWheels will be briefly described. Readers who are only interested in the visualization scheme can safely skim through this section.

4.3.1 Overview

As illustrated in figure 4.1, LogWheels is based on the client-server architecture. A server module is located at tier 3, with direct access to the storage used by the log management infrastructure, while the client can be executed on any device allowing custom software execution and Internet access.

A key design decision was to use the HTTPS protocol for communication between the client and the server. This choice partly lends itself to the advantage of using a trusted industry standard, which may ease the process of using the application through firewalls or other network systems, and partly allows the system to utilize existing software written for the World Wide Web infrastructure, raging from web servers and browsers to proxy servers, web accelerators, web cache et cetera.

The server can be implemented as a stand-alone application, or be an add-on to any modern web server through the use of server-side scripts. Similarly, the client can be stand-alone or be implemented through a web browser, through plugins (e.g. Adobe Flash or Flex) or using web technological standards such as HTML and AJAX.

Security is provided through the use of the secure HTTPS standard, in addition to Basic Access Authentication. Data is transferred using the popular format language JSON, which has extensive library support.

The client-server architecture is inspired by Tudumi (see section 3.2.3), with a few significant changes. First, LogWheels extends the log management infrastructure, without making any efforts to re-implement it. Second, it uses standard formats to maximize interoperability.

The following section presents these architectural choices in detail.

4.3.2 Modules

LogWheels consists of two separate applications: a server and a client.

Server

The server is a thin wrapper, accepting requests from the client and serving log records back. It assumes the existence of a consolidated log database, and its only function is to transmit these records to the client. Simple data structure translation and addition of external data can be added, if it's infeasible to include these features in tier 2 of the infrastructure.

The current LogWheels specification does not describe a stand-alone application, and must be tailored to work with an existing tool chain to be a deployable production system. The final step of the roadmap presented in section 7.2 is to integrate the solution with a log management infrastructure.

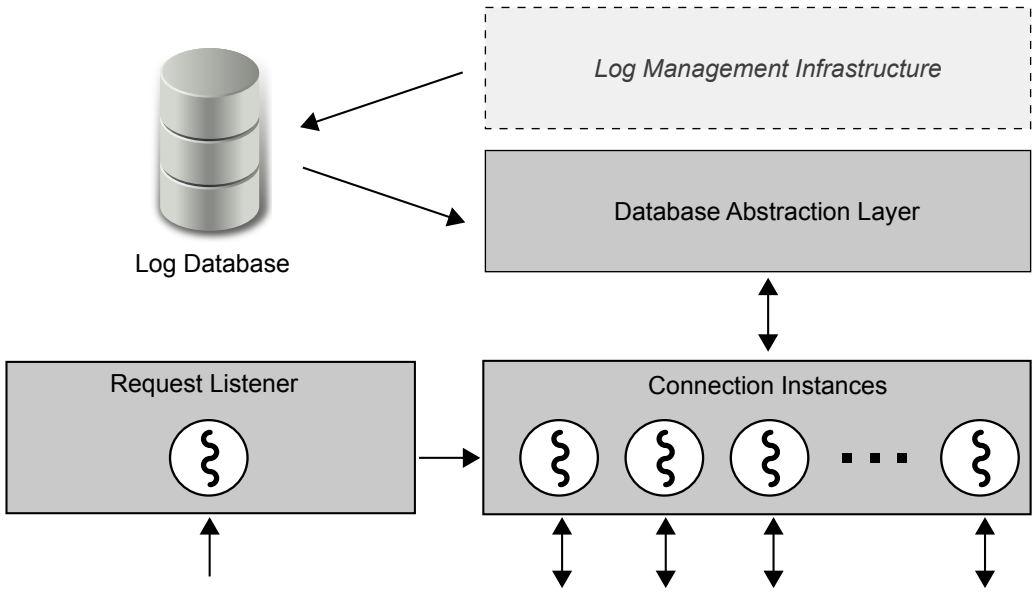

Figure 4.2: Server architecture

The server has a dedicated thread listening for incoming request, and generates new threads for each connection. The protocol is stateless, so it's not a requirement to keep a session going for more than one request, but as discussed in section 4.3.5, it's recommended to use the HTTP keep-alive mechanism to reduce connection overhead [10].

Each connection thread serves one purpose: to access and retransmit log records through a database abstraction layer. The complexity of this layer depends on the underlying infrastructure.

Since the protocol used is HTTPS, a server can be implemented using a scriptable web server, such as Apache or Microsoft IIS [24]. A minimum implementation of the server is given in appendix C, in less than 100 lines of PHP code. Using libraries for database access and HTTP transportation, it should be possible to implement comparably sized servers in other languages.

Client

The client is responsible for requesting log data and presenting it to the end user. Different clients can be constructed for different devices, all they need is the ability to read plain text data over a HTTPS connection.

The recommended visualization scheme requires graphical abilities which may be hard to implement directly in HTML, but other presentation mechanisms can be created within the same architecture. The upcoming canvas technology is likely to provide the functionality needed to implement the visualization directly in a browser [40].

All configuration is done on the client. The user can choose how often to request new data, and how severe a log entry must be to be transmitted. This allows the end user to create "nodes" with different profiles without any modifications to the server.

The architecture of the client will depend on the implementation platform, and no requirements are made.

4.3.3 Communication

All communication between the client and the server is executed using the HTTPS protocol. Choosing the common protocol rather than designing a special purpose protocol was a key design decision in this system.

The Hypertext Transfer Protocol (HTTP) was first documented in 1991, and is currently standardized through RFC 2616 [10]. The protocol is the backbone of the Internet, and much of the traffic between web browsers and servers are performed using it.

HTTPS is the secure extension of HTTP, and extends HTTP by channeling all data through the secure TLS protocol [62]. This security enhancement is described in section 4.3.6.

The main advantage of using HTTP(S) rather than designing a special purpose protocol, is the opportunity to utilize the already existing infrastructure of the Internet. This opportunity is further examined in appendix C.

Client Sessions

A challenge when using HTTP(S) is the stateless request/response nature of the protocol [10]. For a near real-time, session based system, it's natural to consider a stateful push-protocol, where data is pushed from the server to the client through a persistent connection channel.

Bozdag et al. described how a push protocol can be implemented over HTTP, using a technique called long polling, where pending requests are responded to when new data appears [12]. But even though this protocol intuitively is closer to LogWheels' nature, it's not necessarily an improvement to the system.

On the contrary, it turns out using a stateless pull protocol has some useful side effects, primarily a reduction of server complexity and simpler communication. At the same time, it maintains client customizability through the use of request parameters. If real-time capabilities are absolutely required, the pull interval can be set in the subsecond range, and the overhead is still

unlikely to be an issue due to the low number of expected connections. As the LogWheels client only provides visualization intended for humans, hard real-time response requirements are unnecessarily complex.

In order to keep interoperability with web servers and browsers at an acceptable level, it was chosen to avoid too large deviations from the original standard.

The biggest performance penalty is the overhead of frequent HTTPS connection establishment [6]. This can, in part, be solved using the HTTP keep-alive flag [10]. The technique increases memory use on the server, but reduce the necessary bandwidth and processing power.

Initialization Routine

Despite the stateless communication, an initialization routine is recommended. This routine is executed on application level, and its only benefit is increased usability for the end user. The three-step routine is illustrated in figure 4.3.

The first step consists of the client requesting a certificate and displaying its content to the user. The certificate details are displayed on the screen, together with a color coded warning specifying whether the certificate is trusted, untrusted, or potentially revoked.

After validating the certificate, the user enters a username and a password. The authentication details are validated by the server, and if valid, stored in the client memory to prevent the user from having to repeat the authentication procedure on each update. This leaves the application vulnerable to memory scan attacks [52], and the developers of a production system ought to look into ways of hiding the authentication details.

Once the authentication process is complete, the rest of the session consists of the client asynchronously requesting updates from the server. The user is not required to verify the certificate or enter authentication data for the duration of the process, but behind the scenes, the client is strongly recommended to pay attention to the certificate on each request, and abort the connection if it changes. This is discussed further in section 4.3.6.

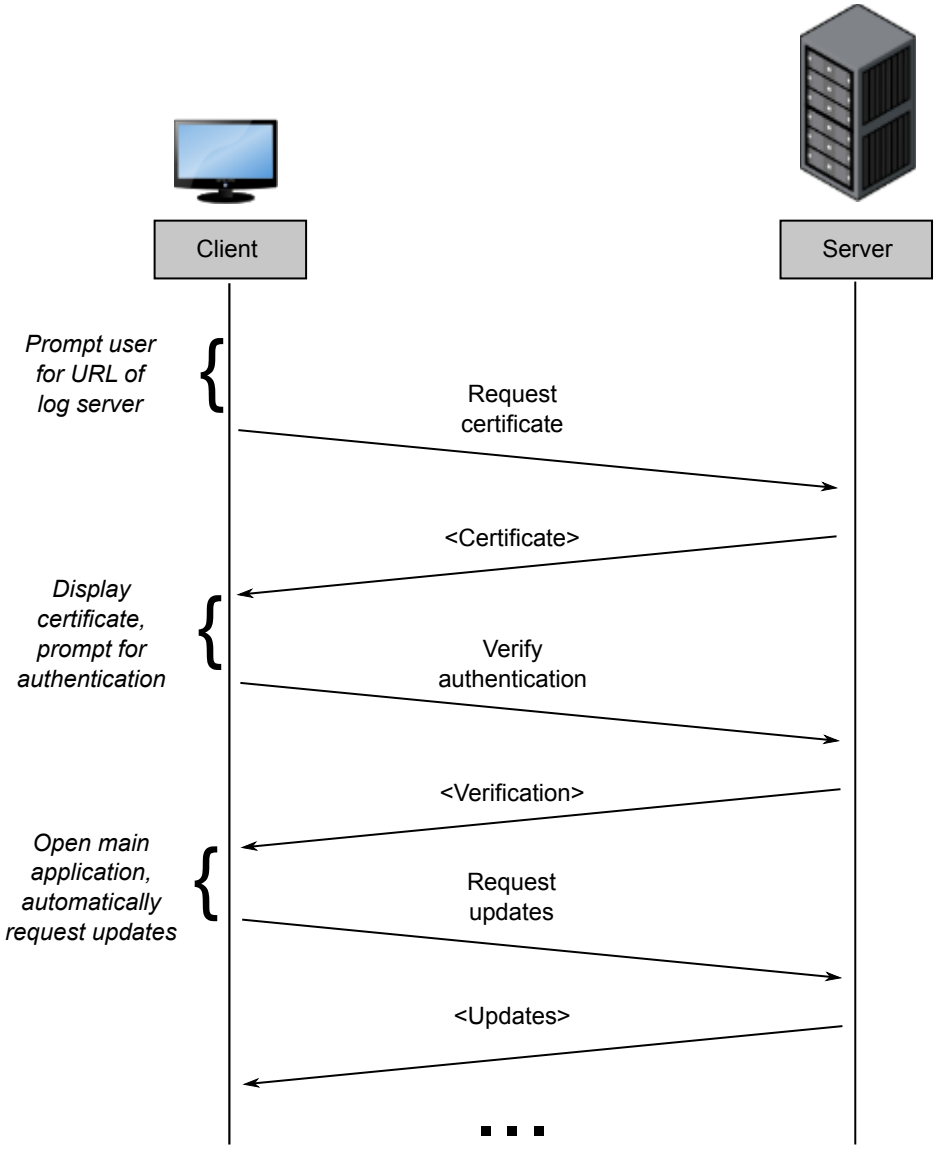

Figure 4.3: Recommended initialization routine

4.3.4 Data Structures

Requested logs are transferred between the client and the server using the data interchange format JSON (JavaScript Object Notation). JSON is standardized through RFC 4627, and is based on the principles of minimal data overhead and easily readable formatting [14].

Compared to the industry standard XML, JSON reduces the markup overhead, and allows even simpler parsing, through standard libraries. The added functionality offered by XML is not needed by LogWheels, as the only data transmitted is similarly formed log records.

A simple example of a JSON file can be seen below:

```
{"name": "John␣Doe", "age": 40, "manager": true}
```

An advantage of using JSON is its derivation from JavaScript literals, which allows simple deserialization both to Java beans (e.g. using google-gson [34]), or JavaScript objects (e.g. using jQuery [43]). This facilitates for seamless integration with both a Java front-end and a browser based client.

Data Fields

Each log record consists of fields, holding key attributes of the incident. It is important to keep the number of mandatory fields to a minimum, in order to allow different underlying log management infrastructures to be supported.

The terminology used is derived from the syslog standard [31, 51]. A complete overview of syslog fields, and a database scheme for the extended rsyslog daemon, can be found in appendix D.

Different log producers will produce different log formats. In order to include as many formats as possible, the log format used by LogWheels is kept simple. This is the format exchanged between the client and the server, intermediate systems may use different formats.

This solution should not be responsible for consolidating different log formats, but will support translation from common repositories to its internal format.

Name	Status	Description
id	Mandatory	Numerical, unique identifier
timestamp	Mandatory	UNIX timestamp
severity	Optional	Importance of event, as listed below
facility	Optional	Name of source application/service
ipaddress	Optional	IP address of the threat source
message	Optional	Textual description of the incident

Table 4.1: Supported JSON fields

The available fields are listed in table 4.1, and described in the list below:

id A unique numerical identifier for each incident. A simple auto incremented integer is sufficient. Notice that the id should be unique on the entire system, so it should be assigned by the server wrapper, not imported from the log producers.

timestamp Indicates when the incident was registered by the log producer. It's denoted as a UNIX timestamp, which numerically represents the number of seconds since the "UNIX epoch" at midnight of January 1, 1970, UTC [19].

severity A numerical value derived from the syslog standard. Some log producers assign their own values, depending on internal configuration. If the log producer does not specify severity, the log management infrastructure or the server wrapper can determine its value based on other metrics associated with the incident. It could also be possible for users to configure severity modifications, for instance stressing the importance of one log producer. The severity levels from RFC 5424 are given in table 4.2 [31].

facility The source of the event, given as a textual name. Syslog provides a list of facilities, which is supported by many popular log producers, but the field does not have to correspond to the syslog standard. Any name can be given, representing an application, such as "sshd", "intrusion detection system", or the name of an internal network component. This field is exclusively used for filtering and searching on the client side, and

Severity	Description
0	Emergency: system is unusable
1	Alert: action must be taken immediately
2	Critical: critical conditions
3	Error: error conditions
4	Warning: warning conditions
5	Notice: normal but significant condition
6	Informational: informational messages
7	Debug: debug-level messages

Table 4.2: Syslog severity levels

is not used to prioritize incidents, unless specified on the server side. A complete list of syslog facilities can be found in appendix D.

ipaddress IPv4 or IPv6 address of the user agent suspected of causing the incident, given as a textual dot-decimal notation [17, 61]. If no address is provided directly by the source application, attempts can be made to extract it from the textual message. A complete lack of IP addresses should cause the incident to be registered as an internal event. This is discussed in chapter 6.

message A raw textual description of the log record. If it's desired to reduce traffic, the message can be left empty, forcing the client to request the message specifically. This reduces traffic overhead by preventing the transfer of large textual messages that are never displayed to the end user.

A simple example of a response is listed below:

```
{"id": 500, "timestamp": 1301753520, "facility": "sshd", "severity": 2,
 "ipaddress": "192.0.32.10", "message": "Failed␣login␣from␣[...]"}
```

The JSON representation can be converted to one of many internal data structures, such as an object or an associative array, using library functions.

4.3.5 Protocol

The protocol used is a subset of the HTTPS protocol. It's highly recommended to use HTTP libraries for this communication, rather than relying on a re-implementation of parts of the HTTPS protocol, to avoid a reliance on the current prototypical specification and support communication with third party software, such as web browsers or servers.

Three primitives are required: 'version', 'update', and 'detail'. All primitives are stateless, and contain the authentication string and necessary parameters. They return an HTTP status code [10], in addition to content, if available.

Primitives are sent as GET requests to a base URL entered by the user, as specified by RFC 1738 [11]. An example is given below:

```
https://192.168.10.1/?action=update&timeFrom=1301753500&timeTo=1301754700
```

The initial URL entry by the user should be a base URL – which in the above example would be "https://192.168.10.1/". The additional parameters are added by the application when required. The 'action' parameter is alway mandatory, and is used to separate between different primitive actions. Each action may require further parameters to valid.

The 'version' primitive serves three purposes. First, it verifies that the client is talking to a LogWheels server (or a destination claiming to be a LogWheels server). This is not an act of security, but an act of error prevention. When the user enters a URL, this request can verify that the URL exists and hosts a server. Second, it can be used to verify the authentication string (the request will fail if the username or password is wrong). Third, it returns a version number, which can be used to provide backward compatibility in future versions of the application.

The 'update' primitive will return all incidents recorded in a specified time frame, exceeding a specified severity level. The LogWheels client will have a configurable polling interval, and at the end of every interval, it should request new updates:

$$(lastTimestamp, lastTimestamp + pollInterval]$$

This will ensure that all incidents are returned once and only once, regardless of changes to the polling interval.

Finally, the 'detail' primitive is used to get a full log record. The server can choose to return full records in the ordinary 'update' request, but if the polling interval is long or the data amount large, the textual message can be left out. The client must then ask specifically for the full record every time the user attempts to view the record details.

Request	
action	version
Response content	
Version string	

Table 4.3: Version request

Request	
action	update
timeFrom	timestamp
timeTo	timestamp
(minSeverity)	severity (int)
Response content	
JSON log records separated by newlines	

Table 4.4: Update request

Request	
action	detail
entry	id (int)
Response content	
JSON log record	

Table 4.5: Detail request

4.3.6 Security

LogWheels makes two security enhancements to the default behavior of HTTP. The first is the use of HTTPS (secure HTTP), which negotiates an encrypted communication channel and provides protection against eavesdroppers and man-in-the-middle attacks [62]. The second is the use of Basic access authentication, which extends HTTP or HTTPS with stateless client authentication [26].

HTTPS

A disadvantage of the HTTP protocol, is that it only supports unencrypted communication [10].

The increased need for sensitive applications on the web created the demand for a secure HTTP standard. A first edition was first developed by Netscape in 1994, then using SSL encryption, a protocol which was formally specified in 2000 using TLS [62]. The concept is simple: regular HTTP traffic is run over the secure Transport Layer Security (TLS) protocol. The transport layer TLS protocol is designed to prevent eavesdropping, tampering, or message forgery [18].

While the prototype accompanying this report supports plain-text HTTP communication, it's important to note that this scheme should **not** be considered secure, and never be used in a production system. HTTP support can be disabled in the source code by modifying a single line of code in the Configuration class.

Certificates

When using the HTTPS standard, public key certificates are used to identify the server. Using public key cryptography, the server transmits a digitally signed document, containing an identifier and a public key [33].

Ordinary web sites relies on a web of trust, where more trusted entities are used to verify less trusted entities, with a handful of certificate authorities

serving as root nodes [13]. LogWheels can use self-signed certificates, as long as a secure distribution channel for keys is found. For small organizations, this is no problem, as the keys can be distributed offline. By ordering the client only to trust servers with known certificates, man-in-the-middle attacks and impersonator servers are mitigated.

LogWheels uses standard X.509 certificates. The client should not require revalidation of the certificate on each request. If, however, the certificate is modified during a single session, a visual indicator should warn the user, as this may indicate an attack.

Basic access authentication

Basic access authentication was specified by RFC 2617 [26], and works by appending a header containing a base 64-encoded username and password in every request. Used in conjunction with HTTP, the authentication data is sent in plain text, which adds little security, but when used in combination with the encrypted HTTPS standard, the standard is a secure transportation of authentication details.

This authentication mechanism has been in use for many years on the World-Wide Web. It is included in most modern browsers, using a dialog based login scheme, and can easily be configured into most modern web servers, such as Apache (through mod_auth_basic [25]).

By default, basic access authentication is vulnerable to eavesdropping and repeat attacks, but these threats are mitigated by the use of HTTPS [28]. Its main weaknesses are the use of a single authentication string which can be used indefinitely by a malicious user if compromised.

A newer standard, called Digest access authentication, was defined by RFC 2617, and is based on sending cryptographic hashes and nonce values rather than the easily reversible base 64-encoding [26]. However, the need for hashing the authentication string is of less importance when used in combination with an encrypted communication channel, and the protocol never caught on in real-world applications [28].

4.4 Visualization

While the architecture is rather conventional for a client-server application, the novelty of the solution lies in the visualization scheme.

The goal of the visualization process was to increase situational awareness in a user friendly manner. In order to assure that a minimum of necessary data was available, the w^3 premise was used. In addition to the chosen combination of existing visualization components, this section introduce the incident wheel, which is a new addition to the visualization scene.

4.4.1 World Map

The dominant component in the visualization solution, and the backdrop for the incident wheel, is a world map. The map covers the 'where' attribute of the w^3 premise, by mapping security incidents to geographical locations.

Geographical location is a simple classification of incidents, but the classification itself is not the main driver to use a map. In distributing the incidents on a canvas, the pattern seeking abilities of the human brain is being utilized, using an easily recognizable component [50].

Projection

When representing the three-dimensional globe in a two-dimensional plane, there are several projections to choose from. Some are area conserving, some are distant preserving, other are compromises between these or other projections [70].

For this application, the most interesting option is the *equirectangular projection*. In this projection, the lines of latitude and the meridians of longitude are both straight, parallel, equidistant lines, forming a perfect grid of rectangles [70]. These properties are beneficial in computer visualizations, as the relationship between a longitude-latitude pair and a map position is always a simple, linear transformation.

The disadvantage of forcing an almost perfectly circular area into a rectangular grid, is the poor area preservation [70]. While the equator has a circumference of roughly 40 000 kilometers, this figure declines until hitting the pole points, where the circle of latitude is a single point. As all latitude lines are mapped to the length of the image projection, this causes distortions to areas away from the equator.

While the sociopolitical consequences of distorted map projections have been discussed, and usage of the projection in navigation dismissed, the side effect can, in fact, be beneficial to a security visualization [35]. The areas decreased are primarily in Africa and Central America, while countries such as South Korea, China, and the United States have their areas exaggerated [9]. These hotspots will therefore tolerate more traffic without being cluttered.

Geolocation

A security incident is converted to a geographical location by the use of a technique known as *geolocation*. While serious proposals have been made to incorporate location information in the Domain Name System (see RFC 1876 [16]), the closest approximation today relies on active probing of single hosts, or building tables based on IP address ranges [37].

No geolocation scheme is bulletproof. This is partly caused by inaccuracies or gaps in the tables themselves, partly by the habit of attackers to use proxies, botnets, or zombies to execute their attacks [37]. As long as the incidents are distributed across the map, this should not interfere with the pattern seeking abilities, nor should the incidents position in a map alone determine the course of action for the operators.

Implementation

The geolocation process, which is illustrated in figure 4.4, consists of three main steps. First, extract an IP address from the log record, second, convert the IP address into a geographical location, and third, transform the geographical location into a location on the screen.

The map is placed in a two-dimensional cartesian reference frame, with the origin located in the upper left corner. This is a standard viewport for computerized graphics [8]. A translation function can be defined:

$$f : (longitude, latitude) \rightarrow (xCoord, yCoord)$$

An equirectangular map has the 0 meridian (Greenwich) located in the center, and spans +/- 180 degrees. Similarly, equator is located in the center, spanning +/- 90 degrees [70]. This gives f a domain: [(-180, 180),(-90,90)] and a co-domain: [(0, imgWidth), (0, imgHeight)], resulting in the function:

$$f(x, y) = [(imgWidth/360) * longitude + (imgWidth/2)),$$
$$-(imgHeight/180) * latitude + (imgHeight/2)]$$

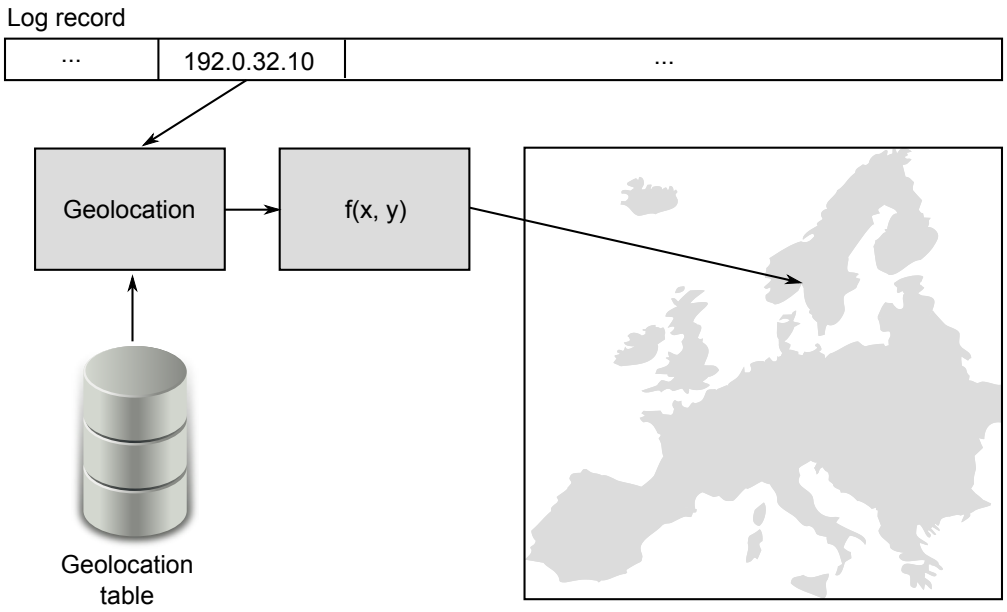

Figure 4.4: Geolocation and transformation

4.4.2 The Incident Wheel

The world map provides a spatially distributed view of events, classified by their geographical properties. It is, however, not sufficient for presenting security incidents [50]. The map lacks the 'what' and 'when' attributes of the w^3 premise, and experiences heavy cluttering once several incidents occur in the same general area.

One of the contributions of this thesis is a new visualization component – the incident wheel – which attempts to satisfy the w^3 premise within the context of a world map. The incident wheel is a representation of a set of security incidents occurring in the same geographical area, and presents the severity and quantity of the incidents in the area. Through the use of zooming and panning, the user is free to choose the granularity, which allows the incident wheel to provide information about areas ranging from continents to single cities or even neighborhoods.

The incident wheel has three properties, derived from Livnat et al's w^3 premise [50]. These are defined as follows:

(where) $\rightarrow$ (position in map)

(what) $\rightarrow$ (color/icon)

(when) $\rightarrow$ (angle)

Where Instead of placing security incidents directly in the map, the occurrence of an incident will either spawn a new incident wheel, or attach the incident to a previously created incident wheel, depending on whether a wheel was covering the location of the new incident or not.

The wheel spans a certain geographical area, determined by the zoom level of the surrounding map, and contains all incidents which occurs within that region in a defined time frame. Once the last incident leaves the time frame, the incident wheel disappears. As a consequence, the world map will, at any given time, be populated by a set of incident wheels, all containing one or more incidents.

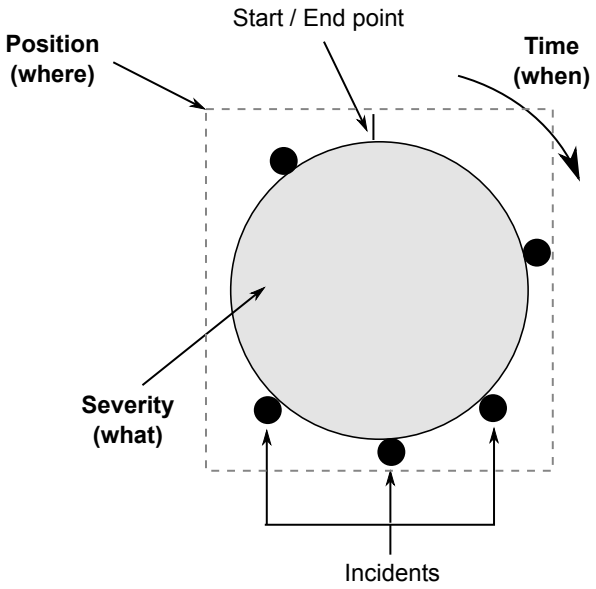

Figure 4.5: The incident wheel (not to scale)

What As indicated by the name, the body of the incident wheel is a circle. The inner area of the circle is used to give a rough classification of the incidents the wheel contains. Since the wheel can contain several different incidents, the classification must be a function converting an incident set to a single parameter.

The recommended function is maximum severity, that is, how severe is the worst incident in this area. This allows users to rule out insignificant incident wheels, and rather investigate the serious ones further.

Within the limited space, a small icon or letter can be placed in the center of the circle. Alternatively, color can be used to separate different classes. An example can be to place the number of incidents in the wheel, as it can be hard to count incidents in a populated area.

When Along the circumference of the circle, smaller dots are used to represent incidents. Each incident contained by the wheel is given a separate dot. This gives a visual representation of incident quantity in a geographical

region: the more dots, the more incidents.

When an incident occurs, the dot is placed at the top of the circle. As time passes, the dot rotates clockwise around the circumference, until it ends up in the topmost point again, and disappears. This mechanism has the advantage of using the well known clock metaphor, and the visualization can benefit greatly from using an interval corresponding to one of the clock hands, such as 12 hours, 60 minutes, or 60 seconds.

The movement is given by a simple parametric function:

$$(x, y) = (-r\cos\left[\tfrac{2\pi t}{T} + \tfrac{\pi}{2}\right], r\sin\left[\tfrac{2\pi t}{T} + \tfrac{\pi}{2}\right]), t = [0, T]$$

The time frame T depends on how many incidents are registered by the system, and should be configurable or automatically adjusted by the system.

Example An incident wheel is illustrated in figure 4.5. The wheel is shown out of context, but would be placed in a map to illustrate the geographical position of its contained incidents.

This wheel contains five incidents, each represented by a black dot. The top left dot is the oldest incident, which will disappear as soon as it hits the topmost mark. The rightmost dot is the most recent. Assuming a one hour rotation cycle, the incidents happened roughly 13, 22, 29, 38, and 54 minutes ago. The color of the central area can be used to indicate the severity of the most severe incident, leaving a quick impression of whether this is routine log records or if something serious may have happened.

The combination of the world map and the incident wheels thus fulfills the w^3 premise [50]. By a single glance on a computer screen, it is possible to detect *where* security incidents have occurred in a given time frame, *when* they happened, and *what* they are, in terms of severity to the system being observed.

This component therefore serves as a complete implementation of the overview stage on Shneiderman's information seeking mantra, thereby utilizing the human brain's perceptual abilities [69].

4.4.3 Threat History

The world map and incident wheel fulfills the w^3 premise, but does not support all of Shneiderman's recommended tasks. The main weakness of the overview is its limited time frame. If the time span is too long, the map will clutter up. Unfortunately, in most networks this time frame will be shorter than the intervals at which logs are observed.

For example, if the specified time frame is 60 minutes, someone must look at the screen at least once per hour all around the clock to avoid missing incidents.

The solution is to allow investigation of former states. The 'history' task can be satisfied if the user is given a brief overview of the system state over time, and is allowed to investigate these former states. Watching a former state replaces the real-time map with the state of the system at the specified moment. Once the state has been investigated, the user can go back to 'live mode' for an animated overview of the current status.

To represent the threat history, a simple chart is sufficient. A histogram or a line graph can both be used. The time frame should be significantly longer than the time span of the incident wheels, and should give a reasonable time for user observations. A good example is 72 hours, which allows an incident to occur on Friday afternoon, and still be discovered by Monday morning.

4.4.4 Gauge

Finally, the simplest component used in this solution is the gauge. A gauge is a simple measure of activity in the system, and it works by adding up and displaying the number of incidents in the incident wheel time frame. If the map gets cluttered, the gauge can be used to see a quantitative representation of activity in the system.

The main advantage of the gauge is the simple pattern seeking advantage. Users will quickly learn what a normal number is, and will be alerted if the current value deviates from the norm.

4.5 User Interface

The previous section described the visual transformation, this section proceeds by walking through the possible view transformations. LogWheels uses Shneiderman's information seeking mantra, and creates three panels, one for each phase: overview, zoom-and-filter, and details on demand.

4.5.1 Overview

The overview screen contains the three custom visualization components: (1) the world map with incident wheels, (2) the threat history, and (3) the gauge. A recommended setup is provided in figure 4.6.

In this view, the user should be able to get an overview of the entire collection, limited by the pre-determined time frame [69]. It also provides basic filter-and-zoom capabilities, by allowing the user to change the time frame to study or using zoom-and-pan to determine the incident wheel granularity (i.e. change which parts of the current incident set which should be in focus).

The following interactivity should be supported by the overview panel:

- It should be possible to zoom and pan the map, using industry standard mouse motions. The scroll wheel should zoom, while dragging the map around should pan. When the viewport is modified, incident wheels must update, and include only the events occurring in their new area. Thus, zooming in on an incident wheel may cause it to split in to several wheels, each spanning a smaller geographical area.
- Clicking on the threat history should cause the map to change into a historical view, showing the status at the moment the user selected. Clicking in the rightmost area of the history graph should force the application back in live mode, where wheels are displayed in real time.

The overview provides a starting point, in addition to provide simple filtering in the spacial and temporal domain.

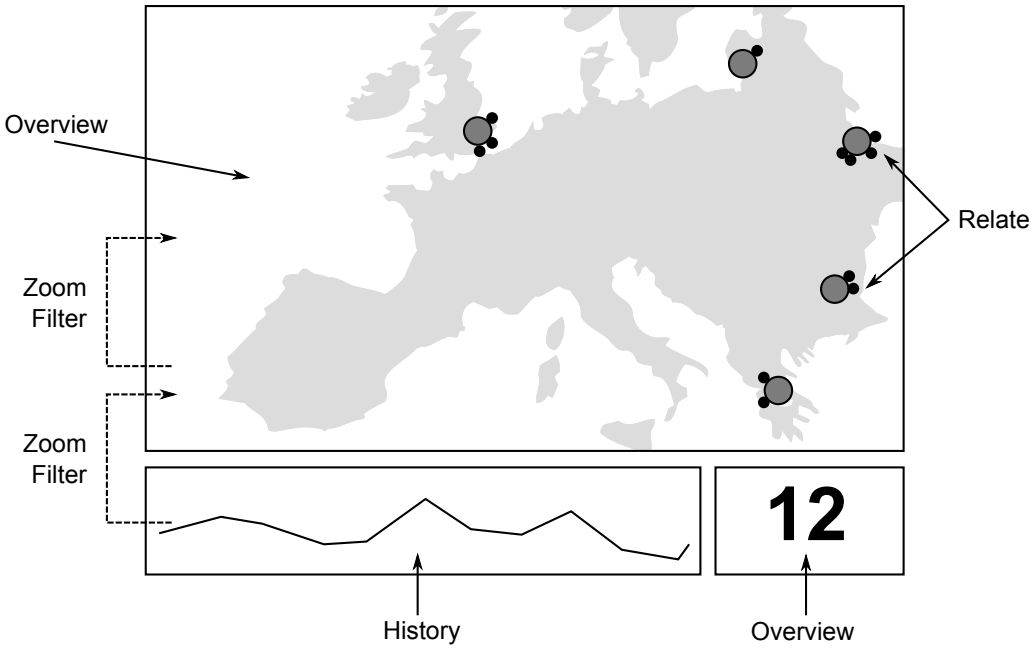

Figure 4.6: Overview panel with tasks

4.5.2 Filter View

The filter view corresponds to Shneiderman's zoom-and-filter, and allows detailed filtering to be performed [69].

By default, the view should display the largest amount of incidents which is reasonable to keep in memory. A simple list displays the key parameters of the incidents, and allows basic operations such as sorting and scrolling.

In addition to the list, a search panel allows the user to enter parametrical constraints, and filter the list. Table 4.6 provides a minimum set of parameters to be supported.

It should be possible to combine some or all filter constraints. By default, the value of each component should not affect the list, so that the list is only filtered by the constraints directly entered by the user.

No interactivity is required, besides the opportunity to input search terms and perform a search action.

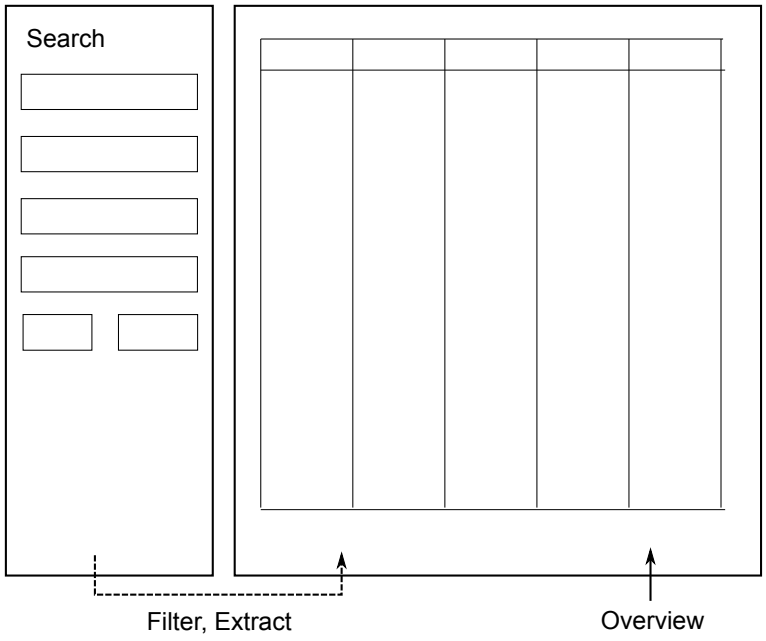

Figure 4.7: Filter panel with tasks

The purpose of the filter view is to allow a more detailed investigation of incidents than the overview panel can. Using the list component allows more data to be presented at once at the cost of a less attractive visualization and the disappearance of pattern seeking opportunities.

The filter view is illustrated in figure 4.7.

Name	**Example**
IP prefix	"192.168."
Time between	Two dates and times
Minimum severity	"Warning"
Facility	"sshd"
Message contains	"root"

Table 4.6: Filter constraints

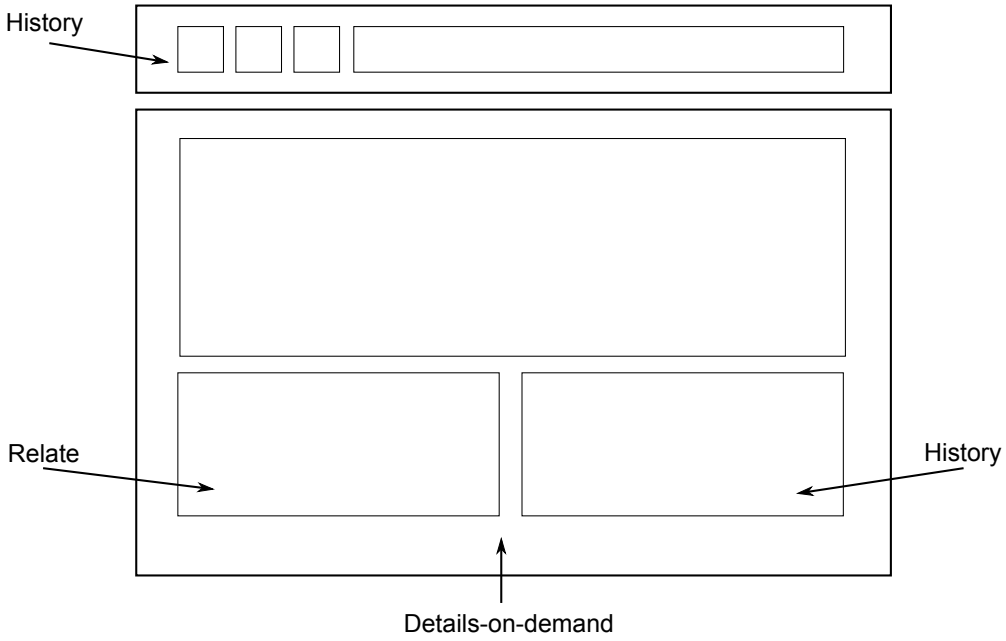

Figure 4.8: Detail panel with tasks

4.5.3 Detail View

The detail view corresponds to Shneiderman's details-on-demand [69]. When a single security incident is chosen, a report is generated, containing all raw data about the event, in addition to any relevant information from third parties, incidents suspected to be related, and potentially links to sources of further information.

Using browser like navigation, the detail view allows the user to "surf" through the incident history, investigating a thread of incidents without leaving the detail view.

In addition to details about the incident being on display, the report takes advantage of the principles of temporal and spatial locality by providing hyperlinks to other incidents which are close in time or caused by the same agent. This gives the end user a quick overview of incidents which are likely to be relevant.

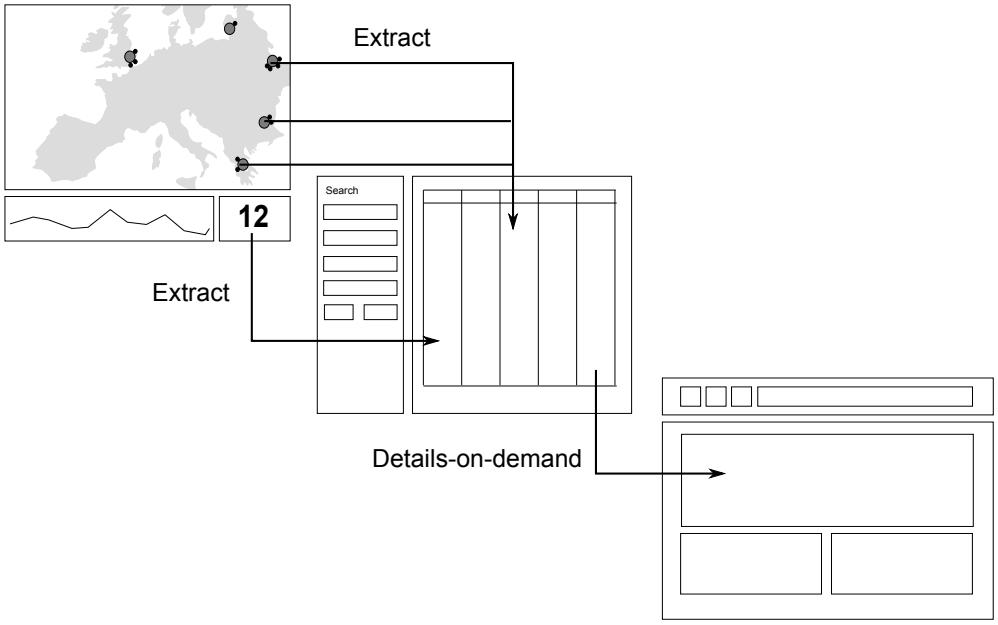

Figure 4.9: Interpanel navigation

4.5.4 Interpanel Navigation

The natural navigational flow is to start in the overview panel, select an incident wheel to get information about its events, select an incident of particular importance, and then move around in the detail view. The following operations are possible:

- Clicking an incident wheel should open the filter view, with only the events from that wheel displayed.
- Clicking on the gauge should have a similar effect, but for all incidents occurring in the time frame given.
- Double clicking an event in the filter view should cause a report to be generated and presented.

In addition to these actions, it should be possible to move between panels freely, for instance using tabs.

Chapter 5

Proof-of-Concept

A prototype of LogWheels was developed in order to demonstrate the ideas put forth, and provide a basis for empirical tests in the future. This chapter presents the prototype, and some use cases demonstrating how it can be used.

Section 5.1 presents the prototype, and the scenario used in this chapter.

Section 5.2 briefly explains how a server can be configured to serve the client.

Section 5.3 describes how LogWheels can be setup to extend the infrastructure.

Section 5.4 provides screenshots of the prototype in action, while walking through some common use cases for the application.

5.1 Overview

In order to demonstrate LogWheels, a proof-of-concept client and a mock server was developed in Java. The purpose of this prototype is to illustrate the ideas put forth in this thesis, and serve as a basis for future empirical tests. The software is not developed for production systems.

Two modules are attached to this report:

LogWheels Client
An almost full implementation of the specification. Not stable, and lacking in some functionality, but should be complete enough to get an impression of how the final product should "look and feel".

Mock Server
Emulates a server, by producing simulated incidents. This allows simple testing of the client in a controlled environment, without requiring any configuration or honeypot activity. Through a graphical user interface, the server allows its user to modify frequency of incidents, or generation of special purpose events. Only HTTP traffic is supported.

A detailed instruction on how to execute the prototype can be found in appendix A. Technical information about the software, and how the source code can be modified, can be found in appendix B.

In this chapter, the client was used in combination with a virtual server to simulate execution the entire log management infrastructure.

Scenario

By simulating the setup of a small startup company, the purpose of this chapter is to demonstrate how a complete log management infrastructure can be set up, where in the infrastructure LogWheels fit, and finally to provide some specific use cases, illustrating how LogWheels can be used in everyday situations.

5.2 Infrastructure

Before using LogWheels, it's necessary to implement a log management infrastructure. The details for this configuration is outside the scope of this thesis, but a basic walkthrough is presented in order to demonstrate how an organization would approach the problem.

5.2.1 Server Setup

A web server was set up using the LAMP paradigm – Linux (operating system), Apache (web server), MySQL (relational database management system), and PHP (dynamic web pages) – a standard open source software stack for web servers. Using a popular distribution such as Ubuntu, these services can all be installed during the OS installation routine [76].

Apache was configured to accept SSL traffic on port 443, with a self signed certificate generated using OpenSSL [24]. A root folder for secure traffic was created and protected using Basic Access Authentication [25]. Appendix C details how the web server should be configured, and contains a simplified version of the PHP script used to serve incidents to the client.

The log management infrastructure was set up on the same server using rsyslog (see section 3.1). While this solution is not ideal, a server with low traffic should be able to handle the load. Rsyslog is installed by default on Ubuntu, the Linux distribution of choice. Ordering rsyslog to send incidents directly to a MySQL database is done by installing the MySQL plugin module, available in the aptitude package manager as 'rsyslog-mysql'. The following lines were added to /etc/rsyslog.conf [30]:

```
$ModLoad ommysql

*.* :ommysql:127.0.0.1,Syslog,rsyslog,<chosen password>
```

After restarting the rsyslog service, incidents should be stored in the SystemEvents table in the Syslog database, as seen in figure 5.1.

ID	DeviceReportedTime	Priority	Message
71	2011-06-02 13:41:22	5	vegare : TTY=pts/0 ; PWD=/var/www/secure ;
72	2011-06-02 13:41:22	5	vegare : TTY=pts/0 ; PWD=/var/www/secure ;
73	2011-06-02 13:41:37	6	Accepted password for vegare from 192.168.56.
74	2011-06-02 13:41:37	6	Accepted password for vegare from 192.168.56.
75	2011-06-02 13:41:37	6	pam_unix(sshd:session): session opened for us
76	2011-06-02 13:41:37	6	pam_unix(sshd:session): session opened for us
77	2011-06-02 13:41:41	5	pam_unix(sudo:auth): authentication failure;
78	2011-06-02 13:41:41	5	pam_unix(sudo:auth): authentication failure;
79	2011-06-02 13:41:50	1	vegare : 3 incorrect password attempts ; TT
80	2011-06-02 13:41:50	1	vegare : 3 incorrect password attempts ; TT
81	2011-06-02 13:41:51	6	pam_unix(sshd:session): session closed for us
82	2011-06-02 13:41:51	6	pam_unix(sshd:session): session closed for us

12 rows in set (0.00 sec)

Figure 5.1: Excerpt from rsyslog's MySQL database

5.2.2 Log Producers

Once the server is configured with all necessary software, the next step is to secure it. In the context of LogWheels, the interesting question is which hardware and software entities which should be chosen to produce logs.

Available log producers include intrusion detection systems, intrusion prevention systems, authentication servers, routers, and firewalls [46].

Any organization running network connected hardware must analyze their system for vulnerabilities, and take the necessary precautions to discover malicious incidents. The result of these incidents should include log records being produced.

Most log producers should output syslog data by default. This data must be filtered, sent to the database, then retrieved by the client.

Example

A simple example is the use of secure shells. The Secure Shell Protocol (SSH) allows remote connections to a server through a secure channel [80], and is a common way to perform work on a server which is located remotely or not

connected to the necessary input and output devices. When allowing remote clients to login to a server, it's vital to monitor the tool for attempted attacks.

By default, the sshd daemon will output syslog data using the facility AUTH and the log level INFO [4]. The daemon outputs routine information, but also important information about failed login attempts. By using the filtering techniques mentioned in the next section, it's possible to isolate the interesting properties of sshd, such as:

```
# Failed login
Failed password for <user> from <ipaddress> port <port> ssh2
# Successful login
Accepted password for <username> from <ipaddress> port <port> ssh2
```

Aggregation techniques can be used to avoid overflow during a brute force attack, in which a series of usernames and passwords are tried in sequence in search for a valid combination [33].

In LogWheels, it would be beneficial to know when someone is trying to login to your ssh server, but perhaps even more important to know when it succeeds. If only one or two administrators are supposed to have access to the server, any successful login not hailing from the organization's home town warrants immediate attention. False positives would be easy to discard. By applying the proper filters, the system administrators will know within seconds if someone attempts to login to their server, and if they succeed, a more severe warning will be made, potentially resulting in an SMS alert.

Due to the remote monitoring paradigm, even a successful attacker must be able to clean the logs before they are sent to the client, or to break all the machines hosting clients as well. The attacker can obviously kill or replace the LogWheels server, but will have to race time to complete this task before an alert is sent to the system administrator. Alerts should not depend on client requests, but be handled in real-time by the log management infrastructure.

While monitoring the sshd daemon is a good idea, it's far from sufficient by itself. A proper analysis should be made, ensuring that the most critical systems are being monitored, and that necessary incidents are being reported to the LogWheels interface. External penetration testing is recommended.

5.2.3 Filter Configuration

Using the default configuration, everything received by rsyslog is logged. This is not desirable for a monitoring system, as loads of uninteresting routine messages will be stored. The client can filter on severity, but it's a waste of server resources to maintain a database of uninteresting details.

A simple filter can be added to the log management infrastructure, as described by section 2.2.4. Filtering parameters can be facility, severity, content of text message or similar attributes. The user of LogWheels can perform their own filtering in the GUI, this filtering permanently discards information we have determined to be irrelevant.

Two main approaches exist for filtering [67]:

Blacklist Block entities based on signatures or heuristics

Whitelist Accept only pre-approved entities

The whitelist approach is unlikely to be a satisfactory solution for most LogWheels scenarios. While they remove all noise, the operator is at risk of missing critical events the filter is unaware of. A blacklist approach will cause much unnecessary noise at first, but after some tuning it should be possible to remove everything but the interesting incidents.

A desirable expansion to the client is the opportunity to affect the filter directly from the client, by marking a message as trivial, preventing it from showing up again. This was not included in the prototypical specification, but could be considered during the empirical phase of the project.

External or home made tools can be used for high quality filtering, but it's also possible to implement basic filter functionality directly in rsyslog, through /etc/rsyslog.conf. An example is given below [29]:

```
:msg, regex, "Failed␣password␣.*␣ssh2"
```

Using expression based filtering, it's even possible to include conditional logic and string operations on the log events [29]:

```
if $syslogfacility-text == 'AUTH' and $msg startswith 'Failed' and
not ($msg contains '127.0.0.1') then <logsource>
```

In addition to filtering, it's possible to add other techniques from section 2.2.4, such as normalization or aggregation, through configuration or tools. This remains outside the scope of this report, and readers are directed to the rsyslog documentation for more information on implementation.

The end result of this process should be a MySQL database continuously being filled with the security incidents which are determined to be threats to the system, and which should be fed to LogWheels for investigation.

5.3 LogWheels Setup

Installing and setting up LogWheels is a simple task once the log management infrastructure is running correctly. A server wrapper must be developed to fit the infrastructure, and once it's in place, the client can be executed on any device supporting Internet communication.

A simple server implementation can be found in appendix C. A more comprehensive server should be implemented as a part of the toolkit development process described in section 7.2.

It's recommended to set up different nodes, and configure these to have different profiles. For instance, a client running on a mobile phone could be limited to emergencies, while an office computer could be responsible for handling all threats. The client is available in prototype form. It currently only support Java enabled devices with a resolution of more than 1024x768, which excludes mobile phones, but implementing other clients adhering to the standard could be part of the toolkit development process described in section 7.2.

5.4 Use Cases

This section will walk through the daily routine of the organization, and include screenshots of the proof-of-concept.

5.4.1 Daily Routine

For a small organization, it's common to assume that the office is populated between 8:00 and 17:00, but not around the clock. Inside the office, a spare monitor can be used to display LogWheels all day, with full coverage of all incidents. Employees can glance at the screen whenever they walk by, looking for unexpected behaviour.

Alternatively, the person in charge of server security can have a similar setup on his own computer. At least once per hour, he is required to look at the application, and go through suspicious incidents.

In addition, the system administrator can install LogWheels at home, and configure the application only to display critical incidents. That way the chance of discovering incidents increase, without bothering people at home regularly.

It's also recommended to have a system for SMS or e-mail alerts on emergencies, but this functionality would be part of the log management infrastructure, not LogWheels.

Every morning, the system administrator should walk through the past 24 hours using the history graph. On Monday mornings, the entire weekend should be examined. If no alerts have arrived, it's unlikely that the system has been broken, but it's still worthwhile looking through the history, trying to find patterns or suspicious behaviour.

Technically inclined members of the organization should be encouraged to play with LogWheels, learning about the system and the threats it's exposed to. By setting up a LogWheels node in an open area, non-technical employees can also inspect the screen when available, warning the system administrator if something is wrong.

5.4.2 Screenshots

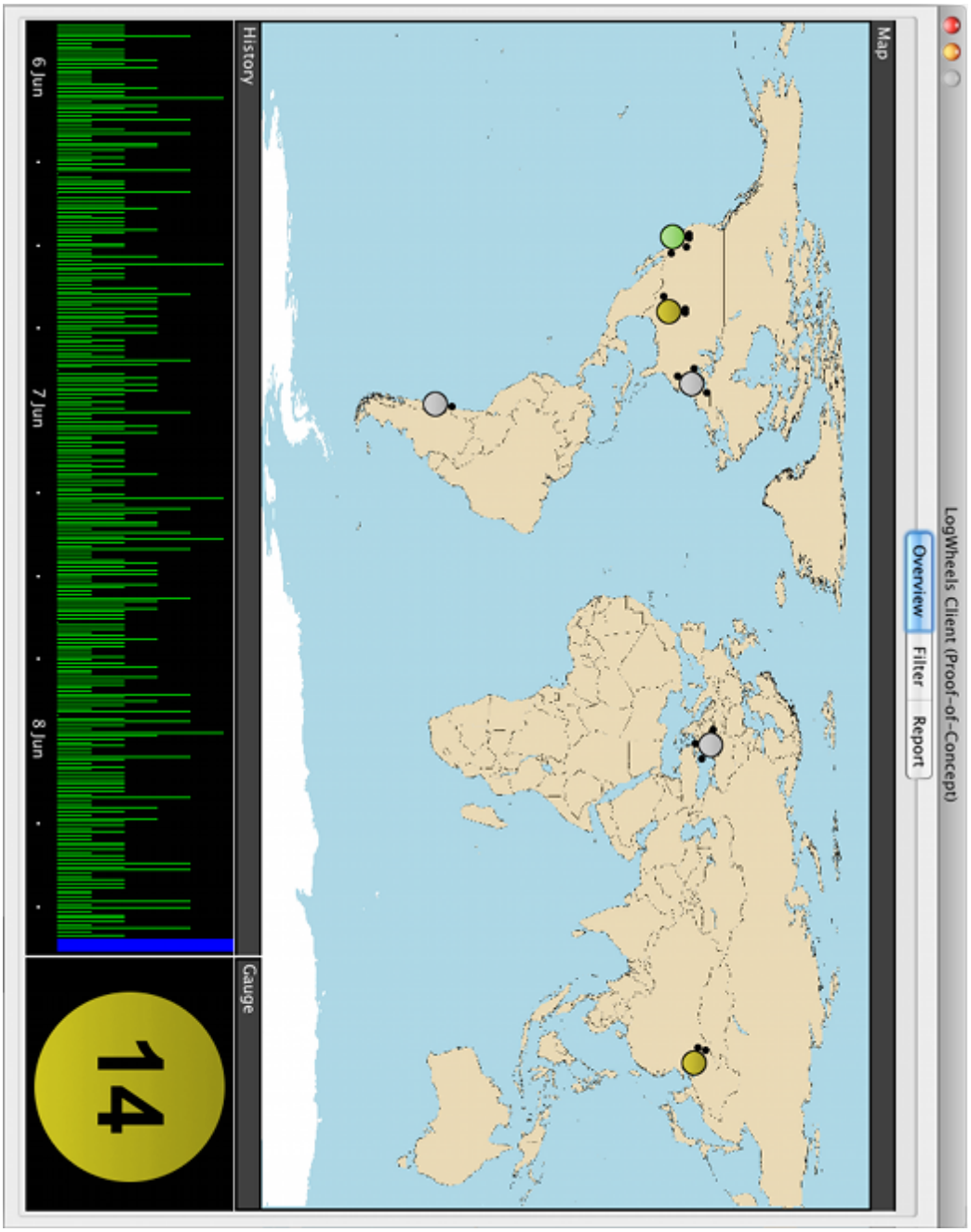

Figure 5.2: Overview screen

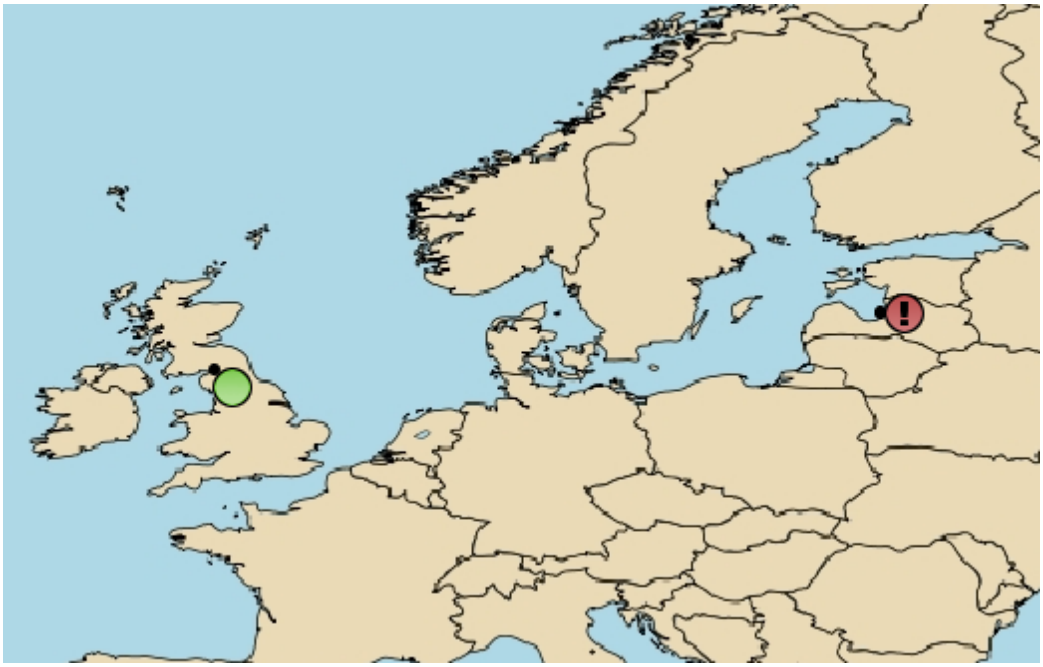

Figure 5.3: Closeup on Europe

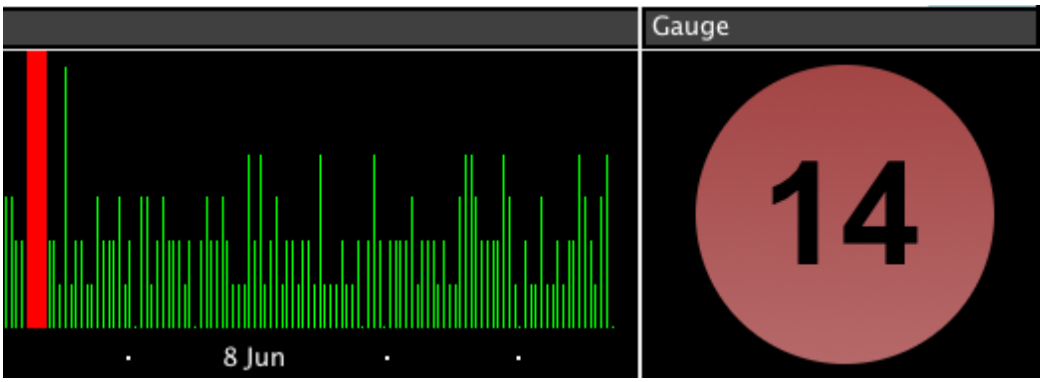

Figure 5.4: Closeup on history and gauge

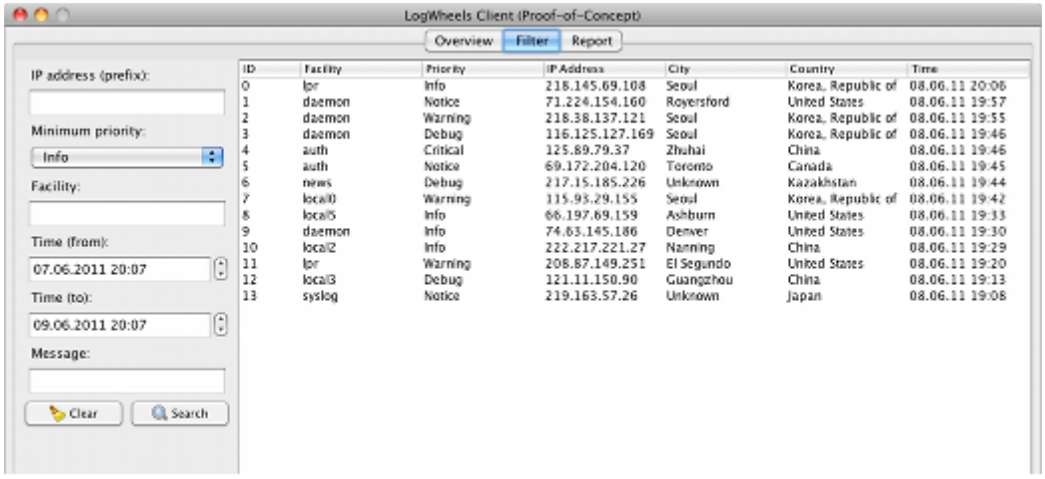

Figure 5.5: Filter screen

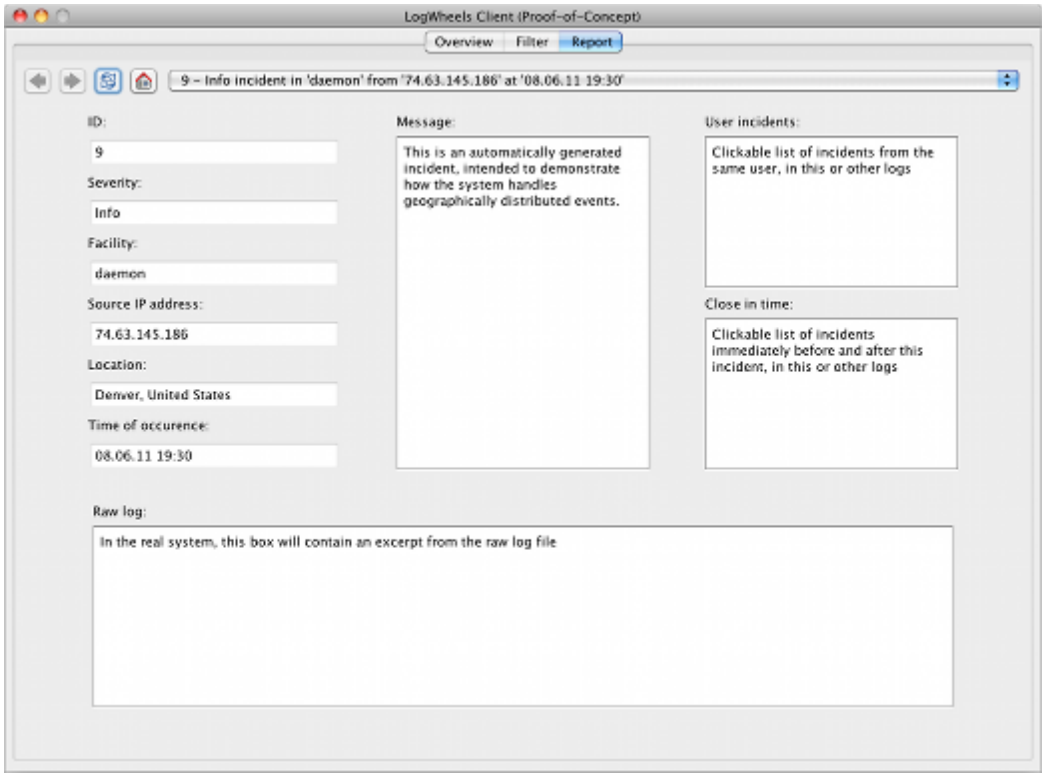

Figure 5.6: Report screen

Chapter 6

Discussion

This chapter contains reflections on how the prototype met the project goals, which known weaknesses the proposal contains, and which aspects it succeeds to respond to.

Section 6.1 describes how the project goals from section 1.2 were met.

Section 6.2 proceeds by describing some of the limitations of the system, and how these can be solved.

6.1 Goal Achievement

The goals of this thesis were to investigate log usability, propose a new mechanism to improve the usability of security incident logs, and develop a proof-of-concept of the ideas proposed.

PG1 – Examine the state of the art in security log usability

The theoretical study (chapter 2) covers the fundamentals of security logs, the technical aspects of establishing a log management infrastructure, the challenges relating to situational awareness and security usability, and finally a walk-through of usability improving techniques, including the field of visualization and human-computer interaction.

The practical study (chapter 3) addresses log management infrastructure implementations, with a particular focus on visualization components and solutions. A deliberate decision was made to avoid investigating specific implementations of ideas that would not be used in the development of LogWheels, such as data normalization or report generation.

PG2 – Propose a mechanism to improve the usability of security logs

Two mechanisms were proposed (chapter 4): a remote monitoring solution named LogWheels, and a new visualization component named the incident wheel. The incident wheel is the core component of LogWheels, resulting in a novel contribution to the field of security visualization.

Several conceptual drivers were listed in section 4.1. These were firmly rooted in the theoretical study from chapter 2 and 3, assuring that the ideas proposed were based on a fundament of high quality research.

The incident wheel presents three key dimensions of security incidents: 'what', 'when', and 'where', which was proposed by Livnat et al. as sufficient to improve situational awareness [50]. When combined with Shneiderman's in-

formation seeking mantra – "overview first, zoom and filter, then details-on-demand" [69] – the result is a solution offering both high level situational awareness and low level knowledge of specific incidents.

After being exposed to the empirical exploration and feasibility study recommended in section 7.2, and adjusted according to the results found, LogWheels should have potential to be a useful addition to the security toolchain of small organizations.

PG3 – Develop a proof-of-concept, illustrating the ideas proposed

A proof-of-concept client and a mock server was developed (chapter 5), using a total of 3116 lines of Java code.

The client is an almost complete implementation of the LogWheels client. It lacks some functionality, particularly in the report details, but is comprehensive enough to illustrate all the ideas proposed and give a feeling of how the application will work. As no graphical designer was involved in the process, the user interface is still somewhat rough, but the basic design and interaction interface adhere to the ideas proposed.

As LogWheels still requires considerable configuration of the infrastructure to be usable, in addition to the requirement of security incidents occurring on the server under observation, it was decided to implement a mock server, which simulates security incidents. The mock server uses simple statistics to generate these events. The model is not rooted in any empirical studies, but by configuring the main parameters it should be possible to simulate both low and high traffic servers.

Despite the lack of a proper LogWheels server implementation, the client should be able to communicate with such a server without any changes. This is in part demonstrated by chapter 5, where a virtual server was setup and rsyslog messages were transmitted directly to an external LogWheels client. This experiment can be replicated by interested parties, but was not feasible to include as an attachment to the report.

6.2 Limitations

Methodological Limitations

Due to the time constraints described in section 1.3, it was infeasible to bring this project beyond the idea and mockup phase. The ideas proposed are based on a foundation of theoretical work by leading academics in the field, but this is not a replacement for an empirical exploration.

The prototype must be subjected to a feasibility study, determining the practicality of implementing the solution in a real-world scenario, in addition to an empirical study searching for measurable effects of using the visualization scheme. As Shneiderman and Plaisant writes, "information visualization must be more than 'cool'; they have to provide measurable benefits for realistic tasks" [69]. The prototype is constructed using well-documented principles, but it's still necessary to empirically test the solution before going any further.

A roadmap for this process is outlined in section 7.2.

Dependency on IP Addresses

The visualization scheme relies on geographically distributed incidents, which again relies on the availability of an IP address causing each security incident. Unfortunately, many log producers do not include an IP address at all, while others present the IP address as part of the textual description.

Extracting the IP address from a textual message causes an overlap of functionality with the log management infrastructure (consolidation), and should be considered when selecting a log management infrastructure to be used in a complete software solution. Compensating for the lack of IP addresses is a bigger problem. In the prototype, unknown events are drawn internally – that is, in the location of the server – but if too many incidents occur in the same place, the visualization scheme provides less value, as the pattern-seeking abilities are poorly utilized.

In addition, attackers using zombies or proxies will not be discovered by the system. This is not a serious problem for the visualization solution, but a more general problem in the field of incident investigations.

Scalability Limitations

The proposal was designed to be appropriate for small organizations, but will not be applicable for larger systems. Too many incidents will cause the visualization to clutter up, and the benefits described will no longer apply. This is a weakness to the proposal as a universal solution, but is inherent in the system by design.

This problem is common in visualization solutions, as there is a practical limit of how much information it's possible to represent in the space limited by a computer screen. While more aggressive compression or normalization techniques can be applied, this is likely to make the visualization less intuitive and less usable.

The Challenge of Configuration

While LogWheels may increase the usability of consolidated security logs, it can not determine which incidents are actually important. Most log producers come with their own set of standards, but even if a toolkit was created, integrating LogWheels with an existing log management infrastructure, they would only be able to provide a decent default setting.

Different organizations have different needs, and should prioritize incidents differently. This contradicts the purpose of LogWheels, which is to provide a "black box" approach to security log monitoring. Simplifying the monitoring process too much may in fact be detrimental to the security, as it may cause a false sense of security even if the tool misses important incidents.

In a final product, it's important to guide the user towards a fitting configuration, while acknowledging that some properties can not be determined for every organization. Users should therefore be encouraged to include various log sources.

Chapter 7

Conclusion

The final chapter concludes the project, and provides a roadmap for further work.

Section 7.1 consists of the statements of inference made during the report, and summarizes the contributions made to the field of security usability and security visualization.

Section 7.2 contains a roadmap for further work on the prototype, and suggests a few project ideas ranging from undergraduate level to larger graduate projects.

7.1 Conclusion

The motivational driver for this thesis was the desire to help smaller organizations with the log monitoring process, leading them towards higher situational awareness and thus a more secure system. The core idea was to present the content of security logs holistically within the context of a well-designed presentation interface, encouraging frequent use.

Security logs contains much of the information needed to protect a system, but mundane plain text presentations and seemingly arbitrary inspection routines does not provide satisfactory situational awareness. In order to achieve the necessary system comprehension, principles from the fields of security usability, visualization, and human-computer interaction were combined, and a set of conceptual drivers for a visualization solution were identified.

Using these principles, and inspiration from state of the art solutions, two new concepts were proposed:

LogWheels is a security log visualizer, supporting remote monitoring of security incident logs through visualization and user interaction. A suggested architecture and a communication protocol were designed, and the overall visualization scheme presented.

The Incident Wheel is a new security visualization component. The component presents three key dimensions of security incidents, laying the foundation for high situational awareness through an intuitive graphical display.

By developing a proof-of-concept, the ideas can be demonstrated using real or simulated security data. By exposing the proof-of-concept to an empirical exploration, it is the hope of the author that the idea eventually results in a usable toolkit, guiding smaller organizations towards higher system comprehension, and inspiring them to delve deeper into the world of information security.

7.2 Further Work

While the solution is based on several published theoretical sources, it has yet to be subjected to real world applications. In order to transfer the ideas presented in this thesis into a usable security solution, three main steps remain, as described in the sections below.

Empirical Testing

An idea has been proposed, but as Shneiderman and Plaisant makes painfully clear, it's not enough for a visualization to be "cool", its positive impact must be documented through measurable parameters [69].

For this solution, the question to ask is whether using the system will make the organization's computing systems more secure. A good first step is to investigate whether users intuitively understand the visualization abstractions, whether they are able to pick up pattern deviations as theory suggests, and whether they actually pay attention to the tool in a real world setting.

Performing paper prototyping or laboratory tests of the interface can be formulated as an undergraduate school assignment. Creating a framework for observation in the wild, and following up with a real world example, is suggested for a more comprehensive graduate project.

In particular, it would be interesting to expose LogWheels to the Situational Awareness Global Assessment Technique (SAGAT), which examines the system from the operator's point of view [21]. Jean Scholtz has presented several metrics for evaluating information visualizations empirically, and his works are recommended for the designers of an empirical test [65].

Parts of the empirical study should be an incremental improvement of the ideas presented. The proof-of-concept is open source, and future researchers are encouraged to modify the configuration parameters and presentation components, adapting the application to its users, not the other way around.

Feasibility Study

Another potential show stopper is the question of whether the solution is feasible for real world applications. The architecture should be able to handle large loads, as demonstrated by the Internet every day, but the visualization has a window of event frequency which is likely to be surpassed by organizations once they hit a critical amount of security log data.

Collecting data from various startup companies can assist a student in the task of measuring whether the visualization holds for average organizations. The underlying log management infrastructure can be used to add or remove data is most cases, but if the visualization is unable to provide any benefit for the most important incidents, it's unlikely to be of any use to the organization.

A recommended task for a student is to measure for which frequency window the visualization provides any benefit, and compare this value to the average activity on the servers of startup companies on university campus or in a startup incubator.

Toolkit Integration

The solution is set to extend an existing log management infrastructure. At the same time, it should at the same time *not* be neutral of the underlying infrastructure, as this would cause a reimplementation of infrastructure tasks.

However, a system designed to be simple to install and use can not require the user to setup a complete log management infrastructure and modify the source code of the solution in order to adapt to the chosen tool chain.

Once the architecture and visualization scheme has been empirically verified, the final step should be to integrate it with an existing tool chain and aim for a single installation tool, covering all the needs of a small organization.

Which log management infrastructure is suitable and how to properly integrate the solution, is a recommended task for further examination.

Bibliography

[1] Syslog Wiki. http://www.syslog.org/wiki/Main/Tools. Retrieved February 2011.

[2] syslog-win32. http://syslog-win32.sourceforge.net/. Retrieved March 2011.

[3] syslogd(8) - Linux man page. http://linux.die.net/man/8/syslogd. Retrieved March 2011.

[4] OpenSSH SSH daemon config file – Linux man page, September 1999. Retrieved from http://linux.die.net/man/5/sshd_config on May 2011.

[5] Mohammed AlZomai, Bander AlFayyadh, Audun Jøsang, and Adrian McCullagh. An Experimental Investigation of the Usability of Transaction Authorization in Online Bank Security Systems. In Ljiljana Brankovic and Mirka Miller, editors, *Sixth Australasian Information Security Conference (AISC 2008)*, volume 81 of *CRPIT*, pages 65–73, Wollongong, NSW, Australia, 2008. ACS.

[6] George Apostolopoulos, Vinod Peris, Prashant Pradhan, and Debanjan Saha. Securing electronic commerce: reducing the SSL overhead. *Network, IEEE*, 14:8–16, August 2000.

[7] Stefan Axelsson, Ulf Lindqvist, Ulf Gustafson, and Erland Jonsson. An Approach to UNIX Security Logging. In *Proc. 21st NIST-NCSC National Information Systems Security Conference*, pages 62–75, 1998.

[8] M. Pauline Baker and Donald Hearn. *Computer Graphics with OpenGL*. Pearson Prentice Hall, Upper Saddle River, NJ 07458, 3 edition, 2004. International Edition.

[9] Tim Belcher and Elad Yoran. Riptech Internet Security Threat Report: Attack Trends for Q1 and Q2 2002. II, July 2002. Retrieved from http://eval.symantec.com/mktginfo/enterprise/white_papers/ent-whitepaper_symantec_internet_security_threat_report_ii.pdf May 2011.

[10] Tim Berners-Lee, Roy T. Fielding, James Gettys, Jeffrey C. Mogul, Henrik Frystyk Nielsen, Larry Masinter, and Paul J. Leach. Hypertext Transfer Protocol – HTTP/1.1. RFC 2616 (Draft Standard). Technical report, Internet Engineering Task Force, June 1999.

[11] Tim Berners-Lee, Larry Masinter, and Mark McCahill. Uniform Resource Locators (URL). RFC 1738 (Proposed Standard). Technical report, Internet Engineering Task Force, December 1994.

[12] Engin Bozdag, Ali Mesbah, and Arie van Deursen. A Comparison of Push and Pull Techniques for AJAX. In *Proceedings of the 2007 9th IEEE International Workshop on Web Site Evolution*, pages 15–22, June 2007.

[13] Matt Cooper, Yuriy Dzambasow, Peter Hesse, Susan Joseph, and Richard Nicholas. Internet X.509 Public Key Infrastructure: Certification Path Building. RFC 4158 (Informational). Technical report, Internet Engineering Task Force, September 2005.

[14] Douglas Crockford. The application/json Media Type for JavaScript Object Notation (JSON). RFC 4627 (Informational). Technical report, Internet Engineering Task Force, July 2006.

[15] Cyberoam. Cyberoam iView - Open Source SIEM. http://sourceforge.net/projects/cyberoam-iview/. Retrieved March 2011.

[16] Christopher Davis, Paul Vixie, Tim Goodwin, and Ian Dickinson. A Means for Expressing Location Information in the Domain Name System. RFC 1876 (Experimental). Technical report, Internet Engineering Task Force, 1996.

[17] Stephen E. Deering and Robert M. Hinden. Internet Protocol, Version 6 (IPv6). RFC 2460 (Draft Standard). Technical report, Internet Engineering Task Force, December 1998.

[18] Tim Dierks and Eric Rescorla. The Transport Layer Security (TLS) Protocol. RFC 5246 (Proposed Standard). Technical report, Internet Engineering Task Force, August 2008.

[19] Curtis E. Dyreson and Richard T. Snodgrass. Timestamp semantics and representation. *Information Systems*, 18(3):143–166, February 1993.

[20] Mica R. Endsley. *Situation Awareness Analysis and Measurement*, chapter 1, pages 3–28. CRC Press, 10 Industrial Avenue, Mahwah NJ 07430, 2000.

[21] Mica R. Endsley. Designing for Situation Awareness in Complex System. In *Second intenational workshop on symbiosis of humans, artifacts and environment*, Kyoto, Japan, 2001.

[22] Eric Fitzgerald, Anton Chuvakin, Bill Heinbockel, Dominique Karg, and Raffael Marty. Common Event Expression (CEE) Overview. Technical report, The CEE Editorial Board, November 2010. Retrieved from `http://cee.mitre.org/docs/Common_Event_Expression_Overview.pdf` in January 2011.

[23] Python Software Foundation. 35.15. syslog — Unix syslog library routines – Python v2.7.1 documentation. `http://docs.python.org/library/syslog.html`. Retrieved March 2011.

[24] The Apache Software Foundation. Apache SSL/TLS Encryption - Apache HTTP Server. `http://httpd.apache.org/docs/2.2/ssl/`. Retrieved May 2011.

[25] The Apache Software Foundation. Authentication, Authorization and Access Control - Apache HTTP Server. `http://httpd.apache.org/docs/current/howto/auth.html`. Retrieved April 2011.

[26] John Franks, Phillip M. Hallam-Baker, Jeffery L. Hostetler, Scott D. Lawrence, Paul J. Leach, Ari Luotonen, and Lawrence C. Stewart.

HTTP Authentication: Basic and Digest Access Authentication. RFC 2617 (Draft Standard). Technical report, Internet Engineering Task Force, June 1999.

[27] Vegard Fremstad. Effektiv logganalyse i et heterogent driftsmiljø. Master's thesis, Norwegian University of Science and Technology, 2009.

[28] Kevin Fu, Emil Sit, Kendra Smith, and Nick Feamster. Dos and Don'ts of Client Authentication on the Web. In *Proceedings of the 10th conference on USENIX Security Symposium*, volume 10 of *SSYM'01*. USENIX Association, August 2001.

[29] Rainer Gerhards. *Filter Conditions - rsyslog.conf rsyslog*. Adiscon, 2008. Retrieved from `http://www.rsyslog.com/doc/rsyslog_conf_filter.html` in May 2011.

[30] Rainer Gerhards. *Writing syslog messages to MySQL*. Adiscon, February 2008. Retrieved from `http://www.rsyslog.com/doc/rsyslog_mysql.html` in April 2011.

[31] Rainer Gerhards. The Syslog Protocol. RFC 5424 (Proposed Standard). Technical report, Internet Engineering Task Force, March 2009.

[32] Rainer Gerhards. An Introduction to Syslog. `http://www.monitorware.com/common/en/seminarsonline/intro-syslog.php`, March 2011. Retrieved March 2011.

[33] Dieter Gollmann. *Computer Security*. John Wiley & Sons, LTD, 111 River Street, Hoboken, NJ 07030, 2 edition, 2006.

[34] Google. google-gson - A Java library to convert JSON to Java objects and vice-versa - Google Project Hosting. `http://code.google.com/p/google-gson/`. Retrieved April 2011.

[35] Andreas Hegna (Watchcom Security Group). Personal interview, January 2011.

[36] Marc Grégoire and Luc Beaudoin. Visualisation for Network Situational Awareness in Computer Network Defence. In *Visualisation and the Common Operational Picture*, 2005.

[37] Bamba Gueye, Steve Uhlig, and Serge Fdida. Investigating the imprecision of IP block-based geolocation. In *Proceedings of the 8th international conference on Passive and active network measurement*, PAM'07, 2007.

[38] Andreas Hegna. Visualizing Spatial and Temporal Dynamics of a Class of IRC-Based Botnets. Master's thesis, Norwegian University of Science and Technology, 2010.

[39] Hewlett-Packard. News release: HP to Acquire ArcSight. `http://www.hp.com/hpinfo/newsroom/press/2010/100913xa.html`, September 2010. Retrieved March 2011.

[40] Ian Hickson. HTML5 - A vocabulary and associated APIs for HTML and XHTML (W3C Working Draft). `http://www.w3.org/TR/html5/the-canvas-element.html#the-canvas-element`, May 2011. Retrieved May 2011.

[41] James A. Hoagland. Audit Log Analysis Using the Visual Audit Browser Toolkit. `http://seclab.cs.ucdavis.edu/projects/awb/Visual.Audit.Browser.html`, July 1995. University of California, Davis.

[42] Debra G. Jones and Mica R. Endsley. Sources of situation awareness errors in aviation. In *Aviation, Space, and Environmental Medicine*, volume 67, pages 507–512, 1996.

[43] The jQuery Project. jQuery: The Write Less, Do More, JavaScript Library. `http://www.jquery.com/`. Retrieved April 2011.

[44] Audun Jøsang, Bander AlFayyadh, Tyrone Grandison, Mohammed Alzomai, and Judith McNamara. Security usability principles for vulnerability analysis and risk assessment. In *The Proceedings of the Annual Computer Security Applications Conference (ACSAC 07)*, pages 269–278, Miami Beach, FL, December 2007.

[45] Audun Jøsang, Mohammed AlZomai, and Suriadi Suriadi. Usability and Privacy in Identity Management Architectures. In *Proceedings of the fifth Australasian symposium on ACSW frontiers - Volume 68*, ACSW '07, pages 143–152, Ballarat, Australia, 2007.

[46] Karen Kent and Murugiah Souppaya. Guide to Computer Security Log Management. Special Publication 800-92, National Institute of Standards and Technology, Gaithersburg, MD, September 2006.

[47] KIWI. Kiwi Log Viewer | Log Analyzer | Kiwi Syslog. http://www.kiwisyslog.com/kiwi-log-viewer-overview/. Retrieved March 2011.

[48] Kiran Lakkaraju, William Yurick, and Adam J. Lee. NVisionIP: NetFlow Visualization of System State for Security Situational Awareness. In *Proceedings of the 2004 ACM workshop on Visualization and Data Mining for Computer Security*, VizSEC/DMSEC, pages 65–72. ACM, 2004.

[49] Yarden Livnat, Jim Agutter, Shaun Moon, Robert F. Erbacher, and Stefano Foresti. A Visual Paradigm for Network Intrusion Detection. In *In Proceedings of the 2005 IEEE Workshop on Information Assurance And Security*, pages 92–95. IEEE Computer Society, 2005.

[50] Yarden Livnat, Jim Agutter, Shaun Moon, and Stefano Foresti. Visual Correlation for Situational Awareness. In *Proceedings of the 2005 IEEE Symposium on Information Visualization*, pages 95–102, Washington DC, November 2005. IEEE Computer Society.

[51] Chris Lonvick. The BSD syslog Protocol. RFC 3164 (Informational). Technical report, Internet Engineering Task Force, August 2001.

[52] Carsten Maartmann-Moe. Forensic Key Discovery and Identification. Master's thesis, Norwegian University of Science and Technology, 2008.

[53] CRN Magazine. SIEM: A Market Snapshot. http://www.crn.com/news/security/197002909/siem-a-market-snapshot.htm, February 2007. Retrieved March 2011.

[54] Phillip Q. Maier. *Audit and Trace Log Management: Consolidation and Analysis*. Taylor & Francis Ltd, 6000 Broken Sound Parkway NW, Suite 300, Boca Raton, FL 33487, 2006.

[55] Raffael Marty. SecViz - Security Visualization. http://secviz.org/category/image-galleries/graph-exchange. Retrieved February 2011.

[56] Raffael Marty. *Applied Security Visualization*. Pearson Education, Inc, 501 Boylston Street, Suite 900, Boston, MA, 2008.

[57] Fuyou Miao, Yuzhi Ma, and Joseph Salowey. Transport Layer Security (TLS) Transport Mapping for Syslog. RFC 5425 (Proposed Standard). Technical report, Internet Engineering Task Force, March 2009.

[58] Geir Mork (Norman). Personal interview, March 2011.

[59] OSSIM. The Open Source SIEM. `http://sourceforge.net/projects/os-sim/`. Retrieved March 2011.

[60] Donald Pitts. Log Consolidation with syslog. Technical report, SANS Institute, December 2000. Retrieved from `http://www.syslog.org/wiki/uploads/Main/log_consolidation_with_syslog.pdf` in March 2011.

[61] Jon Postel. Internet Protocol. RFC 791 (Standard). Technical report, Internet Engineering Task Force, September 1981.

[62] Eric Rescorla. HTTP Over TLS. RFC 2818 (Informational). Technical report, Internet Engineering Task Force, May 2000.

[63] RSA. enVision. `http://www.rsa.com/node.aspx?id=3170`. Retrieved March 2011.

[64] rsyslog. The enhanced syslogd for Linux and Unix rsyslog. `http://www.rsyslog.com/`. Retrieved March 2011.

[65] Jean Scholtz. Beyond Usability: Evaluation Aspects of Visual Analytic Environments. In *2006 IEEE Symposium On Visual Analytics Science And Technology*, pages 145–150, 2006.

[66] BalaBit IT Security. syslog-ng - Multiplatform Syslog Server and Logging Daemon. `http://www.balabit.com/network-security/syslog-ng/`. Retrieved March 2011.

[67] Dave Shackleford. Application Whitelisting: Enhancing Host Security. white paper on `http://www.sans.org/reading_room/analysts_program/McAfee_09_App_Whitelisting.pdf`, October 2009. Sponsored by McAfee.

[68] Ben Shneiderman. The Eyes Have It: A Task by Data Type Taxonomy for Information Visualizations. In *Proceedings of the 1996 IEEE Symposium on Visual Languages*, Washington, DC, USA, 1996. IEEE Computer Society.

[69] Ben Shneiderman and Catherine Plaisant. *Designing the User Interface: Strategies for Effective Human-Computer Interaction*. Pearson Education, Inc, 4 edition, 2005. International Edition.

[70] John Parr Snyder. *Flattening the earth: two thousand years of map projections*. The University of Chicago Press, Chicago 60637, 1997.

[71] Syslog4j. Syslog4j: Complete Syslog Implementation for Java. `http://www.syslog4j.org/`. Retrieved March 2011.

[72] Cisco Systems. Cisco Security Monitoring, Analysis and Response System (MARS). `http://www.cisco.com/en/US/products/ps6241/`. Retrieved March 2011.

[73] Tetsuji Takada and Hideki Koike. Tudumi: Information Visualization System for Monitoring and Auditing Computer Logs. *International Conference on Information Visualization*, page 570, 2002.

[74] Edward R. Tufte. *The Visual Display of Quantitative Information*. Graphic Press LLC, Cheshire, Connecticut, 2 edition, 2001.

[75] Princeton University. WordNet: A Lexical Database for English. `http://wordnetweb.princeton.edu/perl/webwn?s=log`. Retrieved February 2011.

[76] Ubuntu wiki. ApacheMySQLPHP - Community Ubuntu Documentation. `https://help.ubuntu.com/community/ApacheMySQLPHP`. Retrieved May 2011.

[77] Yang Xie, Kyeong S. Jeong, Wei Pan, Arkady Khodursky, and Bradley P. Carlin. A Case Study on Choosing Normalization Methods and Test Statistics for Two-Channel Microarray Data. *Comparative and Functional Genomics*, 5, July 2004.

[78] Ka-Ping Yee. Aligning Security and Usability. *IEEE Security and Privacy*, 2:48–55, September 2004.

[79] Xiaoxin Yin, William Yurick, and Michael Treaster. VisFlowConnect: NetFlow Visualization of Link Relationships for Security Situational Awareness. In *Proceedings of the 2004 ACM Workshop on Visualization and Data Mining for Computer Security*, VizSEC/DMSEC. ACM, 2004.

[80] Tatu Ylonen. The Secure Shell (SSH) Protocol Architecture. RFC 4251 (Proposed Standard). Technical report, Internet Engineering Task Force, January 2006.

[81] Mary Ellen Zurko and Richard T. Simon. User-Centered Security. In *Proceedings of the 1996 workshop on New security paradigms*, NSPW '96, pages 27–33, New York, NY, 1996. ACM.

Appendix A

User Guide

A mock server and a proof-of-concept client was developed in Java, and should be runnable on any system with Java support and a resolution larger than 1024x768. Tests have been performed in Windows 7, Mac OS X, and Ubuntu 11.04, using 32- and 64-bit Java 1.6.0_17.

A.1 Installation

Both the client and the server are attached to this thesis. The zip file contains source code, executable class files, all required static resources, and JavaDoc in a web browsable format. To install the application, simply unzip the file in any folder on the local file system.

A.2 Execution

In order to test the client, a compatible server must be available. It's recommended to use the mock server attached to this thesis, as the server maintains full compatibility with the protocol while requiring zero configuration and no need for malicious traffic.

1. Navigate to the LogWheels folder.
2. Execute the shell script mock-server.sh (on Mac OS or Linux) or win-mock-server.bat (on Windows). This should launch a GUI.

3. Start a server on your preferred port. Port 80 or 443 is recommended if they are free on your system, otherwise 8080 is a commonly used HTTP alternative.

4. If you want real time data updates, start a data generator or insert your own incidents to the server database.

Once the server is running, the proof-of-concept client can be tested.

1. Navigate to the LogWheels folder.

2. Execute the shell script client.sh (on Mac OS or Linux) or win-client.bat (on Windows). This should launch a GUI.

3. Enter the URL of the server, with port number. If you run the server locally on port 8080, use http://localhost:8080/

4. Ignore the warning about the unencrypted connection. There is no need to protect randomly generated data.

5. The client prompts for a username and a password. These are "admin" and "test" respectively.

6. A new GUI window should open, with an overview screen displayed. Navigate freely to inspect the implementation.

New incidents can be generated while the client is running. The default polling interval is 10 seconds, meaning that your event should be visible in the client within few seconds.

Run in Browser

For a simple test of the interoperability, any web browser can be used to access the incident data.

1. Launch a web browser.

2. Enter the URL of the server. If you run the server locally on port 8080, use http://localhost:8080/?action=version
3. The browser prompts for a username and a password. These are "admin" and "test" respectively in this prototype.
4. This should display "LogWheels Prototype".
5. Enter another action to see the corresponding result.

A.3 Troubleshooting

Due to scope constraint, the application has not been subjected to any comprehensive tests. It's not designed to handle unexpected input values, and remains untested on most system configurations.

If the shell scripts are unable to launch the application, try to execute them from the command line, rather than a graphical tool. The applications should be launched by their respective bin/ directories, and a full Java command can be found in the .sh or .bat file.

If any runtime error occurs, try to restart the application. If a specific Java error occurs, modify the source code and recompile the project.

The application has been confirmed to work in Windows 7, Mac OS X, and Ubuntu 11.04, using 32- and 64-bit versions of Java 1.6.0_17, on a stationary Mac Pro and a Dell Inspiron 9400 laptop.

Appendix B

Implementation Details

The prototype was written in Java, and consist of 3116 lines of Java code, distributed between two projects: the proof-of-concept client (2446 LOC) and a mock server (670 LOC). This appendix contains an overview of the source code, primarily for the client.

B.1 Folder Structure

The attached zip file contains two directories, one for each of the applications (client and server). Each of the directories consists of the following subdirectories:

Name	**Function**
bin/	Executable files (.class)
doc/	Browseable JavaDoc (.html)
img/	Images used in the GUI (.png, .jpg)
lib/	Java libraries (.jar)
src/	Java source code (.java)

Table B.1: Prototype directories

In addition, the client has a separate folder for the GeoIP database, used for IP geolocation.

B.2 Package Structure

The projects are placed in the packages 'no.ntnu.vegare.visualizer.client' and 'no.ntnu.vegare.visualizer.server' respectively. The class hierarchy for each project is listed below:

Client package	Description
auth	View and controller for the authentication frames
config	A single class for prototype configuration
connection	Model and controller for HTTP(S) connection handling
db	Model and controller for SQLite database interaction
main	Model, view, and controller for the main GUI frame
.details	Detail panel
.filter	Filter panel
.overview	Overview panel
.gauge	Gauge component
.history	History graph component
.map	Map and incident wheel components
prototype	Utility classes that are only relevant in the prototype
utils	Utility classes for resources, time handling and ip geolocation

Table B.2: Client package structure

Server package	Description
background	Threads running in the background (server and generator)
db	SQLite database implementation
gui	User interface and user interaction
.panels	The three main panels in the main frame
utils	Utility classes for resource handling and data generation

Table B.3: Server package structure

B.3 Architecture

The server was developed for simplicity, with the view and controller merged in single classes, and a simple thread pool handling all background threads. The architecture will not be examined any further, but should be simple to grasp for anyone glancing through the main classes.

The client is based on the model-view-controller architectural pattern. In this pattern, data is stored in model classes, the user interacts with the application through view classes, and interaction is handled by controller classes. The interaction follows the lines shown in figure B.1.

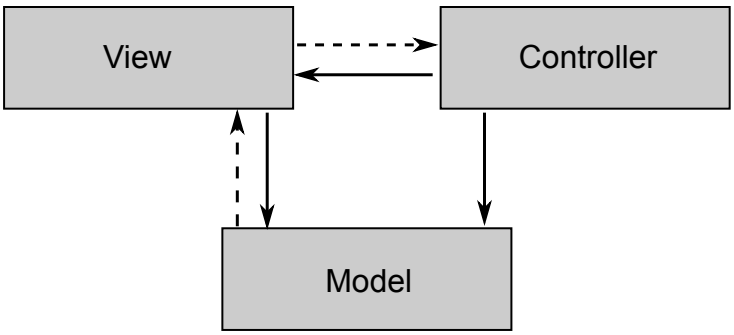

Figure B.1: Model-view-controller architectural pattern

Concurrency

In addition to the threads generated by the Java Virtual Machine, there are four main threads running in the application at any given time:

1. The ServerPollDaemon polls the server for new updates at a specified interval, and updates the database.

2. The MainController handles the main execution path of the process, including the main GUI.

3. A separate thread is used to animate the main canvas at a specified number of frames per second.

4. Java Swing uses a separate thread to handle user interaction, and may spawn separate threads behind the scene.

In addition, a separate thread is used for the authentication procedure in the beginning of the application lifetime, but this is only executed once, and never concurrently with other threads.

Main Code Path

The execution of the client is performed in four steps:

```
public static void main(String[] args) {

    HttpsConnection conn = new HttpsConnection();
    AbstractLogDatabase db = new SQLiteDatabase();

    DataGenerator.createBacklog(db);

    AuthController auth = new AuthController(conn);
    auth.establishConnection();

    Thread th = new Thread(new ServerPollDaemon(conn, db));
    th.setDaemon(true);
    th.start();

    MainController main = new MainController(db);
    main.launchGui();
}
```

In the prototype, the very first step is to generate a history of generated log data.

Second, an authenticated connection is made to the server. This is a blocking method call, only returning when a connection is successfully established.

Third, the server poll daemon is constructed. This thread runs in the background, polling the server at an interval given by the configuration class, and entering new log records in the database.

Finally, the client GUI is launched. This module uses a model-view-controller based architecture, presenting data from the database in real time, while allowing user navigation between the different panels.

B.4 Libraries

The following external libraries were used during the development of the prototype:

Name	**Function**	**URL**
SqliteJDBC	In-memory database	`http://www.zentus.com/sqlitejdbc/`
MaxMind GeoIP	IP geolocalization	`http://www.maxmind.com/app/java`
Commons Codec	Base 64 coder	`http://commons.apache.org/codec/`
google-gson	JSON library	`http://code.google.com/p/google-gson/`
JAX-WS	HTTP server	`http://jax-ws.java.net/`

Table B.4: External libraries used in prototype

The graphics used in the prototype were provided by the Tango Desktop Project and the Mozilla Foundation, both public domain sources.

Java code lines were generated using David A. Wheeler's 'SLOCCount'

Appendix C

Off-the-Shelf Implementation

As a side effect of choosing the HTTPS protocol, any compatible off-the-shelf software can be used to implement the service. This appendix contains a summary of how these tools can be used.

C.1 Server

Any scriptable, HTTPS compatible web server can be used. Apache (`http://httpd.apache.org/`), Lighttpd (`http://www.lighttpd.net/`), and Microsoft IIS (`http://www.iis.net/`) are popular choices.

The script should be placed in a secure folder. How to configure the web server to use HTTPS is included in the documentation. In order to password protect access to the server, Basic HTTP authentication is used. For apache, this is done by creating a .htpasswd file with an encrypted user-password combination:

```
htpasswd -c /usr/local/apache/passwd/passwords username
```

and the following command in the .htaccess file:

```
AuthType Basic
AuthName "Restricted␣Files"
AuthUserFile /path/to/.htpasswd
Require valid-user
```

A script must be used to transfer a request into a set of JSON log entries. A popular and easy-to-install example is PHP scripts, which are widely supported. It interacts natively with most popular database management systems, and does not require any configuration besides the installation.

Assuming a correctly setup rsyslog installation, the following PHP script is in fact a fully usable server for the prototype client:

```
<?php

if (!isset($_GET['timeFrom']) || !is_numeric($_GET['timeFrom'])
||  !isset($_GET['timeTo'])   || !is_numeric($_GET['timeTo']))
{
      die("Invalid␣parameters");
}

$link = mysql_connect("<host>", "<username>", "<password>");

if (!link) {
      die("Could␣not␣connect:␣" . mysql_error());
}

$selected = mysql_select_db("log", $link);

if (!$selected) {
      die("Could␣not␣use␣database␣'log':␣" . mysql_error());
}

$action  = mysql_real_escape_string($_GET['action'], $link);
$minTime = (int)$_GET['timeFrom'];
$maxTime = (int)$_GET['timeTo'];
$minSev  = (int)$_GET['minSev'];


if ($action == "update")
{
      $json = "";

      $query = sprintf("SELECT␣*," .
                       "UNIX_TIMESTAMP(DeviceReportedTime)␣as␣Time␣" .
                       "FROM␣␣␣␣SystemEvents␣" .
                       "WHERE␣␣UNIX_TIMESTAMP(DeviceReportedTime)␣>␣%d␣" .
                       "AND␣␣␣␣␣UNIX_TIMESTAMP(DeviceReportedTime)␣<=␣%d␣" .
                       "AND␣␣␣␣␣Priority␣<␣%d",
                               $minTime, $maxTime, $minSev);

      $result = mysql_query($query);

      while ($row = mysql_fetch_assoc($result))
```

```
        {
                $json .= "{";
                $json .= "'id': " . $row['ID'] . ', ';
                $json .= "'timestamp': " . $row['Time'] . ', ';
                $json .= "'facility': " . $row['Facility'] . ', ';
                $json .= "'severity': " . $row['Priority'] . ', ';
                $json .= "'message': '" . $row['Message'] . "'";
                $json .= "}\r\n";

                echo $json;
        }
}
else
{
   echo "LogWheels Prototype";
}

mysql_close($link);

?>
```

Note that rsyslog does not guarantee a source IP address for every event, thus the IP is excluded from this demonstration. The lack of an IP significantly reduces the value of the visualization scheme. As a consequence of this, if the log management infrastructure lacks IP addresses, the server should attempt to extract them from the log messages or other sources.

The above PHP server would under no circumstance fulfill the requirements of an actual organization, but it shows the minimum amount of work required to setup a server which communicates effortlessly with the client, adhering to the standard.

C.2 Client

All web browsers can be used to access any server supporting the full protocol. Because the protocol specified that JSON should be output rather than HTML, the URL can not be accessed directly. JSON is, however, perfectly adapted for the use of AJAX (asynchronous JavaScript and XML).

JavaScript can be used to asynchronously access the file, parse the JSON and convert it to a modification on a HTML page. Using the open source

library JQuery (http://www.jquery.com/), the following lines of code fetches the last log entries:

```
$.ajax({
   url: '<url>',
   username: '<username>',
   password: '<password>',
   success: function(result) {
      // Load JSON incidents and display
   },
   error: function(result, status, errThrown) {
      // Output error message
   }
});
```

Displaying data according to the rules for the visual transformation is harder in HTML than using the Java canvas. The new HTML5 standard supports Canvas, but the technology is still immature, and only browsers released the past few years supports the new standard.

Java applets can be used through a browser, but the technology is practically obsolete by today's web standards. A more popular approach is Flash.

Appendix D

Syslog Details

Using the rsyslog ommysql module results in the following table:

```
CREATE DATABASE Syslog;
USE Syslog;
CREATE TABLE SystemEvents
(
      ID int unsigned not null auto_increment primary key,
      CustomerID bigint,
      ReceivedAt datetime NULL,
      DeviceReportedTime datetime NULL,
      Facility smallint NULL,
      Priority smallint NULL,
      FromHost varchar(60) NULL,
      Message text,
      NTSeverity int NULL,
      Importance int NULL,
      EventSource varchar(60),
      EventUser varchar(60) NULL,
      EventCategory int NULL,
      EventID int NULL,
      EventBinaryData text NULL,
      MaxAvailable int NULL,
      CurrUsage int NULL,
      MinUsage int NULL,
      MaxUsage int NULL,
      InfoUnitID int NULL ,
      SysLogTag varchar(60),
      EventLogType varchar(60),
      GenericFileName VarChar(60),
      SystemID int NULL
);
```

Facility	Description
0	kernel messages
1	user-level messages
2	mail system
3	system daemons
4	security/authorization messages
5	messages generated internally by syslogd
6	line printer subsystem
7	network news subsystem
8	UUCP subsystem
9	clock daemon
10	security/authorization messages
11	FTP daemon
12	NTP subsystem
13	log audit
14	log alert
15	clock daemon
16	local use 0 (local0)
17	local use 1 (local1)
18	local use 2 (local2)
19	local use 3 (local3)
20	local use 4 (local4)
21	local use 5 (local5)
22	local use 6 (local6)
23	local use 7 (local7)

Table D.1: Syslog facilities

Severity	**Description**
0	Emergency: system is unusable
1	Alert: action must be taken immediately
2	Critical: critical conditions
3	Error: error conditions
4	Warning: warning conditions
5	Notice: normal but significant condition
6	Informational: informational messages
7	Debug: debug-level messages

Table D.2: Syslog severity levels