CHURCHLADY: A STUDY IN INTRUSION DETECTION SYSTEMS

by

Selena Marie Brewington

A THESIS

Presented to the Department of Computer
and Information Science,
and the Honors College of the University of Oregon
in partial fulfillment of the requirements
for the degree of
Bachelor of Arts

June 1999

Approved: 

Professor Virginia Lo

An Abstract of the Thesis of

Selena Marie Brewington for the degree of Bachelor of Arts

in the Department of Computer and Information Science to be taken June 1999

Title: CHURCHLADY: A STUDY IN INTRUSION DETECTION SYSTEMS

Approved: [redacted]

Professor Virginia Lo

In the wake of the rapidly expanding Internet, proactively securing computer systems using intrusion detection tools has become an essential function of system administration. This thesis focuses on Unix–like operating systems and describes Churchlady, a prototype implementation of an alert–level based intrusion detection system. Churchlady incorporates four utilities currently in use at the University of Oregon to perform intrusion detection. Through the alert–level framework, and the creation of a uniform event format, Churchlady accomplishes the goals of information highlighting, selection, aggregation and interpretation. Also, a framework is established for automating responses to intrusion–related events, and future work is suggested in the area of decision making which utilizes alert–levels.

Their custom concern for
the people – build up monuments
and steeples to wear out our eyes.
"Custom Concern," Modest Mouse

To my mom.

Acknowledgments

No thesis is created in a vacuum, and as much as everyone else, I am the product of the many individuals who contribute in large and small ways to my life.

If I can point to a single source of inspiration for my topic, it would have to be my work at the Computing Center. In the fall of September 1997, I took a job with Network Services to work on issues related to network and host security. Dale Smith is the director of that group, and I owe much to the work environment he created there.

Steve VanDevender was a mentor who told me, in not so many words, that my job with Network Services wasn't so much about security as it was about correct system management. Thanks to him and Bob Jones for offering advice and direction.

Anne Laux, my long–suffering friend, proof–read many copies of my prospectus and offered a sympathetic ear when I needed it. Many thanks.

Daniel Zappala, Ginnie Lo and Paul Petrequin were gracious enough to form my thesis committee. USENIX supplied a big grant to work on SANTA, and indirectly, Churchlady.

I also want to thank Jeremy Zane, who pointed me toward my first computer job; Chandra Chitneni, who hired me even though I had no experience; to John David Duncan for listening to me and reading yet another copy of my prospectus; Hervey and Dan at Microservices, for being great bosses and all around nice guys who didn't care that I was running Linux on my desktop machine; to Gregston Chu, who tirelessly answered my questions about Linux, Unix and computers in general; to Jenny Noble for putting me up at her place for so many nights; and to Michelle Brown for demanding that I go drink beer with her.

Finally, Dustin Harris was supportive, consoling and ego–boosting when I really needed it. My love and gratitude goes out to you.

Table of Contents

I. Introduction 10
A. Motivation 12

II. Background 13
A. Intrusion Classifications 14
B. Anomaly Detectors 17
C. Misuses Detectors 18

III. Problem 21
A. IDS at the University of Oregon Computing Center 21
B. Example: How sumon handles one type of intrusion 21

IV. Churchlady: The Prototype 24
A. Overview 24
B. Churchlady Model 25
C. Churchlady Example 27
D. Status 28

V. Evaluation of Churchlady 29
A. Strengths 29
B. Weaknesses 30
C. Conclusion 30
D. Future Work 32

Works Cited and Consulted 34

Appendices
A. Chuchlady Implementation 38
B. Glossary 41
C. Figures 44

List of Figures

1. Reported security incidents at the University of Oregon, 1997–99 45

2. State transition diagram of a root–exploit 46

3. Data flow in Churchlady 47

4. Event() audit record structure for Churchlady 48

5. Sample output from `sumon` 49

6. Sample output from COPS 50

7. Sample output from `tcplog` 51

8. Signature event list for `tcplog`, `sumon`, `md5sum` and COPS 52

9. `sumon` and `md5sum` state transition diagram 53

10. `tcplog` connection state transition diagram 54

CHAPTER I

INTRODUCTION

> UNIX was not designed with security in mind, and for that reason no UNIX system can be truly secure. (Nemeth 539)

Security maintenance is a necessary and inconvenient part of a computer system administrator's job. In the *UNIX© System Administration Handbook*, the authors go so far as to say that security only increases in inverse proportion to convenience (Nemeth 540). The modern computing environment typically places Unix–like systems on the Internet. Because access to these systems through the Internet is largely uncontrolled, there is an increased possibility that a computer will be attacked and penetrated with potentially catastrophic results. This requires that system administrators be more vigilant and make use of intrusion detection tools. These tools are designed to detect suspicious activity and/or respond to that activity by notifying systems administrators or by taking other protective actions.

System administrators often deal with security issues the same way that they handle normal configuration and monitoring of a system: they write scripts. Scripts are short programs that perform a small number of specialized tasks. Because they are written quickly to solve an immediate problem, syntax and usage are often idiosyncratic, and they are rarely designed with reuse or integration in mind. There are a large number of tools that offer piecemeal assistance with intrusion detection, but few that integrate a number of these tools together to create a comprehensive system.

Churchlady is an intrusion detection prototype that establishes a framework for the use of pre–existing, independently developed intrusion detection tools in a coordinated, more effective manner than is achieved by each tool alone. It monitors a computer

system for malicious violations of computer security by consolidating the information gathered by the various tools into once place and conducting an integrated analysis of the data.

Each intrusion detection tool integrated into Churchlady is a monitor that records a specific type of information. Some tools generate normal profiles of system activity and report anomalies, while others search for a signature set of characteristics that may identify an intrusion. All information is typically recorded in log files that list events in chronological order, usually with the date and time of the activity. Because log files contain so much information, the administrator who sifts through them is subject to information overload. Churchlady will help prevent information overload by reducing the amount of data that needs to be examined. Using the filtered information from each tool, Churchlady analyzes the relationships among events to more efficiently and accurately alert the management staff about potential intrusions.

My work in developing Churchlady has led me to a rich understanding of the subtleties involved in implementing an effective intrusion detection system (IDS). Churchlady was successful in reducing the huge amounts of data amassed in logfiles through its information filtering engines. Information integration was also achieved through the configurable filtering mechanism for the database back end. Also, these tedious tasks were automated and the model calls for further work in creating automated responses to the aggregated information. On the other hand, Churchlady does not afford greater insight on how to generically respond to events based on their relationships to each other. This result, however, does not mean that the filtering model itself failed. Rather, the data set may not be rich enough to support my framework. Expanding the

Churchlady model to a richer domain or set of tools is an interesting topic for further research.

Motivation

Practical problems in system administration are what motivated this project. Because Unix was not designed with security in mind, most security measures are clumsy and unfocused. Log files are enormous and disorganized, recording everything from ordinary system events to critical failures. Few, if any, attempts are made to automate responses to intrusions. Tools to solve security crises are often developed on demand. While these tools are often useful immediately, they are not integrated well with existing systems, and their usage, syntax and logging styles are non–standard.

The frequency and severity of intrusions demand higher security than ever before, but the overall workload of a system administrator, or sysadmin, has not adjusted to accommodate increased time spent on security. Sysadmins often wear a number of hats in their respective workplaces, adding system administration on to other job responsibilities without taking away any of the other responsibilities that their jobs entail. When problems arise, fast and effective solutions are valued over lengthy research to find the best possible solution available. Taking pre–existing tools and trying to make them work better without rewriting them is a common first approach to an urgent problem.

Reusing old programs may not be the best approach, but it is often the most expedient in terms of a sysadmin's time. And if that approach ultimately fails, usually the sysadmin has learned enough to rewrite the tools so that they work the right way, or at least in the way that is needed. This thesis works from those assumptions, attempting to implement an intrusion detection tool using what is already available.

CHAPTER II

BACKGROUND

In a recently conducted study, over $136 million in damages was reportedly caused in 1998 by computer "hackers" (The Industry Standard). However, when considering the state of computer security overall, it's important to keep in mind that "[c]omputer crime...tends to be vastly underreported" (The Industry Standard). Companies rarely like to disclose the fact that their machines have been compromised, particularly when large amounts of money are at stake. In January 1989, news broke on a group of Australian hackers who had stolen more than $500,000 from Citibank, a large American banking and crediting company. The attacks went unreported for over six months after they were first noticed. In addition to the money transfers, thousands of credit card numbers, and their corresponding accounts that were worth millions of dollars, were being re–distributed on computer bulletin boards by the Australian computer underground (Dreyfus).

Closer to home, ten system compromises were reported to the a University of Oregon security mailing list in 1998, and six were reported in the first half of 1999 (Figure 1). The number of detected unauthorized probes of security has also increased substantially over the past two and a half years. This illicit activity, for the most part, fits into an identifiable pattern.

To develop an intrusion detection systems, the different types of intrusions must be identified and then characterized. After characterizing the vulnerabilities, generalizations may be made about how to detect or prevent the intrusions.

Intrusion Classifications

There are many types of security violations, and a number of ways to classify each one. Most attacks can be identified as either active or passive. Active attacks occur when action is taken to circumvent a security feature. Passive attacks usually involve a program that observes and records information about a user process in a way that the user cannot normally detect. The origination point of an attack further divides the groups, either coming from a legitimate user or an external attacker.

Active attacks include viruses, worms, buffer–overflows, and some denial–of–service attacks. These attacks are sometimes used together. A virus is a "program that searches out other programs and 'infects' them by embedding a copy of itself in them" (Raymond). A virus cannot, however, infect another computer without assistance. A human would have to transport the virus to another computer via a floppy disk or by copying an infected file to a healthy computer.

A worm is a special kind of virus that spreads an infection by using computer networks to travel from machine to machine. A famous instance of this attack was the Morris Worm. In 1988, Robert T. Morris unleashed this program and "brought the Internet to its knees" (Raymond). His program connected to computers over the network and tried to break into them using well–known security vulnerabilities. The worm was wildly successful, and because of what Morris described as a programming error, it created a huge amount of network traffic which disrupted the proper functioning of the Internet.

Buffer–overflow attacks are most often used to get root access, i.e. most privileged access to a computer system. Buffer–overflows take advantage of inadequate bounds–

checking on data stored in memory. For example, a programmer can write a program that declares an array of data, and then puts data elements into that array. This type of array has a finite amount of memory space allocated to it. This programmer might not check to see that the data stored in the array is the correct type and size, and might accidentally write data beyond the finite space an array is supposed to occupy. it is possible to exploit programming mistakes like these and force that poorly–written program to execute arbitrary commands. Depending on the circumstances, that arbitrary command could allow a person to gain root–level access to a computer system. Overflows can occur in a variety of memory areas associated with a user's program. For the past couple of years, "smashing the stack" attacks have been the most widely recognized buffer–overflows (Aleph1).

Denial–of–service attacks are designed to prevent legitimate users from using a computer system. These can come in a number of forms. A very crude denial–of–service is an email bomb. A malicious person could send thousands of email messages to a specific email address. The recipient might not be able to open his or her mailbox because it would be enormous, and that person's computer might malfunction or cause an application to crash. The intended target of abuse is not the only person that can be affected by an email bomb. The entire server trying to process the email might crash, denying service to all of the other users of that computer system. Other denial–of–service attacks include overloading the network with control messages, such as a SYN–flooding attack ("TCP SYN Flooding"), or the ping–of–death ("Denial–of–service Attack via ping").

Passive attacks are harder to detect than active attacks. Generally, a passive attack

does not generate a change in the observable state of the target system, unless system checking tools are used to periodically verify the integrity of data. Active attacks may be used to get unauthorized access and install a passive tool. Typically, these tools collect important information like passwords. The tool is installed and left to run for a few days or weeks to ensure that some data useful to the attacker is gathered.

A Trojan horse is a "malicious, security–breaking program that is disguised as something benign, such as a directory lister, archiver, [or] game" (Raymond). For example, a Trojan horse might be embedded in telnet, the software that is used to establish connections to remote computer systems. When a user tries to connect to a computer system via telnet, the Trojan horse asks for a password. Before allowing the user to login, the Trojan horse copies that password into a file stored in a secret location. Later, the attacker can copy the file containing the passwords and then use any account he or she has the password to.

Another kind of passive attack are characterized by network sniffers that attempt to capture useful information from the hardware network connections. Some sniffers are designed to detect particular kinds of information, while others collect everything. For example, `linsniffer` "is designed to be small and do one thing – capture passwords" (Kill9). `tcpdump`, on the other hand, is capable of capturing many different kinds of network information, but can be configured to focus on a certain type (Network Research Group).

Computer security attempts to prevent and correct the many situations which develop into security vulnerabilities. The previous section barely scratched the surface in describing the types of attacks that are possible. However, all the different kinds of

attacks can be classified into broad categories, and discrete actions leading to compromise identified.

Intrusion detection tools designed to detect these attacks are generally grouped into two categories: anomaly, and misuse detectors. Anomaly detection, such as outlined by Denning's paper "An Intrusion Detection Model," create statistical models of behavior on a computer system to determine when activity is normal or abnormal. Misuse detection, on the other hand, relies on pre–defined sequences of actions, or a single key event being detected. Both approaches have value, and each has a number of shortcomings.

Anomaly Detectors

Anomaly detectors range from the system administration who reads through log files to sophisticated statistics–generating programs. The underlying principal depends on a known normal state and comparing events to that state to determine whether or not the event falls within the normal range of activity.

System logging is the most widely used anomaly–based intrusion detection tool. System administrators manually check logfiles generated by system loggers, searching for abnormal events. `sysklogd`, a popular logging utility, receives information from various programs that are running, and records that information in a file. Logfiles end up holding very large amounts of information. For example, by default, `sendmail` logs all of the connections that take place when email is sent or received. On *darkwing*, over 50,000 email messages are sent in a day (VanDevender). On February 9, 1999, over 39,000 lines of mail–related log entries were recorded in an ten hour period, generating a logfile of about eight megabytes (VanDevender). These files are so large that a system administrator might not catch isolated anomalous log entries by merely scanning the file

visually. To deal with these huge amounts of information, some sysadmins just look at every file that is created and read through everything. Others write scripts, or use combinations of utilities to search through the files and find important information. *WOTS* performs exactly these tasks by aggregating data in a useful way and finding important events (Curtis).

Automated anomaly detection, as suggested by Dorthy Denning, provides distinct advantages over manual approaches. Denning's intrusion detection model strives to establish a normal pattern of activity by collecting data and generating statistical profiles for system activity. Denning identifies three key types of metrics: event counts, intervals between events and resource usage measurements. Each system event is recorded in an audit record that includes the actions, the subject and object of actions, metrics, time–stamps and exception conditions.

Anomaly detectors have the important ability to detect previously unknown vulnerabilities. However, the unknown vulnerability must effect some sort of measurable change in the system. If the auditing program does not generate data for an event or events involved in an intrusion, then the model fails to detect the vulnerability. Profiles generated from data samples also may not be accurate indicators of future performance. On a system whose usage patterns are erratic, statistical profiles can not accurately identify normal behavior, causing the IDS to generate more anomalies and a large number of false positives.

Misuse Detectors

Tools designed to detect misuse are supplied with events or sequences of events that signal an intrusion. There are two approaches to misuse detection. Under one model, an

IDS consults a database of suspicious activities that is maintained by some third party, or the system administrators themselves. Under the second model, a state transition analysis is conducted to identify key actions, or actions that lead to security compromises, to be monitored for by the IDS. Users can customize the list of security problems that are monitored.

As new vulnerabilities are discovered, each tool's database must be updated. These databases of faults introduce a large maintenance overhead. The development of large repositories of security information have made the task of keeping these databases up–to–date much easier. However, even the most quickly updated tools are vulnerable to exploits which have not yet been made public. If a new security exploit is used that has not yet been entered in the database, the tool will never detect the intrusion. To anticipate new exploits, misuse detectors might search for an event common to the known exploits. Unfortunately, this sort of analysis is difficult, and because misuse detection tools are generally run at a less frequent rate and are more well–known than profiling tools, there is a very real possibility of the tool being disabled or tricked in some way before it has a chance to run again and detect the intrusion.

Checking for unexpected files in a computer system is a simple and easy to implement kind of misuse auditing. *Tripwire* (Tripwire) generates a database of *checksums* for important files, and checks if those files have been modified by comparing what is stored in the database with a freshly generated checksum (ZDNet).

State transition analysis offers an alternative to the huge databases of vulnerabilities for misuse detectors.

> State transition analysis is based on the premise that all computer penetrations share two common features. First, penetrations require the attacker to possess

> some minimum prerequisite access to the target system... Second, all penetrations lead to the acquisition of some previously unheld ability. (Ilgun 184)

Penetrations are detected by creating models of initial and compromised states and the "state transitions that an attacker performs to achieve the compromise" (185). This way, intrusion scenarios are decomposed into their key, or signature, actions. By describing each of the actions necessary to effect a penetration, the signature action(s) can be identified that would prevent the penetration from occurring.

A common intrusion scenario involves a user installing an exploit program that takes advantage of a security vulnerability in a program installed on the target system. When an exploit program is executed, it inserts special instructions that allow the attacker to execute another program, such as a shell, at elevated privilege. The net gain is that the attacker is now the owner of a new process whose privileges are greater than before.

When state transition analysis is applied to this scenario, the signature action responsible for the actual compromise is obvious (Figure 2). Between states S_{c-1} and S_c, the attacker executes a shell with root permissions. As a result, in state S_c now contains the root–owned shell process and the attacker has root privileges. Both of these state changes happen in the transition between S_{c-1} and S_c. If that state transition could be detected or prevented, than state S_c could be avoided or detected and then terminated.

CHAPTER III

PROBLEM

IDS at the University of Oregon Computing Center

The system administration group in the University of Oregon Computing Center, composed of Bob Jones, Steve VanDevender and Jon Neher, approach the problem of security in a number of ways. They use `syslogd`, Tripwire, and COPS, as well as the locally developed `sumon` and `iomon` to monitor for intrusions. Also, they maintain an independent log host, isolate the network that important systems operate on and support email notification of intrusions sent to remote hosts. All of these actions together function as an intrusion detection system.

This IDS is far from perfect, however. Information about possible intrusions is spread across many different mediums. Multiple logfiles, binary files containing data describing user activity and email all contain information critical for putting together the whole picture when an intrusion occurs. System administrators must comb through this log information and it rarely illustrates anything other than the chronological relationship of events. The systems staff has no comprehensive strategy for aggregating, reviewing or reducing the information that needs to be monitored. They also lack automated responses to intrusions that would make the system more secure, and no resources are available to develop an entirely new, integrated and automated system to perform these tasks.

Example: How `sumon` handles one type of intrusion

The UO Computing Center staff developed `sumon` in direct response to a common security problem: unauthorized root access, or the situation where unauthorized users attempt to gain elevated privileges to all the system resources at great risk to the integrity

of the system. sumon checks for *child processes* that are owned by root, but whose *parent processes* are not. Details of sumon's operation and design principles are included below because they form a central part of Churchlady, which builds on sumon's strengths while attempting to correct its weaknesses.

Examples of processes detected using sumon are programs run with *set–uid root* permissions. These programs must run *as–root*, or at an elevated level of privilege, because the programs must access special files or devices that are normally only accessible to the superuser. An intruder who gains unauthorized access and spawns off a root–owned shell from an unprivileged user account would be detected because the parent process – the initial login by a legitimate, although maliciously inclined, user – should not have the ability to create a root–owned child process, which possesses greater privileges than the parent.

To create its audit records, sumon uses the system utility ps, a monitoring program that generates output about running processes on a system. The search criteria sumon uses is simple: find any process that has unauthorized root privileges. The *detection level* established by sumon is significantly different from audit records normally recorded via syslogd. While there may not be a general descriptive term for the detection level of all data found in files such as /var/log/messages or /var/log/syslog, most programs executed by ordinary users do not log output messages to syslogd. Unless process accounting is turned on, there is generally no reliable record of the programs that a user executes when he or she is logged in. Through a process known as *polling*, sumon creates a record of a user's activity if information in ps's output matches the search criteria.

By focusing on a critical and detectable change in the state of a computer system, the root–compromise is made detectable immediately upon appearing. Responses to a vulnerability detection are a critical part of sumon's design. Responses include:

- attach to running processes and capture the activity of the process,
- email periodic updates to the staff about the activity of the errant process, and
- create archives of directories associated with the process.

Because the days of around–the–clock human surveillance of computer systems are near gone, automated responses to intrusion are essential. A system administrator might be asleep, on vacation, or out to lunch when intrusions occur, so a computer system needs self–defense mechanisms.

sumon proved its effectiveness this past year by detecting a real intrusion by a user exploiting a recently disclosed security vulnerability. Observing the evolution of sumon from an interesting idea and prototype to a complex and useful tool raises questions about how sumon's underlying design assumptions could be used in a more general context to improve the way that logging and system maintenance is handled across campus. The key shortcomings that Churchlady is designed to address were: 1) inability to identify Trojan horse applications that are authorized to run as–root, 2) lack of automated, preventative responses to intrusions, and 3) lack of consistent event–reporting format.

CHAPTER IV

CHURCHLADY: THE PROTOTYPE

Overview

Churchlady is designed to address some of the shortcomings of the IDS in place at the University of Oregon Computing Center. Churchlady combines four pre–existing monitoring tools in use at the Computing Center, enriching them by creating an environment that incorporates more sophisticated information processing and decision–making as well as increased automation of preventative actions.

My goals for Churchlady are both information and decision–making related. Information goals include highlighting, selection, aggregation and interpretation. Decision–making goals include informing the system administrator of anomalies or correct functioning of a system, correcting the state of the system, protecting a system from intrusion and automating all of these things.

Information highlighting, selection and aggregation are all accomplished through filtering rules. These rules search for and save suspicious or interesting events, and discard uninteresting events. When enough suspicious events are collected, or a high alert–level event occurs, related events are emailed to a system administrator. Aggregation is accomplished by grouping related events together regardless of the logging utility that recorded them, and minimizing the number of similar events in a certain time frame that generate alerts. Information interpretation is performed by a second filtering ruleset, which analyzes events after they have been grouped together.

Automation is a very important goal of this system. Churchlady should help perform repetitious tasks. Killing processes, grouping together and highlighting events are all

things that many sysadmins do "by hand." Having Churchlady complete those tasks without laborious human intervention makes the job of administering systems more pleasant.

By generalizing fault–detection to a "well–known process level", Churchlady creates an intrusion detection system that is robust, light–weight and easy–to–use. *Well–known* describes process activity that is logged by commonly used logging utilities. Churchlady should detect known security vulnerabilities, not put undue stress on the system it is monitoring, and be simple to install and configure.

Churchlady Model

Conceptually, Churchlady is broken into two parts: a decision–making filter for system events, and a database backend that stores, sorts, and periodically discards events. This conceptual model is described by Figure 8. Each independent utility generates its own log file. This log file is interpreted by a Perl filter which translates each event logged by each utility into a common Churchlady event format. Then, the decision–making filter looks at each event and determines what kind of alert–level will be assigned, what information will be stored in the Churchlady event structure (see Figure 4 for an example), and where each event will be stored in the database backend. Next, the database backend performs maintenance work on the events by periodically discarding old events and recognizing when a response to an event is necessary.

Churchlady was designed to filter events from the independently functioning utilities described below:

- sumon: Logs events describing root–owned processes. Information about the processes that are detected is logged in an email message (Figure 5).

- COPS: Logs anomalous events designed to detect common security vulnerabilities, such as incorrect permissions on files and directories, or malformed entries in special files. Generates a log file that has information about each anomaly found per line of output (Figure 6).
- `md5sum`: Creates a unique identifier for any file that can be used to verify that the contents of the file have not been changed since the last time the identifier, or checksum, was generated. The program `md5sum` outputs a character and number string to standard output.
- `tcplog`: Logs each TCP connection to a computer. `tcplog` generates events in the style of `syslogd` (Figure 7). Each line contains a different record.

Imbedded in this architecture is an alert level system whose goal is to provide a general indicator of the security status of a computer. The four alert levels, INFO, WARN, ALERT, and FATAL, should facilitate aggregation of the decision making process by specifying generic actions to be taken at different levels of alert. The advantage of using an alert level system is that less state information needs to be maintained about a specific intrusion scenario to perform important tasks. To create an alert level system for Churchlady, the signature actions for each event logged by the logging utilities were identified, and corresponding alert levels generated by each event were defined (Figure 8). Dependencies between the different events were identified and two scenarios that might benefit from the alert level system emerged. They are modeled in state–transition format in Figures 9 and 10.

Finally, Churchlady implements a common event format to which all events from the independent utilities are translated (Figure 4). Each event contains the critical,

identifying features contained in each record. By creating a universal format, information aggregation and interpretation can be automated with generic filtering rules. Without a common format for Churchlady, human intervention would be required to interpret each different event format.

Churchlady Example

The differences between the current IDS and Churchlady are illustrated well by following their respective responses to a root–compromise intrusion situation (Figure 2). In this scenario, a malicious user gains access to a system in some manner. Using this access, the user gains root–level privileges, compromising the system. The attacker then uses the root–level access to make unauthorized modifications to the system.

The current IDS uses `sumon` as the detection mechanism. `sumon` identifies the user that possesses the unauthorized root access. After successful identification, `sumon` sends email to system administrators that contains information about the intrusion and creates an archive of the user's home directory. No other tools that function as part of the overall IDS produce output that is marked as relevant to this intrusion.

There are several problems with the existing system. Information that is relevant to the intrusion is logged by other utilities, but is not included in the `sumon` output. There is no way to distinguish between a Trojan horse or a legitimate system application running with root privileges because certain applications are statically marked as "safe" in `sumon`'s configuration file. No immediate action is taken to secure the system from unauthorized modification by the intruder.

Churchlady handles the root–compromise situation differently. First, the root process is detected. Using both output from `sumon` and `md5sum`, the program which

was executed is checked to determine if it is authorized to run at elevated privileges. If the process is unauthorized, the event is passed on to the database and grouped in with related events. The events might be related by any number of characteristics including user or originating host. Email notification is sent to the system administrator with comprehensive event information based on the groupings. Finally, Churchlady will offer the ability to automatically kill an errant process or block the user's access to the system in some way.

Status

A prototype implementation of Churchlady exists (Appendix A). It is implemented in Perl and C++, and was developed on the Linux operating system. The data structures are complete and have been tested on sample sets of data translated by the Perl preprocessor.

My focus throughout the implementation stage of this thesis was the development of an alert level system to perform important aggregation in the decision making portion of the project. As such, the alert level system is incomplete. Because the responses to the two scenarios have inherently different characteristics and require different responses, there was no general ruleset to handle both cases. A different alert level was assigned to each scenario, but without additional test cases, no general solution is available.

No formal testing of the system has been conducted.

CHAPTER V

EVALUATION OF CHURCHLADY

Strengths

Churchlady's approach to intrusion detection offers important improvements over the current IDS in use at the UO Computing Center. Data is integrated and aggregated by the filtering mechanisms. Rather than relying on manual searching through many logfiles for related information, logfiles are processed by Churchlady in an expedient manner.

The framework for automated response enables instantaneous reactions to critical situations. Previously, quick responses from system administration staff were relied on to protect the system in all situations. System administration staff should allow the computer to correct intrusion situations as quickly as possible, without human intervention.

The decision–making capability in the filters reduces the time that is spent by system administrators tracking down each problem, and the uniform format for events originating from different logging utilities makes interpretation of each event much easier. Previously the logging utilities incorporated in Churchlady did nothing to correlate the information logged by each, or to consistently create data structures to define events. A responsive environment that supports information–gathering actions in response to suspicious events further enhances the intrusion detection ability of a system.

By using existing tools, the time spent developing software was greatly reduced. System administrators rarely have enough spare time to develop entirely new applications. The existing tools each perform specialized tasks very well, and system administrators benefit from organizing the output of each in a systematic way.

Weaknesses

The primary weakness of Churchlady is found in its use of the four logging utilities described in the previous section. Specifically, only two major types of intrusions that are amenable to an alert–level system were discovered through state–transition analysis. In the sumon scenario, the legitimacy of a running process is determined by checking if the binary that executed the process exists and that its state corresponds to a legitimate binary (Figure 9). The second scenario searches for a number of connections to unauthorized services, and if more than a threshold number occur, causes an alert level increase (Figure 10).

Both of these situations could be handled much more simply by augmenting sumon, and by creating a very simple script to watch for errant TCP–connections. The other security vulnerabilities that could be detected using the four event loggers are not dependent on one another in a detectable way; in general, a state change indicated by one event cannot be corroborated by another event, using the tools available, to come to a different conclusion about the state of the system. Each high level event is serious enough by itself that each demands immediate attention. There is no need to wait for additional information to inform the system administrator that action needs to be taken.

Conclusion

Developing the prototype for this application opened my eyes to many deficiencies in current logging techniques used by IDS. Additionally, developing a C++ application in the Unix environment presents many interesting problems in the area of debugging and rapid development.

In order to facilitate easy reconfiguration of system tools, the level of logging

information should be dynamically configurable. Rather than having to kill and restart a process, each process should be responsible for monitoring the configuration files that are related to it and updating behavior accordingly. `cfengine`, `ippl` and WindowMaker are utilities that utilize this configuration model. Dynamic reconfiguration reduces the number of processes that must be killed and restarted, resulting in fewer run–time errors or loss of data resulting from the early termination of a process.

Overall, the granularity of logged information needs to be increased. Many programs log no information at all. Standard utilities should be linked with libraries that facilitate logging of anomalous activity. Rather than imposing another layer of monitoring, all programs should be capable of generating their own event records in response to illegal or suspicious behavior. Traditional process accounting generates far too much useless information, leading many sysadmins to ignore these logs until after a major intrusion is discovered. Suspicious actions leading to the intrusion are often visible through the logging utilities, and many damaging actions committed before the intruder is discovered. If a library of auditing functions were created and linked with common utilities, many damaging actions could be simply prevented, and suspicious actions detected before major damage to a system occurs.

Establishing a common event structure for intrusion detection tools has an effect similar to increasing the granularity. The common event structure gives a frame of reference for future application developers as to what level of log information is interesting to the people who will eventually be using the application, and has the added benefit of speeding up an IDS that relies on the information that is logged.

A higher level view of these utilities reveals their inability to communicate with each

other at all. An IETF working group called Common Intrusion Detection Framework (CIDF) has written several documents that address those concerns. They are establishing a Common Intrusion Specification Language (CSL) to work with General Intrusion Detection Object (GIDO). CSL establishes a uniform way of describing events and GIDOs are the objects generated by an intrusion detection system to describe a potential intrusion.

Developing this application in C++ has proved to be a significant liability. I spent many hours debugging NULL-pointer exceptions and odd printf() bugs. I did find a debugging utility called ddd (Zeller) that functions as a front end for gdb (Cygnus). ddd proved to be invaluable for debugging, and after I started using it, I more than halved the time I spent debugging each segmentation fault.

Future Work

One of the overarching goals for development of Churchlady was to integrate it with a graphical environment. Through a grant from USENIX, a fellow student and I have finished the first version of SANTA, a graphical system administration tool written in Java. A re-implementation of Churchlady to work directly with SANTA is planned. The Event structure and alert levels are critical for passing information to the GUI. Different alert levels will cause nodes that are displayed in Churchlady to have different colors, such as red for a fatal alert level, and green for default. The network pictures generated by this scheme are of particular interest.

Re-implementing the security tools involved in the development of Churchlady is another topic for further research. Each tool might be updated to produce a standard format event structure, such as the one suggested by CIDF. Also, additional security

tools that, for example, notice when users attempt to open a different user's email, or use commands that access privileged information would be useful to add to the current tool–set. Currently, these sorts of actions are not logged unless process accounting is turned on. More importantly, the relative significance of those types of activities goes unremarked.

WORKS CITED AND CONSULTED

Aho, Alfred V, et al. "The AWK Programming Language." Menlo Park, California: Addison–Wesley, 1988.

Aleph1. "Smashing The Stack For Fun And Profit." Online posting. 9 Nov 1996. Bugtraq. 2 Feb. 1999 <http://www.cse.ogi.edu/DISC/projects/immunix/StackGuard/profit.html>.

Balasubramaniyan, Jai Sundar, et al. "An Architecture for Intrusion Detection using Autonomous Agents." COAST Technical Report 98/05. 11 June 1998.

Barkocy, Muffy. "SATAN." 16 Aug. 1995. 3 Feb 1999 <http://www.fish.com/~zen/satan/satan.html>.

Burgess, Mark. "cfengine." Oslo College. 26 May 1999 <http://www.iu.hioslo.no/cfengine/index.html>.

Burgess, Mark. "Computer Immunology." 28 Oct. 1998. Oslo College. 3 Feb. 1999 <http://www.iu.hioslo.no/~mark/research/AIdrift/AIdrift.html>

CERT. "Denial–of–service Attacks via ping." 5 Dec. 1997. 2 Feb 1999 <http://www.cert.org/advisories/CA–96.26.ping.html>.

CERT. "TCP SYN Flooding and IP Spoofing Attacks." 24 Aug. 1998. 2 Feb 1999 <http://www.cert.org/advisories/CA–96.21.tcp_syn_flooding.html>.

Klaus, Christopher. "Network Packet Capture FAQ." Version 3.01. 2 Feb 1999 <http://www.iss.net/vd/packcapt.html>.

CIDF. "Common Intrusion Detection Framework." 3 Dec. 1998. University of California at Davis. 27 May 1999 <http://olympus.cs.ucdavis.edu/cidf/>.

Conover, Matt. "w00w00 on Heap Overflows." Online posting. 28 Jan 1999. Bugtraq. 2

Feb. 1999 <http://www.geek-girl.com/bugtraq/1999_1/0377.html>.

Cowan, Crispin. "Re: w00w00 on Heap Overflows." Online posting. 28 Jan 1999. Bugtraq. 2 Feb. 1999 <http://www.geek-girl.com/bugtraq/1999_1/0393.html>.

Curtis, Tony. "Perl tools for download." 3 Feb 1999 <http://www.vcpc.univie.ac.at/~tc/tools/>.

Cygnus. "GDB: The GNU Debugger." 30 Mar 1999. 26 May 1999 <http://sourceware.cygnus.com/gdb/>.

Dale, Nell, Mark Headington and Chip Weems. "Programming and Problem Solving with C++." Boston: Jones and Bartlett Publishers, 1997.

Davie, Bruce S. and Larry L. Peterson. "Computer Networks: A Systems Approach." San Francisco: Morgan Kaufmann Publishers Inc., 1996.

Denning, Dorthy. "An Intrusion Detection Model." IEEE Transactions on Software Engineering SE-13.2 (1987): 222-32.

Dreyfus, Suelette. "Underground: Tales of Hacking, Madness and Obsession on the Electronic Frontier." Australia: Mandarin, 1997.

Galvin, Peter and Abraham Silberschatz. "Operating System Concepts." 5th ed. Berkeley: Addison-Wesley, 1998.

GNU. "GNU General Public License." 2 Jun. 1991. Version 2. 26 May 1999 <http://www.gnu.org/copyleft/gpl.txt>.

Ilgun, K., et al. "State transition analysis: a rule-based intrusion detection approach." IEEE Transactions on Software Engineering 21.3 (1995): 181-99.

Kernighan, Brian, and Rob Pike. "The Practice of Programming." Berkeley: Addison-Wesley, 1999.

Kernighan, Brian, and Dennis M. Ritchie. "The C Programming Language." Englewood Cliffs: Prentice–Hall Inc., 1988.

Kill9 (kill9@succeed.net). "Sniffers." 2 Feb 1999 <http://main.succeed.net/~kill9/hacking/tools/sniffers/>.

Lo, Virginia. Personal email correspondance. 2 Feb. 1999.

Martinez, Michael J. "The Great Hacker Divide." 4 Feb. 1999. ABCNEWS.com. 8 Feb. 1999 <http://abcnews.go.com/sections/tech/DailyNews/hackers990203.html>.

Nemeth, Evi, et al. "Unix© System Administration Handbook." 2nd Ed. New Jersey: Prentice Hall PTR, 1995.

Network Research Group. "README." 12 Jun 1997. Lawrence Berkeley National Laboratory. 2 Feb 1999 <ftp://ftp.ee.lbl.gov/tcpdump.tar.Z>.

Prata, Stephen. "C++ Primer Plus." New Delhi: Galgotia, 1995.

Ranum, Marcus, et al. "Implementing a Generalized Tool for Network Monitoring." Paper presented at LISA XI 1997. 26 Oct. 1997.

Raymond, Eric. "The New Hacker's Dictionary." 28 October 1998. 2 Feb 1999 <http://www.spawned.com/jargon/jargon_toc.html>.

Sibert, W. Olin. "Malicious Data and Computer Security." 28 Oct. 1996. 8 Feb. 1999 <http://www.fish.com/security/maldata.html>.

Sobirey, Michael. "Intrusion Detection Systems." 11 June 1998. 3 Feb 1999 <http://www–rnks.informatik.tu–cottbus.de/~sobirey/ids.html>.

THC (awesome@hem.passagen.se). "Archive of Hacked Websites – Hacked Homepages." 8 Feb 1999 <http://www.onething.com/archive/>.

The Industry Standard. "Business Spotlight: Corporate Computer Security." 21 Sept.

1998. 26 May 1999
<http://www.thestandard.com/articles/display/0,1449,1757,00.html?ext.wired>.

Tibbits, George. "Microsoft goofs in China." infowar.com. 10 Feb 1999

Zeller, Andreas. "DDD – The Data Display Debugger." 16 Mar 1999. Technical University of Braunschweig. 26 May 1999 <http://www.cs.tu–bs.de/softech/ddd/>.

Rapalus, Patrice. "Cyber attacks rise from outside and inside corporations: Dramatic increase in reports to law enforcement." 5 Mar. 1999. Computer Security Institute. 27 May 1999 <http://www.gocsi.com/prelea990301.htm>.

Deraison, Renaud. "Nessus Project." 6 May 1999 <http://www.nessus.org/>.

APPENDIX A

CHURCHLADY IMPLEMENTATION

Churchlady is implemented in C++ and Perl, and is object–oriented in design. The portion written in Perl is composed of preprocessing scripts that translate widely varying log formats into a common format that the C++ portion understands.

1. *Event()*

Events are the base data structure for storing all audit record information. Each Event has a set of ten attributes that describe an occurrence on a system (Figure 4). The attributes are defined below:

- *time:* time at which an audit record was logged
- *logger:* logging utility used to generate the audit record
- *host:* the host from which the audit record originated
- *connect_orig:* if the record is of a network event, the network name of the host that initiated the connection
- *user:* name of the user initiating a connection or process
- *process:* name of the process that the audit record was generated for
- *file:* name of file operated on by the process
- *proc_num:* number of the process that the audit record was generated for
- *port_num:* if a network–related event, number of the active port
- *level:* base alert–level that this Event corresponds to

Events have two main methods: *printVals()* and *setVals(char*)*. printVals() prints a single line to *stdout* containing all information in the Event. setVals() accepts a string of characters that corresponds to an audit record. Audit records are accepted in the following format:

TIME HOST C_ORIGIN SERVICE USER PROCESS_NAME PROCESS_NUMBER FILE ALERT_LEVEL

Not all records will have information for all fields, so empty records are represented as '–'. Error checking has not been completely implemented in the parsing routines, although a prototype method *checkbuf()* has been implemented and is a wrapper for all

the text manipulation routines in setVals(). Additional error checking can be inserted in this routine in the future.

2. *EventList()*

EventList is a simple linked–list implementation. EventList inherits from the base class *List*. I use EventList like a queue. New Events are pushed on the stack with *InsertAfter(Event*)*, and as Events age, they are removed from the head of the list with *Remove()*. *CheckTimeVal()* removes Events that are more than one day old.

3. *AccumMap()*

AccumMap is another linked–list inheriting from List. AccumMaps contain *AccumNodes* that describe each EventList in terms of its *AccumType*, a text string that serves as a keyword identifier to differentiate EventLists with the same AccumType, and a pointer to the appropriate EventList. The AccumTypes correspond to fields in an Event that is filtered into an EventList. For example, one AccumType is *HOST*. If an EventList was collecting based on the HOST entry in an Event, each HOST entry in each Event would be compared to the string contained in the AccumNode. If a match is found, the Event is deposited into the corresponding EventList. Conceptually, each AccumNode is a separate filter (Figure 3)

In addition to matching fields in Events, AccumMaps have an *ignore* list of AccumTypes and corresponding character strings. If an Event matches anything in the ignore list, it is dropped. AccumMap attempts to match Events in the ignore list first, so ignores have higher precedence than the filters in each AccumNode.

AccumMaps have two important functions: *ReturnEventListPointer(AccumType, char*)* and *FilterEvent(Event*)*. ReturnEventListPointer() returns either an EventList

pointer from an AccumNode that matches the AccumType and string passed to it, or NULL if no EventList is found. FilterEvent() takes in an Event and returns a zero if the Event is to be ignored, or a one if the Event was placed in an EventList.

4. *Detector()*

Before a Detector starts monitoring events, it reads a configuration file that establishes a filtering scheme for all future events. The configuration file can be re–read periodically to reconfigure the system. After the filter is established, the *monitor()* method is invoked and the Detector can start reading in event records. There is some pre–processing involved in reading in records, currently handled by a perl script.

monitor() reads in records a line at a time and creates Event objects. Each Event is passed through a filter and the EventList assigned to a particular filter is located to deposit the Event in. There is a catch–all list labeled misc for Events that are not caught by any AccumMap filters.

When AccumMaps are searching for an EventList to deposit an Event in, they check to see if any alert parameters have been satisfied or exceeded. If so, they return the appropriate value to the Detector, and the Detector takes some sort of action. The actions include mailing the system administrator the contents of an EventList and the rule that provoked the response, killing a recently detected process, and launching `cfengine` with a new configuration.

APPENDIX B

GLOSSARY

artificial intelligence: abbreviated to AI. A "branch of computer science concerned with making computers behave like humans. The term was coined in 1956 by John McCarthy at the Massachusetts Institute of Technology." (ZDNet)

C: "The name of a programming language designed by Dennis Ritchie during the early 1970s and immediately used to reimplement Unix; so called because many features derived from an earlier compiler named 'B' in commemoration of its parent, BCPL. (BCPL was in turn descended from an earlier Algol–derived language, CPL.) Before Bjarne Stroustrup settled the question by designing C++, there was a humorous debate over whether C's successor should be named 'D' or 'P'." (Raymond)

cracker: "One who breaks security on a system. Coined ca. 1985 by hackers in defense against journalistic misuse of hacker (q.v., sense 8)...Thus, there is far less overlap between hackerdom and crackerdom than the mundane reader misled by sensationalistic journalism might expect. Crackers tend to gather in small, tight–knit, very secretive groups that have little overlap with the huge, open poly–culture this lexicon describes; though crackers often like to describe themselves as hackers, most true hackers consider them a separate and lower form of life. Ethical considerations aside, hackers figure that anyone who can't imagine a more interesting way to play with their computers than breaking into someone else's has to be pretty losing. Some other reasons crackers are looked down on are discussed in the entries on cracking and phreaking. See also samurai, dark–side hacker, and hacker ethic. For a portrait of the typical teenage cracker, see warez d00dz." (Raymond)

crash: "A sudden, usually drastic failure. Most often said of the system (q.v., sense 1)." (Raymond)

darkwing: a SparcServer–1000 currently running the Solaris 2.6 operating system. Historically, use of darkwing was restricted to faculty and graduate students at the University of Oregon.

email bomb: a series of email messages delivered with the intent of overloading a computer mail server, or over–filling a person's email folder by using up available disk space.

exploit: a name a program that can use a security vulnerability in another program to gain unauthorized access to computer resources; as in *root–exploit.*

ftp: "Abbreviation of File Transfer Protocol, the protocol used on the Internet for sending files." (ZDNet)

gladstone: a machine similar to darkwing, usually running the same operating system. Most users are undergraduates at the University of Oregon.

hacker: "A person who enjoys exploring the details of programmable systems and how to stretch their capabilities, as opposed to most users, who prefer to learn only the minimum necessary." (Raymond)

IRC: "Short for Internet Relay Chat, a chat system developed by Jarkko Oikarinen in Finland in the late 1980s. IRC has become very popular as more people get connected to the Internet because it enables people connected anywhere on the Internet to join in

live discussions. Unlike older chat systems, IRC is not limited to just two participants." (ZDNet)

kill: A command that can be used to terminate a running process.

Linux: "The free Unix workalike created by Linus Torvalds and friends starting about 1990 (the pronunciation /lee'nuhks/ is preferred because the name 'Linus' has an /ee/ sound in Swedish)." (Raymond)

network: A group of computers connected, possibly through some physical medium such as coaxial cable or optical fiber. (Davie 10)

network sniffer: see network traffic monitor.

network traffic monitor: A program that looks at traffic on the network it is connected to.

oregon: one of a cluster of DEC computers at the University of Oregon running the VMS operating system.

packet: A block of data that corresponds to some piece of application data such as a file, a piece of email or an image. (Davie 12)

Perl: "[Practical Extraction and Report Language, a.k.a. Pathologically Eclectic Rubbish Lister] An interpreted language developed by Larry Wall (<lwall@jpl.nasa.gov>, author of patch(1) and rn(1)) and distributed over Usenet. Superficially resembles awk, but is much hairier, including many facilities reminiscent of sed(1) and shells and a comprehensive Unix system–call interface. Unix sysadmins, who are almost always incorrigible hackers, increasingly consider it one of the languages of choice. Perl has been described, in a parody of a famous remark about lex(1), as the "Swiss–Army chainsaw" of Unix programming. See also Camel Book." (Raymond)

polling: the act of repeatedly requesting information from a program or device.

root: " The superuser account (with user name 'root') that ignores permission bits, user number 0 on a Unix system." (Raymond) As the term superuser suggests, root has the ultimate power, and is capable of modifying any file on a system.

root–exploit: a program that enables the user of the program to gain root–access privileges to a computer system.

Solaris: "An Unix –based operating environment developed by Sun Microsystems. Originally developed to run on Sun's SPARC workstations, Solaris now runs on many workstations from other vendors. Solaris includes the SunOS operating system and a windowing system (either OpenWindows or CDE). Solaris currently supports multithreading, symmetric multiprocessing (SMP), integrated TCP/IP networking, and centralized network administration. A Wabi emulator is available to run Windows applications." (ZDNet) Both darkwing and gladstone run Solaris 2.6 currently.

switch: a network device that copies frames of information from one computer network to another.

sysadmin: A contraction of system administrator. "[V]ery commonly used in speech or on–line to refer to the systems person in charge on a computer." (Raymond)

telnet: "To communicate with another Internet host using the TELNET (RFC 854) protocol (usually using a program of the same name.)" (Raymond) This is a very common way of accessing gladstone or darkwing. You might start a telnet window using a program like NCSA Telnet, or SimpTerm and once logged in, execute the

command pine to read your email.

traffic: I use this term a couple times to refer to the packets a network traffic monitor can see on a network.

Trojan horse: an application that replaces a legitimate program, but whose purpose is to either collect sensitive information or behave in a manner not consistent with the program it replaces.

user: "Someone doing 'real work' with the computer, using it as a means rather than an end." (Raymond). Also, someone with a legitimate account on a computer. For example, a University of Oregon student is a user of *gladstone*.

APPENDIX C

FIGURES

From: Steve VanDevender <stevev@darkwing.uoregon.edu>
Sender: owner-uosecurity@lists.uoregon.edu
To: uosecurity@lists.uoregon.edu
Subject: uosecurity: a retrospective
Date: Thu, 3 Jun 1999 18:24:18 -0700 (PDT)

Someone asked if I knew of any statistics on scans, probes, and breakins to campus computer systems. I don't know if anyone else has been collecting such information, but I just finished a scan through over two years of archived uosecurity postings to come up with this summary based on posted reports.

Here a scan/probe is an attempt from a particular site to look for vulnerable systems at the U of O; if multiple U of O systems were scanned it is still counted as one incident if it originated from the same place at about the same time. A compromise is any undesired or unauthorized system access, such as root compromise, use of accounts, successful removal/replacement of files, or denial of service.

Note that these are just incidents detected and reported by uosecurity subscribers. Undoubtedly there have been many more incidents than those on campus.

Month	# scans/probes	# system compromises
1997/03		2
1997/08	1	
1997/09	1	1
1998/01		1
1998/03	1	
1998/04	2	
1998/05	2	
1998/06	1	1
1998/07	1	
1998/08	4	1
1998/09	7	3
1998/10	9	4
1998/11	3	
1998/12	12	
1999/01	10	2
1999/02	4	1
1999/03	8	1
1999/04	12	
1999/05	18	2

Note the upward trend. We still have our work cut out for us.

Figure 1. Reported security incidents at the University of Oregon, 1997–99.

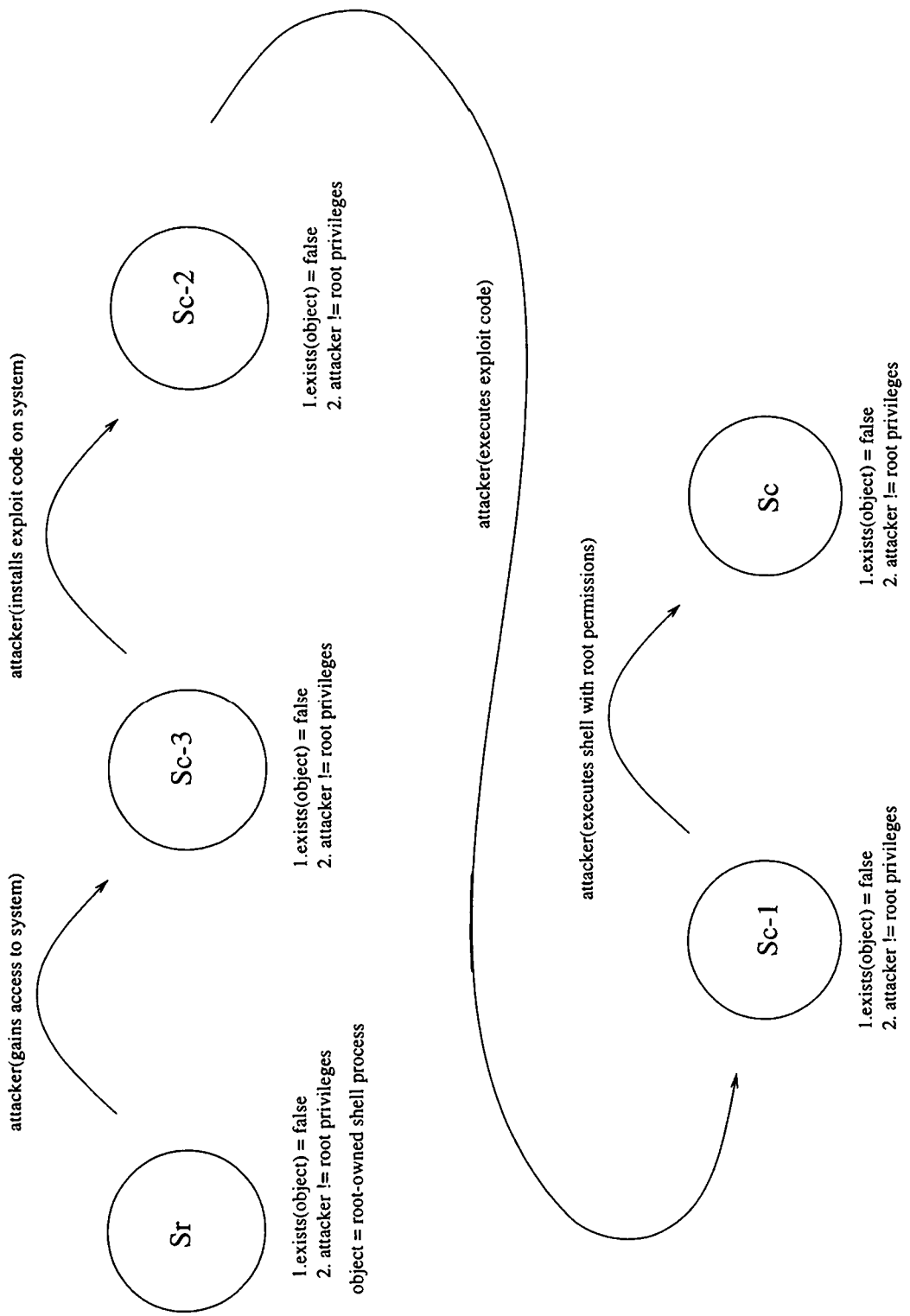

Figure 2. State transition diagram of a root–exploit.

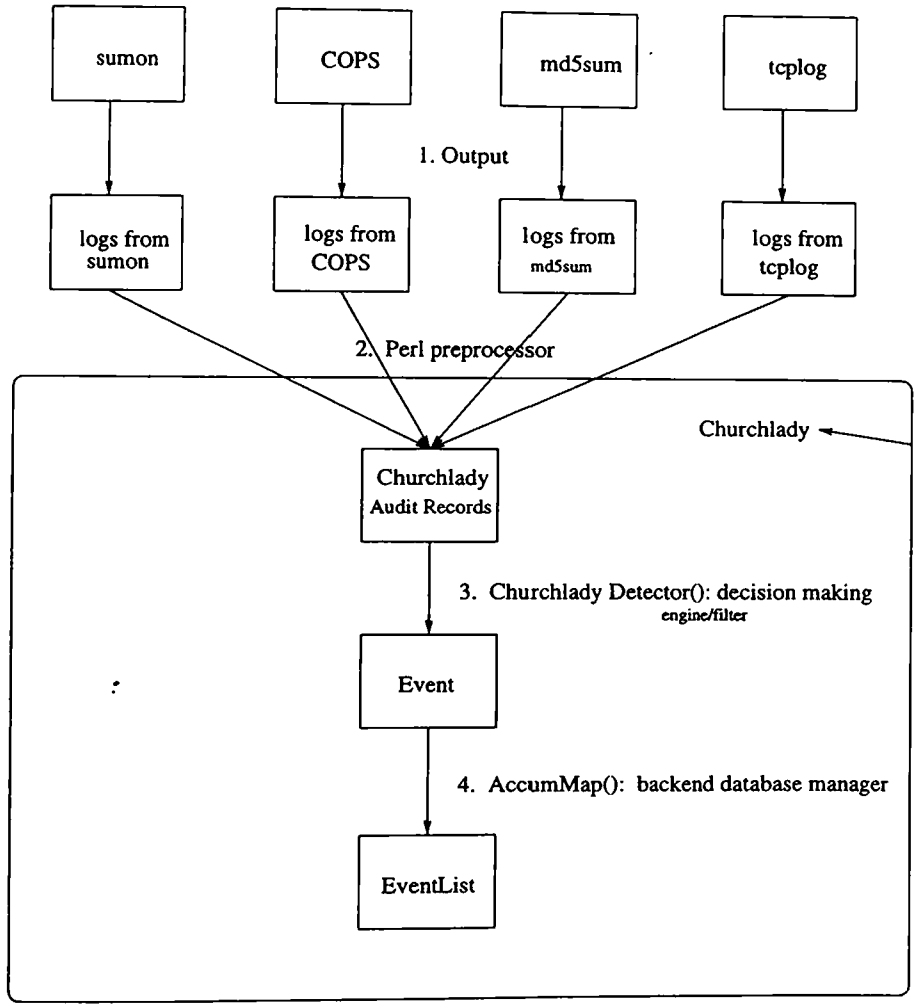

Figure 3. Data Flow in Churchlady.

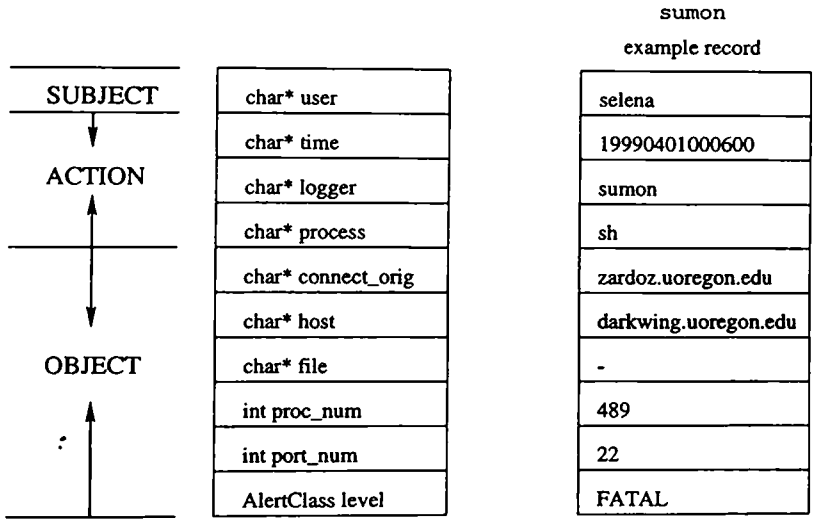

Figure 4. Event() audit record structure for Churchlady.

```
Received: from darkwing.uoregon.edu (root@darkwing.uoregon.edu
   [128.223.142.13])
   by hexadecimal.uoregon.edu (8.9.3/8.9.3) with ESMTP id KAA25179
   for <stevev@hexadecimal.uoregon.edu>; Thu, 18 Feb 1999 10:22:22
   -0800 (PST)
Received: (from root@localhost)
   by darkwing.uoregon.edu (8.8.8/8.8.8) id KAA19096;
   Thu, 18 Feb 1999 10:22:20 -0800 (PST)
Message-Id: <199902181822.KAA19096@darkwing.uoregon.edu>
From: System Operator <operator@darkwing.uoregon.edu>
To: bj@oregon.uoregon.edu, jneher@tenorman.uoregon.edu,
       stevev@hexadecimal.uoregon.edu
Subject: ** SUID ALERT ** sbrewing => root on darkwing (TTY pts/87, EXEC
   /sbin/jsh)
Date: Thu, 18 Feb 1999 10:22:20 -0800 (PST)

======== Current sessions for user sbrewing (Thu Feb 18 10:22:18========
sbrewing   pts/87       Feb 18 10:21   (localhost)

================ Recent Activity on pts/87 ================
 ========  Session activity -- Thu Feb 18 10:21:56  ========
     USER   PID  PPID   SID NI    STIME TT             TIME COMMAND
*    root 18770 18539 18539 20 10:21:33 pts/87         0:00 sh
 sbrewing 18539 18537 18539 20 10:20:56 pts/87         0:00 -bash
  Process 18770 is running executable /sbin/jsh as root via command sh.
* Indicates the first current process deemed noteworthy for this
   session.

 ========  Session activity -- Thu Feb 18 10:22:18  ========
     USER   PID  PPID   SID NI    STIME TT             TIME COMMAND
*    root 18770 18539 18539 20 10:21:33 pts/87         0:00 sh
 sbrewing 18539 18537 18539 20 10:20:56 pts/87         0:00 -bash
  Process 18770 is running executable /sbin/jsh as root via command sh.
* Indicates the first current process deemed noteworthy for this
   session.

The following processes are being watched:
   18770 (/spare/lost+found/#0475969)
   18539 (/spare/lost+found/#0475970)
```

Figure 5. Sample output from sumon.

```
Content-Length: 1816
Return-Path: <root>
Received: (from root@localhost)
   by darkwing.uoregon.edu (8.8.8/8.8.8) id BAA03026
   for systems@darkwing.uoregon.edu; Fri, 19 Feb 1999 01:28:21 -0800
   (PST)
Message-Id: <199902190928.BAA03026@darkwing.uoregon.edu>
Content-Type: text
From: 0000-Admin(0000) <root>
Subject: Daily COPS report for darkwing
Date: Fri, 19 Feb 1999 01:28:21 -0800 (PST)


ATTENTION:
Security Report for Fri Feb 19 01:28:18 PST 1999
from host darkwing

**** root.chk ****
**** dev.chk ****
**** is_able.chk ****
Warning!  /etc/security is _World_ readable!
Warning!  /usr/adm/spellhist is _World_ writable!
**** rc.chk ****
Warning!  File /var/spool/calendar/callog.root (in /etc/rc2.d/S90mon_cm)
   is _World_ writable!
**** cron.chk ****
Warning!  File /home21/ode/public_html/excite/aindex.pl (inside root
   executed file ) is _World_ writable!
Warning!  /home21/ode/public_html/excite/aindex.pl (in
   /usr/spool/cron/crontabs/ode) is World writable!
**** group.chk ****
Warning!  Duplicate Group(s) found in /etc/group:
adm daemon
**** home.chk ****
Warning!  User bin's home directory /usr/bin is mode 0775!
Warning!  User adm's home directory /var/adm is mode 0775!
Warning!  User lp's home directory /usr/spool/lp is mode 0775!
**** passwd.chk ****
Warning!  Password file, line 7, user smtp has uid = 0 and is not root
   smtp:x:0:0:mail daemon user:/:
Warning!  Password file, line 18, uid > 8 chars
   committees:x:1307:176:UO Committee
   Information:/home9/committees:/bin/csh
Warning!  Password file, line 19, uid > 8 chars
   carljohann:x:21013:96:Carl Johannesen:/home1/carljohann:/bin/csh
Warning!  Password file, line 20, uid > 8 chars
   registrar:x:7286:186:Registrar's Office Web
   pages:/home7/registrar:/bin/csh
**** user.chk ****
Warning!  User masloot    /home9/masloot/public_html      is mode 0777!
**** misc.chk ****
**** ftp.chk ****
ftp-Warning! Incorrect permissions on "passwd" in /disk1/ftp/etc!
ftp-Warning! Incorrect permissions on "group" in /disk1/ftp/etc!
**** kuang ****
**** bug.chk ****
Warning!  /usr/lib/sendmail could have a hole/bug!  (CA-88:01)
Warning!  /usr/lib/sendmail could have a hole/bug!  (CA-90:01)
Warning!  /bin/mail could have a hole/bug!  (CA-91:01a)
```

Figure 6. Sample output from COPS.

```
Apr  7 01:23:23 zardoz tcplog: www connection attempt from
green.alexa.com
Apr  7 01:27:03 zardoz tcplog: smtp connection attempt from
zephyr.isi.edu
Apr  7 01:33:59 zardoz tcplog: smtp connection attempt from
freshmeat.portal.redhat.com
Apr  7 02:19:05 zardoz tcplog: smtp connection attempt from
darkwing.uoregon.edu
Apr  7 02:19:16 zardoz tcplog: smtp connection attempt from
darkwing.uoregon.edu
Apr  7 02:27:14 zardoz tcplog: www connection attempt from
samuel.soongsil.ac.kr
Apr  7 04:39:42 zardoz tcplog: www connection attempt from
surprise.mchh.siemens.de
Apr  7 05:01:09 zardoz tcplog: auth connection attempt from
darkwing.uoregon.edu
Apr  7 05:13:51 zardoz tcplog: smtp connection attempt from
darkwing.uoregon.edu
Apr  7 05:24:44 zardoz tcplog: smtp connection attempt from
darkwing.uoregon.edu
Apr  7 05:25:53 zardoz tcplog: smtp connection attempt from
zephyr.isi.edu
Apr  7 05:25:57 zardoz tcplog: smtp connection attempt from
zephyr.isi.edu
Apr  7 05:28:13 zardoz tcplog: smtp connection attempt from
darkwing.uoregon.edu
Apr  7 05:46:48 zardoz tcplog: smtp connection attempt from
darkwing.uoregon.edu
Apr  7 05:47:02 zardoz tcplog: smtp connection attempt from
darkwing.uoregon.edu
Apr  7 06:11:33 zardoz tcplog: smtp connection attempt from
tower.nfr.net
Apr  7 06:27:08 zardoz tcplog: www connection attempt from 194.68.97.39
Apr  7 06:27:15 zardoz tcplog: smtp connection attempt from
darkwing.uoregon.edu
Apr  7 06:37:19 zardoz tcplog: smtp connection attempt from
darkwing.uoregon.edu
Apr  7 06:44:39 zardoz tcplog: smtp connection attempt from
present.cirl.uoregon.edu
Apr  7 06:54:41 zardoz tcplog: smtp connection attempt from
darkwing.uoregon.edu
Apr  7 07:00:17 zardoz tcplog: smtp connection attempt from
darkwing.uoregon.edu
Apr  7 07:09:22 zardoz tcplog: smtp connection attempt from
darkwing.uoregon.edu
Apr  7 07:11:07 zardoz tcplog: smtp connection attempt from
lacrosse.redhat.com
Apr  7 07:14:21 zardoz tcplog: smtp connection attempt from
tower.nfr.net
Apr  7 07:17:00 zardoz tcplog: smtp connection attempt from
tower.nfr.net
```

Figure 7. Sample output from `tcplog`.

logger	*Audit record*	*AlertClass*	*action*
`tcplog`	known and/or offered service	INFO	
`tcplog`	unknown or not–offered service	WARN	
`sumon`	suid binary on authorization list	INFO	`md5sum` check
`sumon`	suid binary not on authorization list	FATAL	disable
`sumon`	root owned shell on authorization list	WARN	
`sumon`	root owned shell not on authorization list	FATAL	kill process
`md5sum`	binary check ok	INFO	
`md5sum`	binary not ok	FATAL	disable program
`md5sum`	binary checksum not found	FATAL	diable program
COPS	world readable	WARN	
COPS	group writable	WARN	
COPS	incorrect permissions	WARN	
COPS	world writable	ALERT	chmod file or directory
COPS	bad .shost or .rhost contents	ALERT	disable file
COPS	bad passwd file entries	FATAL	remove bad entries
COPS	new device detected	FATAL	

Figure 8. Signature event list for `tcplog`, `sumon`, `md5sum` and COPS.

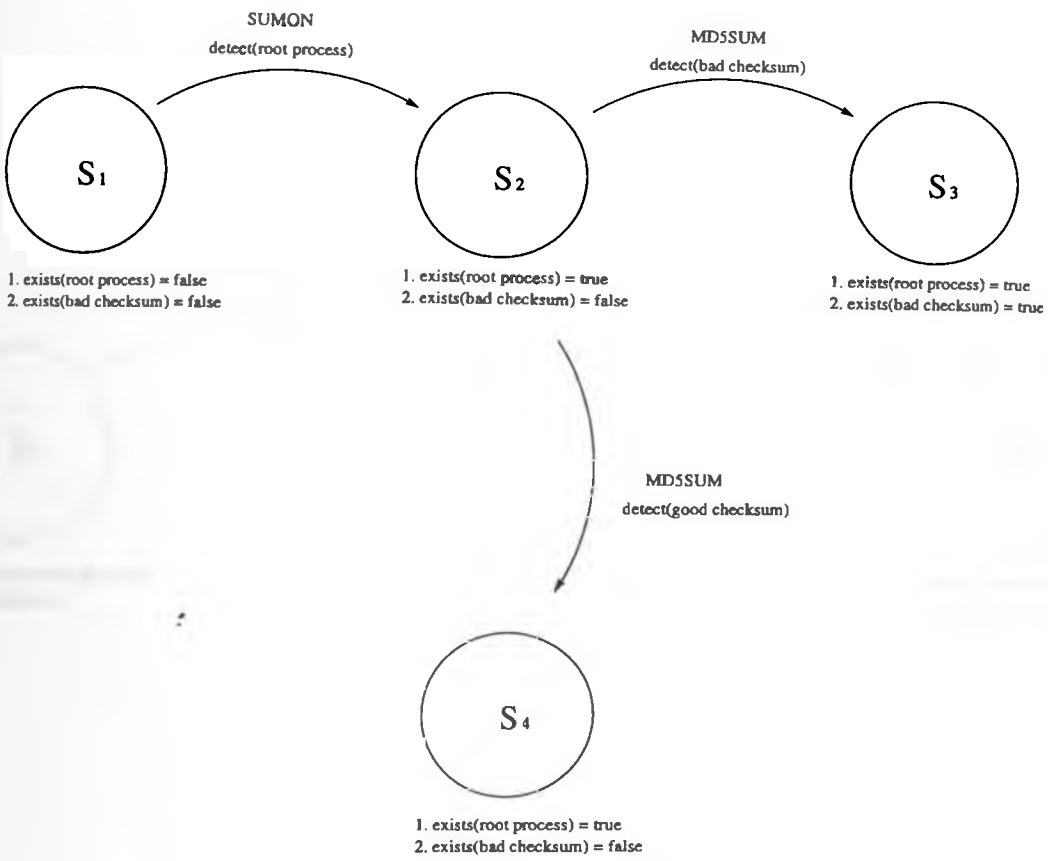

Figure 9. `sumon` and `md5sum` state transition diagram.

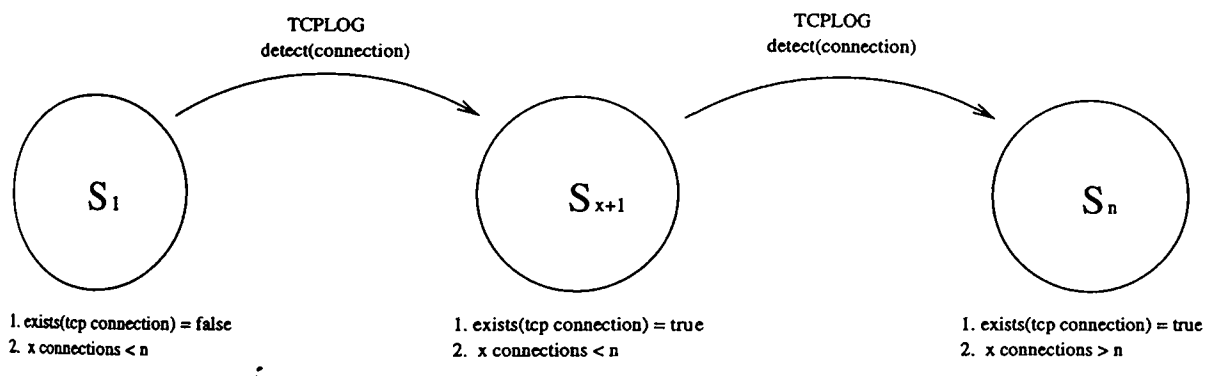

Figure 10. `tcplog` connection state transition diagram