Fighting Institutional Memory Loss: The Trackle Integrated Issue and Solution Tracking System

Daniel S. Crosta, Matthew J. Singleton – Swarthmore College Computer Society
Benjamin A. Kuperman – Swarthmore College

ABSTRACT

For part-time sysadmins, a record of past actions is an invaluable tool that provides guidance in repairing or extending system services. However, requiring sysadmins to keep a detailed log of changes made to a live system can often seem like a low priority task when compared to addressing long and growing to-do lists. This problem is worse if the system administrator is a part-time volunteer and an overworked student. In this paper we present Trackle, an integrated trouble ticket and solution tracking system which takes the legwork out of creating and maintaining this sort of institutional memory. Furthermore, Trackle is designed to allow untrained student sysadmins to bootstrap their knowledge by peeking over the shoulders of their more experienced colleagues – even if those colleagues graduated years earlier. We accomplish this by tracking the exact actions taken by sysadmins, showing what lines were changed and in which configuration files. We allow experienced and inexperienced sysadmins alike to freely annotate and cross-reference these shell session logs through an integrated Wiki web interface.

Introduction

The Swarthmore College Computer Society (SCCS) is a group of student volunteers who provide services to more than 2,000 current students, alumni, faculty, and staff of Swarthmore College. We provide UNIX shell accounts, email, group mailing lists, web space for individuals and student groups, Wikis, database access, and general computer expertise. We also maintain a public lab of eight Debian GNU/Linux and two Apple Mac OS X computers. The sysadmins meet weekly for an hour to keep up to date on projects, problems, and planned improvements, and communicate via email between meetings.

Unlike many large UNIX administration setups, the SCCS has no full-time trained staff. The student sysadmins are hired once a year, usually in groups of two or three, and most graduate after serving as a sysadmin for two years. At any time we have about eight sysadmins on staff. Since a part of SCCS's mission is to provide an environment in which interested students may learn the art of system administration, we recruit students with technical backgrounds ranging from "I can email and browse the web," to "my desktop PC is a Beowulf cluster." This disparity in background and the high turnover rate presents us with the dual problems of how to preserve our collective knowledge and how to train future generations.

For many years, our primary means of training sysadmins has been the SCCS staff email list and archived diaries. The diaries are a cluttered repository; we use the staff email list as often to tease one another as to discuss pertinent policy, administrative issues, or help requests from our users. Though sysadmins are supposed to report changes to our servers and lab machines to the list, we often have to rediscover particular configurations when problems arise. Furthermore, `mbox` mailboxes are a hard format to browse, particularly for new, inexperienced sysadmins who are eager to learn but more comfortable browsing the web than navigating the sometimes murky waters of UNIX shell prompts, commands, and filesystems.

In the last three years, SCCS staff have used a private Wiki to document changes, configurations, and internal policies as well as for collaborative planning. We have found the unconstrained and highly interlinked environment very valuable to our operations, but relatively little of the content on the Wiki is pertinent system configurations or records of changes made. As with the email diaries, the extra time and effort required to self-report changes made to our systems presents too high a hurdle for our volunteer sysadmins, each of whom has a range of other academic and extra-curricular commitments.

For training, we needed a system that preserved the ability to associate high level conceptual descriptions of desired configurations or problems with the particular shell actions or configuration file changes to put those ideas into effect. Unlike the Wiki and email list, we wanted a system that did not require sysadmins to take on the extra burden of having to write up complicated records of their actions after long, sometimes frustrating problem-solving sessions. Furthermore, since our sysadmins are hired without any requirement of previous experience, we wanted a tool which could help new sysadmins learn the subtleties of UNIX administration and our particular setup without requiring any former familiarity with UNIX.

To respond to these concerns, we created Trackle, a web- and console-based integrated issue and solution tracking system. Each problem reported to the system, either by our users or by SCCS staff, creates a ticket in the issue database. Open issues may be viewed through the web where they can be freely annotated, edited, and cross-referenced with the built-in Wiki; or through a console interface, from which a tracked shell session may be started. All actions taken in the shell session, and all files changed, are recorded in the Trackle database, and create an entry in the ticket history log. These shell session logs are then editable in Wiki fashion.

Existing and Related Work

We first attempted to discover if any existing tools could be used to solve this problem. Since a sysadmin's job is primarily task-oriented, we began our search with trouble ticket and issue tracking systems. Issue tracking systems seem to have lost appeal in the literature since the late 1990s. Before that, there were one or two issue tracking papers per year in LISA, most of which explored different extensions to the basic trouble ticket idea.

One early tool, Request 2 [8], was designed with a pedagogy of training junior system administrators in mind. With Request 2, senior sysadmins could divide and delegate work to less experienced admins, and oversee their work or offer advice.

PLOD, the Personal LOgging Device [6], enables sysadmins to easily self-report by providing a set of command-line Perl scripts which the sysadmin may use to make notes to him or herself about work just completed. PLOD simplifies the tasks of keeping records consistently and making them centrally available. PLOD did not include, and was not integrated with, any issue tracking system.

The LOS Task Request System [9], designed several years ago by another SCCS sysadmin, allows users to create ticket-like task requests through a web interface. Each task request is associated with a task description, which includes validation logic and the commands required to execute the task. This alleviates sysadmin workload by automating the tedious tasks of collecting and verifying information for repetitive changes to the system.

A related field of more current interest is configuration management and version control. Configuration management systems attempt to centralize or abstract configuration details for large numbers of similar hosts. Tasks include initial setup, maintenance, troubleshooting, and incorporating local changes back into the global description. For a more careful consideration of the history and future of configuration management, see [1, 3, 7].

The SCCS currently has no plans to use a configuration management system at our site. Our two primary servers do not share enough common configuration to yield much advantage from such a system, while our public lab machines, which are configured similarly, are periodically reinstalled to a known consistent state for security reasons. For these reasons, we do not believe we would gain enough benefit from a configuration management system to justify the setup effort.

Much of the time our sysadmins spend working involves user-owned files, such as spam filter settings, dotfiles, and web access configurations. Since these files may be created and deleted by users at any time, a configuration management system would not be a good choice for monitoring changes to these files.

Moreover, though configuration management may be considered an industry best practice, we believe no automatic configuration system can adequately replace a sysadmin's ability to get dirty with manual configuration and hand-tuning of UNIX services and applications. We would rather our sysadmins learn how to discover a problem at its source and implement a solution appropriately than merely learn how to control a single piece of software, no matter how powerful it may be.

One of the chief goals of Trackle is to provide a platform for training inexperienced sysadmins. We believe that it is important to gain an understanding of UNIX cause and effect by observing and learning about configuration files, logs, commands, and scripts. If new sysadmins only learn to rely on abstract configuration management systems, they might find themselves unprepared to deal with problems outside the scope of those systems.

None of the systems we considered deploying covered our needs for a system to track issues, actions, and train new sysadmins. A ticket tracking system would help organize and abstract the institutional memory knowledgebase, but would still require self-reporting. A configuration management system would keep accurate records for the files that it tracked, but would not provide a good platform for training incoming sysadmins. Having searched and failed to find just what we wanted gave us a chance to reevaluate our problems, and we decided that only by implementing a custom solution could we meet our requirements.

Design of Trackle

From the beginning, the design of Trackle has been motivated by a philosophy of minimalism and simplicity. We believe that the most useful system for our needs is one that imposes as little as possible on our sysadmins in terms of new workflows and interfaces to learn. We want Trackle to stay out of the way of sysadmins, but at the same time to automate as much of its own data collection and presentation tasks as possible.

It is always difficult to find the right balance of the automatic and the manual, the consistent and the customizable. Given the sort of information Trackle manages, we believe the right place for the cut is at data collection and initial presentation. Creating the complex knowledge structures through annotation and cross referencing is left to the sysadmins as a secondary task, since it is not one that a computer is likely to do well.

Trackle is intended for smaller groups of sysadmins with a relatively low volume of requests. It is designed to be used effectively by both expert and inexperienced sysadmins. Trackle's information organization methods are best suited to small groups who have time to annotate and cross-reference collected data. The goal of allowing this rehashing and reorganization of information is to provide a platform for self-directed training of inexperienced sysadmins on their own time.

Requirements

Two classes of users: Since the SCCS naturally has two classes of users, sysadmins and SCCS end users, Trackle was designed to accommodate both types. Differentiating the web interface for the two types of users allows us to improve both security and convenience. Unauthenticated end users are denied sensitive information, such as email addresses in tickets or files in logged shell sessions, and are not shown potentially confusing prompts when creating tickets. Sysadmins, on the other hand, need access to all the information about tickets and shell logs, and should know enough to understand the more detailed ticket attributes.

Low barriers to use: One problem with existing ticket tracking systems is the often overwhelming amount of information that is requested to submit a ticket. With Trackle, we wanted to make filing a ticket as easy as possible so that end users and sysadmins alike can move quickly through the web forms. Our interfaces are designed to be intuitive so that they can be used without having to waste time reading documentation.

For end users, we wanted Trackle to be as easy to use the first time as the tenth. End users do not have to register accounts with Trackle to create or subscribe to tickets. Instead we use an email confirmation system to verify an end user's identity.

High-level organizational tools: Trackle's central goal is to provide a framework for flexible representation of the collected data. It is designed to facilitate Wiki-like annotation and cross-referencing rather than imposing a fixed organizational structure. Objects in Trackle support Wiki formatting, enabling them to link to each other. With these tools, the recorded data can be arranged, indexed, and presented in the ways most useful to the sysadmins who need access to it.

Few dependencies: Trackle was designed to be dependent on as few external software packages as possible. It is specifically designed to work with the packages and versions available in the Debian Sarge GNU/Linux distribution (our deployment platform); however, we wanted to make Trackle freely available for any interested groups for use on many platforms. To ensure easy portability, it is not closely tied to any particular versions or packages.

Ticket System

As discussed earlier, we decided to use a ticket tracking system as the basis for Trackle's higher-level categorization of captured data. We briefly considered RT,[1] but decided that it would be too confusing and cluttered with information to be useful to inexperienced sysadmins. Instead, we decided to use a relative newcomer to the trouble ticket world, Trac,[2] as our foundation.

Trac is a BSD-licensed bug tracking system with integrated version control and Wiki. Trac embodies a minimalistic approach to ticketing which fit well with our design goals: it presents a clean, interface intuitive to both our end users and sysadmins, and has relatively few features and ticket attributes specific to bug tracking. Many of the features we wanted (easy annotation of tickets and shell histories, high-level organization and cross-referencing through the integrated Wiki, file change visualization) were already implemented, and adding our own functionality was very easy due to the well-organized Trac API.

However, Trac required more than cosmetic changes to fit our needs. Despite its flexible permission system, Trac does not natively support multiple classes of users, nor does it have ticket locking. Additionally, we had to modify the ticket status logic to accommodate unconfirmed tickets.

Tracking File Changes

Tracking file changes was the biggest challenge encountered when designing Trackle. Because we do not use any configuration management or version control, we needed to ensure we could get a before and after view of each file modified during a sysadmin's shell session. We considered several alternative strategies before eventually settling on a custom interposition library. The possibilities may be roughly split into kernel-space solutions (LVM snapshots, UnionFS, FUSE, C2 audit logs) and user-space solutions (plugins/history interpretation and library interposition).

LVM snapshots: The Linux Logical Volume Manager[3] (LVM) provides a way to create a time-frozen snapshot of a volume. In Trackle, we could use a LVM Snapshot to capture the pre-shell tracking session state of the system and then compare the two filesystems at the end of the session. LVM works below the filesystem and captures changes by tracking changed disk blocks rather than changed files. To use LVM with Trackle we would have to relate changed

[1] http://www.bestpractical.com/rt/

[2] http://trac.edgewall.com/

[3] http://sourceware.org/lvm2/

disk blocks to changed files, or compute a recursive diff between the two hierarchies; the former would be prohibitively hard and would be tied to particular filesystem implementations, and the latter would be prohibitively slow. It is also unclear how to differentiate what files were changed by the user during the shell session and which files were merely changed by other users or processes during that time. Additionally, it would impose LVM as a runtime dependency for Trackle and limit Trackle to use on Linux.

UnionFS: UnionFS[4] is a meta-filesystem that stacks one or more existing, mounted filesystems into one hierarchy. It supports copy-on-write so that a read-only filesystem can be made to appear as a read-write filesystem. This functionality is used by many Live Linux Distributions, like SLAX[5] and KNOPPIX.[6] When operating on a file in a UnionFS stack, the top-most instance of the named file is used. This means that in order to keep a initial copy of the file for Trackle's use, the underlying filesystem would have to be mounted read-only. Since it would be impractical to remount the entire filesystem hierarchy read-only, Trackle would have to create a read-only bind mount of the root hierarchy, to use as the bottom of the UnionFS stack. On top of that would be placed a read-write filesystem mounted elsewhere to capture the revised versions of files. Because this third filesystem is hierarchically removed from the active root filesystem, any changes made during the shell tracking session would not be reflected to the root filesystem unless manually copied (for instance, by a special Trackle command) or when synchronized at the end of the session. Additionally, UnionFS requires the insertion of a new kernel module, which some sysadmins may be hesitant to allow. At this time, UnionFS is only available for Linux.

FUSE: The Filesystem in Userspace[7] (FUSE) technique works similarly to the interposition library approach we finally adopted. By use of a special kernel module, some or all filesystem-related library calls can be delegated to a user space daemon. Trackle could use FUSE to make note of which files the user is accessing and copy the files' initial state before returning from the user's library call. This provides about the same level of flexibility as using an interposition library. Using FUSE requires loading a kernel module and mounting and unmounting filesystems. In order to use FUSE with Trackle, we would have to create additional setuid-root binaries to handle these tasks. FUSE is currently only available for Linux and FreeBSD.

C2-like audit logs: Many operating systems include auditing facilities to track file accesses as one of the requirements for a trusted computer system [10, 2]. There are projects to bring this type of auditing to GNU/Linux such as SNARE for Linux.[8] and SELinux[9] These systems require kernel modifications, and are typically enabled for the entire system instead of just a process and its subprocesses. Consequently, it is difficult to dynamically enable auditing, which results in the generation of overwhelming amounts data. Additionally, these audit logs only notify us that a file has been modified after the fact, limiting our ability to determine what changes were made.

Plugins and shell history interpretation: In theory it is possible to capture the majority of file changes made by a sysadmin by writing plugins for the editors Vim and Emacs to record files opened or saved, and by looking at the command history of the shell to infer which files might have been modified. This approach can never be more than approximate as there are many other ways to change a file other than by these two editors. Even writing plugins for these editors will not capture changes made to underlying files by wrapper programs like `vipw` that act on temporary files. Additionally, editor plugins do not capture file changes made by users at the command line either by shell redirection ("`echo stuff >> outfile`") or file-related commands (`mv`, `cp`, `rm`, `chmod`, etc.).

While parsing a shell history file might be able to discover these kinds of changes, it would have to happen after the commands changing the files had run. This would prevent us from tracking file changes, and the list of command patterns to check would be quite large. Wrapping all file change tools and implementing a custom shell would impose a large (if not insurmountable) maintenance task.

Library interposition: Most binary executables on UNIX are dynamically linked and access files through standard C library functions (`open`, `unlink`, `chmod`, etc.) The dynamic linker/loader checks the environment variable `LD_PRELOAD` for a list of shared libraries to be loaded and searched before all others. These libraries are called *interposed libraries* because any function that is defined in one of these libraries intercepts the call to standard libraries.

We use a custom interposition library to intercept file-related library calls so that we can track changes to files during a shell session. When we intercept the call, we are able to gather information about the current state of the file before passing the request on to the real library call. We are also able to observe the resulting changes and collect additional information as needed.

We decided to use an interposition library for a number of reasons. It can be written so that it only captures the events we are concerned with, including privileged operations (using `sudo`, `su`, etc.). It is a userspace solution, so it is not dependent on any particular kernel and is portable to most other UNIX platforms. Finally and perhaps most importantly, the majority of the functionality was already present in a

[4] http://www.unionfs.org/
[5] http://slax.linux-live.org/
[6] http://knoppix.org/
[7] http://fuse.sourceforge.net/
[8] http://www.intersectalliance.com/projects/snare/
[9] http://www.nsa.gov/selinux/

component of Audlib [4, 5] designed to track user and sysadmin actions for detecting abuse of authority.

Trackle Architecture

Trackle consists of four functional components, each of which communicates with a central database:

1. The web interface, which allows full access to the ticket database, Wiki pages, and shell session logs. It is the primary interface to all Trackle data.
2. The console tools, which allow sysadmins to access the ticket database, and begin shell tracking sessions to capture changes made to resolve a problem.
3. The interposition library, which is responsible for tracking changes to files during a tracked shell session.
4. The email notification system, which keeps end users and sysadmins informed of ticket status changes, and the email confirmation system which enables unauthenticated end users to interact with aspects of the web interface.

The web interface, console tools, and email system are all written in Python and communicate directly with the central PostgreSQL[10] database. The shell tracking session is written in C, and communicates through the console tools (see Figure 1).

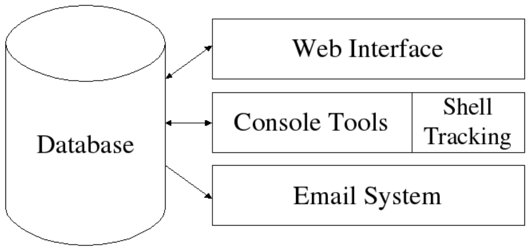


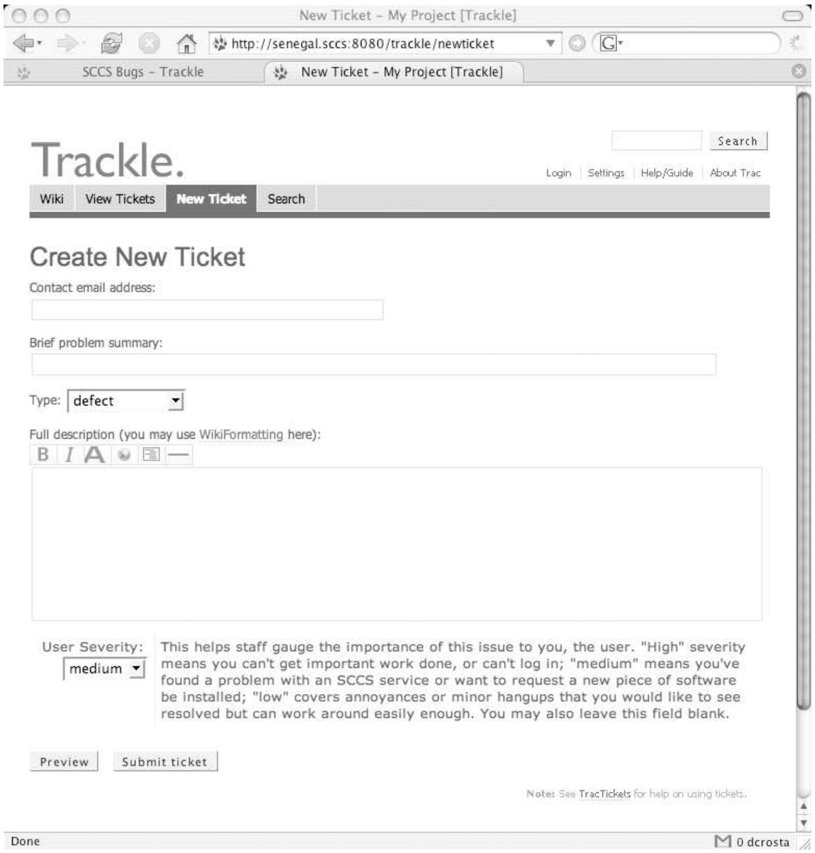
Figure 1: Trackle consists of four components which store their data in a central PostgreSQL database.

Web Interface

The Trackle web interface provides full access to tickets, shell logs, and an integrated Wiki. The web interface has been designed with simplicity and integration in mind. Like most of Trackle, the web interface is written mostly in Python. Trackle uses Clearsilver[11] for its HTML templates. The web interface recognizes two classes of users, anonymous end users and authenticated sysadmins. Because they are not required to log in, end users are required to complete email confirmation when creating tickets. Sysadmins

[10] http://www.postgresql.org/

[11] http://www.clearsilver.net/



Create New Ticket

Contact email address:

Brief problem summary:

Type: defect

Full description (you may use WikiFormatting here):

User Severity: medium

This helps staff gauge the importance of this issue to you, the user. "High" severity means you can't get important work done, or can't log in; "medium" means you've found a problem with an SCCS service or want to request a new piece of software be installed; "low" covers annoyances or minor hangups that you would like to see resolved but can work around easily enough. You may also leave this field blank.

Note: See TracTickets for help on using tickets.

Figure 2: The ticket creation form for anonymous users is streamlined so that a user can file a ticket quickly. The set of fields displayed and the explanation shown next to each is configurable.

will be able to authenticate with the system, which will allow them to perform privileged tasks.

Tickets: Trackle's integrated issue tracking system is more than just a to-do list. It provides the initial framework for organizing automatically collected data, linking an abstract description of a problem and the concrete steps required to solve it. All users may create tickets. End users are presented with a streamlined form (Figure 2) requesting:

- contact email address
- brief problem description
- problem type
- problem severity

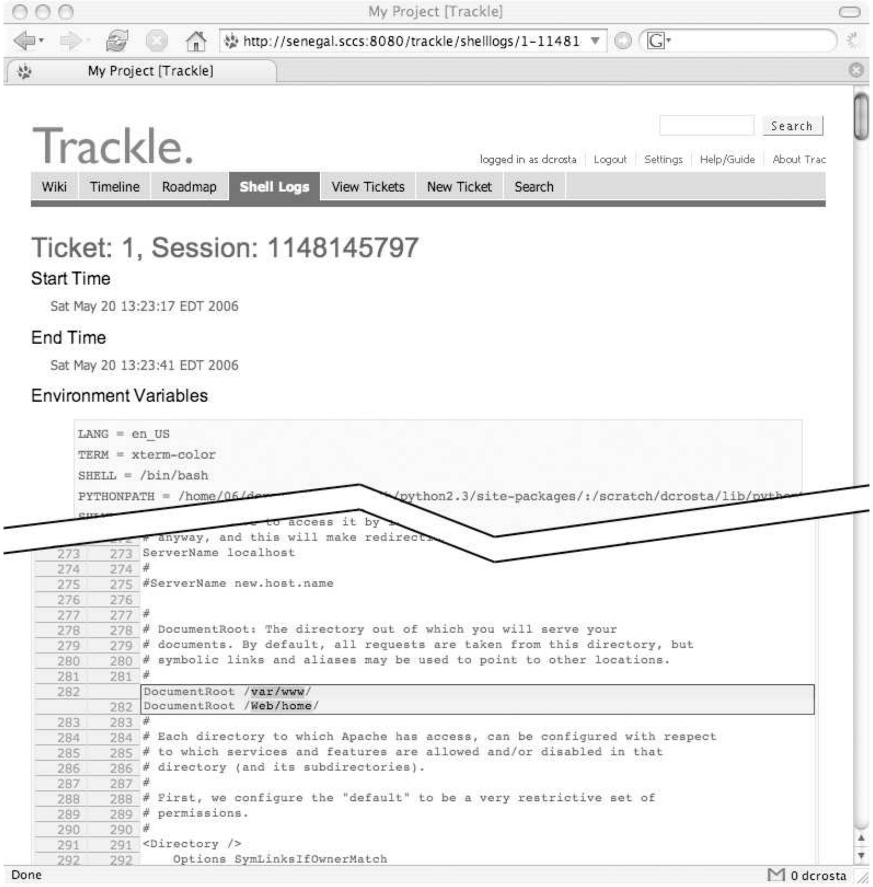
In addition to these fields, authenticated sysadmins are prompted for more information which would not be relevant to end users:

- priority
- sysadmin assignment
- keywords
- subscription list

Ticket browsing is also available to anonymous and authenticated users. Again, the two user groups have very different views. Anonymous users are presented with a minimal amount of information about the ticket, for convenience, security, and privacy. They can also subscribe to existing tickets. Authenticated users have access to the entire ticket history, have the ability to change the ticket fields, and can close, reopen, and assign tickets.

Wiki pages: Trac's integrated Wiki required very little modification for Trackle. A Wiki enables users to add, remove, and modify content easily and quickly directly through their web browser using a simple formatting language. Wiki pages are visible to both authenticated and anonymous users, but are only modifiable by authenticated users and can be made private. Most long text fields in Trackle support Wiki formatting to allow even more possibilities for cross-referencing and annotation.

Shell session logs: All the information recorded in a shell tracking session is presented in the web interface in a shell session log (Figure 3). They are accessible to authenticated sysadmins through an index where they are sorted by ticket, with the most recently added shell log appearing first. The logs are also linked individually from the associated ticket. In keeping with the ideal of providing the most functionality while imposing the least structure, Trackle shell



Ticket: 1, Session: 1148145797

Start Time

Sat May 20 13:23:17 EDT 2006

End Time

Sat May 20 13:23:41 EDT 2006

Environment Variables

```
LANG = en_US
TERM = xterm-color
SHELL = /bin/bash
PYTHONPATH = /home/06/dc               /python2.3/site-packages/:/scratch/dcrosta/lib/pyth
```

```
                     to access it by
# anyway, and this will make redirecti
ServerName localhost
#
#ServerName new.host.name

#
# DocumentRoot: The directory out of which you will serve your
# documents. By default, all requests are taken from this directory, but
# symbolic links and aliases may be used to point to other locations.
#
DocumentRoot /var/www/
DocumentRoot /Web/home/
#
# Each directory to which Apache has access, can be configured with respect
# to which services and features are allowed and/or disabled in that
# directory (and its subdirectories).
#
# First, we configure the "default" to be a very restrictive set of
# permissions.
#
<Directory />
    Options SymLinksIfOwnerMatch
```

Figure 3: The shell session log displays all the data that was recorded during the shell session. At the top, the shell history, start and end times, and environment variables are displayed. Below, the contents of files that were changed are shown with deletions highlighted in red and additions in green.

logs are editable and support Wiki formatting. The Wiki-like nature of the shell logs allows for easy annotation and cross-referencing.

After the Wiki portion of the log (which contains shell history, environment variables, and any other pertinent shell data), the modified files are displayed as colored diffs. All of the modified files are stored in the database, but not all are displayed by default. The shell log edit page provides a list of all files from which individual files can be selected for display.

Console Tools

The Trackle console tools consist of several scripts supporting the main interface, trackle-cli. Like most of the rest of Trackle, the console tools are written in Python for easy extensibility and maintenance.

Trackle-cli: The primary interface to the ticket database from the console is trackle-cli, a screen-oriented program for the curses environment. It is written using DTK,[12] a Pythonic wrapper for the curses Python module. Following our goal of minimalistic interfaces, trackle-cli is designed to give enough information that sysadmins can quickly find a ticket and begin working on it, but not so much that they are bogged down by text-filled screens of details. The two primary screens of trackle-cli are the ticket overview and ticket detail screens.

The ticket overview screen (Figure 4) lists the open tickets, showing for each the values of a configurable set of ticket fields. The default set shows the ticket's numeric ID, summary, owner, and time of last change. Sysadmins navigate this list using the keyboard and may either select an existing ticket or create a new ticket.

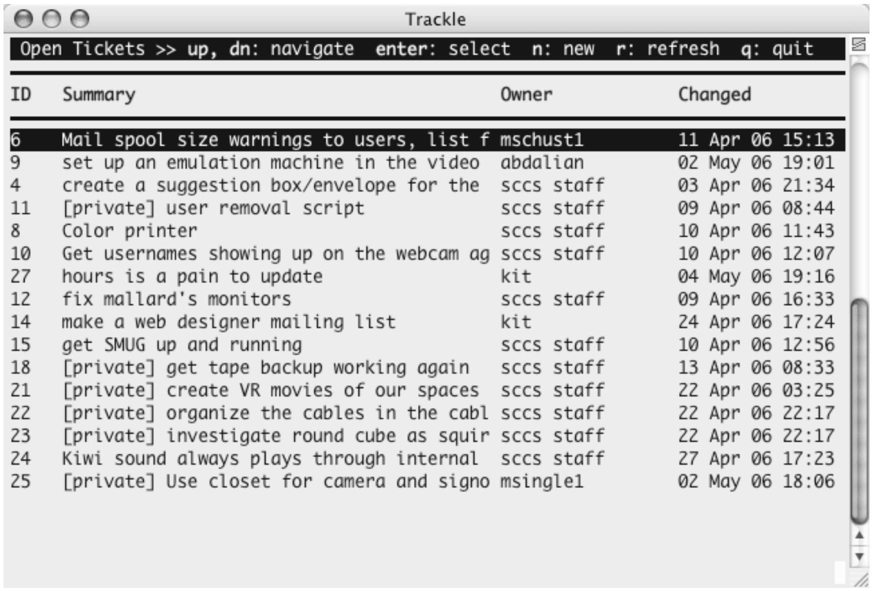
The ticket detail screen (Figure 5) shows the most pertinent details of an individual ticket, with short ticket fields displayed above the fold and the longer ticket description shown below in unformatted Wiki text. Some ticket information, notably the ticket change log, is not available through the console interface due to space constraints.

Trackle-cli supports editing of all visible information in the ticket detail screen. After selecting a field, the focus is moved to either a text editing field or an enumerated selection field which displays appropriate available values. Enumerated fields can have possible values added to them through the web interface or trackle-admin program. Ticket editing in trackle-cli is designed to allow a sysadmin to correct minor errors found in a ticket – full-featured editing is available in the web interface.

Sysadmins can edit the ticket description using the editor set in the VISUAL or EDITOR environment variable. When the editor terminates, the description display area is updated with the new value. If the sysadmin wishes to save the changes back to the Trackle database, trackle-cli prompts for a ticket changelog message using the same method.

From the ticket detail screen, the sysadmin can start a tracked shell session. Trackle-cli is suspended and replaced by a shell with a modified environment. Upon completion of the shell session, the sysadmin is shown a list of all the files that were modified during the session (see Figure 6). The sysadmin then selects files to display in the web shell session log. This selection may be changed later through the web interface. After selecting the files and saving the session log, the sysadmin is returned to the ticket detail screen.

[12] https://firefly.student.swarthmore.edu/trac/wiki/DTK

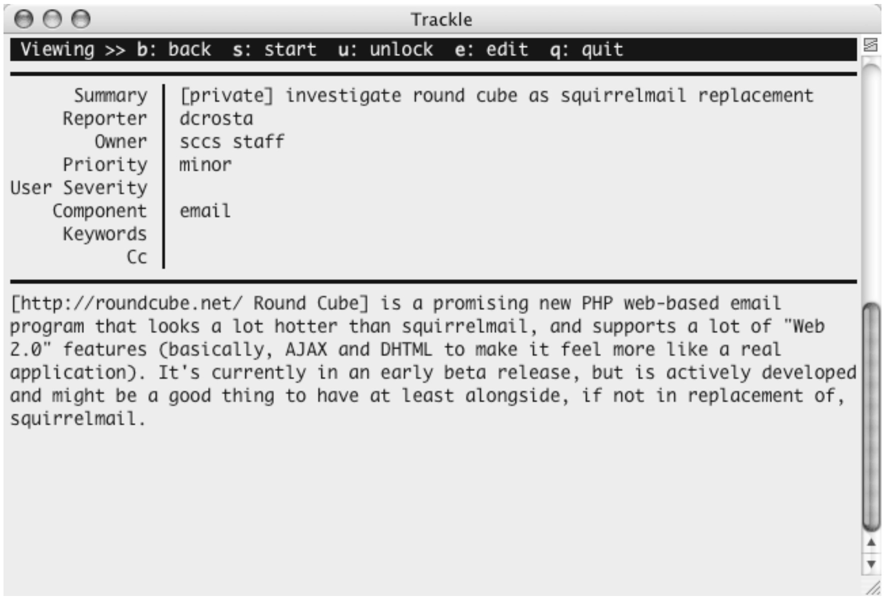


```
 Open Tickets >> up, dn: navigate  enter: select  n: new  r: refresh  q: quit

ID   Summary                                   Owner           Changed

6    Mail spool size warnings to users, list f mschust1        11 Apr 06 15:13
9    set up an emulation machine in the video  abdalian        02 May 06 19:01
4    create a suggestion box/envelope for the  sccs staff      03 Apr 06 21:34
11   [private] user removal script             sccs staff      09 Apr 06 08:44
8    Color printer                             sccs staff      10 Apr 06 11:43
10   Get usernames showing up on the webcam ag sccs staff      10 Apr 06 12:07
27   hours is a pain to update                 kit             04 May 06 19:16
12   fix mallard's monitors                    sccs staff      09 Apr 06 16:33
14   make a web designer mailing list          kit             24 Apr 06 17:24
15   get SMUG up and running                   sccs staff      10 Apr 06 12:56
18   [private] get tape backup working again   sccs staff      13 Apr 06 08:33
21   [private] create VR movies of our spaces  sccs staff      22 Apr 06 03:25
22   [private] organize the cables in the cabl sccs staff      22 Apr 06 22:17
23   [private] investigate round cube as squir sccs staff      22 Apr 06 22:17
24   Kiwi sound always plays through internal  sccs staff      27 Apr 06 17:23
25   [private] Use closet for camera and signo msingle1        02 May 06 18:06
```

Figure 4: Trackle-cli's ticket overview list, sorted by change date. The "Help Bar" at the top lists currently active keybindings. Trackle-cli is designed to work in an 80 × 24 character window.

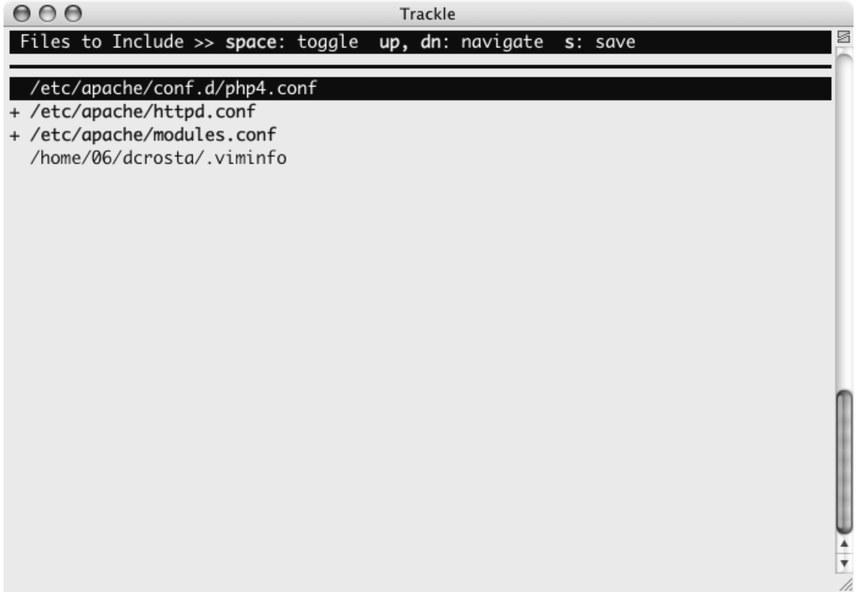
Trackle-shell-prompt: Our experience at the SCCS with the beta release of Trackle showed that even experienced sysadmins occasionally forgot whether they were in an ordinary shell or in a tracked shell spawned by trackle-cli. To address this, we created trackle-shell-prompt, a helper script which detects whether the user is running inside a tracked shell session by checking for certain environment variables. If so, it prepends the prompt with "Trackle" in green boldface. Trackle-shell-prompt also sets its exit code to 0 when in a shell tracking session, or 1 when not, so that it can be used in shell scripts.

Trackle-admin: This tool is used to configure the run-time settings of Trackle stored in the central database. Trackle-admin is used to add, update, or remove accounts (only sysadmins need an account for Trackle – end users may use the web interface without authentication), and the following enumerated ticket attribute fields: component (a brief description of the area of the system affected by a problem), milestone, priority, user severity, and ticket type. Wiki pages can be imported from or exported to plain text files. Like trackle-shell-prompt, the exit code returned by trackle-admin is set to 0 on success, or another value on error.



```
Viewing >> b: back  s: start  u: unlock  e: edit  q: quit

      Summary | [private] investigate round cube as squirrelmail replacement
     Reporter | dcrosta
        Owner | sccs staff
     Priority | minor
User Severity |
    Component | email
     Keywords |
           Cc |

[http://roundcube.net/ Round Cube] is a promising new PHP web-based email
program that looks a lot hotter than squirrelmail, and supports a lot of "Web
2.0" features (basically, AJAX and DHTML to make it feel more like a real
application). It's currently in an early beta release, but is actively developed
and might be a good thing to have at least alongside, if not in replacement of,
squirrelmail.
```

Figure 5: Trackle-cli displays the most pertinent ticket details, and allows simple editing to correct any mistakes. From this screen the sysadmin may begin recording shell actions to be associated with this ticket.



```
Files to Include >> space: toggle  up, dn: navigate  s: save

  /etc/apache/conf.d/php4.conf
+ /etc/apache/httpd.conf
+ /etc/apache/modules.conf
  /home/06/dcrosta/.viminfo
```

Figure 6: Trackle-cli allows the sysadmin to decide which files will be displayed in the web version of the shell session log.

Shell Session Tracking

The ability to track shell sessions sets Trackle apart from previous solutions. The tracking system removes the burden of self-reporting from the sysadmin. Relying on the sysadmin to report exactly what changes were made can be problematic. After a long session, even the most conscientious sysadmin may be unable to remember exactly what they did, let alone record it accurately. A seemingly inconsequential change made early on may be overlooked after tackling a more frustrating issue.

Recording incorrect or incomplete information defeats the purpose of tracking changes because it reduces the accuracy of the report and degrades the utility of the system as a training tool. The goal of Trackle is to automate as much record collection as possible in order to avoid human error.

To provide an accurate summary of a shell session, Trackle collects information about the session:

- start and end time
- environment variables
- shell history
- copies of all created, removed, or modified files

Most of these items are easy to collect using built-in Python commands. The history of shell commands can be obtained from the bash shell.[13] This information is stored in a state directory created for each tracked shell session.

Recording file changes is the tricky part. For this we use library interposition. Our interposition library, libtrackle, is based on the work done by Kuperman and Paksoy for Audlib. We can intercept library calls by writing our own version of the call in question with the same signature. Effectively, we wrap the library call with additional functionality to support Trackle. In this function we record any information we need and then pass the call on. This whole process is transparent to the running program, and as long as the parameters and return values are passed through to the real version without modification, interposition does not change the program's behavior.

In order to catch file modifications we need to intercept file-related calls (fopen, chmod, etc.) before they get to the kernel so that we can make an initial copy of the file in question. Once we have a copy, we allow all further calls on that file to pass through to the kernel without logging. Just before the program terminates, we make another copy of the file to compare against the original version.

In addition to intercepting calls, interposition allows us to define functions that are called when the library is first loaded and when the process terminates. We use the initialization routine to cache local state variables originally set in the environment by the console tools. In the finalization function we iterate over all files that have been accessed during the execution of the program and make a final copy of all files which have been modified.

A file might be modified several times during the course of a tracked shell session. We record the initial copy only once but overwrite the final copy each time a program terminates. Therefore, the difference information generated consists of changes made over the entire session.

Some programs written with security in mind attempt to disable library interposition (for good reasons), and sysadmins are encouraged to use these sorts of tools (e.g., sudo) when changing system configuration. In order to collect these changes, we need to prevent these programs from disabling libtrackle. For example, the environment created for executing vim by "sudo vim /etc/filename.cfg" lacks the LD_PRELOAD environment variable, effectively disabling libtrackle. We could not find a way to override this behavior in sudo without modifying its source code. We circumvented this issue by overriding the exec library calls with our own versions that reset the LD_PRELOAD variable to the proper value before executing the program.[14] This solution ensures that libtrackle is loaded for all dynamically linked programs.

Email Notification and Confirmation

Email plays two roles in Trackle, providing notification to sysadmins and end users, and email confirmation to prevent abuse of the web interface by unauthenticated users.

Whenever a sysadmin changes a ticket's status, like closing an open ticket or assigning a new ticket, a configurable list of staff email addresses (in SCCS's case, our staff email list address) is sent a notification indicating the status change and any comments or changed fields changed at the same time. Subscribed users receive an abbreviated version of this email containing just the status change information and the comment.

Trackle also provides the `trackle-notify` script which sends periodic updates to the staff email addresses. The email contains a list of all open tickets, details for each open ticket, and a list of ticket status changes since the previous periodic update was sent. Trackle-notify is designed to be run through cron or another task scheduler, therefore it produces no output under normal circumstances.

The email confirmation system allows end users to confirm their identity without requiring them to get, remember, and use a separate set of user names and passwords. When an end user creates a ticket or subscribes to an existing ticket, Trackle sends an email to

[13] Trackle currently only supports the bash shell. Support for tcsh is planned for a future release.

[14] For setuid programs, ld.so only honors values of LD_PRELOAD that are in the default library search path and are also setuid. A user would have needed administrative access to put such a library in place, so this is not a major vulnerability.

the address the end user entered. Users click the link in the email to make the action take effect.

Sample Workflows

Trackle is designed with human interaction in mind, and as such any description of its components does not fully capture the feel of the application. In this section we describe two sample workflows to demonstrate the expected usage of Trackle: 1) the lifecycle of a typical ticket created by an end user, and 2) a sysadmin annotating existing resources for future reference.

Ticket Lifecycle

Suppose SCCS user Alice is unable to access the administration page for her group's email list. She points her browser at Trackle's ticket creation page, http://bugs.sccs.swarthmore.edu/ , and begins to fill out the form. She is prompted for her email address, a brief summary, a ticket type (software bug, software request, security issue, etc.), a user severity (that is, how severe the issue is to Alice) and a longer description of the problem. Alice notices the link to a reference on Wiki formatting, and helpfully formats her description into several sections including error output she received, a link to the page with the problem, etc.

When Alice clicks the submit button, she is taken to a screen informing her that she will receive an email containing instructions and a link to confirm her ticket, and that her email address has been automatically subscribed to the ticket for status updates. Without clicking this link, her ticket will not show up in ticket reports, and the sysadmins will not be notified. Once she clicks the link, Trackle marks the ticket as confirmed and emails the sysadmins to notify them of the newly created ticket.

Now suppose SCCS user Bob is also unable to access the administration page for his group's email list. Because he read the page on ticket etiquette, he knows to first use Trackle's built-in search feature to see if any open tickets are already filed for this problem. He notices the ticket Alice just created, and confirms that it is for the same issue he is facing, so he does not need to file his own ticket. But Bob is impatient and has important work to do on his list, and wants to know as soon as the problem is resolved. Fortunately, Trackle allows Bob to subscribe to the ticket Alice created, again using email confirmation to verify his identity.

Now suppose SCCS sysadmin Melvin checks either his email or the Trackle website and notices the newly created ticket. Being our resident expert on Mailman list administration, Melvin SSH'es to our primary web and email server, runs trackle-cli, and begins working on the ticket. Behind the scenes, trackle-cli has locked his ticket so that no other sysadmin can begin working on the same issue. His shell history as well as any files he changes are now being recorded, so as he investigates and solves the problem, all his actions are logged. When he types 'exit' to leave the shell tracking session and returns to trackle-cli, he is shown a list of all the files whose contents or permissions he changed, and he can select which will be included in the log by default. When he saves the shell session log to the database, the ticket is automatically unlocked.

Since, by virtue of his Mailman wizardry, Melvin resolved the problem for Alice and Bob, he opens up the ticket's page in Trackle's web interface, makes some remarks to be appended to the ticket's history, and marks the ticket closed. Trackle then emails all the subscribed users to notify them that the issue has been resolved. This notification includes the remarks Melvin appended to the ticket's history.

Annotation and Cross-Referencing

Suppose SCCS sysadmin Wendy, our resident Wiki enthusiast, decides it is time for her to learn more about Mailman. She begins by searching through the existing Wiki pages, tickets and shell session logs to find anything that might be related to Mailman lists. She creates a new Wiki page, which she protects from anonymous access since it may contain sensitive information that should not be leaked to the public, and begins adding links.

In Trackle, all Wiki pages, tickets, shell session logs, and milestones are linkable objects, and all have built-in Wiki support so that any of them can link to any of the others. Wendy takes advantage of this by not only including forward links from the new Wiki page to the related tickets and shell session logs, but by editing those objects to include return links to the new page she is creating.

By default, the shell session logs that are captured by Trackle are pretty bare and contain just collected information. Since Wendy is in no hurry, she is at ease to spend some time investigating the effects of the commands recorded in the shell history of the session log created by Melvin above. She can integrate this information, along with links to other Trackle objects, online documentation, or any website, by editing the Wiki page at work behind each shell session log. She can remove commands from the history that are not pertinent so that the shell session log contains just the relevant information for future reference. She can also change the set of files which are displayed in the shell session log, overriding the defaults Melvin selected at the end of his shell tracking session.

Future Plans

As with any large software project, the initial public release of Trackle is far from complete. As frequently happens with new software, some of the most interesting features were not suggested until the SCCS started using Trackle, and others were inspired as a byproduct of the development process. Many of these new feature ideas that came to us mid-project made it in to the initial public release, but some did not. The following are planned features for Trackle that have not yet been implemented.

Ticket extensions: Early feedback from SCCS sysadmins has shown that some extensions to the ticket system may be useful, particularly the ability to express relationships among active tickets. A dependency relationship (the completion of ticket B can only happen after ticket A is closed) could hide or deprioritize tickets depending on others, or emhpasize tickets which are depended on. A parent-child relationship would be used to group several related tickets (e.g., configuration upgrades) under one parent (e.g., Linux distribution upgrades). Ticket due dates would enable Trackle to automatically escalate a ticket's importance over time. Finally, private tickets (not visible to unauthenticated end users) would be used to track sensitive information such as hiring or policy debates.

Multiple machine support: All of the components of Trackle interact with one central PostgreSQL database. Currently, the Trackle console tools do not keep track of host-specific info, so only one machine can be tracked per instance of Trackle. Because PostgreSQL communicates over TCP, it should not be difficult to add network functionality to the console tools. It would then be possible to install the console tools on any number of properly configured servers and clients, and have them all report back to one central database.

There are, however, some situations where a per-machine instances of Trackle might be useful. Trying to track many disparate issues occurring on unrelated machines would become cumbersome and is unnecessary. An alternative that might provide the best of both worlds would be a hierarchical approach. Rather than storing all the data on one central server, leave the data distributed, but allow communication between the individual instances of Trackle. For example, deploy one network-wide overview server, one site-wide overview server per site, and install Trackle individually on the machines at each site. This could help mitigate some of the scaling issues that would come with very large databases.

File revision control: We briefly mentioned configuration management tools, some of which include revision control functionality. Trackle is based on the open source program Trac, which includes tight integration with the Subversion revision control system, and could provide revision control for all files touched during shell session tracking. A straightforward approach of storing one revision per session would not work, as the files in question may also change between tracked shell sessions. A better approach is to create a revision at each of the before and after states for each file in the repository.

Further high-level abstractions: One recent innovation of Web 2.0 technologies is the establishment of new interface and organization paradigms that more closely model how people think about information. In particular, the tagging concept, where arbitrary words or phrases are associated with each idea or object in a system, supported by an interface which makes suggestions about tags to apply, would further increase the utility and scalability of Trackle. Some tags for shell session logs could be automatically generated, for instance, a tag for each file and each directory involved in a particular shell tracking session. Tags could also be assigned for software packages involved in a shell tracking session, by integrating with package management tools (dpkg, rpm, etc.).

Conclusions

Our experience with the SCCS staff email list and Wiki has shown that relying on self-reporting leads to missing, incomplete, or inaccurate reports of changes made to our systems. The poor quality of these reports makes it hard to find the source of a particular change. Further, an inaccurate report might cause a sysadmin trying to duplicate past steps to cause new errors instead of repairing existing ones. Additionally, the presence of inaccuracies degrades confidence in all reports. Trackle alleviates these problems by keeping consistent and accurate records so that sysadmins can focus on solving problems rather than the tedious task of keeping logs.

Trackle's integrated Wiki has allowed us to begin collecting related topics into a sysadmin training manual and how-to guide. This evolving guide allows new sysadmins to ask more sophisticated questions of their experienced colleagues. Trackle allows sysadmins to learn on their own time and at their own pace so that no one gets bored or left behind.

Often, volunteer sysadmins learn the intricacies of a system only when it breaks. Trackle allows us to learn by reflection rather than by struggling to fix a critical error. This leads to more efficient use of office time for those not present when problems occur. We also learn from authentic situations rather than toy problems or contrived examples.

Because our shell tracking tools operate transparently, Trackle can be used to complement existing change/configuration management systems. Though many configuration management systems have already solved the problem of discovering file changes, Trackle goes further to associate file changes with a particular issue. Additionally, Trackle detects changes to any files, not just those that are already being monitored by a configuration management system.

Availability

Trackle is open source software, licensed under the BSD license. You may download stable Trackle releases and documentation from http://www.sccs.swarthmore.edu/org/trackle/ .

Acknowledgements

We would like to thank Benjamin A. Kuperman and Mustafa Paksoy for their work on Audlib [5], on which libtrackle is based. We would also like to thank Edgewall Software and the numerous contributors to Trac, without which Trackle would not have been possible.

Author Biographies

Daniel S. Crosta graduated from Swarthmore College in June, 2006, with a B.A. in Computer Science. During his tenure at Swarthmore, he served three years as Systems Administrator for the Swarthmore College Computer Society, most recently as Lead Systems Administrator. He has also participated in research in Computer Graphics at Princeton University, and in Computer Vision at Swarthmore College. Since July, 2006, he has been working as a Software Developer at Wireless Generation in New York City. Contact him electronically at dcrosta@sccs.swarthmore.edu .

Matthew J. Singleton is a currently a senior at Swarthmore College in Swarthmore, PA, where he is a double-major in Computer Science and Linguistics. He is also the Lead Systems Administrator for the Swarthmore College Compter Society. He has participated in Computational Linguistics research in the Department of Computer Science at Swarthmore College. Reach him electronically at msingle1@sccs.swarthmore.edu .

Benjamin A. Kuperman received the M.S. and Ph.D. degrees from the Department of Computer Sciences at Purdue University in 1999 and 2004. He is an assistant professor at Oberlin College in Ohio and previously taught at Swarthmore College in Pennsylvania. While at Purdue, he was a researcher in the Center for Education and Research in Information Assurance and Security (CERIAS) for five years and was affiliated with COAST before that. His main areas of research are on host-based computer security monitoring systems and OS level audit systems. Reach him electronically at Benjamin.Kuperman@oberlin.edu .

Bibliography

[1] Anderson, Paul and Edmund Smith, "Configuration Tools: Working Together," *Proceedings of LISA 2005: 19th Systems Administration Conference*, pp. 31-37, December, 2005.

[2] *Common Criteria for Information Technology Security Evaluation*, http://www.commoncriteria.org/ .

[3] Evard, Rémy, "An Analysis of UNIX System Configuration," *Proceedings of LISA 1997: 11th Systems Administration Conference*, pp. 179-193, October, 1997.

[4] Kuperman, Benjamin A., *A Categorization of Computer Security Monitoring Systems and the Impact on the Design of Audit Sources*, Ph.D. thesis, Purdue University, West Lafayette, IN, August, CERIAS TR 2004-26, 2004.

[5] Paksoy, Mustafa and Benjamin A. Kuperman, "Audlib: Generating computer security audit logs with interposing libraries," Presented at 2005 Swarthmore College Sigma Xi poster session, September, 2005.

[6] Pomeranz., Hal "PLOD: Keep Track of What You're Doing," *Proceedings of LISA 1993: 5th Systems Administration Conference*, November 1993.

[7] Roth, Mark D., "Preventing Wheel Reinvention: The psgconf System Configuration Framework," *Proceedings of LISA 2003: 17th Systems Administration Conference*, pp. 205-211, October, 2003.

[8] Sharp, James M., "Request: A Tool for Training New Sys Admins and Managing Old Ones," *Proceedings of LISA 1992: 4th Systems Administration Conference*, October, 1992.

[9] Stepleton, Thomas, "Work-Augmented Laziness with the LOS Task Request System," *Proceedings of LISA 2002: 16th Systems Administration Conference*, November, 2002.

[10] US Department of Defense, *Trusted Computer Systems Evaluation Criteria* (also known as the 'Orange Book') Technical Report DoD 5200.28-STD, DoD Computer Security Center, Fort Meade, MD, December, 1985.