The Making of "The Advanced Persistent Threat You Have: Google Chrome"

Todd Heberlein

Google's software update system can serve as a model Advanced Persistent Threat (APT). APTs often embed programs in a penetrated system. These programs wake up from time to time, call home, download additional programs and instructions to carry out, and modify systems. Google's software update performs all these steps too. Furthermore, because the Google Chrome browser is so widely used and updated so frequently, Google's update process provides analysts ample opportunity to test their data sources, tools, and skills for their ability to detect and reconstruct the "attack". The paper "The Advanced Persistent Threat You Have: Google Chrome" made the claim that if the analyst could not perform the analysis of Google's update system, they were probably not prepared for malicious APTs. The paper then provided a partial reconstruction of the update activity. This paper describes how the analysis in that first paper was performed. It describes the computer system, data collection, and analysis tool. It then shows how the tool and data were used to reconstruct the "attack".

1 Introduction

The paper "The Advanced Persistent Threat You Have: Google Chrome" (hereafter referred to as "The APT You Have" paper) describes Google's software update approach as a good model for an Advanced Persistent Threat (APT). Google's system is a model APT because it behaves like APT code and it is readily available in production systems.

Google's system behaves like code used by APTs. The "attacker" (for the moment, pretend Google is evil) has, probably unknowingly to you, deposited a program on your system deep in your home directory (not under /Applications). It runs periodically and automatically behind the scenes; you probably never know when it runs. This program reaches out to a server of the attacker's choosing, and from time to time downloads new programs and commands (e.g., shell scripts) crafted by the attacker. It then executes those programs and commands modifying critical software and files on your system. Then it cleans up after itself leaving virtually no evidence behind. In the instance discussed in "The APT You Have" paper only the Google Chrome binary and supporting files are changed, but there is nothing preventing the downloaded programs and commands from doing anything the attacker might want to do to your system.

Because Google Chrome is so widespread, there are probably instances of it running on production machines in most organizations. Theses are the very machines that may be targeted by malicious APTs.

The combination of APT-like behavior on production machines makes it a model APT that security administrators should practice their skills on. Can the analyst, on noisy and chaotic production

machines, detect the model APT running, detect the changes it makes, and reconstruct the "attack"? If the analyst can't perform these tasks for Google's software update system, he probably isn't ready to deal with malicious APTs.

"The APT You Have" paper shows a partial reconstruction of the attack. It shows how the bootstrapping program gets the ball rolling, the cascade of processes that are created and the programs they run, and it traces the flow of data from it being downloaded from the Internet to its insertion in the /Applications directory tree.

This paper shows how that graph (reproduced in Appendix A of this paper) was constructed using the information from audit data and the Audit Explorer analysis tool.

Section 2 describes the production machine, the data used, and the analysis tools used to analyze the "attack." Section 3 details how Audit Explorer was used to reconstruct the graph. It begins in Section 3.1 with a general discussion on strategy described as "find a thread then follow the thread". The rest of Section 3 shows how the strategy is applied. Section 4 wraps up with summary and concluding remarks.

2 The Set-up

The computer system used in "The APT You Have" paper was a production system used every day. This section describes the environment including the system, data source, audit configuration, analysis approach, and filter rules.

(Note: Because this is a production system, I am not releasing the original audit data. Audit data contains file and directory names, email folder names, email attachment names, web server addresses, and other data that can be sensitive.)

2.1 The System

The system is a Mac Pro workstation (MacPro1,1). At the time the data was collected it was running the latest version of Mac OS 10.6 (Snow Leopard). This workstation is shutdown every night and restarted every morning. For a variety of technical reasons, this makes audit analysis much easier than it is for servers that run continuously or systems that are frequently put to sleep and rarely shutdown. Each of these scenarios create slightly different audit analysis strategies which will be a topic for a future paper.

2.2 Audit Data

The word "audit" is overloaded and can mean many things depending on who is using the term and in what context. This paper refers to "audit data" as data specifically collected by the operating system for security purposes. For Apple's Mac OS X, it is the BSM data which, by default, is saved in the directory

```
/var/audit
```

For Windows 7, the equivalent is the Security.evtx data which, by default, is saved in the directory

```
%SystemRoot%\System32\Winevt\Logs
```

2.3 Audit Configuration

Apple's BSM audit system is primarily controlled by the file

```
/etc/security/audit_control
```

Apple's default configuration doesn't collect anything useful. The government's recommended audit configuration for Macs containing classified information isn't very useful either. Without a well-configured auditing system, you cannot get useful results when you analyze the audit trail. Figure 1 shows the audit_control configuration used for this analysis. The critical field is "flags:"; it is set to "all". Lion has a few additional (and undocumented) fields, but the primary ones a user should change are the same.

```
dir:/var/audit
flags:all
minfree:5
naflags:lo,aa,pc,nt
policy:cnt,argv
filesz:4G
expire-after:40G
```

Figure 1

2.4 Analyzing the Audit Data (ae_batch and Audit Explorer)

While Audit Explorer lets a user select a BSM audit trail file to analyze, the command-line tool ae_batch (for "audit explorer batch" mode) is often more convenient because the user can escalate privileges before running the program. Apple's BSM audit data is protected, so programs running under under normal user privileges cannot access the data. Apple doesn't allow in-program privilege escalation for applications distributed through the Mac App Store, so the Audit Explorer GUI application cannot directly read the BSM audit data. Because you can escalate privileges before running ae_batch (either from a shell or automatically via launchd) the program can directly read the BSM audit data and then save the analysis results to a separate file/bundle. The analysis results can then be opened by the Audit Explorer GUI application.

For more information on ae_batch, see the Help documentation in Audit Explorer or the online documentation at:

```
http://www.netsq.com/Tools/AuditExplorer/Docs/Manual_1.1/index.php?section=CLI
```

The system used for the data in this paper uses a launchd configuration that automatically runs ae_batch, saves the results, and emails a summary. Furthermore, for testing various filter configurations I often run directly from the command line. Figure 2 shows a typical example:

```
$ sudo /Applications/Audit\ Explorer.app/Contents/MacOS/ae_batch \
> -dst . \
> -filt ~/Documents/FilterRuleSets/default_plus_APTs.nsqFilt \
> -src /var/audit/20120405161339.20120406034159
```

Figure 2

This results in a file (actually a bundle) named 20120405161339.20120406034159.nsqAE which can be opened in Audit Explorer. (If the path given by the -dst option is a directory, the results file name is based on the original BSM audit trail file name.)

2.5 Filter Rules

Audit Explorer uses filter rules to highlight potentially interesting activity. These are starting points for exploring the audit data. In other words, this is one of the primary ways Audit Explorer says, "Here is a thread". Audit Explorer uses a default filter set, but it can use custom filter rules as well.

"The APT You Have" paper used a custom filter rule set, default_plus_APTs.nsqFilt. It is available online at:

`http://media.netsq.com/Downloads/default_plus_APTs.nsqFilt`

Audit Explorer Filter Editor, a free (albeit an early beta) application, can open, review, extend, and edit the filter set. Filter rule files are just XML text files, so they can easily be emailed or otherwise shared between users. Audit Explorer Filter Editor can be found at:

`http://www.netsq.com/Tools/AEFilterEditor`

3 Reconstructing The Graph

This section discusses how we build the graphs described in "The APT You Have" paper. The graph was drawn using the program iDraw, but all the data used to construct the graph came from the information Audit Explorer supplied.

3.1 General Approach

In actuality, there is one super-graph that represents all the activity in your system. Every process is created from another process (except the initial process); processes create, delete, read, and write files; processes create or accept network connections; and so on. All these objects and actions can be represented by a giant graph. Any malicious activity is simply a subgraph (or subgraphs) inside the giant graph.

Analyzing potential malicious activity then begins with finding a piece of the malicious subgraph, and then expanding your view until you have found the complete bounds of the malicious subgraph (or at least the bounds that you are interested in). This is a “find a thread” then “follow the thread” strategy. “Finding a thread” can be thought of as “detection”, as in detecting something suspicious. “Following a thread” can be thought of as forensics.

In practice, the detection and forensic activities often inform each other. When given just a small piece of the graph (e.g., a process writing to a file), essentially a detection event, it is often impossible to definitively determine if that piece truly is part of some malicious activity or not. Reconstructing a larger part of the graph surrounding that initial piece provides the context for evaluating that small piece. This reconstruction of the graph is the forensics activity.

In addition to determining if a piece of the graph is associated with malicious activity or not, the larger, surrounding subgraph lets you answer the questions “So what?” and “How did that happen?”. The first question is answered by looking at activity downstream in the graph from your starting point. The second question is answered by looking upstream in the graph from your starting point.

3.2 Unusual Executions

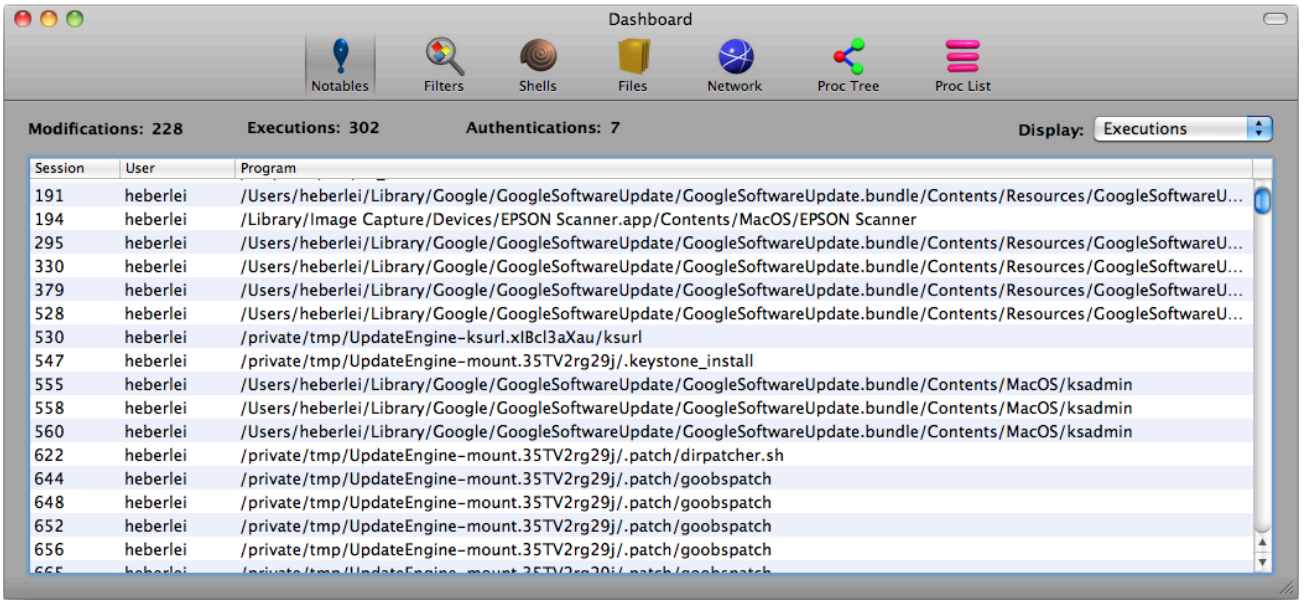

Figure 3

Under Audit Explorer’s Notable Events tab are events that are, well, notable. This was Audit Explorer 1.0’s attempt to say “Hey, here are some potentially interesting threads to follow!” The notable events highlighted three broad categories of activities: (1) modifications of files in core system directories, (2) Executions of programs outside the primary application directories, and (3) authentications where a user must authenticate himself to the system (e.g., when logging in or escalating privileges). For this day’s activities there were 228 modifications to the system, 302 executions of unusual programs, and a password entered 7 times. The pop-up menu on the right selects which group to display. Figure 3 shows the “Executions” group.

Modern computer systems generally store most of their programs in a handful of well known directories. For example, on the Mac the handful of locations includes the directories /bin, /usr/bin, and /Applications. An executable installed (and run from) outside these directories may indicate that the program was dropped there by an attack (e.g. a maliciously crafted PDF causing Acrobat Reader to drop a program in the user's home directory). Audit Explorer's "Notable Executions" group flags execution of these programs.

On Google software update days, the Modifications and Executions groups lights up like Christmas trees. There are a few instances of programs in this group for this computer that make sense, including the GoogleSoftwareUpdateAgent itself as well as the EPSON scanner application. These are programs installed in a user's Library directory when installing various applications. The vast majority of those 300+ executions shown in Figure 3, however, stem directly from the update system itself. These programs include ksurl, .keystone_install, dirpatch.sh and goobspatch. These programs are dropped into temporary directories during the update process, run, and then removed. The user is never asked permission before these programs are downloaded and run; indeed, the user will probably never be aware what is happening. The programs only exist on the computer for a couple of seconds to a couple of minutes before they are gone. In other words, this is similar to actions associated with programs used by some APTs.

While zooming in on almost any of these alerts is generally a fine place to start exploring and will lead you directly to the Google update process, the volume of alerts can be overwhelming. Because of this, Audit Explorer's Filters tab usually provides a better place to start exploring.

3.3 Filters: Any Connections

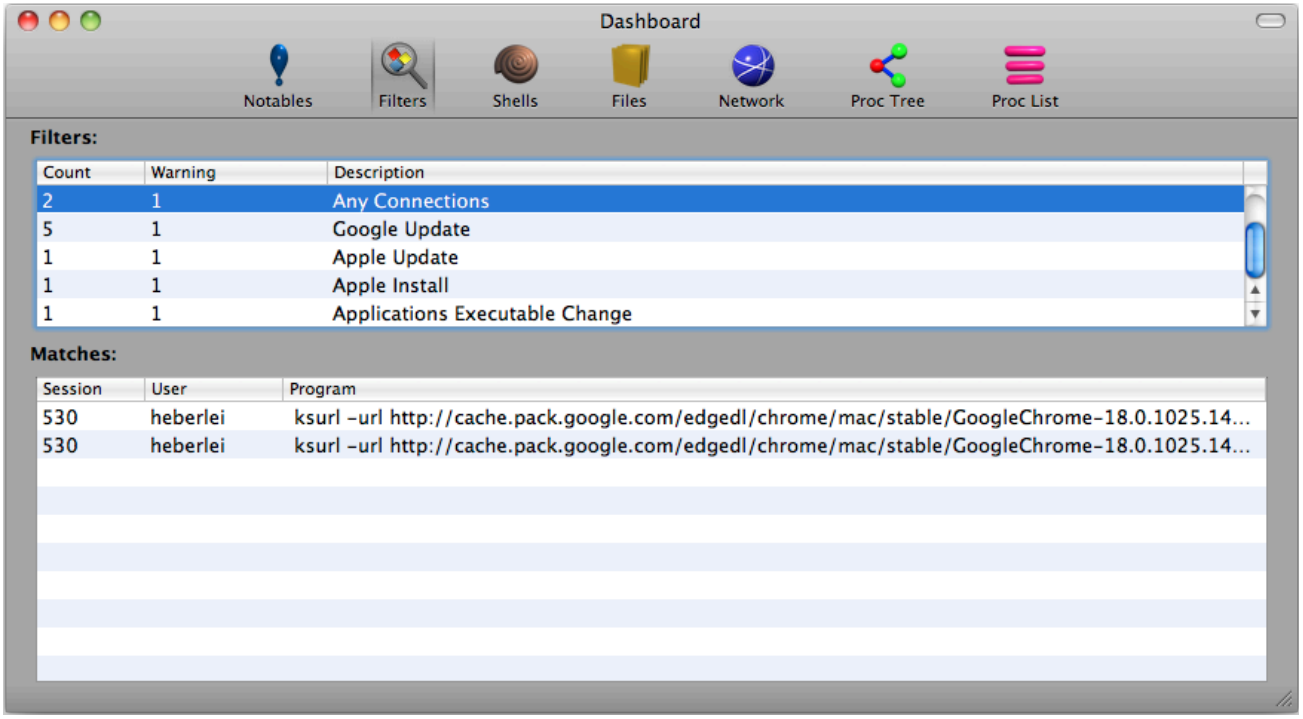

Figure 4

Version 1.1 of Audit Explorer added a Filters feature that is more powerful and expressive than the original Notable Events feature. In particular, the default Filter rules exempt programs that can trace their ancestry back to the Google software update process. In some ways this exemption is like punching a hole in your firewall with all the risks that that entails, but it does cut down on the number of alerts.

As it turns out, however, the Google software update still triggers at least one alert, the "Any Connections" alert (see Figure 4). Our custom rule set also includes two additional rules that highlight different parts of the update process. These will be covered in Sections 3.4 and 3.6.

The "Any Connections" rule selected in Figure 4 (in the top list) flags programs that are run from unusual directories (much like the Notable Executions) ***and*** make a network connection. In other words, it flags potentially dropped in programs that also communicate over the network.

The lower list show two events that match this filter, both from the same program called ksurl. ksurl, a program dropped in by the GoogleSoftwareUpdateProgram communicates with two web servers, thus triggering the rule twice.

These alerts are "Here is a thread" messages from Audit Explorer. Double clicking on an alert zooms in on the process which provides information to start building the graph in the "The APT You Have" paper.

3.3.1 ksurl Process Details

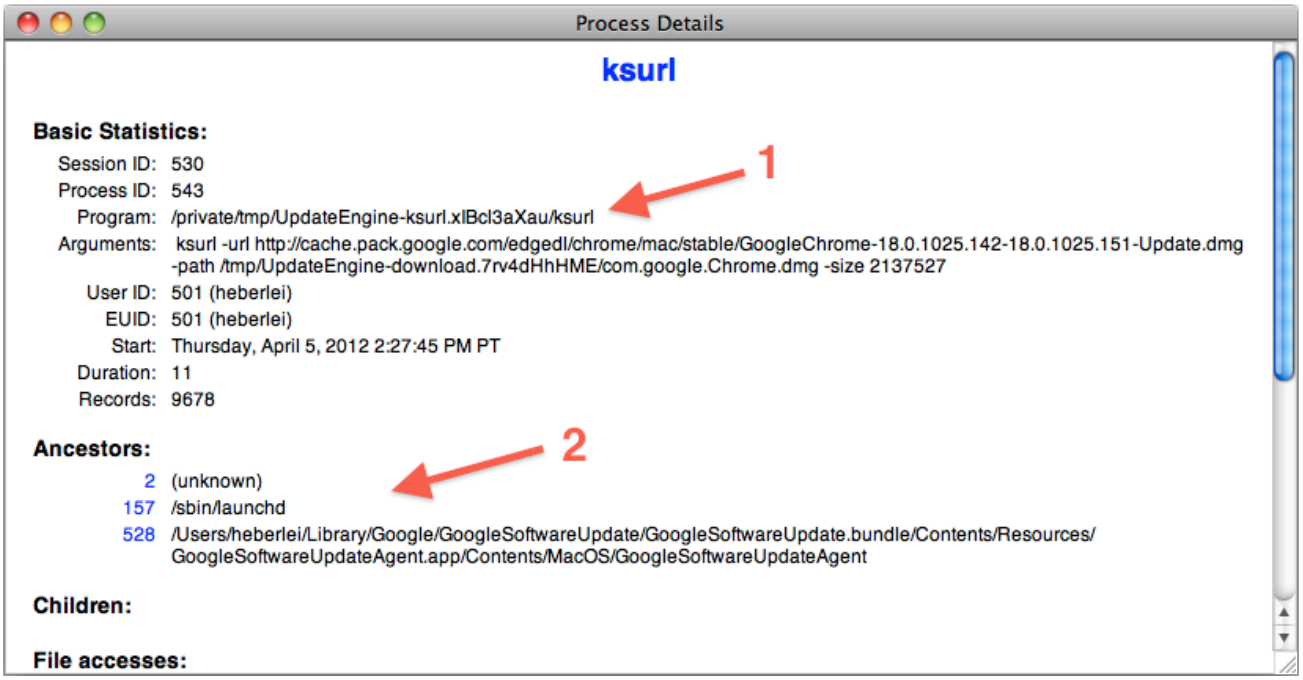

Figure 5

Double clicking on the ksurl session highlighted in the Filters tab in Figure 4 brings up the “Process Details” window shown in Figure 5. This window contains a wealth of information about a process. “Process Details” windows provided most of the information used to construct the graph in the “The APT You Have” paper. The next few paragraphs and figures show the information that can be extracts from this one instance of the ksurl process.

The line pointed to by label 1 in Figure 5 shows the full path to the ksurl program. This shows that the program is running out of a temporary directory. Figure 6, a slice of the graph from “The APT You Have” paper, shows the path to this file and the fact that the kusrl program is executing this code. One thing to note which will become important later is that the “Process Details” window in Figure 5 shows the path to the program beginning with “/private/tmp”, where as Figure 6 simply shows the path beginning with “/tmp”. This is an artifact of the Mac OS operating system. “/tmp” is simply a pointer to “/private/tmp”, so both prefixes can be used to access the same directory or file.

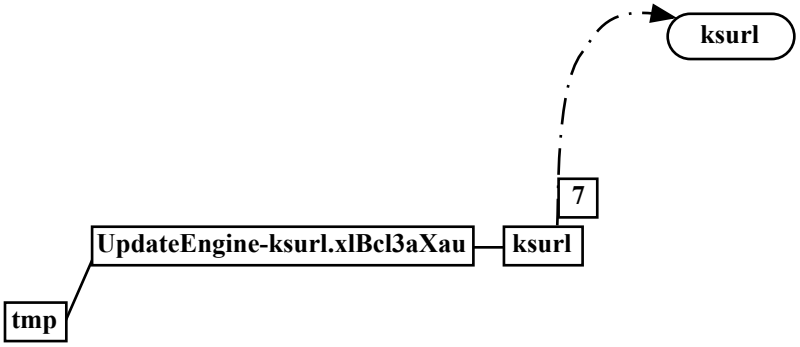

Figure 6

Label 2 in Figure 5 points to the process’s ancestors. With the exception of the first process created on the computer, each process is a child of another process. The “Ancestors” list shows the sequences of processes that eventually created the ksurl process. The first process is labeled as “unknown” because it started before the first record in the audit trail file. This process created a launchd process; the launchd processes created the GoogleSoftwareUpdateAgent (GSUA); and the GSUA process created the ksurl process. Figure 7 shows this process ancestry information layered into the graph.

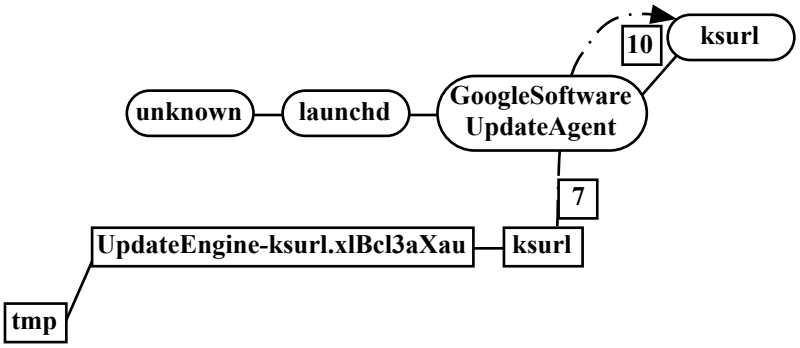

Figure 7

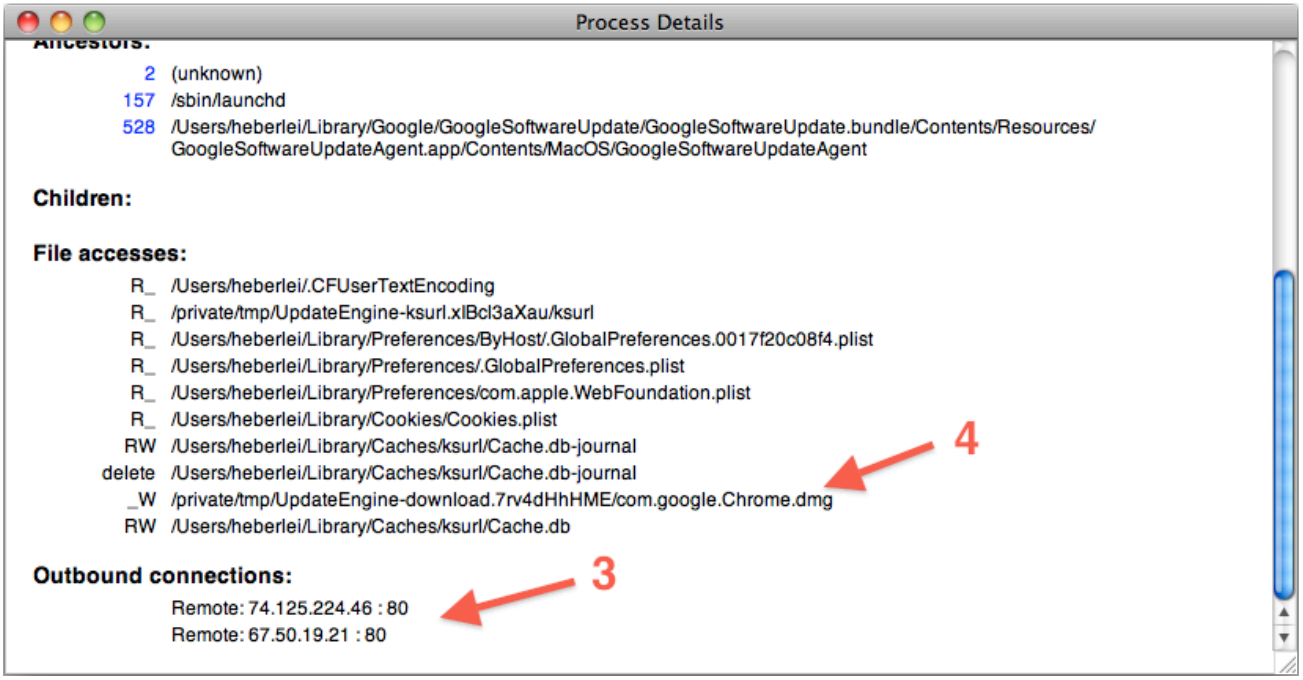

Figure 8

Scrolling down to the bottom of the "Process Details" window shows two more important pieces of information. Label 3 in Figure 8 shows the servers that the ksurl process connected to. The IP addresses for these servers are 74.125.224.46 and 67.50.19.21. Both connections were to port 80 on these servers. Figure 9 shows this information layered into the graph.

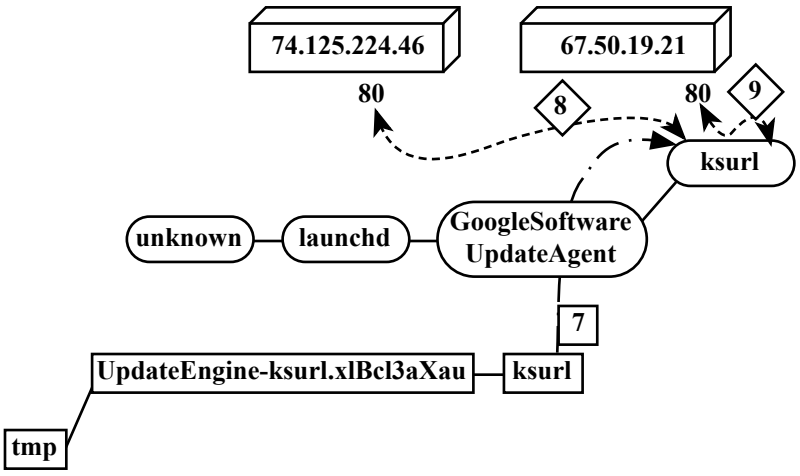

Figure 9

Label 4 in Figure 8 shows a file that ksurl wrote to (or created). The path is

```
/private/tmp/UpdateEngine-download.7rv4dHhME/com.google.Chrome.dmg
```

This shows that ksurl downloaded a Disk Image file. Figure 10 shows this information layered into the graph. (Again, the "/private/tmp" was shortened to "/tmp").

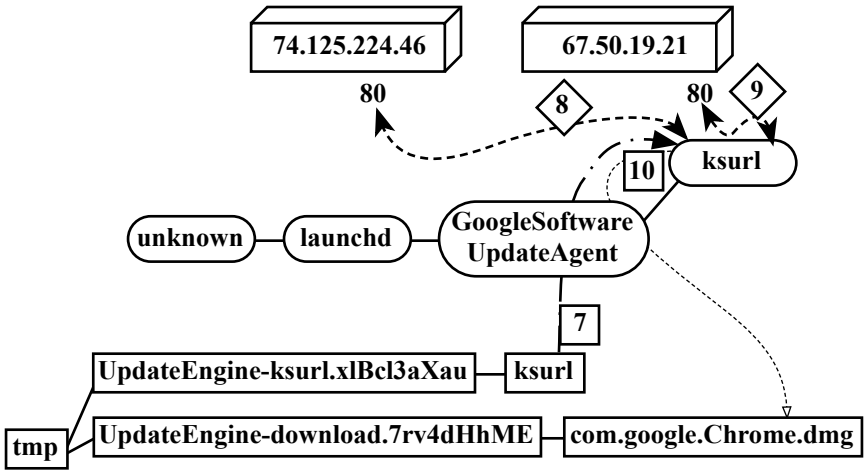

Figure 10

Figure 10 shows the relevant portion of the complex graph from "The APT You Have" paper that can be constructed with the ksurl process details. Two questions that you may ask at this point are: "What happens to the file com.google.Chrome.dmg?" (a "So what?" question) and "How was the ksurl program created in the first place?" (a "How did that happen?" question). The next section explores these answers.

3.3.2 ksurl Files

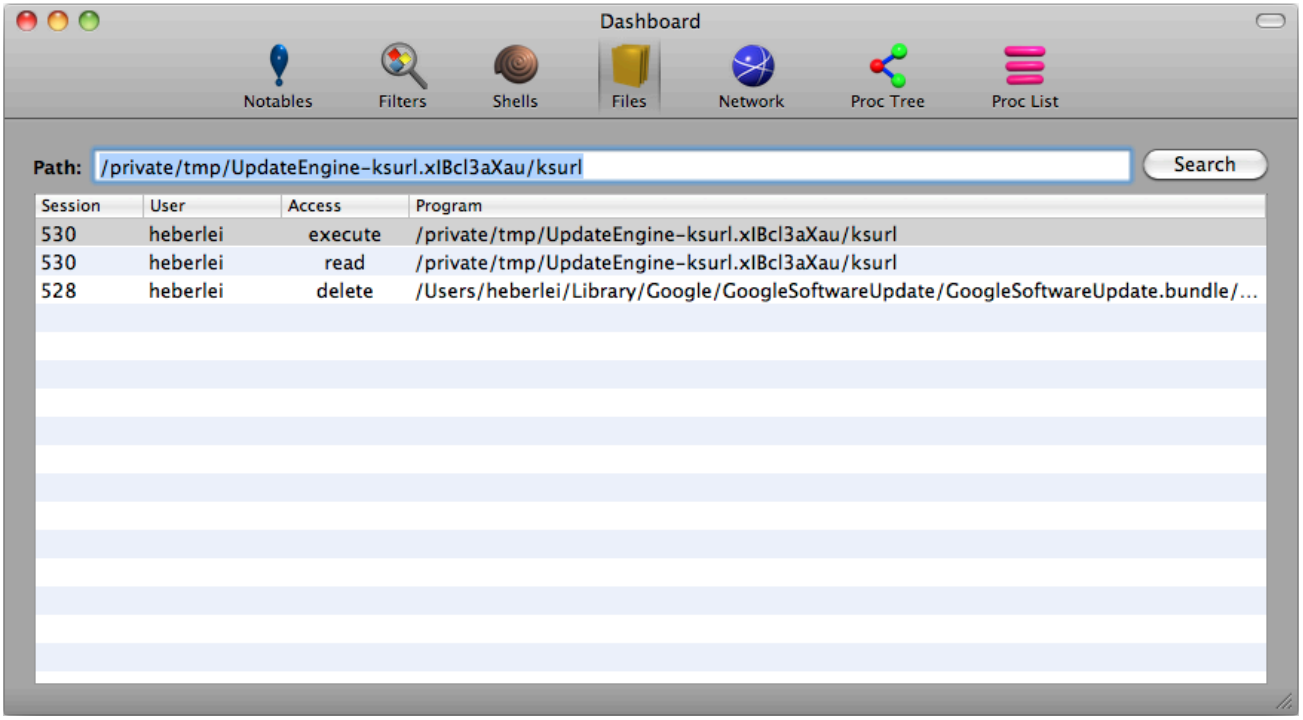

Figure 11

Audit Explorer's Files tab lets a user trace the history of a file. Given a file path, the Files tab shows what processes performed what operations on this file (or "file path" to be more precise). To answer the question "How was the ksurl program created in the first place?", enter the program file's path. Figure 11 shows the results of the search.

The first line shows the ksurl process executing the file. The second line shows the same process reading the file. This is a common pattern in the audit trails: execute followed by a read of the same file. The last line shows the GoogleSoftwareUpdateAgent deleting the ksurl file (getting rid of the evidence, so to speak). So the list shows the file being executed and deleted, but it doesn't show the file being created. Why not?

The answer lies in that issue mentioned earlier: "/tmp" points to "/private/tmp". Sometimes a program accesses a file with "/tmp" and sometimes with "/private/tmp", so when dealing with paths that begin this way, always try the alternative prefix. Figure 12 shows the search with the alternative path.

Figure 12 shows that the GoogleSoftwareUpdateAgent process created the ksurl program (zooming in on this process would show GoogleSoftwareUpdateAgent first wrote a temporary file name and then renamed it to ksurl). The two searches (one beginning with "/private/tmp" and one with "/tmp") show the program file being created, executed, and deleted. Figure 14 shows kusrl's creation layered into the graph.

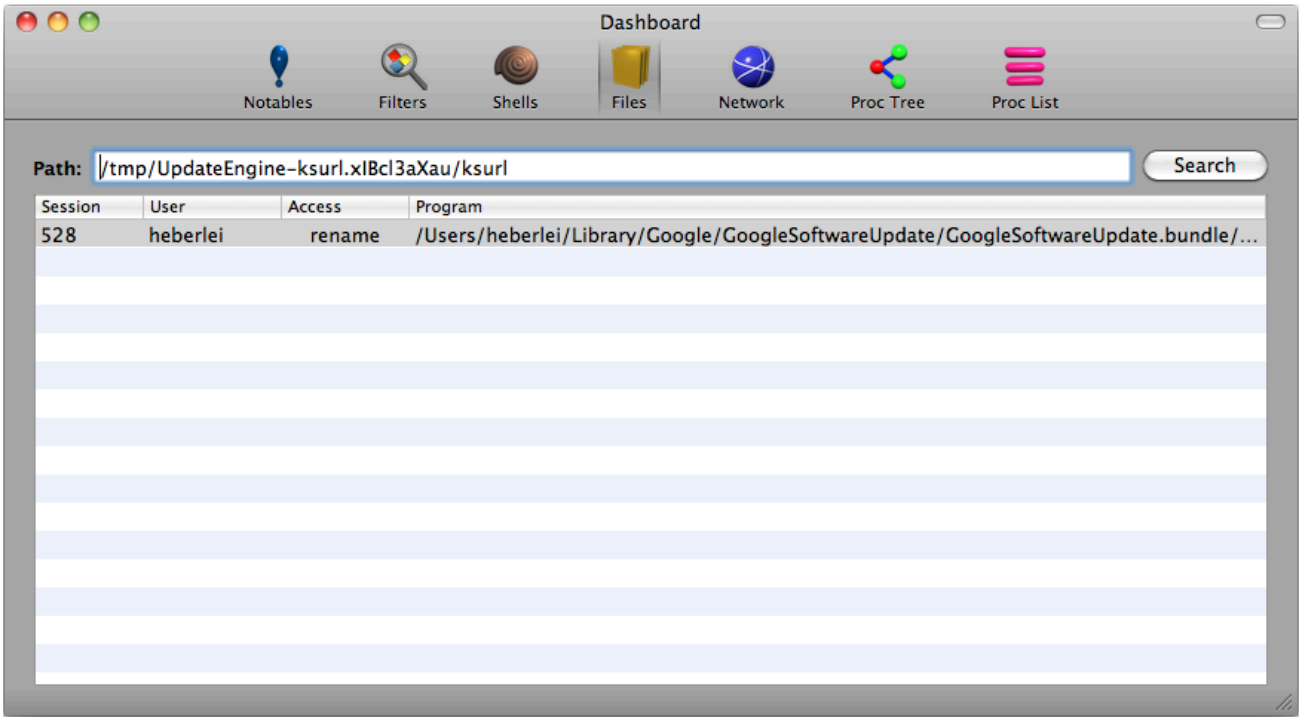

Figure 12

What about the question "What happens to the file com.google.Chrome.dmg?" (a "So what?" question). Yes, a process was created, it executed the program ksurl (an executable that was created on the fly by GoogleSoftwareUpdateAgent and then deleted). Did its execution have any consequences? Was the system changed? Were any user files exfiltrated?

The ksurl's "Process Details" window showed that it created no child processes. But it did create a file, namely com.google.Chrome.dmg. Answering the "So what?" question now becomes "So what happened to the file com.google.Chrome.dmg?"

The Files tab helps answer the question. Figure 13 shows the history of this file. First it shows ksurl writing to the file, but it also shows that GoogleSoftwareUpdateAgent read the file and then deleted it (getting rid of the evidence of the file's existence). To continue answering the "So what?" requires zooming in on the GoogleSoftwareUpdateAgent to find out what it did with the data from com.google.Chrome.dmg. For the sake of brevity, this paper does not follow this step. Figure 14 shows GoogleSoftwareUpdateAgent reading the data layered into the graph.

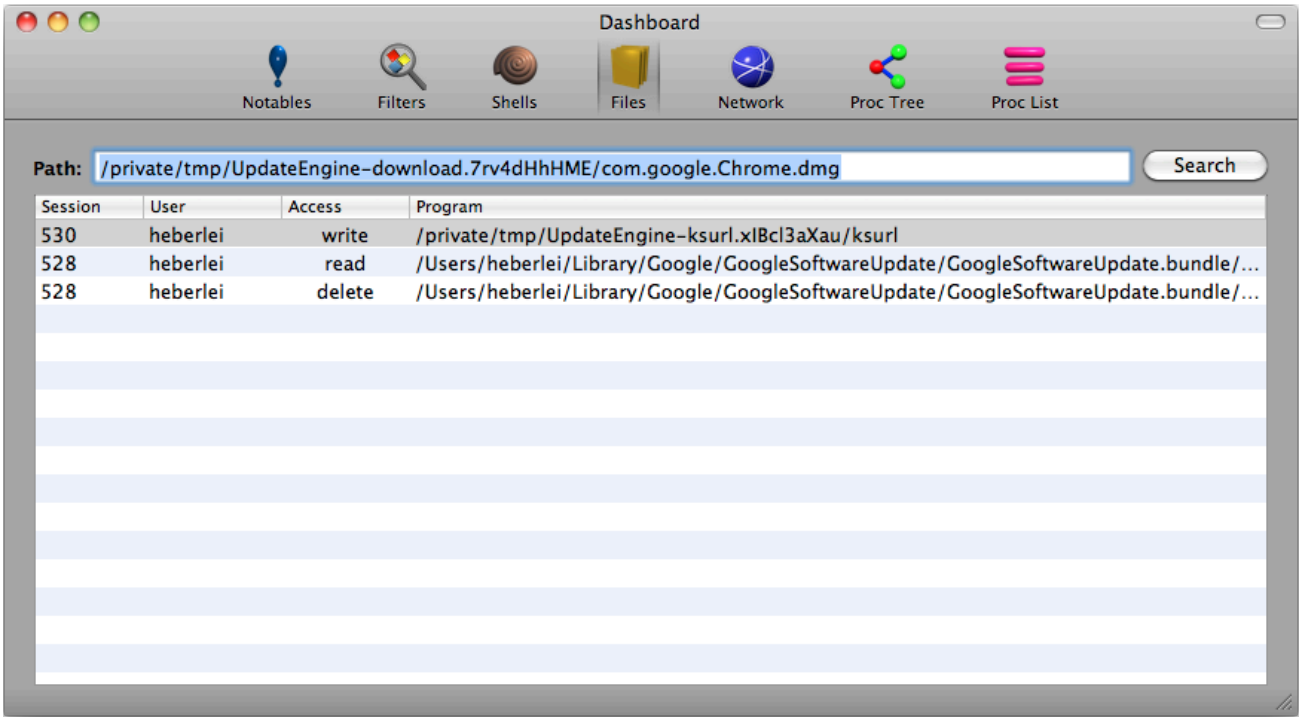

Figure 13

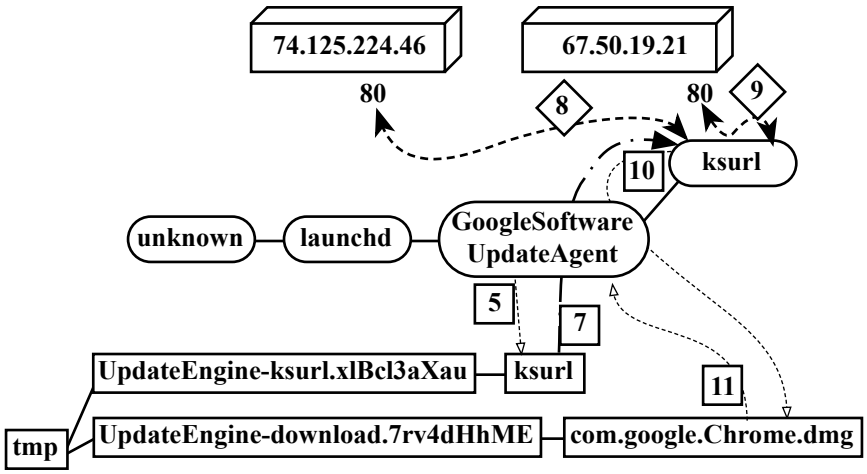

Figure 14

3.4 Google Software Update

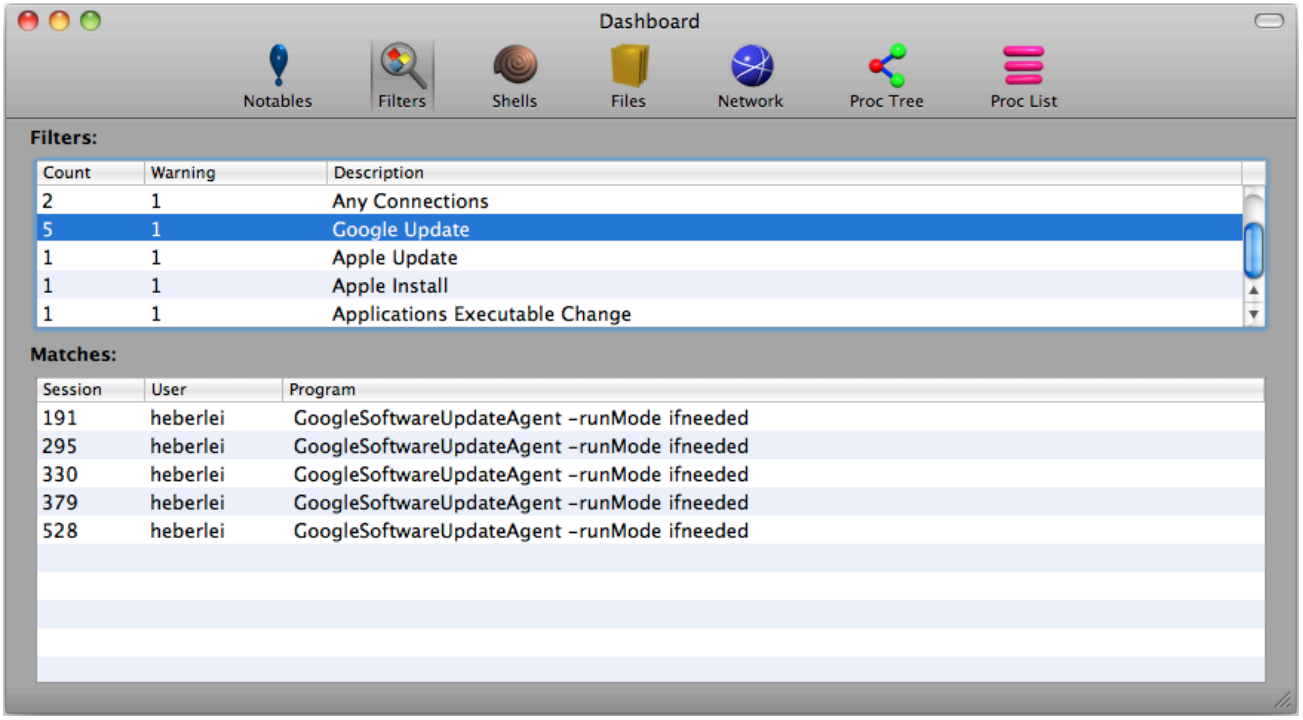

Figure 15

The "Google Update" filter rule is a custom rule (see Figure 15). This rule flags the start of the GoogleSoftwareUpdateAgent (GSUA). While Audit Explorer's default set of Filters now exclude unusual program executions or system modifications that can trace their ancestry back to GSUA, this custom filter rule keeps an eye on Google's activity. Figure 15 shows the program is started multiple times a day. Most of the time the program starts, makes one network connection, and then exits. In other words, most of the time there is no update to install, so the update process simply exits. Zooming in one one of these processes shows this behavior. Figure 16 shows the details for one of these processes. As the arrow points out, this particular instance of GSUA created no child processes.

Sometimes though, an update is available, and that triggers a whole cascade of activities. Figure 17 shows an example of this. It is most obvious by looking at the Children processes section – six child processes are created, including the running of the .keystone_install script which will ultimately lead to many modifications of the system.

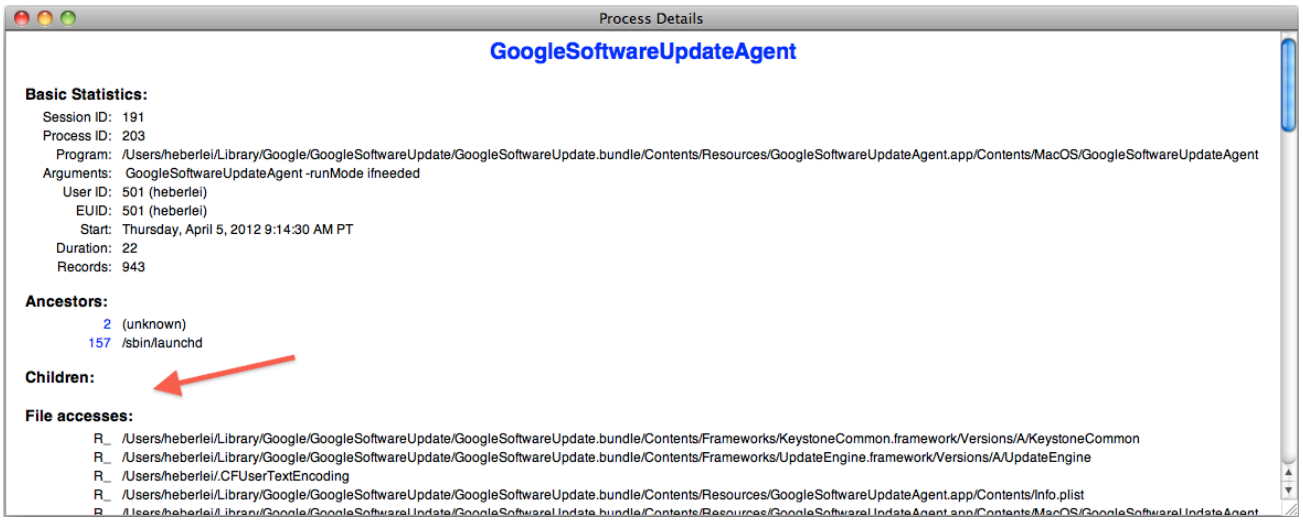

Figure 16

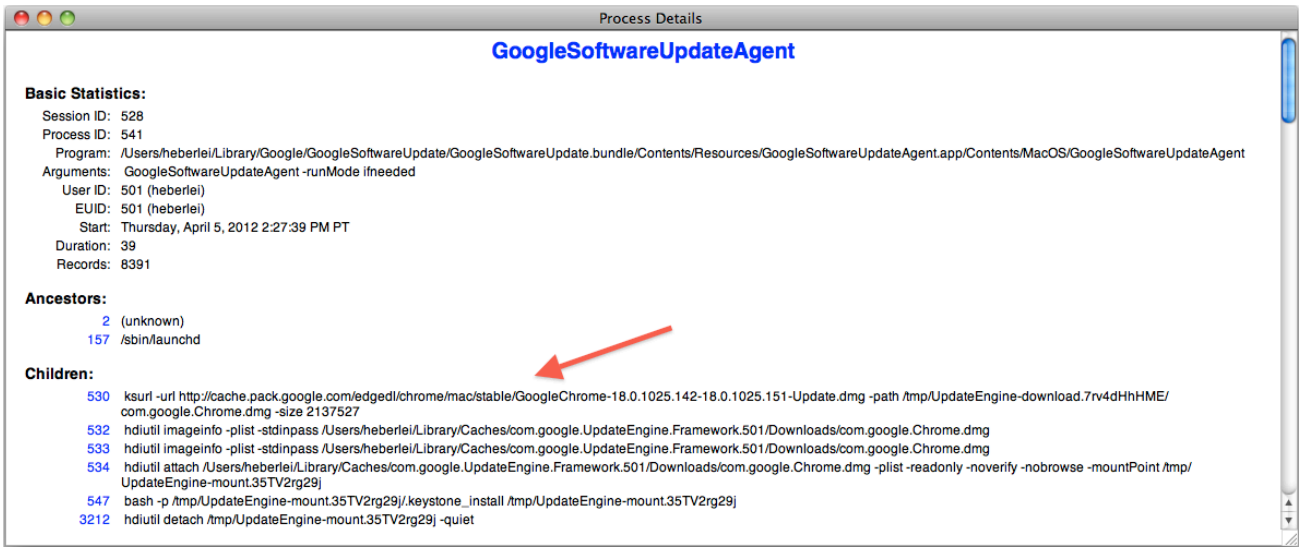

Figure 17

Figure 18 shows the portion of the graph from "The APT You Have" paper constructed from this data. The arguments passed to the children processes shown in Figure 17 provide clues at to what is happening. The ksurl process downloads a disk image, and then the first two hdiutil processes extract information about this disk image (using the imageinfo command with the -plist option). The third hdiutil attaches the disk image to the file system. .keystone_install is a bash shell script from this disk image which will start a large cascade of processes to be executed leading to a change to the system. Finally the last hdiutil command detaches the file system, thus getting rid of the evidence.

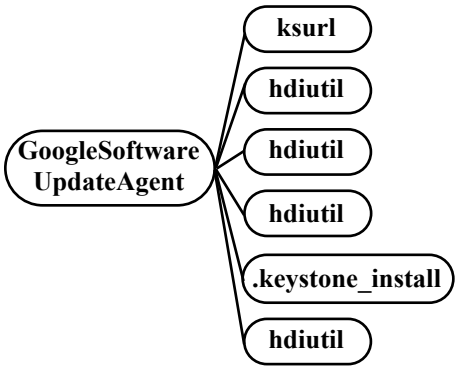

Figure 18

3.5 Modification With cp

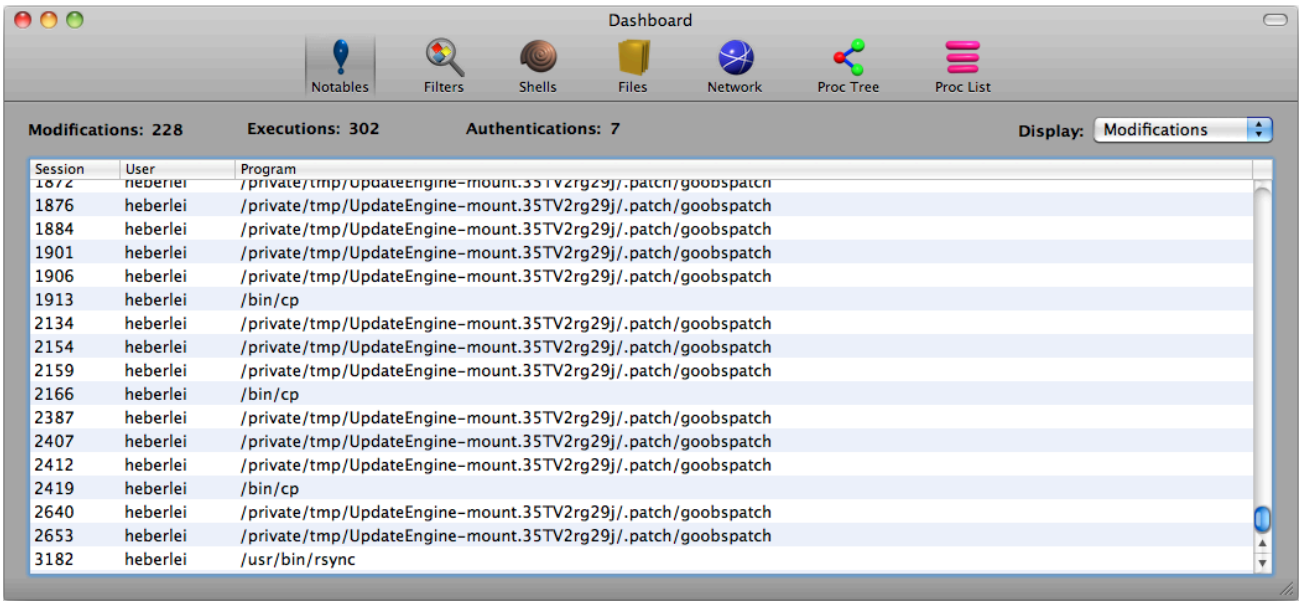

Figure 19

Returning to the "Notables" tab, the "Modifications" group shows a lengthy list of processes that made changes to critical programs, configuration files, and resource files. Changes to these programs and files can affect the integrity of the system. Changes can indicate that an attacker Trojaned programs, inserted backdoors, modified antivirus software, or performed other compromises to the system. This "change detection" strategy dates back to early versions of tools such as Tripwire and the DOE's Security Profile Inspector (SPI). These tools typically based their detection on changes to dates or checksums of the files. Audit Explorer bases its change detection on the actual operations that modify the files.

As mentioned in Section 3.2 with regards to the Notable Executions group, because of the volume of events that automated updates can generate, this detection strategy is being deprecated in favor of the

more expressive Filters mechanism. Still, it can be instructive to select a few events to understand what the process changed, how the process came to be, and where the data came from that was used to modify the system. As Figure 19 shows, Google's software update process generally uses a combination of cp, rsync, and goobspatch to modify the system. cp and sync are standard system programs, and no amount of analysis will identify these programs as malware. They are being used exactly as they were designed to be used. goobspatch, on the other hand, is a program that was downloaded over the Internet.

"The APT You Have" paper selected one of the copy events, cp, and constructed the process and data flow events associated with that event. This paper won't go through the full data flow (it is quite complex), but this paper shows another way to view the process creation flow.

In addition to "Process Details" window, Audit Explorer's "Proc Tree" tab provides another view of the events leading a process. Figure 20 shows the process tree leading to the cp process used in "The APT You Have" paper. This not only shows a process's ancestry, but it shows its immediate siblings, its parent's siblings, and so on. This provides insights as to how a single process, in this case the cp process, fits into a larger context.

For example, the process tree view shows dirpatcher.sh performed many operations, and a quick look can see a repeating patterns of "cp, stat, chmod, touch" (where this instance of cp came from) and "mkdir, goobspatch, stat, chmod". Zooming in on any of these processes will show the arguments passed to each command and thus provide incredible detail about what is happening at each step.

Figure 21 shows the portion of the graph from "The APT You Have" paper that represents the process tree shown in Figure 20.

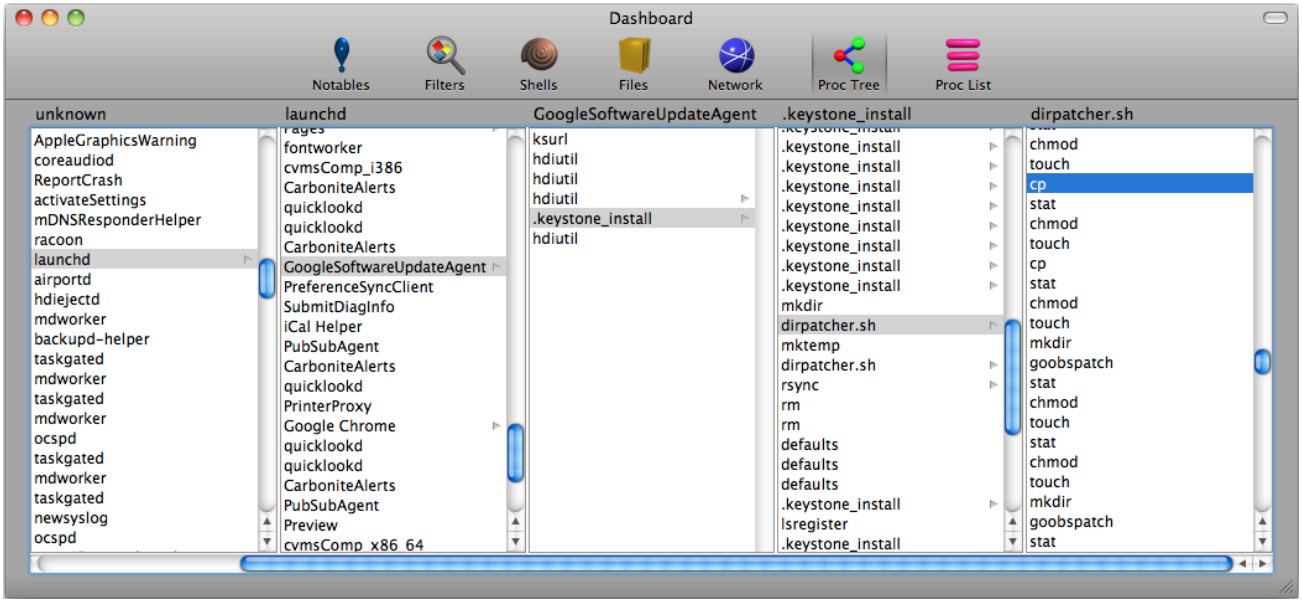

Figure 20

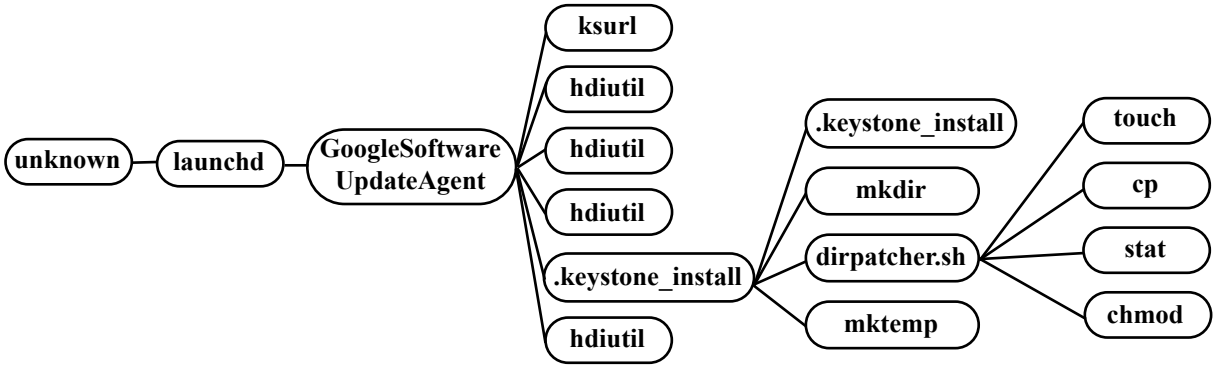

Figure 21

3.6 Application Executable Change

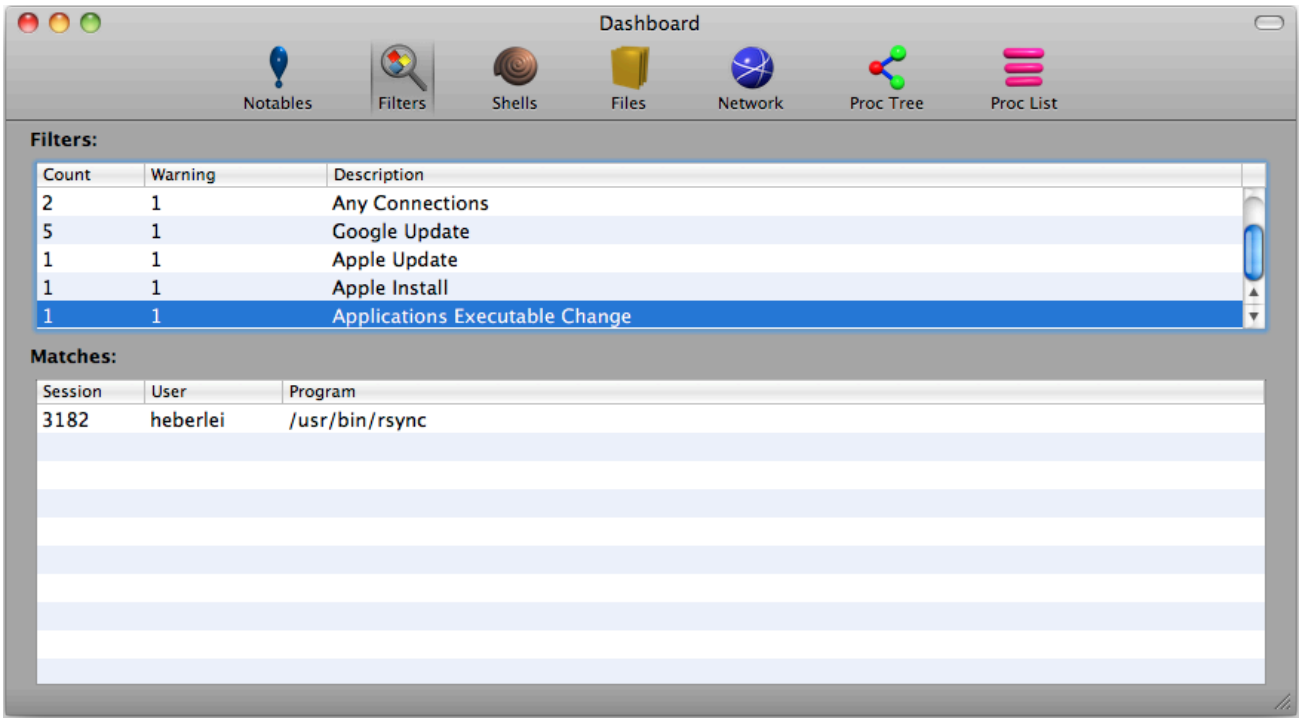

Figure 22

As mentioned in the previous section, the "Notable Modifications" mechanism can generate many alerts. Flagging any changes to files in the /Applications directory tree (part of your system) often triggers a large number of alerts each time an application is updated because many applications include a very large number of resource files (e.g., all the icons and images displayed at some point by the program). This is why Notable Modifications includes 228 changes when Google is updated. To address this, Audit Explorer's default filter excludes any changes that can trace its ancestry back to the GoogleSoftwareUpdateAgent.

However, knowing each time a program is changed can be informative. The custom rule "Applications Executable Change" looks for only changes to executable programs under the / Applications directory structure. In other words, it ignores all the changes to resource files such as the waitCursor.png file shown in "The APT You Have" paper.

Figure 22 shows one change was made to an executable, and it was made by the rsync program.

Figure 23 zooms in on the rsync process. The window shows several elements that helped build the graph in "The APT You Have" paper. The Ancestors list (label 1) shows the sequence of processes that eventually give rise to this rsync process. The File accesses list shows several instances of rsync opening a file read-write (RW) (the file name ending in a random sequence of letters and numbers) and then renaming the file. This is a common approach many programs use: write to a temporary file name and then, when the writing is done, rename the file to its final name. The third group is where the executable is changed (label 2). The initial temporary file name is ".Google Chrome.OQwVWD" then that file is renamed "Google Chrome". This is the change to the Chrome executable program – a program that users will enter account names and passwords for many web services, credit card numbers, and other sensitive data. Trojaning this executable can lead to security compromises across the Internet. Being able to detect a change to such an important file, determining the sequence of activity that led to that change, and knowing where the data came from that modified the program are critical capabilities an organization should have.

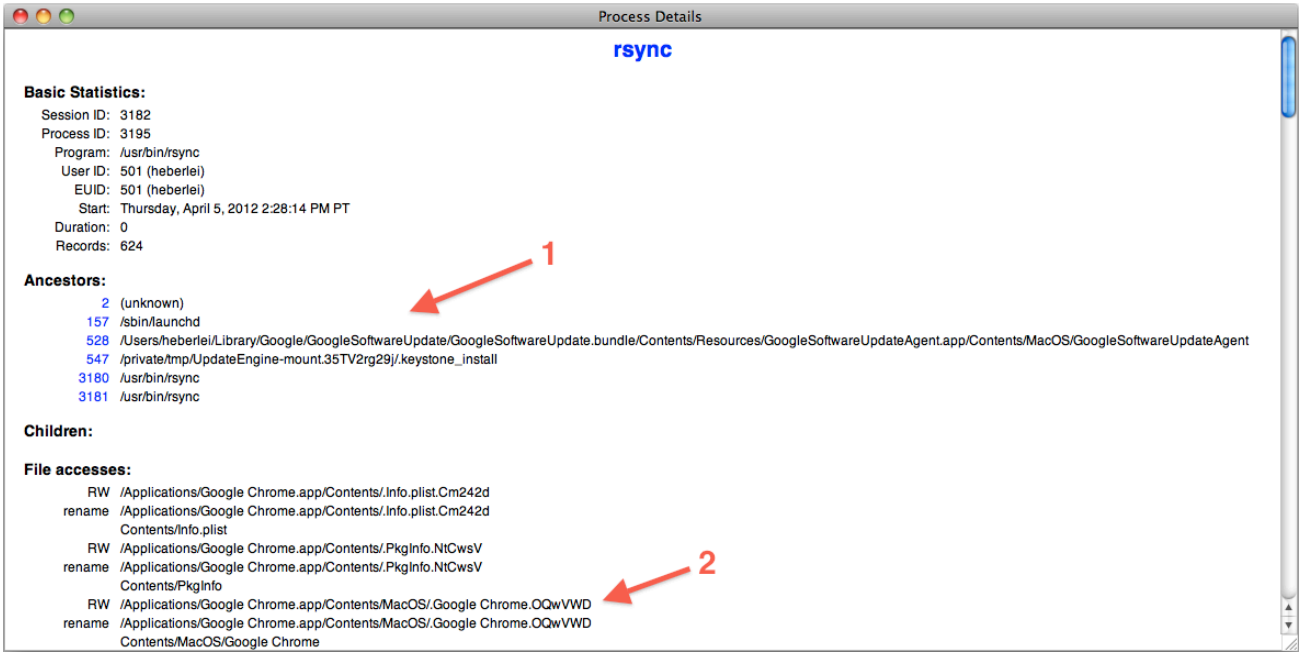

Figure 23

In addition to using the information in this one window to create part of Figure 1 in "The APT You Have" paper, the information in this one window was used to create Figure 3 in that paper (reproduced here as Figure 24). In many ways, this simple slice of control and data flow is much easier to understand: rsync changed the executable, and it can trace its ancestry back to the

GoogleSoftwareUpdateAgent. If you trust the GoogleSoftwareUpdateAgent, an analyst can probably feel comfortable trusting this change.

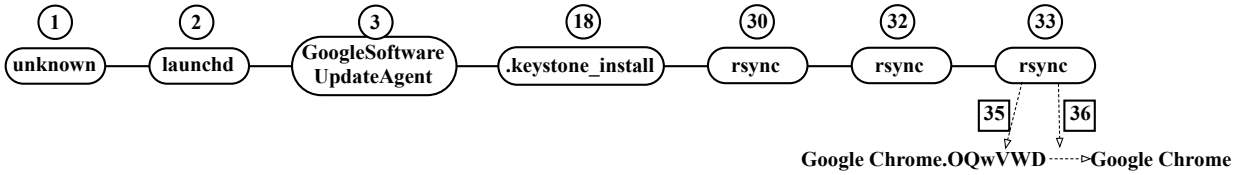

Figure 24

4 Conclusions

Google's software update process is a model Advanced Persistent Threat. While benign, it carries out many of the same tasks associated with APTs, including running a program in the background without the user's knowledge, downloading and executing new code/instructions from the Internet, and modifying executable code and resources for a critical application. Furthermore, Google Chrome is probably widespread within most organizations, and it updates frequently. This provides many opportunities for security administrators to test their skills detecting and analyzing activity common to APTs. The paper "The Advanced Persistent Threat You Have: Google Chrome" describes many of these steps, and at the core of that paper is a complex graph (Figure 1) showing all the steps needed to analyze the change of one resource file and one executable. This paper describes how Audit Explorer was used to extract the relevant information from the audit trail to build that graph.

Section 2 describes with the system, data, and initial analysis steps. The data comes from a production system, not a testbed system with just a few scripted background processes.

Section 3 is the core of the paper showing how many pieces of the graph were found and their connections to other pieces in the graph were determined. Our general approach is described as "find a thread" then "follow the thread". Audit Explorer, while not designed specifically to be an intrusion detection system, can highlight potentially suspicious actions; this is the "find a thread" step. Audit Explorer lets you zoom in on the process that created the suspicious event; this information is displayed in the "Process Details" window. This is where you begin to "follow the thread" (the forensics analysis) determining where the program came from, how it was started, what child processes it may have created, and what files or network servers it might have exchanged data with. This paper looked at three processes – ksurl, cp, and rsync – and show how different parts of Audit Explorer were used to reconstruct major pieces of the graph from "The APT You Have" paper.

Appendix A: Figure 1 from “The APT You Have” Paper