Unix systems monitoring with FCA

Uta Priss

Edinburgh Napier University, School of Computing,
www.upriss.org.uk

Abstract. There are many existing software tools for identifying specific and severe IT security threats (virus checkers, firewalls). But it is more difficult to detect less severe and more general problems, such as disclosure of sensitive or private data. In theory, security problems could be detected with existing tools, but the amount of information provided is often too overwhelming. FCA is a promising technology in this application area because it helps to reduce and explore data without prescribing what it is that is searched for from the start. This paper demonstrates the use of FCA for analysing Unix system data with respect to IT security monitoring.

1 Introduction

The background for this paper is IT security. Recently, two major international cyber attacks made for news headlines[1]. Most PC users will be aware of the need for virus checkers. But apart from the major risks from illegitimate software, legitimate software can also have unwanted side-effects either because it might disclose information to third parties without the user's knowledge or because it might contain faulty code. For example, a fair amount of personal detail might be stored by a web browser for the purpose of auto-completion of regularly used web forms. Presumably, users can protect themselves by switching on a "private mode" in their browser, but as discussed by Aggarwal et al. (2010), such private modes can never be completely effective because it is not easy to decide conclusively which data must be kept private and which not. Also, since web technologies are constantly evolving, there will probably always be some risks that can be exploited. Currently, many browsers leak information about the browsing history which can be accessed by other websites via "history stealing". Wondracek et al. (2010) claim that it may even be possible to use this to identify individual users of social networking websites.

Special purpose tools (such as browser plugins, virus checkers, and firewalls) may not be sufficient to detect all such problems because they tend to focus on specific, known problems. Another approach is to use monitoring tools which are more general purpose and aimed at detecting when something "odd" happens or, in other words, using a data mining approach for IT security. As usual in data mining applications, the data supplied (in this case by system commands and logfiles) can be overwhelming and difficult to analyse without the help of tools. In addition to the amount of information, system data often contains cryptic codes and abbreviations, which are more aimed at

[1] http://en.wikipedia.org/wiki/Operation_Aurora and http://en.wikipedia.org/wiki/Stuxnet

developers than users. In this paper, it is demonstrated how Formal Concept Analysis (FCA)[2] can be employed to assist users in making sense of system data. Interestingly, just the conceptualisation of the data already helps with the problem of understanding cryptic codes and abbreviations because codes become more meaningful if a user sees the usage context of a code. In the future, the use of FCA in this area could be supplemented with additional data mining tools, but even FCA alone already provides interesting results. So far there appear to be surprisingly few papers in the literature about FCA applications for IT security as summarised in Section 2.

There are many existing approaches to IT security which is a broad topic. In this paper, only Unix systems (such as Linux and Apple Macintosh OS X) are considered and the assumption is made that problem detection is conducted by using standard Unix commands for reading directory and process structures and logfiles. Most likely the FCA-based high-level technologies discussed in this paper are also applicable to Microsoft Windows. A Windows filesystem is accessible to a Linux partition on the same machine. Furthermore, even though the actual commands are different in Windows, the structures (using users, files and processes) are similar. But that is not further discussed in this paper.

There are numerous existing tools for monitoring operating systems and exploring logfiles[3]. But most of the existing tools only allow to monitor for known events or are very specialised and apply only to a specific type of logfile. Tools such as PandoraFMS[4] facilitate the simultaneous monitoring of many servers and many types of problems using sophisticated graphical reports. But users need to specify exactly what the software is supposed to monitor, in what manner and at what times. With respect to logfiles, there are, for example, many tools which allow to monitor and analyse web server logfiles[5] which record how many users have looked at the webpages, which countries they are from, and the dates and kinds of browsers used. Monitoring for web server errors with such tools is already slightly more difficult. The tools might list the most commonly produced errors and warnings of server-side scripts, but it may not be so easy to detect a hacking attempt against the server with such logfile analytic software, in particular if the hackers are using a novel method. There is, of course, also software for detecting hacking attempts, but such tools often focus on particular techniques (often very low-level[6]) and do not usually help users to explore the data in a more general manner.

It appears that so far a suitable, freely available, multi-purpose, high-level Unix systems monitoring tool does not yet exist. There are, however, existing tools that can be used as building blocks for such a tool (for example, logfile analytic tools, data mining tools and FCA tools). A modular design for such a tool is as follows:

1. data extraction (use existing tools for collecting data from the system and storing it in comma-separated files)

[2] This paper does not provide an introduction to FCA. Information about FCA can be found online (http://www.fcahome.org.uk) and in the main FCA textbook by Ganter & Wille (1999).

[3] Such as http://swatch.sourceforge.net/

[4] http://pandorafms.org

[5] Such as http://sourceforge.net/projects/awstats/ and http://www.webalizer.org/

[6] Such as http://www.snort.org/

2. context building (use pre-defined conceptual scales and heuristics for extracting formal contexts)
3. lattice representation (use existing FCA software for visualisation)
4. post-processing (summarise information from lattices in textual format that can be read by users who do not know FCA)

Since the tools for steps 1 and 3 already exist, the focus of this paper is on step 2. Step 4 is left for future research. It would be nice to have ways of summarising information from lattices to make such a tool available for general users. But for now, the focus is on expert Unix users. Presumably, learning to read FCA lattices is far easier than becoming an expert Unix user, therefore this should not be a big hurdle.

The main aim of this paper is the practical application of FCA to IT security. But there is also a theoretical contribution in the area of "Data Weeding" (Priss & Old, 2011), which is the art of selecting appropriate sets of objects and attributes from a complex, many-valued set of data. In general it is difficult to know in advance what lattice to construct for which data set. A similar problem was previously explored by Priss & Old (2010 and 2011) with respect to lexical databases, where neighbourhood lattices appeared to be the most useful structure. With respect to Unix data, the interaction between lattice structures and hierarchies (of files) and sequential (temporal) data is most interesting.

Section 2 of this paper provides an overview of existing IT security FCA research. Section 3 provides some background on temporal data and file hierarchies as used in Unix operating systems. Section 4 suggests five main modelling tasks in this area. Section 5 provides examples of what can be achieved. This is followed by a conclusion.

2 Published FCA research in the area of IT security

Since FCA is commonly used in the data mining area and since IT security is a commercially, legally and politically important application topic, it is surprising that there does not appear to be a greater amount of FCA research in this area. According to an article in the New York Times (Farley, 2006), FCA is used by the US National Security Agency for studying patterns in telephone networks, presumably to detect terrorist cells. Due to the secrecy of such agency no further details about this FCA application are known.

Maybe the first paper on IT-security and FCA was published by Becker et al. (2000). It describes a tool which models dependencies among security guidelines using the Toscana software. The underlying FCA methods used are the usual ones for many-valued contexts.

There are several papers which model the dependencies between software packages or code modules with FCA in order to detect security-related trends. These are based on a method first established in software engineering by Lindig & Snelting (1997). With respect to IT security, Neuhaus and Zimmermann (2009) use this method to trace software vulnerabilities in Linux packages. They model the partially ordered set of the dependencies as a concept lattice which is weighted by the known security risks associated with some packages. Using the lattice one can then determine the risks for any package and observe how certain packages are the main cause of the vulnerabilities.

Another IT security method based on Lindig & Snelting's approach is described by Ganapathy et al. (2007) for identifying security-sensitive operations in legacy code.

There are several papers on using FCA for web log analysis (Zhou[7] (2004) and Pohle & Spiliopoulou (2002)). As explained above, web log analysis is not necessarily aimed at IT security, but since security-related information is often derived from logfiles, any methods developed in this area are potentially relevant for security analysis as well.

3 Some Unix background on temporal data and file hierarchies

A Unix operating system provides many sources for detecting problems, often involving logfiles. Of particular interest is temporal data and file hierarchies. Detailed runtime information about processes is available. For each file and directory, the operating system stores when it was last accessed, modified or had its status changed. Temporal information on Unix, however, can be tampered with by users (for example, using the "touch" command). Thus, hackers can change file modification times in order to cover their tracks. But it would be difficult, even for hackers, to remove all traces of their actions. For Unix processes, temporal coincidence can be (but does not have to be) an indication of causal relationship. Files that have identical modification times are often created by a single process or related processes. For example, certain files are updated regularly at the same time when the computer boots up. Installing or updating a software package will lead to several files with the same modification times, although when files are downloaded as part of a software package they sometimes keep their modification time from when they were created on the original computer. In that case, the status change time will be more accurate in showing when the file was installed. Thus, interpreting the temporal information of files requires some knowledge of how times are affected by the operation system.

The file hierarchy in Unix tends to be mostly a tree hierarchy although there can be a few "symbolic links". In traditional Unix systems, the file hierarchy tended to be fairly simple and, ideally, files were placed in standard locations so that they could be found by users. In modern Unix systems, different strategies for file locations might be mixed. For example on Apple's OS X, some files are placed using traditional Unix directories (using lowercase letters), some files are placed in Apple's special hierarchy (starting with uppercase letters) and, if package management software is used, this might be placed in yet another location. Furthermore, the directory structures created by integrated development environments (IDEs) tend to be complex and not really human-readable. Therefore it is usually not possible to gather useful security-related information by manually looking at files or directories. Even if a search is performed for files that relate to a certain pattern, the information that is retrieved can be so complex that it cannot be processed manually.

In this paper, the main Unix commands considered are "find", "stat" and "lsof". These commands allow to select parts of the file hierarchy based on search criteria, to

[7] There is a paper published in 2009 in an India-Pakistan-based journal which has very similar content. Because of the dates I am assuming Zhou's work is original and the other one is plagiarised.

print file attributes and to see which files are opened by a running process. Although some of the options for these commands differ between different flavours of Unix the basic functionality should be the same for all modern versions. We have written a couple of very basic Perl scripts which convert output from these Unix commands into formal contexts. It is our intention to package the scripts as a toolkit with FcaStone and FcaFlint[8] once we have decided which context forming strategies appear to be most useful.

Apart from monitoring existing, standard information sources, another option is to deliberately collect information in order to detect possible problems. A reason for this is because files and directories change on a regular basis and even backups might not store relevant temporal and other attributes accurately. One possibility is to take static snapshots, for example of the original configuration of a newly installed operating system (running processes, top-level files, standard logfile entries), and to take further snapshots at regular timespans in order to detect changes. With respect to a newly installed piece of software, snapshots might be taken right before and right after the installation and while the software is running, although it may also be sufficient to conduct a search for files that were changed within the last couple of minutes right after the software has been installed. Another possibility is to use tracing software that records dynamic, runtime data. This is because security sensitive events might happen between snapshots and might not be recorded.

4 Modelling with FCA

Although, systems data can be collected in many different ways, the data tends to be of similar types. In particular, tree hierarchies and posets (for directory and process structures), temporal attributes and many-valued formal contexts (for users, processes, devices, files, etc) are of interest. This section discusses *five commonly occurring types* of structures in Unix data and how these can be modelled with FCA. A useful FCA notion for this discussion which may not be widely known is "contingent". An attribute contingent is the set of attributes in the intent of a concept which belong to the concept but do not belong to any superconcept. An object contingent is defined in the dual manner.

1) Temporal data: an interesting conceptual question is temporal chunking and granularity. Often files whose times differ by just a few seconds will have been created by the same process. In some cases, it might even be suitable to consider files that were created on the same day (during the same session) as being related. There are different possibilities: one can use some heuristic for determining chunks depending on the distribution of the temporal events; one can use FCA for determining the units by calculating a lattice of the raw temporal data and then using the contingents for determining temporal chunks; or one can use a simple strategy of ignoring units smaller than minutes, hours or days. Our experiments (in the next section) seem to indicate that the last choice (which is the easiest) is sufficient, but a user needs to decide which cut-off point (hours, minutes, seconds) is appropriate for which set of data. On a more abstract FCA

[8] http://fcastone.sourceforge.net/

level, the question is how a sequential structure (of temporal units) on the objects or attributes interacts with the conceptual structure of the lattice.

2) Many-valued formal contexts for logfiles and Unix command output: logfiles can have different syntactic structures, such as comma-delimited versus tab- or space-delimited, single-lined versus multi-lined entries, but these details can be dealt with through data-preprocessing. Otherwise, logfiles tend to be similarly structured: usually starting with a timestamp, standard attributes and a short, standardised description of an event. The output for the Unix commands "find", "lsof" and "stat" also is of similar nature. It can be assumed that at the conceptual modelling stage, the data is represented as a many-valued formal context. At least one attribute of such a context tends to be temporal. The others tend to be from a standard set of attributes (users, processes, paths, etc). In some cases one further attribute contains a brief message describing the event. Standard FCA techniques for modelling many-valued contexts exist and need not be further elaborated in this paper.

3) Observing the same data at different times: if snapshots of system data are taken as described in the previous section, formal contexts might be formed which contain the same set as objects and attributes but each belonging to a different snapshot. If data from different snapshots is combined in one lattice, deviations from symmetry could indicate changes. Alternatively a separate lattice could be constructed for each snapshot in which case algorithms would need to be employed that compare lattices. This is interesting, but not further elaborated in this paper.

4) A poset of the file hierarchy: apart from temporal information, which is sequential and can be chunked, the other important re-occurring data structure is the poset formed by the file and directory hierarchy. As discussed in the previous section, Unix file hierarchies tend to be too complex to be manually examined in detail. FCA can help reducing and structuring file hierarchy information. An interesting theoretical question arises as to how the poset on the objects or attributes interacts with the lattice structure. A concept lattice might be used to simplify the poset of the file hierarchy as is demonstrated in the next section.

5) Conceptual scales for recording security information: one problem for IT security is the amount of misinformation that is posted on the web because there is no quality control for web content (e.g. web forum discussions, deliberate misinformation by some companies), nor do search engines consider content quality for their rankings. For example, one can type any process name from the Windows Task Manager into a search engine and will instantly retrieve webpages which claim that this process might be a virus even though in most cases it is completely harmless. In a similar manner to how virus scanners download virus definition files, it might be possible to create conceptual scales which are manually verified by experts and which contain information about normal structures in operation systems which can be used as a comparison to actually occurring structures so that users can check whether something they noticed on their computer is normal or suspicious. This would be a complex task and is beyond the scope of this paper.

5 Four examples of exploring Unix system data with FCA

This section explores several examples of concept lattices in the area of Unix systems data monitoring. The examples below are a first attempt at determining heuristics for context construction in this domain. The examples are all generated by using short scripts that process the output of Unix commands or logfiles on an Apple OS X computer. FcaStone has then been used to produce the pictures, which have not been manually edited. The diagrams use "minimal labelling" which means that objects (in the bottom half of the concept boxes) also belong to their superconcepts; attributes also belong to their subconcepts. Or, in other words, only the contingents are written for each concept. Once we have explored a sufficient number of examples, we intend to package the scripts into a toolkit. The data has not otherwise been modified with the exception that some of the terms have been abbreviated in order to make the figures more readable in a printed medium.

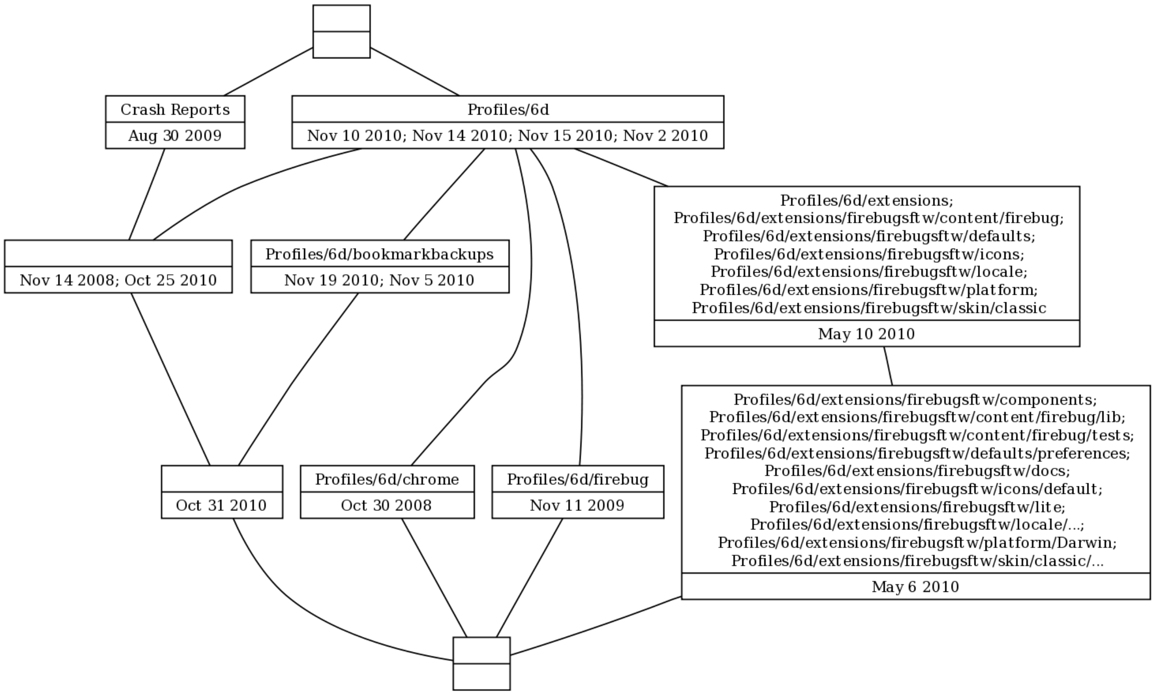

Crash Reports
Aug 30 2009
Profiles/6d
Nov 10 2010; Nov 14 2010; Nov 15 2010; Nov 2 2010
Profiles/6d/extensions;
Profiles/6d/extensions/firebugsftw/content/firebug;
Profiles/6d/extensions/firebugsftw/defaults;
Profiles/6d/extensions/firebugsftw/icons;
Profiles/6d/extensions/firebugsftw/locale;
Profiles/6d/extensions/firebugsftw/platform;
Profiles/6d/extensions/firebugsftw/skin/classic
May 10 2010
Nov 14 2008; Oct 25 2010
Profiles/6d/bookmarkbackups
Nov 19 2010; Nov 5 2010
Profiles/6d/extensions/firebugsftw/components;
Profiles/6d/extensions/firebugsftw/content/firebug/lib;
Profiles/6d/extensions/firebugsftw/content/firebug/tests;
Profiles/6d/extensions/firebugsftw/defaults/preferences;
Profiles/6d/extensions/firebugsftw/docs;
Profiles/6d/extensions/firebugsftw/icons/default;
Profiles/6d/extensions/firebugsftw/lite;
Profiles/6d/extensions/firebugsftw/locale/...;
Profiles/6d/extensions/firebugsftw/platform/Darwin;
Profiles/6d/extensions/firebugsftw/skin/classic/...
May 6 2010
Oct 31 2010
Profiles/6d/chrome
Oct 30 2008
Profiles/6d/firebug
Nov 11 2009

Fig. 1. The files in the Firefox library directory

The first example combines temporal data (file modification times) and file hierarchy data. The data was collected from the Firefox library directory by listing all files in the subdirectories together with their full path names (as formal attributes) and their file modification times (as formal objects). The relationship between objects and attributes was defined as “at that date this file or a file below this directory was modified”. This means that for each full path name all the superdirectories were determined and added to the set of attributes as well. A lattice created directly from this data would be too big to visualise. A bit of experimentation revealed that the lattice could be reduced to a more convenient size (Figure 1) by considering only days and ignoring hours, minutes and seconds for the objects and by ignoring the bottom-level file/directory for each attribute. (Day-level chunking is not a limitation. Even on a larger server, some parts of the file hierarchy will not get modified after installation or update. This makes a reduction to days feasible. Obviously, there will be other parts of the file hierarchy where day-level chunking is not possible.)

In order to make the lattice more readable a few further reductions were applied to the labels. The formal objects are “restricted” (Priss & Old, 2011) because only dates which belong to at least two files are included. This does, in this case, not change the lattice structure but reduces the number of object labels. Furthermore, the labelling of the attributes was reduced by using the posets of the file hierarchy as follows: for each contingent, the poset was reduced by listing only the top and bottom elements. If all files or subdirectories under a directory belong to the same contingent, this was abbreviated by writing “/...” at the end of the directory name and omitting the files or subdirectories.

In this case the attribute order in the lattice is a tree itself and does not contradict the file hierarchy. For example, none of the concepts jointly under the distinct directories “Crash Reports” and “Profiles/6d” have any attributes in their contingent. One could reduce the path names further by omitting the directories of higher level concepts, but we do not think this will be applicable for many examples and decided against using it.

Considering that the complete dataset of this example contains 62 directories and 477 files, Figure 1 provides a human-readable overview of the data which might not easily be obtained without FCA. Figure 2 demonstrates the effect the contingent-based reduction has on the file hierarchy. On the left is the complete hierarchy (as produced by the Unix “tree” command). On the right is the hierarchy of the attribute set from Figure 1. The reduced hierarchy is focused on the actual events of when files were created or changed.

An expert Unix user can now interpret the data of Figure 1. Crash reports can occur at any times. The dates in the contingent of Profiles and bookmarkbackups tend to be recent. Backups are automatically created and deleted on a regular schedule. The browser is configured to delete cookies and similar data when the browser closes. Therefore certain backup and profile files will be modified on a regular basis. Three dates relate to the firebug extension, presumably representing the times when that extension was installed or updated and last used.

The most interesting date is November 14, 2008 because this date is both under Crash Reports and Profiles in an area where all other dates are recent. In order to understand the significance of this date, an event-driven lattice needs to be constructed. This lattice consists of all files (on the whole computer, not just in this directory) that were modified on November 14, 2008 and their exact modification times. To save space, the lattice is not shown in this paper, because this method is demonstrated for another example below. The event-driven lattice for November 14, 2008 explains that Firefox was installed for the first time on that day. It highlights which other directories belong to a Firefox installation. It might seem contradictory that the date of the chrome[9] directory is earlier (October 2008) but that is because when files are unzipped they keep their modification times from when they were created in the original location. A check revealed that the status change time of the chrome directory is also November 14.

The event-driven lattice almost allows to write a personal diary of the user of that computer for that day showing that the day was started with preparing teaching materials, followed by software installation. One could probably pinpoint the time a lunch break was taken on that day. The amount of historical detail revealed is quite surprising.

[9] Incidentally this is a good example of how less experienced users could be confused by internal file names. The word "chrome" here does not mean that Google's Chrome browser is installed but is a name that is used by Firefox for certain settings.

```
|-- Crash Reports
`-- Profiles
    `-- 6d
        |-- bookmarkbackups
        |-- chrome
        |-- extensions
        |   `-- firebugsftw
        |       |-- components
        |       |-- content
        |       |   `-- firebug
        |       |       |-- lib
        |       |       `-- tests
        |       |-- defaults
        |       |   `-- preferences
        |       |-- docs
        |       |-- icons
        |       |   `-- default
        |       |-- lite
        |       |-- locale
        |       |   |-- bg-BG
        |       |   |-- ca-AD
        |       |   |-- cs-CZ
        |       |   |-- da-DK
        |       |   |-- de-DE
        |       |   |-- el-GR
        |       |   |-- en-US
        |       |   |-- es-AR
        |       |   |-- es-ES
        |       |   |-- fa-IR
        |       |   |-- fr-FR
        |       |   |-- hr-HR
        |       |   |-- hu-HU
        |       |   |-- hy-AM
        |       |   |-- is-IS
        |       |   |-- it-IT
        |       |   |-- ja-JP
        |       |   |-- ko-KR
        |       |   |-- nl-NL
        |       |   |-- pl-PL
        |       |   |-- pt-BR
        |       |   |-- pt-PT
        |       |   |-- ro-RO
        |       |   |-- ru-RU
        |       |   |-- sk-SK
        |       |   |-- sl-SI
        |       |   |-- sv-SE
        |       |   |-- tr-TR
        |       |   |-- uk-UA
        |       |   |-- vi-VN
        |       |   |-- zh-CN
        |       |   `-- zh-TW
        |       |-- platform
        |       |   `-- Darwin
        |       `-- skin
        |           `-- classic
        |               |-- breakOn
        |               |-- mac
        |               |-- trace
        |               `-- win
        |-- firebug
        |-- minidumps
        `-- searchplugins
```

```
|-- Crash Reports
`-- Profiles/6d
    |-- bookmarkbackups
    |-- chrome
    |-- extensions
    |   `-- firebugsftw
    |       |-- components
    |       |-- content/firebug
    |       |   |-- lib
    |       |   `-- tests
    |       |-- defaults
    |       |-- docs
    |       |-- icons
    |       |   `-- default
    |       |-- lite
    |       |-- locale
    |       |   `-- ...
    |       |-- preferences
    |       |-- platform
    |       |   `-- Darwin
    |       `-- skin/classic
    |           `-- ...
    `-- firebug
```

Fig. 2. The FCA based reduction of a file hierarchy

But because the topic of this paper is security, Figure 3 shows an event-driven lattice for another example. In this case an experiment was conducted. The Firefox browser was opened and a video on a news website was watched; then the browser was closed. Using Unix “find”, all files in the user’s Library directory were retrieved which had been accessed during the last 10 minutes. A lattice was constructed of the files and their exact times (ignoring seconds). The attribute contingents were reduced in the same manner as in Figure 1. This lattice shows two directories which are used for storing Adobe Flash Cookies and which were accessed at the same time as Firefox’s cache. Since all other files and directories in this lattice either relate directly to Firefox or are very general (such as Fonts), this lattice would alert a user to the existence of Flash Cookies, which are different from ordinary web cookies and can be used to create cookies which are undeletable for ordinary users [10].

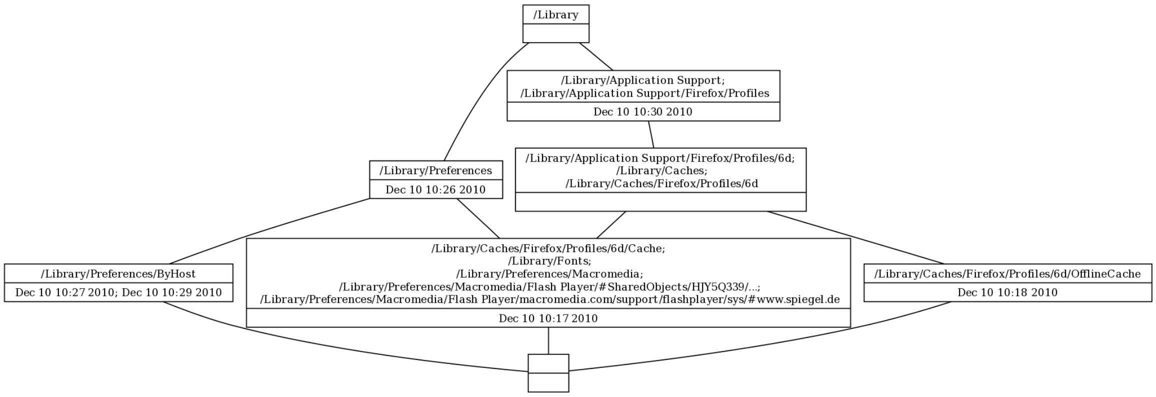

/Library
/Library/Application Support;
/Library/Application Support/Firefox/Profiles
Dec 10 10:30 2010
/Library/Preferences
Dec 10 10:26 2010
/Library/Application Support/Firefox/Profiles/6d;
/Library/Caches;
/Library/Caches/Firefox/Profiles/6d
/Library/Preferences/ByHost
Dec 10 10:27 2010; Dec 10 10:29 2010
/Library/Caches/Firefox/Profiles/6d/Cache;
/Library/Fonts;
/Library/Preferences/Macromedia;
/Library/Preferences/Macromedia/Flash Player/#SharedObjects/HJY5Q339/...;
/Library/Preferences/Macromedia/Flash Player/macromedia.com/support/flashplayer/sys/#www.spiegel.de
Dec 10 10:17 2010
/Library/Caches/Firefox/Profiles/6d/OfflineCache
Dec 10 10:18 2010

Fig. 3. An event-driven lattice

The first example shows that it is possible to retrieve interesting information about software activity (and user activity) by analysing temporal attributes of files and directories. The second example demonstrates how monitoring of system data might alert users to security and privacy issues which are not visible through normal application software (current browsers do not let users explore Flash Cookies).

So far the examples started either with a directory (Figure 1) or a time unit (Figure 3) and used historical data. In order to establish a more general overview of currently active processes, the “list of open files” (lsof) command is useful. Unfortunately, the complete output of that command is too complex to be visualised as a lattice. Figure 4 shows an example of the complete data set of user-owned processes, but restricted to showing only the top two directories of each full pathname. The formal objects are names of processes; the attributes are the files (libraries) that are opened by the processes. The user had two browsers open (Firefox and Safari), was looking at some image or pdf file (Preview), executed the lsof command on the command-line (Termi-

[10] As demonstrated by Samy Kamkar in his “Evercookie”http://samy.pl/evercookie/

nal, tcsh) and was editing a Latex file (TeXShop). The overview shows that distinct types of processes are running: there are traditional Unix commands (tcsh, lsof) which do not use Apple's special libraries. Apple-specific commands can be divided into Applications that were started by a user (under /Applications) and programs that run all the time (under /System, but not under /Applications). The only oddity in the lattice is that TeXShop is not connected to /Users/upriss (in contrast to Safari and Firefox) even though TeXShop is used to edit files under that directory. An explanation might be that TeXShop does not keep these files permanently open. This is a general problem with using lsof: events that happen very quickly, such as writing to a file or downloading content to a web browser, will only be listed by lsof if they happen at that very instant in time. If one needs to be certain to catch all such events, one needs tracing software instead of lsof.

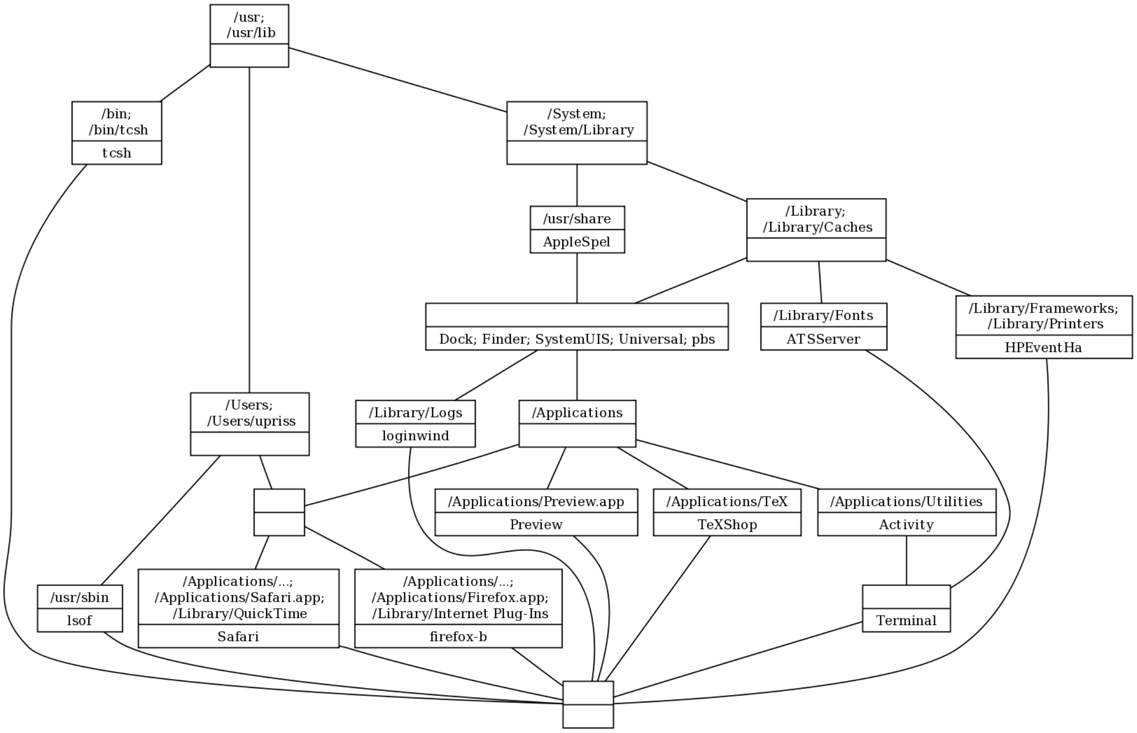


Fig. 4. List of open files

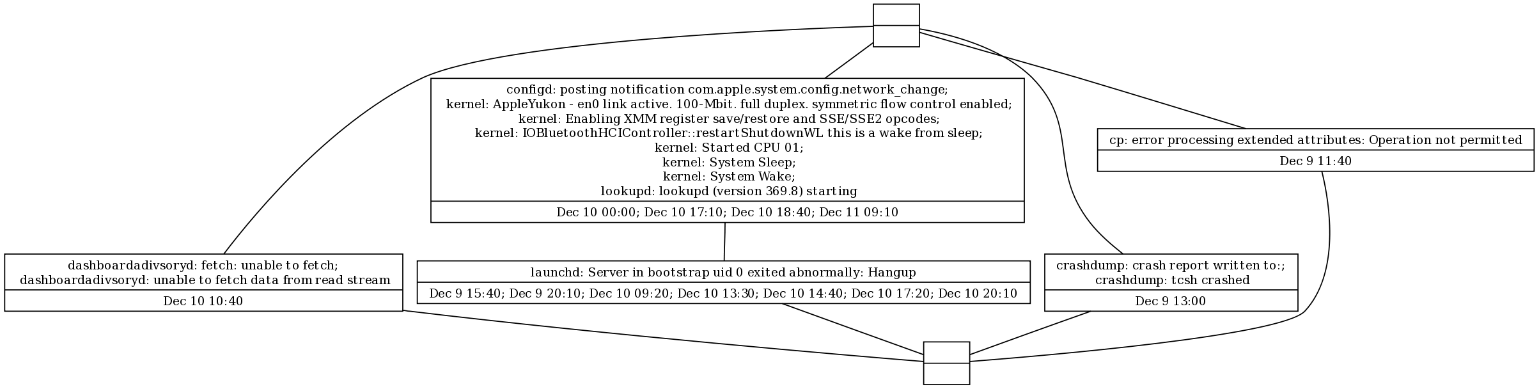
The lattice does not permit to detect any security problems, but then presumably there are no viruses or other rogue processes running on this computer. Apart from detecting rogue processes, another possible security application of lsof data is similar to Neuhaus & Zimmermann's (2009) lattice of software vulnerability dependencies. Neuhaus & Zimmermann determine the risk associated with each program based on software dependencies. A lattice of lsof data shows the same dependencies but with respect to actually running processes. Of course, in order to investigate the relevant libraries the full data would need to be used instead of using just the top two directories. A lattice of the complete data set is too complex, but if one constructs sublattices for the tree under each top-level directory at a time (/Library, /Applications, /Users) or for subsets of the objects, it becomes manageable.

The final example shows a lattice of the system logfile. In this case the preprocessing consisted of omitting seconds and rounding minutes down to multiples of 10. Furthermore, there are several messages in the file which re-occur frequently but with changing process id numbers. These process id numbers were removed. The resulting lattice is shown in Figure 5. In the system logfile, activities relating to a single event can be spread across several lines. But because they have the same timestamp, they are automatically identified as a single event. In theory it could happen that an event was started a second before the full hour and finishes a second after the full hour. In that case the event would be split but unless this happens often, this would be obvious from the lattice. Figure 5 shows that regular events are automatically grouped. The largest contingent in the lattice refers to the events that follow a wake-up from sleep. In that case the launch daemon needs to be restarted. The subconcept shows that the launch daemon also restarts at other times. The other three concepts refer to one-time events: a crash, a dashboard error and a copy error. There were no security problems detected, but since one-time events are highlighted by the lattice, any new or unusual activity would be visible. A further strategy would be to make a regular copy of these lattices and to compare them occasionally to see whether anything has changed.

6 Conclusion

This paper investigates the use of FCA lattices for analysing Unix system data. The experimental results so far are promising. On a theoretical level, the paper shows that the relationship between lattices and other structures (sequential data and other posets) is of interest. The lattice structure can be used to modify the other structures (for example reducing the complexity of file hierarchies based on FCA contingents). With respect to temporal data, this paper suggests that the easiest approach of not using the lattice structure but simply chunking the time units according to some threshold is sufficient for these applications. It might be of interest, however, to consider methods of Temporal Concept Analysis (Wolff, 2002) in the future.

The paper shows that it is possible to produce lattices that summarise the information and allow data exploration. The challenge lies in finding an appropriate “Data Weeding” (Priss & Old, 2011) technique, i.e., determining which heuristics of object and attribute selection are appropriate. A user still needs to experiment with each new data set. One strategy would be to start with a lattice of the full data. If that is too large, the temporal units could be made larger (ignoring seconds, etc) and the file hierarchy can be reduced (omitting top or bottom levels or parts of the tree). In that manner a user could alternate between making choices and viewing the lattice until the complexity has been sufficiently reduced. One potential application for this might be police investigations of computer harddrives. Currently, it takes the police in Scotland up to three years



```
configd: posting notification com.apple.system.config.network_change;
kernel: AppleYukon - en0 link active. 100-Mbit. full duplex. symmetric flow control enabled;
kernel: Enabling XMM register save/restore and SSE/SSE2 opcodes;
kernel: IOBluetoothHCIController::restartShutdownWL this is a wake from sleep;
kernel: Started CPU 01;
kernel: System Sleep;
kernel: System Wake;
lookupd: lookupd (version 369.8) starting
Dec 10 00:00; Dec 10 17:10; Dec 10 18:40; Dec 11 09:10

cp: error processing extended attributes: Operation not permitted
Dec 9 11:40

dashboardadivsoryd: fetch: unable to fetch;
dashboardadivsoryd: unable to fetch data from read stream
Dec 10 10:40

launchd: Server in bootstrap uid 0 exited abnormally: Hangup
Dec 9 15:40; Dec 9 20:10; Dec 10 09:20; Dec 10 13:30; Dec 10 14:40; Dec 10 17:20; Dec 10 20:10

crashdump: crash report written to:;
crashdump: tcsh crashed
Dec 9 13:00
```

Fig. 5. System log

to investigate a harddrive that has been seized because of suspected criminal files on the computer. An FCA-based tool might help a police officer to investigate and process such harddrives faster. The computer could be booted from a USB Linux drive. The file hierarchy data could then be downloaded and stored in a database and processed as described in Figures 1 and 3.

Apart from a GUI interface that would be required, more pre-processing would be useful. For example, as discussed for the system log, if a logfile contains a re-occurring message, each time with a different identifier, the identifier might need to be omitted. Thus, some heuristic parsing software would be useful. We intend to continue with the experimentation with more types of data (for example, firewall and networking data) and on a variety of different flavours of Unix in order to develop a stable set of heuristics which will then be implemented as a toolkit. We also intend to investigate existing software for logfile processing and data mining in more detail in order to determine whether there are existing tools that can be combined with our methods.

References

1. Aggarwal, Gaurav; Bursztein, Elie; Jackson, Collin; Boneh, Dan (2010). *An Analysis of Private Browsing Modes in Modern Browsers.* Proceedings of Usenix Security,
2. Baoyao, Zhou (2004). *Intelligent Web Usage Mining.*
3. Becker, Klaus; Stumme, Gerd; Wille, Rudolf; Wille, Uta; Zickwolff, Monika (2000). *Conceptual Information Systems Discussed Through an IT-Security Tool.* Proceedings of the 12th European Workshop on Knowledge Acquisition, Modeling and Management, Springer, LNCS 1937.
4. Farley, Jonathan David (2006). *The N.S.A.'s Math Problem.* The New York Times, May 16.
5. Ganapathy, Vinod; King, David; Jaeger, Trent; Jha, Somesh (2007) *Mining Security-Sensitive Operations in Legacy Code using Concept Analysis.* Proceedings of the 29th International Conference on Software Engineering.
6. Lindig, Christian; Snelting, Gregor (1997). *Assessing Modular Structure of Legacy Code Based on Mathematical Concept Analysis.* International Conference on Software Engineering, p. 349-359.
7. Neuhaus, Stephan; Zimmermann, Thomas (2009). *The Beauty and the Beast: Vulnerabilities in Red Hat's Packages.* Proceedings of the 2009 USENIX Annual Technical Conference.
8. Pohle, Carsten; Spiliopoulou, Myra (2002). *Building and Exploiting Ad Hoc Concept Hierarchies for Web Log Analysis.* Proceedings of the 4th International Conference on Data Warehousing and Knowledge Discovery, Springer, LNCS 2454.
9. Priss, Uta; Old, L. John (2010). *Concept Neighbourhoods in Lexical Databases.* In: Kwuida; Sertkaya (eds.), Proceedings of the 8th International Conference on Formal Concept Analysis, ICFCA'10, Springer Verlag, LNCS 5986, p. 283-295.
10. Priss, Uta; Old, L. John (2011). *Data Weeding Techniques Applied to Roget's Thesaurus.* In: Knowledge Processing in Practice. Springer Verlag, LNAI 6581, p. 150-163.
11. Ganter, Bernhard, & Wille, Rudolf (1999). *Formal Concept Analysis. Mathematical Foundations.* Berlin-Heidelberg-New York: Springer.
12. Wondracek, Gilbert; Holz, Thorsten; Kirda, Engin; Kruegel, Christopher (2010). *A Practical Attack to De-Anonymize Social Network Users.* In: 2010 IEEE Symposium on Security and Privacy, p. 223-238.
13. Wolff, Karl Erich (2002). *Interpretation of Automata in Temporal Concept Analysis.* In: U. Priss, D. Corbett, G. Angelova (eds.): Integration and Interfaces. Tenth International Conference on Conceptual Structures LNAI, 2393, p. 341-353.