Kaleidoscope: A Collaborative File Analysis Tool

Ryan A. Mast
rmast@cs.ucsd.edu

Abstract

Kaleidoscope is a web-based tool designed to facilitate collaborative analysis of file contents by a group of individuals. It aims to provide a simple interface to allow quickly exploring what the contents of a file look like when viewed in different ways (with different views dubbed lenses). A set of functions and format is provided to make the development of new lenses for viewing additional data types or data structures simple yet flexible, with the ability to implement lenses that do not need to conform to a fixed output format, such as being able to interpret and display a portion of a file as image data or a series of audio samples. Furthermore, Kaleidoscope provides the ability for users to collaborate in the process of annotating the data contained in a file as they discover what the data represents, along with the ability to discuss what different pieces of data are with new comments and changes to the format of the data being viewed showing up in real-time, so that multiple users can work together to discover the structure of a file.

1 Introduction

Within the sysnet group at UCSD, there are a number of computer security related projects that involve reverse engineering files, and having multiple people look at the same file in parallel is difficult.

One of the challenges we have encountered in analyzing files with is the difficulty of sharing information on the structure of the file with a group of individuals. Even more so than just sharing information, working with a group to reverse engineer a file at the same time is cumbersome at best with the current set of tools available, as there are no general purpose tools for reverse engineering file contents that support collaboration out of the box. The best of tools have some form of version control system in place that allows users to commit a new version and send people a new link with their changes.

Other times we are forced to use SVN for version control of binary files, a use for which it is not ideal. Since most of the files that store annotations are not text files, SVN is unable to merge them together. That means that when someone wants to work on a file, they need to get a lock on the file so that another person will not make a change that would result in a conflicting version when they go to commit their own changes. As a result, there can only be one person making annotations at a time, which is far from ideal.

As a result, we have decided to create a tool that we can use for collaborative file analysis. Our goal was to create a tool that is flexible enough that it can be extended to be useful in analyzing a wide variety of file formats, and to encourage collaboration by letting multiple users work on analyzing the same file simultaneously.

2 Related Work

Current tools for analyzing a file are designed with a single user in mind and do not provide a simple means of collaborating with other users. Furthermore, these tools are often purpose-built with a single type of file in mind rather than investigating the contents of an unknown file type. As an example, there are numerous tools to look at the structure of image file formats (such as JPEG, PNG, and GIF images), PE headers, and ROMs for various systems, yet all of them have are designed to only inspect the contents of a single type of file whose format has already been documented.

For more general purpose discovery of the structure of a file, the most common tool used is a hex editor (of which both web and desktop apps exist). The majority of these are limited in the way the data can be viewed and annotated, and even the best of them do not provide a simple way for multiple users to annotate the same file together in real-time. They require an external means of communicating and sharing the files with their annotations such as lengthy exchanges of emails or committing

and checking out annotation files using a version control system.

3 Design Evolution

From the start, one of the goals for Kaleidoscope was that collaboration to analyze a file should be easy. This meant that when inviting another user to work with you, they should not have to jump through hoops to install a particular program, get plugins set up and working, or need to send files or new links back and forth every time a change is made. Part of that includes being able to run Kaleidoscope on a wide range of platforms including Windows, Mac OS X, and Linux is important. Therefore, the choice of making a web app was clear. Users can simply upload the file and begin annotating it, then to invite another person to collaborate they send the other person a link; there is no need to email files back and forth, or send them an updated link every time a change gets made. Both users can also be working in parallel to improve efficiency and reduce the amount of duplicated work.

3.1 Initial Design

The initial prototype for Kaleidoscope was based on the idea that users should be able to show the structure of a file in whatever way they chose. The first implementation focused on this aspect of the design by letting a user freely add and delete sections that spanned a particular section of a file and display that section in various ways. Each section could then be freely moved around to wherever the user felt it fit best to show the content of the file.

This design did not fit the primary purpose of Kaleidoscope, which is to analyze the contents of a file, not alter its contents. The ability to add and remove sections and move them around freely was not intuitive because the contents of the file remain in a static order and are immutable. Allowing users to add and delete sections did not fit in with the primary goal of discovering what different portions of data within a file represent, and commenting on a section that could later be deleted did not make sense.

3.2 Splitting and Merging

As a result of the initial prototype, the design was changed to better capture the fact that users are discovering the structure of a file, not altering it. This led to the paradigm of splitting an merging portions of a file based on the data contained in a particular region. This proved to be a more intuitive way of looking at looking at the contents of a file, and is an easier concept for users to understand.

For example, many file types have header and data sections. This separation of header and data sections is much more elegantly captured by the notion of splitting a file into different sections rather than adding sections. Furthermore, the concept of sections of a file can be taken a step further by breaking down the contents of the header into individual fields based on the type of data contained in them. This allows users analyzing a file a high level of granularity with which they can look at the sections of a file.

3.3 Comments

Comments were then tied to byte offsets within a file so that users are able to make notes as they go. Oftentimes the process of discovering what the data in a file means happens in a non-linear fashion, with a user discovering what one thing piece of data represents only after finding out what another piece of data is for. By associating comments with a byte offset and providing a discussion thread, users are able to keep notes on what they believe a piece of data represents, and then refine their ideas about that data as they continue to analyze the rest of the file. Of course, sometimes the initial idea is correct and future ideas deviate; maintaining a discussion history allows a user or their collaborators to look back and see how and why initial ideas changed so that they are able to ensure that they are not straying from an early assumption that was initially correct.

Finally, this entire notion of viewing data in different ways led to the notion of a plugin system called lenses that can be used to easily change the way data in a section is displayed. As the number of different file types and data formats is innumerable, it would be impossible to provide a single set of lenses that could fulfill every need. As such, we provide a straightforward API for developing new lenses that can display a section in a custom way. This could range from anything as simple as handling a new data type, to showing a known c struct, to displaying part of a file as an image or playing the data contained within as audio.

A side benefit of the ease with which a lens can be developed is that once a file has been broken down into highly granular sections, a lens could be written for the entire file header, and then the sections could be merged back together and displayed with the custom lens. And if the user wants to see the data in a different way, it is extremely simple for them to switch lenses.

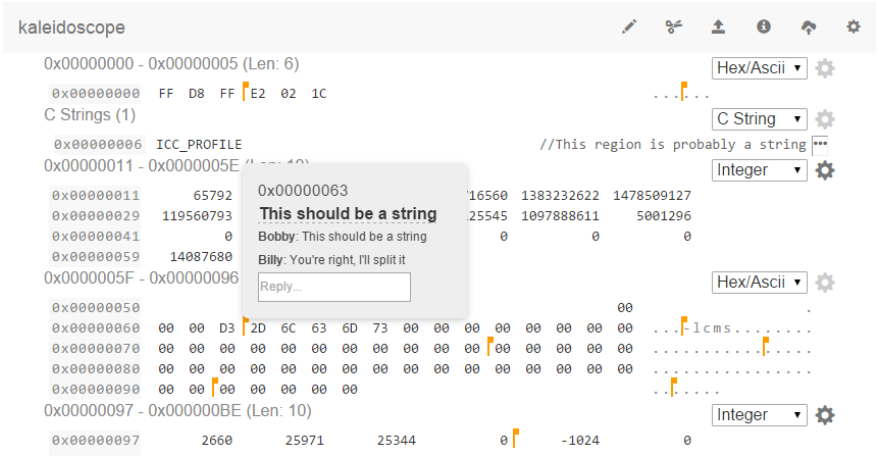

Figure 1: The overall Kaleidoscope UI, showing several comment markers and an open comment thread

4 Implementation

As previously mentioned, Kaleidoscope is a web app. Shown in Figure 1 on page 3 is the user interface presented to users when they open a workspace in their browser.

4.1 Technologies

The front end is written using HTML5 and Javascript. It uses jQuery to handle a lot of the DOM manipulation needed for the user interface. In addition to jQuery[7], a number of libraries are used for UI elements. Bootstrap[8] is used as a framework for the page and includes the icons used for various tools, Drop.js[3] is used for the comment popovers, X-editable[9] is used for styling a few of the input controls, and a modified jQuery slimscroll[5] is used for the scrollbar.

The back end server is written in Node.js[4], which has simplified development considerably since both the front end and back end are using the same programming language and some of the code can be shared between them. Routing for URLs on the back end is handled by Express[6]. The WebSockets used for communication between the server and clients are handled by Socket.IO[1]. In particular, Socket.IO has proven to be invaluable for minimizing the issues associated with support for WebSockets not being consistent across all browsers; in order to provide real-time collaboration to users by keeping their changes synchronized, it is vital for communication between individual clients and the server to be robust.

As a special note, disassembly of potential binary instructions is handled by the Capstone engine[2], a disassembly framework that supports several popular processor architectures.

4.2 Lens API

Kaleidoscope was designed to be extended by users adding their own lenses to display data in a format that is the most suitable for the file they are analyzing. A full list of API functions can be found in Table 1 on page 4. However, there are two main functions that handle the bulk of the work and allow lens authors to quickly implement new lenses.

The first of these functions is *preprocessData*, and it gives the lens an opportunity to process the data for its section and prepare it for displaying to the user. For some lenses this is crucial because processing a large amount of data can be very taxing for the processor. By processing the data ahead of time and getting it ready for display, the lens feels more responsive.

The second function that lenses must implement is *generateView*, which handles everything related to displaying the content through a particular lens. For some lenses the work required may be minimal, however others may need to take advantage of the work they were able to perform ahead of time in order to feel responsive.

4.3 Lens Helpers

Encouraging collaboration is a key feature of Kaleidoscope. To support lens authors, a number of features were added to make the job of implementing collaborative features in lenses easier. The biggest of these are the lens setting and comment thread APIs.

The lens setting API, is the simpler of the two, and requires the developer to implement 2 functions. One is a *getSettingsList* function that returns a list of the settings that the user should be able to customize. From there, the API takes care of generating a form with all of the required fields and their current values as returned by the *getSettingsList* function. Figure 2 shows an example of

Name	Description
getContentHeight()	*Returns the overall height of the content within the lens.*
getHeaderSummary()	*Returns a short header summary for the section when viewed with the lens.*
getName()	*Returns the name of the lens.*
generateView()	*Returns a view of what the user should see based on the current scroll position and content width, along with the height the content output should take up.*
preprocessData()	*Allows the lens time to process data for faster display later. For content that has a flexible width, it can depend upon the current width of the browser window.*
windowResized()	*Callback function that is called every time the browser window gets resized.*
updateComments()	*Callback function that is called when a new comment has been added in the section the lens is currently viewing so that it can update the displayed comments and comment markers.*
addCommentContent()	*Optional callback function that provides a summary of the most recent comment added for quickly refreshing the comments shown to a user.*
updateThreadSummary()	*Optional callback function that indicates when the summary for a comment thread has been updated.*
getSettingList()	*Returns the list of settings for a lens.*
changeSettings()	*Callback function that informs the lens of any new settings that affect how it should display its contents.*

Table 1: API for lens authors

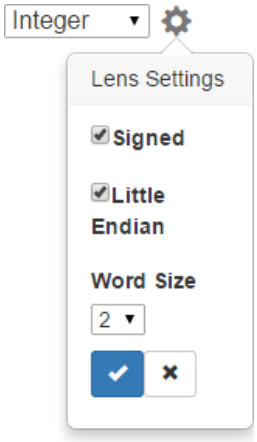

Figure 2: An example of the lens setting form generated for the Integer Lens

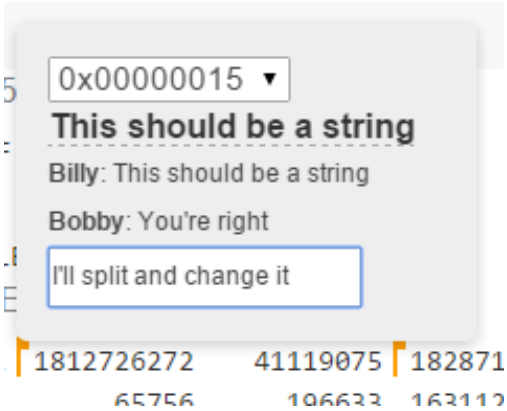

Figure 3: A discussion popover containing multiple threads

the form generated for the Integer Lens. If there are no settings that a user can change for a lens, the lens author can just return an empty list. The other function that must be implemented by a lens developer is a callback function that is used to inform a lens of when settings have changed, regardless of whether it is the local user or one of his or her collaborators that made the change.

The comment popover API helps lens authors by providing them with a popover that can be added to their lens by calling a few functions with the appropriate arguments. The popover can be set to display a range of discussion threads, and any element within the lens can be used as the target for making the popover appear. The orange comment flag visible in fig:comment can be used in a custom lens, however it is also trivial for a lens to use a different form of comment marker. Of course, the ambitious lens author is free to implement their own system of displaying comment threads if desired.

4.4 Lens Examples

As a starting point for users and lens authors, there are several lenses already implemented.

The basic lens is the Hex/Ascii Lens shown in Figure 4. This lens looks like what you would typically see in a hex editor or viewer. It uses the orange comment marker for showing where comments are (as can be seen in Figure 1 on page 3), and demonstrates how a lens can have

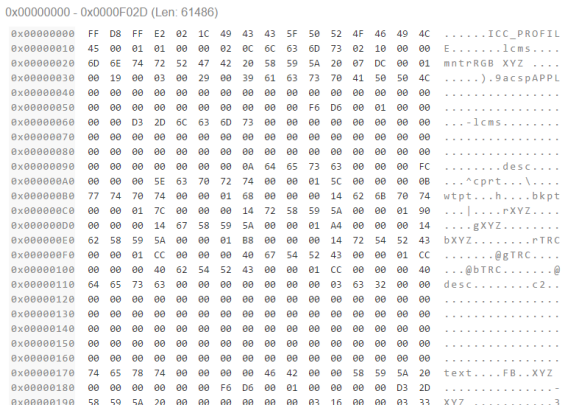

Figure 4: Hex/Ascii Lens

a flexible width that adjusts to the size of the browser window.

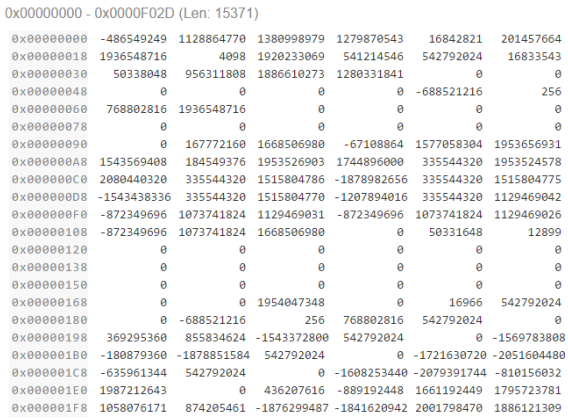

Figure 5: Integer Lens

The Integer Lens shown above in Figure 5 expands upon the features used by the Hex/Ascii Lens. In addition to the basic commenting system, it shows how clicking on comment markers can be used to display multiple discussion threads. The Integer Lens also makes use of Lens Settings so that users can customize what kind of integer the data is interpretted as when viewed through this lens.

The C String Lens shown above in Figure 6 demonstrates how data does not need to have a fixed number of bytes per element to be displayed, by interpreting the data in the section as a series of null terminated C Strings. This lens is easily extended to implement an Assembly Lens, in particular since it uses an alternative method of displaying comments that resembles those used when writing code.

Finally, an implementation of the Assembly Lens is shown in Figure 7. This is based off the C String Lens and provides lens settings that allow the user to select

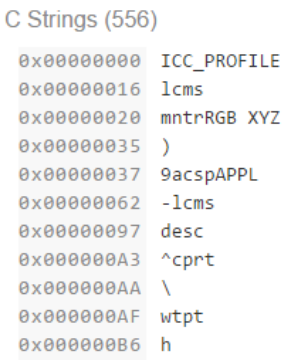

Figure 6: C String Lens

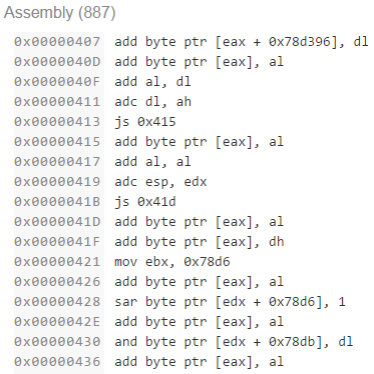

Figure 7: Assembly Lens

the processor architecture they would like to disassemble code for.

5 Future Work

A large part of this project is allowing a user to view data in a variety of different ways to facilitate the process of analyzing the data contained within files. As part of achieving this goal, Kaleidoscope was designed to make the process of creating new lenses to process and view data in different ways simple. There are a few additional lenses that could be added for common data types, in addition to a generic struct lens that would allow users to specify a structure format (such as for a header) and display the data in a given region with that format.

However, a large portion of the future work involved will be focused on improving the lens development process by adding new features. In particular the performance with tasks that require a large amount of preprocessing (such as disassembling code) is a concern so a standard method of implementing a background worker is one of the first priorities. Another challenge is handling files that are larger than a few Megabytes, so an effective method of sending chunks of a file and caching

them in the browser will need to be implemented to both reduce bandwidth usage and improve performance when using Kaleidoscope on systems like tablets or older computers that have a small amount of RAM.

The other goal of the project, facilitating collaboration, has a number of areas in which it could improve. The current commenting system could be extended to support the concept of highlighting a range of bytes to associate with a comment. Such an addition would allow users to more clearly denote the region of a file that a comment pertains to.

Another area of collaboration that becomes increasingly important as more users begin working on a project is communication between users. To simplify communication between large groups of people, the current commenting system could supplemented with both text chat and voice chat for a workspace, so that people with the workspace open are able to communicate without needing to switch to an external program. Also, as a project grows in size controlling who has access becomes important (especially with sensitive files), so user management features that allow a project owner to restrict who can edit and view a particular project workspace will become a vital feature.

References

[1] AUTOMATTIC. Socket.io, 2015.

[2] COSEINC. Capstone, 2015.

[3] HUBSPOT. Drop.js, 2015.

[4] JOYENT, INC. Node.js, 2015.

[5] PIOTR ROCHALA. jquery slimscroll, 2014.

[6] STRONGLOOP. Express, 2015.

[7] THE JQUERY FOUNDATION. jquery, 2015.

[8] TWITTER INC. Bootstrap, 2015.

[9] VITALIY POTAPOV. X-editable, 2012.