Escola d'Enginyeria de Telecomunicació i Aeroespacial de Castelldefels
UNIVERSITAT POLITÈCNICA DE CATALUNYA BARCELONATECH

MASTER THESIS

TFG TITLE: An enhanced SleuthKit GUI for digital forensics

DEGREE: Master's degree in Applied Telecommunications and Engineering Management (MASTEAM)

AUTHOR: Fernando Román García

ADVISOR: Juan Hernández Serrano

DATE: October 24, 2017

Title : An enhanced SleuthKit GUI for digital forensics

Author: Fernando Román García

Advisor: Juan Hernández Serrano

Date: October 24, 2017

Overview

The amount of digital information is growing every day. Due to its persistence is often used as evidence on judgements. Digital forensics is a science that is intended to found this proofs in order to make a reconstruction of the facts.
There are many tools to extract this information. Two well-known user interfaces are *EnCase Forensic* and *Autopsy*. Those softwares offer the possibility to see the file system of an image without modifying it (to preserve proof integrity), recover deleted files, generate timelines and even search functions to find clues that helps their cases.
However, there are no open-source user-friendly interfaces that can run over *Windows*, *Linux* and *Max OS*. In this project a new software, *Img-spy*, was created to perform those operations regardless of the operating system.
With such a goal, in this project we have built *The Sleuth Kit JavaScript* wrapper, which provides a JavaScript interface to the widely-use cross-platform open-source *Sleuth Kit* C library. With such a wrapper, JavaScript can be used to develop new applications for digital forensics analysis.
Besides being multi-platform, this project has been targeted to create a user-friendly framework for the usual digital forensics' work flow. This obvious purpose is very difficult to achieve because users have different preferences. Then, *Img-spy* user interface is intended to be customizable, giving the option to adjust the size of the interface panels and supporting multiple themes.
The results of this project are a good example of how a scalable application can be built using flux-like architecture based on *React-Redux* packages. Those modern libraries create a unidirectional data flow that starts with actions that modify the current state of the application and so affect the views of the user interface.

CONTENTS

LIST OF FIGURES

LIST OF TABLES

LISTINGS

INTRODUCTION

Nowadays it is impossible to think our daily lifes without digital technologies. They are involved in most of our daily actions, such as, to talk with friends, to read news, documents transport and entertainment. For this reason, our electronic devices can be considered as digital sources of information that can even prove possible alibis.

Digital forensics science, or digital forensics, is defined as:

> "The use of scientifically derived and proven methods toward the preservation, collection, validation, identification, analysis, interpretation, documentation, and presentation of digital evidence derived from digital sources for the purpose of facilitation or furthering the reconstruction of events found to be criminal, or helping to anticipate unauthorized actions shown to be disruptive to planned operations" [4]

In several cases, when a search warrant is given to investigators, they can proceed with the acquisition and analysis of a suspect's digital data source, for instance, computer hard disks, mobile phones, etc. Using this information, they can look for evidences that prove the suspect's innocence or guiltiness.

There are many ways to do this analysis, but the more generic one presented in [5], consists on:

Pre-Process This is one of the most important steps; although it is sometimes forgotten. It consists of all the tasks needed to be performed before the collection of data, including the necessary approvals by relevant authorities.

Acquisition & Preservation Once everything is prepared, now the investigator can acquire the data. An image and a digest of the analyzed data have to be computed. The image is an exact copy of the data and the digest is the result of a fixed-size unidirectional function, called hash, that summarizes the image. Among other properties, a hash function must be collusion-resistant; that is to say, that it has to be computationally infeasible to find two images with the same digest or to modify a image in such a way the digest remains the same.

That way, if anyone modifies the image and the digest is computed again using the modified image, a different value will be retrieved. So this procedure stands to guarantee the proof integrity.

Although not very formal, in the rest of the document we will use the term hash to refer to both the hash function and the digest (the result of the hash function), for it is the most widely way of using it.

Analysis This is the main phase of the investigation. It consists on extracting the relevant information acquired before. Many techniques can be used: data recovery, words lookup, etc. Note that, on this step, some tools may be used to be able to perform a proper analysis.

Presentation A good analysis is useless if it is not presented in an easy to understand manner. After this phase, the innocence of the suspect has to be proven or refused.

Post-Process Physical and digital proofs or evidences have to be returned and a report must be written to finish the case.

There are many digital forensics analysis tools. A very complete software is **EnCase Forensic** [6]. It has a lot of tools that can be used from the acquisition step until the final report. But its main disadvantage is that it is not open source.

Another well-known software regarding digital forensics is **The Sleuth Kit** [7]. This open-source cross-platform toolkit is a collection of terminal applications and a C library that allows you to analyze disk images and recover files from them. The Sleuth Kit team also developed **Autopsy**, which is a Windows' user interface to efficiently analyze hard drives and smart phones.

Due to the lack of open-source cross-platform tools with a friendly user interface, the objective of this project is to develop one that eases the forensic analysts' task addressing from the very beginning the main functionalities of a digital forensics software. The start point is to use The Sleuth Kit's C library [7] since it is a cross-platform very-mature project.

In order to achieve this goal, *Cross-platform development* (Chapter 1) will explain **why not all applications are cross-platforms** and will show several options to build one. Finally, will talk a little more in detail about the one that fits better with this project, Electron [8].

After the technological environment is decided, *Software architecture* (Chapter 2) talks about the evolution of model-view-controller to flux architecture and how it is implemented **to create a scalable application**.

Digital forensics analysis (Chapter 3) get deeper in how a digital forensics analysis can be done using The Sleuth Kit toolkit. Then, it explains the development of an important part of this project, **The Sleuth Kit JavaScript** (TSK-js), with a Node.js [9] C wrapper that gives access to The Sleuth Kit functionalities on Node.js.

Finally, ***Img-spy*** (Chapter 4) is the name of **the application that fulfills this project's objective**. It starts with a tour using the application and finishes explaining some difficult or interesting parts of the development.

CHAPTER 1. CROSS-PLATFORM DEVELOPMENT

In the early begining of computers, each manufacturer had to decide how to access to their hardware functionalities. Since there were no standards for some cases, the first inter-compatibility problems started to appear because programs were not able to run on different computers.

When Operative Systems (OS) appeared, they started to use drivers. A driver is an interface that maps logic functions (for instance, turn on a led) to hardware operations (short circuit some part of the electronics). Those drivers are often provided by manufacturers to let its hardware work when using an OS.

Nowadays desktop applications are not programmed directly for a specific hardware. OS are in charge to expose an application programming interface (API) to access those functionalities.

For a software, being cross-platform means being able to run on different OS. However, since the APIs exposed by the OSs are not equal, programs are not cross-platform by default.

1.1. Technological soup

Lots of technologies has been created aimed to writing just one code that runs in as many platforms as possible. The motivation to achieve this goal is obvious, to reduce the development cost. But, as opposed to this gain, it results in some performance reduction.

Most intuitive solution is to use cross-platform frameworks. Those have conditionals that uses an API function or another depending on which operative system is used to compile the library.

But the common way is with scripting programming languages. Those are human readable strings that are interpreted by a program called interpreter during execution time. One step further is to compile the source code into an intermediate pseudo-machine code that is easier to understand for the interpreter. This second group is known as virtual machine programming languages. Their main advantage is that programs run faster, but in contrast, developers lose some time compiling the code each time they want to test the application. Both technologies can run over all the platforms that its interpreter supports.

The last solution is to use transcompilers or source-to-source compilers. They translate from one programming language to many others. This translation could be done into a scripting language (there intercompatiblity is provided) or can make a specific translation to a compiled language depending on the OS.

More than twenty years putting effort to solve this problem has created a technological soup (Figure 1.1). Qt, Python, JavaScript, Java and Haxe [10][11][12][13][2] are some examples among many others.

Figure 1.1: Some cross-platform technologies

1.1.1. Qt

Qt is a complete cross-platform software framework with ready-made UI elements, C++ libraries, and a complete integrated development environment with tools for everything you need to develop software for any project [10].

```
1  #include <qapplication.h>
2  #include <qpushbutton.h>
3
4
5  int main( int argc, char **argv )
6  {
7      QApplication a( argc, argv );
8
9      QPushButton hello( "Hello world!", 0 );
10     hello.resize( 100, 30 );
11
12     a.setMainWidget( &hello );
13     hello.show();
14     return a.exec();
15 }
```

Listing 1.1: Qt hello world

Code of Listing 1.1 creates a window and add a button that says Hello World with size 100x30.

1.1.2. Python

Python is an example of a scripting programming language. The main design rule was to be very easy to read. For this reason, Python is the only language where tabulation rules must be strictly followed to make the program work.

It is also considered an object-oriented language suitable for many purposes. It has a clear, intuitive syntax, powerful high-level data structures, and a flexible dynamic type system [14].

Python is often used to create terminal programs or web services because GUI are not supported by default, but there are many frameworks that can be used, for instance, **Kivy**.

```
1 from kivy.app import App
2 from kivy.uix.button import Button
3
4 class TestApp(App):
5     def build(self):
6         return Button(text='Hello World')
7
8 TestApp().run()
```

Listing 1.2: Kivy hello world

As Listing 1.2 shows up, a window with is created with a button which text is ”Hello World”.

1.1.3. JavaScript

JavaScript is also a well-know widely-used scripting programming language. It is commonly used alongside with HTML and CSS as core technologies to make webpages. Moreover, due to its flexibility, it is used in many other fields like game programming or desktop applications.

It has many interpreters but all of them has to fulfill ECMAScript specification. V8 is an example of JavaScript engine used in Google Chrome and Node.js [9]. This latter example is widely used to build terminal applications and web services.

Some years ago, developing trends were to create web applications instead of desktop ones since it has less development cost and all platforms have browsers indeed. For this reason, web technologies have grown a lot and today can be compared with desktop technologies.

But web applications have several technological problems. The main one is that since the code is download from an external non-trusted source, the browser must limit the access to the computer for security reasons [15]. For instance, web applications can not read the user's file system. Moreover, programs have higher load periods due to those downloads.

Another important fact is that user experience (UX) will never be as good as using a desktop application due to it is embedded inside the browser and some space is lost due to menus.

1.1.3.1. Electron

Electron is a Node.js framework that lets developers build cross-platform desktop applications using JavaScript, HTML, and CSS [8]. As seen before, web technologies are now able to race desktop ones in terms of performance and scalability.

The idea behind Electron is take profit of the fact that Node.js and Chrome uses the same JavaScript engine (V8) to let Node.js instantiate Chrome windows and use them as user interface. Moreover, Electron Chrome windows are able to use all extra Node.js modules, for instance, to be able to access the computer's file system.

UX is improved respect web pages because Chrome menus are removed and application code is now inside user's computer.

For short, Electron help developers to create cross-platform desktop applications using the same technologies that the ones used to create a webpage.

1.1.3.2. NW.js

NW.js, previously known as node-webkit, works on the contrary of Electron. It is just a Chrome web browser where you can call all the node modules [16]. So the main difference is the entry point of the application. In Electron, the start point of your program execution is Node.js while on NW.js the entry point is an HTML file.

1.1.4. Java

This well-known programming language is one of the most used to build cross-platform desktop applications due to its maturity. Java is a general-purpose, concurrent, class-based, object-oriented language and it is designed to be easy to learn in order to let many programmers to have fluency in the language [17].

Classic compiled languages, such as, C and C++ , directly compile into machine code, which is directly interpreted by the hardware. As opposed to C and C++ , Java can be considered a virtual machine programming language.

```
1 // HelloWorldApp.java
2 class HelloWorldApp {
3     public static void main(String[] args) {
4         System.out.println("Hello World!");
5     }
6 }
```

Listing 1.3: Java hello world

Listing 1.3 is an example of Java program that prints the test "Hello World!" into the terminal. All files must contain just one class and the name of the file must match. The static function called main is the entry point of the application and in this example uses the standard output to print a message.

1.1.5. Haxe

Finally, Haxe is a high-level open source programming language that can be transcompiled many other languages. It has an ECMAScript-oriented syntax but with the peculiarity that can be typed [2].

Table 1.1 shows up many languages in which Haxe can be compiled. Then lets see a bit of Haxe syntax.

Name	Output Type	Main usages
JavaScript	Sourcecode	Browser, Desktop, Mobile, Server
Neko	Bytecode	Desktop, Server
PHP	Sourcecode	Server
Python	Sourcecode	Desktop, Server
C++	Sourcecode	Desktop, Mobile, Server
ActionScript 3	Sourcecode	Browser, Desktop, Mobile
Flash	Bytecode	Browser, Desktop, Mobile
Java	Sourcecode	Desktop, Server
C#	Sourcecode	Desktop, Mobile, Server

Table 1.1: Haxe use cases [2]

```
1 // Main.hx
2 class Main {
3     static public function main(): Void {
4         var text: String = "Hello World!";
5         trace(text);
6     }
7 }
```

Listing 1.4: Haxe hello world

As we can see in Listing 1.4, Haxe can be typed defining variables that way: `var text: String`. This code can be compiled for example to Python other languages using the terminal command:

```
user@host:/home/user$ haxe -main Main.hx -python main.py
user@host:/home/user$
```

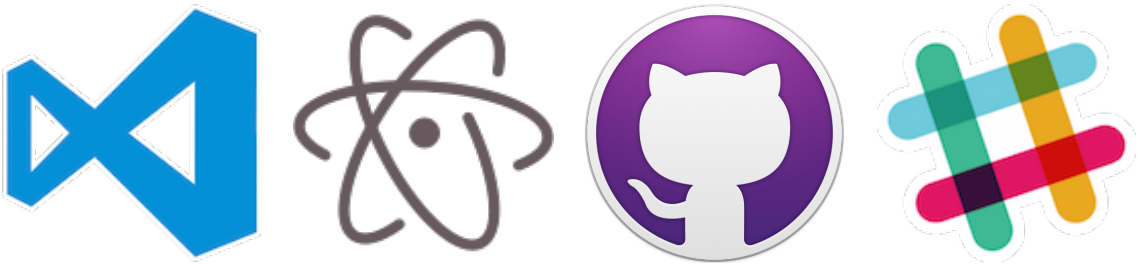
Listing 1.5: Haxe python transcompilation command

1.2. Electron

The technology that fits better with the project's specifications is Electron. There are several projects like this one that are developed using this framework. Even it is probably not as mature as other ones, it had a big impact in the developers community. Figure 1.2 shows several applications developed using Electron [8].

Moreover, the development of web applications has been evolved very fast with the objective to have complex programs with a very good scalability and without putting so much effort.

Then, some examples show how Electron works in a little more of detail. Complete code is available on Appendix A.



(a) Visual Studio Code (Microsoft) (b) Atom (c) Github Desktop (d) Slack

Figure 1.2: Electron application examples

1.2.1. Main process

JavaScript is a single threaded programming language. In order to be able to work in parallel, Electron has several processes. The initial one is called main process. It is in charge of create all browser windows, communicate them, to build application menus and control context events.

```
1  const {app} = require('electron')
2
3  /** ... **/
4  app.on('ready', createWindow)
5
6  // Quit when all windows are closed.
7  app.on('window-all-closed', () => {
8      // On macOS it is common for applications and their menu bar
9      // to stay active until the user quits explicitly with Cmd + Q
10     if (process.platform !== 'darwin') {
11         app.quit()
12     }
13 })
14
15 app.on('activate', () => {
16     // On macOS it's common to re-create a window in the app when the
17     // dock icon is clicked and there are no other windows open.
18     if (win === null) {
19         createWindow()
20     }
21 })
```

Listing 1.6: Electron app events

Code of Listing 1.6 is an example of the most common handled events. The most important is the *"ready"* event, where the main window of the application is created.

1.2.2. Browser window

Once event *"ready"* is called, browser windows can be created. Each window has its own independent process called renderer. The way to create windows is shown on Listing 1.7.

```
1  const {BrowserWindow} = require('electron')
2  const path = require('path')
3  const url = require('url')
4
5  /** ... **/
6
7  let win
8
9  function createWindow () {
10     // Create the browser window.
11     win = new BrowserWindow({width: 800, height: 600})
12
13     // and load the index.html of the app.
14     win.loadURL(url.format({
15         pathname: path.join(__dirname, 'index.html'),
16         protocol: 'file:',
17         slashes: true
18     }))
19
20     // Emitted when the window is closed.
21     win.on('closed', () => {
22         /** ... **/
23         win = null
24     })
25 }
26 /** ... **/
```

Listing 1.7: Electron window creation

Browser windows can be created using several parameters to define its initial behavior, for instance: width, height, position, etc. They can load files using several protocols (HTTP, FTP, etc) or directly access to local hard disk. In Electron, the most common way is to use local files to reduce load time and let the application work without Internet connection. Finally, window objects also have events, for example: *"closed"*, *"ready-to-show"*, *"move"*...

1.2.3. Interprocess communication

Finally, to be able to communicate the main process and the renderer processes, Electron provides inter-process communication (IPC). By default, it can provide communications main-to-renderer and renderer-to-main. Therefore, if renderer-to-renderer communication is needed, a bridge must be implemented inside the main process.

1.3. Web technologies

Core World Wide Web technologies, as seen before, are JavaScript, HTML and CSS. Nevertheless, there are many other specialized tools to solve specific deficits those technologies have. For instance, Typescript, SCSS and Gulp are used to enhance scalability and maintainability.

1.3.1. Typescript

Typescript[18] is a programming language created by Microsoft that is a typed superset of JavaScript that compiles to plain JavaScript. It has two main advantages:

Typed language Since Typescript is a typed language and it has a compilation process, any mismatch of parameters used in a function call are detected before testing anything. Also, Integrated Development Environments (IDEs) can help developers highlight those mismatches because type information is provided.

Compiles to plain Javascript There are many web browsers and not all of them support current ECMAScript standard. Using TypeScript developers don't need to take care of it because target JavaScript version can be selected.

Many JavaScript libraries also provide its TypeScript typings. Using this information, the compiler can detect if the library is properly used.

```
1  // Play it on JsFiddle: https://jsfiddle.net/q9ezexgh/
2  interface Person {
3      name: string;
4      surname: string;
5  }
6  function personGreetings(whom: Person) {
7        return whom.name + " " + whom.surname;
8  }
9
10 type GreetingsCallback<T> = (whom: T) => string;
11 class Greetings<T> {
12     constructor(private cb: GreetingsCallback<T>) {}
13
14     public to(whom: T) {
15         document.body.innerHTML = this.cb(whom);
16     }
17 }
18
19 const greetings = new Greetings<Person>(personGreetings);
20 greetings.to({ name: "John", surname: "Smith"});
```

Listing 1.8: TypeScript example

Listing 1.8 is a TypeScript example that shows some advanced type declarations. As can be seen, it supports templates, interfaces and even defining function types.

Each time TypeScript is compiled, a tool called TsLint can be used to check if some programming style guides are followed. Follow this rules, even if it is not strongly needed, is very useful to keep code easy-to-read. For instance, a common rule used in JavaScript is to name variables following lower camel case and upper camel case for classes.

1.3.2. Sass

Like TypeScript, SASS is a code transcompiler, but it translates from SASS to CSS. It is designed to improve CSS functionalities.

```
1  $app_green : #00B819;
2
3  @mixin flex-row() { display: flex; flex: row; }
4
5  .app {
6      @include flex-row;
7      background-color: $app_green;
8
9      &.hide { display: none }
10
11     button { color: blue; }
12
13     .flex-row {
14         @include flex-row;
15     }
16 }
```

Listing 1.9: Sass example

Listing 1.9 helps to see all the advantages that SASS provides. The main ones are:

Nesting CSS selectors Improves code scalability.

Variables Helps to define palettes, sizes, etc.

Mixins Similar to functions, they are very useful to define common behaviors.

1.3.3. Gulp

Gulp is a toolkit for automating painful or time-consuming tasks in your development workflow, so you can stop messing around and build something [19].

This project has several routine tasks, such as launch electron, compile TypeScript and SASS each time a file change happens, etc. For this reason, several gulp tasks are created to help development.

```
1  var sass = require('gulp-sass');
2  var sourcemaps = require('gulp-sourcemaps');
3  var watch = require('gulp-watch');
4
5  module.exports = function(gulp, config) {
6      gulp.task('sass', sassTask);
7      gulp.task('watch:sass', watchScss);
8
9      //////
10
11     function sassTask() {
12         return gulp.src('src/style/**/*.scss')
13             .pipe(sourcemaps.init())
14             .pipe(sass({outputStyle: 'compressed'}).on('error',
                   sass.logError))
15             .pipe(sourcemaps.write())
16             .pipe(gulp.dest('dist/style'));
```

```
17      }
18
19      function watchScss() {
20          watch('src/style/**/*.scss', sassTask);
21      }
22  }
```

Listing 1.10: Gulp task to compile SASS

For instance, Listing 1.10 shows how a task to compile all .sass files is implemented.

CHAPTER 2. SOFTWARE ARCHITECTURE

"Most systems work better they are kept simple rather than complicated" [20]. This is the main statement of the KISS principle, acronym for "Keep it simple, stupid". KISS philosophy is very used on software development because code tends to chaos and disorder. If the implementation of a functionality is not properly thought, it adds complexity to the program work flow. Therefore, to reduce the architecture entropy should be one of the main design patterns on any software.

Model-View-Controller (MVC) is a well-known software architecture that consists on the use of this tree elements to build a user interface. It was introduced by Trygve Reenskaug in the seventies [21]. When web applications appeared, this model was applied in many important projects like Microsoft Support.

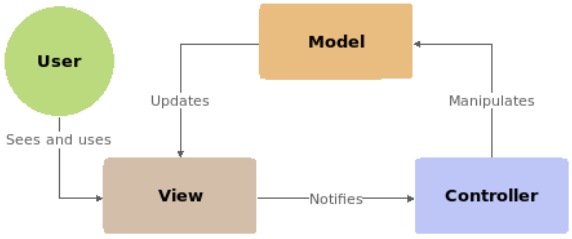


Figure 2.1: Diagram of interactions within the MVC pattern. [1]

As shown on Figure 2.1, it is a simple model to represent a user interface (UI). What user sees is represented by the view. Each time the user performs an action, view notifies the controller and it modifies the current model. Since view is watching changes on the model, this change is detected and, thus, view is affected.

The problem is that this architecture becomes complex and complex while increasing the number of views. Actions from a view can affect other views' model and this changes can trigger other actions. This complexity could even generate unexpected loops as Figure 2.2 proves.

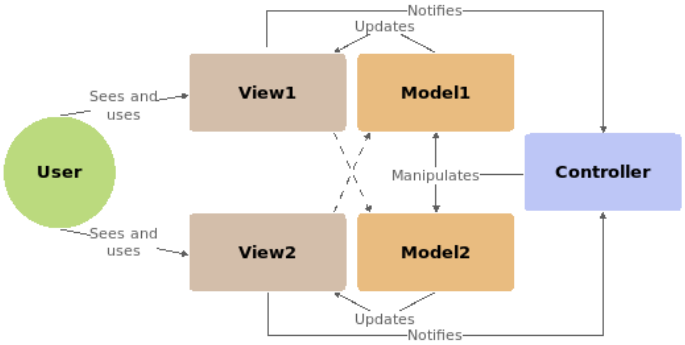


Figure 2.2: Diagram of interactions within the MVC pattern with many views

2.1. Flux architecture

In order to solve the MVC scalability problem, Facebook launched Flux, represented on Figure 2.3. Now views are watching a plain object (state) saved inside the store. Actions are also plain objects that are dispatched changing the current stored state.

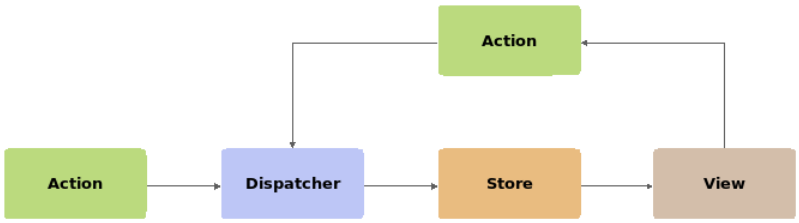


Figure 2.3: Diagram of Flux

Even through views just wait change events produced by the store and render themselves using the current state information. Note that the render processes must not call any action.

2.2. React

React is an implementation of the Flux view block. It is a component-based JavaScript library to build user interfaces [22]. Each React component can have input properties, state, and must implement a render function. Moreover, to make things easier, React supports JSX which are JavaScript files where HTML code can be directly used.

```
1  class HelloMessage extends React.Component {
2    render() {
3      return (
4        <div>
5          Hello {this.props.name}!
6        </div>
7      );
8    }
9  }
10
11 ReactDOM.render(
12   <HelloMessage name="World" />,
13   mountNode
14 );
```

Listing 2.1: React hello world

Listing 2.1 shows up an example of React component. It is stateless, it has a property call name and an easy render function to create a *div* with text. Render functions can also have components inside themselves. That way, React creates a component tree 2.4.

Components may need to perform actions. Those cannot be performed on render function, as seen before. Instead, to do that each component have a lifecyle. It consists of a set

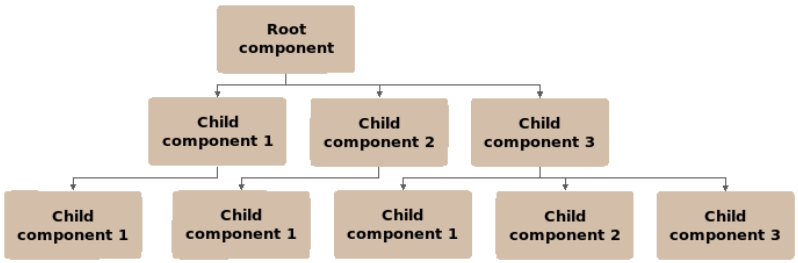


Figure 2.4: React tree

of functions that are triggered when some external events happen. Most used ones are *"componentWillMount"* and *"componentWillUnmount"*. Components also can listen to its view events, for instance button clicks, to perform actions.

2.3. Redux

Redux was inspired by several important qualities of Flux. Like Flux, Redux prescribes that the model update logic is concentrated in a certain layer of the application ("stores" in Flux, "reducers" in Redux) [23]. Redux have tree principles must be followed [24].

Single source of truth The state can be stored easily if just one store is used. Furthermore, the application will be easy to debug. Once again, keep it simple.

State is read-only The only way to update the state is using actions. Each time the state changes, a new independent state must be created.

Changes are made with pure functions Reducers must be pure functions, that results in the following statement: "using the same initial state, repeating the same list of actions must produce the same final state".

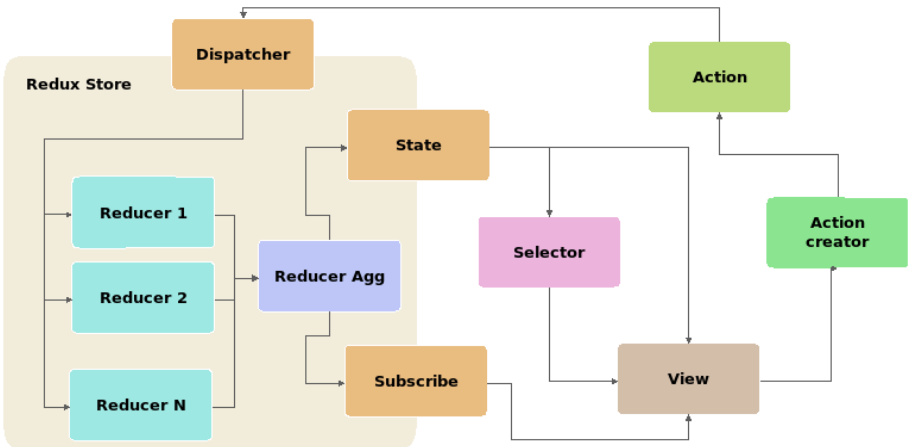


Figure 2.5: Redux architecture

Figure 2.5 shows up the overall Redux's architecture. First of all, a store must be created. This store contains reducers that map action, created by action creators, into state

changes. If needed, selectors can be used to cache some state costly transformations. Store also lets to subscribe to state change notifications.

2.3.1. Action creators

An action is a plain JavaScript object that contains a field called type to identify it and other optional fields. Same type of action can be created in many places so it is extremely recommended to use action creators.

Action creators are functions that create those objects. Normally are defined using lambdas. An example of action creator is shown on Listing 2.2. This one is intended to create an action that appends a text item into the state.

```
1 const appendItem = (text) => ({
2     type: "APPEND_ITEM",
3     payload: text
4 });
```

Listing 2.2: Redux action creator

2.3.2. Reducers

Reducers aim to return new states each time an action is produced. They must be implemented using pure functions. A function must follow tree statements to be considered pure.

The fist one is **mapping**. To fulfill it means that same input always results on the same output. Therefore, pure functions can not generate random numbers, current date, nor depend on external variables. If this statement is not followed, functions are difficult to test since its return value is unpredictable. To make a function fulfill mapping is as easy as adding unpredictable values as parameters. Listing 2.3 is an example of function that multiplies a random value with a parameter.

```
1  // Play it on JSFIDDLE: https://jsfiddle.net/dqvnmv2d/
2  const multKo = (x) => Math.random() * x;
3  const multOk = (x, rnd) => rnd * x;
4
5  let res;
6  let x = 5;
7
8  // KO
9  res = multKo(x);
10 document.body.innerHTML += "Imposible to test!</br>";
11
12 // OK
13 let rand = Math.random();
14 res = multOk(x, rand);
15 if (res === (x * rand)) {
16       document.body.innerHTML += "It works!</br>";
17 } else {
18       document.body.innerHTML += "It doesn't work!</br>";
```

```
19 }
```

Listing 2.3: Mapping example

The second statement is **avoid side effects**. This means that functions must not modify anything outside them. If a pure function needs to access outside, maybe it is ill-designed.

```
1  // Play it on JSFIDDLE: https://jsfiddle.net/zrL8cr27
2  const bob = { name: "Bob", age: 25};
3
4  // KO
5  const print = (person) => {
6       document.body.innerHTML += person.name + "(" + person.age +
            ")</br>";
7  }
8  print(bob);
9
10 // OK
11 const toString = (person) => {
12      return person.name + "(" + person.age + ")</br>";
13 }
14 document.body.innerHTML += toString(bob);
```

Listing 2.4: Avoid side effects example

The last one is **no external mutable**. This means that the output of a function must be independent to changes other variables. But how can this happen? Easy, when a JavaScript object is assigned to another variable, it is a reference of the first one, not a copy. If any of both variables changes, it will affect also to the other. A simple example of this issue is shown on Listing 2.5.

```
1  // Play it on JSFIDDLE: https://jsfiddle.net/yzL6k1L4/
2  var personA = { name: "John", age: 32 };
3  document.body.innerHTML += personA.name + "</br>"; // Prints John
4  var personB = personA;
5  personB.name = "Anna";
6  document.body.innerHTML += personB.name + "</br>"; // Prints Anna
7  document.body.innerHTML += personA.name + "</br>"; // Prints Anna too!
```

Listing 2.5: Object reference effect

Therefore, an example of non pure function due to no external mutation is to return an object parameter. In order to follow no external mutable principle, the returned object must be a copy of the one provided as parameter (Figure 2.6).

```
1  // Play it on JSFIDDLE: https://jsfiddle.net/edw6uoca/1/
2  const setNameKo = (person, name) => {
3      person.name = name;
4      return person;
5  };
6  const setNameOk = (person, name) =>
```

```
7     Object.assign({}, person, { name: name });
```

Listing 2.6: No external mutation example

Then, applying this three principles, Listing 2.7 shows up an example of reducer for the action created on Listing 2.2.

```
1  function appendItemReducer(state, action) {
2      if (!state) state = []; // Default state
3
4      switch (action.type) {
5          case "APPEND_ITEM":
6              return [ ...state, action.payload ];
7
8          default: // Return always the state!
9              return state;
10     }
11 }
```

Listing 2.7: Redux reducer

This function is called by the store each time an action is dispatched. Reducers can be aggregated using a function provided by Redux called *"combineReducers"*. Listing 2.8 shows an example.

```
1  const aggReducer = combineReducers({
2      items: appendItemReducer,
3      counter: counterReducer
4  });
```

Listing 2.8: Reducer aggregation

2.3.3. Selectors

Selectors are functions that can cache the return value and it is just recomputed if any of the input parameters changes. For instance, a selector can be created to select items that starts with letter 'I'. This example is shown on Listing 2.9.

```
1  const getItems = (state) => state.items;
2
3  const getItemsSubset = createSelector(
4      [getItems],
5      (items) => items.filter((item) => item.startsWith("I"))
6  );
```

Listing 2.9: Redux selector

2.3.4. Store

Creating a store is as easy as providing the root reducer and the initial state. Then actions can be dispatched and a function can be subscribed to detect any state change.

Listing 2.10 shows how to create a store. Selector created in Section 2.3.3. is used to compute a subset of items just if the item list has change. Also, several actions are dispatched.

```
1  // Play it on JsFiddle: https://jsfiddle.net/9s57z0om/2/
2  const store = createStore(aggReducer, {});
3
4  store.subscribe(() => {
5      const currState = store.getState();
6      render(containerA, getItems(currState));
7      render(containerB, getItemsSubset(currState));
8      counter.innerHTML = currState.counter;
9  });
10
11 store.dispatch(appendItem("Item 1"));
12 store.dispatch(appendItem("Item 2"));
13 store.dispatch(appendItem("AnotherItem 1"));
14
15 button.addEventListener("click", () => {
16    store.dispatch(increaseCounter());
17 });
```

Listing 2.10: Redux store creation and management

2.4. Action observables

Observables are data streams provided by a library called Reactive Extensions for JavaScript (rxjs). The idea is to treat those streams as asynchronous events and apply fluent query operators to describe behaviors.

```
1  // Play it on JsFiddle: http://jsfiddle.net/mc4eekcp/7
2  Rx.Observable
3      .fromEvent(btn, "click")
4      .scan(
5          (info, i) => ({ curr: new Date(), last: info.curr }),
6          { curr: new Date(), last: null } // Initial "info" value
7      )
8      .map((info) => (info.curr - info.last) / 1000)
9      .subscribe(
10         (diff) => text.innerHTML = diff + " seconds since last click"
11     )
```

Listing 2.11: Rxjs observable to count seconds until last click

As shown on Listing 2.11, complex behaviors can be implemented in an easy to read way using this library. The example shows up how to count the difference in time between two

button clicks. To do that, an observable is created to watch the event click of a button. The operator scan caches the last returned value sending it as first parameter of the callback function. Last scanned is mapped to get the difference in seconds of last and current time values. Finally, this difference is printed on a text element.

Since Redux's reducers must be pure functions and there are many complex behaviors that have to be chained when some actions are produced, Redux Observables provide a middleware that handles action streams using rxjs observables. The observable is executed after the action has been reduced. So Redux architecture (Figure 2.5) must be complemented with Figure 2.6.

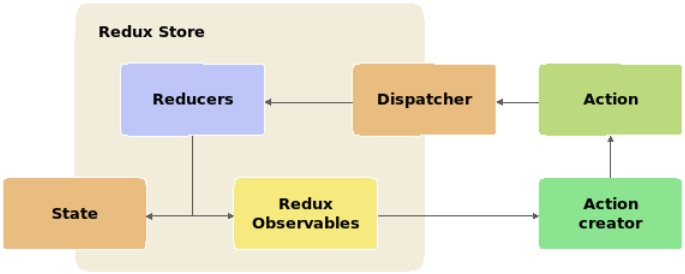


Figure 2.6: Redux Observables architecture

2.5. React-Redux

To let React and Redux work together, Redux store must be provided to whole React tree. Also, action creators and state have to be mapped into React properties.

A package called react-redux adds a React component named *"Provider"* that inserts a store inside the React tree. Children of *"Provider"* can connect to it by using the *"connect"* function. It needs two parameters, one to map the store state to component's properties and a second one to map actions.

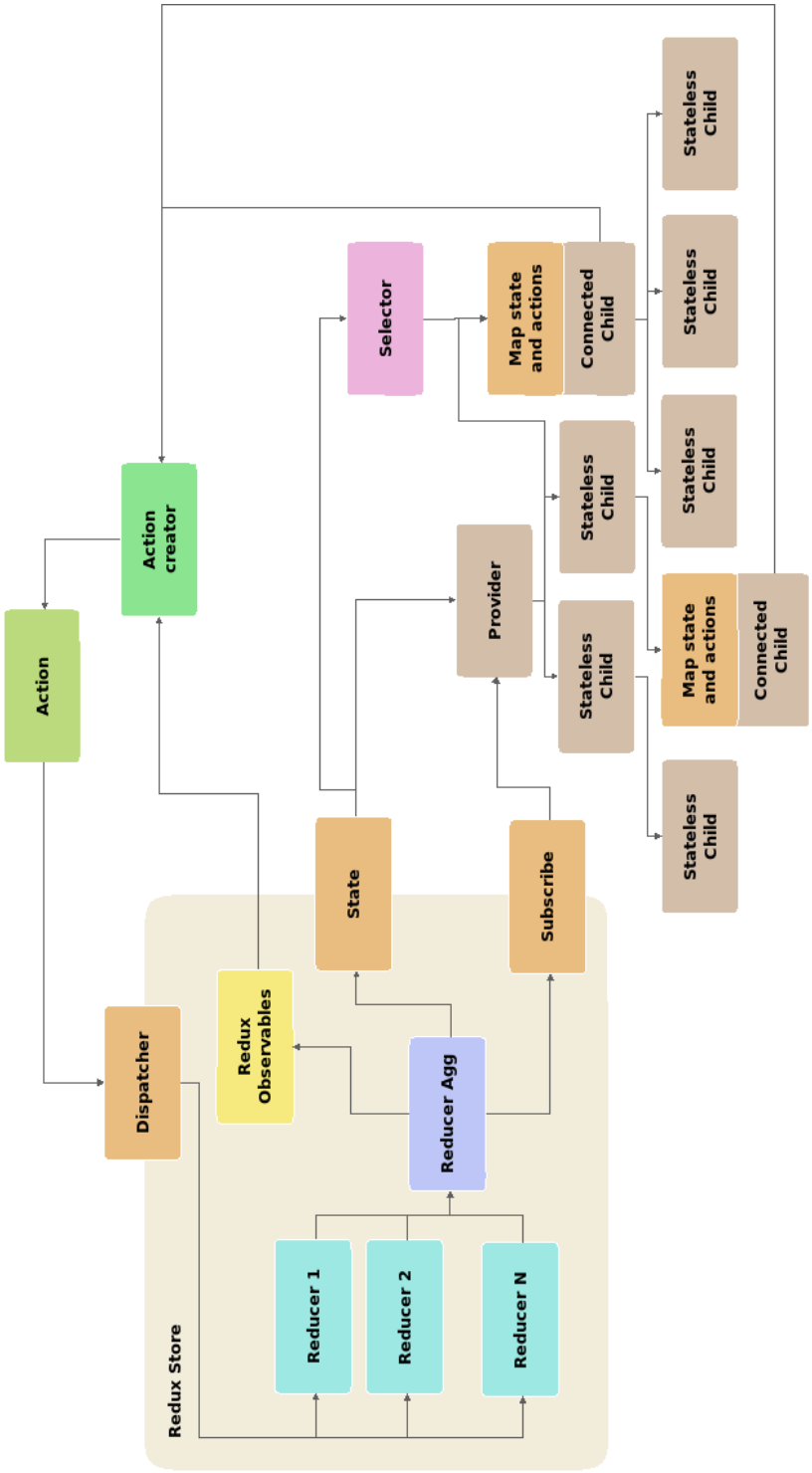


Figure 2.7: Overall architecture

CHAPTER 3. DIGITAL FORENSICS ANALYSIS

The Sleuth Kit (TSK) is an open source cross-platform collection of terminal applications and a C library that allows investigators to analyze disk images and recover files from them [7]. It is divided in file system and volume tools.

The first group allows investigators to examine the file system of a suspect computer in a non-intrusive way. Main file system tools are shown on Table 3.1.

Command	Description
fsstat	Shows file system details and statistics
fls	Lists allocated and deleted file names in a directory
ffind	Finds allocated and unallocated file names
icat	Extracts the data units of a file

Table 3.1: Main file system tools from TSK [3]

Volume tools are the ones that brings off the layout of disks and another media. They are needed to identify where partitions are located to be able to proceed with a file system analysis. Table 3.2 lists main volume tools.

Command	Description
mmls	Displays the layout of a disk
mmstat	Display details about a volume system

Table 3.2: Main volume tools from TSK [3]

In order to explain how The Sleuth Kit works easily, an example will be used. In that specific case, an investigator will extract the timeline for an image and will look for a file called *letter.txt* to print its content.

3.1. The Sleuth Kit analysis

To start the TSK analysis, the command *mmls* is used to output the information of all partitions. If volume type is not specified, TSK will autodetect it. The output will look like Listing 3.1. This example has tree volumes:

Primary Table (#0) It contains the primary table.

Unallocated Unallocated space.

Win95 FAT32 (0x0C) This partition has an offset of 56 sectors and it contains a file system formatted with FAT32.

```
user@host:/home/user$ mmls hdd-001.dd
DOS Partition Table
```

```
Offset Sector: 0
Units are in 512-byte sectors

Slot        Start       End         Length      Description
00: Meta    0000000000  0000000000  0000000001  Primary Table (#0)
01: -----   0000000000  0000000055  0000000056  Unallocated
02: 00:00   0000000056  0007925759  0007925704  Win95 FAT32 (0x0C)
user@host:/home/user$
```

Listing 3.1: TSK mmls output example

To perform a deeper analysis, file system tools must be used. In that case, those will need the offset of 56 sectors to be specified. The first tool, *fsstat*, will display some file system characteristics, for instance, root inode, sector size and the file system layout.

Also, the registers inside the disk can be listed in a non-intrusive way using *fls*. In this specific example (Listing 3.2), the file system contains four folders. One of them, *_NTITL~1* is deleted. Also, some virtual registers are displayed ($MBR, $FAT1, $FAT2, $OrphanFiles). If it is necessary to go through the file system tree, a directory inode can be specified to list registers inside it or analyze whole disk recursively.

```
user@host:/home/user$ fls -o 56 hdd-001.dd
d/d 4:  .Trash-1000
d/d * 5: _NTITL~1
d/d 7:  Personal
d/d 9:  Imagenes
v/v 126563459: $MBR
v/v 126563460: $FAT1
v/v 126563461: $FAT2
d/d 126563462: $OrphanFiles
user@host:/home/user$
```

Listing 3.2: TSK fls output example

Several times, it is necessary to export the whole file system in order to obtain processed information using other tools. The command arguments that are necessary are:

```
user@host:/home/user$ fls -m / -o 56 -r hdd-001.dd > fs.csv
user@host:/home/user$
```

Listing 3.3: Export file system to CSV

This CSV file can also be used to generate timelines using a specific TSK tool called *mactime*. The output format of *mactime* can be also adjusted to process it with any CSV program. Columns shown on Listing 3.4 are: date, file size, activity type, Unix permission, User & Group IDs, inode and file name. Allowed activity types are: modified (m), accessed (a), changed (c) and created (b).

```
user@host:/home/user$ haxe -main Main.hx -python main.py
Xxx Xxx 00 0000 00:00:00 338558 ..c. r/rrwxrwxrwx 0   0       10758
    /Imagenes/bufon.png
        10 ..c. r/rrwxrwxrwx 0     0       10889   /Personal/file.txt
      4096 ..c. d/drwxrwxrwx 0     0       11142   /.Trash-1000/info
```

```
      4096 ..c. d/drwxrwxrwx 0    0       11144   /.Trash-1000/files
        70 ..c. r/rrwxrwxrwx 0    0       11274
            /.Trash-1000/info/carta.txt.trashinfo
        72 ..c. r/rrwxrwxrwx 0    0       11401   /.Trash-1000/files/carta.txt
      4096 ..c. d/drwxrwxrwx 0    0       4       /.Trash-1000
         0 ..c. d/drwxrwxrwx 0    0       5       /_NTITL~1 (deleted)
      4096 ..c. d/drwxrwxrwx 0    0       7       /Personal
      4096 ..c. d/drwxrwxrwx 0    0       9       /Imagenes
Fri Aug 25 2017 00:00:00 338558 .a.. r/rrwxrwxrwx 0   0       10758
    /Imagenes/bufon.png
        10 .a.. r/rrwxrwxrwx 0    0       10889   /Personal/file.txt
      4096 .a.. d/drwxrwxrwx 0    0       11142   /.Trash-1000/info
      4096 .a.. d/drwxrwxrwx 0    0       11144   /.Trash-1000/files
        70 .a.. r/rrwxrwxrwx 0    0       11274
            /.Trash-1000/info/letter.txt.trashinfo
        72 .a.. r/rrwxrwxrwx 0    0       11401   /.Trash-1000/files/letter.txt
      4096 .a.. d/drwxrwxrwx 0    0       4       /.Trash-1000
         0 .a.. d/drwxrwxrwx 0    0       5       /_NTITL~1 (deleted)
      4096 .a.. d/drwxrwxrwx 0    0       7       /Personal
      4096 .a.. d/drwxrwxrwx 0    0       9       /Imagenes
Fri Aug 25 2017 15:06:56 338558 m..b r/rrwxrwxrwx 0   0       10758
    /Imagenes/bufon.png
Fri Aug 25 2017 15:07:18  4096 m..b d/drwxrwxrwx 0    0       9       /Imagenes
Fri Aug 25 2017 15:07:22     0 m..b d/drwxrwxrwx 0    0       5       /_NTITL~1
    (deleted)
Fri Aug 25 2017 15:07:30  4096 m..b d/drwxrwxrwx 0    0       4
    /.Trash-1000
Fri Aug 25 2017 15:08:04    72 m..b r/rrwxrwxrwx 0    0       11401
    /.Trash-1000/files/carta.txt
Fri Aug 25 2017 15:08:06    70 m..b r/rrwxrwxrwx 0    0       11274
    /.Trash-1000/info/carta.txt.trashinfo
Fri Aug 25 2017 15:08:12  4096 m..b d/drwxrwxrwx 0    0       11142
    /.Trash-1000/info
< 4096 m..b d/drwxrwxrwx 0    0       11144   /.Trash-1000/files
Fri Aug 25 2017 15:08:22    10 m..b r/rrwxrwxrwx 0    0       10889
    /Personal/file.txt
      4096 m..b d/drwxrwxrwx 0    0       7       /Personal
user@host:/home/user$
```

Listing 3.4: Haxe python transcompilation command

As Listing 3.4 shows, the inode 11401 contains a file called letter.txt. Print its content can be easily done using the command *icat*.

```
user@host:/home/user$ icat -o 56 hdd-001.dd 11401
Mr. Someone,
You must know I am guilty!

Best regards,
Nobody Jones
user@host:/home/user$
```

Listing 3.5: Export a file contained inside the file system

3.2. The Sleuth Kit JavaScript

Bearing in mind the need to develop a wrapper to connect TSK with a JavaScript application **The Sleuth Kit JavaScript (TSK-js) has been created**. This wrapper must be able to execute the same analysis carried out in the previous chapter (Chapter 3.1.) but using JavaScript.

It has been raised to create a JavaScript object with one construction argument in with the image file path is specified. This object will have the necessary methods to perform the entire analysis. Those methods are explained on Table 3.3.

Method	Arguments	Returns
Analyze	None	Type (disk/partition) and if type disk, returns its partitions specifying the offset sectors
List	Offset sectors and inode	List file system files
Get	Offset sectors and inode	Buffer with inode content
Timeline	Offset sectors, inode and callback function	Timeline results
Search	Needle, offset sectors, inode and callback function	None

Table 3.3: TSK-js object methods

Search and Timeline take some time to execute. That is why both have a callback function as parameter. Each time a result is found, this callback function is called.

An example of JavaScript code that can perform the analysis proposed on Chapter 3.1. is shown of Listing 3.6.

```
1  // analysis.js
2  const { TSK } = require("tsk-js");
3
4  main(process.argv[0]);
5
6  ////
7
8  function searchRecursive(needle, img, imgaddr, inode, cb) {
9      const files = img.list({ imgaddr, inode });
10     files
11         .filter((f) => f.name === needle)
12         .forEach((f) => cb(f));
13
14     files
15         .filter((f) => f.type === "directory")
16         .forEach((f) => searchRecursive(needle, img, imgaddr, f.inode,
               cb));
17 }
18
19 function analyzePartition(img, imgaddr) {
20     // Search file
21     searchRecursive("carta.txt", img, imgaddr, undefined, (file) => {
22         const { inode } = file;
23         const buff = img.get({ imgaddr, inode });
```

```
24
25          console.log("File found!");
26          console.log("Print it's content:");
27          console.log("--------------------------");
28          console.log(buff.toString());
29          console.log("--------------------------");
30      });
31
32      // Generate timeline
33      const timeline = img.timeline(() => {}, { imgaddr });
34      console.log("Timeline length '" + timeline.length + "'");
35  }
36
37  function analyzeDisk(img, res) {
38      res.partitions
39          .filter((p) => p.hasFs)
40          .forEach((p) => analyzePartition(img, p.start));
41  }
42
43  function main(imgfile) {
44      const img = new TSK(imgfile);
45      const res = img.analyze();
46      if (res.type === "disk") {
47          analyzeDisk(img, res);
48      } else {
49          analyzePartition(img, 0);
50      }
51  }
```

Listing 3.6: The Sleuth Kit JavaScript analysis

This analysis can be executed using the following command:

```
user@host:/home/user$ node analysis.js hdd-001.dd
File found!
Print it's content:
--------------------------
Mr. Someone,
You must know I am guilty!

Best regards,
Nobody Jones
--------------------------
Timeline length '29'
user@host:/home/user$
```

Listing 3.7: Execute The Sleuth Kit JavaScript analysis

The implementation of this package is done using C++ and node-gyp that is a compiler to create Node.js addons [25].

A TypeScript types file is provided in order to help transcompiler with type validations. Typings can be found on Appendix B. This package is available on Node Package Manager (https://www.npmjs.com/package/tsk-js). The source code is also available on GitHub (https://github.com/FRG-UPC-Deliverables/tsk-js).

CHAPTER 4. IMG-SPY

As seen in the introduction, the main objective of this project is to develop an open-source cross-platform software with a user-friendly interface. For this purpose, **Img-spy was created**. The overall concept of the application is to give some well-defined tools to be used in a digital forensics analysis.

4.1. Application walk around

The first time the application is launched, a window asking to select a case appears (Figure 4.1). That means to select a folder of user's local computer. If it was used before, all previous progress will be recovered.

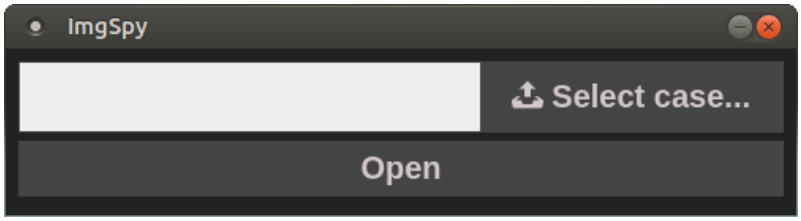


Figure 4.1: Img-spy case selector

Once the case to work is picked, the main application window (Figure 4.2) is shown. This one contains three tools needed to proceed with the analysis. Each tool is represented with an icon.

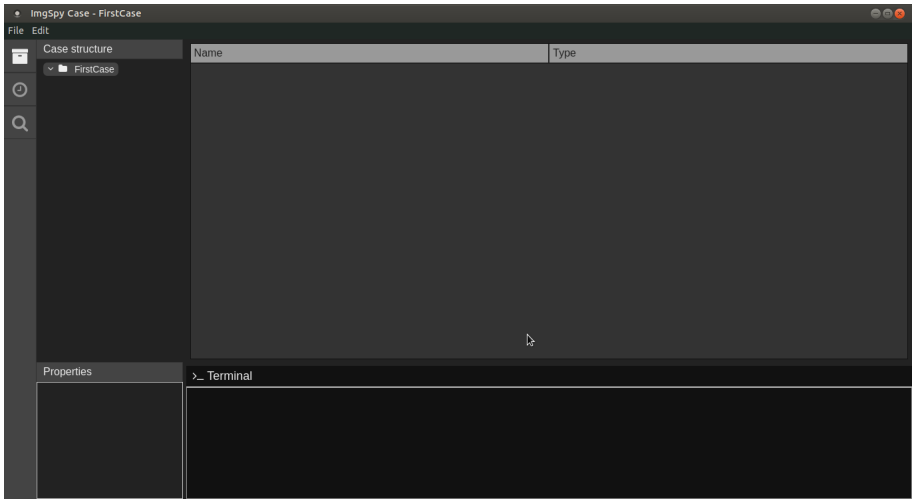
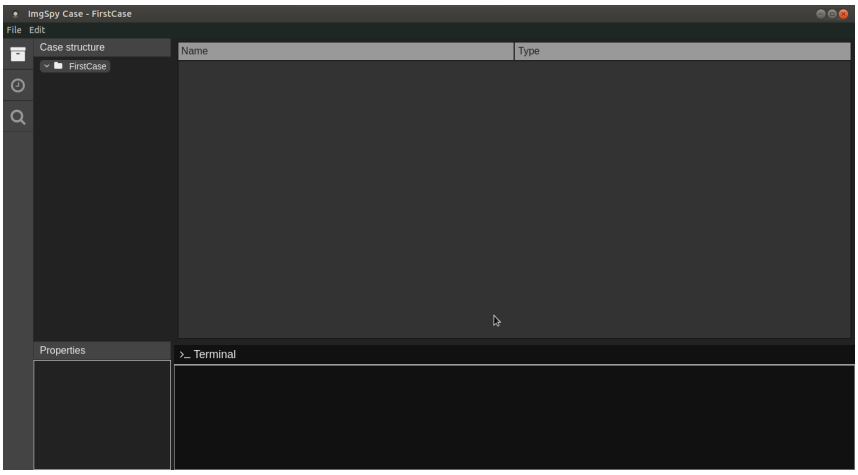


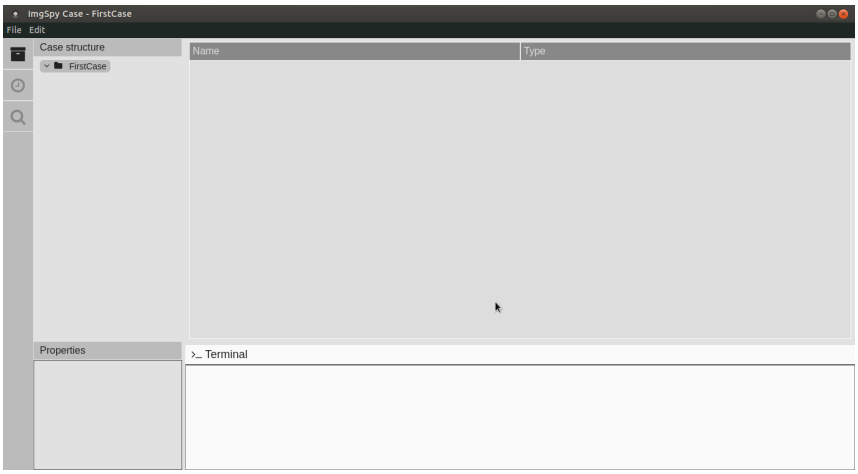
Figure 4.2: Img-spy case window

- **Explorer.** It examines images in a non-intrusive way.
- **Timeline.** It generates a list with the activity on the file system.
- **Search.** It searches a string inside an image.

One of the main objectives of the project is to get a user-friendly user interface for the typical digital forensics activities. Many people find inconvenient to work many hours with a bright screen while others prefer to have more light in order to see things clearly. For that reason multiple themes are supported (see Figure 4.3).



(a) Dark theme



(b) Light theme

Figure 4.3: Img-spy multiple themes support

4.1.1. Explorer

As seen before, *Explorer* examines images in a non-intrusive way. To do that, tsk-js analyze, list and get functions are used. Its layout contains four panels: case structure, properties, active item content and terminal. The size of all panels can be changed to let user feel more comfortable with the interface.

4.1.1.1. Case structure

All folders inside the selected case folder are shown using a tree in this panel (Figure 4.4).

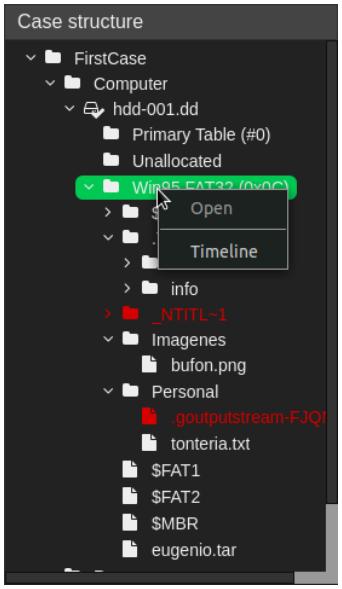


Figure 4.4: Explorer case structure panel

The *Explorer* is always watching case folder file system and each change is detected launching an action to updates the application global state refreshing the case structure tree.

If one file has the extension *.dd*, function tsk-js analysis is executed in parallel and the image digest is computed. Once the proper hash is provided on the image properties panel (Chapter 4.1.1.2.), the content inside the image is added into the case tree using tsk-js list function. Items in red color are deleted.

Each item tree has a context menu. All files can be opened with the OS preferred applications with the open option.

Folders inside an image also have an option to generate a timeline. In case of a disk image, partitions are treated as directories.

Finally, files have an option to export its content using tsk-js get function.

4.1.1.2. Properties

This panel (Figure 4.5) shows the properties of the item selected on the case structure (Chapter 4.1.1.1.). Image properties are:

Hash This is the only editable field. Is the expected digest of the image.

Type This value is retrieved from the tsk-js analysis. Can be disk or partition.

Partitions If type is disk, it shows up the number of partitions

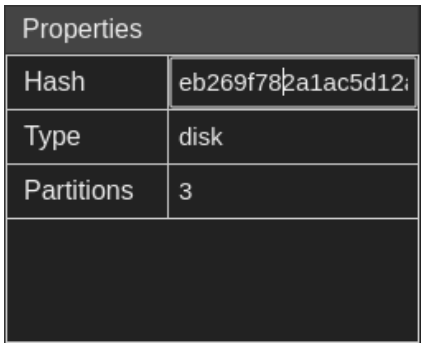


Figure 4.5: Img-spy explorer properties

4.1.1.3. Active item

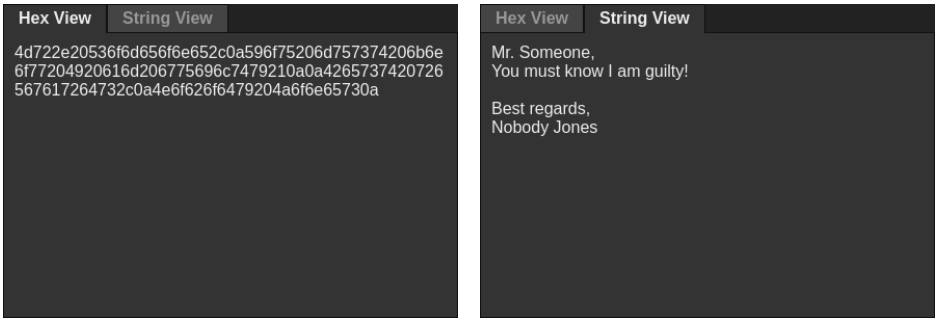
It displays the information of current selected item on the case structure (Chapter 4.1.1.1.).

On the one hand, when a folder is selected, active panel shows the files inside it (Figure 4.6). If the user clicks one row, the current active item will be the clicked one, updating also the case structure (Chapter 4.1.1.1.).

On the other hand, Figure 4.7 shows the active panel when a file is selected. Since the content type of the files is not known, two type of views are supported: hexadecimal and string. It only displays the first thousand characters to not freeze the user interface. A future line of the project is to implement a view with more functionality.

Name	Type
$OrphanFiles	directory
.Trash-1000	directory
_NTITL~1	directory
Imagenes	directory
Personal	directory
$FAT1	file
$FAT2	file
$MBR	file
eugenio.tar	file

Figure 4.6: Img-spy active panel with a folder selected



(a) Hexadecimal view (b) String view

Figure 4.7: Img-spy active panel with a file selected

4.1.1.4. Terminal

There are many asynchronous tasks that keep some time to be executed, for instance, tsk-js analysis. The terminal logs those tasks and possible outputs. Figure 4.8 shows some logs.

Figure 4.8: Img-spy explorer terminal

4.1.2. Timeline

The second tool, *Timline*, generates a table with all the file system actions. Case structure context menu from *Explorer* (Chapter 4.4) has an option to generate those tables. The function tsk-js timeline is executed to get this information.

The generation of timelines is computed on parallel using a child of electron's main process. Some items are received with short periods of time. Instead of delivering one action

each time an item is received, in order to reduce unnecessary fast renderings, timeline results are buffered and delivered each 100ms at most.

4.1.2.1. Timeline list

Timeline list (Figure 4.9) contains all generated timelines. When its generation is finished, the data of the timeline is stored on the current case settings and thus auto-saved.

Each item of the list has a context menu export a timeline using CSV format. The other button is used remove the timeline item from the list and settings.

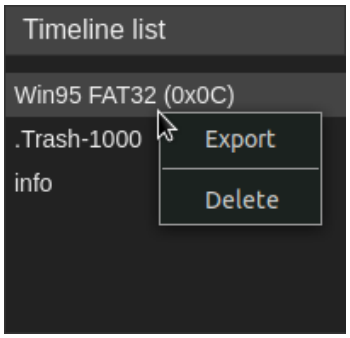


Figure 4.9: Img-spy timeline list

4.1.2.2. Timeline table

The timeline table (Figure 4.10) contains the results of a specific timeline. The user can order columns clicking the headers and move to different pages using previous and next buttons.

Path	Actions	Date	Inode
Imagenes	mo,cr	2017-08-25T13:07:18.000Z	9
Imagenes/bufon.png	ch		10758
Imagenes/bufon.png	ac	2017-08-24T22:00:00.000Z	10758
Imagenes/bufon.png	mo,cr	2017-08-25T13:06:56.000Z	10758
Personal	ch		7
Personal	ac	2017-08-24T22:00:00.000Z	7
Personal	mo,cr	2017-08-25T13:08:22.000Z	7
Personal/.goutputstream-FJQN5Y	ch		10887

Previous Page 6 of 7 Next

Figure 4.10: Img-spy timeline table

The fist column is the item path and the second one are the two first letters of the actions (created, modified, changed and accessed). Then the date of the action, and the last is the inode.

4.1.3. Search

Finally, *Search* uses tsk-js search functionality to let the user look for a string inside the image. This execution is executed in parallel and when all results are retrieved, they are saved on settings, as *Timeline* (see Chapter 4.1.2.).

4.1.3.1. *Search from*

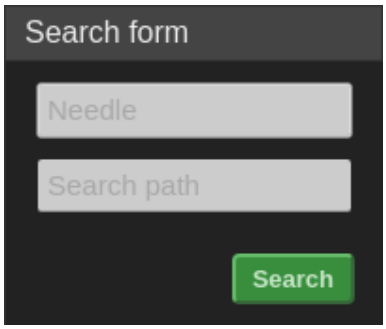


Figure 4.11: Img-spy search form

Search form (Figure 4.11) has two inputs: needle and search path.

Needle Text that will be looked inside the image.

Search path This field is not editable. It has the path of the selected item on the case structure (Chapter 4.1.1.1.).

When user clicks search button, the search process starts to look for matches on background and each time a result is retrieved, it appears on the search results table.

4.1.3.2. *Search list*

All queried search appear on the search list (Figure 4.12). When the look up finishes, it is stored on the current case settings and thus auto-saved.

Each item of the list has a context menu with the option to delete this item from the list and settings.

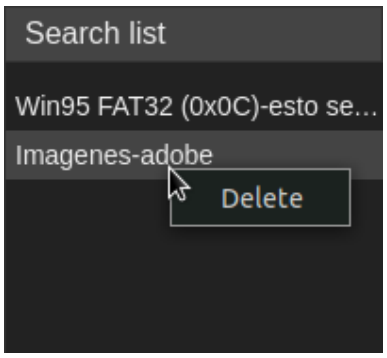


Figure 4.12: Img-spy search list

4.1.3.3. *Search results*

The results of the selected search item are displayed on a table inside this panel (Figure 4.13). This table, as *Timeline* table, also supports custom order by clicking the headers.

Figure 4.13: Img-spy search results

The first column shows the path of the files that contain the needle and how many times it appears. If the toggle button is clicked, the row uncollapses and shows the context and index where the string if appears inside this file.

4.2. Key developments

There were many difficult or interesting parts on the development of this project. Some of them are multiple themes support, file system watcher, auto-save settings, tsk-js workers and resize panels. Also, some plug-ins where used to increase the development speed. The source code will be on this document since it is not its objective. Nevertheless, it can be found on GitHub (https://github.com/FRG-UPC-Deliverables/img-spy).

4.2.1. Multiple themes support

When designing user interfaces, colors are very important. A good consideration is to take profit of SASS define palettes and work with them. That way, to support multiple themes is as easy select some palettes to work. The palette format used is the one defined by Google Materials [26]. Each theme has:

Main palette Is used for the main layout.

Primary palette Define colors for components that must be highlighted.

Warn palette Notifies the user that something is wrong. Red palette is normally used.

Background Default background color.

Foreground Default foreground color.

4.2.2. File system watcher

A React component has been created to launch redux actions notifying file system changes. It uses Node.js package *"chokidar"* to watch those changes efficiently. Watcher is bound using *"componentWillMount"* lifecycle function.

Then, several Redux-Observables epics chain actions, such as launching tsk-js analysis function if the payload contains an image.

4.2.3. Auto-save settings

To implement the auto-save functionality, some epics actions map to update settings model action. The auto-save epic (Listing 4.1) launch another to save them inside the user's disk.

```
1  const autoSaveSettings = (
2      action$: EpicObservable<any>,
3      store: Store<ImgSpyState>
4  ) =>
5  action$
6      .ofType(
7          actions.UPDATE_SETTINGS,
8          actions.UPDATE_SOURCE,
9          actions.UPDATE_THEME)
```

```
10      .debounceTime(100)
11      .mapTo(applySettings({close: false}));
```

Listing 4.1: Mapping example

4.2.4. The Sleuth Kit JavaScript workers

As explained before, JavaScript doesn't support multithreading (Chapter 1.2.). To do not freeze any Electron process while performing tsk-js tasks, main process launch four subprocesses.

Those subprocesses act as workers. Each time a task is received, if any worker is free, it takes this tasks and execute it. If none is free, the task is queued.

4.2.5. Resize-panels

An other React component was created to perform the resizing of panels. This component launch a resize-start Redux action when the slider is mouse down, a resize when the user moves the mouse and a resize-stop when the mouse is released.

Each time those actions are received, the resize-panel refresh its children size using CSS properties.

4.2.6. React plug-ins

All the application forms uses *react-redux-form* package. It provides React components that store the form model inside the redux state. In this manner, the form values are accessible from any other React component.

Timeline and Search result tables are drawn using *react-tables*. This package has React components to render tables very fast. Those tables have order by, pagination and row aggregation functionalities implemented.

4.3. Code quality analysis

There are many ways to check the code quality. One tool that can be used is *sloc*. It can generate detailed information about files inside the project folder. Also, using a Regex, some files can be omitted. The following command generates a CSV with this information.

```
user@host:/home/user$ sloc -d -f csv -e lib src/ > code-quality.csv
user@host:/home/user$
```

Listing 4.2: Code quality CSV extraction using sloc

Using this CSV, some interesting metrics can be computed. Table 4.1 show some examples. Expected values are retrieved empirically from some famous softwares [27].

Metric	**Value**	**Expected**
Total files	190	-
Source lines	8779	-
Max. source lines per file	301	< 500
Avg. source lines per file	≈ 46	< 64
Min. source lines per file	1	> 4
Files without comments	70.5%	-
Comments / Source	2%	-
Avg. consecutive lines	4	-

Table 4.1: Code analysis metrics

The fist group of parameters give information about the size of the project. Then, regarding lines per file, the project is fulfilling the quality expectations. However, the file with 301 should be restructured to improve even more the code quality.

The third block clearly show that comments must be added. Nevertheless, the average number of lines without space is small. That also improves the code readability.

CONCLUSIONS

The development of a cross-platform digital forensics applications is a long process with many key decisions. The first one is to choose a technological environment. This task is not that easy due to the great amount of very-different solutions. But, nowadays web technologies are standing strong in terms of cross-platform compatibility. **A well-known solution** that many companies are using today is **Electron** [8], which uses Chromium as library to instantiate Chrome windows.

Once the work environment is well-defined, an overview of the application should be provided. The software architecture defines it by using black boxes and defining their interactions. Model-view-controller (MVC) is a mature widely-used architecture. Its main disadvantage is that becomes complex when increasing the number of views with interaction between themselves. Since this is the case of the application we have developed, MVC is not a good architecture in this case.

The direct evolution of MVC is Flux, which was defined by Facebook to fix its scalabilty problem. In order to solve this problem, Flux purposes the use of a unique state to render whole the application. In this project we have used **React-Redux** to build **flux-like architecture** Nevertheless, other libraries, such as Redux-Observables, have been used in order to address React-Redux lacks.

With those good bases, a complete application can be build but there is still a need to implement the computationally complex operations of the digital forensics work flow. Those have to be coded in a low programming language to guarantee a good efficiency. The Sleuth Kit provides a C library that implements those operations. Therefore, **The Sleuth Kit JavaScript, a Node.js wrapper, has been created** to let JavaScript use those operations.

Finally, **Img-Spy, the digital forensics application developed to fulfill the goal of this project, defines three tools**: *Explorer*, *Timeline* and *Search*. Those tools let an investigator to analyze the file system of an image in a non-intrusive way, create a timeline based on actions performed on the disk and search which files contain a specific string.

During the development process, many interesting or difficult tasks where found, for instance, watching changes on the file system of user's computer, performing actions in parallel using JavaScript and defining a user-friendly interface supporting, for example, resize-panels or multiple themes.

Looking ahead, many optimizations and features can be added to Img-spy. First, *Explorer* tool can be improved in terms of performance adding Redux selectors and reducing React rerenders. Also, the hash is always computed without asking the user, consuming too many resources that are not always required.

Timeline tool can be also improved by adding graphs to represent those actions in a more visual way. Add filters can also help investigators to remove useless information. For instance, a useful filter could be to see just the files of a specific search result.

In many cases, search files by name is also needed. Therefore, *Search* tool can implement this operation. More custom searches can be executed using Regex.

There is a saying that "Programming can be fun, so can cryptography; however they should not be combined" [28]. So regarding code quality, comments can be added to help new

developers understand easily how the software works.

Finally, new tools can be added in order to detect mismatches on file extensions or to help investigators to write the final report. Just take always into account that, as Gordon Bell said:

> “Every big computing disaster has come from taking too many ideas and putting them in one place”.

BIBLIOGRAPHY

[1] Wikipedia - mvc. https://en.wikipedia.org/wiki/Model-view-controller. Accessed: 15-10-2017. vii, 13

[2] What is haxe. https://haxe.org/manual/introduction-what-is-haxe.html. Accessed: 13-10-2017. ix, 3, 6, 7

[3] Tsk tool overview - sleuthkitwiki. https://wiki.sleuthkit.org/index.php?title=TSK_Tool_Overview. Accessed: 19-10-2017. ix, 23

[4] Collective work of all DFRWS attendees. Road map for digital forensic research. In *A Framework for Digital Forensic Science*, 2001. 1

[5] Roslan Ismail Yunus Yusoff and Zainuddin Hassa. Common phases of computer forensics investigation models, June 2011. 1

[6] Encase forensic. https://www.guidancesoftware.com/encase-forensic. Accessed: 11-10-2017. 2

[7] The sleuth kit. http://www.sleuthkit.org/. Accessed: 11-10-2017. 2, 23

[8] Electron web page. https://electron.atom.io/. Accessed: 13-10-2017. 2, 5, 7, 39

[9] Node.js. https://nodejs.org/en/. Accessed: 22-10-2017. 2, 5

[10] Qt framework. http://www.qt.io/what-is-qt/. Accessed: 12-10-2017. 3, 4

[11] Welcome to python.org. https://www.python.org/. Accessed: 21-10-2017. 3

[12] Javascript - wikipedia. https://en.wikipedia.org/wiki/JavaScript. Accessed: 21-10-2017. 3

[13] What is java. https://www.java.com/en/download/faq/whatis_java.xml. Accessed: 21-10-2017. 3

[14] Ss Um An and Guido Van Rossum. Python for unix/c programmers copyright 1993 guido van rossum 1. In *Proc. of the NLUUG najaarsconferentie. Dutch UNIX users group*, 1993. 4

[15] Securing your web browser. https://www.us-cert.gov/publications/securing-your-web-browser. Accessed: 22-10-2017. 5

[16] Nw.js web page. https://nwjs.io/. Accessed: 13-10-2017. 6

[17] J. Gosling, B. Joy, G. Steele, G. Bracha and A. Buckle . *The Java® Language Specification - Java SE 8 Edition*. Oracle America, 2015. 6

[18] Typescript - javascript that scales. https://www.typescriptlang.org/. Accessed: 14-10-2017. 10

[19] Gulp.js. https://gulpjs.com. Accessed: 14-10-2017. 11

[20] Wikipedia - kiss principle. https://en.wikipedia.org/wiki/KISS_principle. Accessed: 15-10-2017. 13

[21] T. Reenskaug. The model-view-controller (mvc) - its past and present, June 2011. 13

[22] React. https://reactjs.org/. Accessed: 15-10-2017. 14

[23] Redux - prior art. http://redux.js.org/docs/introduction/PriorArt.html. Accessed: 15-10-2017. 15

[24] Redux - three principles. http://redux.js.org/docs/introduction/ThreePrinciples.html. Accessed: 15-10-2017. 15

[25] Redux - prior art. https://github.com/nodejs/node-gyp. Accessed: 21-10-2017. 27

[26] Color - style - material design. https://material.io/guidelines/style/color.html. Accessed: 21-10-2017. 35

[27] Programming practices. https://softwareengineering.stackexchange.com/questions/66523/how-many-lines-per-class-is-too-many-in-java. Accessed: 24-10-2017. 36

[28] Steve McConnell. *Digital Forensic Science - Issues, Methods, and Challenges*. Jenson Books Inc, 1995. 39

APPENDICES

APPENDIX A. ELECTRON HELLO WORLD

This example is based on the electron quick start.

```
<html>
<header>
    <title>Hello world</title>
</header>
<body>
    Hello world!
    Node version is:
        <script>document.write(process.versions.node)</script>
    <script>
        // Render process has access to Node.js modules
        const os = require('os');
        const {remote} = require('electron')
        const {Menu, MenuItem} = remote

        console.log(os.platform());

        const menu = new Menu()
        menu.append(new MenuItem({label: 'MenuItem1', click() {
            console.log('item 1 clicked') }}))
        menu.append(new MenuItem({type: 'separator'}))
        menu.append(new MenuItem({label: 'MenuItem2', type:
            'checkbox', checked: true}))

        window.addEventListener('contextmenu', (e) => {
            e.preventDefault()
            menu.popup(remote.getCurrentWindow())
        }, false)
    </script>
</body>
</html>
```

Listing A.1: Electron index HTML file

```
// menu.js
const {app, Menu} = require('electron')

const template = [
  {
    label: 'Edit',
    submenu: [
      {role: 'undo'},
      {role: 'redo'},
      {type: 'separator'},
      {role: 'selectall'}
    ]
  },
  {
    label: 'View',
    submenu: [
      {role: 'resetzoom'},
      {role: 'zoomin'},
      {role: 'zoomout'},
```

```
      {type: 'separator'},
      {role: 'togglefullscreen'}
    ]
  }
]

module.exports = Menu.buildFromTemplate(template)
```

Listing A.2: Electron menu creation

```
// main.js
const {app, BrowserWindow} = require('electron')
const path = require('path')
const url = require('url')

const menu = require("./menu")

// Keep a global reference of the window object, if you don't, the window will
// be closed automatically when the JavaScript object is garbage collected.
let win

function createWindow () {
    // Create the browser window.
    win = new BrowserWindow({width: 800, height: 600})
    win.setMenu(menu);

    // and load the index.html of the app.
    win.loadURL(url.format({
        pathname: path.join(__dirname, 'index.html'),
        protocol: 'file:',
        slashes: true
    }))

    // Emitted when the window is closed.
    win.on('closed', () => {
        // Dereference the window object, usually you would store windows
        // in an array if your app supports multi windows, this is the time
        // when you should delete the corresponding element.
        win = null
    })
}

// This method will be called when Electron has finished
// initialization and is ready to create browser windows.
// Some APIs can only be used after this event occurs.
app.on('ready', createWindow)

// Quit when all windows are closed.
app.on('window-all-closed', () => {
    // On macOS it is common for applications and their menu bar
    // to stay active until the user quits explicitly with Cmd + Q
    if (process.platform !== 'darwin') {
        app.quit()
    }
```

```
45 })
46
47 app.on('activate', () => {
48     // On macOS it's common to re-create a window in the app when the
49     // dock icon is clicked and there are no other windows open.
50     if (win === null) {
51         createWindow()
52     }
53 })
```

Listing A.3: Electron main

APPENDIX B. THE SLEUTH KIT JAVASCRIPT TYPINGS

A typings file is a TypeScript file that only contains types. It is used by TpyeScript to detect type mismatches.

```
1  declare module 'tsk-js' {
2      export class TSK {
3          constructor(imgfile: string);
4
5          analyze(): ImgInfo;
6          list(opts?: TskOptions): Array<ImgFile>;
7          get(opts?: TskOptions): Buffer;
8          timeline(cb?: TimelineCallback, opts?: TskOptions):
               Array<TimelineItem>;
9          search(needle: string, cb?: SearchCallback, opts?:
               TskOptions): void;
10     }
11
12     export interface ImgInfo {
13         type: "disk" | "partition";
14         partitions?: Array<PartitionInfo>;
15     }
16
17     export interface TskOptions {
18         imgaddr?: number;
19         inode?: number;
20     }
21
22     export interface PartitionInfo {
23         description: string;
24         start: number;
25         end: number;
26         size:number;
27         hasFs: boolean;
28     }
29
30     export interface ImgFile {
31         path: string;
32         name: string;
33         allocated: boolean;
34         type: "directory" | "virtual" | "register" | "unknown";
35         inode: number;
36         hasChildren?: boolean;
37     }
38
39     export type DiskAction = "access" | "modify" | "creation" |
           "change";
40     export type TimelineCallback = (list: Array<TimelineItem>) =>
           void;
41     export type SearchCallback =
42         (file: ImgFile, context: Buffer, index: number) => void;
43
44     export interface TimelineItem {
45         path: string;
```

```
46          name: string;
47          allocated: boolean;
48          type: "directory" | "virtual" | "register" | "unknown";
49          inode: number;
50          actions: Array<DiskAction>;
51          date: Date;
52      }
53  }
```

Listing B.1: The Sleuth Kit JavaScript typings