SEARCH STRING SUPPORT
FOR LAW ENFORCEMENT AGENTS

BY

IGNACIO PÉREZ-IBÁÑEZ

A THESIS SUBMITTED
IN PARTIAL FULFILLMENT
OF THE REQUIREMENTS
FOR THE DEGREE OF
MASTER OF SCIENCE IN COMPUTER SCIENCE

UNIVERSITY OF RHODE ISLAND

2011
ABSTRACT

Most digital forensic analysis tools allow the specification of search strings as *regular expressions* and the search for all terms included in keyword lists (for example, slang terms that criminals use to refer to certain drugs). Unfortunately, due to the combination of characters and meta-characters used in regular expressions, many agents have problems writing them. Also, it can take them years to build a solid, complete keyword list, adding a few new terms with each case that they investigate.

I created a repository so that law enforcement agents can create, access, download and comment on validated search string expressions and keyword lists. In this project I collaborated with the team from the URI Digital Forensics Center. While the DFC was in charge of developing a regular expression generator, I was fully responsible of creating the repository, which included choosing the software engineering methodology to follow, designing and creating the database, designing and implementing the repository, developing a test plan and testing the deliverable. My repository is flawlessly integrated with this tool and agents can easily upload new search strings. This project has followed sound software engineering practices and the repository has been deployed in the Electronic Crime Technology Center of Excellence (ECTCoE) site of the National Institute of Justice (NIJ).
# TABLE OF CONTENTS

ABSTRACT ................................................................................................................. ii

TABLE OF CONTENTS .......................................................................................... iii

LIST OF FIGURES ................................................................................................ iv

1. INTRODUCTION .................................................................................................. 1

2. BACKGROUND .................................................................................................... 6

3. APPROACH ......................................................................................................... 16

3.1 PRELIMINARY WORK ......................................................................................... 16

3.2 SOFTWARE ENGINEERING METHODOLOGY .............................................. 17

3.3 DRUPAL MODEL ............................................................................................. 19

3.4 DATABASE DESIGN AND CREATION ........................................................... 22

3.5 CODING ............................................................................................................ 24

3.6 TESTING .......................................................................................................... 25

4. RESULTS ............................................................................................................. 28

5. PRODUCT EVALUATION ...................................................................................... 37

6. CONCLUSION ..................................................................................................... 40

7. APPENDIX 1: TEST PLAN .................................................................................... 43

8. BIBLIOGRAPHY .................................................................................................. 61
# LIST OF FIGURES

<table>
<thead>
<tr>
<th>FIGURE</th>
<th>PAGE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Figure 1: Agile Development Model Workflow</td>
<td>13</td>
</tr>
<tr>
<td>Figure 2. Development of a product using the Agile Development Approach</td>
<td>14</td>
</tr>
<tr>
<td>Figure 3: Original Drupal concept model</td>
<td>21</td>
</tr>
<tr>
<td>Figure 4: Data structure diagram for LESSA’s database</td>
<td>23</td>
</tr>
<tr>
<td>Figure 5. LESSA’s Keyword List Taxonomy (partial sample)</td>
<td>30</td>
</tr>
<tr>
<td>Figure 6. Subcategory view with option to add a new keyword list</td>
<td>31</td>
</tr>
<tr>
<td>Figure 7. System to select keywords to be exported (partial view)</td>
<td>31</td>
</tr>
<tr>
<td>Figure 8. Thumbs up / thumbs down voting system (partial view)</td>
<td>32</td>
</tr>
<tr>
<td>Figure 9. Form to add terms to an existing keyword list</td>
<td>33</td>
</tr>
<tr>
<td>Figure 10. Confirmation form after adding keywords to an existing keyword list (one item was successfully added, two were already in the existing list)</td>
<td>33</td>
</tr>
<tr>
<td>Figure 11. Form to add a new keyword list</td>
<td>34</td>
</tr>
<tr>
<td>Figure 12. Sample comment</td>
<td>35</td>
</tr>
<tr>
<td>Figure 13. Search expression generator with option to submit an expression to the repository</td>
<td>36</td>
</tr>
</tbody>
</table>
1. INTRODUCTION

During the analysis phase in a digital forensics investigation, law enforcement agents use forensics tools such as FTK, Encase or X-Ways Forensics\(^1\) to search suspects’ computers and storage media looking for evidence. These tools allow an agent to enter search strings as regular expressions, and then use those search strings to return matches in the data. Regular expressions are a formal syntactic language used in computer science and mathematics to specify classes of strings. They employ meta characters to specify sets of actual characters. For instance, the following:

```
[a-z#~\_!\#\$\^\%\*\(\)\-\+\@][a-z#~\_!\#\$\^\%\*\(\)\-\+\@]*(com)|(biz)|(edu)|(gov)|(info)|(mil)|(net)|(org)|(tv)|(uk)|(jp)
```

is an EnCase GREP regular expression to capture the class of strings for most valid email addresses. One can create a wide variety of regular expressions, such as expressions to search for telephone numbers, dates, social security numbers, license plates, employee id numbers, etc.

Specifying regular expressions involves learning a language of meta-characters that many law enforcement agents find difficult — particularly when trying to form effective strings that do not cause an overwhelming number of false positives or do not miss evidence—. An investigator would use a regular expression to search for all files that contain a specific type of content in a storage media. The above-mentioned search tools, would perform a low level search on binary data of the media under analysis, finding not only the evidence that is stored in files, but also that which is kept in fragments of deleted files, in file headers, etc. Unfortunately, this kind of search is

\(^1\) Forensic Toolkit (FTK) is a software suite developed by Access Data; Encase is a product
extremely time-consuming and a poorly designed regular expression could mean a
significant waste of time and resources (as it will monopolize a powerful analysis
station with very expensive software for long periods of time).

There are common regular expressions that are useful in different general types
of investigations (e.g. common strings for cases focused on drug trafficking, for
money-laundering, for child pornography, etc.). Agents do not need to recreate those
continuously, but could benefit from a repository where common and well-designed
regular expressions are available for them to use. My project is based on the belief
that allowing the law enforcement community to share these strings so that each
individual agent does not have to continuously re-create them would increase the
efficiency and effectiveness of their investigations. This belief comes from the
experience of the URI Digital Forensic Center (DFC) team. Since the URI Faculty
Senate approved the Graduate Program in Digital Forensics in 2006, the DFC has
collaborated with law enforcement agencies for a mutually beneficial symbiosis. The
DFC also considers law enforcement agents as the ultimate end-users of the results of
the group's research and, as such, excellent resources for ideas to further develop and
for future initiatives. This feedback has allowed us to find extremely valuable
information about the needs of such agencies and the agents working in them.

Many of URI's students reported that during their internships with law
enforcement agencies they had spent a great deal of their time working with regular
search expressions. Investigators often ask them to create such expressions or to
modify the ones that they had already created in order to produce more efficient
results and avoid "false hits." Law enforcement agents either lacked the knowledge
of the required syntax or found it too hard to remember (unless they used it constantly). Unfortunately, conducting a low level search on binary data is quite time-consuming. Using a poorly designed or a poorly written regular expression can either not produce the expected results by missing evidence or producing a high number of false positives.

Another powerful choice that these specialized digital suites offer is the option to look for specific terms included in a keyword list. The suite explores a device and returns any hit of a word included in the selected keyword list. Beyond a few standard lists that these applications include as models, officers have to create their own lists based on their previous experience, such as terms that they have found in other cases and past investigations. It generally takes years for an agent to create a good set of keyword lists.

I have created a repository so that law enforcement agents can create, access, download and comment on validated search string expressions and keyword lists. In this project I collaborated with the team from the URI Digital Forensics Center (DFC). While the DFC was in charge of developing a regular expression generator, I was fully responsible for creating the repository, which included choosing the software engineering methodology to follow, designing and creating the database, designing and implementing the repository, developing a test plan and testing the deliverable. My repository is flawlessly integrated with this tool and agents can easily upload new search strings. To develop the prototype of the repository, I followed sound software engineering practices. The repository has already been deployed, is being used by law enforcement agents and is accessible from the Electronic Crime Technology Center of
Excellence (ECTCoE) portal (http://ectcoe.net), in the area of resources, under the section “On-line interactive tools.” The repository has been deployed under the name "Law Enforcement Search String Assistant" (LESA) (accessible through the site: http://ectcoe.net/lessa/). With LESA, agents can access and share keyword lists relevant in cases focused on drug trafficking, child pornography, fraud, and most other criminal investigations, which include typical and slang words, words from foreign languages, etc. They can also access the Regular Expression Generator and the regular expressions stored in the repository. Some of the options that agents have available include:

- Browse keywords and / or regular expressions by crime category.
- Search for stored keywords and / or regular expressions using the LESA Search Engine.
- Download keywords as a text file ready to import into EnCase, FTK, and other digital forensics tools.
- Share their keywords with others by adding them to existing lists or creating new keyword lists.
- Vote on the effectiveness of keywords to help establish which keywords work best.
- Comment on keywords to guide others in their use.

The rest of this thesis is divided into five sections. Section 2, “Background,” provides the reader with the background information about this project. Not only does it explain the process that originated the idea to create this product and the previous work that has been done in this area, but it also provides information about the
backbone technologies through which I constructed the final product. Section 3, Approach, describes the methodology I used in order to complete the project, the different steps that I followed in each phase of the project, making references to the software engineering practices that I took into account and observed. This section also discusses the integration of different tools under a common interface. Section 4, “Results,” describes in detail the final product, the deliverable, and the capabilities that it offers to law enforcement agents. Section 5, “Product Evaluation,” provides an assessment of the quality of the tool and discusses briefly possible future developments. Section 6, “Conclusion,” summarizes the main ideas and issues discussed in this paper, underlining the usefulness and relevance of the tool that we developed. Following this section (Section 7) we present an appendix, the testing document that I used to test our product. The paper concludes with a list of all of the references cited (Section 8).
2. BACKGROUND

During the Fall 2006 semester, the students taking the course CSC 305 "Software Engineering" with Dr. Fay-Wolfe, worked on creating a GREP expression generation tool that would create regular expressions that could be used with the software package EnCase. Subsequently, in 2008 the DFC at URI received a grant from the National Institute of Justice (NIJ) to build:

1. An automated regular expression generator for computer forensics analysis tools (e.g. a generator for EnCase, FTK, X-Ways) that allows investigators to type in simple English keywords and choose options for common variations (e.g. surrounded by blank space, plurals, non-case sensitive).

2. A common, shared repository where investigators can lookup, search, and add certified regular expression search strings by category.

My research project grows out of the second part of this grant. As mentioned in Section 1, I built the repository for keyword lists and certified regular expressions. While developing the product I had in mind that it was going to be integrated in the Electronic Crime Technology Center of Excellence (ECTCoE) portal. The ECTCoE was created by the National Institute of Justice (NIJ) and its mission is to provide state and local law enforcement with training, electronic crime and digital tools and technology to increase their success rate in investigations involving electronic crime and digital evidence, and achieve the highest possible prosecution to conviction rate. In order to achieve these goals, the ECTCoE partners with other law enforcement
agencies, academia and the private sector. Through these collaborations, it tries to add value to projects funded with grants of the NIJ to develop training and tools.

The repository that I created has been deployed under the name Law Enforcement Search String Assistant (LESA). Through it, Law Enforcement agents can access a Regular Expression Generator developed by Peter Cho. Thanks to this generator, law enforcement agents can create regular expressions using a simple, intuitive interface to be used with EnCase and/or FTK. They can also generate sample strings that would match the created regular expression to make sure that it would produce the expected results. Finally, they can test their expressions with sample texts to see what text the expression would locate in the evidence before committing to a time-consuming search of an entire digital media using the forensic tool of their choice. While this is a completely different tool, the Regular Expression Generator was developed concurrently with the repository.

Our repository is built on two main pillars: PHP, and a mySQL database. PHP is a cross platform high-level imperative scripting language\(^2\) that is especially suitable for the construction of dynamic web pages. Some of the peculiarities of this language include: a) the reduced scope of the variables by default; b) being a loosely typed language [2]. In general, the scope and availability of variables is very restricted in PHP. The value assigned to a variable is only present within the function or script where such a variable is defined. If we want to make the value of a variable available to other functions or scripts than the one where it is declared, we would need to

\(^2\) A high-level programming language hides the management of CPU operations (such as memory access and allocation or management of the scope) from the programmer, which makes the development of a program easier and the code more understandable. An imperative language is a language where a series of statements change a program state. A program is built by the combination of a series of subroutines [1].
specify that we want to make such variable global. PHP has several predefined variables called superglobals, many of which are actually an array of other variables. Some of the most common ones are $_GET, $_POST, $_COOKIE, $_SERVER, or $_SESSION. Whenever possible we use superglobal variables to reduce the likelihood of user-injected input. As PHP is a loosely typed language, the data type of a variable is automatically assigned to it at the time data is assigned to it. Despite this, PHP offers the programmer the option to explicitly control the data type of a variable by either using the function settype() or by casting the variable. The principal difference between these two methods is that while settype() actually changes the type of an existing variable, casting the variable will produce a copy of it, leaving the original untouched. Due to the nature of our program, it rarely requires that we performed any data type control.

For the database for our repository, we chose MySQL, the world's most popular open-source relational database management system (RDBMS) according to its developers' website. It is used by some of the largest and most powerful companies in the world, including Facebook, Google, Adobe, Alcatel Lucent and Zappos. It is a high performance, very reliable and easy to use system. MySQL comes with no GUI tools to administer MySQL databases or manage data contained within. We found using the included command-line tools tedious and very time-consuming. Kevin Bryan, server administrator, installed and configured PHPMyAdmin on a server (London) in URI's Department of Computer Science. Thanks to this software package it is much easier to manage MySQL databases, build database structure, and work with data records, and it eliminates the tediousness of typing the MySQL commands.
MySQL is one of the pillars of the widely used LAMP (acronym for "Linux, Apache, MySQL, Perl/PHP/Python") web application software stack and it runs on more than 20 platforms including Linux, Windows, Mac OS and Solaris, making it one of the most flexible RDBMS available. This helped us to create a very portable product in case, in the future, it needs to be moved to a different server.

During this project, I followed solid software engineering techniques. I studied and analyzed three well-known approaches to software engineering: the Waterfall method, the Unified approach and the Agile development model [3].

The waterfall model follows a linear and sequential approach to software design and product development. This model divides the work of creating a system into very different, well-defined and clearly separated phases. In order to move to the next phase one must have completed all the required work in the previous one. If during any step of the process one discovers an error, new development of the product should stop. The errors need to be corrected in all the previous phases before the work can move on. There are seven phases in the Waterfall model. Those are:

- **Definition study / analysis.** In this part the team conducts research and studies in order to get a clear vision of the purpose of the final product.

- **Basic design.** The purpose of this phase is to develop documentation (the most important of which is the Requirements Document [4][5]) that collects in written form all of the requirements that the program needs to fulfill.
• **Technical design / detail design.** In this phase the team studies how the required functionality will be achieved in the final product. The work is divided in engineering units such as programs, modules and functions.

• **Construction / implementation.** This phase is where the actual coding takes places.

• **Testing.** This phase is divided into two different parts: the development of the testing document and the actual testing of the product. The testing document specifies how the product is going to be tested and the criteria that it needs to meet in order to meet the requirements stipulated in the requirements documents. During the actual testing of the product the team performs all of the tests listed in the testing document and makes sure that all of the required features work as expected.

• **Integration.** During this phase the product is deployed and integrated with other existing products.

• **Management and maintenance.** The purpose of this phase is to make sure that the system continues to work as performed.

The second model that I studied was the Unified Process. This model is considered to be more flexible than the Waterfall method, while still keeping a strong structured approach. In the Unified Process these phases will overlap. Generally speaking, the amount of documentation that is produced by followers of the Unify Process is less than that produced by those who adopt the Waterfall method. Despite
that, a considerable amount of time, work and effort goes to producing written documents. It is divided into 4 sequential phases:

- **Inception Phase**: It is the first of the four phases of the Unified Process and it is where the genesis of the process takes place: the idea of the product first takes shape. During this stage, one must prove that the product is necessary (by analyzing the current situation and listing the improvements and benefits that the product will generate), determine the scope of the project, outline use cases, and set the initial requirements.

- **Elaboration Phase**: It is the second phase, during which the vast majority of system requirements must be determined. At the end of this phase one must produce a detailed project plan which should include several documents (requirements, conceptual diagrams, data flow diagrams, etc.) for the entire construction phase. A key part of this phase is to identify who are the stakeholders for the project.

- **Construction Phase**: It is when the coding actually takes place. The product is fully tested based on the test plan (that was created in the elaboration phase) before it is released to the users in the transition phase. If the appropriate documents have been created with enough detail during the elaboration phase (requirements, conceptual diagrams, data flow diagrams, etc.) this phase should be quite trouble-free.

- **Transition Phase**: It is when the product is released to the users. Normally there is a beta release to a limited number of specialized
users who provide feedback based on their trials of the product. Based on that feedback, the product is improved. Training users, developing tutorials, etc. are also part of this phase.

The third and last model that I analyzed was the Agile Development model. The defenders of this approach argue that it is the only software engineering methodology that really focuses on customer satisfaction and on creating a final product that completely satisfies the stakeholders. In fact the Agile Development Manifesto underlines 4 principles in their rationale. They value:

- Individuals and interactions over processes and tools;
- Working software over comprehensive documentation;
- Customer collaboration over contract negotiation;
- Responding to change over following a plan.

Programming teams that follow this approach don’t spend so much time planning or creating documentation; rather they center their efforts on the construction phase of the project. They hold short meetings very regularly (on most occasions they hold a short meeting at the beginning of the working day) in order to update the other members of the team on their progress and to make sure that everybody is moving in the same direction. Such meetings increase collaboration and interaction among the members of a team.

Followers of the Agile Method break tasks into small increments. Once a task is completed and there is a working product, the product is presented to the customers in order to receive feedback. The future development of the product (and the

---

3 From: http://agilemanifesto.org/
development of new features) depends on the feedback the team receives from their stakeholder. In the Waterfall and Unified Process methodologies, all of the requirements are collected at the beginning of the project and in order to change them, one needs to create new documentation. While followers of the Waterfall and the Unified Process do not welcome changes in the requirements, developers that follow the Agile model are open to frequent changes on the requirements. The following figure is a graphical representation of the workflow of the Agile Development model:

Figure 1: Agile Development Model Workflow$^4$.

While this figure represents very clearly the Agile Development Model and Workflow (where processes are often repeated), we must underline that it refers to the

$^4$ From: (http://scm-l3.technorati.com/10/08/14/16157/agile-posters-banners-workflows-best-practices2.jpg)
workflow and the processes. If we wanted to represent the development and
deployment of the actual product, it would be more appropriate to use a spiral. The
spiral does not transmit the idea that we are continuously returning to the same
originating point, but that we are moving forward, increasing our field of action while
integrating previous results.

Figure 2: Development of a product using the Agile Development Approach

Due to the nature of the product, I did not opt for a rigid, sequential waterfall
method, with heavy dependency on documentation and very clear separation about the
different programming phases. As I will further explain in section 3.2 Software
Engineering Methodology, I decided to use a modified version of the Agile
development approach where we also adopted some simplified features of the Unified
process [3]. During most parts of the process, I met frequently (most of the time at
least once a week) with Dr. Fay-Wolfe and others member of URI’s DFC, modified
my product based on my stakeholders’ feedback and continuously listened to their
input, even when that meant adding extra features or radically abandoning the original idea.
3. APPROACH

3.1 PRELIMINARY WORK

Our first steps into this project were based on the previous experience of the Spring CSC 305 class mentioned in section 2 (Background). Peter Cho (main developer of the regular expression generator), Dr. Vic Fay-Wolfe (principal investigator for the NIJ grant), Jacob Fonseca (member of the DFC staff who was very involved in the above-mentioned CSC 305 project) and I met to discuss how to approach the development of the repository. I first reviewed the documentation created by the CSC 305 class and analyzed their product. As the main goals of the project remained the same, I decided to reuse their Requirements document as a starting point. I also had in mind the funded grant proposal of URI's DFC. It is important to understand that a requirements document serves to provide a clear idea of what functionality is expected from the system, but not a description of how to achieve the expected functionality. That is, a requirements document is not the same as a programming guide or a development plan. Speaking in plain English, this kind of document establishes the final destination for the system, but does not provide a map or directions on how to reach it. Knowing where we want to go is vital in order to start planning how to reach the destination; consequently, the creation of a System Requirements Specification always occurs during the first stages of the elaboration phase of any software engineering project.

Unfortunately, other than the general requirements for the final product, none of the code or any other documentation was useful for us. Their work was incomplete,
lacked clarity and used technologies and an approach that I considered inappropriate. Consequently, I had to begin the project without the benefit of that work as a basis upon which to build.

3.2 SOFTWARE ENGINEERING METHODOLOGY

Once we had a clear idea of what we wanted to build, I had to decide what software engineering methodology to follow. In order to do so, I analyzed the three different approaches to software engineering to which I referred in section 2: the Waterfall model, the Unified Process and the Agile model.

I decided to follow an adaptation of the Agile Development model [3]. I felt that the 4 above-mentioned (Section 2) statements of the Agile Manifesto summarized very well our approach to the development of the product. I wanted to create a well-designed, useful tool that facilitated the work of Law Enforcement Agents and the creation of a virtual community. My focus was on the final product, not on the creation of extensive documentation. My approach needed to be flexible in order to be able to quickly adjust to requirement changes based on the feedback of my clients.

I also wanted to achieve high levels of cooperation and integration not only with others members of the DFC group, but also with our final costumers. Consequently, we started meeting regularly, setting small short-term objectives and milestones, and continuously testing our code. At different moments of the project, we asked for feedback from members of the ECTCoE and local law enforcement agencies. On at least three different occasions we presented a partial product to the members of the DFC, members of the NIJ and members of the State Police.
Department. Dr. Fay-Wolfe also presented our work at the NIJ headquarters in DC. We would have liked to have received more input and feedback from the funding institution of this project, the NIJ, but unfortunately it was not physically possible. After each presentation we revisited our product to integrate feedback from the DFC group and the different involved parties.

I reused some elements from the Waterfall model and the Unified Process, for example we created a very detailed Test plan (see Appendix A). The reason why I decided to write a detailed Test Plan was that I considered it extremely important to deploy a reliable and secure product in which all of the features worked as expected, and were easy to use. Although in the end I performed most of the testing myself, the original idea was to have the product tested by me in conjunction with other members of the DFC. A test plan would have ensured homogeneity and uniformity in the testing procedures.

Other than that, these two models (the Waterfall model and the Unified Process approach) were not appropriate for our project. The CSC 305 group had followed the waterfall method to build their product. This approach proved to be too rigid for our needs. Setting all of the requirements at the beginning of the project was impractical as it assumed that stakeholders had a perfect idea of what they wanted. In our case, the DFC presented a proposal to the NIJ and they decided to fund it. Thus, the requirements did not come from the stakeholders, but rather from the developers. On the other hand, following the Waterfall model or the Unified process would have produced a tremendous amount of documentation that our costumers were not interested in receiving / reviewing.
3.3 DRUPAL MODEL

Our first approach to this project was to use an existing open source Content Management System (CMS) to create a model and present it to our clients — the people from the NIJ and law enforcement agents — for a preliminary review.

We explored different possibilities and we found 3 CMSs: Drupal\(^5\)[6], Moodle\(^6\) [7], and Joomla\(^7\) [8]), that due to their characteristics, structure and installation requirements could fit our needs. After further research [9, 10, 11] and comparison among the CMSs, I decided to use Drupal (version 6 at the time)\(^8\) as our platform for the first model. The decision was based on the following arguments:

— Drupal’s status as an established CMS: For the last seven years (2005-2011), this platform has been selected to participate in the Google Summer Code, a global program that offers student developers stipends to write code for various open source software. The recognition by Google, the strong position of this CMS on the web and the huge number of contributors are indicators of the strength of the software and indicates that this product will be maintained in the future.

— The large quantity of documentation and training materials that have been produced and is available for this CMS. Many major publishers have books available on Drupal focusing on a wide variety of topics, from

\(^5\) Dries Buytaert is the founder and lead of the Drupal project. It has over 4,690 developers worldwide.

\(^6\) The Moodle project, founded and lead by Martin Dougiamas, is a free, PHP-based CMS system especially designed with educational purposes in mind. It provides a series of tools that can be used when a constructivist approach to education is used in a classroom setting.

\(^7\) According to its developers, 2.7% of the entire web is powered by Joomla!

\(^8\) Drupal 7.0 beta was released on October 7, 2010. The stable version was released on January 5, 2011.
introduction to the CMS [12, 13, 14] to security [15], to development of new applications and designing interfaces [16], etc.

— Its modularity: Drupal allows the integration of different modules in order to add new capabilities to the basic CMS. There are literally thousands of modules available to integrate into the core system.

— The strong community of developers that supports and develops Drupal: An extremely active community of designer and developers offer their help and time to users and other developers in support forums, blogs, web-pages, etc.

— My previous experience with the CMS: I had developed a site, the e-LRC, where professors of Foreign Languages can share class materials and class plans, exchange opinions, participate in forums, etc.

Our Drupal model was ready by the end of February 2009. After a presentation and a meeting with the DFC staff, we decided what features from the CMS system we wanted to keep for the repository and what other functionality we wanted to add that was not present in the Drupal model. Figure 3 is a screenshot of that model.
For technical and logistical reasons we decided to recreate the system by writing our own code in PHP and MySQL. Customizing Drupal and changing its interface to integrate it into the ECTCoE site proved to be more difficult than what we first had anticipated due to the structure of the package. Also, while many people in the DFC have expertise and knowledge of PHP and MySQL, none have any experience with Drupal. Future developments in the repository and support once it was deployed would be much simpler if more than one person could work on those areas.
3.4 DATABASE DESIGN AND CREATION

The next step was to create the repository using PHP and MySQL. I first created the schema of the database. To do so, first I analyzed our needs, specifically, the data requirements: determining what data needed to be collected as well as the best way to preserve it. The next step was the conceptual database design [17]. In this area we wanted to obtain a comprehensive understanding of the database structure, the interrelationships and constraints. My main priorities in the design of this structure were minimalism, simplicity and understandability. The physical structure of our database was formed by 6 tables and 29 different attributes:

Table AUTHOR

<table>
<thead>
<tr>
<th>AuthorID</th>
<th>Username</th>
</tr>
</thead>
</table>

Table CATEGORY

<table>
<thead>
<tr>
<th>CategoryID</th>
<th>SuperCategoryID</th>
<th>CategoryName</th>
</tr>
</thead>
</table>

Table COMMENT

<table>
<thead>
<tr>
<th>CommentID</th>
<th>AuthorID</th>
<th>KeywordListID</th>
<th>ExpID</th>
<th>Comment</th>
<th>CommentDate</th>
</tr>
</thead>
</table>

Table KEYWORD

<table>
<thead>
<tr>
<th>KeywordListID</th>
<th>KeywordID</th>
<th>Keyword</th>
<th>KeywordLanguage</th>
<th>VotesUp</th>
<th>VotesDown</th>
</tr>
</thead>
</table>

Table KEYWORDLIST

<table>
<thead>
<tr>
<th>KeywordListID</th>
<th>AuthorID</th>
<th>KeywordListDescription</th>
<th>KeywordListDate</th>
<th>CategoryID</th>
</tr>
</thead>
</table>

Table REGEX

<table>
<thead>
<tr>
<th>ExpID</th>
<th>Expression</th>
<th>Title</th>
<th>Tool</th>
<th>AuthorID</th>
<th>Description</th>
<th>VotesUp</th>
<th>VotesDown</th>
</tr>
</thead>
</table>

I followed a top-down design strategy, that is, I started by building a conceptual design schema that contained high-level abstractions and I then applied successive top-down requirements. Each entity has only atomic, single-valued
attributes. Concurrently, I considered the transaction design, that is, I considered the functional characteristics of the transactions that would run on our system.

Please see the data structure diagram of my database created using the Entity-Relationship Model in Figure 4:

![Data Structure Diagram for LESSA's Database](image)

Figure 4: Data structure diagram for LESSA's database

Finally, we worked on our physical database design and creation. As mentioned above, MySQL does not come with GUI, which makes the process of creating the structure and any other required modification of the tables very tedious and repetitive. Kevin Bryan installed PHPMyAdmin, an open source package that provides an interface to interact with MySQL databases, in the servers of URI's Computer Science Department. I used PHPMyAdmin to create the database schema.
and the database files. Since I had already done all of the analysis of the kind of data and designed the schema, this process proved to be straight-forward and relatively easy to complete.

3.5 CODING

Once the database was created, I started working on the PHP code. The objective was to create a tool that would allow law enforcement agents to share keyword lists and regular expressions by easily uploading them to our repository, adding new terms to existing keyword lists, commenting on each other’s contributions, and voting on the expressions. Influenced by the feedback from the NIJ, I decided to drop some features that I had included in the Drupal model. For example, in the prototype I created using Drupal, users could create blogs and wikis, contact each other, upload pictures, etc. The NIJ was not interested in these features and explicitly asked them to be removed from the final product.

When working with PHP, I focused in building a very modular system, trying to make it more easily expandable and to facilitate the reutilization of code. As with every software product, security was a concern. I built a solid, secure system, as transparent to the user as possible (where we hid every implementation detail possible), and where all of the input from users is validated in order to avoid attacks such as SQL injections or Cross-site scripting (XSS) attacks. The result is almost 2,300 lines of code distributed in 23 different files (not including the required files for the generator, CSS styling, etc.). In order to facilitate future developments by other programmers, I commented the code copiously.
Once I had a working model, I populated it with content to make it more valuable to our first users. The content came from a) models included in different digital forensic research tools and b) research on criminal terminology and slang. In this phase, help from Kristina Hatch, Sarah Schultz and Katelyn Felix, student interns at the DFC, was invaluable. While populating the repository some bugs were found that I immediately fixed.

3.6 TESTING

The next step was to create a detailed and strict test plan to make sure that all of the required features worked as expected. My test plan (see appendix 1) addressed three different elements: test coverage, test methods and test responsibilities [18][19]. My repository was tested in a variety of browsers (InternetExplorer, Safari, Mozilla Firefox, Chrome and Opera) and different systems running on different platforms (Windows XP, Windows Vista and Windows 7 and Mac OX 10.4, 10.5 and 10.6) to make sure that all of the features worked independently of the system / browser used. The kinds of tests that we conducted belonged to two different categories:

- Tests that were intended to make sure that the functionality of the repository worked as expected. In order to make sure that the navigability of the site was optimum, I visited all of the links to make certain that none were broken. In order to check the functionality, I completed all of the functions a standard user can perform. In the keyword list repository, I created a new keyword list with words separated by commas, tabs and carriage returns. I tested special cases, such as two words separated by two or more commas, a comma and a
tab, a tab and a carriage return, to verify that our repository would not add those extra characters to the keyword list. Once the keyword list was created I added extra terms to the existing list. I visited existing keyword lists, voted on the usefulness of the terms and commented on them. I also selected all of the terms, unchecked all of the terms, selected only some of them and exported a list of such terms as a comma separated file on each occasion. For the regular expression generator, I first created a regular expression and then added it to the regular expression generator. All of these tests polluted the database, so once all of the assessments of the functionality of the repository were completed and the results were satisfactory, I cleaned the data.

- Tests that were intended to prove that the code was secure. I tested the security of the code against SQL injections. The functions in place to treat meta-characters as regular characters worked as expected and the attacks were unsuccessful. During the summer of 2010 I worked with a student intern from the DFC, Berkeley Churchill because of his expertise with Cross Scripting XSS attacks. These kinds of attacks allow hackers to inject client-side script in the database. When a visitor performs a transaction with the database, the injected code would run on that individual’s machine. Berkeley attacked the site which allowed us to discover the vulnerabilities. I immediately fixed those vulnerabilities. As with the previous kinds of tests, these “attacks” polluted the database, which I had to clean once again.

Once the system was tested, I made all of the required modifications (items that failed the test plan) and made sure that the system was secure, reliable and free of
programming bugs. Once those bugs were fixed, I re-tested the features that had failed the test plan the first time to make sure that everything met our expectations and worked properly.

Finally, I submitted the code and a copy of my database to the DFC staff, so that the proper style could be applied to the repository and it could be integrated into the ECTCoE portal. The DFC staff was in charge of the integration and making sure that my database, PHP and html code was integrated seamlessly with the other web-applications that they support, avoiding inconsistencies, not only in appearance and functionality of LESSA, but also in the way in which the entire portal works. Additionally, we tried to avoid any unnecessary duplication of data structures, code, features, etc. The final result is that our repository was deployed at the ECTCoE portal and it can be accessed and used at: http://ectcoe.net/lessa/
4. RESULTS

I have created a repository so that law enforcement agents can create, access, download and comment on validated search string expressions and keyword lists. In order to do so, I collaborated with the development team from the URI Digital Forensics Center who developed the regular expression generator. To develop the prototype of the repository, I followed sound software engineering practices. The repository has already been deployed and is being used by law enforcement agents. It is accessible from the website of the Electronic Crime Technology Center of Excellence (http://www.ectcoe.net), in the area of resources, under the section “On-line interactive tools.” The package of the repository integrated with the regular expression generator has been deployed under the name "Law Enforcement Search String Assistant" (LESSA) (accessible through the site: http://ectcoe.net/lessa/).

With LESSA agents can access and share keyword lists for drug trafficking, child pornography, fraud, and most other criminal investigations, which include typical and slang words, words from foreign languages, etc. They can also access the Regular Expression Generator and the regular expressions stored in the repository. Some of the options that agents have available include:

- Browse keywords and / or regular expressions by crime category.
- Search for stored keywords and / or regular expressions using the LESSA Search Engine.
- Download keywords as a text file ready to import into EnCase, FTK, and other forensics tools.
• Share their keywords and with others by adding them to existing lists or creating new keyword lists.

• Vote on the effectiveness of keywords to help establish which keywords work best.

• Comment on keywords to guide others in their use.

The repository is fully integrated with the search string generator, so that once an agent has created a search expression, it can be easily added to the repository. Also, I built on the concept of creating an on-line virtual community for law enforcement agents. Nowadays, the only method by which agents share search expressions and keyword lists is either by personal networking or by participating in specialized forums. The amount of feedback that users can provide in those forums is limited. Many agents are reluctant to share their materials in such forums if they do not personally know other users. The fact that our repository is sponsored by the NIJ serves to build confidence in the system, its security and its capabilities.

In order to facilitate locating keywords and / or regular expressions related with a particular crime category, we created a taxonomy that serves to classify the content. The categories included in our taxonomy are: Homeland Security crimes; Narcotics (crimes related with drugs trafficking and cultivation); Property (crimes against property); Sexual crimes; Violent crimes; White Collar; and Miscellaneous. Each of these categories is divided in different subcategories so that visitors can access the content more easily. Close to each subcategory, between brackets, we display the number of keyword lists available in it. For example, the Homeland Security category includes four subcategories: Cyberterrorism, Espionage, Terrorism and Miscellaneous.
In order to come up with this taxonomy we conducted extensive on-line research, looked at how certain Police Departments are organized and asked for feedback from the members of the DFC and different law enforcement agents. An example of this taxonomy can be found in Figure 5:

**Browse Keyword Lists**

### Homeland Security

- Cyber Terrorism [2]
- Espionage [2]
- Miscellaneous [0]
- Terrorism [10]

### Narcotics

- Cultivation [1]
- Distribution [2]
- Miscellaneous [15]
- Trafficking [1]

### Property

- Arson [4]
- Burglary [2]
- False Pretenses [1]
- Larceny [1]
- Miscellaneous [0]
- Receiving Stolen Goods [1]
- Robbery [1]

Figure 5: LESSA’s Keyword List Taxonomy (Partial Sample)

The main categories are just placeholders, that users cannot access. We tried to avoid a common mistake of many content repositories, that is, presenting the user with a great number of choices that will route their request to a new situation where they will once again be presented with a great number of choices, and so on. Users do not have control over the taxonomy, that is, they cannot add or remove categories.

Once a user enters a subcategory, she will be presented with a list of all of the available keyword lists. From that page, the user can either access an existing keyword list or create a new keyword list (see Figure 6).
Figure 6: Subcategory view with option to add a new keyword list

If the user decides to access an existing keyword list, she will be presented with all of the terms included in it. She then will be able to select all the terms by clicking on the "Check all" button, select only some of the terms by clicking on the selection box close to each term, or unselect all of the terms by clicking on the "Uncheck all" button. Once the user has made her selection of the terms, she will be able to export it to a comma separated file by clicking on the "Export to text file" button (see Figure 7).

Figure 7: System to select keywords to be exported (partial view)
By default, the name of that list is the same as that of the keyword list. A user can vote on the usefulness and appropriateness of a term by clicking on the "thumbs up" / "thumbs down" button next to each term. Once the vote is submitted, it is immediately scored and displayed close to the term, and the vote icons for that term will then be unavailable --grayed out—(see Figure 8).

A user can also add terms to an existing keyword list by clicking on the "Add Keywords" button. If she chooses this last option, a form will appear. In the main area, the user can write new terms by separating them with commas, tabs or carriage returns. Once a language is chosen, the form can be submitted (see Figure 9).
Add keywords to "Cocaine Slang"

Keywords (please enter each term followed by either a comma or a carriage return)

- white powder
- white sugar, white tornado
- wicky stick
- wings
- witch, wollie, woolah woolas
- woolies
- woolly blunts
- working fifty
- working half
- wrecking crew

Select a language
- English

Please sign your contribution (choose a username)

Add to Database

Figure 9: Form to add terms to an existing keyword list

The system will then check if each individual term is already part of the keyword list. If that is the case, the user will be informed that a certain term was already part of the keyword list and that, consequently, it has not been added to the keyword list (see Figure 10). If the term was not in the keyword list, the user will receive a "success" message letting her know that the term will be added to the keyword list as soon as an administrator approves it.

Confirmation Form

Any keywords shown below, as having been successfully added, have been submitted for review. They will be available to the public once they have been approved by the administrator.

- ✔️ Keyword canuto successfully added to the KeywordList.
- ❌ Keyword cartacho already exists in KeywordList.
- ❌ Keyword cannabis already exists in KeywordList.

Return to the Keyword List

Figure 10: Confirmation form after adding keywords to an existing keyword list (one item was successfully added, two were already in the existing list).
If a user decides to add a keyword list to an existing subcategory, she will be presented with a form with the following fields: Keyword list name; Keyword list items; category and subcategory selection (which is automatically pre-populated by the system, but can be modify by the user); and language. If the user selects the name of an existing keyword list, the items will be automatically added to the existing list (as long as they are not already there). As when adding terms to an existing keyword list, a user can add terms separated by commas, tabs or carriage returns (see Figure 11).

**Browse Keywords - Add a keyword**

**Please enter the following information**

Name of the Keyword List (ex. Marijuana Slang):

Keywords (please enter each term followed by either a comma or a carriage return)

Category: Narcotics  
Subcategory: Trafficking  
Select a language  

English  

Please sign your contribution (choose a username)

Add to Database

Figure 11: Form to add a new keyword list.
A user can also share her opinion about a particular keyword list by submitting a comment. Such comments must be signed with a username and have a limited length of 2,000 characters (see Figure 12).

<table>
<thead>
<tr>
<th>Author</th>
<th>Comment</th>
<th>Date</th>
</tr>
</thead>
<tbody>
<tr>
<td>inaki</td>
<td>This is a very useful keyword list. Thank you so much for sharing it with the community</td>
<td>07/15/11 10:48:05</td>
</tr>
</tbody>
</table>

Figure 12: Sample comment.

Through the left navigation menu, law enforcement agents can access a repository of regular expressions. As in the case of the keyword lists, we created a taxonomy to make the content more easily accessible. Due to the nature of regular expressions, the categories in the taxonomy differ significantly from the ones used for the keyword lists. As in the case of keyword lists, law enforcement agents can vote on the usefulness of a specific regular expression and comment on it. Agents can also add their own regular expressions to existing categories.

From the left navigation menu, one can access the Regular Expression Generator developed by Peter Cho. Although I will not describe the generator and its operation in detail, I must underline that, through it, law enforcement agents can generate regular expressions for EnCase and/or FTK from a simple, intuitive interface. Also, they can test their new search strings against the generator, so that they will be able to a) produce sample matching strings; and b) test their regular expressions against sample texts to see if it produces the expected results before performing a time-consuming analysis of the evidence. Once the agent is satisfied with the results,
she can easily upload it to the repository by clicking on the “Submit to database” button (see Figure 13).

Figure 13: Search expression generator with option to submit an expression to the repository.

The main benefit of this system is that it allows law enforcement agents to access resources that other agents have found useful. It will significantly impact their work, as it will simplify the process of obtaining validated keyword lists and search expressions. Also, it will open new venues for agents to collaborate, independently of the agency for which they work. Finally, if used properly, having access to a search expression generator and this repository should impact the productivity and efficiency of the law enforcement agents by reducing the amount of time that the analysis phase of the different confiscated devices requires and by improving the quality of the results.
5. PRODUCT EVALUATION

During the development process the repository has undergone multiple revisions that have proved it to be an easy to use tool, and a solid, reliable product that produces the expected results. The repository benefited not only from the expertise of the development team, but also from the feedback that we received from the members of URI's DFC. We performed different presentations in front of them and released a beta version for testing. Once we integrated all of their feedback into the repository, Dr. Fay-Wolfe gave a demonstration of the product and its capabilities to personnel of the NIJ and the ECTCoE. They were very satisfied with the results. We performed the few modifications that they requested (mostly aesthetic) and the repository was finally released to the ECTCoE portal. Currently, it is available for law enforcement agents to use.

Other than the criteria stated in the Test Document (see appendix 1) that make sure the repository behaves as it is supposed to\(^9\), the metrics to measure the quality of the product are difficult to quantify. Our evaluation of the product focused on 4 areas:

— **User-friendliness**: We focused on creating a repository that is user-friendly and easy to navigate. Unfortunately, there are no quantitative metrics for this area. We made a conscious effort to keep a clean interface, where the information is easily located. Since we were aware that, in the end, our repository was going to be part of the ECTCoE site, we simplified

---

\(^9\) The repository was thoroughly tested before the integration into the ECTCoE site and the code passed all of the tests. The DFC staff was in charge of the integration into the DFC site. I am not responsible for any modification that they had made in order to perform that integration.
the navigation menu, in order for all areas of the repository to be accessed with just a few clicks. The navigation bar at the top of the page allows users to navigate the ECTCoE main categories, while all of the links related with LESSA are in the left navigation menu. In order to make the functionality of the tool more intuitive we drew from design conventions and extended practices in the most commonly used programs, tools and sites.

— Appropriate organization of the content. The taxonomy that we created to access the different keyword lists was carefully designed, so it would allow us to group the keyword lists under main categories, but not creating a web so extensive that information could be difficult to locate. In order to do so we first searched for models on-line and then we requested feedback from the DFC members and law enforcement agents.

— Security and modularization of the code to facilitate maintenance and scalability. Our backbone code written in PHP is very modular, which allows the administrators of the site to easily update it. It also allows great control over the customization of the product and simplifies future development. It is also very secure; we made sure that our code sanitizes every input and escapes any special characters to avoid SQL injections or cross-site scripting (XSS) attacks, that would allow an attacker to inject client-side script into our database. Those special characters, if present, are treated as regular characters. To avoid unnecessary repetition of code, all
of our sanitizing functions are stored in a file (clean.php) and we use the include option of PHP whenever it is necessary to validate input.

— Facilitation of the creation of a law-enforcement agent community. In the repository, users are responsible for the content. I focused on adding enough features that would facilitate access to useful resources and interaction among law-enforcement agents. The system to add content to the repository has been designed to require the minimum possible effort from users in order to increase the exchange of materials. Law enforcement agents can easily upload keywords and keyword lists by copying the content of their existing lists and pasting them in a form field. The voting system allows an easy way to rate keywords that requires virtually no time from users. Thanks to the commenting feature users can provide feedback to the rest of the community on the usefulness of a certain search expression, keyword or keyword list.

My repository scores high in all these categories. The final result is an easy to use and customize, secure, very interactive, web 2.0 tool that fills a void in the broad spectrum of available tools for law enforcement agents.
6. CONCLUSION

I have developed a repository for law enforcement agents to be able to access and share keyword lists and regular expressions. These materials will help reduce the number of false hits and the chances of missing evidence during the analysis phase of an investigation, when searching digital media using specialized digital forensics tools. It will also help create a sense of virtual community, where users can benefit from the feedback of other colleagues.

Future developments of the tool could focus on providing an authentication mechanism that ensures that only law enforcement agents have access to LESSA. This requires the coordination and collaboration of the NIJ and ECTCoE (for example, making sure that the credentials presented by users are legitimate is far beyond the scope of URI's DFC). It would be necessary to find a working and suitable model for authentication, in order to make sure that all users do in fact belong to law enforcement agencies. Currently, when a user adds new content to the repository (for example a word to an existing keyword list), the administrator has to approve it before it is actually added. This is the only way that we can prevent malicious users from polluting the data. Users have no accountability for any of their contributions. Different options were discussed with the ECTCoE personnel, but finally the DFC did not reach an agreement on the best way to address these issues. User-authentication would allow users to create a more defined identity and we could give them more control over the content that they share (for example, a user could erase or modify some of that content, if he wished to do so). It would also reduce the presence of the
tool administrator: currently this person has to monitor every contribution in order to make sure that no ill-motivated users pollute the site with inappropriate content. If all users were confirmed law enforcement agents, such monitoring would become unnecessary.

Another area of future development is with language support. In the current version, when adding terms to an existing keyword list or when creating a new keyword list the user can indicate the language of those terms (among 6 possibilities: English, French, German, Italian, Russian and Spanish). Unfortunately, there is no real language support, as terms from different languages can end up in the same keyword list. Also, in the future developments of the repository, users should be able to restrict their searches by language and choose between a much wider selection of languages.

Our repository comes to fill a void in the array of digital forensics tools available to law enforcement agents. It integrates flawlessly with the Regular Expression Generator developed by Peter Cho. Not only is it easy to navigate and presents the information in a way that makes it easy to locate: it is also easy to customize, update and expand. It is a real Web2.0 tool, where users are in charge of the creation and development of content.

It builds on the idea of a community of users, where everybody benefits from the contributions and comments of the rest of the users. Users can grade each other’s contributions and share regular expressions and keyword lists. They can select specific terms based on their preferences, needs or the feedback received from other users and export it to comma separated text files, that can be easily imported in a
variety of digital forensic tools. They can also upload their existing lists easily, without the need to format them in very complicated ways. One of the main advantages of LESSA is that it has been designed with the final user in mind, building a user-friendly, easy to manipulate, useful tool. The time to learn how to use LESSA is virtually inexistent, as its draws from conventions and interfaces found in the most commonly and extensively used programs and sites.
7. APPENDIX 1: TESTING PLAN

REGEX REPOSITORY TEST PLAN

Test Plan v 1.2

Ignacio Pérez-Ibáñez

September 2010
### 1. Revision Page

<table>
<thead>
<tr>
<th>Date</th>
<th>Revision</th>
<th>Description</th>
<th>By</th>
</tr>
</thead>
<tbody>
<tr>
<td>09/07/10</td>
<td>v.1.0</td>
<td>First Draft</td>
<td>IPI</td>
</tr>
<tr>
<td>09/13/10</td>
<td>v.1.1</td>
<td>Modifications + corrections</td>
<td>IPI</td>
</tr>
<tr>
<td>09/20/10</td>
<td>v.1.2</td>
<td>Modifications</td>
<td>IPI</td>
</tr>
</tbody>
</table>
2. Table of Contents

<table>
<thead>
<tr>
<th>Section #</th>
<th>Section</th>
<th>Page #</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Revision Page</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>Table of Contents</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>Test Plan Identifier</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>Introduction</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>Test Items</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>Features and Functions to Test</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>Approach/Strategy</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>Actions</td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>Item Pass/Fail/Neutral Criteria</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>Suspension Criteria and Resumption Requirements</td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>Test Deliverables</td>
<td></td>
</tr>
<tr>
<td>12</td>
<td>Test Environments</td>
<td></td>
</tr>
<tr>
<td>13</td>
<td>Schedule</td>
<td></td>
</tr>
<tr>
<td>14</td>
<td>Risks and Contingencies</td>
<td></td>
</tr>
</tbody>
</table>

\(^{10}\) The page numbers of the original Test document have been removed to avoid confusion and conflict with the page numbering of this dissertation.
3. Test Plan Identifier

REGEX Repository Test Plan.

4. Introduction

Most forensic analysis tools allow the specification of search strings as regular expressions. Regular expressions are a formal syntactic language used in computer science and mathematics to specify classes of strings. They employ meta-characters to specify sets of actual characters. Many law enforcement agents find forming these regular expressions difficult—particularly in forming effective strings that do not cause an overwhelming number of false positives or do not miss evidence. Since a run of an analysis tool can take hours (even days), having the search strings be correct and effective is crucial; re-running an analysis due to ineffective search strings can be a substantial waste of time.

The above-mentioned forensics analysis tools also provide the functionality to search for specific keyword lists. They will search a device and return any results were any of the specified keywords appear.

Our repository will provide an environment where law enforcement agents can share keywords, keyword lists and regex expressions with other investigators. They will also be able to comment on the usefulness of others’ contributions and to vote on how useful the different keywords are.

The purpose of this document is to provide a clear structure and guidelines that will help us test the site and make sure that everything works correctly before the site
becomes “live.” The site will be tested as many times as necessary in order to make sure everything works as expected.

5. Test Items

The scope of this Testing activity will include:

- REGEX Repository v.1 website
- Windows and Macintosh based client platforms

The scope of this Testing activity will not include:

- the Regex generator itself.

6. Features and Functions to Test

The philosophy of the testing is risk-based testing, i.e. each test case will be prioritized as High, Medium, or Low priority and then scheduled accordingly (Highest first). The following Features and Functions are classified as high priority, medium priority or low priority. In the following section of the document, once an item has been listed and explained, we indicate its importance writing (high), (medium) or (low) at the end of the paragraph. Testing will include the following aspects of the REGEX REPOSITORY website:

6.1 Accessibility

6.1.1 Is the site friendly to color blind viewers? (low)
6.1.2 Does the text resize appropriately, without disrupting the layout? (high)
6.1.3 If images are turned off do alternative texts appear? (medium)
6.2 Coding standards

6.2.1 The site has been designed using standard PHP, SQL and HTML language. The code must be secure, easy to read and understand and appropriately commented. (high)

6.3 Compatibility

6.3.1 Web browsers should accurately render the webpage when viewed with different screen resolutions. (high)

6.3.2 When the browser is resized the website resizes accordingly. (high)

6.3.3 The content of the webpage is clearly readable when printed. (medium)

6.4 Content

6.4.1 Content should meet the following requirements:

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Follows the general layout of the site</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>The visitor’s location is always apparent</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>The page correspond to its title</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>The URL correspond to its title</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>No spelling mistakes</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>It has the Universal Header</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>It has the Universal Footer</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>It displays the ECTCoE logo</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Links on the navigation menu appear always in</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
the same order | |
| There are no broken links | |
| Links take the visitor to the expected location | |
| Page displays extended characters appropriately | |

**Functionality**

6.5.1 index.php

-- There is no special functionality to be tested on the index.php page.

6.5.2 search_do.php

-- It will search if the term to be searched appears as a keyword in any of the existing keywordlist. PASS / FAIL

-- It will search if the term to be searched appears as part of the title of any keyword list. PASS / FAIL

-- If the search returns any positive results, it will create a link to the appropriate keyword list(s). Such links work appropriately. PASS /FAIL

6.5.3 browse.php

-- It will display the name of the super-categories in a font larger than the font used for the categories and will list all the categories in an unordered list below the appropriate supercategory. PASS / FAIL

-- The categories are displayed in alphabetical order (except the miscellaneous category, that always comes last). PASS / FAIL

---

All items in this section are considered high priority.
-- The categories link to all of the keyword lists stored under it (will take
visitors to the category_show.php page and pass the categoryID using the
GET method - will show on the browser navigation bar). PASS / FAIL

6.5.4 category_show.php

-- It will display the name of all of the keyword lists stored under it in
alphabetical order. If no keyword list has been added to such category, visitors
will be informed. PASS / FAIL

6.5.5 keywordList_show.php

-- This page will display first the name of the keyword list and then all the
keywords in the keyword list. Next to each term, it will display the number of
positive and negative votes that a term has received. Users will be able to vote
using the thumbs-up / thumbs-down icons. PASS / FAIL
-- Once a visitor votes on one term, the vote count will be refreshed and grayed
out (consequently a user will not be able to vote again on that term unless he
refreshes the page). PASS / FAIL
-- The Check All button will select all of the terms in the keyword list. The
Uncheck All button will unselect all terms. Once a button is pressed, nothing
will happen if immediately following the visitor clicks again on the same
button. PASS / FAIL
-- Each term can be selected / unselected individually by clicking on the box
next to it. PASS / FAIL
-- The Export to Text File button will export the selected terms to a CSV file. The name of the file will be the same as the keyword list name (spaces replaced by underscores). If no term is selected, the site will export an empty file. PASS / FAIL

-- The Add Terms button will bring visitors to the keyword_form.php page (this page will display the name of the existing keyword list). PASS / FAIL

-- The Add Comment button will bring visitors to the comment_form.php page (this page will display the name of the existing keyword list). PASS / FAIL

6.5.6 keyword_form.php

-- If a visitor tries to visit this page directly she will receive the following error message: Missing data: keywordListID or keywordListDescription not found. PASS / FAIL

-- List of keywords will be read as separated terms if they are separated by commas, tabs, or new line carriage (return). PASS / FAIL

-- Double commas (,,) will be ignored. PASS / FAIL

-- A contributor won't be able to submit the form without choosing a username. If the username selected by the contributor does not exist in the database, it will be directly added. PASS / FAIL

6.5.7 keywordList_form.php
-- If a main category is not selected, the Keyword List will be added to the miscellaneous category. PASS / FAIL

-- A contributor will not be able to submit the form without choosing a username. If the username selected by the contributor does not exist in the database, it will be directly added. PASS / FAIL

-- Once the main category is selected, a pull down menu will show up with the appropriate subcategories. PASS / FAIL

-- If a subcategory is not chosen, the term keyword List will be added to the miscellaneous subcategory of the appropriate category. PASS / FAIL

-- If the category is changed, the menus will refresh accordingly. PASS / FAIL

-- List of keywords will be read as separated terms if they are separated by commas, tabs, or new line carriage (return). PASS / FAIL

-- Double commas (,,) will be ignored. PASS / FAIL

6.5.8 comment_form.php

-- If a visitor tries to visit this page directly she will receive the following error message: Missing data: keywordListID or keywordListDescription not found. PASS / FAIL

-- A contributor will not be able to submit the form without choosing a username. If the username selected by the contributor does not exist in the database, it will be directly added. PASS / FAIL

6.5.9 comment_show.php
-- If a visitor tries to visit this page directly, she will receive an error message stating that a keywordlist ID has not been passed. PASS / FAIL
-- A user will receive an informational message, if there are no comments available for a keyword list. PASS / FAIL
-- Comments will be displayed in chronological order (most recent first).
PASS / FAIL
-- Comments will be displayed in a table with the following columns: Author / Comment / Date. PASS / FAIL

6.5.10 regex_add.PHP
-- If a visitor tries to visit this page directly, she will receive an error message stating that data are missing and will link to the main page. PASS / FAIL
-- If a visitor tries to submit an expression without all of the required data, she will receive a message indicating what data are missing. PASS / FAIL
-- A contributor will not be able to submit the form without choosing a username. If the username selected by the contributor does not exist in the database, it will be directly added. PASS / FAIL

6.5.11 regex_show.PHP
-- It will display the name of all of the regex expressions stored under it in alphabetical order. If no regex expressions are available, visitors will be informed. PASS / FAIL
6.5.12 regex_show2.php

-- If a visitor tries to visit this page directly he will receive the following error message: ERROR: No ExpID passed. PASS / FAIL

-- The expression will be listed in a table including the following fields: Author / Title / Expression / Description / Votes.

-- Once a visitor votes on one term, the vote count will be refreshed and grayed out (consequently a user will not be able to vote again on that term unless he refreshes the page). PASS / FAIL

6.6 Navigation

6.6.1 All the links of the website will be checked to ensure that they meet the following specifications:

6.6.1.1 The link should not be broken (unless the target has not yet been developed) and goes to the most appropriate location – see file Navigation.xls. *(high)*

6.6.1.2 If the link points to a directory (instead of a specific webpage), the link ends with a slash. *(low)*

6.6.2 Each page must have a meaningful page name.

6.6.2.1 The index page must be bookmarkable. *(medium)*

6.7 Usability
6.7.1 The web page controls, behavior and even aesthetics remain consistent across the entire web site. (high)

6.8 **PHP Shared Functions**

6.8.1 A library of shared PHP functions has been developed to serve as a framework for developing the web site. These functions do not create a complete web page, so they cannot be tested using the page-oriented process described elsewhere in this document. Other pages will fail to work correctly if these functions do not work correctly.

6.8.2 To test the shared functions, simply use the site as a standard user. If any of the functions fail, you will receive an error message. Copy and paste that information into the testing report, indicating what steps you follow before getting that error.

7. **Approach/Strategy**

The testing will use a combination of manual and automated testing. Due to the short period of time allotted for test execution, units (pages) will be tested in order of their creation and the webpage’s source code for the page under test will be frozen while being tested. Except for critical fixes that are blocking the test efforts, changes will not be scheduled while a unit of code is being tested.
Once a page is tested the tester will fill out a report using the Test Form. Then, the generated report will be sent to ipi@uri.edu. When creating the report, testers will include a screenshot (if possible) for clarification when they consider it necessary.

7.1 All multi-page transactions, which require the browser to maintain some sort of session with the website, will be tested to ensure that the session is maintained under the following conditions:

7.1.1 Tester will clear the disk and/or memory cache mid-way through a series of Web pages.
7.1.2 Tester will use the browser’s Back (and subsequently Forward) buttons mid-way through the series of pages.
7.1.3 Tester will use the Go and/or History buttons to revisit previous pages out of synch.
7.1.4 Tester will use the browser’s Reload button mid-way through a series of pages.
7.1.5 Tester will resize the browser window mid-way through a series of pages.
7.1.6 Tester will abort mid-way though the series of pages.

8. Actions

8.1 Check if alternative texts appear instead of images when images in browser are turned off (6.1.3):

*Action 8.1.1*  
IE: Tools → Internet Options → Advanced → Show pictures (turn off), Safari: Safari → Preferences → Appearance → Display images when the page opens (turn off),
Firefox: Tools → Options → Content → Load Pictures
Automatically (turn off).

8.2 Check does new bookmark appear on bookmark list and has it corresponding name and correct link (6.6.2.1):

*Action 8.2.1* (for creating a bookmark for current page)

IE: Favorites → Add to Favorites.. → Add,
Safari: Bookmarks → Add Bookmark.. → Add,
Firefox: Bookmarks → Bookmark This Page.. → Ok.

*Action 8.2.2* (for checking link of new bookmark)

IE: Favorites → click on created bookmark,
Safari: Bookmarks → Bookmarks Bar → click on created bookmark,
Firefox: Bookmarks → click on created bookmark.

8.3 Check if the text and images resize properly and do not overlap or come out of given tables (6.3.2):

*Action 8.3.1*

IE: View → Text Size → Click on larger, largest, smaller, smallest,
Safari: View → Click on “Make Text Bigger” and “Make Text Smaller,”
Firefox: View → Text Size → Click on Increase and Decrease.
9. Item Pass/Fail/Neutral Criteria

Tester will retain the decision as to whether the total and/or criticality of any or all detected incidents/defects warrant the delay (or reworking) of the REGEX REPOSiTORY website.

10. Suspension Criteria and Resumption Requirements

In general, testing will only stop if the site under test becomes unavailable. If testing is suspended due to the website becoming unavailable, testing will be resumed once access to the website is reestablished.

11. Test Deliverables

The following documents will be generated as a result of these testing activities:

- Test Plan (this document)
- Test reports for each tested page

12. Test Environments

Test team will use a set of desktop and laptop machines, running on different platforms (Windows & Mac)

Testers will use the following Windows based browsers:

- MS IE 8
- Mozilla Firefox
- Chrome

Testers will use the following Mac based browser:
• Safari
• Mozilla Firefox
• Chrome

When possible, Browser settings (cache size, # of connections, font selection etc.) will be left unchanged (i.e., the installation defaults will be used for all testing).

13. Schedule

Test execution for each available page is expected to start immediately after the test plan has been approved and the page is available.

Producing the Test Report is expected to be completed within 3 business days of completing the test execution.

<table>
<thead>
<tr>
<th>Date</th>
<th>Event</th>
</tr>
</thead>
<tbody>
<tr>
<td>09/07/10</td>
<td>v.1.0</td>
</tr>
<tr>
<td>09/13/10</td>
<td>v.1.1</td>
</tr>
<tr>
<td>09/20/10</td>
<td>v.1.1</td>
</tr>
</tbody>
</table>

14. Risk and Contingencies

The following seeks to identify some of the more likely project risks and propose possible contingencies:

• Website becomes unavailable – Testing will be delayed until this situation is rectified.

• A large number of defects/incidents makes it functionally impossible to run all of the test cases – As many test cases as possible will be executed, the testers in conjunction with Project Managers will ultimately make the
decision as to whether the number of defects/incidents warrants delaying the implementation of the production version.

- Not enough time to complete all test cases. Some test cases will be skipped, starting with the lowest priority.
8. BIBLIOGRAPHY


