

IMPROVING SECURITY THROUGH SEARCH ENGINE PRESENCE AUDITS

BY

BRIAN CAVANAUGH

A Thesis Submitted to the School of Graduate Studies
in Partial Fulfillment of the Requirement for the Degree of
Master of Science

Southern Connecticut State University

New Haven, Connecticut

August 2009

IMPROVING SECURITY THROUGH SEARCH ENGINE PRESENCE AUDITS

BY

BRIAN CAVANAUGH

This thesis was prepared under the direction of the candidate's thesis advisor, Dr. Lisa Lancor, Department of Computer Science, and it has been approved by the members of the candidate's thesis committee. It was submitted to the School of Graduate Studies and was accepted in partial fulfillment of the requirements for the degree of Master of Science.

Lisa Lancor, Ph.D.
Thesis Advisor

Hrvoje Podnar, Ph.D.
Second Reader

Winnie Yu, M.S.
Department Chairperson

Sandra C. Holley, Ph.D.
Dean, School of Graduate Studies

Date

ABSTRACT

Author: Brian Cavanaugh
Title: IMPROVING SECURITY THROUGH SEARCH ENGINE PRESENCE AUDITS
Thesis Advisor: Dr. Lisa Lancor
Institution: Southern Connecticut State University
Year: 2009

Search engines employ automated software agents called spiders, crawlers, or robots (known as bots). Search agents traverse Internet sites looking for semantic information embedded within publicly visible web pages. If any files that are not intended for public viewing are erroneously placed in a public directory on a web server, they can be indexed by a search engine. Once indexed, this information can be located by a search engine query, resulting in an information leak.

The use of a search engine for untoward purposes, particularly to compromise or exploit a computer system or network, is a relatively new phenomena. This thesis presents a software tool to help mitigate the risk of an information leak by querying the search engine in the same way that an attacker might. The querying of the search engine has been automated in order that it be done quickly, effectively and periodically.

ACKNOWLEDGMENTS

I wish to express my gratitude to everyone who has contributed to making the completion of this thesis possible. Special thanks to Dr. Lisa Lancor the thesis advisor and Dr. Hrvoje Podnar, the second reader.

TABLE OF CONTENTS

CHAPTER 1: INTRODUCTION.....	1
1.1. <i>Background</i>	1
1.2. <i>Literature Review</i>	4
CHAPTER 2: SEPA SOFTWARE TOOL.....	7
2.1. <i>About</i>	7
2.2. <i>Use Cases</i>	9
2.3. <i>Formulating and Storing Queries</i>	10
2.4. <i>Target Manager</i>	13
2.5. <i>Running a Job</i>	18
2.5.1. <i>Graphical interface</i>	18
2.5.2. <i>Console interface</i>	19
2.6. <i>Results Viewer</i>	20
CHAPTER 3: DESIGN OF SEPA SOFTWARE TOOL.....	26
3.1. <i>High Level Architecture Design</i>	26
3.2. <i>Programming Language and Dependencies</i>	27
3.2.1. <i>Programming language</i>	27
3.2.2. <i>Dependencies and requirements</i>	28
3.3. <i>Component Design</i>	29

3.3.1. <i>Interfaces</i>	29
3.3.2. <i>Query database</i>	29
3.3.3. <i>Target configuration file</i>	30
3.3.4. <i>Query engine</i>	30
3.3.5. <i>Reporting engine</i>	31
3.3.6. <i>Results repository</i>	32
3.3.7. <i>Application log file</i>	33
CHAPTER 4: SEPA SOFTWARE IMPLEMENTATION	35
4.1. <i>Graphical Interface</i>	36
4.1.1. <i>Tabbed frame</i>	36
4.1.2. <i>About tab</i>	37
4.1.3. <i>Query DB tab</i>	37
4.1.4. <i>Target configuration</i>	37
4.1.5. <i>Results tab</i>	38
4.2. <i>Console Interface</i>	40
4.2.1. <i>Menu system</i>	41
4.3. <i>Engine Code</i>	41
4.3.1. <i>TargetManager class</i>	41
4.3.2. <i>SEPAQueryBuilder and SEPAResultsProcessor classes</i>	42
4.3.3. <i>Query submission</i>	44
4.3.4. <i>Result processing</i>	45
4.3.5. <i>Engine events</i>	47

4.3.6. <i>Query database</i>	47
CHAPTER 5: EXPIREMENT IMPLEMENATION.....	49
5.1. <i>Experiment Implementation</i>	49
CHAPTER 6: RESULTS.....	53
6.1. <i>API Bug</i>	53
6.2. <i>Manual vs Automated Query Results</i>	55
6.3. <i>Findings</i>	58
6.3.1. <i>Backup files</i>	59
6.3.2. <i>Configuration management</i>	60
6.3.3. <i>Devices</i>	61
6.3.4. <i>Error messages</i>	61
6.3.5. <i>Log files and reports</i>	62
6.3.6. <i>Log in portals</i>	63
6.3.7. <i>Privacy related</i>	64
6.3.8. <i>Technology profile</i>	66
6.4. <i>Software Usability</i>	67
CHAPTER 7: CONCLUSION AND FUTURE WORK.....	68
7.1. <i>Conclusion</i>	68
7.2. <i>Future Work</i>	71
APPENDICES.....	73
Appendix A: <i>Converting the Goolag Database to a SEPA QueryDB</i>	73
Appendix B: <i>Detailed Manual vs Automated Query Result Counts</i>	75

REFERENCES..... 84

LIST OF TABLES

Table 1. <i>Search Engine Questions Translated to Specific Queries</i>	11
Table 2. <i>SEPA Java Library Dependencies</i>	28
Table 3. <i>General Questions for a Systems Administrator</i>	51
Table 4. <i>Categories and Query Counts</i>	52
Table 5. <i>Summary of Result Counts by Category</i>	56
Table 6. <i>Summary of Query Ratings by Category</i>	59
Table 7. <i>Search Queries and Result Counts</i>	75

LIST OF FIGURES

<i>Figure 1.</i> SEPA graphical user interface.....	8
<i>Figure 2.</i> System administrator use case diagram.....	9
<i>Figure 3.</i> Maintaining the query database.....	12
<i>Figure 4.</i> Graphical interface for target configuration.....	16
<i>Figure 5.</i> Possible Yahoo! query parameters.....	17
<i>Figure 6.</i> Running and monitoring a query job.....	19
<i>Figure 7.</i> Console interface used to launch a query job.....	20
<i>Figure 8.</i> Hierarchal results display.....	22
<i>Figure 9.</i> Viewing query job parameters.....	24
<i>Figure 10.</i> Launching a result URL with the default browser.....	25
<i>Figure 11.</i> SEPA software architecture diagram.....	27
<i>Figure 12.</i> Query database XML schema.....	30
<i>Figure 13.</i> Partial screen shot of the eclipse IDE.....	35
<i>Figure 14.</i> Software factory pattern making run time decisions.....	44
<i>Figure 15.</i> Code for an HTTP request.....	45
<i>Figure 16.</i> Graphical representation of manual result counts vs automated result counts.....	57
<i>Figure 17.</i> Data dump found on a target site with user names and passwords.....	60
<i>Figure 18.</i> Interesting result URLs from a query looking for WS_FTP.LOG files.....	63

Figure 19. A directory listing with credit card validation source code.....65

Figure 20. A confidential letter giving access to a security report..... 65

Figure 21. A spreadsheet with a credit card image embedded in it..... 66

Figure 22. Google removal request tool..... 71

CHAPTER 1: INTRODUCTION

1.1. Background

Recent findings from the annual computer crime survey conducted by the Computer Security Institute show that the average annual loss for participating companies has significantly grown from \$168,000 in 2006 to \$350,000 in 2007. Cumulative total losses for all the companies in 2007 are estimated at \$67 million (Richardson, 2007). Every day networks and individual computers are compromised and exploited by unauthorized individuals. The intrusions happen for a variety of reasons including information intelligence, thrill seeking, or unauthorized service exploitation.

The initial phase of identifying and assessing a target computer is often called the *reconnaissance phase*. In this phase, attackers try to find as much information as possible about the network and systems before compromise. The type of information sought during this phase includes: knowledge of the hardware, software policies, and architecture of the target network. According to Cole (2002), if an attacker performs this information-gathering stage correctly and in enough detail, access is almost guaranteed. In this phase, intruders use Internet search engines to perform reconnaissance and to search for potential system vulnerabilities.

Search engines employ automated software agents called spiders, crawlers, or robots (known as bots). Search agents traverse Internet sites looking for semantic information embedded within publicly visible web pages. If any files that are not intended for public viewing are erroneously placed in a public directory on a web server, they can be indexed by a search engine. Once indexed, this information can then be located by a search engine query, resulting in an information leak. Possibly the most powerful and popular search engine today is the Google search engine closely followed by the Yahoo! search engine.

Google hacking is the term associated with the crafting of advanced queries intended for search engines with the goal of discovering sensitive information. The use of advanced search operators allows for refined filtering which is necessary to hone in on the result set that is being sought. A number of publicly available Google hacking example repositories can be found on the Internet with queries included within the repository that are integrated with the Google search engine (Long, 2007b).

Google and other search engines such as Yahoo!, have an application program interface (API) that allows for the development of software applications that are able to retrieve information from the search engine. In order to lessen potential performance degradation, some APIs have certain limitations. For example, Yahoo's API limits automated search requests to 5,000 in a 24-hour period; and for ambiguous reasons, Google no longer allows automated queries sent to its search engine. As of December 2006, Google does not welcome and has limited automated searches against its API.

To maintain a secure site, it is critical to know not only what site information is available to a search engine, but also what information related to the web site is already contained in the search engine's database. By querying a search engine in the same way that an attacker would in the reconnaissance phase of an attack, a system administrator can quickly detect and correct such findings. Unfortunately, manually querying a search engine's database for vulnerabilities is nearly impossible given the potentially large number of queries that must be executed. Thus, automating the querying of the search engine so that it can be done quickly, easily and periodically is critical.

The implications of having public repositories accessible and indexable by search engines are readily apparent. The proposed solution to mitigate the risk of having an information leak is to automate the submission of advanced search queries to the Yahoo! Search service. The question to be answered is if the results from the submission of automated search queries are of comparable quality and effect as those submitted manually.

This thesis presents the software application SEPA (Search Engine Presence Audit), to aid in the submission, retrieval and analysis of advanced search queries to find potential vulnerabilities of a target web site located in a search engine database. The software application communicates with a search engine, manages a list of queries and presents results to an end user in an easy-to-analyze manner.

1.2. Literature Review

The depth of the discussion concerning the reconnaissance phase of an attack can vary largely (Cole, 2002; Long, 2005; Long et al., 2006; Scambray & McClure & Kurtz, 2001). Every treatise surveyed on the subject matter indicates that attackers want to know as much about the target as possible and will sometimes invest substantial time gathering this information. Although methodologies vary, attackers always try to maintain a large degree of separation between themselves and the target. Hence, search engine reconnaissance is appealing because an attacker can often profile a site without ever visiting it. Penetration testers, who are hired to evaluate the security of a system by simulating a hacker's actions, almost always use search engine reconnaissance (Long, 2005; Long et al.; Mowse, 2003; Peake, 2003).

The first book on search engine reconnaissance, entitled *Google Hacking for Penetration Testers* (Long, 2005), was written by a veteran penetration tester. In fact, the author coined the phrase "Google hacking" and brought the power of search engines in the context of security to the public eye. Advanced search techniques presented in the book not only locate exploits, login portals, network hardware and sensitive data, but also preserve anonymity by use of cached pages and translation service. The book illustrates the problem and also details the knowledge and techniques to counteract it.

Numerous articles have been published related to Google hacking. Most of the literature either describes what Google hacking is and how it is used (Granneman, 2004; Lancor & Workman, 2007; Mowse, 2003) or reports on vulnerabilities that were exploited using Google

hacking techniques (Danhieux, 2004; Kotadia, 2005). Another reference reports on companies and organizations who declare a “call to arms” to defend against Google hacking (Verton 2002).

Requisite skills for effective search engine reconnaissance and protection from Google hacking are being taught in classrooms. A limited number of resources related to search engine reconnaissance correspond to educational references. The SANS (SysAdmin, Audit, Network, Security) Institute offers a widely available course entitled “Power Search with Google” (formerly called “Google Hacking & Defense”). Lancor & Workman (2007) describe Google hacking as a critical component in a graduate course on web security.

Overall, the literature emphasizes that advanced search queries that a wily hacker would ask a search engine must be similarly asked by an administrator. This is done to ensure that administrators are aware of the information that is available to the public. As previously discussed, queries should be automated to streamline the discovery process. Currently, there are a few tools that do this: Site Digger, Gooscan and Wikto (Foundstone, Inc., 2007; Long, 2007a; Sensepost, Inc., 2007). They all automate queries against Google. Site Digger requires an authentication key that is no longer obtainable; while others work outside of Google's terms of service.

The objective of the thesis is to mitigate the risk of an information leak by querying the Yahoo! search engine in the same way that an attacker might. As stated previously, the querying of the search engine manually in a reasonable time frame is not feasible so the

submission of the advanced search queries has been automated in order that it be done quickly, effectively and periodically. Automated result counts are compared to manual result counts to measure any difference in the manual and automated query submission methods.

The SEPA software application developed for this thesis uses Yahoo's search service but can be easily made to work with other search engines as well. Another distinction of SEPA is the way in which the results are displayed. Result URLs which were not found in a previous job are displayed in red. This denotes the delta or changes in the result set since the last time a query job was run. This results display methodology was not found in any other tool surveyed. The implementation of the tool, interfaces, and results will be discussed in depth in the following chapters.

CHAPTER 2: SEPA SOFTWARE TOOL

2.1. About

A search engine presence audit is the act of querying the search engine to discover what information it contains about your site. The purpose of the software is to automate the search engine presence audit (SEPA). Figure 1 shows the graphical interface provided by the SEPA software tool. The about tab simply shows who designed and implemented the tool. The other three tabs offer the core functionality to maintain a list of advanced search queries, run query jobs for target sites and view the results. Each of these tabs will be discussed in depth.

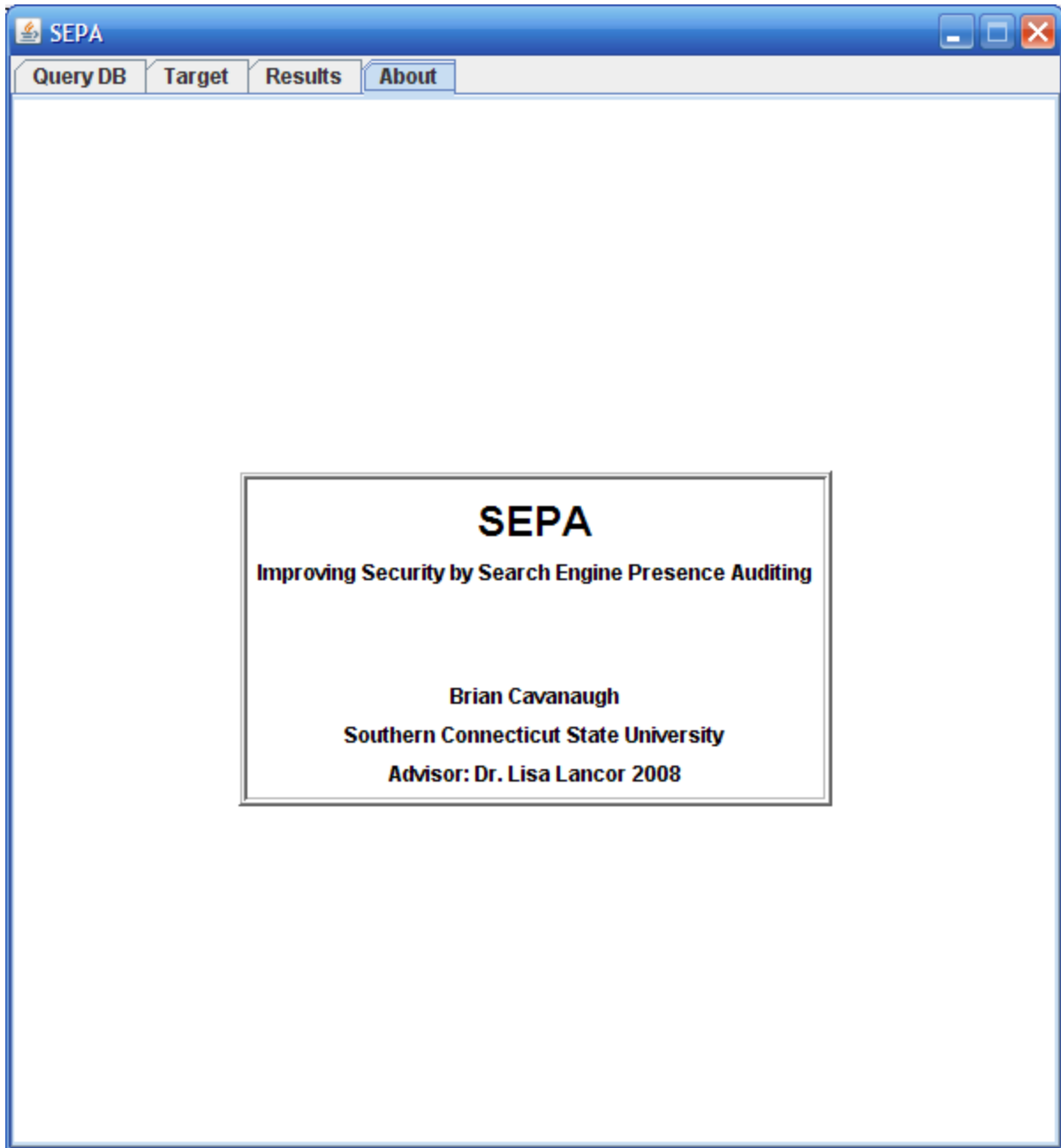


Figure 1. SEPA graphical user interface.

2.2. Use Cases

Administrators can formulate the same kinds of questions about their own network that an attacker would. Using advanced search techniques these questions can be translated to powerful and refined queries to be submitted to the search engine. Figure 2 shows a use case diagram of the SEPA software application. The SEPA software tool communicates with a search engine, manages a database of queries, runs a list of queries against a target site and presents results in an easy-to-analyze manner. Each one of these tasks maps directly to the tabs that are shown in the graphical interface displayed in Figure 1. Multiple target sites can be specified together with appropriate site queries. In addition to the graphical user interface, the software can be accessed via a console or command line prompt. The command line prompt supports a limited set of functionality and is provided for the purpose of scheduling autonomous runs only.

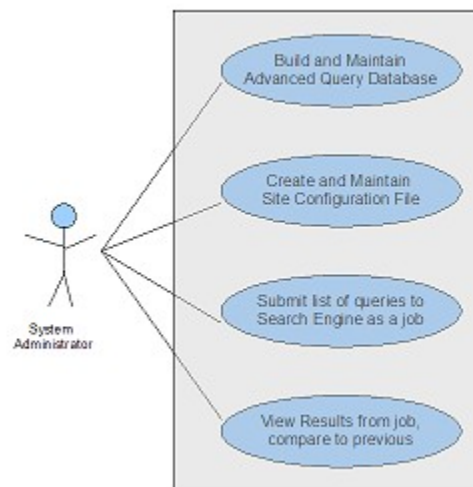


Figure 2. System administrator use case diagram.

2.3. Formulating and Storing Queries

The SEPA software tool provides a graphical interface for managing the queries provided by the system administrator. A list of questions an administrator might want to ask a search engine are:

- Are there any log files or network scan results publicly available?
- Are there any devices available on the Internet which should not be?
- Are there any error messages with too much information?
- Are there any documents with sensitive information such as passwords?
- Are there any architecture and policy details which should not be public?
- Are there any ways to easily obtain server versions?
- Are there any ways to locate login portals?
- Are there any configuration files or default files in public directories?
- Is there any source code in public directories?
- Are there any backup files in public directories?

Some example questions and how each might be translated into specific search queries are presented in Table 1.

Table 1
Search Engine Questions Translated to Specific Queries

Question and Translation to Search Query

Has the robots.txt file been indexed by a search engine?

"robots.txt" "Disallow:" filetype:txt

Has a scan report from the Nessus scanner leaked to a public directory?

intitle:"Nessus Scan Report" "This file was generated by Nessus"

Has a phpinfo file leaked to a public directory?

intitle:phpinfo()

Has a mysql dump file leaked to a public directory?

"#mysql dump" filetype:sql

Queries can easily be added, removed and updated via the graphical interface. A reference number, category title, query string, and description are stored with each entry. The queries are categorized for organizational and reporting purposes. The drop down list of categories is editable. In order to add a query to a new category, the System Administrator must clear the form and type in the category name in the drop down list widget. Figure 3 shows the user interface for managing the query database.

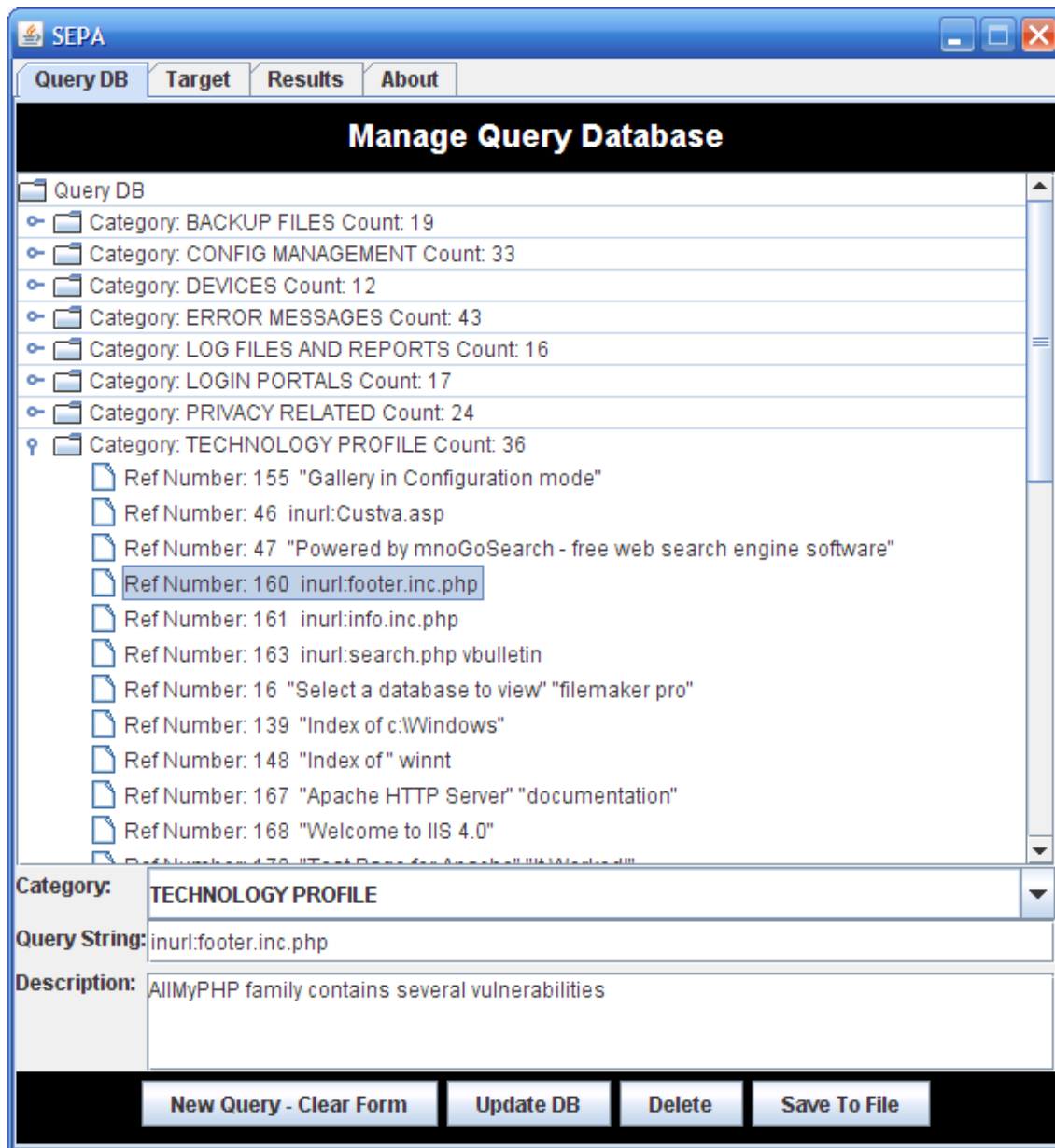


Figure 3. Maintaining the query database.

2.4. Target Manager

SEPA's graphical interface provides a way to manage search configurations for each saved target site. Target site configurations can easily be added, removed and updated. When a new target site is added, a target configuration file is automatically setup with specific target information and the defaults found in the default configuration file.

Currently, the default configuration file is set to communicate with the Yahoo! search service. The Yahoo! search service offers a variety of parameters (Yahoo! Developer, 2008b). These parameters can be changed by an administrator by simply modifying the target configuration file. No internal SEPA software code modifications are required. For instance, if the systems administrator would like to receive up to 100 results instead of the default ten, he/she only needs to modify the target configuration file available on the target configuration tab.

The target configuration file is a text file and the setup is similar to a Unix system or services configuration file with key-value pairs being separated by the first space in a line. The key or parameter is the first word on a line, and the value of the parameter is the remaining part of the line. A line starting with a # symbol is a comment line. Comments are used to convey additional information about usage and parameters to the end user.

Because SEPA is initially configured to work with the Yahoo! search engine, the administrator must obtain a Yahoo! application ID which can be freely obtained through a

registration process (Yahoo! Developer, 2008a). An example of a Yahoo! application ID is:

```
Tebo6bnV34E8ub418hBOGpooQnpv5V_rGV8xxx7W62m27TExxxfDls60t8g9uA--
```

This ID uniquely identifies a client, is required and is sent with each web request. Each client identified by this unique ID is limited to a predefined number of queries in a single 24 hour period. When this application ID is placed in the `DefaultTargetConfig.txt` file located in the `targetdata` directory underneath the SEPA root application folder, it will be included in every target configuration file created by default.

Some other query parameters are required and listed under the `queryParams` parameter in the target configuration file. The `queryParams` parameter is, by default:

```
queryParams querySite appId site results
```

It indicates what parameters will be included in each query URL built and can be seen in Figure 4. The `querySite` parameter is the base of the request URL being built. This is the location of the search service. The `appId` parameter is appended to the base URL. After the `appId` parameter the query from the query database is attached and then each parameter included in the `queryParams` parameter is subsequently appended. The `queryParams` parameter allows the parameters to be dynamically included and ordered in a query.

The `site` parameter is used to limit the queries to a particular site. To change the site we wish to limit queries to, we would modify the value of the `site` parameter located in the file or remove it entirely for no site limitation. Upon target creation the value of the `site` parameter is set dynamically to the target name being created.

The Yahoo! search service offers a number of other parameters such as region, type, and country. For the most part, they are meant to limit or restrict search results. Queries are already limited to a site and the full search language of Yahoo! Search is supported with the query parameter. The search query itself is generally used to limit the result set with the SEPA software application. If the administrator would like to include any of these parameters in a query job, he or she only has to define them and include them in the `queryParams` parameter list. Figure 4 shows all of the available query parameters.

The `category` parameter contains a list of query categories. It can be used to limit a query job to a list of queries contained in a category or set of categories. Upon target creation the current list of categories in the database is found and stored as a comma delimited list in the newly created target configuration file. By default, the `category` parameter is commented out and not used. When the `category` parameter is not used, a query job will run for every category in the database. Figure 4 shows the interface screen used to create, configure and remove target sites for the SEPA software.

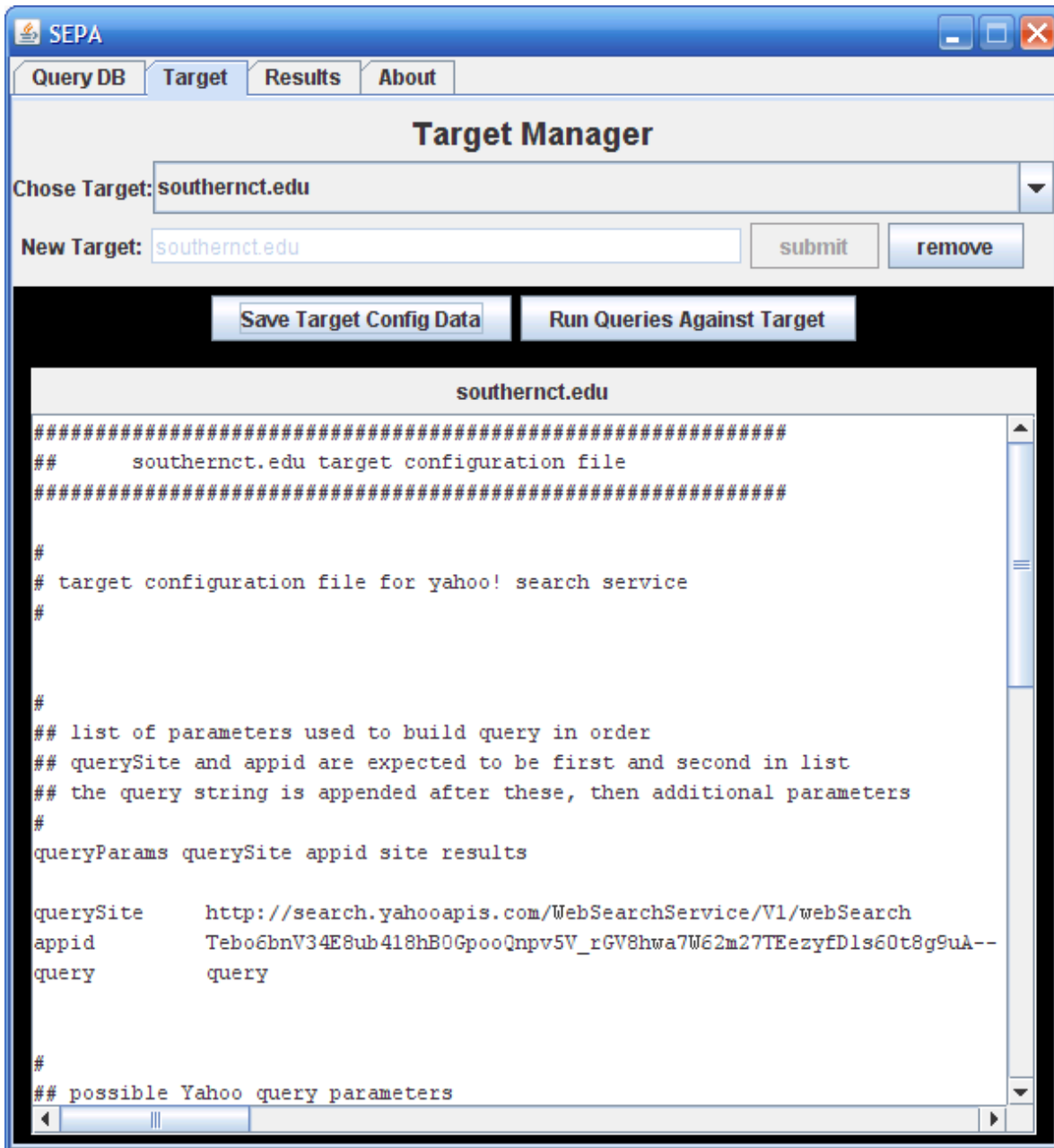


Figure 4. Graphical interface for target configuration.

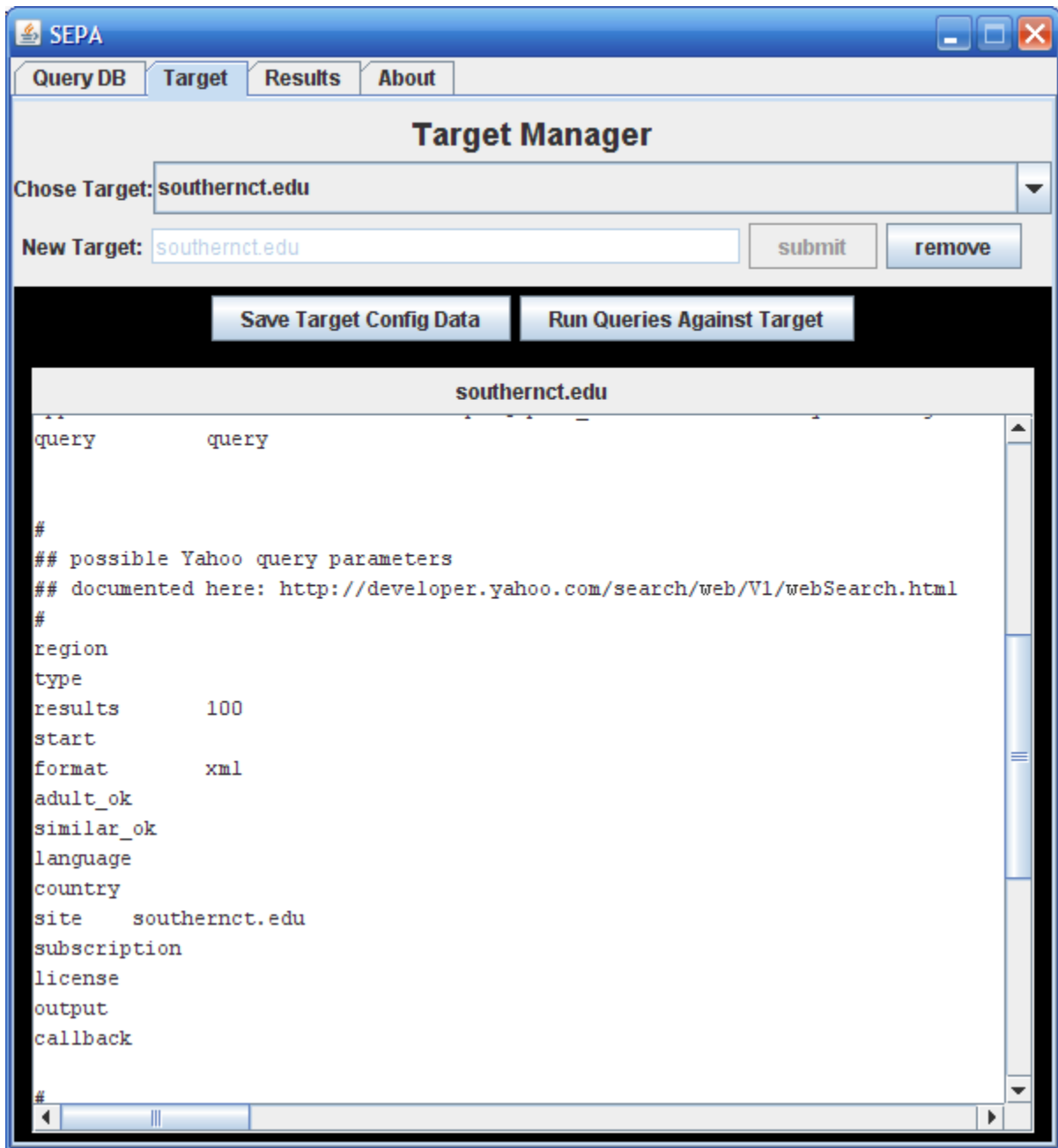


Figure 5. Possible Yahoo! query parameters.

2.5. Running a Job

2.5.1. Graphical interface. To run a list of queries against a target, the “Run Queries Against Target Button” is used. This button is found on the same screen which is used to manage target sites. The button launches a dialog which is used to start a job and monitor its progress. When the start button is pressed, a list of queries is built from the parameters in the target configuration file. Each query in the list is then submitted to the search engine asynchronously. The progress bar indicates the percentage of completed and remaining queries. Each query is shown in the dialog box after it has been submitted and results have been obtained. A query job ends with a completion message. Multiple jobs can be run at the same time and other portions of the application are available while running a job. When a job is complete the results are available for viewing. Figure 6 shows the dialog used to run a job and monitor its progress. With this particular configuration there are 1420 queries in the query job. The snapshot of the Progress Monitor dialog shows that 116 queries have been completed and that this is 8% of the total.

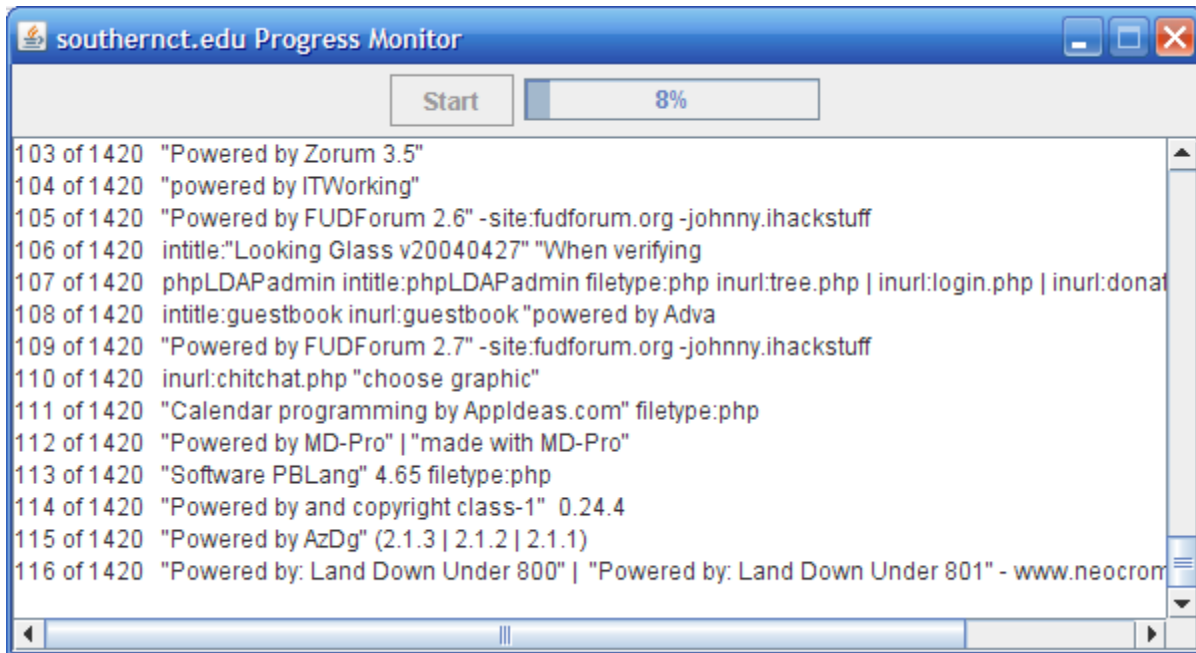


Figure 6. Running and monitoring a query job.

2.5.2. *Console interface.* A query job can also be run from a command line prompt and is provided so that a query job can be scheduled to run autonomously. In order to do this, the user needs to navigate to the SEPA application root directory and launch the application with the command:

```
java -jar sepaConsole.jar [targetname].
```

Optionally, a user can include the name of a target as a parameter. If a target is not supplied as an argument, one can be chosen from a list of target sites that are already configured. An example of the console interface is depicted in Figure 7.

```
C:\WINDOWS\system32\cmd.exe - java -jar sepaConsole.jar
09/29/2008 08:06 PM          2,131 README.txt
10/12/2008 08:43 PM          43,733 sepa.jar
10/12/2008 08:43 PM           6,222 sepaConsole.jar
10/12/2008 08:42 PM          40,622 sepaengine.jar
          4 File(s)          92,708 bytes
          8 Dir(s)  1,993,572,352 bytes free

F:\SEPA>java -jar sepaConsole.jar

-----
SEPA Tool      21.10.08 21:53:09
-----

Main Menu
-----

1. Run a Query Job
2. Exit
Enter Menu Choice: 1

-----
Run Job
-----

[2008-10-21 21:53:14,823] INFO      0[main] - org.sepa.engine.TargetManager.listT
argets(TargetManager.java:41) - Found 3 targets

-----
Target List
-----

1. testnet.net
2. southernct.edu
3. yale.edu
Chose site by number:
```

Figure 7. Console interface used to launch a query job.

The console interface is only provided to run jobs autonomously or when a graphical environment is not available. Target configuration and job results are not provided in the console interface. However, the results file is human readable and can be found in the target data directory or viewed with the results viewer in the graphical interface at another time.

2.6 Results Viewer

SEPA is designed to display results in a hierarchical fashion. Administrators can

easily navigate to the detailed results that are of interest. A top level results summary of a query job shows the number of queries executed, the start time of the job, the total number of returned results and the number of new results. This top level results summary is expandable to a category summary, which is expandable to a query summary and then to the specific query results. New results which were not in the last job are highlighted in red. This feature is strategically introduced in order to help administrators keep track of result changes since the last time the tool was utilized.

Figure 8 shows the graphical interface used to view results for a target site and a job. There are two target sites configured. Fifteen job result files were found for the `southernct.edu` site but only ten are displayed. The jobs are listed in chronological order with a notification that more results could be found in the target data directory at the end. Jobs, categories and queries which have result URLs that were not in the previous results file are highlighted in red. The number of queries, results and new results for each node are also displayed.

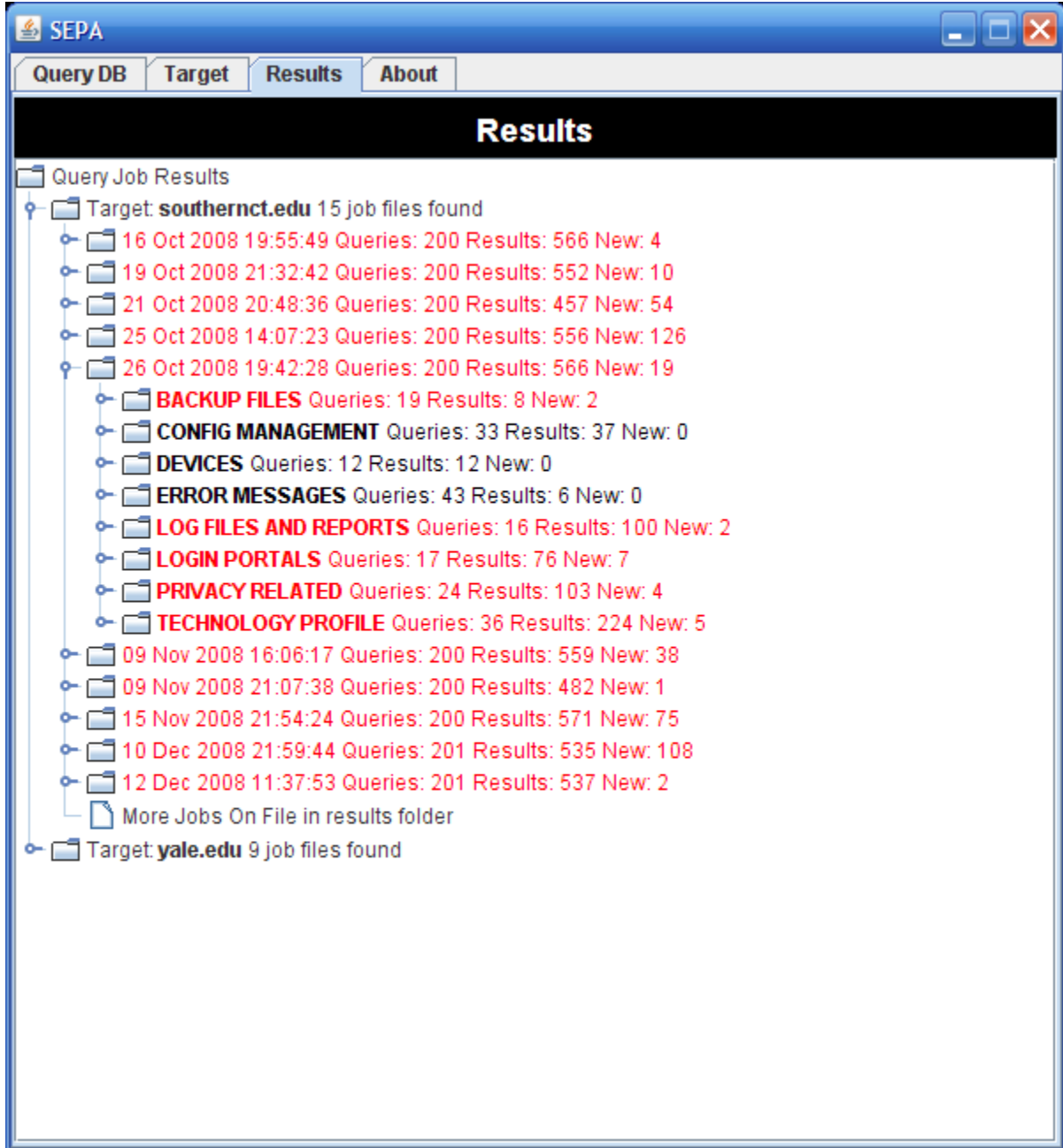


Figure 8. Hierarchal results display.

Right clicking job summary nodes, query summary nodes and result URL nodes in a results tree will reveal menu options. A job summary node offers an option to delete a job file and also display all the parameters that were used to run the selected job, (see Figure 9). A query summary node (shown underneath the job summary and category summary nodes) will reveal options to launch the query with the host's default browser using either Google, Yahoo! or to perform a *Yahoo REST Search*. REST stands for Representational State Transfer and describes the architecture of the Yahoo! Web Services. By selecting the Yahoo REST Search option, the query that was originally submitted to the Yahoo! Search API is resubmitted and the results from the service are displayed in the XML format. Right clicking a result URL from a query reveals the menu option to “Launch URL” in the host's default browser, (see Figure 10). This loads the page as it is published today by the site and not the cached version that was actually indexed by the search engine.

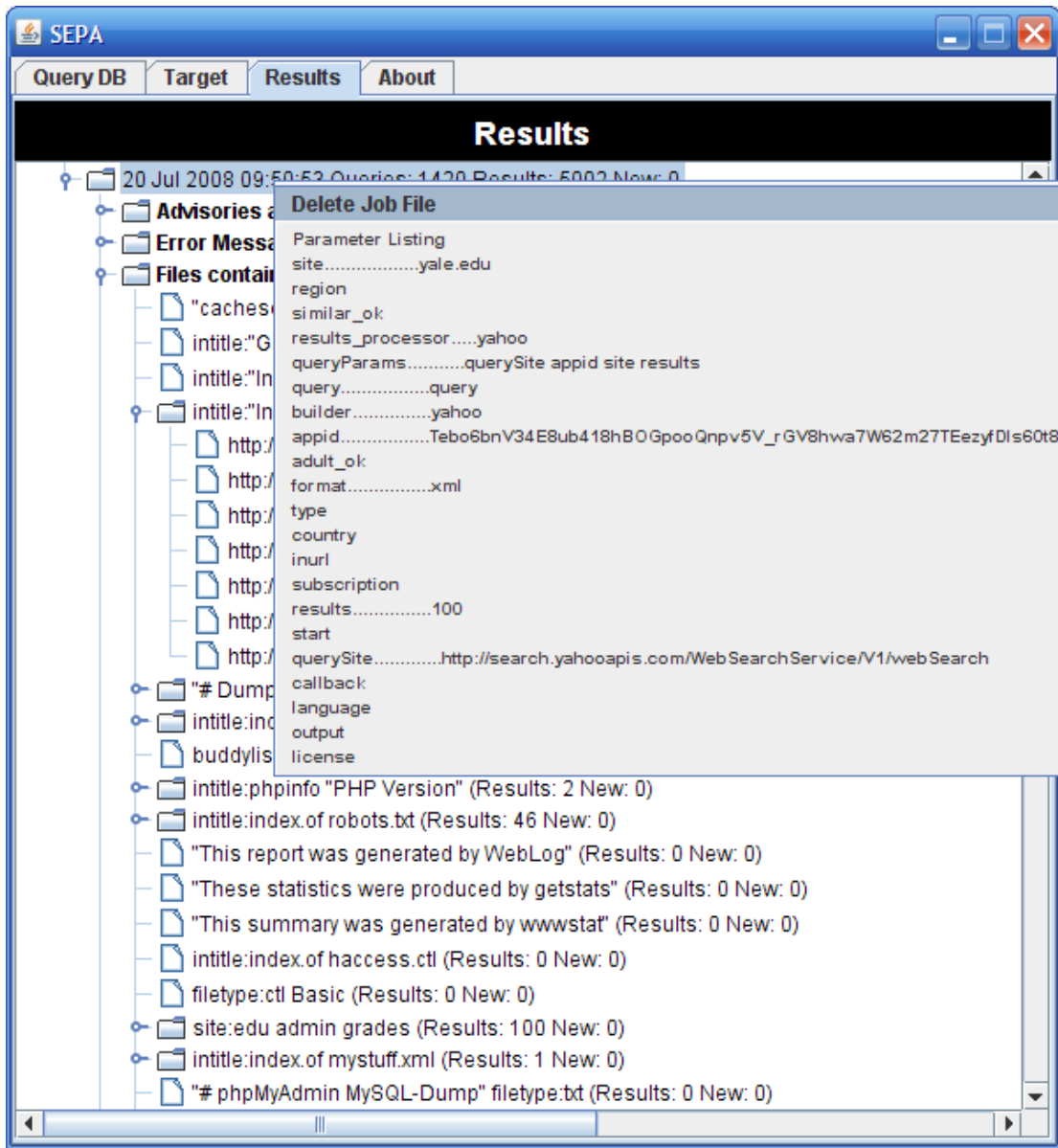


Figure 9. Viewing query job parameters.

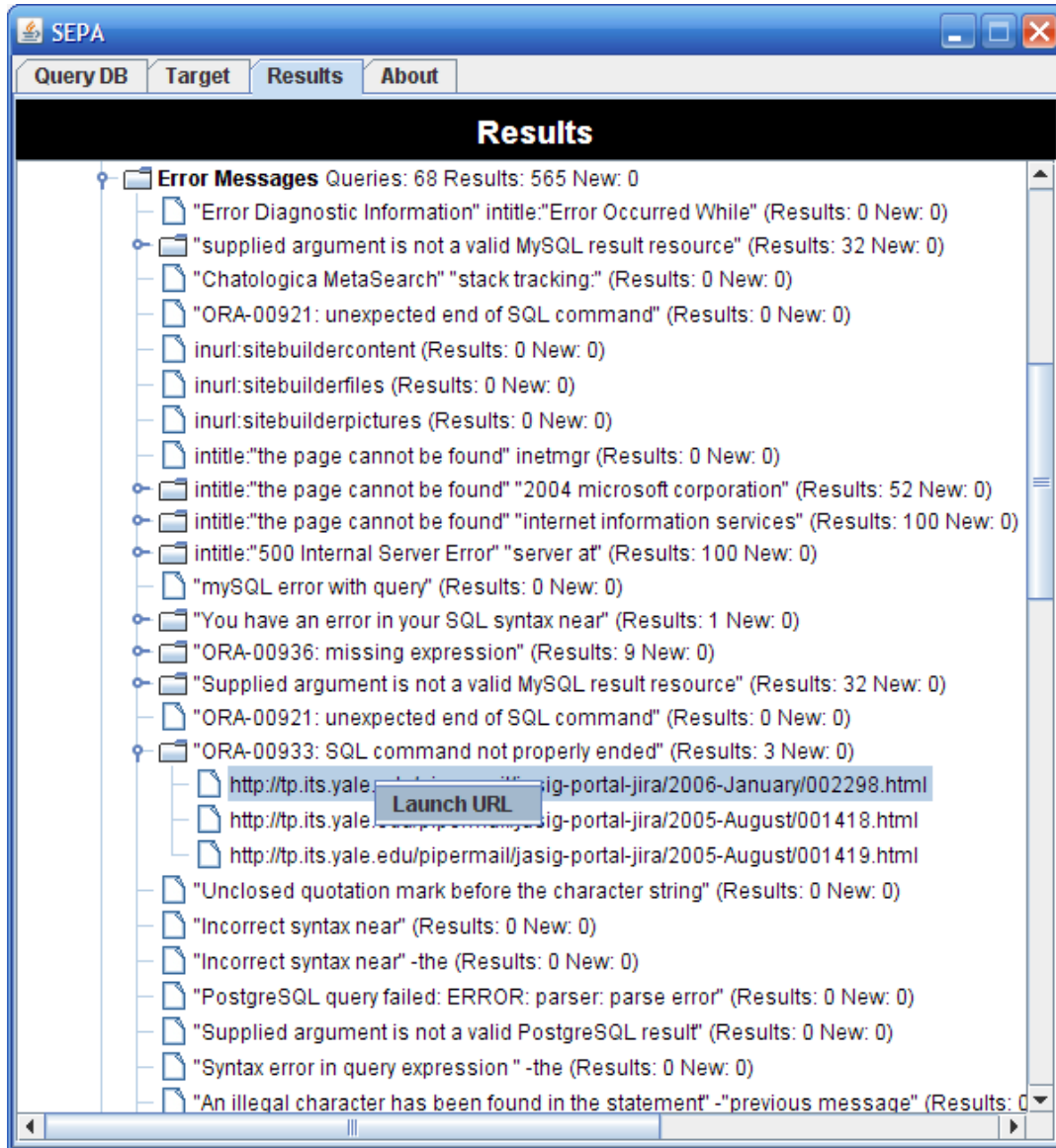


Figure 10. Launching a result URL with the default browser.

All the presented results are also saved in text files that can easily be analyzed by a human observer, or an external software program. The next chapter presents the architecture of the SEPA software and details each of its components.

CHAPTER 3: DESIGN OF SEPA SOFTWARE TOOL

This chapter presents the architecture and design of the SEPA software tool. These are details on the programming language utilized and the individual component design that makes the SEPA software easily configurable, extensible and portable.

3.1. High Level Architectural Design

SEPA is composed of seven major components (see Figure 11). The Query Engine is the crux of the system as it communicates with the Yahoo! Search service through the Internet with Yahoo's API. The Graphical User Interface facilitates the creation and maintenance of a database of advanced search queries in the Query Database as well as the maintenance of a Configuration File that specifies what query parameters need to be used and what queries to run for a site while a command line console interface allows for the autonomous running or scheduling of query jobs. The Reporting Engine stores the results in a flat database in a human-readable form. The results are most optimally viewed via the graphical interface. Every transaction is logged to the Application Log File with a logging framework that can be configured to log to multiple interfaces at different levels and in different formats. The software is configurable, extensible and portable due to its design and implementation.

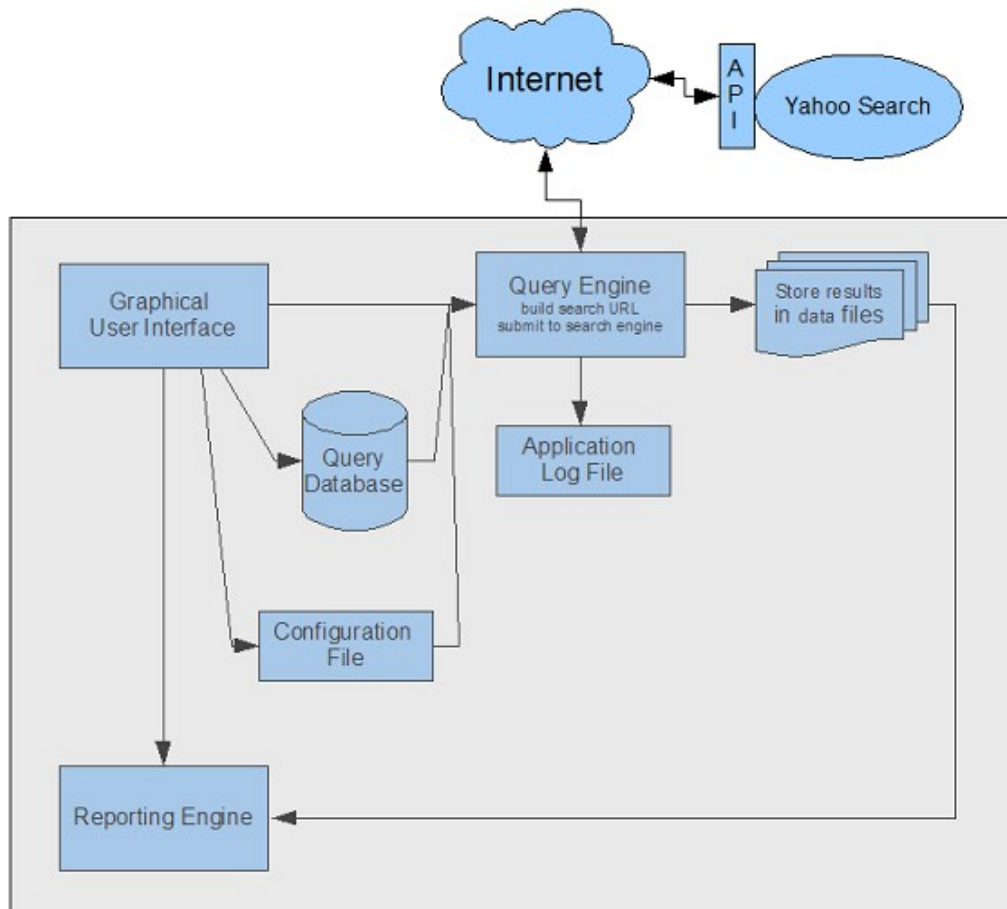


Figure 11. SEPA software architecture diagram.

3.2. Programming Language and Dependencies

3.2.1. *Programming language.* The SEPA software tool is written in the Java programming language which was chosen for its inherent object-oriented features. The application software is dynamically configurable and is easily extensible. The Java platform which includes the compiler, virtual machine and class libraries is very similar to the .NET

platform and the C# programming language. However, Java is platform independent and therefore application software developed with Java can run on multiple operating systems.

3.2.2. *Dependencies and requirements.* Table 2 below shows the Java libraries required to run SEPA.

Table 2
SEPA Java Library Dependencies

Java Library Name and Version
Java Runtime Environment 1.6.0_03
Commons HttpClient 3.1
Commons Logging 1.0.4
Log4j 1.2.15
Commons Codec 1.2

The Java run time environment is required to run SEPA and must be installed on the host machine before the software can be used (Java Downloads, 2008). Additional libraries are required to run the SEPA software but they are packaged with the software and included in the `SEPA\lib` directory. If the SEPA software is to be built from the source code, each of these jar file libraries has to be included in the host's `CLASSPATH` environmental variable. Additionally, the SEPA software application requires an Internet connection and the application's directory structure must exist on a writable file system.

3.3. Component Design

3.3.1. Interfaces. The SEPA software tool has a graphical interface which is used to add, update and delete target site configurations, manage the query database, and view results. Additionally, a query job for a target site can be launched from the command line with the command:

```
java -jar sepaConsole.jar [targetname]
```

The console interface was provided in order that query jobs could be scheduled and run autonomously. If an additional interface for the software becomes necessary, it could easily be implemented or added because the code behind the graphical and the console interfaces are separated into their own software packages. The code in the org.sepa.engine does not require any of the interface components and can stand alone.

3.3.2. Query database. The query database contains the search queries which are submitted to the Internet search service. The queries are categorized for organizational and reporting purposes. Along with the actual query string, a description, reference number and category is stored. All data is stored in a human readable text file with an extensible markup language (XML) using the UTF-8 character encoding scheme. The query database can be modified with a text editor, XML editor or maintained via the graphical interface. Figure 12 shows a graphical representation of the XML schema used. Appendix A describes how query repository from another Google hacking tool was converted to this schema using a text editor

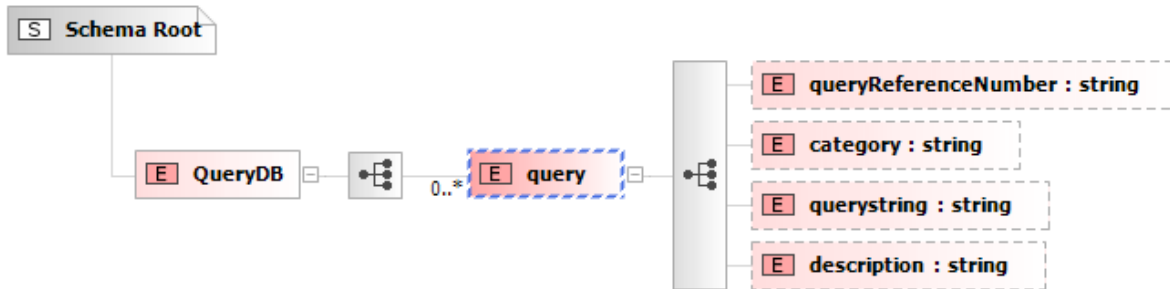


Figure 12. Query database XML schema.

3.3.3. *Target configuration file.* A target Configuration File is stored in each target folder. The target configuration file helps make the software easily extensible and configurable. A target configuration file has information which is used to not only communicate with an Internet search service, but also how to process and store the results. The configuration file is read by the query engine each time a job is started. When a target is added, a folder for the new target is created in the `targetdata` directory. The `DefaultTargetConfig.txt` which is also located in the `targetdata` directory is copied into the target folder as `targetConfig.txt` and modified for the new target. The `targetConfig.txt` file is a text file and is human readable. It can be modified with a text editor or with the SEPA graphical interface. It contains information indicating how to build each query URL, what queries to include in a query job, and what results processor to use.

3.3.4. *Query engine.* The Query Engine communicates with the Internet search service via its API. At startup, the query engine begins by reading the target configuration text file from the target directory. The parameters in the configuration file are used to build a list of

queries to submit to the search service. The arguments for each query parameter are encoded and the request is sent to the Internet search service using the `GET` method. In order not to overwhelm the search service, there is a short rest time between each submission of about 700 milliseconds, which is defined in the constants library class of SEPA. When the query engine receives the results, it uses the results processor indicated in the target configuration file to hand the results over to a results writer. The results writer is also known as the reporting engine.

A software design pattern commonly known as a *factory* was implemented to aid in dynamically deciding what results processor and query builder to use. The default query builder and results processor are made to interface with the Yahoo! Search service. If it was desired to use another search service, the SEPA software could easily be extended and configured to use different results processing and query building components. These two components would be added to the software and the only internal modifications necessary would be to the switch statements in the `SEPAFactory.java` file in the `org.sepa.engine` package. The switch statement uses the `builder` and `results_processor` parameters in the `targetConfig.txt` file to decide what builder and results processor to use. If these parameters are not found, the builder and results processor implemented for the Yahoo! Search service are used.

3.3.5. Reporting engine. The reporting engine or results writer stores results which are received from the query engine. The results are stored in a text file in human readable form and are viewable with a text editor or are more optimally viewed via the graphical interface.

Each query submitted to the Internet search service is recorded in a results file along with its query reference number, category and the time of submission. Additionally, the query URL which was built by the query engine is stored along with a results summary line indicating the total results available, total results returned and the start position of the returned result set. The Yahoo! Search API will return up to 100 results. Every returned result URL is listed underneath the query information. The queries are stored in the results file in the order in which they are submitted.

Below the listing of the queries and results of a job, there is a summary section. The job summary contains the start and end times of the job along with a query count, the number of results and the number of new results. Underneath the job summary information, a summary by category is presented. Each category summary indicates how many results were returned and how many of the results are new. Each new result URL is then listed. A result is considered new if a previous job can be found and the result URL is not in the result set of the last job previously run. Finally, the parameters as they were found in the `targetConfig.txt` file when the job was run are also listed.

3.3.6. Results repository. The results repository is where the result files can be found. There is one result file for each job run. The name of a result file is the time a job was started and is in the format of `yyMMddHHmmssZ` where *Z* indicates the RFC 822 time zone. When a target is created, a directory with the target name is created and a results directory under this target directory is also created. All result files for each respective target are stored in this results

directory. A result file is a text file and is human readable. It is optimally viewed with the graphical interface but this is not necessary. Because the results repository is made up of a collection of flat files, the application is easily installable and portable from one computer system to another. The result files are easily parsed and the results could easily be fed into another query engine.

3.3.7. Application log file. The Jakarta Commons *HttpClient* component used in the SEPA software implementation requires a Java logging framework but does not dictate which one to use. The log4j logging framework worked with the *HttpClient* component and was also found to be configurable, full featured and complete. This made it a natural fit for the SEPA software. It can be configured via the `log4j.properties` text file and no changes to the application binary have to be made to increase the logging level. The application only needs to be made aware of the `log4j.properties` configuration file by including its location in the CLASSPATH environmental variable along with other dependencies.

The log4j logging mechanism has six built-in levels of hierarchal logging and can be extended to have custom levels. The built in levels are: FATAL, ERROR, WARN, INFO, DEBUG and TRACE. All messages for SEPA are logged at the INFO level and various messages concerning possible failures are logged at the ERROR level. The reading of the query database along with the operations of the query engine are logged. The logging in the query engine includes the choosing of the query builder and results processor. Each query URL is logged with and without the encoded parameters. By increasing the logging level to DEBUG all of the

messages being logged by the SEPA software and also messages from the HttpClient component are captured. The data gathered at the DEBUG level is quite substantial and should be avoided except for debugging purposes.

The log4j framework is composed of three main components namely loggers, appenders and layouts. Graphical log viewers such as Chainsaw are available as separate components (Apache Chainsaw, 2009). The loggers component is defined and implemented in the software as described above. The appenders and layout components are attached to the logger. The appender defines where the output goes. There are a number of choices available. Some of which includes the console, files, and remote servers or services. Messages for the SEPA application software are logged to the console and then the `log4j.org.sepa.engine.log` file in the `log` directory. The log will rotate and overwrite themselves when the size reaches 100K. Up to ten log files will be stored. In addition to the message, the time, thread and location in the code are also included in the log file.

CHAPTER 4: SEPA SOFTWARE IMPLEMENTATION

As stated in section 3.2.1, the SEPA software tool was implemented with the Java programming language. While programming in Java does not require an integrated development environment (IDE), the use of the Eclipse development environment facilitated the development of the software tool (see Figure 13).

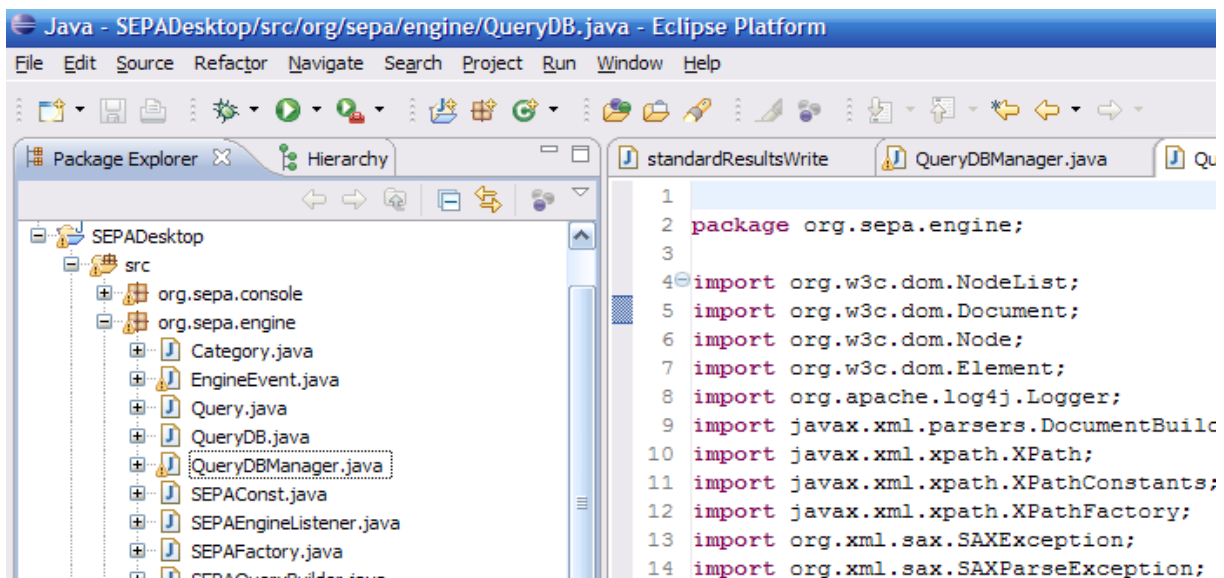


Figure 13. Partial screen shot of the eclipse IDE.

The software is organized into three major logical sections called packages. These separate packages are for the graphical interface, console interface and the core functionality or

engine. The software was organized and designed not only to be extensible and configurable but also to be easy to maintain and use.

4.1 Graphical Interface

4.1.1. Tabbed frame. The graphical user interface (GUI) was designed and implemented with Java Swing components. Swing is an API for providing a GUI for Java programs. This widget toolkit provides components such as buttons, drop down lists and text boxes along with layout managers which are used to display the widget components. Swing is part of Sun Microsystems' Java Foundation Classes (JFC). The widget components are generally customizable by extending the software classes and creating an alternate implementation.

The code for the GUI is in the `org.sepa.ui` package. The entry point or starting point for the GUI is in a file called `SEPAGUI.java`. This file defines a number of constants for the GUI such as the height and width of the interface and then instantiates the main GUI class, `TabbedFrame`. The `TabbedFrame` class extends the `JFrame` class and provides a top level container for other components. The `TabbedFrame` displays four tabs and each of these has a corresponding class. These components are the about pane, results pane, query DB pane, and a target configuration pane which all extend the `JPanel` class. Swing components are often composites of other Swing components.

Each tab displayed by the `TabbedFrame` class makes use of the `GridBagLayout` or

the BorderLayout layout manager. A layout manager defines how things will appear inside of a component and organizes or arranges them for display. JPanels are often embedded inside of other JPanels, and each panel has their own layout manager. Complex layouts can be comprised of multiple smaller layouts which are simpler and easier to manage. The layouts for the SEPA GUI were created in this manner.

4.1.2. About tab. The about pane simply displays information about the application and documents who created it, why and when. This information is formatted with the Hypertext Markup Language (HTML) and embedded in a JLabel. It is the simplest of the four tabs displayed by the `TabbedFrame`. It is also the tab that is displayed on startup of the application.

4.1.3. Query database tab. The Query DB pane provides a graphical interface for managing the query database and uses a `QueryDBManager` class. A form which includes `JButton`, `JPanel`, `JComboBox`, and `JTextArea` components is included and used to maintain the query database. The form provides create, update and delete functionality for the list of queries. The data is displayed in a hierarchical fashion with a `JTree`. The logical tree is created by assembling `DefaultMutableTreeNode`s in a `DefaultTreeModel`. First, the root node of the tree is created and then category nodes are attached to the root node. Each category node attaches the proper query nodes underneath itself. Once the `DefaultTreeModel` is assembled, it is then added to the `JTree` for display and presentation.

4.1.4. Target configuration tab. The target configuration pane provides the

functionality to update, delete and maintain target sites. It is also possible to launch a query job from this tab. Similar to the other tabs or panes, it extends the `JPanel` class and uses the `GridBagLayout` layout manager to organize and display components. The tab itself is a `JPanel` but it also contains other `JPanels` which are used to organize components logically and group them for display. Essentially, a form containing multiple forms is built.

A class called `ProgressFrame` defined in the `ProgressFrame.java` file is used to interface with the Query Engine. The `ProgressFrame` class extends the `JFrame` class and presents a dialog to start a query job. This class is instantiated when the "Run Queries Against Target" button is activated. An action listener detects when the start button is pressed and starts a query job in its own thread. This allows the rest of the application to be available while a query job runs. An activity monitor keeps track of the progress and updates the progress bar of this frame every 100 milliseconds. The number of queries completed since the job began versus the total number of queries in the job is tracked. This is done by placing completed queries into a list and removing them after a short time interval with a method called `getCurrent`. When the number of queries removed from the completed list is equal to the total number of queries queued for the query job, the query job is considered completed.

4.1.5. Results tab. The `ResultsPane` class provides and displays components for the results tab. The `ResultsTree` class is the main component of the `ResultsPane`. The job of the `ResultsTree` class is to display data and respond to events. It extends the normal functionality of a `JTree` to display data in a hierarchical fashion. By implementing the methods

required for an interface, the class can receive notification of events and respond accordingly.

The `ResultsTree` class implements the `ActionEventListener` and `EngineEventListener` interfaces. The events which the `ResultsTree` class is interested in are mouse events such as right clicks and query job completion messages published by the query engine.

A results tree is composed of nodes. There are a number of different node types attached to the tree. These are the root node, target nodes, job nodes, category nodes, query nodes and result nodes. Each of these has its own class and provides custom functionality by extending the standard `JTree DefaultMutableTreeNode` class. An example of custom functionality would be coloring the node red if a result URL underneath it was not in the last query job.

The construction of the results tree is started by creating a root node. When the root node is created, target nodes representing each target site are attached to this node. Then, each underlying node attaches their own respective underlying nodes. A target node attaches only its own job nodes which go on to attach category nodes which attach query nodes which attach the result nodes. Finally, the root node is added to the `JTree` for display.

The classes behind each node type are the `ResultsRootObj`, `ResultsTargetObj`, `ResultsJobObj`, `ResultsCategoryObj`, `ResultsQueryObj` and `ResultCell` classes. These are all used for displaying the data. The data is actually gathered by classes and static methods located in the engine package. Only one instance of a

static method exists and it only has to be used and not created or instantiated each time it is needed. The static methods for gathering target data are located in the `TargetManager` class. An instance of a `SEPAResultsReader` class is created for each job found. The `SEPAResultsReader` class is considered to be part of the reporting engine and processes all result files. It takes the data in a result file and puts it into data objects for use by the results tree.

4.2. Console Interface

The console interface was created to allow a query job to be scheduled and run autonomously or from a remote session where a graphical interface might not be possible. The console interface provides a limited set of functionality. If a target site is passed in as a parameter, a job will be automatically run for this target site and the program will exit when the job is done. When a target site is not provided, options to exit or run a job for a site are presented.

4.2.1. Menu system. The code for the console interface is separated logically from the code for the graphical interface and the engine. It resides in the `org.sepa.console` package. The `SEPAConsole` class defined in the `SEPAConsole.java` file has the main method and is the entry point for the console interface. Several other classes are defined in the console package. An abstract class `SEPAMenu` is used to define data structures and methods that a menu should have and offer. Another class, `SEPAMenuBuilder` is used to create menus and add items. These two classes are meant to decouple or abstract the menu operations from the actual

implementation and make the menu system easily extensible. The files `consoleMenu.java` and `TargetConsole.java` contain most of the actual menu system.

The `TargetConsole` class in the `TargetConsole.java` file lists the target sites and processes the input. It uses the `SEPAMenuBuilder` class and the `TargetManager` class located in the engine package to create and present the list of target sites. Once the target choice is determined control is returned to the main method where an instance of a query engine is created to run a query job. After the query job is run, the main menu with options to run a query job or exit is presented again.

4.3 Engine Code

The `org.sepa.engine` package contains the source code that is the crux of the system. It is comprised of over twenty files and classes. The `SEPAQueryEngine` class defined in the `SEPAQueryEngine.java` file is at the center of it all and uses most other components in the package directly or indirectly. Additional components used throughout the application are classes which are used to store data, offer static utility methods or defined constants. The following subsections details how the different classes and components work together to create the SEPA engine code base.

4.3.1. TargetManager class. Previously, the `TargetManager` class was mentioned. It contains all of the methods for creating, updating, and removing a target site. This includes the

associated target configuration file and result files after they are initially written. The `SEPAQueryEngine` class uses the `readConfig` method to get a hash map (key, value pairs) of parameters contained in the target configuration file. It needs these to determine how to build the list of queries and run the query job. The target configuration file is read each time a query job is started.

4.3.2. SEPAQueryBuilder and SEPAResultsProcessor classes. A design goal of the software was to make it easily extensible and maintainable. In part this is because of the ever changing nature of the search service. The `SEPAQueryEngine` class makes use of a software factory pattern and abstract classes to help meet this design goal. Two classes defined as abstract are the `SEPAQueryBuilder` and `SEPAResultsProcessor` classes. They do not actually do anything except define the methods required by the classes actually doing the building and processing. One reason for the abstraction is so that a software factory pattern can dynamically determine which query builder and result processor to use at runtime.

A class extending the `SEPAQueryBuilder` class is used to build a list or queue of queries to submit to the search service and a class extending the `SEPAResultsProcessor` class is used to process the results from the search service. By default, the actual query builder and result processor used when communicating with the Yahoo! search service are the `yahooQueryBuilder` and `yahooXMLResults` classes respectively. If a different query builder or result processor is desired then a class could be implemented and plugged into the application by changing only the switch statement in the `SEPAFactory.java` file. Parameters

in the target configuration file indicate what query builder and result processor to use. Figure 14 shows SEPAFactory.java file containing the SEPAFactory class which decides what results processor and query builder to use.

```
package org.sepa.engine;

import java.util.Map;
import org.apache.log4j.Logger;

/**
 * SEPAFactory - decide what query builder to use
 * a query builder submits a queue of queries to the Query Engine
 *
 */

public class SEPAFactory {

    static Logger log4j = Logger.getLogger("org.sepa.engine.SEPAFactory");

    /**
     * getBuilder - static method to decide what builder to use
     * gets specification from parameter list which is an associative array
     * uses builder key to find value and decided what builder to instantiate
     * returns a standard builder if none is specified
     * @param queryParameters list of parameters read from configuration file
     * @return SEPAQueryBuilder
     */
    public static SEPAQueryBuilder getBuilder(Map<String, String>
queryParameters){

        SEPAQueryBuilder queryBuilder;
        String builder = queryParameters.get("builder");

        if(builder != null && builder.equals("yahoo")) {
            queryBuilder = new yahooQueryBuilder();
            log4j.info("query builder initalized: " + builder
        } else
        {
            queryBuilder = new yahooQueryBuilder();
            log4j.info("parameter for query builder not found");
            log4j.info("default query builder initalized:
yahooQueryBuilder");
        }

        return queryBuilder;
    }

    /**
     * getResultsProcessor - static method to decide what results processor to
     use

```

```

*/
public static SEPAResultsProcessor getResultsProcessor(String target,
    Map<String, String> queryParameters) {

    SEPAResultsProcessor resultsProcessor;
    String processor = queryParameters.get("results_processor");

    if(processor != null && processor.equals("yahoo"))
    {
        resultsProcessor = new yahooXMLResults(target, queryParameters);
        log4j.info("results processor initialized: " + processor);
    } else
    {
        log4j.info("parameter for results processor not found");
        resultsProcessor = new yahooXMLResults(target, queryParameters);
        log4j.info("default results processor initialized: yahooXMLResults");
    }
    return resultsProcessor;
}
}

```

Figure 14. Software factory pattern making run time decisions.

The purpose of the query builder is to create a list of query URLs to submit to the search service. It uses the queries in the query database and the parameters in the target configuration file to do this. The parameters indicate how the query URL should be built. A URL is built and the queries from the database are inserted into the URL. A queue of query objects containing two search URLs with and without encoded parameters is created and sent to the query engine for processing. The URL without the encoded parameters is only used for logging and display. The URL with the encoded parameters is submitted to the search service.

4.3.3. Query submission. The actual submission of the query URL to the search service is done by the query engine with the Java HTTP client. Figure 15 shows the code for an HTTP request. An HTTP client and a request object are both created. All requests are submitted with the HTTP Get method. The request object is sent in as a parameter to the client's

executeMethod method. The resulting status code is checked for errors and any error encountered is logged. The query object is placed in the queue for completed queries. The result is sent to the result processor for processing and the connection is closed. Each query is submitted asynchronously with a short rest time between each request. The duration of the rest time is defined in the SEPAConst class.

```
try {
    client = new HttpClient();
    method = new GetMethod(encoded_request);
    statusCode = client.executeMethod(method);
    statusLine = method.getStatusText();

    if (statusCode != HttpStatus.SC_OK)
    {
        log4j.error("Method failed: " + statusLine);
        log4j.error("HTTP Status Code " + statusCode );
    }
    rstream = method.getResponseBodyAsStream();
} catch (HttpException e) {
    log4j.error("Fatal protocol violation: " + e.getMessage());
    e.printStackTrace();
} catch (IOException e) {
    log4j.error("Fatal transport error: " + e.getMessage());
    e.printStackTrace();
} catch (InterruptedException e){
    log4j.error(e.getMessage());
    e.printStackTrace();
} catch (Exception e){
    log4j.error("Query Engine exception ");
    log4j.error(e.getMessage());
    e.printStackTrace();
} finally
{
    completedQueries.add(queryObj);
    resultsParser.processResults(rstream, queryObj, statusCode,
    statusLine);
    method.releaseConnection();
}
```

Figure 15. Code for an HTTP request.

4.3.4. Result processing. The result processor takes the result of an HTTP request and

processes it into result objects which are handed over to the result writer for storage. The default result processor processes XML results from the Yahoo! search service. XML Path Language (XPath) is used to parse the XML results retrieved from the search service. Any errors encountered are logged but should not stop the program. The data from the XPath queries is put into a `SEPAResultsObj` object and handed over to a `SEPAResultsWriter` class for storage.

The `SEPAResultsWriter` converts result data from the object and puts it in a text file. As each query completes, the results writer adds the result data to the file. The `SEPAResultsObj` class, which actually contains the data also contains the method for formatting the data for output. This is done by overriding the `SEPAResultsObj` object's `toString` method. The result writer also uses a `SEPAResultsSummary` object to maintain summary information such as query counts, result counts by category and new results by category. This is done as each result object is put into the data file. This summary information is added to the end of a results file when a query job is complete. The parameter information from the target configuration file is also added.

The `SEPAResultsReader` class converts the data from the text file and puts it back into a list of `SEPAResultsObj` objects. It processes the results data and the summary data. All of the logic for parsing a result file is centralized and contained in this class. In general, the results reader would be used by the code collectively considered to be the reporting engine but it is also used by the results writer. In order to know if a result is new, previous results need

to be known.

4.3.5. Engine events. Also included in the engine package is an `EngineEvent` object and a `SEPAEngineListener`. The `EngineEvent` object is used in conjunction with the listener to notify other classes inside and outside of the engine package that an engine event has happened. The way it works is that whoever needs to publish or announce an event creates an `EngineEvent` object with the proper data inside of it. The notifier then submits the `EngineEvent` object to the `SEPAEngineListener`. The listener maintains a list of subscribers or classes which are interested in receiving notification of events. When the `EngineEvent` object is handed over to it, the listener notifies the subscribers. Currently, only the `SEPAResultsWriter` uses this mechanism to announce that a new data file is available. The results tab in the GUI listens for this event and displays the new results after receiving notification.

4.3.6. Query database. Finally, the query database is described. The loading, updating and saving of the database is divided into two classes. These are the `QueryDBManager` and the `QueryDB` classes. The `QueryDB` class provides the core functionality to maintain the database. This is mainly the reading and writing of the XML data file. It converts the XML data in the text file to a list of `Query` objects with XPath queries. Additionally, there are methods to create a new reference number (unique id) for a query, return a distinct list of categories in the database and a list of queries associated with a category. The `QueryDBManager` is intended to be a go-between for classes outside of the engine package

and the `QueryDB` class itself. Classes in the `org.sepa.ui` package use the `QueryDBManager`, which calls methods in the `QueryDB` class to provide its functionality. This provides a layer of security, abstraction and delegation for the query database.

CHAPTER 5: EXPERIMENT IMPLEMENTATION

The implications of having public repositories accessible and indexable by search engines has been discussed and are apparent. The proposed solution to mitigate the risk of having an information leak is to automate the submission of advanced search queries to the Yahoo! Search service. The software which was designed and implemented for this has also been discussed. The question to be answered is if the results from the submission of automated search queries are of comparable quality and effect as those submitted manually. This chapter discusses what was done to determine this.

5.1. Experiment Implementation

All experimentation was focused on sites belonging to educational institutions. In most university or educational environments every student, faculty and employee generally has web publishing capabilities. Additionally, a large number of organizations or groups utilize the services of the institution to publish information. Ostensibly, organizations that have more content and more people publishing content to the world wide web are more susceptible to a harmful information leak.

In addition to the general error free operation of the software which was tested

throughout the design and implementation phases, the design and usability of the software was verified using a qualitative approach with one professional penetration tester and two system administrators. The software was introduced to them and after they surveyed SEPA the simple question was asked, "If your boss asked you to use this tool periodically to survey a target site would it be helpful?"

A quantitative approach was taken to verify that advanced search queries submitted by automated means return results comparable to the same advanced search query submitted manually. It was first verified that the advanced search query worked for at least one case on the world wide web. Then the advanced search query was limited to a particular site for testing. The queries were organized into seven categories. These were Backup Files, Configuration Management, Devices, Error Messages, Log Files and Reports, Login Portals, Privacy Related, Technology Profile. The queries in each category were designed to answer questions about the target network which an administrator might ask his or herself in a general sense. These questions can be found in Table 3. The questions mirror the category names.

Table 3
General Questions for a Systems Administrator

-
- | Question |
|--|
| • Are there any backup files in public directories? |
| • Are there any log files or network scan results publicly available? |
| • Are there any devices available on the Internet which should not be? |
| • Are there any error messages with too much information? |
| • Are there any documents with sensitive information such as passwords?
Are there any architecture and policy details which should not be public? |
| • Are there any ways to easily obtain server versions? |
| • Are there any ways to locate login portals? |
| • Are there any configuration files or default files in public directories? |
| • Is there any source code in public directories? |
-

Most of the queries were found in publicly available on-line repositories of Google hacking queries. Some were crafted or modified with the idea that they would work well with the SEPA software and the Yahoo! Search service. Sometimes a query could fit into two or more categories but was generally considered a better fit for one or another.

The submission of the automated queries was done in quantities of 200. This is a little more than the number of queries available in two of the popular repositories found on-line. There were no time constraints for the completion of a job. However, a short rest time of a half-second between the submission of each query was used. This rest time was used in order to not overload the search engine service. The categories and their query counts are displayed in Table 4.

Table 4
Categories and Query Counts

Category Name	Count
Backup Files	19
Configuration Management	33
Devices	12
Error Messages	43
Log Files and Reports	16
Login Portals	17
Privacy Related	24
Technology Profile	36

Because search services use a different index for queries submitted via the web user interface than those submitted autonomously via the API (McCown, F. & Nelson, M., 2007), one of the goals of this research was to verify that the automated results are comparable to those from the web user interface. To verify that this was the case, each query was submitted to the web user interface and the result counts were compared to the result counts of the same query submitted through the API.

CHAPTER 6. RESULTS

This chapter includes problems that were encountered using the Yahoo! Search API, manual vs. automated submission result counts and findings broken down by categories. The results are an assessment of how effective the queries were at finding what they were designed to find.

6.1. API Bug

Initially, only queries from on-line repositories were used. The results returned by queries submitted by automated means to the Yahoo! Search API were not the same as those results from queries submitted manually through the web interface. Not all of the advanced search operators are honored in the same manner as they are when submitted to the interface. Particularly, the `intitle` operator did not work properly. Many security queries search for a particular document being exposed by a directory listing. A page showing a directory list has the "Index Of" in the title so this operator is very common for security queries searching for a particular document. An example of a query using the `intitle` operator to search for a log file of previously typed commands for a mySQL database is:

```
intitle:"Index of" .mysql_history
```

The results from the automated queries using the `intitle` operator were often returning

unexpected and inexplicable results. When the cached version of a result is viewed, the terms that were used in the search are highlighted. By viewing the cached page it was apparent that the numeral one was used in place of the `intitle` text. The queries would find a document containing the numeral one and whatever else was specified in the query. While, this produced unexpected results, the information was not totally useless since the additional part of the query was security related. This bug was reported by more than one user in the `yws-search-general` message group (Yahoo! Search General Developer Support, 2008).

Due to the bug encountered with the automated queries submitted with the Yahoo! Search API, all queries were modified to not use the `intitle` operator. The returned results due to these modifications were less specific and prone to false positive results. The false positive results are not a security threat but is an explicable result due to the way the query was formed. When the occurrence of the "Index Of" string is not limited to the title of the document the possibility of a false positive result is increased. The occurrence of this exact phrase is not common but the likelihood is not negligible. For instance, it might happen that a query looking for the "Index Of" phrase in any location of a document would pickup a document published by the library system explaining how to use an index of periodicals. If the same query were to apply the `intitle` operator and it worked properly the same document being published by the library system would be excluded from the result set because the "Index Of" phrase is found in the body of the document and not in the title.

6.2. Manual vs Automated Query Results

Each one of the queries that was used for testing was submitted manually via the web user interface. The queries were restricted to two different institutions sites of different sizes. Size being measured by the number of pages indexed by the Yahoo! search service. The number of pages indexed was simply measured by the number of results available from a query limiting results to the site with no other operators or search terms being employed. These two institutional sites were the `southernct.edu` and the `yale.edu` sites having 75,220 and 2,335,882 indexed pages respectively.

The result counts of manual submissions were compared to the automated result counts. If a result count was greater than 100, then 100 was used. This is because SEPA will only display up to 100 result URLs. If a query returns more than 100 results, then it probably needs to be reviewed and refined for the situation. A summary of the results can be seen in Table 5 and are represented with a bar graph in Figure 16. Appendix B contains a table which shows the result counts for each query. It is evident that the index used for the web interface is different from the one which serves the automated queries. In general the results counts were very close and usually within ten percent of each other. If the results differed by more than ten percent, it was generally due to one query producing a large result set that was not present in the other result set. For instance, the error messages category displayed large differences due to a single page with dynamic content generating an error message. The same page was indexed multiple times with

different parameters being submitted to the script generating it. A query submitted by the web user interface had more than 100 results URLs for this one page while the API interface did not yield any.

Six to eight hours was spent submitting the 200 queries manually for both sites surveyed. It was not possible to submit all of the queries for both sites in one sitting. However, all of the queries for one site were done in a single three to four hour sitting. The same queries submitted autonomously with SEPA took minutes. This goes to show the amount of time it would take for an administrator to police a site manually.

Table 5
Summary of Result Counts by Category

Category	Query Count	Result Counts			
		southernct.edu		yale.edu	
		Manual	Automated	Manual	Automated
Backup Files	19	8	8	370	367
Configuration Management	33	39	36	219	181
Devices	12	12	13	100	100
Error Messages	43	102	7	441	219
Log Files and Reports	16	0	0	2	2
Login Portals	17	188	172	750	705
Privacy Related	24	109	115	1058	1046
Technology Profile	36	279	219	627	588
Totals	200	737	570	3567	3208

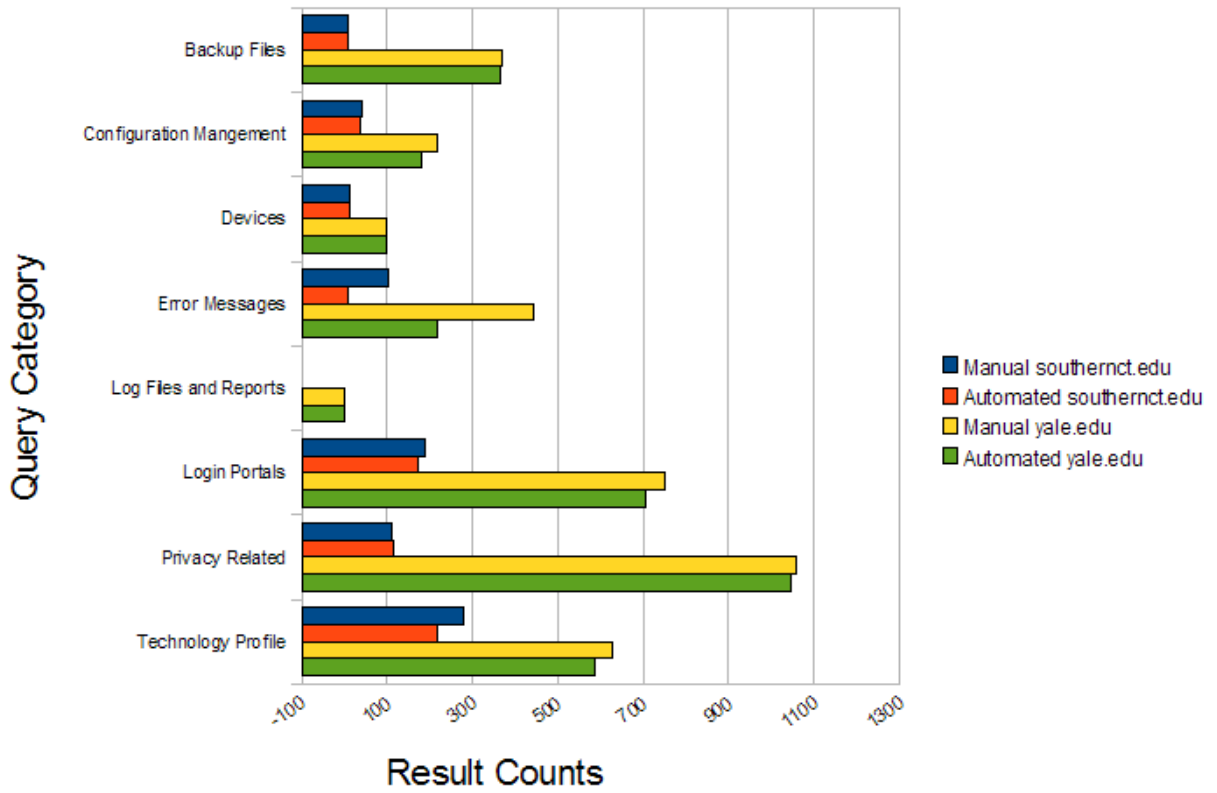


Figure 16. Graphical representation of manual result counts vs automated result counts.

6.3 Findings

One of the objectives of this research was to mitigate the risk of having an information leak by automating the submission of advanced search queries to the Yahoo! Search service. Purposely, the term "information leak" has not been explicitly defined because the definition would change from institution to institution or from administrator to administrator. Additionally, the definition would probably be congruent with the security posture of the institution. For instance, one administrator might go to great lengths to obfuscate the web server type and version, while another may feel this is not worth the effort because there is no way to ultimately hide this information. However, some things are indisputable information leaks. For example, if a document is labeled confidential and then contains a user name, password and a URL to an information resource, this is clearly an information leak.

The effectiveness of each query was rated. A query either identified no vulnerability, a possible vulnerability or a definite vulnerability. Only results from queries targeted for the `southernct.edu` site were considered. Table 6 shows the counts summarized by category followed by a discussion of findings for each category in the following subsections.

Table 6
Summary of Query Ratings by Category

Category	Query Count	Query Ratings for southernct.edu		
		No Vulnerability Identified	Possible Vulnerability Identified	Definite Vulnerability Identified
Backup Files	19	16	3	0
Configuration Management	33	32	1	0
Devices	13	12	1	0
Error Messages	43	40	0	3
Log Files and Reports	15	14	0	1
Login Portals	17	11	6	0
Privacy Related	24	13	10	1
Technology Profile	36	31	5	0
Totals	200	169	26	5

6.3.1. Backup files. It is often mistakenly thought by people publishing content to public web directories that if there is no link to data in public web directories, this data will not be discovered or indexed by a search engine. Queries in this category look for data that was placed in public directories under this misguided assumption. For both targeted sites queried, this category had a number of results that were interesting. Comma delimited data files, Sql dumps, Microsoft Access databases, web logs and directories named backup, bak, save, and sav were all easily found. Figure 17 shows a data dump from a MySQL database with user names and passwords. The user names have been obscured with an image editing tool.

```

--
-- Table structure for table `users`
--

CREATE TABLE `users` (
  `id` tinyint(4) NOT NULL auto_increment,
  `username` varchar(20) NOT NULL default '',
  `password` varchar(32) NOT NULL default '',
  PRIMARY KEY (`id`)
) ENGINE=MyISAM DEFAULT CHARSET=latin1 AUTO_INCREMENT=10 ;

--
-- Dumping data for table `users`
--

INSERT INTO `users` VALUES (1, ' ', 'd06d83b507e76168fbde306d912fbcd3');
INSERT INTO `users` VALUES (5, ' ', 'a42118c8e4bd95d0319ed23249f787e0');
INSERT INTO `users` VALUES (4, ' ', '2ad3bb035c92dcd07a5271c9d9d9b86c');
INSERT INTO `users` VALUES (6, ' ', 'ae66db116570f0d4c35036470297897a');
INSERT INTO `users` VALUES (7, ' ', '8d3aaed6ca9bea54c3beb9aa0c4093d1');
INSERT INTO `users` VALUES (3, ' ', '010c8d472c5c9500bcf23816db0f9e3f');
INSERT INTO `users` VALUES (2, ' ', '906a38f24c1234f7d45e283d41b632bc');
INSERT INTO `users` VALUES (8, ' ', '42d4def7fe18cf6edd6d71033dee2f91');

```

Figure 17. Data dump found on a target site with user names and passwords.

6.3.2. *Configuration management.* Search queries in this section look for exposed configuration files such as password files, command history files or other information indicating a misconfiguration such as an exposed intranet (an internal network that should not be public). A result URL in this section would usually indicate an indisputable problem or information leak. A false positive result or results that are not truly a problem do sometimes appear. The false positive results are usually from a manual page or similar source explaining how the configuration file should appear and not the actual configuration file. Unexpectedly, some results in this category showed a development site that was indexed despite the existence of a robots.txt, which is supposed to restrict what directories a search engine is supposed to index. The file was either not setup correctly or completely ignored by the search engine bot (an

automated agent) traversing the site. It is also possible that a link in other publicly available content caused these URLs to be included in the search engine's index.

6.3.3. *Devices* Twelve of the two hundred queries (six percent) were in this category. Additional queries for this category could be found in on-line repositories and easily integrated into the SEPA tool. The queries sought a number of devices such as printers, copiers, DVR clients and web cameras. The easiest and most exciting devices to find are web cameras. A simple query, `inurl:webcam` which was designed to look for any type of web camera came up with some surprising results for both target sites. Some of the result URLs had both webcam and other words such as adult, fetish, free and naked in them. This indicates that the servers on these target sites had been previously compromised or something had gone wrong with the indexing of the public pages.

6.3.4. *Error messages*. Queries in this category are probably of the most use to administrators, programmers and attackers. The search service is very much like a free automated quality assurance team. It was found that queries in this category rarely suffered from false positive results. The results often detail directory structures and are a perfect indicator of the technology being utilized by the site. Sometimes an information leak or problem is currently resolved but a cached page memorializes it. This was seen in the result set of both target sites by observing the result URL without any error messages and the cached page with the error messages.

6.3.5. Log files and reports. The sixteen queries in this category searched for very specific documents revealing the results of security scans, access logs and web statistic reports. Found on both target site were occurrences of WS_FTP.log files. This log file from a popular file transfer program often reveals directory structures, filenames and user names associated with the transfer of the files from the client to the server. Figure 18 shows interesting result URLs from a query looking for WS_FTP.LOG files. The URLs are interesting because they are associated with an on-campus credit card system.

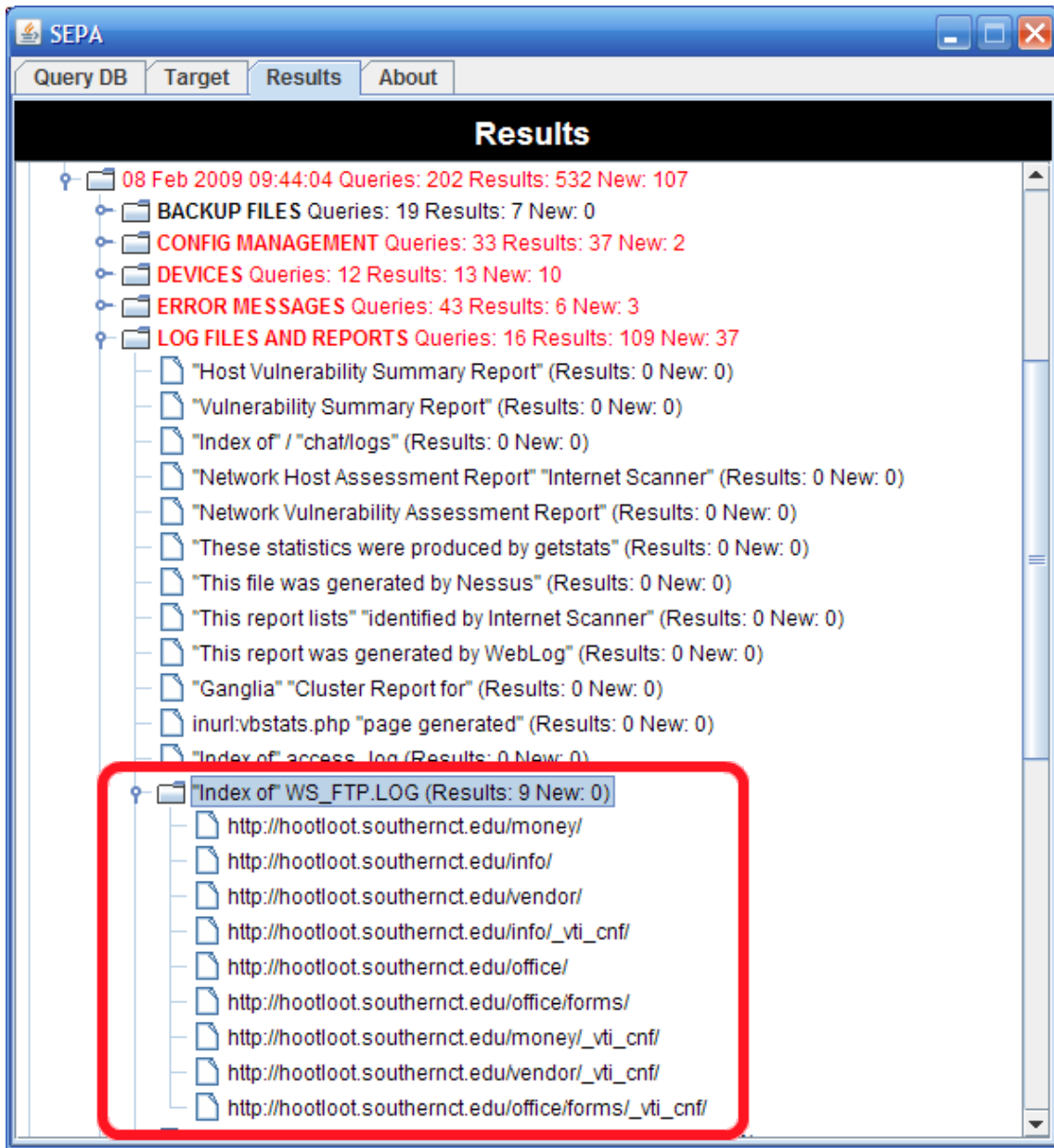


Figure 18. Interesting result URLs from a query looking for WS_FTP.LOG files.

6.3.6. *Log in portals.* It turns out that log in portals are easy to find. Results from this category were almost never falsely positive. Attackers and legitimate web clients alike might be interested in the location of the log in portal. Administrators should know where the log in

portals are but this information is useful because it should be noted what portals are available to the public (not all should be) and what information has been published about accessing these.

6.3.7. Privacy related. The queries in the privacy category suffered the most from false positive results. However, some very interesting things were found amongst these. Credit card validation source code (see Figure 19), a confidential letter containing a link to a security report along with the required user name and password (see Figure 20), a spreadsheet detailing purchases and an embedded image of a credit card (see Figure 21) were all found with queries in this category. For presentation in this thesis the sensitive information in each image has been obscured with a smudge mark but was found intact and visible.

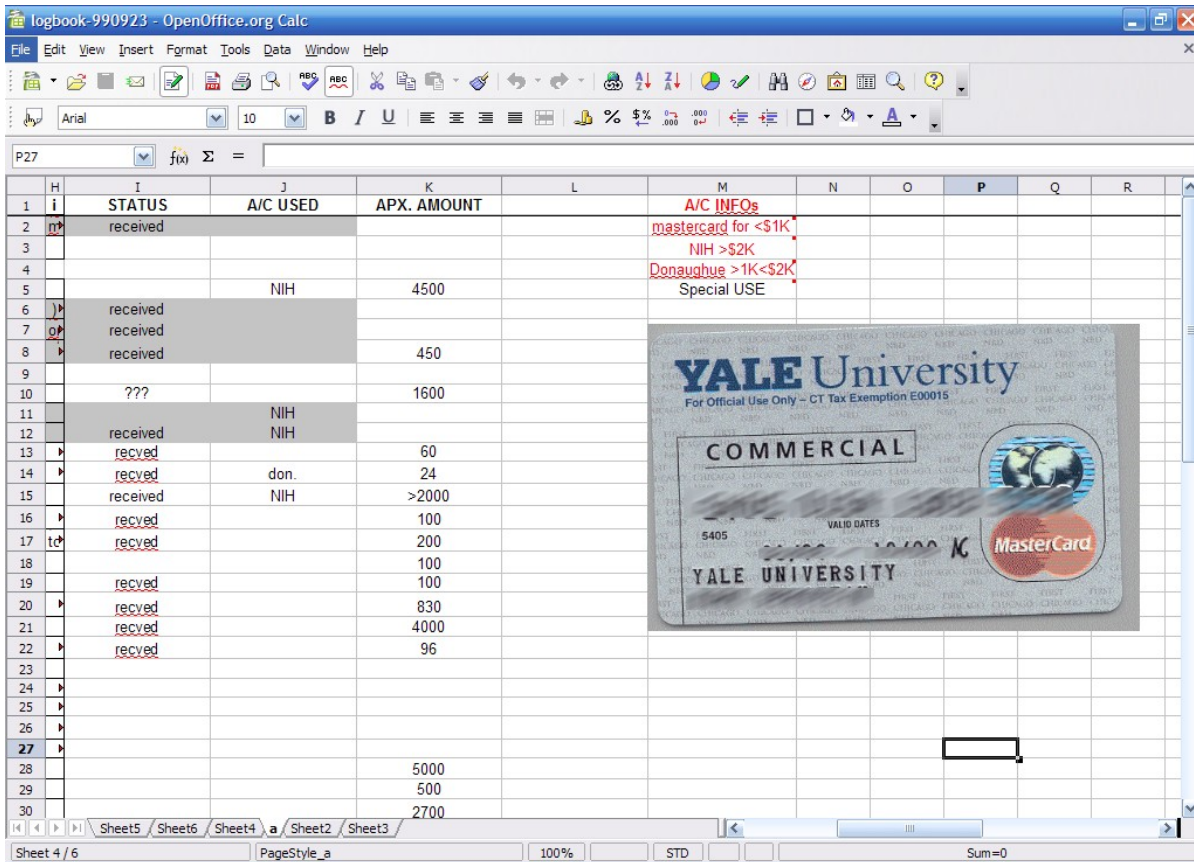


Figure 21. A spreadsheet with a credit card image embedded in it.

6.3.8. *Technology profile.* A technology profile is meant to detail what technologies are being used. It really does not matter if one or one hundred results are returned. Often, the simple fact that there was a result indicates what technologies are being utilized on a site. For example, it is fairly easy to determine the server type and version by looking at page footers which are automatically generated by the server. Queries in this section could often be cross listed in other sections. For example the results of a query in the privacy related category was successful at not only revealing credit card validation source code but also revealed details about the web server type, version and modules being used (see Figure 19). Numerous results, not just

those in this section, were successful at detailing the technologies employed by a site.

6.4. Software Usability

Two administrators and one professional penetration tester surveyed the software. The first administrator reported that he liked the queries but would rather use spidering software to secure the network. Spidering software would traverse the web site on its own but not be able to take advantage of search queries found in on-line repositories. Additionally, he felt that the job should be scheduled and the results emailed. The SEPA software does have a command line interface to support the autonomous running of query jobs. A separate script would have to be written to email the results file, the SEPA software does not have an embedded Simple Mail Transfer Protocol (SMTP) engine to independently send email.

The second administrator had little to no feedback, it seemed that he had trouble analyzing the results. However, the professional penetration tester reported that he appreciated the lightweight portable design of the tool and commented on the way the results appeared. The red highlighting of the results not found in the previous query job or delta display was reported to be unique and useful. He did have a couple of suggestions for adjustments to the tool but the final comment was that "this tool will be in my war chest moving forward".

CHAPTER 7: CONCLUSION AND FUTURE WORK

Using a search engine for untoward purposes generally for the reconnaissance phase of an attack has been dubbed as "Google hacking". This thesis presented a software tool to help mitigate the risk of an information leak on the Internet by querying a search engine in the same way that an attacker might. The querying of the search engine was automated so that it could be done quickly, effectively and periodically. Because Google no longer allows automated searches, the question to be answered was if a web site's security could be improved by using the Yahoo! search service. Additionally, this research sought to determine if automated query results through an API were any different than results from queries entered manually through Yahoo's web user interface.

7.1. Conclusion

It was shown that the queries submitted manually were using a different index than those submitted by automated means. However, the results from this index were not so far out of sync that it posed a problem. This confirms the findings in a paper titled "Search Engines and Their Public Interfaces: Which APIs are the Most Synchronized?" (McCown, F. & Nelson, M., 2007). The number of results from an advanced search query rarely exactly matched those from the web user interface but were generally within ten percent. Submitting the queries by

automated means through the API was 300 times faster than submitting the same queries manually and as a result it is not something that could be managed on any regular basis. This further confirms that the process has to be automated.

The advanced search queries were categorized for reporting purposes. Some categories had better results than others. For instance, queries looking for log in portals and error messages were highly successful and accurate while queries designed to discover personal information suffered from more false positive results. The software was designed to handle the unavoidable false positives by highlighting results in red that were not in the last job. This feature made the results easier to review each time a query job was run. It was found that queries returning over one hundred results should be refined and that larger sites naturally had more false positives. Although certain queries produce numerous results, an analogy has been made to an aspect of physical security. If an attacker is willing to go through the dumpster hunting for valuable information then they would be willing to look through any number of result URLs.

The professional penetration tester found the SEPA software tool very useful and currently uses the tool for professional security audits. However, the system administrators reviewing SEPA did not find it as useful as it was hoped they would. In general this would indicate that the SEPA software tool is a good fit for security experts and that the tool does what was it was intended to do. The general populus might require some formal instruction in using the search engine as a reconnaissance tool and an introduction to SEPA. The concepts of Google hacking can often be confusing to those not familiar with computer security problems and

concepts.

Numerous findings on both the `southernct.edu` and the `yale.edu` sites made the use of the software with the Yahoo! search service worthwhile. In one instance a credit card image was found embedded in a spreadsheet that also listed IT equipment purchases. The person whose name was on the credit card was contacted. They were very concerned and thankful for the information. They also asked if any more sensitive information could be found. The reason for the information leak was that it was not properly understood that files which did not have a direct link to them and accessible through the Internet could end up in a search engine's index. A large number of files were exposed and accessible because of this.

If a page containing sensitive information is indexed and should not be it should be requested that it be removed from the search service's index. Each search service has a means to request that information be removed from its index and cache. With the Yahoo! search service, an administrator would have to authenticate themselves by embedding a meta tag in the home page of the site or by placing a text file containing an authentication key in the root directory. After an administrator is authenticated, a page can simply be requested to be removed from the index with a web form. Google has a similar URL removal form. Figure 22 shows the Google removal request form for requesting that a page be removed from the cache and the URL from the index.

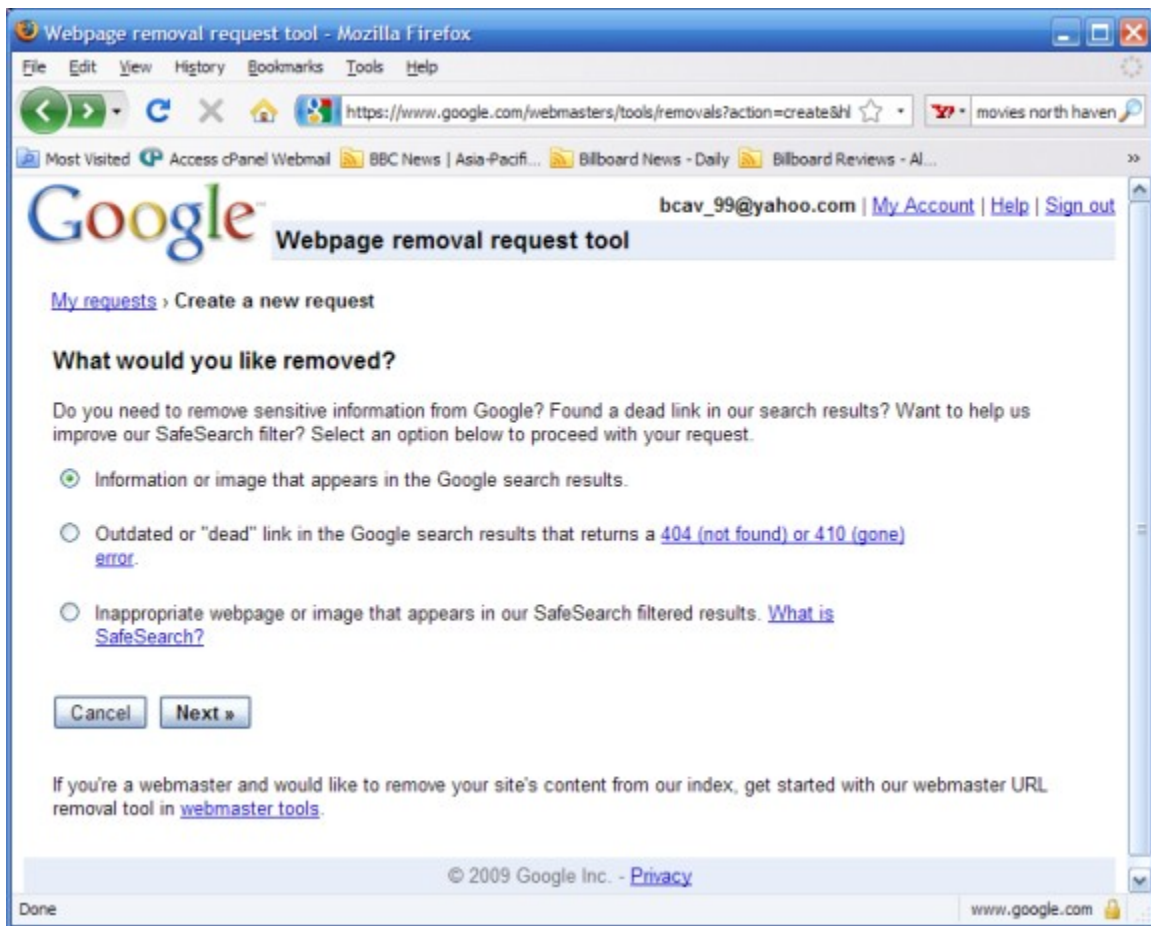


Figure 22. Google removal request tool.

7.2. Future Work

There are many additional features that could be implemented for the SEPA software tool. This includes support for multiple databases and additional search services. Every effort was made to make the software configurable and extensible. More configuration options could be presented. Especially in the area of viewing results. For instance, result URLs that were not in

the last query job are highlighted in red. A user might want a result to be highlighted in red only if it has never been seen before. Finally and possibly most importantly, a central query repository or on-line database could be established. The SEPA software tool could then subscribe and automatically update its query database with the latest definitions.

APPENDIX A

Converting the Goolag Database to a SEPA QueryDB

In December of 2007, the well known hacker group based in Lubbock Texas called Cult of the Dead Cow published an application which queries the Google search engine outside of its terms of agreement. The application came with an XML query database of over 1400 queries. This database of queries was converted for use by the SEPA application.

To make the query database useable by the SEPA software, the search and replace functionality of a full featured text editor was used to convert the XML tags in the Goolag database to the SEPA format.

<GoolagScan>	became	<QueryDB>
<Dork>	became	<query><queryReferenceNumber>
</Dork>	became	</queryReferenceNumber>
<Category>	became	<category>
</Category>	became	</category>
<Query>	became	<querystring>
</Query>	became	</querystring>
<Comment>	became	<description>
</Comment>	became	<description></query>
</GoolagScan>	became	</QueryDB>

When converting a database there might be some special character considerations.

These special characters could cause problems while reading the XML and should be replaced with their equivalents.

If these characters are found in the data they should be replaced.

< should be replaced with <
> should be replaced with >
" should be replaced with "
' should be replaced with '
& should be replaced with &

APPENDIX B

Detailed Manual vs Automated Query Result Counts

Table 7
Search Queries and Result Counts

Advanced Search Query	southernct.edu		yale.edu	
	Manual	Automated	Manual	Automated
Category: Backup Files				
"Index of" "index.html.bak"	0	0	0	0
"Index of" "index.php.bak"	0	0	0	0
"Index of" "index.jsp.bak"	0	0	0	0
"Index of" ".htpasswd" "htpasswd.bak"	0	0	0	0
inurl:backup "Index of" inurl:admin	0	0	0	0
"Index of /backup"	0	0	0	0
"# phpMyAdmin MySQL-Dump" inurl:txt	0	0	0	0
"# phpMyAdmin MySQL-Dump" "INSERT INTO" -"the"	0	0	0	0
"# Dumping data for table" inurl:".csv"	0	0	1	1
inurl:backup inurl:mdb	2	2	100+	100+
inurl:".bak"	0	0	7	7
inurl:save	0	0	100+	100+
inurl:".sav"	2	2	60	57
"Index of" secring.bak	4	4	0	0
"#mysql dump" inurl:sql	0	0	0	0
"# Dumping data for table"	0	0	1	1
"Index of" htpasswd.bak	0	0	0	0

"Index of" passwd passwd.bak	0	0	1	1
Category: Backup Files Totals	8	8	370	367
<hr/>				
Category: Config Management				
"Index of" "fpcount.exe"	0	0	0	0
"Index of" "msadcs.dll"	0	0	0	0
"Index of" "trillian.ini"	0	0	0	0
allinurl:"auth_user_file.txt"	0	0	0	0
"Index of" "config.php"	0	0	2	2
"Index of" "/etc"	39	36	100+	100+
inurl:xls username password email	0	0	0	0
inurl:htpasswd htpasswd	0	0	6	5
"Index of" ".htpasswd" "htgroup" -"dist" -apache -htpasswd.c	0	0	0	0
"Index of" administrators.pwd	0	0	0	0
"Index of" etc shadow	0	0	65	61
"Index of" secring.pgp	0	0	0	0
inurl:config.php dbuname dbpass	0	0	0	0
"Index of" master.passwd	0	0	0	0
"Index of" .mysql_history	0	0	0	0
"index.of" passlist	0	0	0	0
inurl:passlist.txt	0	0	0	0
"Index of..etc" passwd	0	0	0	0
"Index of" spwd.db passwd -pam.conf	0	0	0	0
"Index of" .bash_history	0	0	0	0
"Index of" .sh_history	0	0	0	0
"Welcome to phpMyAdmin" AND " Create new database"	0	0	0	0
inurl:install/install.php	0	0	0	0
"Netscape FastTrack Server Home Page"	0	0	0	0
i_index.shtml "Ready"	0	0	0	0
inurl:tech-support inurl:show Cisco	0	0	0	0
<hr/>				

inurl:'cgiirc.config'	0	0	0	0
"Index of" robots.txt	0	0	35	12
"osCommerce" inurl:admin filetype:php	0	0	0	0
"phpMyAdmin" "running on" inurl:"main.php"	0	0	0	0
"Index of" service.pwd	0	0	0	0
"Index of" pwd.db	0	0	1	1
"Welcome to Intranet"	0	0	0	0
Category: Config Management Totals	39	36	219	181
<hr/>				
Category: Devices				
aboutprinter.shtml	0	0	0	0
inurl:camera.php	0	0	0	0
inurl:webcam	12	13	100+	100+
"powered by webcamXP" "Pro Broadcast"	0	0	0	0
"Live View / - AXIS"	0	0	0	0
"Live View / - AXIS" inurl:view/view.sht	0	0	0	0
webeye inurl:login.ml	0	0	0	0
"DVR Web client"	0	0	0	0
camera linksys inurl:main.cgi	0	0	0	0
"Home" "Xerox Corporation" "Refresh Status"	0	0	0	0
"Smoothwall Express" inurl:cgi-bin "up * days"	0	0	0	0
inurl:"MultiCameraFrame?Mode="	0	0	0	0
Category: Devices Totals	12	13	100	100
<hr/>				
Category: Error Messages				
"the page cannot be found" inetmgr	0	0	0	0
"supplied argument is not a valid MySQL result resource"	1	1	37	33
"access denied for user" "using	0	0	80	42

password"				
"ORA-00921: unexpected end of SQL command"	0	0	0	0
"A syntax error has occurred" inurl:ihtml	0	0	0	0
"access denied for user" "using password"	0	0	80	42
"An illegal character has been found in the statement" -"previous message"	0	0	0	0
"Can't connect to local" warning	0	0	72	40
"Chatologica MetaSearch" "stack tracking:"	0	0	0	0
"detected an internal error [IBM] [CLI Driver][DB2/6000]"	0	0	0	0
"Fatal error: Call to undefined function" -reply -the -next	0	0	4	4
"Incorrect syntax near"	0	0	0	0
"Incorrect syntax near" -the	0	0	0	0
"ORA-00933: SQL command not properly ended"	0	0	3	3
"PostgreSQL query failed: ERROR: parser: parse error"	0	0	0	0
"Syntax error in query expression " -the	0	0	0	0
"Unclosed quotation mark before the character string"	0	0	0	0
"Warning: Cannot modify header information - headers already sent"	100+	5	5	4
An unexpected token "END-OF-STATEMENT" was found	0	0	0	0
"Error Diagnostic Information" "Error Occurred While"	0	0	0	0
filetype:asp "Custom Error Message" Category Source	0	0	0	0
"the page cannot be found" inetmgr	0	0	0	0

"the page cannot be found" "internet information services"	0	0	0	0
"500 Internal Server Error" "server at"	0	0	0	0
"Under construction" "does not currently have"	0	0	14	3
"supplied argument is not a valid MySQL result resource"	1	1	37	33
"mySQL error with query"	0	0	0	0
"ORA-00921: unexpected end of SQL command"	0	0	0	0
"ORA-00936: missing expression"	0	0	11	9
inurl:sitebuildercontent	0	0	0	0
inurl:sitebuilderfiles	0	0	0	0
inurl:sitebuilderpictures	0	0	0	1
"You have an error in your SQL syntax near"	0	0	0	1
"Supplied argument is not a valid PostgreSQL result"	0	0	0	0
warning "error on line" php sablotron	0	0	0	0
"the page cannot be found" "2004 microsoft corporation"	0	0	0	0
"seeing this instead" "test page for apache"	0	0	1	1
"Test Page for Apache" "It Worked!" "on this web"	0	0	0	0
"ASP.NET_SessionId" "data source="	0	0	0	0
"Warning: Division by zero in" "on line" -forum"	0	0	1	1
"Unable to jump to row" "on MySQL result index" "on line"	0	0	0	0
"Warning: mysql_connect(): Access denied for user: '*@*' "on line" -help -forum"	0	0	80	1
"500 Internal Server Error"	0	0	16	1

Category: Error Messages Totals	102	7	441	219
<hr/>				
Category: Log Files and Reports				
"Host Vulnerability Summary Report"	0	0	0	0
"Vulnerability Summary Report"	0	0	0	0
"Index of" / "chat/logs"	0	0	0	0
"Network Host Assessment Report" "Internet Scanner"	0	0	0	0
"Network Vulnerability Assessment Report"	0	0	0	0
"These statistics were produced by getstats"	0	0	0	0
"This file was generated by Nessus"	0	0	0	0
"This report lists" "identified by Internet Scanner"	0	0	0	0
"This report was generated by WebLog"	0	0	0	0
"Ganglia" "Cluster Report for" inurl:vbstats.php "page generated"	0	0	0	0
"Index of" access_log	0	0	0	0
"Index of" WSFTP.LOG	0	0	0	0
"statistics of" "advanced web statistics"	0	0	0	0
"Usage Statistics for" "Generated by Webalizer"	0	0	2	2
Category: Log Files and Reports	0	0	2	2
<hr/>				
Category: Login Portals				
"log in"	100+	100+	100+	100+
"wbem" compaq login	0	0	0	0
inurl:admin login	1	1	78	51
inurl:changepassword.asp	0	0	0	0
"Index of" upload.asp	0	0	5	4
"Index of" AT-admin.cgi	0	0	20	6
"Index of" global.inc	16	14	100+	100+

"osCommerce" inurl:admin inurl:php	0	0	0	0
"Remote Desktop Web Connection"	0	0	1	1
"Terminal Services Web Connection"	0	0	0	0
inurl:manyservers.htm	0	0	0	0
admin login	19	12	100+	100+
inurl:main.php phpMyAdmin	0	0	0	0
inurl:main.php Welcome to phpMyAdmin	0	0	0	0
inurl:login	25	21	100+	100+
"change password"	19	16	100+	100+
inurl:"password"	8	8	100+	100+
"default login"	0	0	46	43
Category: Login Portals Totals	188	172	750	705
<hr/>				
Category: Privacy Related				
"Index of" "people.lst"	0	0	0	0
"not for distribution" confidential	0	0	2	2
"Index of" finance.xls	0	0	11	6
inurl:finances.xls	0	0	0	0
"Index of" inbox dbx	0	0	0	0
"Index of" dead.letter	1	1	100+	100+
"Index of" inbox	0	0	15	14
"Index of" mail	0	11	100+	100+
"Index of" ws_ftp.ini	0	0	0	0
inurl:admin inurl:xls	0	0	0	0
mystuff.xml "index of"	0	0	0	0
site:edu grades admin	23	19	100+	100+
"Index of" / lck	5	5	100+	100+
inurl:admin inurl:asp inurl:userlist	0	0	0	0
inurl:admin inurl:userlist	0	0	0	0
"Index of" personal	22	21	100+	100+
"Index of" private	8	8	100+	100+
inurl:index.of.protected	0	0	0	0

"Index of" protected	4	4	100+	100+
"Index of" secret	4	2	100+	100+
"Index of" secure	2	2	100+	100+
"Social Security (SSN)"	0	0	1	1
"Security Incident"	7	7	37	31
inurl:private	33	35	100+	100+
Category: Privacy Related Totals	109	115	1058	1046
<hr/>				
Category: Technology Profile				
"Gallery in Configuration mode"	0	0	0	0
inurl:Custva.asp	0	0	0	0
"Powered by mnoGoSearch - free web search engine software"	0	0	0	0
inurl:footer.inc.php	0	0	0	0
inurl:info.inc.php	0	0	0	0
inurl:search.php vbulletin	0	0	0	0
"Select a database to view"	0	0	0	0
"filemaker pro"				
"Index of c:\Windows"	0	0	0	0
"Index of " winnt	0	0	16	10
"Apache HTTP Server"	0	0	11	11
"documentation"				
"Welcome to IIS 4.0"	0	0	1	1
"Test Page for Apache" "It Worked!"	0	0	0	0
"powered by opensbd" +"powered by apache"	0	0	0	0
"Index of" master.passwd	0	0	0	0
"This summary was generated by wwwstat"	0	0	0	0
"robots.txt" + "Disallow:" filetype:txt	0	0	0	0
"Thank you for your order" +receipt	0	0	1	1
index.of cgiirc.config'	0	0	0	0
"Index of" haccessctl	0	0	8	5
inurl:htaccess Basic	0	0	2	2

"Index of " Apache "server at"	53	24	100+	100+
inurl:ipsec.conf -manpage	0	0	0	0
inurl:ipsec.secrets -history -bugs	0	0	0	0
inurl:ipsec.secrets "holds shared secrets"	0	0	0	0
"Index of" mt-db-pass.cgi	0	0	0	0
"phpinfo.php" -manual	0	0	2	2
"cacheserverreport for" "This analysis was produced by calamaris"	0	0	0	0
"Select a database to view" "filemaker pro"	0	0	0	0
"Welcome to PHP-Nuke" congratulations	0	0	0	0
"YaBB SE Dev Team"	0	0	0	0
inurl:shop "Hassan Consulting's Shopping Cart Version 1.18"	0	0	0	0
6.0.2.5516	19	19	100+	100+
firewall -glossary -personal	46	46	100+	100+
VPN gateway	100+	100+	86	56
"Apache/2.2.2 (Fedora)"	0	0	100+	100+
"Server at"	61	30	100+	100+
Category: Technology Profile Totals	279	219	627	588

REFERENCES

- Apache Chainsaw. (2009). *Apache Chainsaw*. Retrieved February 1, 2009, from <http://logging.apache.org/chainsaw/index.html>
- Cole, E. (2002). *Hackers Beware Defending Your Network from the Wiley Hacker*. Indianapolis, IN: New Riders.
- Danhieux, P. (September 2004). Danhieux What is Santy bringing you this year?. Retrieved December 1, 2007, from http://www.sans.org/reading_room/whitepapers/threats/1592.php
- Foundstone, Inc.,(2007). *SiteDigger Software*. Retrived on December 1, 2007, from <http://www.foundstone.com/us/resourcesfree-tools.asp>
- Granneman, S. (March 10, 2004). *The perils of Googling, A tool for good and evil*. Retrieved January 20, 2008, from http://www.theregister.co.uk/2004/03/10/the_perils_of_googling/.
- Java Downloads. (2008). *Java Downloads for All Operating Systems*. Retrieved November 8, 2008, from <http://www.java.com/en/download/manual.jsp>
- Kotadia, M. (January 14, 2005). *Google hacking expected to boom in 2005*. Retrieved December 16, 2007, from <http://news.zdnet.co.uk/security/0,1000000189,39184116,00.htm>
- Lancor, L., & Workmarm, R., (2007). *Using google hacking to enhance defense strategies*. Retrieved September 30, 2007, from <http://portal.acm.org/citation.cfm?id=1227504.1227475>
- Long, J. (2005). *Google Hacking for Penetration Testers*. Rockland, MA: Syngress Publishing.
- Long, J. (2007a). *GooScan Software*. Retrieved September 15, 2007, from

http://johnny.ihackstuff.com/downloads/task_cat_view/gid,16/

Long, J. (2007b). *Google Hacking Database*. Retrieved September 15, 2007, from

<http://johnny.ihackstuff.com/ghdb.php>

Long, J., & Aaron B., & Foster J., & Hurley, C., & Vincent, L., & Petruzzi, M., & et al. (2006).

Penetration Tester's Open Source Toolkit. Rockland, MA: Syngress Publishing.

McCown, F. & Nelson, M. (2007). *Search Engines and Their Public Interfaces: Which APIs are the Most Synchronized?*. Retrieved December 5, 2008, from

<http://www2007.org/htmlposters/poster868/>

Mowse (2003). *Google-Knowledge: Exposing Sensitive Data with Google*. Retrieved December

16, 2007, from <http://www.gosecure.ca/SecInfo/>

Peake, C. (July 16, 2003). *Red Teaming: The Art of Ethical Hacking*. Retrieved December 1,

2007, from http://www.sans.org/reading_room/whitepapers/auditing/1272.php

Richardson R. (2007). *2007 CSI Computer Crime and Security Survey*. Retrieved November 29,

2007, from <http://i.cmpnet.com/v2.gocsi.com/pdf/CSISurvey2007.pdf>

Scambray, J., & McClure, S., & Kurtz, G. (2001). *Hacking Exposed: Network Security Secrets*

& Solutions. Berkley, CA: Osborne/McGraw-Hill.

Sensepost, Inc.,(2007). *Wiko Software*. Retrived September 9, 2007, from

http://www.sensepost.com/research_tools.html

Verton, D. (February 11, 2002). *Web sites seen as terrorist aids*. Retrieved December 16, 2007,

from <http://www.computerworld.com/industrytopics/energy/story/0,10801,68181,00.html>

Yahoo! Developer. (2008a). *Register your application on the Yahoo! Developer Network*.

Retrieved October 21, 2008, from <http://developer.yahoo.com/wsregapp/>

Yahoo! Developer. (2008b). *Web Search Documentation for Yahoo! Search Web Services*. Retrieved October 21, 2008, from

<http://developer.yahoo.com/search/web/V1/webSearch.html>

Yahoo! Search General Developer Support. (May 8, 2008). *yws-search-general : Message: Re: Difference between yahoo web search and yahoo web service search*. Retrieved June 14, 2009, from <http://tech.groups.yahoo.com/group/yws-search-general/message/1320>