Searching in Anubis Reports

Michael Weissbacher
mweissbacher@iseclab.org

Technical Report
December 5, 2011
Course: Seminar (mit Bachelorarbeit)
Supervisor: Ulrich Bayer

Abstract

Anubis is a Windows Malware–Analyzer based on TTAnalyze [1], information that is gained from running the analysis is stored partly in a database and partly in XML reports. There was no frontend to search through the database except using handmade queries on the database client, there was also no way of searching within the XML–reports. I was handed over the task to find a solution that would make searching through the database and XML–reports efficient and user–friendly. Full–text search can be constrained by a filter that works with the database turned out to be the best solution.

1 Introduction

Anubis [2] is a tool for analyzing the behavior of Windows PE–executables with special focus on the analysis of malware. Execution of Anubis results in the generation of a report file that contains enough information to give a user a good impression about the purpose and the actions of the analyzed binary. The analysis is based on running the binary in an emulated environment and watching i.e. analyzing its execution. The analysis focuses on the security–relevant aspects of a program's actions, which makes the analysis process easier and because the domain is more fine–grained it allows for more precise results.

The generated output includes detailed data about modifications made to the Windows registry or the file system, about interactions with the Windows Service Manager or other processes and of course it logs all generated network traffic. The analysis process stores information in a relational database as well as in XML files (reports). The reports were not searchable at all, the database was only searchable through a command–line interface.

2 Previous Work

Searching through large datasets is not a novel problem, many products are available. I evaluated several full–text search engines that seemed to be promising. COTS frontends for the database search were not evaluated since its much easier to build them from scratch.

2.1 Requirements

- Full–text search: searching through the generated reports as text and ignoring the XML–structure and metadata. Important features are: word stemming, boolean expressions, stop–words, hit highlighting and a decent scoring system that would rank relevant hits higher.

- Database search: unlike full–text search data has to be specified exactly, there is no ranking. Preferably the interface should be dynamic or easy to adopt to changes in the database design.
- Large dataset: the analysis data covers 2 Million reports (1.5 TB) at the moment of writing and grows at about 4000 reports a day. The reports are gzip compressed. Speed is important, response times should be reasonable.

2.2 Other surrounding conditions

- External/managed search was ruled out because of data confidentiality. Letting the reports be indexed by Google would cover the full–text part, but would not help with database related queries.
- Full–text and database search is necessary. Two interfaces would work, but being able to use both search options from one interface would be a major plus since it would be more comfortable and allow narrowing of searches. A full–text search product should either be able to adopt to specific database tables or be customizable so the feature can be added manually.

2.3 Google Search Appliance

Google Search Appliance (GSA) is Google's enterprise search product. It comes as hardware with preinstalled software in three sizes: Google Mini (300.000 documents), GB-7007 (10.000.000 documents) and GB-9090 (30.000.000 documents). The look and feel is similar to google.com but can be customized. Like the internet version of Google the appliance crawls through defined sources in the intranet. GSA is very feature rich, it understands 200+ different filetypes, Lotus Domino, Microsoft Sharepoint crawling etc. Indexing databases is possible, tables can be used as metadata for documents, it is possible to "join" documents with tables as metadata, but specific fields can not be queried. Generally the GSA is a black box (software and hardware–wise), some features can be configured, but everything else is hard to modify or unmodifiable. [3] [4]

2.4 Solr/Lucene

Apache Lucene is an open source project, the flagship project is an information retrieval library written entirely in Java. Solr is a Lucene sub–project, it is an enterprise search product originally developed by CNET and now by the Apache foundation. It adds abstractions to Lucene that make common tasks easier, with Solr a basic functioning system can be put together quickly. Lucene alone can not be deployed that fast.

The Solr server can be queried with HTTP GET requests and returns XML documents that can be associated with a stylesheet. Indexing data also works via HTTP, scripts for posting documents are included. It comes with a lot of high level work for Lucene but is still customizable. Solr can be run in a Jetty or Tomcat container, this works out well. Solr is not a finished "product" that can be installed within minutes, it lacks a start–stop script and some other programs that might be important for deployment, they need to be selfmade. What makes Solr attractive is that changes can be made everywhere, it is something that can be hacked fast – if necessary. [5] [6]

2.5 MySQL stored search index

MySQL is used as database system for Anubis. Since version 3.23.23 full–text search is supported. Currently reports are stored directly in the filesystem, not as blob or text fields in the MySQL database. Various reasons back up this decision: it would not scale up with the amount of reports, the overhead (locking, SQL queries) is not necessary and makes backups is easier. Storing reports is not the task which the database is employed for. Full–text search in MySQL is not very feature rich, e.g. there is no hit highlighting. MySQL full–text might work for small indices, but is the wrong tool in this case. [7]

2.6 Summary of evaluation

GSA is much more feature rich than Solr, but most of them are not required here. GSA can be configured to search through databases and associate them as metadata to reports, but searching is not as fine grained as required (query specific columns). GSA can crawl the intranet automatically, this makes sense in a heterogeneous network with changing data sources. Anubis reports only need to be indexed when they are created or changed, this information can be obtained from the database to index only specific reports. Solr covers all required full–text search features and its results can be post-processed by a filter that queries the database. Building one interface that accomplishes both tasks is possible. Solr is free, which is also a nice feature.

MySQL stored search index was ruled out because it would not be able to cope with the dataset. Solr beats GSA because it fulfills all requirements for full–text search and more modifications are possible. A uniform interface can be build that works with the Solr index and the MySQL database in the background.

3 Work

Solr would not accomplish all required tasks out of the box. This is an overview of the customizations I made.

3.1 Full–text indexing

Solr's schema.xml file defines the layout of the documents that will be indexed. Multiple fields can be defined for a document and various data types can be used, this can be quite handy if no relational database is used in the first place. Since Solr will only be used to search for reports I used three fields: the result_id to keep results unique, a timestamp to detect whether the report file on disk was changed since the last indexing and a text–field that holds the whole XML report.

Sample scripts for indexing are provided (post.jar, post.sh), but they are slow and not very feature rich. post.jar was so slow that it would have taken over 6 months to index the old reports. Reports had to be transformed into Solr's postable document schema before they could be used by these scripts. I created a script "update_index.py" that would handle the indexing–task more efficiently. Targets can be specified by ID and range, with filenames or by age (index reports which were created within the last x–hours), reports get transformed to Solr's document scheme automatically. The major issue with speed was waiting for I/O, I fixed the performance problems by increasing parallelity and reducing the commit frequency. Using curl directly instead of the provided scripts also helped.

Only recent report versions store the subjects' network dump in a separated file, older ones keep it as nodes in the XML file. Network dumps can be considered as garbage for full–text search, all valuable information is being recognized by the analysis task and ends up in the database. Hence those nodes can be pruned to save space. Reports larger than 10MB could be reduced by up to 99%. I was able to decrease the index's total size by about 80%.

3.2 Database indexing

A table was necessary that would store a brief analysis summary: "simple_profile", several programs were made to extract information into the table. Also a run-once script was made to index old reports, freshly generated reports fill out the table automatically.

3.3 Integration

Solr does not come as an installable package, some scripts have to be self made to integrate with the environment.

- start–stop: Solr can not be handled by the regular start–stop daemon. The Solr PID is different from the PID of the container. A stop key can be handed as argument when starting, to shut down the stop key has to be sent again. Should the regular way of shutdown fail a manual kill and later a non–blockable kill signal will be sent, the PID is being obtained through the ps output.
- health monitoring: Solr version 1.12 was slightly troublesome, it would die now and then without obvious reason and required constant monitoring to start it up if it would fail, or restart it if it would not react to queries or become very slow. With version 1.13 there were no such problems and the script is not required in the current productive environment.
- cron scripts: a script that is triggered hourly will index all reports that have been generated since the last run, a script that is triggered daily will run an optimize command on the index which decreases response times by about 10% compared to an unoptimized index. Committing updates to the index while optimize is running will lead to problems, hence updating is suspended during this time. After optimize is done all reports that were created since the start of optimize will be indexed.

3.4 Web Interface

The first approach to design the interface would only include a textbox and a submit button. I would parse the string and split it up into a part that would be sent to Solr and a part that would be used with the database. Columns of simple_report could be checked by <column name>:<value>, boolean expressions could be used – I had to operate on the Solr and MySQL sets in the PHP–script. At first glance the interface looked easy to use, but users were forced to know all columns of simple_report, I had to solve the problem in a different way.

What I learned from the first attempt was that knowing the whole table structure was not accepteble for the user, hence all options must be visible on the page. Putting the whole table layout on the website with fields for each column (textfield for strings, checkboxes for yes/no fields etc.) worked, but it made the whole site too bloated. On one hand the information must be present, but on the other it should not distract the user. A piece of Javascript would hide the database part of the form, expanded by clicking on a button, otherwise it will not block much of the screen. By naming the box "Filter" it was obvious that it would post–process the Solr results (which get evaluated first) and usage of the Filter–box is not mandatory.

A change in the database structure forced me to change some code, instead of just fixing it I decided to redo the whole interface and make it more dynamic so it could survive future database structure changes without needing to be changed. The current script can be configured with a list of tables and will pull the table structure from MySQL to build the frontend dynamically. The whole filter–box can also be removed by changing settings, this can be necessary should users be required to search which have no permission to see the details of the table structure.

4 Conclusion

Users are able to search through reports using the database as metadata through a single interface. The backend is Solr full–text search and MySQL. Using a text–box for Solr and adding a form that would work as Filter on the results which are returned from Solr turned out to be a good solution. Different ways of interaction with the sources are used, but the user ends up with one merged result that fits both criteria. The feedback was very positive, the filter–box is easy to use, there seems to be no confusion about the way the interface works.

References

[1] Ulrich Bayer, Christopher Kruegel, and Engin Kirda. Ttanalyze: A tool for analyzing malware, 2006.

[2] Anubis: Analyzing Unknown Binaries. URL `http://anubis.iseclab.org`.

[3] GSA Datasheet. URL `http://www.google.com/enterprise/pdf/gsa_datasheet.pdf`.

[4] Butler Technology Audit of GSA 6.0. URL `http://www.google.com/enterprise/pdf/butler_tech_review.pdf`.

[5] Lucene Website. URL `http://lucene.apache.org`.

[6] Solr Website. URL `http://lucene.apache.org/solr`.

[7] MySQL Full–text docs. URL `http://dev.mysql.com/doc/refman/5.0/en/fulltext-search.html`.