Message Digest Detective

NSRL RDS API

Munieshwar (Kevin) Ramdass – *New York University Tandon School of Engineering*

11th May 2017

1 | Project Overview

The Message Digest Detective is an API to access the National Software Reference Library (NSRL) [1] in an efficient manner. NSRL publishes RDS or hash sets periodically. The most recent at the time of this report is the December 2016 RDS containing more than 104 million benign hashes of files found on computers. This hash set is growing and is expected to grow rapidly. NSRL publishes their RDS as ISO files or DVDs. This is efficient if the zipped RDS is relatively small. However, in the coming years, it will be difficult to manage such large RDS. Currently, the December 2016 RDS is about 2.7 GB compressed and 12.8 GB uncompressed. In the future, it will be larger. Developers, cyber investigators, and security personnel can in-cooperate the RDS in their programs. However, after every new update of an RDS, said personnel must redownload the RDS and rein-cooperate the RDS into their software. This is not efficient. There should be a way to update and sync the data, or at least have a single RDS to reference. The RDS itself is hard to transfer over networks and will take up space. Message Digest Detective provides a single source to quickly reference entries in the RDS. The ultimate motivation behind this program is to be able to quickly scan all the files from a sensitive directory like the System32 folder on Windows machines for their respective SHA-1 hashes and compare them against the RDS to determine if a file is benign or not. At a very high level, the Message Digest Detective can be seen as a mirror of VirusTotal [4].

2 | Solution

Background: In a previous attempt to perform searches in the RDS with very limited resources, I implemented a bucket system where the RDS was split into several chucks and can be stored in such a way that allows for multiple read operations on all of the chunks. This does not solve the space problem. Furthermore, with each new update of the RDS, I will have to call a script to split the RDS into buckets and store them. Bulk scans were quick. Bulk scans where especially quick due to the organization of the RDS. The SHA-1 hashes in the RDS were stored in order of number value. Therefore, calculating which bucket a particular SHA-1 hash belonged to was easy once the start and end value of each bucket was known. Once the bucket is found, a linear search is done on said bucket. This linear search is seen as happening in constant time per SHA-1 hash since all buckets except the final bucket, which can be smaller, were the same size and not a function of how large the entire RDS is. For bulk scans, all that is required is a sort by SHA-1 values from the client's scan and then a linear search with respect to the scanned hashes. The overall time to scan was based on how many hashes the client had to compare.

Current Application: In the implementation of the Message Digest Detective described in this report, an API was created so that there is one central RDS that everyone can query. This RDS can be updated at any time and will not affect anyone's program when there is an update to the RDS. The RDS has multiple fields or metadata per file. My previous attempt only allowed for quick access to SHA-1 hashes but no other fields like MD5, ProductCode, OpSystemCode, etc. Using Elasticsearch solves this issue. Quickly searching for any fields is possible with Elasticseach. Furthermore, putting this RDS on the cloud is best since other programs and plugins can utilize this hash set without storing it locally or having a local machine crunching through it. The Message Digest Detective is made scalable using Amazon Web Services (AWS) [3] as described in the following sections.

3 | Architecture

Message Digest Detective is built using Amazon Web Services (AWS) [3]. The core components include Amazon Elastic Beanstalk for deploying a web user interface, Amazon Elasticsearch for storing the NSRL's hash set and detail files, Amazon Simple Queue (SQS) service for the queuing of many client bulk transactions, Amazon Simple Notification (SNS) service for updating clients about their bulk transactions results.

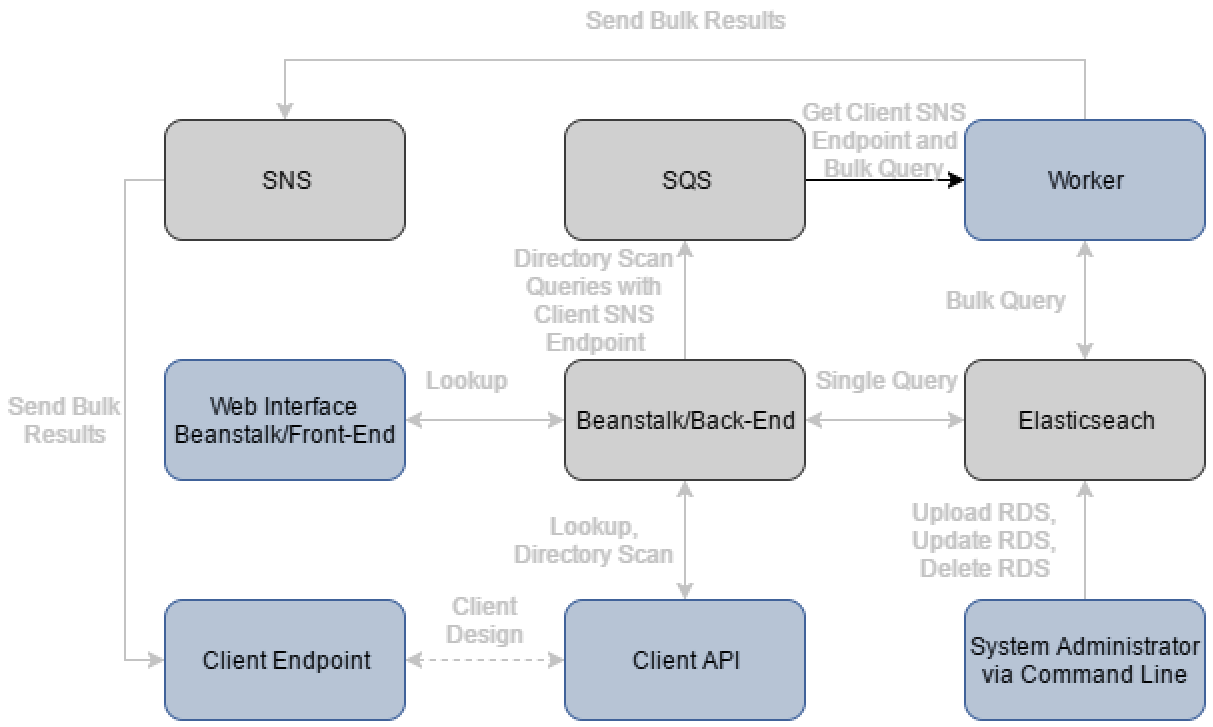
Send Bulk Results
SNS
SQS
Get Client SNS Endpoint and Bulk Query
Worker
Directory Scan Queries with Client SNS Endpoint
Bulk Query
Send Bulk Results
Web Interface Beanstalk/Front-End
Lookup
Beanstalk/Back-End
Single Query
Elasticseach
Lookup, Directory Scan
Upload RDS, Update RDS, Delete RDS
Client Endpoint
Client Design
Client API
System Administrator via Command Line

Figure 3.1: Architecture illustrating the components and their relative interactions.

From the above architecture diagram, there are three interfaces that are supported. There is a web interface that works very much like VirusTotal, a client API that uses Message Digest Detective resources to do bulk transactions via the command line, and a system administrator command line interface to push an RDS onto Elasticsearch, update the RDS, or delete the RDS. Clients using the web interface may enter any of the RDS fields (SHA-1, MD5, FileName, ProductCode, OpSystemCode, etc.) to search for entries of files with those attributes. Clients will eventually be able to see and associate with relating ProductCode and OpSystemCode based on the results. The client command line interface allows for a similar interaction for searching. A client may specify more than one field to do a refined search. A client may do a bulk search by specifying a directory s/he wishes to scan. SHA-1 hashes computed from a scan of all the files within a directory will be queued using SQS and then a worker will search the Easticsearch cluster and send the results to an endpoint provided by the client when entering desired arguments for command. At this time, the endpoint used is a phone number. Results will be in the form of a text to notify if the user has unknown or malicious files from his/her scan.

RDS Format: The current RDS has contains a list of metadata line by line where each line represents metadata about a benign file or software. Each line is comma delimited reflecting the following fields: SHA-1, ProductCode, OpSystemCode, SpecialCode, FileName, FileSize, CRC32, MD5. SHA-1 and MD5 are unique markers for individual files. ProductCode is the company that installed the file or software onto a machine. OySystemCode is a number used to represent a specific operating system. SpecialCode is a marker used occasionally, otherwise it is left as the empty string. FileName and FileSize are self-explanatory. CRC32 is the cyclic redundancy check. This information can be used to trace back the origins of a file. The RDS comes with other metadata files that describe ProductCode, OpSystemCode, and another sub metadata field named Mfg. These files contain information about the company that implemented the file or software.

4 | Implementation Overview

4.1 | AWS Elastic Beanstalk

AWS Elastic Beanstalk is used for the web interface. Clients may insert a keyword they believe is in the RDS and the request will query the AWS Elasticsearch cluster. The back-end of this application is a Python 2.7.13 Flask application using HTML/CSS and JavaScript as its front-end. POST requests from bulk transactions are handled here too. In such a case, the bulk hashes from the client's command line interface are sent to the AWS SQS service. Elastic Beanstalk handles load balancing for requests. Servers (or instances of servers) will grow or shrink to the demand received.

4.2 | AWS Elasticsearch

AWS Elasticsearch stores the RDS. All queries are done here. Instance count may change depending on the query demand. Storage may need to be adjusted to support the size of the growing RDS.

4.3 | AWS Simple Queue Service

When a bulk scan is requested via the client via the command line interface, a list of all the SHA-1 hashes will be sent to AWS Elastic Beanstalk which will then send the list to the AWS SQS service for later processing. The term 'later' here is used loosely.

Ideally a worker node will take care of the queued list of hashes immediately. However, the queuing is used for load balancing reasons. This is to ensure that the AWS Elasticsearch cluster are not over loaded with queries when a lot of bulk scans are requested by multiple clients.

4.4 | AWS Simple Notification Service

Once a worker node receives a result of the bulk scan, the result is sent to the AWS SNS service which will then send the results to an endpoint for the client to see the results. In this case, this endpoint is a phone number. A text will be sent to the client to view a list of hashes that were not found in the RDS and their respective directory local to that client's machine.

4.5 | Scripts

Scripts found in my GitHub (section 7) are the application.py, client.py (command line interface), worker.py, and input.py (system administrator command line interface). All scripts are written in Python 2.7.13.

The web interface is handled by application.py which is a Python 2.7.13 Flask application. It is deployed onto AWS Beanstalk and can be updated at any time.

The client command line interface is client.py which does two functions: searches for specific keyword(s) (specifically key field(s)) or run bulk scans on a local directory. The bulk scans are done by computing the SHA-1 hashes of all the files in said directory and sending the list of hashes to AWS Beanstalk endpoint with another client choice of receiving endpoint – in this case, it is a phone number. The key field search is handled by specifying which field(s) a client thinks the desired keyword can be found. This is a one-time call that will return a JSON object of all the files that match in the AWS Elasticsearch cluster.

The worker.py script handles all the lists of hashes that are in the AWS SQS service. This script will query the AWS Elasticsearch cluster by comparing each hash to the RDS and when finished, it will send the results to the AWS SNS service. The AWS SNS service will then send the notification of results to the client provided endpoint. This script can work locally on a machine or can be run on AWS Beanstalk. Ideally, it should be left running on AWS Beanstalk.

A system administrator can upload, update, or delete the RDS from the AWS Elasticsearch cluster by using the input.py script. This script works by querying the AWS Elasticsearch cluster directly as opposed to sending the request to AWS Beanstalk.

5 | Results

Results from bulk scans is the focus of this project. Bulk scans for 2000 files occur in a manner of seconds. With my previous implementation, it would approach several minutes.

Clearly, using AWS is a better method. This entire system functions very similarly to the VirusTotal API as was my intention. All components of this system are “elastic” and can grow to fit the demand of clients who wish to access the RDS. This is truly a scalable environment for a growing RDS and increasing demand for said RDS. It can easily be in-cooperated into other programs by importing the API as a Python 2.7.13 library.

6 | Summary

6.1 | Lessons Learned

A service such as the Message Digest Detective should not be run locally on commercial machines. The nature of data being searched is big and will continue to grow as time goes by. AWS services provided “elastic” mechanisms that can easily handle load balancing for search transactions. Furthermore, it is cumbersome to keep such large data on a local machine. It uses up space, is hard to move around (especially over networks), and hard to update and synchronize. Elasticsearch is perfect for the main goal of utilizing this dataset: searching. Not only can one search SHA-1 hashes in constant time, all other fields can be searched in constant time also.

6.2 | Future Work

The SQS service is not known to be 100% reliable. In other words, the way the SQS service is implemented, messages can be dropped. Therefore, there is a chance that a client will never obtain his/her bulk results when submitted via the API.

Recall the architecture in section 3. All queries are done to one Elasticsearch cluster. Even though a sort of load balancing is happening by using the SQS service to hold bulk transactions and not letting the Elastic Beanstalk handle the traffic returning the bulk result, all the querying still goes to the Elasticsearch cluster which can overload the Elasticsearch cluster. Surely enough the Elasticsearch instance count can be adjusted automatically, but another approach could be to have a dedicated cluster just for bulk transactions.

The endpoint point (currently a phone number) should be adjusted to support all types of endpoints like email, HTTP/HTTPS, etc. This will be valuable for products like Autopsy – The Sleuth Kit [2]. This forensic tool allows for cyber investigators and developers to create plugins. The NSRL RDS is a hash set commonly used with Autopsy. Autopsy enables cyber investigators to scan hard drives by viewing the entire file system and metadata about everything stored on the hard drive. The current way to check hashes is to use a VirusTotal plugin that checks files’ hashes or using another plugin that requires users to download the NSRL RDS to compare hashes. The latter is slow and cumbersome. Using the Message Digest Detective, all the plugin needs to do is scan the desired directory by computing file SHA-1 or MD5 hashes and then upload the list of hashes to AWS Beanstalk by using the API’s client script. The results of the scan will be reported to wherever the user chooses.

7 | Source

Beanstalk:	http://flask-env.hfv4nmbnse.us-east-1.elasticbeanstalk.com/
GitHub:	https://github.com/mramdass/Message_Digest_Detective_API

8 | References

[1] United States of America. National Institute of Standards and Technology. *National Software Reference Library*. N.p., n.d. Web. <https://www.nsrl.nist.gov/>.

[2] *Autopsy*. *Autopsy*. N.p., n.d. Web. <https://www.sleuthkit.org/autopsy/>.

[3] *Amazon Web Services (AWS) - Cloud Computing Services*. *Amazon Web Services, Inc*. N.p., n.d. Web. <https://aws.amazon.com/>.

[4] *VirusTotal*. *VirusTotal - Free Online Virus, Malware and URL Scanner*. N.p., n.d. Web. <https://www.virustotal.com/>.