Czech Technical University in Prague
Faculty of Electrical Engineering

Department of Computer Science

BACHELOR THESIS AGREEMENT

Student: Jordán Petr

Study programme: Open Informatics
Specialisation: Software Systems

Title of Bachelor Thesis: Implementation security intelligence cache

Guidelines:

1) Explore the interface of common security intelligence source such as ThreatGrid.
2) Find about general approaches to caching data locally for instant access. Collect general approaches to using such cache.
3) Design an architecture stack to provide a simple tool for accessing remote sources with local cache ability.
4) Implement the solution with respect to specific viable use cases.
5) Evaluate its performance and ease of use on sample data and real traffic datasets.

Bibliography/Sources:

[1] Memory systems, Cache, DRAM, DISK - http://www.e-reading.club/bookreader.php/138837/Jacob,_Ng,_Wang_-_Memory_systems._Cache,_DRAM,_Disk.pdf
[2] Java Caching - https://www.youtube.com/watch?v=ciA7rDr1vSg

Bachelor Thesis Supervisor: Ing. Martin Rehák, Ph.D.

Valid until the end of the summer semester of academic year 2017/2018

L.S.

prof. Dr. Michal Pěchouček, MSc.
Head of Department

prof. Ing. Pavel Ripka, CSc.
Dean

Prague, January 12, 2017

Czech Technical University in Prague
Faculty of Electrical Engineering
Department of Computer Science

Bachelor's thesis

Cache inteligence

Petr Jordán

Supervisor: Ing. Martin Rehák, Ph.D.

26th May 2017

Acknowledgements

I would like to thank Jára Boyko and Martin Pospíšil for support during writing this thesis. Also Petr Poliak for language support.

Author statement for undergraduate thesis

I declare that the presented work was developed independently and that I have listed all sources of information used within accordance with the methodical instructions for observing the ethical principles in the preparation of university theses.

In Prague on 26th May 2017 ……

Czech Technical University in Prague
Faculty of Electrical Engineering

Citation of this thesis

Jordán, Petr. *Cache inteligence*. Bachelor's thesis. Czech Technical University in Prague, Faculty of Electrical Engineering, 2017.

Abstrakt

Tato práce je zaměřena na analýzu, návrh a implementaci řešení cache serveru pro ukládání dat ze služby Virus Total s ohledem na možnosti dalšího rozšíření pro další analytické služby. Tato analýza řeší vhodný výběr technologií na základě jejich použitelnosti a výkonu. Implementace si klade za cíl použít vybrané technologie a splnit uživatelské požadavky, které byly v průběhu práce definovány.

Klíčová slova Internetová bezpečnost, Cache služby, VirusTotal

Abstract

This thesis is focused on the analysis, design and implementation of a cache server to store the results of the Virus Total service. The work emphasises on the extensibility of the solution for possible future additions of different analytical tools. The analysis considers the selection of technologies that will be used based on the maturity, usability and real-world performance. The implementation part solves the problem regarding the user defined requirements using the technologies with respect to the analysis outcome.

Keywords Security intelligence, Cache service, VirusTotal

Contents

List of Figures

Chapter 1

Introduction

Today's risks for malware[1] infection have become a daily threat. An average of malware attacks is 200,000 per day. And that's only the malware samples detected by one of the bigger companies. This is statistic from 2016 and the predictions for the following years say that the number of attacks will grow. Malware continues to grow and evolves to bypass antiviruses and other levels of protection[1].

Current trend in malware detection is machine learning. The number of the security companies that mentioned machine learning at 2011 is 5-10 out of 300-500. However since then the situation has changed dramatically. From more than 1500 security companies over 1000 of them is talking mostly about machine learning [2]. Nevertheless machine learning can not be successful without the labeled data.

Information about malware can be obtained from the analysis tools. Analysis tools such as Cisco's ThreatGrid and VirusTotal provide solid malware detection. Also it is important data source for Artificial intelligence development[5]. Data from Virus Total can improve accuracy of detection algorithms. However business limitations (requests limit) make work harder and slower.

This thesis is focused on storing information about files and URLs, which can be used in future analysis. Data are required by many groups so the solution tries to respect every group for future extension. Also this thesis tries to find and implement the best technological solution based on specific use-cases. It should provide service for storing and accessing Virus Total data.

1.1 Motivation and goal of the thesis

The motivation of this thesis is to create a tool for accessing remote sources with local cache ability. This tool will help to simplify and speed up the work with third party analysis services for research group in Cisco.

[1] urlhttps://en.wikipedia.org/wiki/Malware

Based on use-cases prioritization the goal of this thesis is to design and create REST-API service for storing results from Virus Total. The goal is to try out the most suited technologies and use them in final solution.

The implementation will use Spring Boot framework that provides REST-API and MongoDB for storing data. The main modules from Spring Boot are Web, Data MongoDB and testing tools. Database part requires good document design for optional performance. Finally the security requirements must be met.

Chapter 2

Analysis

This chapter introduces analysis tools, specific user requirements, caching systems analysis and technological choices. User requirements were not exactly specified so it describes requirements gathering and their prioritization process. Cache analysis is mainly about finding a solution with the best performance based on user requirements. Last part of analysis deals with programming language and framework choice.

2.1 Analysis tools

In this chapter there are described tools for facilitating detection of viruses, worms, trojans, and all kinds of malware. The first one is Virus Total. It mainly provides scanning results for antiviruses. The second one is Threat Grid. It is Cisco's system which provides analyses itself.

2.1.1 Virus Total

Virus Total stores results from antiviruses. It allows users scan file, URL, IP-addresses or domain name and getting the result. The results contains the last scanning result but we can always send new scan request. There are also important comments on sources from community users which are another sources of feedback. This results help user in decision if given source is malware or not.

2.1.2 Threat Grid

AMP Threat Grid combines static and dynamic malware analysis with threat intelligence into one unified solution. You get timely, in-depth information you need to protect your business from malware of all types. It integrates real-time behavioral analysis and up-to-the-minute threat intelligence feeds with existing security technologies, protecting you from both known and unknown attacks.[6]

2.2 Target customer

My thesis is designed for two groups of Cisco employees - researchers and data analysts. Researchers mainly provide robust solutions for malware detection based on artificial intelligence. One of the sources for their solutions are data from third party services such as VirusTotal, ThreatGrid, etc. So they would like service which holds the third parties data. Second group analyzes captured malware and evaluates its correctness or finding non cached malware. Analysts also research new forms of cyber-attacks. They mainly use third party services as information source for their classification decision. Based on discussion of both teams there is a conclusion. Analysis does not need the service itself but they need its implementation to Domain Buster Tool - internal Cisco tool for domain name and IP address analysis. Another reason why the caching is not a prioritize here is the number of requests. It consumes barely hundreds of request per day. However the solution service must respect requirement for future integration. On the other hand researchers work with large amount of data - hundreds of thousands. Thus the research group is the target customer.

2.3 Research group requirements

The whole CTA team[2] have only a single private API key, but multiple people in research are using it with overlapping queries. The volume of requests that we make every day is limited, sometimes this limit is exceeded and sometimes the limit is not fully used. A common cache would be beneficial to save API access quotas, network bandwidth and time to get the data from the VirusTotal cloud to server located in Prague.
Typical bulk request has around 200k Hashes, i.e. around 6.4MB
The typical size of a VirusTotal-response is 5kB, i.e. total size is around 1GB

2.3.1 Requirements

Request to fetch list of URL, domains or hashes (labeled as resources) from a laptop once per week. Get status from the laptop once per day. Consume the intelligence from a REST server or from a MapR-FS[3].

Acceptance criteria

1. Fetch request can be issued via REST (service)
   a) Implement endpoints for scanning of URL, Domain and file
   b) Limit the size of resources in request
2. Status of requests can be obtained via REST

[2] `http://www.cisco.com/c/en/us/products/security/cognitive-threat-analytics/index.html`

[3] `https://mapr.com/`

a) Implement endpoints for reading of URL, Domain and file
b) Add the parameter to URL, which specify if record is cached

3. Intelligence can be consumed via REST
4. Time-range request parameter
5. All intelligence is exported on file system
6. Immediate / Postponed request parameter
7. All cached resources are exported on file system
   a) Implement daily scheduler for exporting resources

2.3.2 Use cases

There are two main use cases - scan and report resources(domains, URL and files). Both of them require Basic Authentication. Scan (2.1) and report (2.2) are described in activity diagrams shown below.

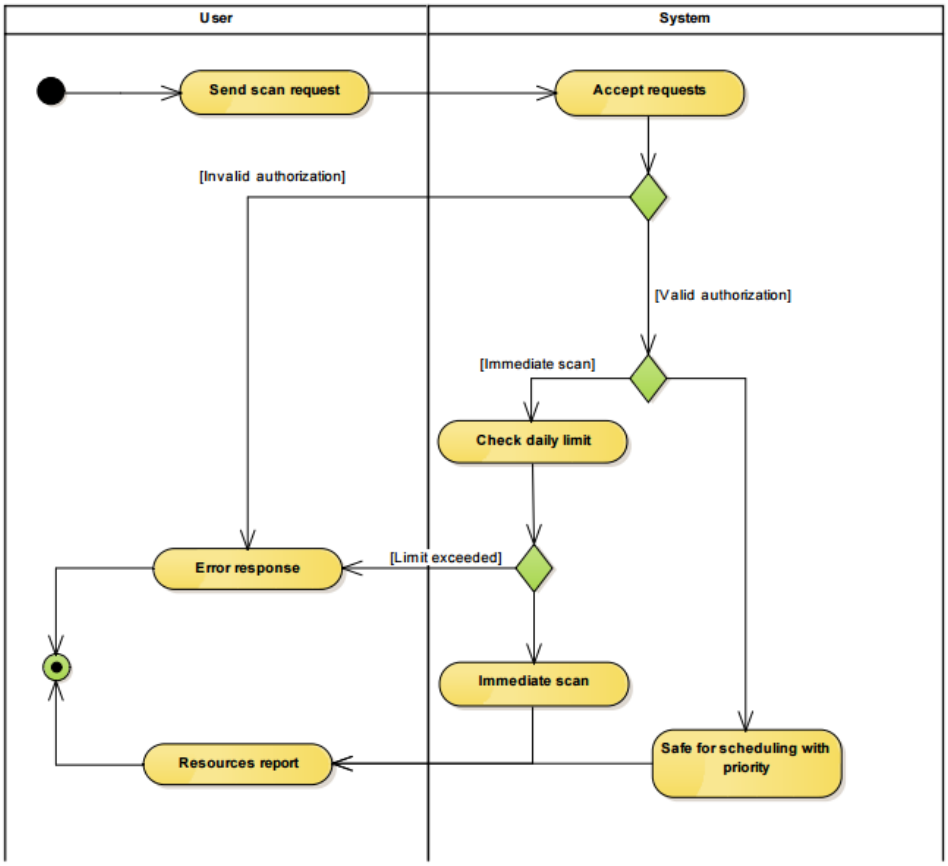

Figure 2.1: Activity diagram describing request for scanning resources

The first diagram describes user's request for a resources scan. Server authorizes request and saves non cached or expired resources for later scheduling.

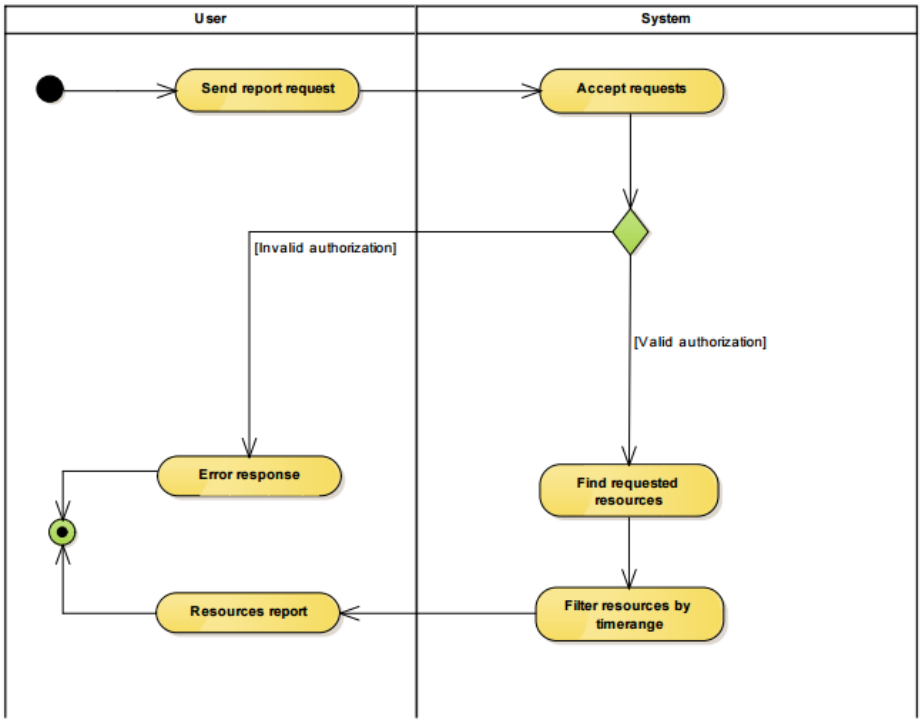

Figure 2.2: Activity diagram describing request for report cached resources

The second diagram describes user's request for getting resources from a server. Server provides filtering options - date range of scanned records and only cached resources. Beside this user can check status of requested requests by using filtration.

2.3.3 Domain model

The model is not complicated as you can see at (2.3). The service does not need to take care of the security itself as it will be protected by firewall and other Cisco's internal systems and/or workflows so users are not required. Everything what the application needs is just hold the scan request and their requests at *VirusTotalRequest*. Other properties mainly for behaviour of scheduling are also part of *VirusTotalRequest* - scheduling itself is explained4.2.2. The model satisfies all caching requirements.

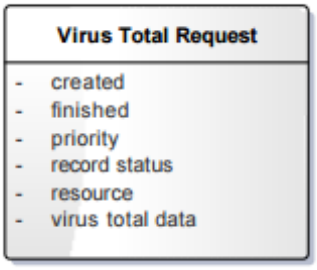

Figure 2.3: Activity diagram describing request for report cached resources

2.4 Cache performance

One of the main tasks is to choose cache service that offers the best performance. The choice of technology depends on two factors - read/write performance for our use-cases and its business limitations - storage size limit and failure resistance. There are three approaches considered in this thesis - SQL databases, NoSQL databases and in memory databases.

The first thing that has to be explained is that NoSQL does not supersede SQL or in memory databases. Every above mentioned technology can be used in special use-cases where it exceeds other technologies regarding performance. There does not exist a technology with the best performance for all use-cases. It always depends on specific use case[3].

SQL is used for managing relational databases and provides various operations on the data in the database. SQL database can provide very useful functionality like transactions, indexing or foreign keys. In general SQL databases are one of the most common choices for projects with many relations and transaction processing like enterprise applications or banking systems.

NoSQL databases are document-oriented. That means the data is stored in a document for instance article holds all related records in a single document such as author information, pictures, article content and tags. That property provides much better performance for CRUD operations. However the cost for better performance is absence of the transactions, relations and some other functionality provided by SQL databases.

In-memory database is a database which stores data in system memory. The volatile provides much faster data operations but the cost for it is data persistence. There are many SQL, NoSQL or key-value in-memory databases but there is still the same problem with persistence. However Redis provides simple synchronization tool to file system, but this synchronization has always delay. It better than nothing, but some records can be lost on shutdown.

2.4.1 Testing data

In order to achieve the most suited cache technology for our system we had to simulate its performance on sample data. All the storages performance are tested with PHP scripts. PHP is chosen based on its easy running on linux machine with PHP and easy integration with mentioned storages. Measured values are the time of writing PHP associative array with sample data and read this data with their conversion to associative array. The size of data is 50 000 records cause of use-cases. The results are depicted on graph(2.4).

2.4.2 MySQL

MySQL is a Well known open-source and one of the most used databases[4]. One of the main factor is its usage in LAMP stack. It is a combination of four technologies - Linux, Apache, MySQL and PHP. MySQL provides many types of storage engines which handle the SQL operations for different table types. In general InnoDB is the most used because of its support for foreign keys. Nevertheless for our use case the MyISAM is a better choice. It has better performance for read/write operations but it does not support some functions like transactions and foreign keys.

2.4.3 PostgreSQL

PostgreSQL is a another SQL database also open-source and more advanced than MySQL. There is wide support of data types, transactions and many new tools like JSONB - json stored in binary representation. It is designed for handling large number of tasks very efficiently. Also there is support for storing procedures which can reduce communication between application and database. Thanks to these procedures can be saved some computing time. That makes PostgreSQL really useful for large applications. On the other hand these functions are a trade-off for some performance which makes PostgresSQL slower than MySQL on average.

2.4.4 Memcached and Redis

Memcached and Redis achieve very close results to each other regarding performance. It surpasses SQL and NoSQL databases in speed but it is not well suited for our use-cases because of persistence. Nevertheless it is good for comparison.

2.4.5 MongoDB

MongoDB is the most popular NoSQL database. It is document-oriented database with BSON(binary json) documents in dynamic schemas. It is also open-source with good documentation. As other document databases MongoDB is always a good choice for large amount of data. In general

user have to well design the structure of NoSQL documents. Thanks to this mongoDB has better performance than PostgresSQL and MySQL. Detailed description is in 3.6.

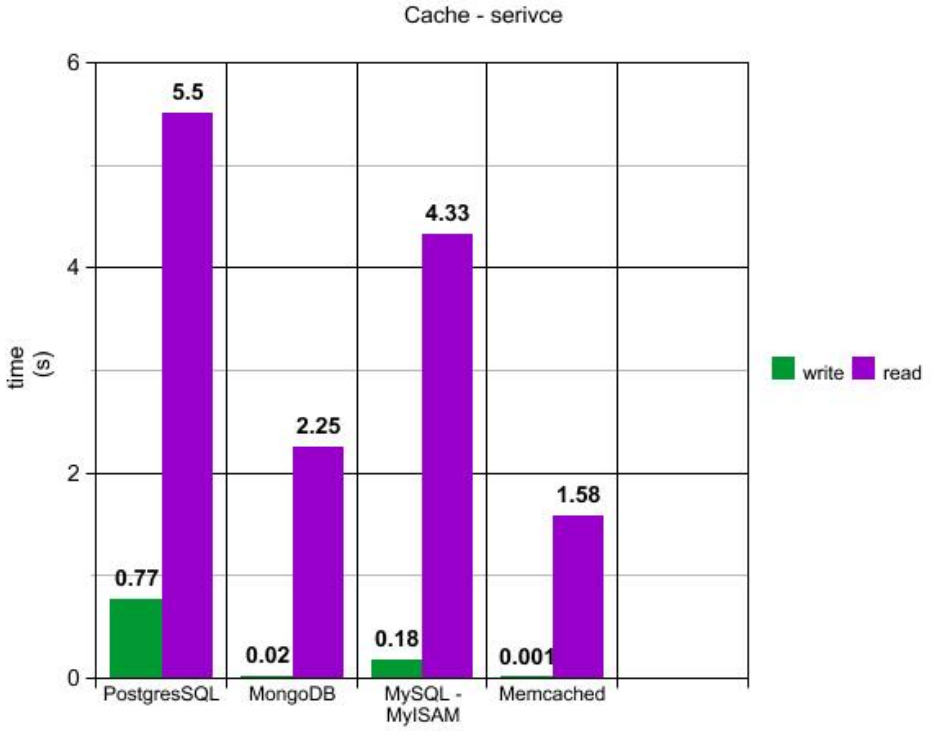

Figure 2.4: Cache performance

2.4.6 Result

Based on the results from the measurements the fastest database is the in-memory Memcached. That is not a big surprise but it can not be used 2.4.4. As expected NoSQL surpassed SQL. That means MongoDB is chosen for this thesis.

2.5 Programming language and frameworks

The language choice affects the whole project. Good general knowledge of the language is needed if you want to use some of its framework. It we want to speak about good framework it is always required to have really good documentation. In this chapter is explained selection of the most appropriate framework. There are three options of languages and their frameworks: PHP Nette framework, Scala Play framework and the last one Java with Spring Boot.

There are also other frameworks such as Django, Ruby on Rails, etc. However we have to respect Cisco's business limitations such as runtime environments and frameworks.

2.5.1 PHP and Nette

PHP is a well known scripting language used mainly for web development. The reason why I wanted Nette is because of my experience and thus my very good knowledge of this framework. It is open source and quite good documented MVC framework. It is easy to configure and run on Apache HTTPD. Also Nette solves many PHP security problems. Nette handle all use-cases well. It can provide REST service and there is easy integration for MongoDB.

2.5.2 Scala and Play

Scala is a modern object-oriented yet functional language which is very popular thanks to the possibility of using Java libraries. Play framework provides two options for language - Java and Scala. It provides really good documentation for both languages and many source code examples. There is good support for REST service and also support for MongoDB.

2.5.3 Java and Spring Boot

Java is reliable and mature object-oriented language. It provides huge number of libraries which make development easier. Spring Boot is derived from Spring framework. User can choose only needed modules from Spring like REST, JPA, Security, etc… It can make Spring Boot really suited for small application or enterprise application. Modules mentioned above provide support for REST and MongoDB.

2.5.4 Result

Spring Boot is finally selected framework. There is not big advantage over Nette and Play. All mentioned frameworks support required technologies and their performance is negligible for our use-cases. As main reasons for picking the Spring Book are great documentation with many examples and possibility of advice from CTA.

Chapter 3

Proposed architecture and design

This chapter explains in detail the architecture of the project. The parameters of the physical server and used web servers. The next part describes configuration through Puppet technology. It is followed by packaging section. Next section is dedicated to selected technologies - MongoDB and Spring Boot. The last section describes in details communication.

3.1 Application design

The whole application is built using the Spring Boot which runs on embedded Tomcat server. Spring Boot is MVC oriented framework. Model layer ensures communication with MongoDB. Controller layer provides REST-API access and view layer displays request data. The last part is internal scheduling for fetching queued data. Application receives requests through Apache HTTPD server only which solves security and resending request to Apache Tomcat. The architecture is visualized on the following figure.

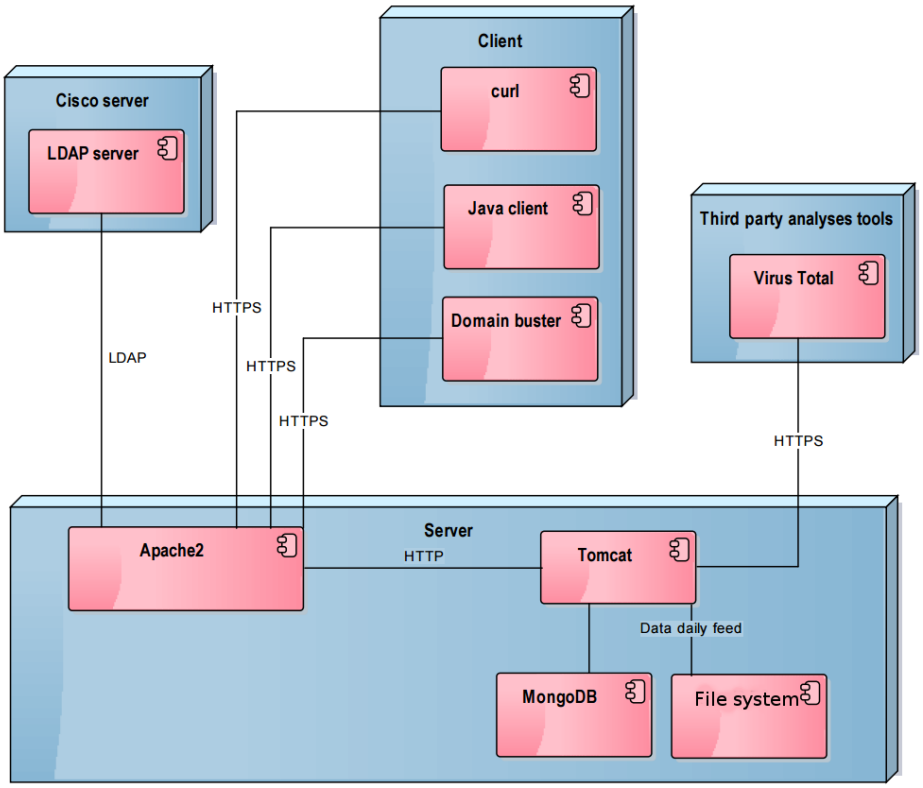

Figure 3.1: Cache performance

3.2 Virtual hardware

This chapter describes selected physical server and web servers in detail. The physical server(Intel machine) chapter describes its performance that matches user requirements. The web server chapter describes a very popular Apache HTTPD web server and built-in Apache Tomcat server for running applications.

Specific configuration and installed software for servers is fully managed by Puppets: Puppet configuration. At first creating GNU/Linux users and enable their connection via ssh keys. The next step is to create a specific folders, installing packages owned by specific users, ensure logging, setup services, setup firewall, cron and backup. There is only necessary configuration for ensuring security and connection. Other running application on Intel machine has own puppet configuration.

3.2.1 Physical server

The server is very powerful machine(3 virtual CPU (3 one-core Intel(R) Xeon(R) CPU E7- 4860 @ 2.27GHz), 32GB RAM, 50GB system + 12TB data (80% is

used)) with many cores, big RAM and many CPUs and it hosts many analytic tools. As mentioned in 2.3 requirements for MongoDB memory usages are big 1GB per bunch of requests and same size for exposing data to MapR-FS. Also the processing of bigger requests have to be quick enough. Finally the location of server is Prague. It reduces cache server response time because main data consumers are in Cisco Prague office.

3.2.2 Apache Tomcat

Spring Boot provides two options for embedded servers - Tomcat and Jetty. Where embedding means user does not have to deploy application to standalone server. There is not a significant performance difference between the two options. Request processing is nearly the same, configuration is easier in Spring Boot via configuration file and Java code. The final decision is to use Tomcat. In short, Tomcat is an open-source reference implementation of the Java Servlet, JavaServer Pages, Java Expression Language and Java WebSocket technologies. We use default configuration [4], which is ready to run. It is important to mention that a direct connection to this server is not allowed. Instead we use Apache HTTPD server as a SSL proxy and Basic Authorizaton server.

3.2.3 Apache HTTPD

Apache2[5] is an open-source web server. It runs on many different platforms from *nix family systems to Microsoft Windows. Our server is configured by Puppets - running as standard Apache HTTPD service. Listening only on port 443 to force secure connection over HTTPS. Apache server also requires authorization by LDAP where credentials are CEC - Cisco's username and password. LDAP and HTTPS are described in 3.7 in more detail.

3.2.4 CentOS 7

The used OS is CentOS 7[6] with systemd, which provides a system and service manage. It is important to mention the used OS, because the selected technologies such as Puppet and RPM require OS support. Also the implemented REST server will run as service in systemd.

[4] https://docs.spring.io/spring-boot/docs/current/reference/html/common-application-properties.html
[5] `https://httpd.apache.org/`
[6] `https://www.centos.org/`

3.3 Puppet configuration

Puppet[7] is a powerful tool designed to manage the configuration of Unix-like and Microsoft Windows systems in a declarative way. That makes puppet platform independent tool for confinguration. User can setup system resources such as package dependencies, user creation, file system modifications, create service etc. This configuration is described in the manifest folder and its *.pp* files. In the pp files a custom declarative language is used. Here is a simple Puppet language example 3.2. The main reason why to use Puppet is holding the whole server configuration in files which provide easy system upgrades and simplifies the releasing. Also there is the possibility of code reusing.

An alternative for puppets are for example bash scripts with configuration. But this way of configuration is usually very system specific and thus the reusability is low. Puppet provides a layer of abstraction which provides ability to configure resources on different platforms. User does not have to use specific commands or dependencies. A good example would be a package installation with *package* command. There is no required prior knowledge of the platform for package install like Ubuntu *apt-get* or CentosOS *yum*.

3.4 RPM

As mentioned before we used CentosOS, that means we could use RPM. RPM Package Manager is a package management system. This tool takes care of packaging the software from source into a distributable packages as well as standard package maintenance such as install, update and remove selected package. In RPM specification you can create GNU/linux users and required folders for application run. There is also the possibility to get additional information about the package like its description, version, repository, and other… Finally RPM solves package dependencies. Our application is packaged by RPM and uploaded to Prague Cisco repository and this repository is added to the server configuration.

3.5 Spring Boot

In the analysis part 2.5.3 there are mentioned basic facts about Spring Boot and the completion of application requirements. In this chapter there are described some topics in more detail. Spring Boot (SB) is presented as a framework with easy configuration that you can run *just like that*. You can select only needed modules and third-party libraries so there is minimal unnecessary dependencies.

By default SB creates stand-alone Spring application packaged with the embedded server. This means that you can run your packaged application(.jar

[7] `https://puppet.com/`

```
class vtcacheconfig {
  require infrarepoconfig
  require javaconfig
  require intelconfig

  File {
        ensure => directory,
        mode => '0755',
        owner => 'tpcache',
        group => 'tpcache'
  }

  package { 'third-party-cache-server': ensure => latest }
  ->
  file { [
        '/data/feeds/third-party-cache',
        '/var/log/third-party-cache' ]:
  }
  ~>
  service { 'third-party-cache-server': enable => true }

}
```

Figure 3.2: Third party cache server

or .war) through *java -jar* command. There are also many production-ready features such as metrics, health checks and externalized configuration 4.4. Finally SB left all of the Spring XML configuration and replaced it by Java configuration classes.

3.6 Database

As mentioned in 2.4.5 the chosen NoSQL 2.4 database is MongoDB. The company's MongoDB service is used by many applications so there are some specific security workflows. There is created specific user for every application. Our user has been given a readWrite role and virustTotal database access. It provides read and write operations to database. Authentication is always against admin database. Admin database is special MongodDB database, which stores all users and their roles.

3.7 Client-server communication

As it is mentioned above, communication between server and client is done only by a REST-API. There are two types of request GET and POST. GET request is used to read data, with specific URL format and parameters returning desired data. The POST request is used to create new data, because this method has body not like GET. Also the POST request requires specific URL and optional parameters. The other HTTP methods are not used and HTTP 405 error is returned. The documentation for request is generated from test.

Communication flow is indicated in 2.3.2. Detailed example of scanning new domain immediately is described below.

1. At the begging the user sends POST request from his client(curl, Java, etc.). Where the URL of request is of the form:

https://secret-url.com/domain/scan?immediate=true

As you can see in URL path there is keyword *domain* and *scan*, which specifies the action. Parameters after the question mark are optional and they provide extended functionality. Finally server requires HTTP Base Authentication where credentials are encoded in header.

```
[{"resource":"029.com"}]
```

Request body

2. The API accepts and processes the request. Fist possible result is that the request is valid and response is HTTP 204 code, which means the request for scanning the domain is created. There is an optional parameter *immediate* requesting that the scan is done immediately. The second possibility is that the request is deemed invalid and HTTP 404 is returned.


```
[
  {
    "id": "591d6da5356f1180ef4a1ce2",
    "createTimestamp": 1495100837176,
    "finishTimestamp": 1495100837638,
    "vtData": {
      "response_code": 0,
      "verbose_msg": "Domain not found"
    },
    "status": "MISSING_IN_VIRUS_TOTAL",
    "resource": "x29.com",
    "type": "DOMAIN",
    "topPriority": false
```

```
  }
]
```

OK response example

```
{
  "timestamp": 1495100924924,
  "status": 404,
  "error": "Not Found",
  "message": "Not Found",
  "path": "/domain/sca"
}
```

Invalid request response example

3. The last step is the client parsing the response.

3.7.1 Protocols

The application uses three protocols. HTTP for communication between Apache HTTPD (proxy) and Apache Tomcat Server. HTTPS is used by Apache HTTPD for communication with outside world. Last one is LDAP protocol used for Cisco authorization.

Hyper Text Transfer Protocol(HTTP[7]) is a protocol for exchanging hypertext documents. It is a stateless request-response protocol.

HTTPS is the secure version of HTTP.

LDAP (Lightweight Directory Access Protocol) is protocol for accessing and maintaining distributed directory information. In our case LDAP provides access to Cisco's users credentials.

Schema of used protocols is in 3.1;

Chapter 4

Implementation

In this chapter is described the implementation of the server side application. There described used libraries and important implementations. Also this chapter shows how is the code tested.

4.1 Third party libraries

All library dependencies are managed by Apache Maven. Maven is application manager for Java projects described by XML in a so called *pom.xml* file. There are many spring modules(REST, MongoDB, WEB), which provide REST controller layer and communication with MongoDB. The second part are modules for testing purpose - Mockmvc, WEB, Actuator. Detaily described in unit testing chapter.

The second groups of libraries starts with unirest - HTTP client for Virust Total API implementation. For easy testing this API here is a Wiremock. To reduce Java boilerplate code(code with little or no alteration) we are using Lombok[8]. Lombok replace boilerplate code by annotations. Another big library is Apache commons IO for simplifying file operations. Finally the last library is the Embedded MongoDB as the name suggests you do not need MongoDB on physical server for testing purpose.

4.2 Gathering data from Virus Total

The first problem to be solved is obtaining the data from VT. We try to find existing library that enables us to work with VT. Based on VT API limitations we have to solve how to daily download data and do not exceed limit.

[8] `https://projectlombok.org/`

4.2.1 VirusTotal client

VT provides three Java community implemented libraries to interact with the public API. But any of these three libraries does not fully satisfy our requirements. Library from Mauricio Correa[9] supports only file scans/reads, library from Kanishka Dilshan[10] does not support domain scan, it isn't well documented and library from VIGHNESWAR RAO BOJJA[11] could not be build.

We created own client, which is fully covered by tests. This client is able to report URL, file or domain and scan these sources. As mentioned before all these methods are tested against WireMock[12] - tool for mocking target URL to return certain data.

4.2.2 Scheduling implementation

In order to fulfill the requirements we needed to implement automatic downloading of the results from VirusTotal. Two options were considered - either the application itself takes care of scheduling the tasks for downloading the results with some internal cron-like functionality or the application would rely on some (external) scheduler that would invoke the fetch via REST endpoints (for example regular cron job that would ping the corresponding endpoints). We decided to go with the first option because of Spring's native support for such case using the *@Scheduled* annotation on corresponding services.

4.2.3 Downloading strategy

Another important task is downloading the data from VT while not exceeding limit as mentioned in 2.3. We needed to come up with a strategy/architecture that is resilient to system failure and respects the daily limit of the service.

First iteration was to use a simple counter that was refreshed at the end of each day. Before each request to VirusTotal a record about the action was created while also checking if the limit hasn't been already met. The counter was strictly respecting the limit but the downfall was that it was not resilient to failure. This was caused by the inability of MongoDB to use transactions. Two problems were imminent - concurrent request wouldn't be able to know about each other thus possibly overcoming the limit, secondly if the request wasn't successful the counter has already been incremented. So there were problems with both upper and lower bound of the counter.

In second iteration we chose to use more straightforward approach. Before each request we would query MongoDB to get the total request count that have already been completed. After completing a request we create a record.

[9] `https://www.xlabs.com.br/java-virustotalapi/`
[10] `http://kdkanishka.github.io/Virustotal-Public-API-V2.0-Client/`
[11] `https://vighnesh.me/virustotal/`
[12] `http://wiremock.org/`

As there is a small possibility that more concurent requests might be handled in the same moment we decided to lower the internal application limit (based on an estimated number of requests) to compensate for that. This solution also do not have to care about server restart, because counter is always counted.

4.3 MongoDB

As mentioned in analysis 2.5.3 Spring Boot provides integration for MongoDB. In this chapter we explain the creation of collections by MongoDB's entity. Also there is described querying to MongoDB by MongoRepository interface, which provides simple querying.

4.3.1 Spring Data MongoDB

Spring allows you to create entity classes which represent objects in the database. In comparison SQL entities such as Java hibernate entities, which could contain many attributes - relations to another tables, id, column rules, etc… Number of MongoDB entity(4.1) should contain all the related information (thus violating the normal forms), so the number of the options are low[13]. We mainly use only the *@Id*, which generates unique identifier.

The interface for creating simple queries is shown in 4.2. As you can see the querying is implemented by a keyword and a value. There are many options for keywords such as *GreaterThan, Not, Within*, etc. The full specification is avaliabe online [14]. Nevertheless this solution isn't very friendly to code refactoring, because when you change the name of a field the code stops working. Also the method name could get really long, which makes code hard to read.

Complicated queries can be implemented by Spring MongoDB query. This approach is used when you can not use simple query for instance the saving of new entity.

[13] `http://docs.spring.io/spring-data/data-document/docs/current/reference/html/#mapping-usage-annotations`

[14] http://docs.spring.io/spring-data/mongodb/docs/current/reference/html/mongodb.repositories.queries

```
public class Customer {

    @Id
    public String id;

    public String firstName;
    public String lastName;

    public Customer() {}

    public Customer(String firstName, String lastName) {
        this.firstName = firstName;
        this.lastName = lastName;
    }

}
```

Figure 4.1: The example of entity

```
public interface CustomerRepository extends MongoRepository<Customer, String> {

    public Customer findByFirstName(String firstName);
    public List<Customer> findByLastName(String lastName);

}
```

Figure 4.2: The example of query

4.4 Configuration

As mentioned above custom Spring Boot configuration is implemented by Java code and not by XML. The configuration solves creating new Beans and connection to MongoDB. Spring Boot Bean defines and creates objects, which are used for dependency injection (fields annotated as @Autowired).

Configuration files

- *application.yml* - Main configuration file, which holds a list of all properties and references to the underlying classes that consume them.
- *logback.xml* - Configuration file, which describes logging level, specific output folder and rollback policy.

It is important to mention that *application.yml* is not part of project, because it is a security risk. *Application.yml* contains secret information and is located in *config* folder, that means the secret data knows only the admin.

4.5 Tests

Why do we test code? The response is: we all make mistakes. This means we always should write tests that validate the functionality of our code. In this chapter we describe libraries used for testing and the process. By Cisco's development process this thesis requires minimally 80% of code coverage (Jacoco framework[15]) and no static analysis errors.

4.5.1 Static analysis

Static analysis analyzes the code without executing it. It could find code smell such as streams that aren't closed or violation of best practices (such as naming conventions etc.). Typical mistakes are lexical, syntactic or semantic. This helps to maintain better code quality. We used open-source FindBugs[16] library. By this analysis we reduced found bugs to zero.

4.5.2 Unit tests

Tests are implemented by JUnit[17] framework, which allows you to write repeatable tests. However in most tests we have to load/mock Spring Boot environment, because we want to *@Atowire* objects, se we use *@SpringBootTest* annotation. For tesing REST-API endpoints we used *@AutoConfigureMockMvc*. This allow you create *MockMvc object* that mock endpoint.

Tests should be runnable in every environment. Nevertheless we are using MongoDB, which is not part of every environment. This problem is solved by Embedded MongoDB[18] for tests only.

4.5.3 API documentation from tests

As mentioned in 4.5.2 the API is tested by MockMvc but this library allows you to generate static *.adoc* files which describe given request. These generated files can be transformed into easily readable documentation by the Asciidoctor library. This documentation is always generated when you run tests. The generated documentation is uploaded to maven site server by *mvn site:deploy* command. Maven site:deploy deploys the generated content to the site URL specified in the *<distributionManagement>*.

[15] `http://www.eclemma.org/jacoco/`
[16] `http://findbugs.sourceforge.net/`
[17] `http://junit.org/junit4/`
[18] `https://github.com/flapdoodle-oss/de.flapdoodle.embed.mongo`

Chapter 5

Pilot testing

This chapter talks about the testing of the finished service in the production environment. It compares the results with the requirements at the start of the work and briefly evaluates the performance. Further it explores the bottlenecks and potential enhancements of the finished service.

5.1 Application test

In this part we evaluate the output of the service and the user experience. Mainly we focus on the user interaction and usability. Unfortunately it is not yet possible to do the final evaluation as the pilot phase will be ongoing for 20 days (beginning on 23.May 2017) to show the actual impact.

5.1.1 MapR

We have already mentioned in chapter 4.2 that the downloaded data are stored on a persistent storage. You can find an example of such results in figure 5.1. The files were thoroughly tested regarding format specification and that the data stored corresponds to the actual results and none are missing. The check revealed that in the future it would be very beneficial to also include the source of the data (Virus Total, Thread Grid, etc.).

```
-rw-r--r--. 1 data mapr 205K May 12 23:55 cache_ 201705132.json
-rw-r--r--. 1 data mapr    0 May 13 23:55 cache_ 201705133.json
-rw-r--r--. 1 data mapr    0 May 14 23:55 cache_ 201705134.json
-rw-r--r--. 1 data mapr    0 May 15 23:55 cache_ 201705135.json
```

Figure 5.1: Daily data export example

5.1.2 REST-API

During the REST-API testing it was confirmed that no unintentional behavior occurs and few future user experience improvements were found. For example the time based filtration could contain date instead of timestamp. Or the option to bind a certain URL only with some specific resource to scan.

These comments were duly noted. However majority of the requests were transferred to future development including any specifications.

5.2 Performance summarization

The application processes around 10000 request with an estimated total computational run time of less than two hours. This estimate was the linearized time so after accounting of the parallelization we can reduce the estimate close to one hour including the communication with the database. Persisting the results on the file system was so low that it is negligible considering the total time.

Chapter 6

Conclusion

At first we needed to gather the user requirements and exactly specify, what is the desired result of this work. The analysis of requirements is described in 2.3. Based on these requirements we created an analysis of possible solutions and the most suited technologies in 2. In this chapter we explored the interface of common security intelligence, collected general approaches to caching data and evaluated their performance. That means we satisfy the analysis goal.

The next part is architecture and design of this application. As mentioned in goals, we should create a tool for accessing remote sources. However this tool has to be part of Cisco's infrastructure and this tool has its own architecture. The tool integration to the Cisco's infrastructure is described in 3. We proposed robust design with easy extensibility, management and reliable security.

In the end we implemented service which satisfies all the use-cases and respects proposed architecture. Based on Cisco's policy we respected Cisco's code management such as code testing and documentation. That means we created a good piece of software

The service is already running (from 26th May 2017). However estimated time for data downloading is around 20 days and downloading started on May. 22, 2017. This means that we can not present the results as of writing this thesis but the expected impact for the research group is high.

As mention in 2.2 next step will be the integration to Domain Buster tool. Also another future plan is to integrate other analysis tools such as ThreadGrid.

Bibliography

[1] Gammons, B. (n.d.). 6 Must-Know Cybersecurity Statistics for 2017 | Barkly Blog. Retrieved May 25, 2017, from <https://blog.barkly.com/cyber-security-statistics-2017>

[2] Cybersecurity and AI - expert panel discussion. (n.d.). Retrieved May 25, 2017, from <https://www.facebook.com/492692310885572/videos/742776755877125/>

[3] SQL vs. NoSQL Databases: What's the Difference?, <https://www.upwork.com/hiring/data/sql-vs-nosql-databases-whats-the-difference>

[4] MySQL customers, <https://www.mysql.com/customers/>

[5] Gaurav Sood (2017). virustotal: R Client for the virustotal API. R package version 0.2.1., <https://www.virustotal.com/>

[6] Threat Grid - Advanced Malware Protection. (2017, April 04). Retrieved May 12, 2017, from <http://www.cisco.com/c/en/us/products/security/threat-grid/index.html>

[7] HTTP - Hypertext Transfer Protocol. (n.d.). Retrieved May 18, 2017, from <https://www.w3.org/Protocols/

Nomenclature

API Aplication Programming Interface

CRUD Create, Read, Update, Delete

CTA Cisco Cognitive Threat Analytics

HTTP Hypertext Transfer Protocol

HTTPD Apache Hypertext Transfer Protocol Server

HTTPS Hypertext Transfer Protocol Secure

JSON JavaScript Object Notation

MVC Model-view-controller

REST Representational state transfer

SQL Structured Query Language

VT VirusTotal

Appendix A

Content of the included CD

Thesis code is intellectual property of Cisco, so it can to be published.

dtbPerformance	Databases performance scripts
text	Thesis text