UNIVERSIDADE DA BEIRA INTERIOR
Engenharia

Dynamic OSINT System Sourcing from Social Networks

Ivan Sousa Fernandes

Dissertação para obtenção do Grau de Mestre em

Engenharia Informática

(2º ciclo de estudos)

Orientador: Prof. Doutor Pedro Ricardo Morais Inácio
Co-orientador: Prof. Doutor João Paulo da Costa Cordeiro

Covilhã, Outubro de 2016

"...to my parents, for their love and strength."

Acknowledgements

During the year in which I have been involved in the development of this masters project, I had the fortunate opportunity to rely on the help and support of many people, who directly or indirectly contributed to its successful conclusion.

First of all, I am thankful to my parents for all the support and motivation they gave me throughout my life, and for providing me the opportunity to pursue an academic path.

I would also like to express my gratitude to my supervisor, Professor Pedro Ricardo Morais Inácio, and my co-supervisor, Professor João Paulo da Costa Cordeiro, first for accepting to be my supervisors, and secondly for the availability, attention and motivation that they gave me during the course of this project.

Last, but not least, I would like to thank all friends that contributed to the development of this project and made this experience richer. I am especially grateful to my colleagues from the laboratory — Bernardo Sequeiros, Acácio Correia, and Manuel Meruje — for their support and willingness to help me.

Thank you all.

Resumo

Atualmente, a World Wide Web (WWW) fornece aos utilizadores enormes quantidades de informação sob os mais diversos formatos: notícias, *blogs*, nas redes sociais, entre outros. O crescimento exponencial desta informação representa um grande desafio para a comunidade em geral, uma vez que a procura e correlação frequente de notícias acaba por se tornar numa tarefa repetitiva, potencialmente aborrecida e sujeita a erros. Apesar da maioria das pessoas ainda fazer o escrutínio da informação de forma manual e regularmente, têm surgido, nos últimos anos, sistemas Open-Source Intelligence (OSINT) que efetuam a vigilância, seleção e extração de informação textual, a partir de redes sociais e da web em geral. Estes sistemas são hoje ferramentas muito populares e úteis aos profissionais de diversas áreas, como a da cibersegurança, onde estar atualizado com as notícias e as tendências mais recentes pode levar a um impacto direto na reação a ameaças.

O objetivo deste trabalho passa pela tentativa de solucionar o problema motivado anteriormente, através da implementação de um sistema dinâmico OSINT. Para este sistema foram desenvolvidos dois algoritmos: um para adicionar, remover e classificar, dinamicamente, contas de utilizadores com *tweets* relevantes na área da segurança informática e outro para classificar as publicações desses utilizadores. A relevância de um utilizador depende não só da sua frequência de publicação mas também da sua importância (status) na rede social, bem como a relevância da informação publicada. Neste último ponto, são propostas funções de prospeção de texto que permitem medir a relevância de segmentos de texto.

A abordagem proposta é inovadora, envolvendo gestão dinâmica da relevância dos utilizadores e das suas publicações, garantindo assim um quadro de fonte de informação mais fidedigna e importante. Para além dos algoritmos e das funções que os compõem (também propostas no contexto deste trabalho), esta dissertação descreve várias experiências e testes usados na sua avaliação. Os resultados qualitativos constatados são pertinentes, denotando uma elevada utilidade prática. Em termos de interface homem-máquina, é disponibilizado um *mural* de informação contínua que vai sendo gerado dinâmica e automaticamente, a partir da rede social *Twitter*, e apresentado sob a forma de uma página web, destacando as notícias apresentadas pelo grau de relevância que possuem (vermelho para relevância elevada, amarelo para relevância moderada e verde para relevância reduzida).

As contribuições principais deste trabalho compreendem os dois algoritmos propostos e a sua avaliação. Um protótipo totalmente funcional de um sistema que os implementa, acompanhado pelo mural que mostra as notícias selecionadas, constituem outro resultado importante do trabalho.

Palavras-chave

Ciber-Segurança, Classificação de *Tweeters*, Classificação de *Posts*, Inteligência *Open-Source*, Mineração de Texto, OSINT, Twitter, *Web Crawler*

Resumo alargado

Este capítulo sintetiza, mas de forma alargada e em língua portuguesa, o trabalho descrito nesta dissertação. Começa-se pela descrição do problema abordado e quais os objetivos a alcançar. De seguida, são apresentadas as tecnologias envolvidas na elaboração do sistema pretendido e são referidos alguns trabalhos relacionados com este. Posteriormente são discutidas as componentes do sistema e os dois algoritmos responsáveis pelas funcionalidades para as quais o mesmo foi desenvolvido. Depois são apresentados os passos necessários para a implementação das componentes. A penúltima secção é dedicada à avaliação dos algoritmos implementados e à discussão dos resultados obtidos. O capítulo termina com a breve discussão das principais conclusões obtidas e com a apresentação de algumas direções para trabalho futuro.

Neste resumo alargado, os acrónimos estão definidos em Português nas suas formas longas. Contudo a forma curta dos acrónimos utilizados está de acordo com o respetivo acrónimo na língua Inglesa, de maneira a manter a consistência. Adicionalmente, para alguns acrónimos não faria sentido traduzi-los para Português, pelo que é mantida a sua designação inglesa e tratados como estrangeirismos.

Enquadramento, Descrição do Problema e Objetivos

Na atualidade, o fluxo de informação textual prolifera abundantemente na Web. A cada dia, uma enorme quantidade de informação é publicada sob os mais variados formatos: notícias, *blogs*, nas redes sociais, entre outros. O acumular da informação cresce realmente a ritmo exponencial constituindo um desafio nas áreas da segurança e inteligência, pois humanamente torna-se incomportável observar e muito menos "digerir" tanta informação. Neste quadro, têm emergido, no últimos anos, sistemas de vigilância, seleção e extração de informação textual, a partir das plataformas da Web.

Os sistemas *Open-Source Intelligence* (OSINT) são já ferramentas muito populares e úteis a profissionais de ciber-segurança, permitindo detetar mais precocemente novas ameaças e assim agir mais atempadamente. Estes sistemas destinam-se a encontrar, recolher e correlacionar conteúdo informativo disponível para o público em geral através das redes sociais, *blogs*, jornais, televisão, etc. Em comunidades como a da Segurança da Informação, ter acesso às notícias e tópicos mais recentes tem um impacto direto na preparação e reação a ameaças, como por exemplo os ataques de dia zero.

Redes sociais como o *Twitter* são, neste momento, fontes facilitadoras dos sistemas OSINT, através da partilha de recursos com a comunidade por parte de utilizadores que são especialistas "creditados" em determinado domínio, e que pesquisam e filtram informação com maior rigor que a comunidade em geral. Por outro lado, a limitação de 140 caracteres por mensagem (*post*), no *Twitter*, leva a um género de publicação já bastante sumariada, fazendo com que

a totalidade da informação de um *post* relevante seja realmente importante e sem rodeios. Contudo, o número de utilizadores aumenta de dia para dia e a comunidade é afetada por utilizadores que publicam sobre os mais diversos temas, sem que por vezes haja uma verificação das fontes de informação, nem uma análise e síntese da mesma. Deste modo, constata-se que o lidar manualmente com a dinamicidade da comunidade e com a quantidade e redundância da informação é ainda hoje uma tarefa desafiadora, repetitiva e potencialmente aborrecida, mas acima de tudo sujeita a erros de incompletude e inconsistência.

O objetivo deste trabalho passa pela tentativa de ultrapassagem dos problemas descritos anteriormente através do desenvolvimento de métodos para lidar com recursos de redes sociais, nomeadamente o *Twitter*, e métodos para classificar publicações recolhidas no âmbito da área de Segurança da Informação. Para tal, foi implementada uma estratégia baseada em dois algoritmos interdependentes: um para lidar com a adição e remoção de utilizadores em tempo real e outro para lidar com a classificação de publicações. Na prática, o primeiro algoritmo deve considerar e avaliar os utilizadores, com base nas suas publicações, de forma a saber qual o seu grau de relevância e se este deve ou não ser adicionado ao nosso sistema. Devem também ser identificados os utilizadores que ao longo do tempo deixam de ser relevantes, devendo portanto ser excluídos. O segundo algoritmo efetua a classificação de um dado *post*, a partir do próprio texto que o compõe, permitindo determinar o grau de relevância de cada *post* e permitindo assim também caracterizar o grau de relevância do utilizador, que não é mais que o agregado da relevância dos seus *posts*. Assim, foi implementado um sistema que integra os algoritmos mencionados numa plataforma web OSINT, alojada numa máquina virtual de um serviço *cloud*, com a finalidade de disponibilizar à comunidade as notícias mais recentes e relevantes na área da Segurança da Informação.

Principais Contribuições

Esta secção apresenta sinteticamente as principais contribuições resultantes do trabalho apresentado nesta dissertação:

- Uma das principais contribuições deste trabalho é o desenvolvimento de um algoritmo que permite a classificação de utilizadores da rede social *Twitter*, através de uma fórmula que contempla o seu estatuto na rede, a sua frequência de publicação e a relevância das notícias publicadas;
- Outra importante contribuição consiste no desenvolvimento de um algoritmo que permite a classificação de publicações dos utilizadores, através de uma fórmula que contempla a relevância do texto presente nas publicações e os *likes* e *retweets* das mesmas;
- A última grande contribuição deste trabalho consiste na implementação de um sistema que integra os algoritmos mencionados numa plataforma web OSINT, alojada numa máquina

virtual de um serviço *cloud*, com a finalidade de disponibilizar à comunidade as notícias mais recentes e relevantes na área da Segurança da Informação.

Tecnologias Utilizadas

O cumprimento dos objectivos propostos para esta dissertação implicam a utilização de tecnologias que permitem a prospecção de dados, a partir de fontes de informação como redes sociais e a *web* em geral. Neste âmbito foram apresentadas duas tecnologias distintas, mas com procedimentos semelhantes, nomeadamente *web crawlers* e APIs. Os *web crawlers* são programas que navegam pela *web*, recolhendo informação das páginas visitadas e seguindo as ligações presentes nas mesmas, que redirecionam o *crawler* para outras páginas. Os *crawlers* requerem a definição dos procedimentos necessários para a prospecção de informação, procedimentos esses que têm de respeitar um conjunto de políticas gerais referentes às páginas a serem visitadas, à frequência com que páginas já visitadas voltam a ser analisadas, para ver se houve alteração da informação aí presente, à atribuição de maior prioridade a páginas mais relevantes e à distribuição das tarefas de *crawling* por vários *crawlers*, de forma a potencializar a eficiência deste processo. As API representam um conjunto de normas e procedimentos que permitem o desenvolvimento de aplicações que comunicam com os serviços para os quais as APIs foram desenvolvidas. Em contraste com os *web crawlers*, as APIs fornecem previamente todos os procedimentos necessários para a prospecção de dados, bem como as políticas que têm de ser respeitadas na utilização das mesmas.

O capítulo evolui para a apresentação das características principais da rede social *Twitter*, utilizada como fonte de informação no desenvolvimento deste sistema, e da sua API. Para poder aceder aos conteúdos do *Twitter*, através da sua API, é inicialmente necessário possuir uma conta registada nesta rede social, para se obterem conjuntos de credenciais que permitem então aceder aos conteúdos pretendidos. A obtenção destas credenciais pode ser efectuada de diversas formas, de acordo com o caso de uso a que elas se destinam, como pode ser visualizado na Tabela 2.2.

De seguida são apresentados alguns trabalhos relacionados com o sistema desenvolvido, que implementam técnicas semelhantes, como utilização de conjuntos de palavras para filtrar as publicações recolhidas, a utilização de mecanismos de processamento de linguagem natural para calcular a relevância de textos e a utilização de procedimentos associados a APIs.

Na fase final deste capítulo foram ainda referidas e descritas brevemente as ferramentas utilizadas na implementação do sistema desenvolvido.

Arquitetura do Sistema e Algoritmos Implementados

Este capítulo começa por abordar as componentes do sistema implementado. Este sistema é composto por quatro componentes essenciais: uma aplicação desenvolvida em Java, uma base

de dados, uma plataforma web e uma máquina virtual. A aplicação desenvolvida em Java é responsável pela selecção e classificação de utilizadores do *Twitter* e pela classificação das publicações associadas aos mesmos. Esta aplicação é composta por cinco classes, de forma a organizar os métodos utilizados de acordo com o tipo de dados com que cada um interage. É também utilizada uma classe externa que permite calcular a relevância de segmentos de texto. A base de dados é responsável pelo armazenamento dos utilizadores recolhidos e das suas publicações. Esta é composta por três tabelas onde são armazenados os utilizadores recolhidos, que são colocados num estado de quarentena ou de produção de notícias, e onde são também armazenadas as notícias publicadas pelos utilizadores que se encontram no estado de produção de notícias. Estas tabelas possuem um número limite de entradas para prevenir que o tamanho da base de dados aumente indefinidamente e para melhor a qualidade dos utilizadores e das notícias armazenadas, com o passar do tempo. A plataforma *web* consiste na interface homem-máquina do sistema e apresenta aos utilizadores as notícias recolhidas pelo mesmo, ordenadas pela sua data de publicação e destacadas pela relevância que possuem (vermelho para relevância elevada, amarelo para relevância moderada e verde para relevância baixa), como se pode observar na Figura 3.1. A plataforma web possui ainda um campo de pesquisa, onde podem ser introduzidas palavras para filtrar as notícias apresentadas no mural. A máquina virtual actua como servidor do sistema desenvolvido e é responsável pela integração das componentes mencionadas anteriormente. A vantagem da sua utilização prende-se com o facto de esta possuir um IP público que permite que a plataforma *web* seja acedida por qualquer utilizador que possua uma ligação à Internet. A forma como todas as componentes interagem umas com as outras pode ser visualizada na Figura 3.2.

Subsequentemente, este capítulo evolui para a descrição dos algoritmos que representam as funcionalidades essenciais do sistema desenvolvido, nomeadamente o algoritmo de classificação de *tweeters* e o algoritmo de classificação de *tweets*. O algoritmo de classificação de *tweeters* efectua uma gestão dinâmica do estatuto dos *tweeters*, baseada no cálculo contínuo da sua importância para o sistema. Esta importância é calculada utilizando uma fórmula que contempla a relevância média das publicações de um utilizador, o número de publicações, relacionadas com a área da segurança da informação, nos últimos trinta dias e o número de seguidores desse utilizador, registados na base de dados e que possuem o estatuto de produção de notícias. O valor obtido para a importância de um utilizador determina então qual o estatuto que o utilizador terá no sistema. Os resultados desta importância encontram-se compreendidos numa escala de 0.0 a 1.0, de forma a facilitar a atribuição de um estatuto. O algoritmo de classificação de *tweets* atribui uma relevância às publicações recolhidas, utilizando uma fórmula que contempla a relevância do texto dessa publicação e o seu número de *likes* e *retweets*. Os resultados da relevância das publicações encontram-se também compreendidos numa escala de 0.0 a 1.0, de forma a facilitar a atribuição de um nível de relevância a cada publicação.

Implementação das Componentes do Sistema

Este capitulo aborda os passos necessários para a implementação de todas as componentes do sistema. Começando pela aplicação Java, esta necessita, numa primeira fase, da adição de algumas bibliotecas que permitem a aplicação comunicar com a API do *Twitter* e com a base de dados e fornecem os métodos necessários para a classificação de segmentos de texto. De seguida são geradas as credenciais que permitem que a aplicação Java aceda aos recursos da API do *Twitter*. Estas credenciais são compostas por quatro atributos, como se pode observar na Figura 4.1. Obtidas as credenciais, é possível criar objectos que irão servir de elo de comunicação com a APi do *Twitter*. De seguida é necessário definir um conjunto de palavras que serão utilizadas para filtrar as publicações selecionadas. As palavras utilizadas para este efeito podem ser visualizadas na Listagem 4.2. É também necessário dispor-se de um *corpus* especifico, do domínio, para que se possa atribuir uma relevância a determinada publicação. Este corpus é criado recorrendo a um *script* desenvolvido em Python que recolhe as notícias publicadas no site `https://threatpost.com`. Seguidamente é criada a base de dados associada ao sistema, de acordo com os atributos ilustrados na Figura 4.2. Finalmente, a aplicação Java pode então ser executada. A sua execução consiste fundamentalmente nos passos apresentados no algoritmo de classificação de *tweeters*, juntamente com a classificação de *tweets*. Na execução da aplicação são também utilizados dois métodos para a actualização regular da relevância das notícias e do corpus utilizado na classificação das mesmas.

Na implementação da plataforma *web* foram criados três ficheiros. O primeiro ficheiro é responsável pela recolha e apresentação das notícias da base de dados. Estas são organizadas pela data da sua publicação e destacadas pela relevância que possuem. O segundo ficheiro é responsável pela recolha e apresentação de notícias que contenham um conjunto de palavras definido pelo utilizador. O terceiro ficheiro contém as formatações e os estilos utilizados na apresentação do conteúdo da página *web*.

Para a implementação da máquina virtual foi escolhido o serviço *cloud* fornecido pela *Amazon* denominado Amazon Web Services. É necessário possuir previamente uma conta registada neste serviço para se proceder então à criação de uma máquina virtual. Esta criação envolve a definição de algumas configurações bastante triviais. Nestas, salienta-se as regras que foram criadas para a *firewall* da máquina virtual. Esses regras destinam-se a permitir o acesso dos utilizadores à plataforma *web* e o acesso do administrador do sistema à máquina virtual, como se pode observar na Figura 4.3. Após a criação da máquina virtual é ainda necessária a criação de uma chave privada que permite que as comunicações entre o administrador e a máquina virtual sejam feitas de forma segura. Em seguida é necessária a instalação de diversos pacotes de *software* que possam ser integrados e executadas as restantes componentes do sistema. Esta instalação só é possível através de uma ligação SSH à máquina virtual, que é feita recorrendo ao programa `putty.exe`. Finalmente resta apenas migrar os ficheiros correspondentes à plataforma

web para a directoria `/var/www/html/` e o ficheiro `.jar` para a directoria `/home/ubuntu/`. De seguida é possível executar o ficheiro `.jar` que inicia a execução do sistema.

Testes e Discussão dos Resultados

Neste capítulo foram realizados alguns testes para estimar os valores que condicionam a atribuição de uma classificação aos *tweeters* e aos *tweets*. O primeiro teste consistiu na selecção aleatória de 1000 *tweeters* e a sua importância, para se tentar procurar um padrão que permitisse estimar os valores para a sua classificação. Dos resultados obtidos neste teste, apresentados no Gráfico 5.1, estimaram-se então estes valores. De seguida procedeu-se a realização de outros testes para avaliar a eficácia dos resultados obtidos no primeiro teste. Selecionaram-se então aleatoriamente 10 *tweeters* com o estatuto "em produção" para serem analisados subjectivamente. Também se estabeleceram alguns parâmetros a que foram atribuídas classificações, pelos analisadores, para se poderem obter resultados quantitativos, para efeitos de demonstração da eficácia da classificação feita pelo sistema.

Para a classificação de *tweets* foram realizados inicialmente dois testes, que consistiram na selecção aleatória de 1000 *tweets* e a respectiva relevância, que ainda não tivessem sido actualizados, e na selecção aleatória de 1000 *tweets* e a respectiva relevância, que possuíssem pelo menos um *like* ou um *retweet*. Estes testes foram executados com a finalidade de se tentar procurar também um padrão que permitisse estimar os valores para a sua classificação. Os resultados obtidos nestes testes podem ser visualizados nos Gráficos 5.4 e 5.5, respectivamente. Dos resultados obtidos estimaram-se então os valores mencionados. De seguida procedeu-se à realização de outros testes para avaliar a eficácia dos resultados obtidos nos testes anteriores. Estes últimos consistiram na selecção aleatória de 50 *tweets* com relevância elevada e 50 *tweets* com relevância moderada, a fim de serem analisados subjectivamente para aferir uma relevância do ponto de vista humano. Foram também definidos alguns parâmetros aos quais foram atribuídas classificações, pelos analisadores, para efeitos de demonstração da eficácia da classificação feita pelo sistema.

No que diz respeito aos resultados, da avaliação quantitativa pode observar-se que a maioria dos *tweeters* e dos *tweets* avaliados possuíam valores, resultantes da análise humana, próximos dos valores atribuídos pelo sistema. Contudo a eficácia das classificações só ficou mais clara através da análise qualitativa desses *tweeters* e *tweets*. Nesta análise foi possível constatar que, de facto, a maioria dos *tweeters* e dos *tweets* avaliados possuem uma relevância significativa para o sistema, o que comprova a eficácia dos métodos de classificação implementados e dos valores que condicionam essas classificações.

Foram ainda realizados outros testes, menos relevantes, na tentativa de estimar o número de *tweeters* e *tweets* que são adicionados às tabelas da base de dados durante um mês.

Conclusões e Trabalho Futuro

Este trabalho foi desenvolvido com o objectivo de ultrapassar o desafio que representa a recolha e classificação de informação proveniente de fontes da *web*. O sistema OSINT desenvolvido focou-se na prospecção e classificação de dados provenientes da rede social *Twitter*. Para este sistema foram implementados métodos que permitissem efectuar uma recolha e gestão de *tweeters* e que permitissem recolher e classificar *tweets*. Para estes métodos foram escolhidos parâmetros que se consideraram pertinentes na obtenção de uma classificação rigorosa para as entidades mencionadas. Dos testes realizados e da análise dos resultados obtidos pode concluir-se que os métodos utilizados no cálculo da importância de um *tweeter* e no cálculo da relevância de um *tweet*, bem como as métricas definidas para a atribuição de uma classificação aos mesmos, denotam uma eficácia significativa, reforçando a utilidade do sistema desenvolvido.

Contudo, o sistema ainda está longe da perfeição, pelo que seria interessante, no futuro, melhorar alguns aspectos em relação ao seu funcionamento. Um dos aspectos que poderia ser melhorado seria a adição de mais parâmetros aos métodos de classificação implementados, como por exemplo a adição da importância das *hashtags*, utilizadas na identificação de tópicos, ao método de classificação de *tweeters*. Algumas *hashtags* são utilizadas frequentemente para a procura de tópicos relevantes, pelo que a sua adição ao método mencionado poderia representar uma mais valia para essa classificação. Também poderiam ter sido implementados métodos que prevenissem a recolha de *tweets* que possuem os URLs, referentes às notícias completas, corrompidos ou inexistentes. Os parâmetros seleccionados para a análise subjectiva e os testes realizados também poderiam ser mais diversificados. Na página web também poderiam ser feitas algumas alterações, nomeadamente o destacamento dos URLs de cada publicação e a inclusão de uma pesquisa mais diversificada para a selecção de notícias a serem apresentadas na página *web*. Finalmente, o sistema ficaria mais completo se tivessem sido implementados métodos que permitissem a recolha de notícias de diferentes fontes de informação, para além do *Twitter*.

Abstract

Nowadays, the World Wide Web (WWW) is simultaneously an accumulator and a provider of huge amounts of information, which is delivered to users through news, blogs, social networks, etc. The exponential growth of information is a major challenge for the community in general, since the frequent demand and correlation of news becomes a repetitive task, potentially tedious and prone to errors. Although information scrutiny is still performed manually and on a regular basis by most people, the emergence of Open-Source Intelligence (OSINT) systems in recent years for monitoring, selection and extraction of textual information from social networks and the Web promise to change the life of some of them. These systems are now very popular and useful tools for professionals from different areas, such as the cyber-security community, where being updated with the latest news and trends can lead to a direct impact on threat response.

This work aims to address the previously motivated problem through the implementation of a dynamic OSINT system. For this system, two algorithms were developed: one to dynamically add, remove and rate user accounts with relevant tweets in the computer security area; and another one to classify the publications of those users. The relevance of a user depends not only on how frequently he publishes, but also on his importance (status) in the social network, as well as on the relevance of the information published by him. Text mining functions are proposed herein to achieve the objective of measuring the relevance of text segments.

The proposed approach is innovative, involving dynamic management of the relevance of users and their publications, thus ensuring a more reliable and important source of information framework. Apart from the algorithms and functions on which they were build (which were also proposed in the scope of this work), this dissertation describes several experiments and tests used in their evaluation. The qualitative results are very interesting and demonstrate the practical usefulness of the approach. In terms of human-machine interface, a *mural* of information, generated dynamically and automatically from the social network Twitter, is provided to the end-user. In the current version of the system, the *mural* is presented in the form of a web page, highlighting the news by its relevancy (red for high relevance, yellow for moderate relevance, and green for low relevance).

The main contributions of this work are the two proposed algorithms and their evaluation. A fully working prototype of a system with their implementation, along with a mural for showing selected news, is another important output of this work.

Keywords

Cybersecurity, Open-Source Intelligence (OSINT), Post Classification, Text Mining, Twitter, Tweeters Classification, Web Crawler

Contents

List of Figures

List of Tables

Listings

Acronyms

ACM Association for Computing Machinery

AMI Amazon Machine Image

API Application Programming Interface

AWS Amazon Web Services

CCS Computing Classification System

CPU Central Processing Unit

CSS Cascade Style Sheets

DBMS Database Management System

DNS Domain Name System

DoD Department of Defense

DSA Digital Signature Algorithm

EC2 Elastic Cloud Computing

FTP File Transfer Protocol

HTML HyperText Markup Language

HTTP HyperText Transfer Protocol

HVM Hardware Virtual Machine

JDK Java Development Kit

JSON JavaScript Object Notation

IDE Integrated Development Environment

IoT Internet of Things

IP Internet Protocol

IT Information Technology

LTS Long Term Support

NLP Natural Language Processing

OSINT Open-Source Intelligence

PHP Hypertext PreProcessor

PIN Personal Identification Number

REST Representational State Transfer

RSA Rivest Shamir and Adleman

SCP Secure Copy

SFTP Secure File Transfer Protocol

SSH Secure Shell

UBI *Universidade da Beira Interior*

URL Uniform Resource Locator

VM Virtual Machine

WebDAV Web Distributed Authoring and Versioning

WinSCP Windows Secure Copy

WWW World Wide Web

XAMPP Cross-platform (package with) Apache, MySQL, PHP, Perl and Python

Chapter 1

Introduction

1.1 Motivation and Scope

Every day, large amounts of information is published in the World Wide Web (WWW) in different formats such as news, blogs, social networks, etc. It is therefore relatively easy for a user to get lost in this sea of data and sometimes difficult to find precise and useful information on a given subject. This problem is aggravated by the dynamic nature of the web and the natural presence of unstructured data and lack of semantic. Nonetheless, it is obvious that the web comprises a source that cannot be neglected for many reasons, with ubiquity and responsiveness among them.

Many communities, teams and institutions benefit greatly of receiving accurate information in a timely manner, as for example cyber-security teams or secret intelligence agencies. When this information is coming from publicly available sources, such as posts from blogs, webpages or social networks, the process of collecting, exploiting and disseminating that information in a timely manner for a specific audience with the intention of addressing a given intelligence requirement is called Open-Source Intelligence (OSINT). This definition is given by the United Stated Department of Defense (DoD) in [OSI06]. In the case of the cyber-security community, getting timely access to the latest news and topics may comprise a decisive difference in the response to threats such as zero-day attacks [Ste07]. It also allows users and administrators to be aware of the existent risks and issues in the most diverse technologies, and thus to increase the overall security status of their applications and software.

Since the web is one of the most fruitful sources of information nowadays, and given the increasing demand for OSINT, it is easy to understand the interest in developing systems that perform the aforementioned tasks in an automatic or semi-automatic manner. Many of them [Kar12c, Kar12b, Kar12a], are backed up by a supporting community that curate news, or focused in well defines sources, such as network, operating system and service scanning. Nonetheless, there are no general public OSINT frameworks based on Twitter, which partly motivated this work. The fact that Twitter already forces authors to summarize their ideas into 140 characters may result in added value when trying to deliver concise information to end-users. Nonetheless, it should be a challenge to make sense out of these small pieces of information for the sake of selecting what is good from what is not.

This master's project was focused on developing a fully automated OSINT system, which integrates means to deal with data sourcing form social networks, namely Twitter. These means involve the collection of data associated with tweeters and their posts, and the classification of the collected data, with emphasis on text analysis. Its scope falls in the intersection of the areas of information security, social networks and Natural Language Processing (NLP). Since it will deal with a social network, Application Programming Interfaces (APIs) and web crawlers will play important roles in this work. Under the 2012 version of the Association for Computing Machinery (ACM) Computing Classification System (CCS), a *de facto* standard for computer science, the scope of the master's project, reflected in this dissertation, is defined by the categories named:

- **Human-centered computing Visualization**;
- *Information systems Information retrieval*;
- *Security and privacy Security services*;
- Security and privacy Social aspects of security and privacy.

1.2 Problem Statement and Objectives

Social networks like Twitter ease the process of sharing resources with the community by users who are *credited* experts in a given field, which search and filter information more accurately than the general community. On the other hand and on Twitter, the 140 characters per message (post) limitation leads to a genre of publication that is already well summarized, making sure that the entire information of a relevant post is indeed relevant and flatly. However, the number of users increases daily and the community is affected by users who publish on various topics without checking the sources of information, or those ones who do not analyze or synthesize it. Thus, it appears that manually handling this dynamic community and the amount of information it generates is nowadays a challenging, repetitive and potentially boring task, but, above all, subject to incompleteness errors and inconsistency.

Given the problem mentioned above, the objectives of this research work can be further detailed as follows:

1. Research, develop and evaluate an algorithm that automatically adds and analyzes and removes sources of information related with information security on social networks according to their activity and significance of their contents;
2. Research, develop and evaluate an algorithm that automatically categorizes, merges and manages Information Security news, sourcing from the Twitter social network;

3. Integrate both algorithms in a web platform that automatically tracks sources of information, categorizes, merges and manages news, and shows them in a visually concise and meaningful manner.

1.3 Adopted Approach for Solving the Problem

To achieve the previously identified objective, a strategy based on two interdependent algorithms was adopted: the first algorithm was focused on handling the addition and removal of users in real time; and the second algorithm was focused on handling ratings for publications. In practice, the first algorithm should select and evaluate users, based on their publications, in order to know the relevance degree and whether it should or should not be added to our system. Also, over the time, users who are no longer relevant should be removed. The second algorithm performs the classification of a post, based on the text itself, allowing the determination of the relevance degree for each post and the characterization of the user relevance degree, which might be more than the aggregate of the relevance of their posts. Thus, it was implemented a system integrating the algorithms mentioned above in a fully automated OSINT web platform, hosted on a virtual machine of a cloud service, in order to provide to the community the most recent and relevant news in the information security area.

The steps to the adopted approach can be further detailed as follows:

1. Contextualize with the problem at hands and with the objectives of this project, as well as with the technologies involved. Revision of the specialized literature and related works;
2. Design and development of an algorithm that automatically adds, analyses and removes sources of information on social networks;
3. Design and development of an algorithm that automatically classifies news coming from the several sources of information, but focused on Twitter;
4. Prototyping of the web platform integrating the aforementioned algorithms;
5. Definition of the metrics and method for evaluation of the algorithms;
6. Testing, fine-tuning and evaluation of the developed algorithms.

1.4 Main Contributions

To the best of the knowledge of the author, the proposed solution to the aforementioned problem is innovative. No other OSINT systems with the characteristics described herein were found in the literature or online. The proposed solution involves the dynamic management of Twitter users through their the assessment of their importance and relevance of their posts on the social

network. There are OSINT systems that deal with information from social networks [Nou13] but not in this way. The main contributions of this work can be described as follows:

1. The design, implementation and evaluation of an algorithm to classify users from Twitter, based on what they publish and on their importance on the surrounding community. The classification can be understood as a measure of the importance of these users to the system (and thus to the cyber-security community);

2. The design, implementation and evaluation of an algorithm to classify Twitter posts, based on their relevance to the topic of cyber-security and feedback from the community. The design of both algorithms included the proposal of formulas to estimate user importance and Twitter relevance, also discussed herein. These formulas combine several variables such as the number of likes, retweets and followers, as well as the integration of NLP techniques to assess the relevance of the text

3. The delivery of a fully functional prototype of a system integrating both algorithms for a proof-of-concept. The system is available online and ready to use. The evaluation of the algorithms was performed after having the system working. It provides a *mural*, publicly reachable, where the most relevant news are shown organized by colors.

1.5 Structure of the Dissertation

The body of the dissertation is composed of 6 chapters and an annex. The contents of each one of them can be summarized as follows:

- Chapter 1 — **Introduction** — presents the scope and the motivation behind the work described in this dissertation, as well as the addressed problem and objectives. It also includes a subsection describing the approach taken to fulfill the objectives and its main contributions for the advance of knowledge. The dissertation structure is presented at the end of this chapter.

- Chapter 2 — **State of the Art and Technologies** — presents and describes the necessary concepts for the preparation of the desired system. It also presents various approaches and methodologies used to solve similar problems to the one addressed in this dissertation.

- Chapter 3 — **System Architecture and Components** — describes the components used in the preparation of the system as well as the two algorithms implemented for the two main features of this system: the Twitter Users Management and the Users Post Classification. It also presents the tools used in the development of the system along with a brief description of each one of them.

- Chapter 4 — **System Implementation** — discusses the requirements and configurations necessary for the implementation of the system components and so that they can interact with each other.
- Chapter 5 — **Application Evaluation** — presents the procedure used for the evaluation of the two implemented algorithms and the reasons for choosing the value of key variables in the application execution. The results are discussed and some conclusions are drawn from them.
- Chapter 6 — **Conclusions and Future Work** — presents the main conclusions of this research work, with focus on the results and the application utility. It also presents some considerations about the future work.
- Annex 1 — **Human Classification of Posts** — contains the tables that summarize the experiments concerning the subjective analysis to the algorithm for classification of posts. These tables are mentioned in chapter 5.

Chapter 2

State of the Art and Technologies

2.1 Introduction

This chapter discusses the theoretical concepts required to fully understand the problem addressed in the scope of this project. Several approaches and technologies related with the work at hands will be discussed in this chapter, as well as the techniques and tools used to implement the proposed solution. This chapter is divided into six main sections. Section 2.2 presents the concept of a web crawler and provides details regarding its way of functioning. Section 2.3 presents the concept of an API and its utility in the development of software applications that communicate with back-end services. Section 2.4 briefly describes the Twitter social network and present its API structure, which played an important role in the system developed within the scope of this work. Section 2.5 presents several approaches to solve similar problems to the one addressed in this project, with emphasis on the similarities and differences between those approaches and the one followed herein. Finally, section 2.6 presents the tools used in the implementation of the developed system.

2.2 Web Crawlers

This section provides a brief introduction to web crawlers, whose concepts were instrumental in some parts of the development of the system proposed in this project.

2.2.1 Overview

The WWW is nowadays providing a huge volume of information to users in the form of hypertext. However, due to the dynamic nature of this information and the presence of unstructured semantics, it may be difficult for a user to find relevant and valid information matching a desired topic. Web crawlers comprise a partial solution to that problem, in the sense that it crawls the web with the intention of categorizing it in some meaningful manner, which can later be used to perform searches and find information. In its most basic implementation, a web crawler is a program that browses the web automatically, downloading content from the visited pages and following the links therein to other pages.

Initially, a list of Uniform Resource Locators (URLs), called seeds, must be provided to the crawler so that it can begin the search. As the crawler visits the provided URLs, it will store information in such a way that the corresponding page can be seen as if one was accessing it on

the web, though it is preserved as a *snapshot*, which represents the state of the page at a given point in time. After storing the information of the current page, the crawler will then identify all the links of the page and add them to a list of URLs to visit in the future, called the crawl frontier. Then it will apply the aforementioned procedures to the new set of URLs in accordance with a set of policies that will be mentioned later. In addition to these features, that show the basic operation of a crawler, others can be included to enable the crawler to perform more complex tasks such as page validation, structural analysis and visualization, update notification, web mirroring and personal assistants/agents [SD11]. The architecture of a standard web crawler is illustrated in Figure 2.1, adapted from [SD11].

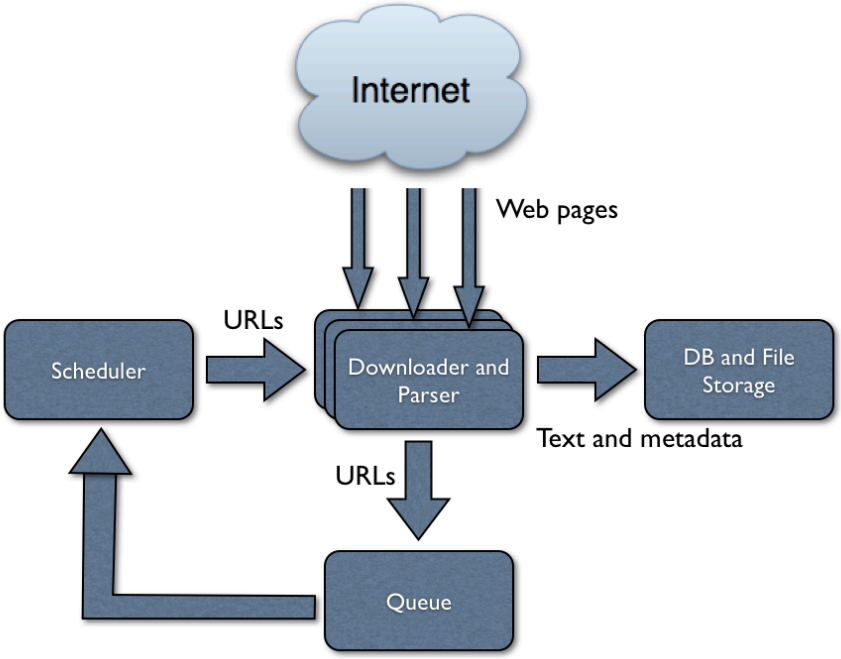

Figure 2.1: Architecture of a standard Web crawler.

Despite the basic procedures of a crawler are relatively easy to understand and implement, there are relevant web features that hinder the crawling process like the huge volume of data on the Web, the speed at which this information changes and the dynamic page generation. The huge volume of information implies the existence of a method to prioritize the most relevant pages, since in a given time frame, the crawler will be just able to download a fraction of all the Web pages it intends to visit. On the other hand, the rate at which information changes and the dynamic nature of the web pages means that it is quite likely that, by the time the crawler completes the download of a website, new pages have been added, changed or removed, causing inconsistency in the information. That said, it is noticeable that the definition of policies by which the crawler is governed will have considerable influence on its operation and performance.

2.2.2 Behavior of a Web Crawler

The behavior of a Web crawler results from a combination of several policies [Car04] such as:

- **Selection policy**, which states which web pages should be visited;
- **Re-visit policy**, which determines when the crawler should check for changes on pages;
- **Politeness policy**, which defines how to avoid overloading the servers responsible for the pages;
- **Parallelization policy**, that defines how to coordinate the operation of distributed Web crawlers.

As stated earlier, a crawler will only be able to download a fraction of the possible web pages with a given time frame. Therefore, it must be able to distinguish the visited pages, according to their importance in relation to the searched topic. The importance associated with each page can be determined through a function admitting the intrinsic quality of that page, and its popularity in terms of links or visits and/or its URL as parameters. There are several approaches on how to select the pages to be visited. The following list only three of them:

- **Topic-focused Web Crawling** — is the most common approach and is motivated by the fact that most people are only interested in a small fraction of the Web content. This method focuses on crawling a small web portion in order to discover a set of related pages with a desired topic. This procedure proves to be essential in the performance of the crawler, due to the finite resources inherent to crawling, such as storage capacity, bandwidth and execution time;
- **URL Normalization** — is a useful approach that allows the crawler to avoid crawling the same page more than once, since in the page hierarchy of links it is likely to find links that refer to pages previously visited. This process is known as URL Normalization or URL Canonicalization and consists of modifying and standardizing a consistent URL. This standardization can be achieved through various procedures, such as the conversion of URLs to lowercase, the removal of segments `.` and `..`, and adding trailing slashes to the non-empty path component;
- **Path-ascending Crawling** — is an approach used by crawlers that intend to download all the possible features of a particular web page. In this method, the crawler follows each ascending path, starting from the initial target URL. For example, given the URL `http://www.di.ubi.pt/Cursos/mei/`, the crawler will attempt to download the contents from `http://www.di.ubi.pt/`, `http://www.di.ubi.pt/Cursos/` and `http://www.di.ubi.pt/Cursos/mei/`, instead of focusing only in pages located in `http://www.di.ubi.com/`

`Courses/mei/`. Researcher Viv Cothey found that path-ascending crawlers are quite effective in finding isolated resources or resources with no previously indicated or found links [Cot04].

Due to the web dynamic nature and the fact that crawling only one of its fractions can last for weeks or months, it is quite likely that several events such as the creation, update and removal of resources has occurred during the execution of the crawling cycle. From a search engine point of view, there is a cost associated with not detecting these events, meaning that the crawler has an outdated copy of some features. This value can be determined by different functions, and two of the most frequently used are *freshness* and *age* [CJ00], shown in equations 2.1 and 2.2:

- **Freshness** — a binary measure that indicates whether the local copy is accurate or not. The freshness of a page p in the repository at time t is defined as:

$$F_p(t) = \begin{cases} 1 & \text{if } p \text{ is equal to the local copy at time } t \\ 0 & \text{otherwise.} \end{cases} \tag{2.1}$$

- **Age** — a measure that indicates how outdated the local copy is. The age of a page p in the repository, at time t is defined as:

$$A_p(t) = \begin{cases} 0 & \text{if } p \text{ is not modified at time } t \\ t \text{ - modification time of } p & \text{otherwise} \end{cases} \tag{2.2}$$

The objective of the crawler is to keep the average freshness of the pages in its collection as high as possible and the average page age as low as possible, by monitoring the number of outdated pages. The evolution of the values of the freshness and age functions for a page along the time is illustrated in Figure 2.2, adapted from [CJ00]. When the age of a page reaches a given threshold, the crawler synchronizes its contents, but the page may have been changed in the meanwhile.

Despite the fact that crawlers can extract information faster and on a larger scale compared to humans, their use can be a heavy burden to the performance of a website. A good example of this occurs when a crawler is running multiple requests per second or downloading large files. The server may have a hard time dealing with the aforementioned or similar events from the crawlers. Web crawlers are thus very useful to perform various tasks related with cataloging of information on the web, but they generally incur in costly activities, specially to web servers, as for example:

- network resources, since the execution of web crawlers requires considerable bandwidth and they perform various procedures in parallel for a long period of time;

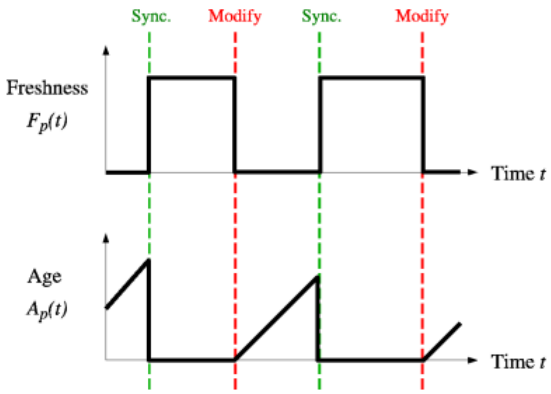

Figure 2.2: Evolution of freshness and age with time. The two events presented in the figure correspond to a change of a web page in a server(event *modify*) and the downloading of that page modified by a crawler(event *sync*).

- server overload, particularly if the number of accesses is very high;
- poor crawler implementation, which can compromise the operation of servers or routers, among other resources they usually can candle;
- use of personal crawlers operated by too many users can also compromise the operation of the network and the servers.

A partial solution to these problems is the use of the robots exclusion protocol, also known as the `robots.txt` protocol, which is a standard that allows administrators of web servers to define the restricted content for crawlers, considerably reducing their execution time. Outside parameters for defining a time interval between crawlers requests to the same server, which is the most effective way to prevent server overload, can also be added to this standard.

The parallelization policy is a strategy for having crawlers running multiple tasks in parallel, in order to optimize the download speed and the computational cost of tasks parallelization. Within this context, Shruti Sharma [SS11] developed a basic crawler and a parallel crawler, and compared their efficiency in the collection and analysis of the web pages in the URLs `http://www.modusporta.com` and `http://www.e-paranoids.com`. Table 2.1 shows the results, highlighting the number of pages visited and the time taken to download and analyze them, for each crawler. The parallel crawlers must have a method to record every new found URL during

URL	Number of visited pages	Basic Crawler time (s)	Parallel Crawler time (s)
http://www.modusporta.com	7	25	21
http://www.e-paranoids.com	74	482	387

Table 2.1: Comparison of the execution time for a basic crawler and parallel crawler, according to [SS11].

the crawling process, preventing the same URL to be visited by two separate processes.

2.3 APIs

In computer programming, an API is a set of routines and standards that allow the development of software applications that communicate with back-end services [CR14]. APIs provide methods for programmers to access or modify the content provided by the intended software and enables the integration of applications , partners, and customers on those services in an easily consumable way. Figure 2.3 illustrates the entities that benefit from the usage of APIs (figure adapted from [Vyb16]).

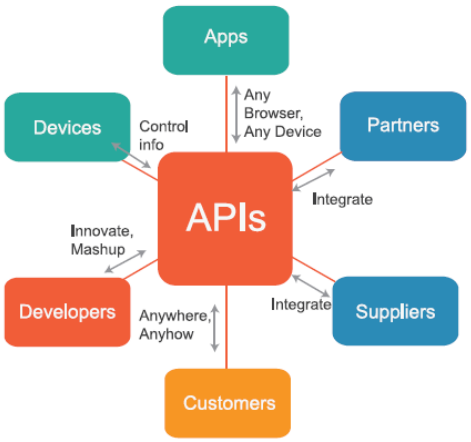

Figure 2.3: Example of an API and its possible associated entities.

The APIs are used in various systems such as databases, web sites and operating systems, and facilitate the development of applications which communicate with these systems by providing libraries for different programming languages. Usually, an API provide methods, inputs, outputs, and underlying types, defining implementation independent functionalities [Vyb16]. By using them, a large amount of software and applications can be developed without compromising the service interface.

The development of APIs provide several benefits for businesses enterprises and the community in general [CR14]:

- **Reduction of costs** — developing APIss reduces software production costs through the reuse of services instead of developing new ones;
- **Increasing business agility** — the use of APIs allows developers to have an higher degree of productivity since they allow the integration of different types of technology;
- **Increasing innovation and new business models** — as APIs allow users to access and handle the resources of a service, it is possible to develop new applications to different contexts, using the available API resources and resources from third-party entities.

The system developed along this project makes extensive use of the Twitter API. The main reasons for including a discussion on these topics herein are that some of the concepts, advantages and limitations of crawlers and APIs inspired some of the implementation choices and system features discussed later on.

2.4 Twitter

This section provides an overview of the Twitter API, one of the other basilar stones of this work.

2.4.1 Overview

Founded in 2006, Twitter is a social network and micro-blogging service allowing users to communicate with each other and with the community through the publication of texts called tweets and the exchange of messages between users. Twitter users are also referred to as Twitters. Twitters may follow other users in order to stay apprised of their updates and to find specific threads. Each user creates its own profile with a brief user biography, the tweets he made and those made by his or her followers. The information content generated by the general community can be accessed directly from the Twitter website or through external applications, using the Twitter API.

2.4.2 Twitter API

The Twitter API provides methods that allows developers to access and manipulate almost all available in Twitter, in an open but limited way, which can be integrated into applications that communicate directly with the Twitter site via HyperText Transfer Protocol (HTTP) requests. Crawlers scrapping this social network contents should thus be aware of the limitations that Twitter imposes. These limitations were also taken into consideration when developing the system described in the remaining part of the dissertation.

The Twitter API is composed of four `objects` that represent the main entities related to all the content on can find on this social network, as follows:

- `Users` - responsible for managing all content published on Twitter;
- `Tweets` - text messages up to 140 characters long, forming the basic building blocks for all Twitter content and generally representing an user status update;
- `Entities` - provide metadata and additional contextual information about content posted on Twitter;
- `Places` - specific named locations with corresponding geo coordinates.

To access and manipulate Twitter content it is necessary to use the methods provided by the API, requiring an authentication made by the client application. This authentication is performed

Use Case	Authentication Method
Sign in with Twitter button on your website	Sign in with Twitter
Read or post Twitter data on behalf of visitors to my website	3-legged OAuth
Have a mobile, desktop, or embedded app that can not use a browser	Personal Identification Number (PIN)-based OAuth
Just want to access the API from one own account	Tokens from `dev.twitter.com`
Required to use usernames/passwords **and** have been approved for xAuth	xAuth
Offer an API where clients send me data on behalf of Twitter users	OAuth Echo
Want to issue authenticated requests on behalf of the application itself	Application-only authentication

Table 2.2: Use cases and authentication methods to obtain access tokens.

using the OAuth authentication service that allows a third party application to get limited access to an HTTP service in a safe manner, since it is not necessary to share passwords between users and third party applications [OAu16]. The OAuth authentication service will generate four credentials that are listed and described below:

- The **Consumer key** — is essentially the API key associated with the Twitter application. This key identifies the client, which may be a website or service that is trying to access resources from an end-user;
- The **Consumer secret** — is the client password that is used to authenticate with the authentication server, in this case comprised by a Twitter server that authenticates the client;
- The **Access token** — is the token issued to the client after successful authentication (using the consumer key and consumer secret). This access token defines the privileges of the client, and the kind of data the client can and cannot access;
- The **Access token secret** — is sent along with the access token as a password (similar to the consumer secret), in order to access resources of the end-user;

Twitter offers two types of authentication to be used according to the needs of a user: *Application-user authentication* and *Application-only authentication*. When using the *Application-user authentication*, the outgoing call to the API will identify both the identity of the application itself and the identity of the user who makes the call. On the other hand, the *Application-only authentication* is used, only the identify of the application that makes the calls is sent. In this case, it is not possible to use methods which require users context such as, for example, the publication of a message. In both cases, it is necessary to get an OAuth access token to the application in order to communicate with the API. This access token can be obtained in several ways, as presented in table 2.2, in where some use cases and the appropriate authentication method are described.

According to the desired content and user requirements, several API variants can be selected

from the existing possibilities: Representational State Transfer (REST) API, Stream API, and Ads API. The REST API variant is responsible for the provision of methods which extract and publish the Twitter content in a static way, where replies to outgoing calls are available in JavaScript Object Notation (JSON) format. The Stream API variant is responsible for monitoring the Twitter content in real time. the REST API methods can be used in this variant in order to obtain information content in a dynamic way, for example, to get the tweets related to a specific user, published during the application execution. The Ads API variant allows integration in the Twitter Ad platform, enabling Twitters to develop tools to manage their own advertising campaigns.

The access to the API methods is conditioned by a request time frame of fifteen minutes, in order to prevent users from overloading Twitter servers with too many requests per time unit. As this social network is worldwide used, without these limits, Twitter server performance would be quickly hindered. The limits are set in [Twi16b] for all existing methods in the API according to the authentication type. The values shown indicate the maximum number of API calls that can be made without getting access denied.

2.5 Related Works

There are a number of approaches and applications for collecting and processing resources and information available on Twitter, but none of them follows a goal and strategy similar to the system that was implemented in the scope of this project.

Jeff Boleng [JB14] has developed a system to monitor relevant information about an attack on a diplomatic mission of the United States of America in Benghazi, Libya, which occurred on 11 September 2012, to inform users about the latest events related with the attack. The developed system consists in a collection of tweets posted in real time, filtered through a set of words associated with the attack and the geolocalization of the author of a tweet. The information is combined to provide a map to the user with the locations and descriptions of the events reported.

Martijn Spitters [MS14] developed an application for detecting eminent threats in messages posted on Twitter. Threat detection is achieved through the identification of the presence of certain keywords in a tweet and the relationship context of these words in various related tweets. For the purpose demonstrating the application, two datasets of tweets were build and used for testing: one containing tweets considered as *threatening* and another one containing the same number of tweets, but chosen randomly and that did not represent a threat.

Alex Wang Hai [Wan10] developed an application for spam detection on Twitter. Users are randomly chosen in real time along with their publications, their friends and their followers.

These users are then classified as spammers or not, according to a set of policies and metrics such as the misuse of mentions and replies, the existence of too many duplicate tweets, the relationship between friends and followers, among others.

Clive Best [CB11] developed a crawling system quite complex that allows users to collect content from various sources of online information. This system enables the addition of any web site or social network such as Twitter, Facebook, etc., implementing methods based on existing algorithms and APIs for the purpose of collecting and processing information. The results are presented to the users on a web platform.

2.6 Used Tools

The approach pursued in the scope of this work to have a dynamic OSINT system, based on Web data and Social Networks, led to the implementation of a system that combines the use of a number of existing tools and techniques. These are briefly presented in this section.

2.6.1 NetBeans IDE

NetBeans Integrated Development Environment (IDE) is a free and open source environment that enables users to develop desktop, mobile and web applications. The IDE supports application development in various languages, including Java, HyperText Markup Language (HTML)5, Hypertext PreProcessor (PHP) and C++. The IDE provides integrated support for the complete development cycle, from project creation through debugging, profiling and deployment [Ora13]. Since the developed system was built in Java with an HTML frontend, this IDE proved to be the most suitable IDE for the work at hands.

2.6.2 XAMPP and LAMP

Cross-platform (package with) Apache, MySQL, PHP, Perl and Python (XAMPP) is a free and open source cross-platform web server solution stack package, developed by Apache Friends, composed by the Apache HTTP Server, MariaDB database (former MySQL), and interpreters for scripts written in the PHP and Perl programming languages, used primarily in the development of web applications and deployment of web servers [Apa16]. The difference between the XAMPP and the LAMP platforms is that XAMPP can be installed in any operating system, while LAMP was implemented only for the Linux operating systems. In addition, the XAMPP platform comes usually with a compiler for the Perl programming language, which might not occur in the LAMP platform.

In this project, the XAMPP platform was used to develop the proposed system in a Windows environment, while the LAMP platform was used in the system deployment in the remote Amazon Virtual Machine (VM).

2.6.3 PuTTY

PuTTY is a free and open-source terminal emulator, serial console and network file transfer application that supports several network protocols, including Secure Copy (SCP), Secure Shell (SSH), Telnet, rlogin, and raw socket connection [PuT16]. It can also connect to a serial port. In this project PuTTY was used to establish a connection with the VM in Amazon, namely to transfer the files and configure the remote server.

2.6.4 PuTTYgen

PuTTYgen is a PuTTY component, consisting in a program to generate Rivest Shamir and Adleman (RSA) and Digital Signature Algorithm (DSA) keys and to convert keys from different file formats [Sof16]. In this project, PuTTYgen was used to convert a `.pem` file to a `.ppk` file.

2.6.5 WinSCP

Windows Secure Copy (WinSCP) is a free and open-source Secure File Transfer Protocol (SFTP), File Transfer Protocol (FTP), Web Distributed Authoring and Versioning (WebDAV) and SCP client for Windows. Its main function is to secure file transfer between a local and a remote computer [Mar14]. Beyond these features, WinSCP offers basic file manager and file synchronization functionality. For secure transfers, it uses SSH and supports the SCP protocol in addition to SFTP. In the context of this project, WinSCP was used to transfer files from the Windows environment to the remote VM environment.

2.7 Conclusions

This chapter discusses a number of technologies related to the subject and implementation of this project, namely the concepts of web crawler and API. These technologies can be combined together or used separately, to produce an OSINT system that allows the data collection and analysis from different information sources. The big difference between them is that while the implementation of a web crawler requires the creation of data collecting procedures from scratch, complying with a set of pre-defined policies, APIs have already a set of well established policies and come with all the necessary data handling procedures, ready to be used.

A brief discussion dedicated to the social network Twitter, which is the source of information for news gathering in the framework of this project, was also included. The essential concepts associated with the Twitter API were presented with the intention of helping to understand the entities and processes involved in the system implementation, which is the focus of the following chapters.

Towards the end fo the chapter, related works are discussed, focusing on techniques and tools employed by others for purposes similar to the ones of this work. Finally, a number of tools used in the development phaes of the system prototyped in the scope of this project are briefly

listed and described. In the next chapters, it will become clear that the implementation of this system and the involved concepts was achieved through the utilization and combination of these tools and technologies.

Chapter 3

System Architecture and Components

3.1 Introduction

This chapter list and describe the components that make up the developed system and the algorithms adopted to implement the two main features of the application: Tweeters Classification and Post Classification. This chapter is divided into four main sections: Section 3.2 is focused on the architecture of the system, discussing its main components; Section 3.3 presents and describes the essential steps for Tweeters classification; Section 3.4 discusses some natural language processing procedures, which are used for Post Classification; Section 3.5 presents some conclusions about the system components functionality, and the adopted approaches to classify Tweeters and posts.

3.2 System Architecture

The developed system consists of four essential components: the application developed in Java, the database associated with the application, the web platform and a VM hosted on a cloud service. Each component has a specific role in the functioning of the system, which is independent of its implementation.

3.2.1 The Java Application

The application developed in Java is responsible for the selection and classification of Tweeters, and the classification of the posts they publish. This application is divided into five main classes, organizing the various methods according to the type of data that each one interacts with. These classes are listed and described below:

- `SearchTwitter.java` — class that contains all the methods that make calls to the Twitter API and manipulate the received data;
- `DBconnection.java` — class containing all the methods responsible for the connection and handling data associated with the database;
- `UpdateCorpus.java` — class containing a call to the method responsible for keeping updated the news corpus[1]. This method is invoked regularly at certain time intervals;

[1] The concept of corpus is addressed later in this section.

- `UpdatesNewsRelevance.java` — class containing a call to the method responsible for maintaining the relevance of each post updated. This also an update procedure which will be invoked regularly at certain time intervals;
- `Main.java` — class responsible for starting and executing the main application loop. Is will also set the timer for the calling the methods in `UpdateCorpus.java` and `UpdatesNewsRelevance.java`.

In addition to these classes, several other classes and methods from the *hultiglib project* [Cor11] was also used, like `HashString.java`, in order to calculate the relevance of text segments.

3.2.2 Database

The database associated with the developed application is handled using the database management system MySQL and it is used for storing the relevant information of Tweeters and tweets they publish. Its schema is comprised of three tables, which can be listed and described as follows:

- `quarantine` — table where the tweeters data is stored;
- `inproduction` — table where the tweeters data is stored for users that have been selected to have their news in the main interface of the developed web platform;
- `news` — table that hold the data for the news published by tweeters stored in the `inproduction` table, along the execution of the application.

All tables have programmatically predefined a maximum number of entries, in order to prevent the size of the database to indefinitely increase, as well as ensure the gradually improvement of the quality of stored users. Management of data in the tables is carried out by methods contained in the Java application described in Section 3.2.1.

3.2.3 Web Platform

The web platform, developed using the HTML5 and PHP languages, provides the latest and most relevant news to the end users, within the domain of cybersecurity. In practice, the platform collects news from the `news` table and presents them to the user in a web page, organized by date of publication and highlighted by the relevance they have, through the use of different colors in each news background. Three colors were used to distinguish three relevance levels: red for high relevancy, yellow for medium relevancy, and green for low relevancy. For each news, the associated author, the news text, the keywords and the publication date are shown. The web page also provides a search field, where words that allow users to specify the displayed news can be introduced. Figure 3.1 is a screenshot that illustrates the web page layout that is presented to users.

Figure 3.1: The interface for the end user of the system developed in the scope of this master's project.

3.2.4 Virtual Machine (VM)

The VM hosted on a cloud service integrates the components described above, acting as a server for the implemented system. One advantage of using a cloud service is the possibility to assign a public Internet Protocol (IP) address to a VM, allowing it to be remotely accessed under certain restrictions set by the administrator of the machine. Therefore, the contents of the web platform can be accessed via the Internet. In this case the Elastic Cloud Computing (EC2) service from Amazon was the elected cloud service to host the application, and the operating system chosen for the VM was the Ubuntu Server 14.04 Long Term Support (LTS) Hardware Virtual Machine (HVM). Figure 3.2 illustrates how the interaction is performed between the end-user application and the various components mentioned above.

3.3 Algorithm for Classifying Tweeters

The Tweeters classification algorithm, included below, performs a dynamic management of the status of these users, based on the continuous calculation of their importance/value (line 6 of Algorithm 1).

During the execution of the application, tweets are continuously collected and selected according to a set of provided keywords. For each suitable tweet, the user name of its author is extracted for the purpose of evaluating his importance to the system, according to Equation 3.1. The result of this assessment determines the status that the user has in the system and the operations to be carried out accordingly.

Each Tweeter starts in the system's database as a member of the *quarantine state*, after his value/status being computed. Each one will there until his value, dynamically computed, reaches

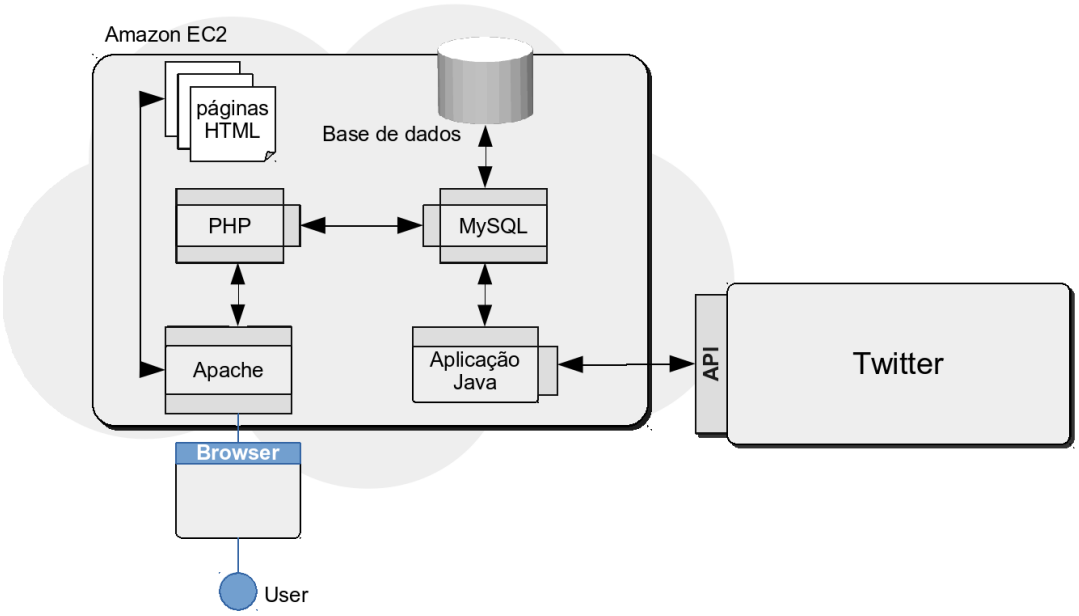

Figure 3.2: Developed system architecture.

a minimum threshold λ_{min}. The value of a Tweeter depends on the relevancy and frequency of his publications, as well as the number of his followers, and the number of likes and retweets of his publications.

If the user is already in the database, he will have one of two mutually exclusive status: quarantine or inproduction. If the user is in the quarantine state, he can either remain there or be promoted to the inproduction state, if the value of his importance is equal or greater than the maximum threshold value λ_{max}. On the other hand, if the user is in the *inproduction state*, he is now eligible to contribute with his news/posts to the web platform. He is now considered as a more relevant Tweeter for the system. However, the process continues to be dynamic and the Tweeter can be demoted and return to the *quarantine state* again, if his value decreases below the maximum threshold value λ_{max}.

Algorithm 1 also shows that the maximum number of Tweeters in `quarantine` $-$ Q $-$ and in `inproduction` $-$ P $-$ is limited, revealing the users transfer policies among the different statutes, when the thr thresholds are reached (see lines 20-26). Whenever the `quarantine` table is complete (full) and a new Tweeter becomes eligible to be inserted, the less important one will be removed to gain space for the new one, maintaining so the predefined fixed quotas. If the candidate's value is inferior than the actual worst value in `quarantine` table, thing is done. When the `inproduction` table is full and a new Tweeter have to be inserted, the one with the lowest importance is demoted to the `quarantine` table and the new one is inserted in the `inproduction` table, unless his value is inferior to the former's one.

The values assigned to the λ_{min}, λ_{Max}, Q and P variables will be discussed later in Chapter 5.

Algorithm 1 Tweeters Management Policy

```
1: procedure ManageTweeters
2: main cycle:
3:     tweets ← Filter(TwitterStream, keywords)
4:     for t ∈ tweets do
5:         u ← getUser(t)
6:         u.value ← Import(u)
7:         if u ∉ System then
8:             u.status ← "quarantine"
9:             Save(u)
10:        else
11:            if u.status = "quarantine" then
12:                if u.value > λ_max then
13:                    u.status ← "inproduction"
14:                end if
15:                if u.value < λ_min then
16:                    Remove(u)
17:                end if
18:            else
19:                if u.value < λ_max then
20:                    u.status ← "quarantine"
21:                end if
22:            end if
23:        end if
24:
25:        if |quarantine| > Q then
26:            μ ← MinValue(quarantine)
27:            Remove(μ)
28:        end if
29:        if |inproduction| > P then
30:            μ ← MinValue(inproduction)
31:            μ.value ← λ_max
32:            μ.status ← "quarantine"
33:        end if
34:    end for
35: end procedure
```

The insertion, removal and change of the Tweeters status is thus driven by the calculation of its importance value, according to the formula in Equation 3.1:

$$Import(u) = 1 - \frac{1}{3 + |Tweets(u)|} - \frac{1}{3 + UserRev(u)} - \frac{1}{3 + |Followers(u)|}. \quad (3.1)$$

This formula is one of the main contributions of this work. It determines the importance of a Tweeter u based on the combination of three characteristics: the number of tweets published by u in the last 30 days ($|Tweets(u)|$), the relevance of tweets published by u ($UserRev(u)$[2]) and the number of u followers ($|Followers(u)|$).

The $|Tweets(u)|$ feature provides an indication on the regularity of a user in terms of publications. A user may publish relevant news, but if he only publish them occasionally, he will not have as much interest to the community as users who publish more often. In turn, the $RevUser(u)$ feature provides the information about the relevance of publications made by a user. The relevance is important for distinguishing users who publish high impact content to the target community from other users. Users with high impact content should obtain a more prominent status on the end user interface. Finally, the $|Followers(u)|$ feature provides information about the followers of a particular user that are already registered in the database and classified as eligible for the publication of news. As these users are already considered relevant in the field of publishing security tweets, a new user who is followed by them will have more prestige and influence in the community when compared to others.

It can be seen that the $Impor(u)$ function is confined to the range $[0, 1]$ by definition, with 0 meaning no importance at all, and 1 meaning maximum importance. In practice, the maximum importance is never reached, but converges gradually to that maximum.

3.4 Algorithm for Classifying Tweets

In any social network, the importance of a user can be estimated in relation to several factors [ZL14]. For example, importance may be derived from measures of centrality, degrees of membership of virtual communities and user activity. In these last ones, measures related to the user's publications fall within, either in terms of quantity and in terms of content relevance. In social networks like Twitter, text comprises the major slice of the content published. Despite the limitation of 140 characters, there are many users who keep an intense publishing rhythm of several tweets per day, many times per hour. It is therefore important to know how to characterize a user by his published tweets, and know which users regularly publish relevant information in the field of cybersecurity.

To achieve this objective, natural language processing procedures were studied and integrated

[2]Defined through Equation 3.4.

in the developed application. The integrated procedures allow on one hand, to measure the relevance of a given post and, on the other hand, select the best users, from the perspective of quantity and quality of their published information. An important user is the one who regularly publishes relevant information (in this case, threats, dangers, warnings, solutions).

It is known that the concept of text relevance tends to be subjectively dependent on its interest to a group of readers. For example, for the same news concerning penetration of a web application, a security expert and an economist may consider different text information elements as being actually relevant. Hence it is necessary to first measure the information in a post and then calculate the relevance within the context of cybersecurity. The calculation of the amount of information from a post is based on the information of its words and the function $InfoTermo$ used by [Fel15] in automatic detection of plagiarism was adapted to this work for the purpose of measuring this amount of information. It is defined as follows:

$$InfoTermo(w|text) = \sqrt{|w|} * log_2 \frac{P\{w|text\}}{P\{w\}}, \quad (3.2)$$

where w is the considered word and $|w|$ its character length[3]. The $P\{w\}$ value is the probability of w estimated in a representative general corpus[4] of the language and $P\{w|texto\}$ is the probability of that same word but in a text subset for a given domain. For example, below one may find the 30 most informative ($InfoTermo$) words (within the cyber security context), for a collection of security news, from the known `ZDNet.com` website:

pentestpartners	virustotal	hacksthe	skilton	nopsec
pentestpartner	felberbaum	prioritization	nobuaki	sucuri
cryptocurrency	contexual	outlander	cyberattacks	vulnerabilities
cyberattackers	cleartext	droppers	plaintext	telnet
pyinstaller	irongate	bitcoin	plunders	untrusted
productcert	karpeles	stuxnet	cyberattack	evolutions

Thus, through this function and having a specific and representative domain corpus[5], it is not only possible to automatically select a specific domain vocabulary, but also to calculate the relevance of a post from a user. For this purpose, the *average InfoTermo* has been defined and used to compute the relevance of a post. Note that all Twitter posts have approximately the same number of words, restricted by the 140 characters limit. So, if $[w_1, w_2, ...w_n]$ is the list of n unique words that occur in a post, then the initial[6] relevance of that post is calculated

[3]The number of the word characters.

[4]A collection of texts, usually quite bulky.

[5]A collection, usually large, of texts, which in our case are from the security domain.

[6]The final relevance of a post is calculated through function $FinalRev$ (3.5), which takes into account additional variables.

according to Equation 3.3:

$$Relevance(post) = \frac{1}{n}\sum_{i=1}^{n} InfoTermo(w_i|security). \quad (3.3)$$

In turn, this function will provide a method to calculate the relevance of a user, in terms of what he publishes. For that, the use of average relevance is proposed and shown in Equation 3.4. If $\{p_1, p_2, ..., p_m\}$ are the last user's m posts, its current relevance would be computed as shown in the following equation:

$$UserRev(u) = \frac{1}{m}\sum_{p_i \in u} Relevance(p_i). \quad (3.4)$$

To calculate the final relevance of a post, Formula 3.5 is proposed, which not only depends on the information contained in the text, but also two additional variables: the number of likes ($\#Likes$) and retweets ($\#Retweets$) for that post, representing the post's community feedback. A post having a high number of likes and retweets is well appreciated by the community and thus more likely to be relevant.

$$FinalRev(post) = \frac{1}{2} + \frac{\arctan(\sqrt{(1+\#Likes)(1+\#Retweets)} * Relevance(post))}{\pi} \quad (3.5)$$

This function geometrically combines the the number of likes and retweets, which will act as an amplifier of the original post relevance. In terms of range the use of the $arctan$ function with additional constants will confine the $FinalRev(post)$ function to the $[0, 1]$ interval, with 0 meaning no relevance and 1 meaning maximum relevance, which in practice is also never reached but converges gradually to that maximum.

3.5 Conclusions

This chapter introduced the architecture for the developed system and the two underlying algorithms responsible for its main features. Regarding the architecture, four distinct components were developed to implement the procedures associated to tweeters and posts classification, to store their associated data, to present the news to the end users, and to facilitate the access to the news from the community in general. As it will become clearer in the next chapter, it is the combination of the components discussed herein that will allow fulfilling the main objectives of this work.

The algorithms for classification of twitters and posts are better described in the second part of the chapter, by introducing and discussing also the supporting mathematical expressions behind them. These functions are one of the novelties of this work, and they combine several char-

acteristics of users and posts (such as number the number of followers, publication frequency, and number of likes and retweets), along with NLP metrics, to describe the importance of users in the community or the relevance of posts. The formula for the classification of posts gives more emphasis to the relevance of the text. Both formulas were designed to output normalized values, easing the understatement of their meaning.

Chapter 4

Implementation

4.1 Introduction

This chapter discusses the arrangements and settings required for the implementation of the several system components. The parameters to be set and the necessary settings supprt the communication between the various components in order to achieve the proposed objectives.

The implemented dynamic OSINT system includes the configuration of several components, as it can be seen in the architecture presented in section 3.2. The system *observes* the current flow of information on the Twitter social network, selecting the most important posts in the cybersecurity field and presents them in a wall (page) on the Web. This proactive and dynamic selection contributes also to classify the relevancy of Twitters, selecting those that become more noticed, due to their relevant postings, and to manage their inclusion and status in the database. This Twetters relevancy is decisive in the process of selecting new tweets. The entire system platform is maintained in the cloud.

The chapter is stuctured as follows. Section 4.2 focuses on the implementation of the Java application and interactions with the database. Section 4.3 discusses the web front-end in a brief manner, while section 4.4 describes the configuration and setting up of the remote VM hosting the system.

4.2 Java Application and Database Implementation

Initially, the system implementation involves the definition of some key resources necessary for its operation. The system is implemented in Java and initially, it requires the addition of several libraries for handling resources, such as access to the Twitter API, through the `Twitter4J` library [Yam07]. The `HultigLib` library [Cor11] was used for text handling and resources concerning NLP. This library is based in other free access NLP resources, such as the OpenNLP [Mor05]. The `Connector/J` library [MyS16] was also used to enable communications with the system database.

Having the main external libraries setup, the necessary access credentials for the Twitter API must be obtained. In this case, it is necessary to use the system administrator account in order to get the access credentials. The necessary credentials and tokens were obtained from

`https://apps.twitter.com`, using method the fourth method mentioned in Table 2.2. First, the system administrator must have a registered Twitter account and then create a new Twitter application (set of credentials) by filling in a form where he inserts the application name, a description and the associated website (if applicable). The user can then obtain the credentials that will allow the client applications to get access to the Twitter API and resources. In particular he will obtain the `Consumer Key`, the `Consumer Secret`, the `Access Token`, and the `Access Token Secret`, as shown in Figure 4.1.

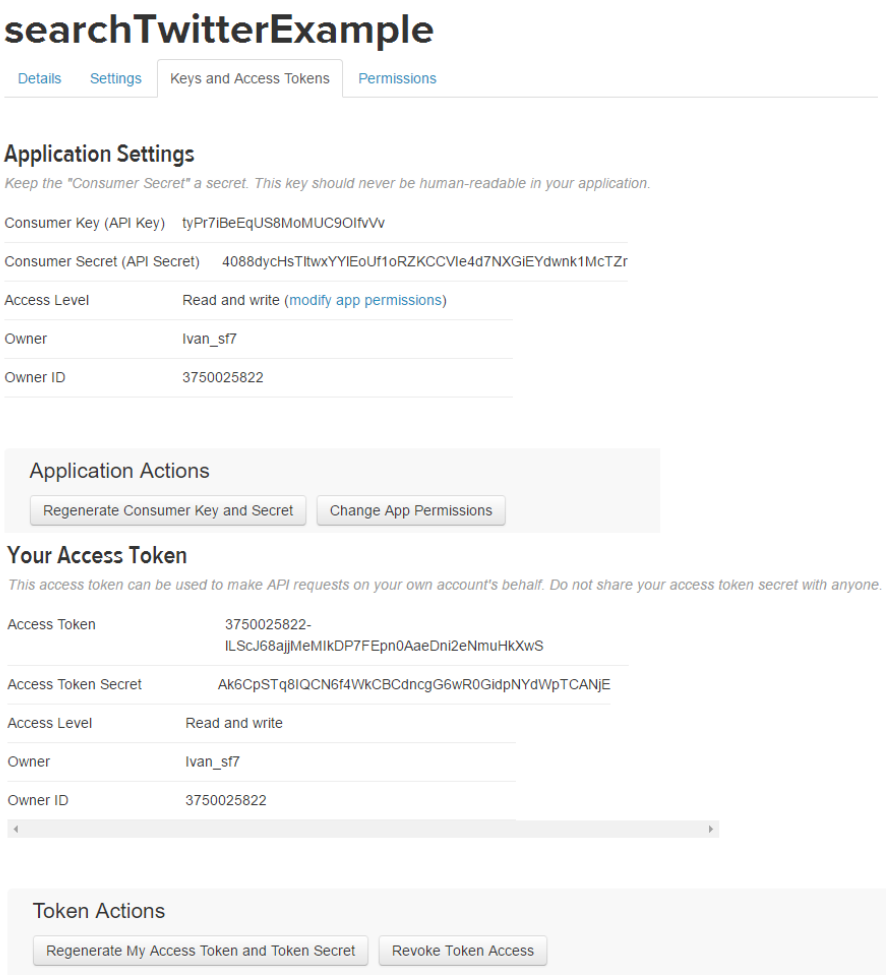

Figure 4.1: Twitter application settings.

After the obtention of such credentials, it is then possible to create an object that can communicate with the API. For that, the credentials, mentioned above, must be provided to the method responsible for creating an authenticated object to call the API (`getTwitterObject(...)`), as shown in Listing 4.1.

Having met the aforementioned conditions, four applications were generated, one for each type of API call since, according to the terms and conditions for the use of the Twitter API, each set of credentials can only be used for a use case [Twi16a]. Thus, the generated sets of credentials

Listing 4.1: Method for creating a Twitter object.

```
public static Twitter getTwitterObject(String CONSUMER_KEY, String CONSUMER_SECRET,
          String ACCESS_TOKEN, String ACCESS_TOKEN_SECRET) {
       ConfigurationBuilder cb = new ConfigurationBuilder();
       cb.setDebugEnabled(true);
       cb.setOAuthConsumerKey(CONSUMER_KEY);
       cb.setOAuthConsumerSecret(CONSUMER_SECRET);
       cb.setOAuthAccessToken(ACCESS_TOKEN);
       cb.setOAuthAccessTokenSecret(ACCESS_TOKEN_SECRET);
       //Create object that communicates with API
       return new TwitterFactory(cb.build()).getInstance();
   }
```

Listing 4.2: List of keywords used to filter news

```
private static final String keywords[] = {"information security", "cyber security",
       "infosec", "cybersecurity", "websecurity", "cyber-security", "privacy",
       "cybercrime", "cyberwar", "malware", "informationsecurity", "datatheft",
       "dataloss", "dataleak", "CISO", "CSO", "hacker", "mobilesec", "mobilesecurity"};
```

are used: (i) to obtain sets of posts for specified Twitters; (ii) in order to collect user followers; (iii) to obtain likes and retweets of specific posts; and (iv) to create data streams, from which relevant Twitters are selected and stored in the database. Excepting the creation of a data stream, all objects associated with the correspondant authentication credentials are created according with the method shown Listing 4.1. In the case of the data stream, the established procedures for obtaining its authenticated object are the same presented in Listing 4.1, with the exception of the returned object. In this case, it is an object of the **TwitterStream** class, instead of belonging to the **Twitter** class.

As the posts will be selected according to a list of keywords, their definition is a very important step in the implementation of the Java application. To this end, words were selected from the list on the site `https://top-hashtags.com/hashtag/cybersecurity/`. This site covers numerous issues related to cybersecurity. Listing 4.2 shows the custom set of keywords used to filter the collected news.

The creation of a corpus is also essential for the work at hands, as it is this set of texts that allows analysis and assigning a relevance value to a text segment, as shown in Equation 3.3. The creation of this corpus is made using methods provided by the `HultigLib` library [Cor11], which allows loading different text files and returning a single file corresponding to the create corpus. As a corpus specific to the cybersecurity area was not found, an automatic approach consisting in a continuous collection of news published on the `https://threatpost.com` site (The Kaspersky Lab security news service) was adopted. This site is an independent news site that is a leading source of information about Information Technology (IT) and business security for thousands of professionals worldwide. The collection is perfomed via a script written in

Listing 4.3: Method for creating or updating a corpus.

```
HashString hs = new HashString();
try {
     Runtime.getRuntime().exec("/home/ubuntu/feedExtractor/updater.sh /");
  } catch (IOException ex) {
     Logger.getLogger(updateCorpus.class.getName()).log(Level.SEVERE, null, ex);
  }
  try {
     Thread.sleep(10000);
  } catch (InterruptedException ex1) {
     Logger.getLogger(updateCorpus.class.getName()).log(Level.SEVERE, null, ex1);
  }
  hs.processFile("/home/ubuntu/CSecurityPosts.txt", true, false);
  hs.save("/home/ubuntu/corpus");
```

python (`myFeeds.py`), that collects the latest news published in the aforementioned website and stores them in a text file (`CSecurityPosts.txt`), also storing the URL of each news in an additional text file (`listOfStoredPosts.txt`). Storing the URLs of the news prevents that repeated news are collected. The corpus to be used in the system is then produced from the `CSecurityPosts.txt` file, as shown in Listing 4.3.

The whole system is supported by a database that needs to be created and initialized in order to make the system work. No specific requirements are made for the Database Management System (DBMS) and, as long as the database respects the schema shown in Figure 4.2, any DBMS with Java support should work. Nonetheless, it is necessary to adjust the variables that support the establishment of communication between the Java application and the database, namely `host`, `DBname`, `username` and `password`. As previously mentioned, the database consists of three tables, each one containing several attributes that can be viewed in Figure 4.2.

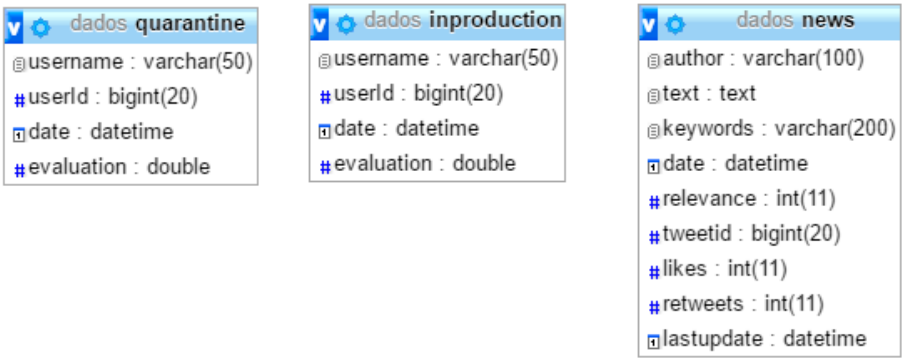

Figure 4.2: Database tables with their attributes.

The value of the variables that influence the functioning of the several algorithms must also be set in the system. These variables are the limit of entries in the tables of the database, the minimum and maximum thresholds used in rating of Tweeters and the minimum and max-

imum threshold values used in post classification. These parameters were defined based on experimental observation, according to the discussion in Chapter 5.

Afterwards, the system follows the execution steps of Algorithm 1. As mentioned in Section 3.3, insertion, removal or change of the status of a Twitter user is dependent on the calculation of the value of their importance according to Equation 3.1. This equation has features that are produced via calls to the Twitter API, which means that it can only evaluate 15 posts and their authors per each 15 minutes interval [Twi16b] (15 is the maximum number of outgoing calls to the API for obtaining the followers of a particular user). During the algorithm execution, and if 15 posts and theirs authors have not been evaluated yet, whenever a user is in the `inproduction` table and it is not demoted to the `quarantine` table (algorithm lines 7-9), the system will collect and evaluate his posts, according to the Equation 3.5. If applicable, the posts are stored in the system database and later displayed on the web page. This procedure is also applied to a user who has been promoted to the `inproduction` table (algorithm lines 11-13). If 15 posts and their authors have already been evaluated in the current 15 minutes interval, the system will not be able to get responses from the API calls. In this case, the algorithm execution is paused until the next 15 minute interval.

Finally, the execution of the application consists in the implementation of the method shown in Algorithm 1, along with the classification of posts and the implementation of two complementary tasks. The complementary tasks are the update of the used corpus and the update of the posts relevance. As it is desirable that the corpus size increases with time, the method in listing 4.3 (responsible for the execution of `myFeeds.py` script) is frequently run, so as to collect the recent news and update the corpus. The corpus update will be continuously carried out at the start of application execution and at every 24 hours since the last update.

The update of the posts relevance is also an essential step because, as the news are collected at the time of its publication on Twitter, they may have zero likes and retweets, which deeply affects their evaluation. As such, a method included in the `upgradeNewsRelevance.java` responsible for the selection of news, sorted in ascending order by the last update date, and for its evaluation is invoked, updating the value of its relevancy in the database. As this method uses calls to the Twitter API, it can only evaluate 180 news per each 15 minutes [Twi16b] (180 is the maximum number of outgoing calls to the API for obtaining likes and retweets of a specific post from a given user). The execution of this method is done 10 seconds after the start of the execution of the method responsible for the corpus update, to be sure that the corpus has been updated already. It is then repeated every 15 minutes and 10 seconds after the first update, to avoid exceeding the limits for the maximum number of API calls.

The final Java application consists of an executable file with the `.jar` format. It stores the compiled classes and metadata associated with the developed Java application, to favor portability

Listing 4.4: Method for selecting news containing a list of provided keywords.

```
row = mysqli_fetch_array($result);
$_autor = $row[0];   $_texto = $row[1];
$_keywords = $row[2]; $_data = $row[3];
$_importancia = $row[4];
if ($keywords == '')
   {
      echo '<script language="javascript">';
      echo 'alert("Insert the keywords before pressing search button!")';
      echo '</script>';
      break;
   }
for ($j = 0; $j < $max; $j++)
{
   if (strpos($_texto,$array[$j]) !== false)
   {
      switch($_importancia)
```

and so that it can be executed in the remote VM.

4.3 Mural Implementation

The mural where the most relevant posts are displayed is web page implemented using HTML5 and Cascade Style Sheets (CSS). The connection with the database and generation of contents dynamically is achieved using PHP and JavaScript. The mural is the human-machine interface for the system, where the collected news by the Java application are present to the end-users. This implementation meets the characteristics listed in Section 3.2.3.

The mural (and the supporting features) is comprised of three files, which can be listed and described as follows:

- `index.php` – is the main file, whose pre-processing and rendering delivers the mural. It is responsible for collecting the news from the database, duly sorted by date, and present them to the users, after highlighting their relevance with colors;
- `search.php` – is a secondary file, responsible for collecting the database news containing keywords provided by a user. Listing 4.4 contains a small excerpt of code that illustrates the functioning of this feature. Keywords are matched within the contents of the posts. JavaScript is used to provide feedback to the user related to warnings;
- `osint.css` – is the CSS file used for formatting the contents of the mural.

4.4 Virtual Machine Configuration

One of the objectives of this project was to make its results readily available to end-users, even if in a prototyping stage. The migration of the Java application, the database and the

web page to a VM hosted on a cloud service is thus an essential step to achieve that objective, since it will enable the web page to be accessed by any user with an Internet connection. The cloud service provided by Amazon, called Amazon Web Services (AWS), was selected to host the system. It offers a wide range of products including storage, databases, analysis, networking, development tools, management tools, Internet of Things (IoT), security, computing and enterprise applications. These services provide to users and companies a greater computing power, increasing the scalability of the developed products and reducing the cost associated with management of a physical infrastructure. AWS allows the creation of an account and free access to services over 12 months, subject to compliance with certain usage limits. The free services available in AWS and its limitations can be found in [Ama14].

In order to use the AWS it is necessary to create a free account by filling the form found in `https://aws.amazon.com/console/`. A user may then log into the created account and select the EC2 service (the only service used in the development of this project), consisting of a web service that provides elastic capacity and resources to cloud computing. By choosing the EC2 service, it is necessary to create and configure an instance of a VM, by filling a form where a number of parameters are defined. The form contains several steps, which are listed and described below along with the procedures to be taken in each one of them:

1. **Choose Amazon Machine Image (AMI)** — AMI is a template that contains the settings for the software to be installed on the VM (operating system, application server, and applications). In this step, the option *Ubuntu 4.14 LTS Server (HVM)*, containing the Ubuntu Server operating system for a 64 bit architecture, was the selected one;

2. **Choose Instance Type** — instances represent virtual servers that can run applications. Various types of instances are available with different combinations of Central Processing Unit (CPU), memory, storage, and networking capacity, allowing the choice of the appropriate set of resources for the application to be developed. In this case, the option *t2.micro* was the selected one;

3. **Configure Instance** — instance configuration allows the user to start multiple instances of the same AMI, request Spot Instances to take advantage of lower prices, assign an instance access management role, among others. The default option was taken in this step;

4. **Add Storage** — allows the definition of storage configurations for the VM, such as changing the used storage device size and the addition of new storage devices. The default values were the selected ones a this step, since they fulfill the storage requirements;

5. **Tag Instance** — allows the categorization and to group the AWS resources in several ways, for example, by the role that each plays in the system or by the owner of that resource. No tags were added as they were not necessary for this VM;

6. **Configure Security Group** — it is the definition of the firewall rules that control traffic to the created VM. In this step, two input rules were added to allow incoming traffic corresponding to the SSH protocol, from the system administrator IP address, and another one to allow incoming traffic corresponding to the HTTP protocol, from any IP address, as shown in Figure 4.3. These two rules state that only the administrator can have access and control over the VM, and that all other users have access to the hosted webpages.

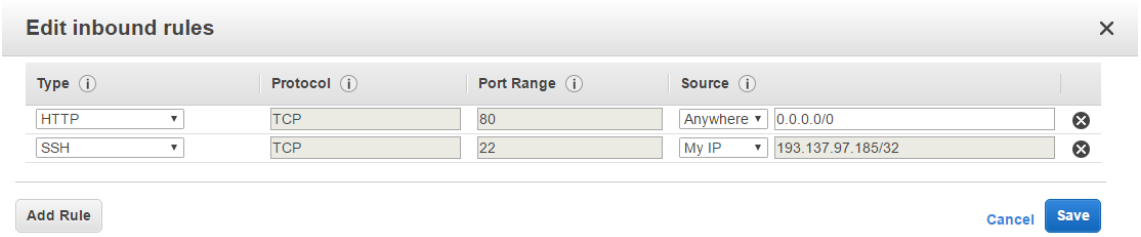

Figure 4.3: Inbound rules for the VM firewall.

7. **Review** — this step presents all settings made for the VM, which can be changed if necessary.

Once the form is submitted, one is prompted with the creation of a pair of keys (a public key that AWS stores, and a private key that the user stores) so that the connection to the VM is performed in a confidential and authenticated manner. In the case of Linux AMIs, the private key is used to perform communications with the VM using the SSH protocol. The generated key is a file with the `.pem` format. After the creation of the keys, the VM is fully configured and can be initialized. Figure 4.4 illustrates the state and the settings of the created VM.

The installation of additional packages, resources and the developed system in the VM is only possible after having it running, through an SSH connection. As the system was initially developed in a Windows environment (and later migrated to Linux), Putty was used for managing the SSH communication with the VM. Nonetheless, *Putty* uses the `.ppk` format for the authentication keys and, as such, the keys in `.pem` format retrieved during the setup of the VM had to be converted with the help of `puttygen`. This program requires tree inputs: the `.pem` file to be converted, the password associated with the `.ppk` file that will be generated, and the desired conversion mode, as shown in figure 4.5.

In order to connect to the remote VM, it is necessary to adjust several parameters in Putty, including the name of the VM, which corresponds to the value of its public Domain Name System (DNS) variable. This value can be found in the characteristics of the VM (see figure 4.4). The private key location needs to be adjusted too, as shown in Figure 4.6. After this process is completed, a command prompt is opened, requesting the insertion of the username for logging into the system (in the described setup it was `ubuntu`), and the corresponding password

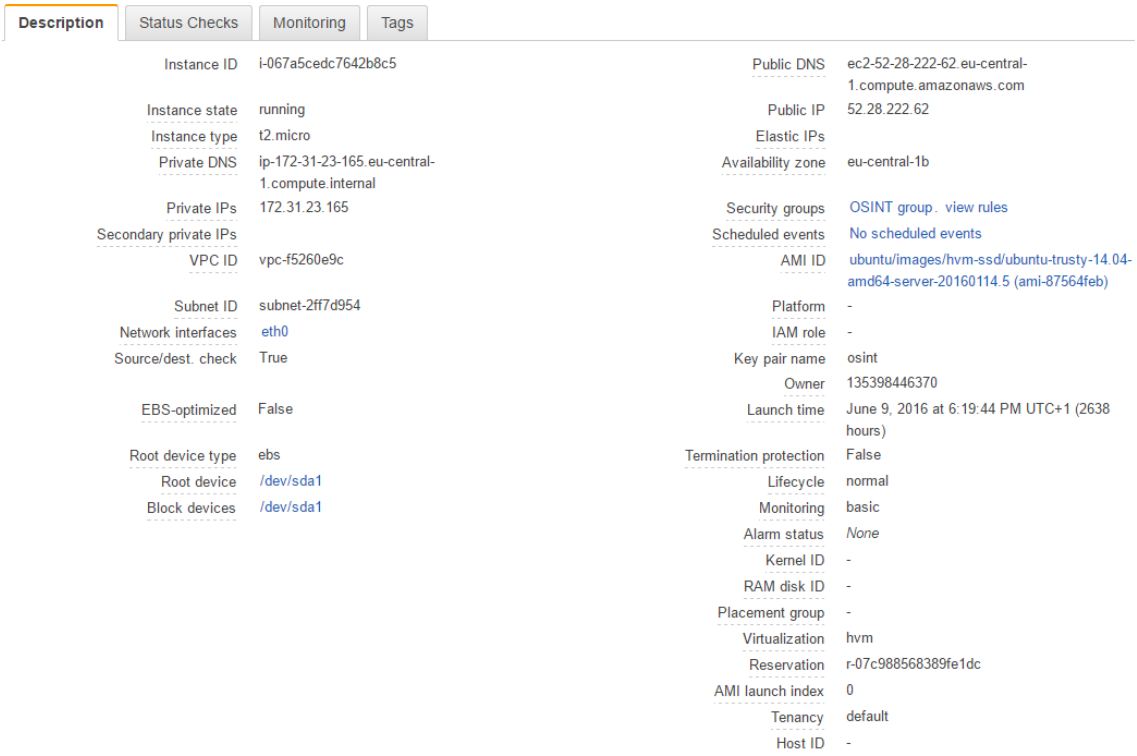

Figure 4.4: AWS VM settings.

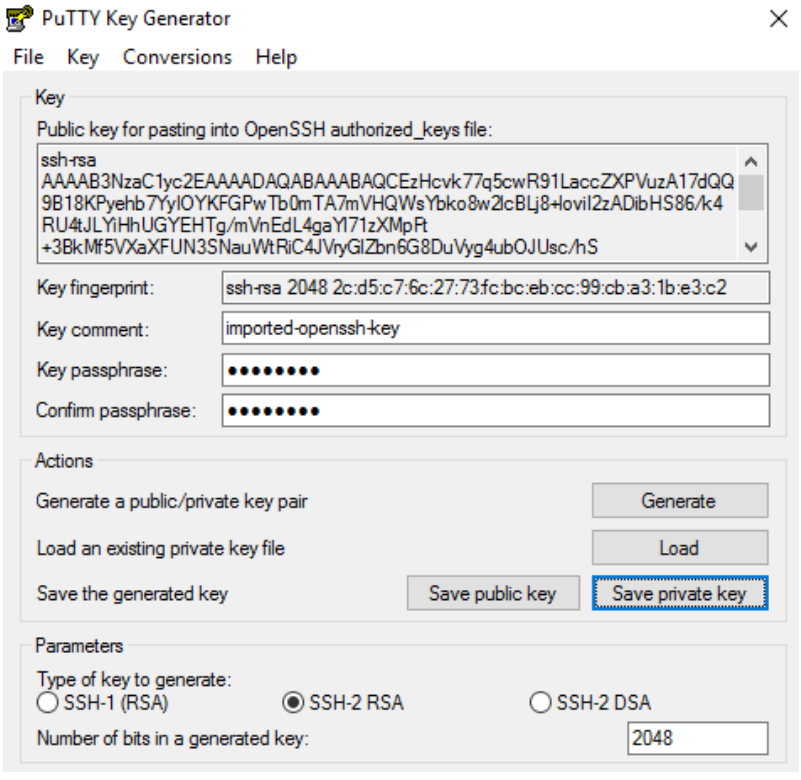

Figure 4.5: Conversion of the private key `.pem` format to the `.ppk` format.

associated with the private key. One can then control the VM remotely.

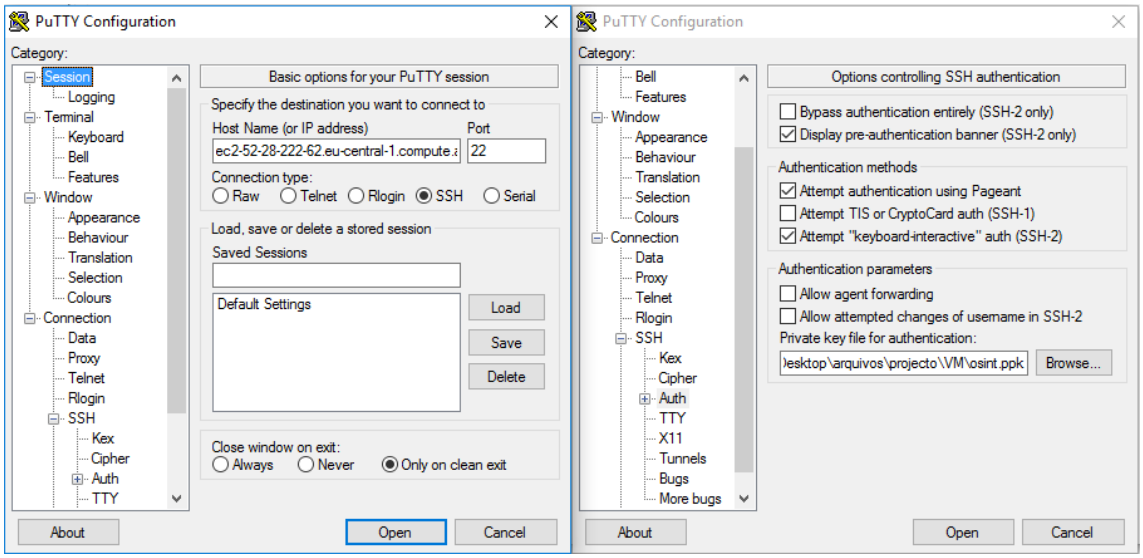

Figure 4.6: Establishing an SSH connection with the VM.

Since the system was mostly developed in Java (with some parts in Python, PHP and HTML), it is necessary to install the appropriate execution environment. The current prototype was tested and shown to be working with Java Development Kit (JDK), which can be easely installed using a command similar to the following one:

```
$sudo apt-get install default-jdk
```

The application requires the storage of information in a database, which is later accessed and organized into a web page. As such it is also necessary to install the services comprising LAMP, formed by the Apache web server, the DBMS MySQL and the PHP language. This may be achieved with commands similar to the following ones:

```
$sudo apt-get install apache2
$sudo apt-get install mysql-server
$sudo apt-get php libapache2-mod-php php-mcrypt php-mysql
```

The script in `myFees.py` uses several external libraries that need to be properly installed. The usage of `pip` (a package management system for Python) will probably make this installation simpler. This package management system for Python 3 can be installed using a command similar to the following one:

```
$sudo apt-get install python3-pip
```

The `feedparser`, `bs4`, `nltk` and `python3-lxml` libraries can then be installed using:

```
$sudo pip3 install feedparser
$sudo pip3 install bs4
$sudo pip3 install nltk
$sudo apt-get install python3-lxml
```

At this point, the VM is practically configured to run the developed system, lacking only the creation of the database, the creation of a user with all privileges on that database, the migration of the files that make up the web page and the installation of the executable `.jar` files, corresponding to the Java application, for the VM. The database can be created directly on the VM or imported from the Windows environment, where it has been initially developed, according to the model shown in Figure 4.2.

The following two lines were issued in the MySQL shell to create the user `osint` on the DBMS with full privileges over `DBname` (the true password was is not shown):

```
$CREATE USER 'osint'@'localhost' IDENTIFIED BY 'password';
$GRANT ALL PRIVILEGES ON DBname.* To 'user'@'localhost' IDENTIFIED BY 'password';
```

The true name of the database, username and true password do not have to follow any rule in order to the system to work, provided that these parameters are also set in the Java code. During the course of this project, the program WinSCP was used for the migration of data and programs to the VM. This program requires also the name of the VM, the associated private key location and the username logging into the VM, as shown in Figure 4.7. The webpage files were copied to the the `/var/www/html/` directory and the `.jar` file was copied to the `/home/ubuntu/` directory of the remote system.

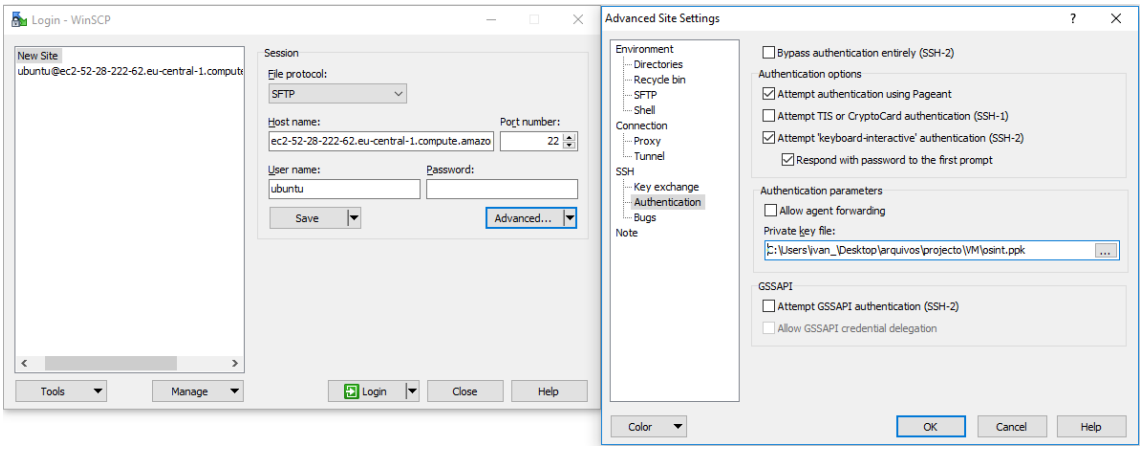

Figure 4.7: Establishing an SSH connection with the VM.

Once all conditions are met, the system can be initiated with a command similar to the following one, where the `nohup` instruction provides the means that allow a continuous execution of the application, even if the user closes the communication with the VM:

```
$nohup java -jar OSINT.jar &
```

Notice that `OSINT.jar` is the application name and `&` indicates that this application will run as a background task. The application output is saved in a file named `nohup.out` for logging reasons, which can be accessed with a common text editor.

4.5 Conclusions

This chapter describes the steps necessary for the proper implementation and combination of the system components. Even though the required features are easy to understand, their integration and adjustment of parameters need to be considered with care. A significant number of conditions have to be met in order for the system to work properly, mainly because of the number of technologies involved. In the current embodiment of the system, a VM was instantiated in AWS, with several additional packages installed after system configuration.

The Java application requires the adjustment of unique credentials, corresponding to every type of data that will be collected from Twitter through calls to its API. If the same credentials are used for different types of calls, or if multiple sets of credentials are used for the same type of call, the associated user account may be banned after a while, jeopardizing system execution. Having the system fully operation, the next phase (and chapter) was focused on the evaluation of the algorithms.

Chapter 5

Application Evaluation

5.1 Introduction

After the implementation of the system components, it was necessary to evaluate its performance, in order to verify if the proposed objectives for this project had been achieved. More specifically, tests were performed to set the values of the parameters associated with the limits of the database tables and the parameters that control the execution of the developed algorithms, presented in chapter 3. Then, tests were performed to assess the effectiveness of the methods for Tweeters classification and post classification. Ratings are assigned according to the definition of the parameters mentioned above. At the end, the results are discussed and some conclusions are presented regarding the most interesting findings.

The performed tests consisted in the collection of information in the database and obtained from the execution of the Java application. Values obtained from the execution of the Java application presume that some key parameters in the algorithms for classification of Tweeters posts were already defined before. These parameters are the minimum λ_{min} and maximum λ_{max} thresholds for Tweeters rating, the minimum ρ_{min} and the maximum ρ_{max} thresholds for the post classification, and the maximum number of entries for the database tables, namely `quarantine` Q, `inproduction` P and `news` N. As no tools allowing the evaluation of the obtained classifications in the application were found, the assessment was performed through a subjective analysis of the results, obtained from the use of the methods implemented for ratings. The purpose is to establish a comparison between the results obtained in the application and the results obtained through the analysis that a human would do with the same type of resources.

Section 5.2 is focused on the definition of the thresholds for classification of tweeters and on the steps that were carried out to achieve that objective. Section 5.3 presents and discusses the results from the classification of tweeters evaluation. Section 5.4 is focused on the steps performed to estimate the thresholds for classification of posts. Section 5.5 presents and discusses the results obtained during the classification of posts evaluation. Section 5.6 is focused on the procedure used to find the limit for the entries of the tables in the database and section 5.7 presents some conclusions about the adopted approach for the evaluation of the performance of the algorithms.

5.2 Thresholds Definition for Tweeters Classification

The first main experiment was conducted to evaluate the performance of the algorithm for classification of Tweeters. This algorithm assigns a rating to tweeters, which represents the importance that they have for the developed system. The tweeter importance is calculated using Equation 3.1 and the classification attributed to him or her is dependent on the thresholds ρ_{min} and ρ_{max}. These thresholds regulate the admission of a user in the system and for his or her election to post news in the main mural.

For the definition of the thresholds regulating the classification of Tweeters, an experiment comprising the random selection and evaluation of 1000 users (lines 5 and 6 from Algorithm 1) was performed. This experiment did not require the definition of the parameters mentioned in section 5.1, since it is not necessary to handle or store information involving the classification of Tweeters and posts in the database. Only the relevance values are involved. The importance attributed to each one of the selected users can be seen in the chart included in Figure 5.1.

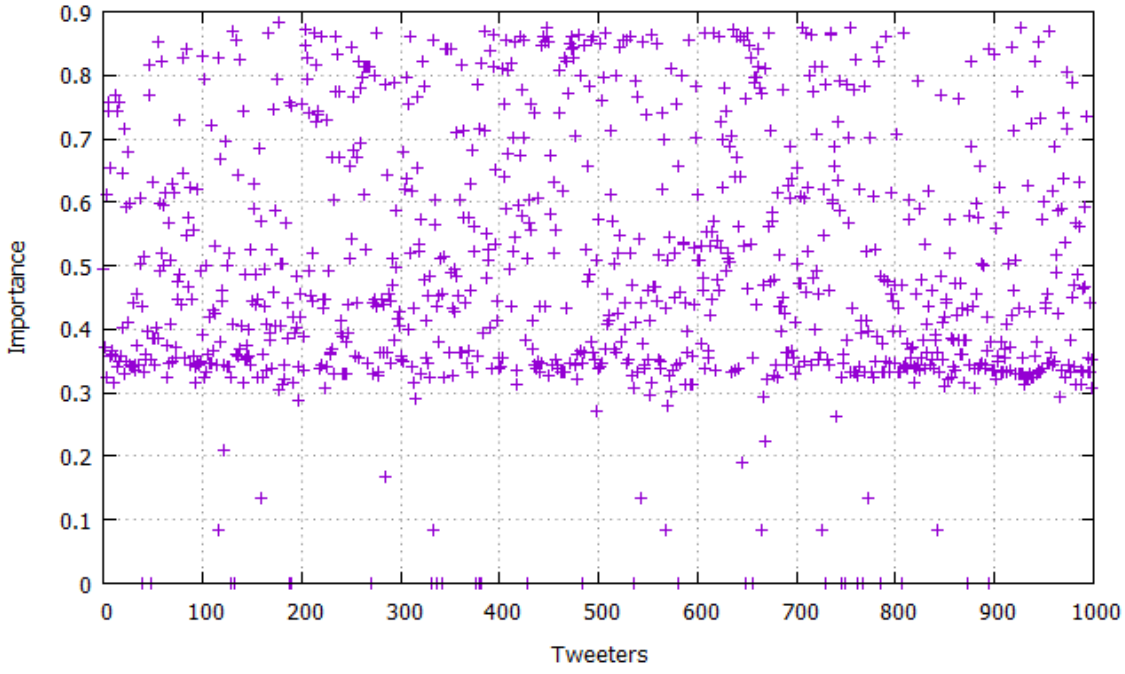

Figure 5.1: Importance value for 1000 randomly selected tweeters.

The results show an average importance of 0.4987 and a standard deviation of 0.2011, with a minimum value of 0.000 and a maximum value of 0.8820. Looking at the chart, it can be seen that most of the selected tweeters have an importance contained in the range $[0.3, 0.6]$. Thus, the average value may be taken as reference for setting the parameter λ_{max}. However, it is important to note that, because the database is initially empty, the value concerning the amount of followers in the database may be zero or quite low for each user. This means that the importance of the tweeter for the system is essentially dependent on the average relevance

of the associated posts and publication frequency in the last 30 days. Therefore, it is expected that the growing number of registered tweeters in the database leads to an increase in their average relevance, as shown in the charts included in figure 5.2 and 5.3.

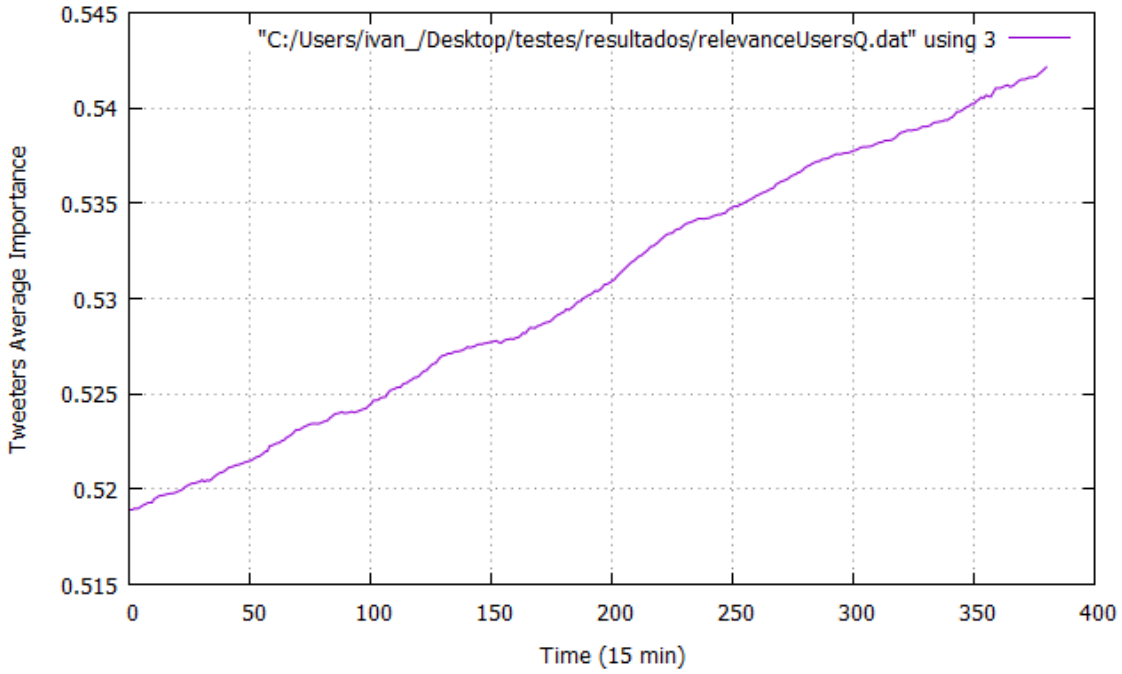

Figure 5.2: Average importance for tweeters in the `quarantine` table over time (15 minutes interval).

The results shown in the charts were obtained by calculating the average importance for tweeters in the `quarantine` and `inproduction` tables. The calculation was carried out during the execution of the application, at every 15 minutes interval, which represents the time frame when new tweeters were added into the tables.

It is important to notice that, in an initial phase, λ_{max} should not be assigned an high value, since it would make the classification of tweeters to be overly exclusive. It would lead to the admission of too few tweeters for news production, eventually leading to the loss of news that could actually be relevant to the community in general. Regarding these aspects, it was decided to assign the value 0.5 to the λ_{max} parameter.

Initially, a low value was assigned to the λ_{min} parameter so that, at an early stage, a larger number of tweeters are admitted into the database. Although the chart in figure 5.1 shows that there is a low number of tweeters with lower relevance than 0.3, they can also be of some importance to the system, in the sense that they can publish relevant news less frequently. That said, it was decided to assign the value 0.1 to the λ_{min} parameter, which leads to the admission of most of the selected tweeters to the database, except those who have published only one or two news with low relevance in the last 30 days.

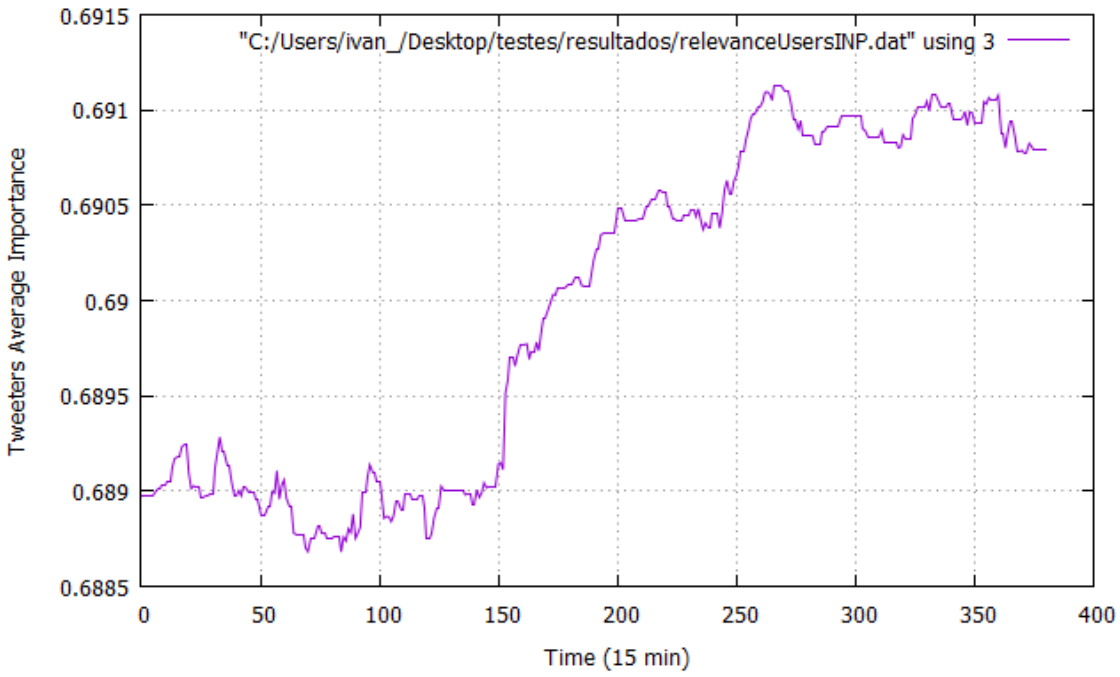

Figure 5.3: Average importance for tweeters in the `inproduction` table over time (15 minutes interval).

5.3 Evaluation of the Classification of Tweeters

After the definition of the λ_{min} and λ_{max} parameters, responsible for the classification of tweeters, it is necessary to verify the effectiveness of this classification. The assessment of that effectiveness resorted to an experiment in which 10 tweeters that were classified for the production of news for the web platform, along with their importance value, were analyzed by 3 human evaluators. All evaluators are working in the cybersecurity area or related subjects. They were asked to assign a classification to these tweeters, according mostly to their opinion, but if possible partially based in the same values that the system uses. Two parameters were defined for that rating: the relevance of the news published by tweeters and the frequency in which they publish news related with cybersecurity. These parameters values were rated on a scale of 0 to 20. The assignment of the value associated with these parameters implies that the human subject has a Twitter account, so that he can access the personal pages of the selected tweeters and analyze the news therein. Table 5.1 shows the results obtained automatically (assigned by the system) and by the human evaluators, for the 10 selected tweeters.

The values in the table suggest that the human evaluators agree that all evaluated tweeters have a significant rating for the published news relevance or publication frequency. These results indicate that these tweeters are in fact important for the system. From the analysis of the twitter pages of each one of these tweeters, it was also possible to conclude that the difference between the importance values assigned by the system to tweeters indicate that, typically, a tweeter with a low importance represents a user that posts, on his page, news about

Username	Importance (application)	Number of Followers	5 Latest Evaluated Tweets Average (human analysis)	Publication Frequency (human analysis)
CorkBIC	0.5203774497055746	3	17	3
10903	0.5844967768375229	1	10	14
BlinkeredJust	0.6132023882413031	2	14	7
tscalzott	0.6770489299725574	9	15	5
SiliconBeachHub	0.7159181560475899	21	15	6
OmniCyber_Sec	0.7570659763963441	12	16	7
ConnectionIT	0.7890891645714112	13	14	10
CyberDomain	0.8351202071494586	172	11	19
JulioCyberSec	0.8589738831749891	134	13	17
Cyber_Healthy	0.8726923035644377	67	14	15

Table 5.1: Results for the classification of tweeters obtained by the system and by the human evaluators (subjective analysis).

various topics. Conversely, a tweeter with an high importance represents a user who mainly publishes, on his page, news related with the cybersecurity field.

5.4 Thresholds Definition for Post Classification

The second main experiment was conducted to evaluate the performance of the algorithm for post classification. This algorithm assigns ratings to posts, which represent their relevance to the system. The post relevance is obtained using Equation 3.5, and the classification value is conditioned by the minimum ρ_{min} and maximum ρ_{max} thresholds, set for the assigning of one out of three possible ratings: 0 for posts with low relevance ($rev(post) < \rho_{min}$), 1 to posts with average relevance ($\rho_{min} <= rev(post) < \rho_{max}$), and 2 for posts with high relevance ($rev(post) >= \rho_{max}$).

Two experiments were performed for the definition of these thresholds. Following these experiments, the values 0.1 and 0.5 were assigned to λ_{min} and λ_{max}, respectively, since tweeters are only collected for posting news in the web page if their importance is greater than λ_{max}. In these experiments, the setting of the other parameters, mentioned in Section 5.1, was not required for the same reasons mentioned above.

The first experiment consisted in a random selection and evaluation of 1000 posts, whose relevance had not been updated yet. These posts are collected at the time of its publication on Twitter, which means that their classification is only conditioned by the post text relevance. The results can be seen in chart, included in figure 5.4. These results show an average relevance of 0.5580 and a standard deviation of 0.0169, with a minimum value of 0.5175 and a maximum value of 0.6587. Looking at the chart, it can be seen that most of the selected news have a relevance contained in the range $[0,54, 0,58]$. Since at the time of publication, each news have 0 likes and 0 retweets, it is expected that the value for the relevance will significantly increase later on.

The second experiment was carried out in order to prove what was said in the previous para-

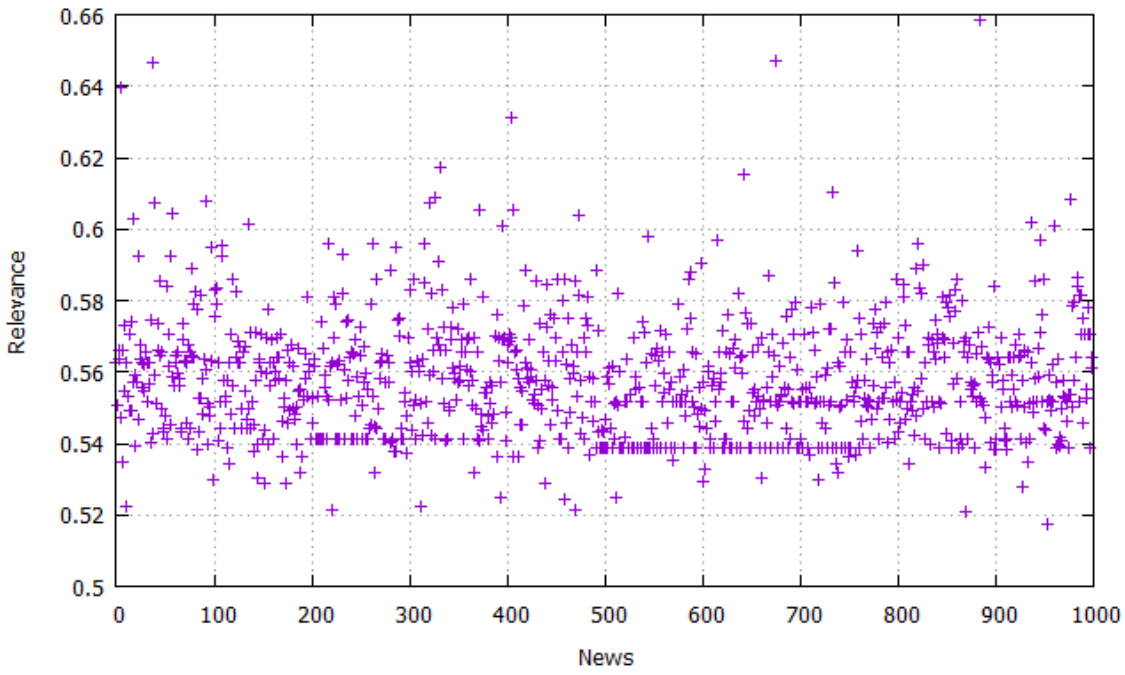

Figure 5.4: Relevance of posts that never were updated by the tool.

graph. This experiment included the random selection and evaluation of 1000 posts that had, at least, one like or one retweet since their publication. These posts have the same text relevance they had before being updated. However, the number of likes or retweets will increase their value to the system, leading to the increase of its final relevance. The results of these experiment are summarized in the chart of figure 5.5.

These results show an average relevance of 0.6473 and a standard deviation of 0.0681, with a minimum value of 0.5218 and a maximum value of 0.9490. From the analysis of the chart, it can be seen that most of the posts have a relevance contained in the interval $[0.55, 0.70]$. Compared to the chart in figure 5.4, it appears that the news average relevance increased approximately 0.1 and there is a greater number of posts with higher relevance values. This behavior motivated the assignment of the values 0.6 and 0.8 to ρ_{min} and ρ_{max}, respectively.

5.5 Evaluation of the Post Classification

After the definition of the ρ_{min} and ρ_{max} parameters, which regulate the classification of posts, it is necessary to verify the effectiveness of this classification. To assess that effectiveness, an experiment with human evaluators was performed, in which 50 posts with high relevance and 50 posts with moderate relevance were selected. For the assignment of a meaningful classification in this subjective analysis, two parameters were used: the post text relevance ($rev(post)$) and an estimate of the number of users in the general community that may be affected by the subject on this post ($nUsers$). These parameters were scored on a scale of 0 to 20. The classification

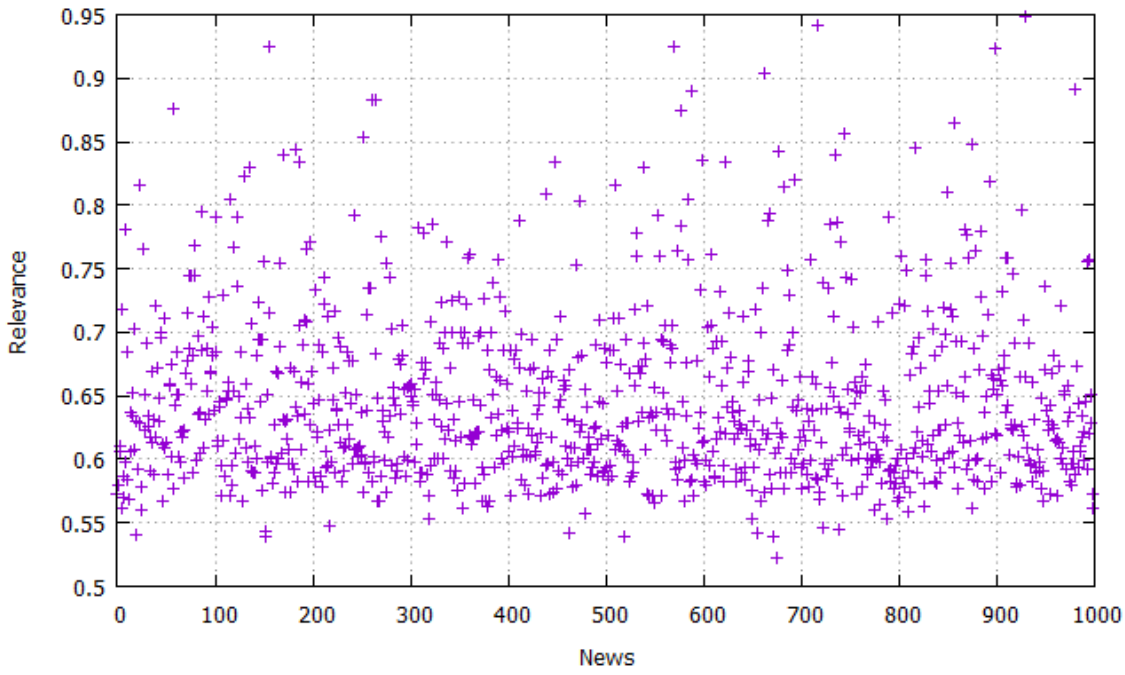

Figure 5.5: Relevance of posts that have had, at least, one like or one retweet since its publication.

of the post, assigned during human analysis, is obtained according to Equation 5.1:

$$Classification(post) = \frac{rev(post) + \frac{nUsers}{2}}{30}. \quad (5.1)$$

Notice that the equation gives gives more weight to the post text relevance, similarly to what happens in equation 3.5. The outputs of this equation are confined to the range 0 to 1, in order to establish a comparison with the results obtained in the classification of the post returned by the system.

The experiment assumes that the classification of a post performed by the system is good if the post gets a similar classification from the human evaluator. Table A.2, included in Annex A for the sake of the size of this explanation, shows all the scores provided by humans for 50 posts with high relevance. Table A.1, included in Annex A, contains a similar compilation of results, but for the 50 posts with moderate relevance.

Observing the results in both tables, it is clear that, in general, 42 posts were in fact considered to be relevant for the system. However, this efficiency is not accurate, since the equation used for human post classification is not strict. However, the human classification for most of the posts was very close to the classification assigned by the system, indicating that most of the posts have a significant degree of relevance. On the other hand, some results were very far from the classification assigned by the system. This is due to the fact that the post analysis performed by the system calculates the post text relevance and not the full news text. The post text is written by the tweeter, and contains, sometimes, relevant terms that do not fit

with the subject addressed in the full news text, which leads a post, that has been classified by the system as relevant, to be less interesting for a human. Nonetheless, the number of these contradictory occurrences is reduced. Furthermore, by performing a very conservative statistical t-test[1], over the evaluations made by humans, it is revealed that the obtained scores for the posts considered as highly relevant to the system (0.65296 ± 0.2) is significantly better (*p-value* $= 0.024321$) than the one obtained for the moderately relevant posts (0.53594 ± 0.29611), which leads to the conclusion that the method for classification of post is effective.

5.6 Limit for the Number of Entries in the Database Tables

The third experiment was performed to define limits for the entries in the database tables. To this end, three tests were performed (one for each table), in order to estimate the average number of tweeters and posts that are daily added in each one of the tables. These tests involve the predefinition of all the parameters mentioned in Section 5.1. The values they were assigned with are as follows: 0.1 to λ_{min}, 0.5 to λ_{max}, 0.6 to ρ_{min}, and 0.8 for ρ_{max}. The value of 10000 was the limit set for the number of entries of all tables at the beginning of the experiment.

The number of tweeters and news added to each table was collected in each test and for 5 different days. The results can be seen in Table 5.2, in Table 5.3 and in Table 5.4. From the

Date	Number of Users
2016-08-30	383
2016-08-31	391
2016-09-03	332
2016-09-05	329
2016-09-06	367

Table 5.2: Number of tweeters inserted in the `quarantine` table per day, for 5 different days.

analysis of the values in Table 5.2, it can be concluded that, on average, 360 users were added per day in the `quarantine` table. Using this value as a reference, the value 10000 was assigned to the parameter Q, which represents an approximation of the total number of tweeters that can be added to this table in one month.

Date	Number of Users
2016-08-30	75
2016-08-31	54
2016-09-03	64
2016-09-05	80
2016-09-06	88

Table 5.3: Number of tweeters inserted in the `inproduction` table per day, for 5 different days.

The values in Table 5.3 suggest that, on average, 72 tweeters were added per each day to the `inproduction` table. Therefore, an average of 2160 tweeters will be added to this table in each

[1]Test of significance for two unknown means and unknown standard deviations — two tails with heterogeneous variance.

month. Despite this value, the value 10000 was added to the parameter P, since no scalability or performance problems are expected with such number of entries.

Date	Number of News
2016-08-30	272
2016-08-31	253
2016-09-03	351
2016-09-05	357
2016-09-06	374

Table 5.4: Number of posts inserted in the `news` table per day, for 5 different days.

As can be concluded from the analysis of Table 5.4, on average, 321 news were added per day to the `news` table. Using this value as a reference, the value 10000 was also assigned to the parameter N, which represents an approximation of the total number of news that can be added to this table in one month.

5.7 Conclusions

This chapter is described the experiments conducted with the purpose of test the effectiveness of algorithms to classify tweeters and tweets. In some of the experiments, tweeters and tweets were randomly collected in an attempt to observe patterns that could allow tunning the system with predefined values for its execution, namely for the parameters that regulate the insertion of tweeters into the database or to the inproduction table, and to classify posts as relevant or not.

After having decided on the predefined values, a series of experiments involving human evaluators were discussed. In these experiments, some tweeters and tweets were selected, along with thee respective ratings assigned by the system. These rating were compared with the results of a subjective analysis. In order to do so, the evaluators also assigned some ratings, in order to make a quanlitative assessment of the system functioning. Although these quantitative results are not very strict, it can be seen that the human evaluation was very close to the evaluation provided by the system. During the subjective analysis, it was also possible to verify that the characteristics used by the system seem to capture what the evaluators observed. In the cases where this did not happened, it was concluded that the problem was linked to the fact that the terms in the tweets did not match the full text of the news, or the URL of the full article might have been corrupted.

Finally, some tests were performed to estimate values for the limits of the number entries for the database tables. These tests consisted in the continuous collection and storage of tweeters and tweets during a month. These results were conditioned by the maximum allowed calls to the Twitter API in each 15 minutes interval.

Chapter 6

Conclusions and Future Work

This chapter provides an overview of the main conclusions drawn from the research work developed during the course of this master's project and presents a few guidelines for further developments of this work. It is organized in two sections. Section 6.1 presents the main conclusion of this work. Section 6.2 discusses ideas on what might be improved or pursued in the future regarding this work.

6.1 Main Conclusions

The WWW is nowadays providing a huge volume of information to users in many forms. Every day, large amounts of information is published in various formats such as news, blogs, social networks, etc. Due to the dynamic nature of the web and the presence of unstructured semantics, it may be difficult for a user to find relevant and valid information matching a given topic of interest. The exponential increase in the amount of information also poses a challenge in many research areas like security and intelligence. In these communities, getting access to the latest news and topics has a direct impact on the preparation and response to threats such as zero-day attacks. It also allows users to be aware of the risks and faults present in the most diverse technologies, used regularly, and to increase the security procedures in their applications and software.

The challenge of effectively selecting relevant information contained on the web is the great motivation for the development of this project. Although there are already technologies and procedures for making the selection of information in an automated or semi-automated manner, most users still makes this selection manually. After a while, this task is repetitive and cumbersome, and sometimes prone to error. The use of OSINT systems for gathering information is thus of increasing value nowadays. These systems are designed to find, collect and correlate informational content available to the general public through social networks, blogs, newspapers, television, etc. These systems usually incorporate web crawlers and APIs when applied to online sources. In its most basic implementation, a web crawler is a program that browses the web automatically, downloading content from the visited pages and following the links therein to other pages. In the implementation of a web crawler, some procedures must be manually set in order to access and manipulate the desired web content. These procedures have to respect a set of policies, involving the selection of pages to visit, the frequency with which those pages are re-visited to check if some pages have changed, the definition of methods for prioritize the selection of the most relevant pages to visit, and the distribution of crawling processes by

various crawlers, in order to increase the crawling performance. On the other hand, an API consists in a set of routines and standards that allow the development of software applications that communicate with back-end services. APIs implement the procedures to collect and manipulate data from a specific software, and the policies that rule that process. Many social networks, such as Twitter and Facebook, offer powerful APIs for interacting with their system.

That said, the main objective of this master's project was to address the previously motivated problem, by implementing a dynamic OSINT system sourcing from social networks. The main idea was for the system to implement procedures to automatically deal with the social network and online sources of information and to effectively select and rate news from the Information Security area. The developed system was focused on gathering information from the social network Twitter, through the use of the Twitter API. For this purpose, two algorithms were developed: one to handle the selection and classification of users from this social network (also known as twitters), and another one for the selection and classification of posts related with the aforementioned area and published by those tweeters. For the algorithm concerning the tweeters selection and classification, a function that includes parameters such as the average relevance of the tweeter posts, the associated publication frequency for the last 30 days and the number of followers, was designed and studied in the scope of this work. From the study of various parameters associated with a tweeter, these three were considered the most relevant for the classification. The results of this function are normalized, in order to produce values within a scale that allows assigning a score to a tweeter. These scores are used by the developed system to decide on whether or not to include twitters in the database for further analysis or for the production of news in the main mural. For the post selection and classification algorithm, a function that includes parameters such as the relevance of the text of the post, its likes and retweets, was also designed and studied. The outputs of this function are also normalized, in order to produce values within a scale that allows assigning scores to posts, which the system considers to be their *relevance* (0 for low relevance, 1 for moderate relevance, and 2 for high relevance). The proposed algorithms and underlying details (such as the classification functions) are the main contributions of this work.

The mentioned algorithms have been implemented in a system composed of four main components: an application developed in Java, a database, a web platform and a virtual machine. The Java application is responsible for the integration of the two algorithms and the definition of some of their associated parameters. There parameters regulate the assignments of ratings to tweeters and their posts. The database is responsible for storing these tweeters and their posts, and all relevant information associated with them. The web platform is the system interface and it is responsible for presenting the most relevant posts to the end users. The virtual machine acts as the system server and it is responsible for the inclusion of the three components mentioned above, so that the implemented system may be accessed by any user with an

Internet connection.

Several experiments were designed and performed to evaluate the algorithms, namely in terms of the classifications they were producing. Some of these experiments resorted to subjective analysis, since no external tools to compare results were available. Several persons were asked to analyze and rate randomly chosen tweeters and posts. The obtained results provide a sense of the system efficiency. During this analysis, it was found that most of the tweeters have the characteristics stipulated for their admission to the production of news for the end-user mural. Thus, it can be concluded that the adopted function used to calculate the tweeters importance and the established parameters for the classification thereof were effectively capturing the meaning of relevancy of users. It was also noticed that the system is using the characteristics that effectively capture the relevancy of posts.

There were some cases where the human evaluator did not consider a given post to be relevant at all, contrarily to the system. This was probably related to how the system evaluates the post text. In most cases, the post text (following a URL) contained relevant terms that did not fit with the full text of the twitter post. In other cases, it was found that the posts text lacked the original news URL or that this URL was corrupted, not allowing a subjective analysis of the news. However, the number of occurrences for these cases was low. Additional tests were performed to assess if the limits of 10000 entries per table was enough. In these tests, the average of tweeters and posts added to the tables per day was calculated, so as to extrapolate the values for 1 month of system activity. It was concluded that these limits were suitable.

Due to all the arguments presented in this section, it can be concluded that the proposed objectives for this project have been fulfilled and that the implemented system demonstrates the potential of the adopted approach and solution.

6.2 Future Work

Although the developed system meets the requirements for which it was designed, it would be interesting to improve some important aspects in the future. One of those aspects concerns the metrics and procedures used for classification of tweeters and posts. For example, one may think of more parameters to include in the designed functions, whose impact would need to be dully analyzed, such as the presence of *hashtags* or the amount of information they convey. Some of these *hashtags* are quite relevant and they are used frequently as a reference in searching for news. Their impact on the post relevance calculation would need to be assessed. Moreover, the inclusion of backup methods to prevent the inclusion of posts with corrupted URLs, or to immediately compare the text of the news in the target URL with the one of the tweet could be valuable additions.

The parameters used in the scoring procedures of the subjective analysis could also be different, and the number of experiments be more diversified. For example, due to time limitations, the experiments concerning classification of posts was performed only for a subset of posts that the system considers to be highly or moderately relevant, but they can be extended to include non-relevant posts also.

The web page working as mural would benefit from a facelift from a designer. URLs for each news should be highlighted and an advanced search form to narrow down the displayed news would also be useful features. This improved search form would allow, for example, the selection of the displayed news by date, source or other special attributes, in addition to the list of words already available. Last but not least, the system would be more complete with the implementation of methods for the collection and classification of news from other sources of information, in addition to Twitter, but this will require substantial effort.

Bibliography

[Ama14] Amazon Web Services. Aws free tier [online]. 2014. Available from: https://aws.amazon.com/free/?nc1=h_ls [cited 08 October 2016]. 35

[Apa16] Apache Friends. Xampp apache + mariadb + php + perl [online]. 2016. Available from: https://www.apachefriends.org/index.html [cited 10 September 2016]. 16

[Car04] Carlos Castillo. Effective web crawling [online]. 2004. Available from: http://chato.cl/papers/crawling_thesis/effective_web_crawling.pdf [cited 11 November 2015]. 9

[CB11] David Horby Clive Best. Iq – a web mining tool. *European Intelligence and Security Informatics Conference*, 2011. 16

[CJ00] Hector Garcia-Molina Cho Junghoo. Synchronizing a database to improve freshness. In *Proceedings of the 2000 ACM SIGMOD International Conference on Management of Data*, pages 117–128. ACM New York, NY, USA, 2000. 10

[Cor11] João Paulo da Costa Cordeiro. The HultigLib: "nuggets" for text processing in java [online]. 2011. Version 1.1. *Centre For Human Language Technology and Bioinformatics* (HULTIG). Available from: https://web.archive.org/web/20160405113417/http://www.di.ubi.pt/~jpaulo/hultiglib/ [cited 2016-04-05]. 20, 29, 31

[Cot04] Viv Cothey. Web-crawling reliability. *Journal of the American Society for Information Science and Technology*, (55 (14)):1228--1238, 2004. 10

[CR14] Bharath Yadla Rama Kanneganti Kiran Somalwar Charu Rudrakshi, Amit Varshney. Apification core building block of the digital enterprise. *DIGITAL SYSTEMS INTEGRATION*, 2014. 12

[Fel15] Cordeiro J. Felipe, B. Deteção automática de plágio em dois atos. ISBN: 978-989-654-248-1. Simpósio de Informática – INForum 2015, 2015. 25

[JB14] Gene Cahill Soumya Simanta Edwin Morris Jeff Boleng, Marc Novakouski. Fusing open source intelligence and handheld situational awareness. *IEEE Military Communications Conference*, 2014. 15

[Kar12a] Karthik Poojary. The foca [online]. 2012. Available from: http://www.computerweekly.com/photostory/2240160131/Nine-must-have-OSINT-tools/6/5-The-FOCA [cited 08 October 2016]. 1

[Kar12b] Karthik Poojary. Ghdb [online]. 2012. Available from: http://www.computerweekly.com/photostory/2240160118/Nine-must-have-OSINT-tools/5/4-GHDB [cited 08 October 2016]. 1

[Kar12c] Karthik Poojary. Metagoofil [online]. 2012. Available from: http://www.computerweekly.com/photostory/2240160112/Nine-must-have-OSINT-tools/4/3-Metagoofil [cited 08 October 2016]. 1

[Mar14] Martin. Introducing winscp [online]. 2014. Available from: https://winscp.net/eng/docs/introduction [cited 10 September 2016]. 17

[Mor05] Morton, T., Kottmann, J., Baldridge, J., Bierner, G. Opennlp: A java-based nlp toolkit [online]. 2005. Available from: http://opennlp.sourceforge.net [cited 16 June 2016]. 29

[MS14] Daniël T.H. Worm Henri Bouma Martijn Spitters, Pieter T. Eendebak. Threat detection in tweets with trigger patterns and contextual cues. *IEEE Joint Intelligence and Security Informatics Conference*, 2014. 15

[MyS16] MySQL. Mysql connector/j 5.1 developer guide [online]. 2016. Available from: http://dev.mysql.com/doc/connector-j/5.1/en/ [cited 16 June 2016]. 29

[Nou13] Pritzkau A. Schade U. Noubours, S. Collaborative text-annotation resource for disease-centered relation extraction from biomedical text. *In proceedings of Military Communications and Information Systems*, pages 1–7, 2013. 4

[OAu16] OAuth. User authentication with oauth 2.0 [online]. 2016. Available from: https://oauth.net/articles/authentication/ [cited 31 August 2016]. 14

[Ora13] Oracle. Netbeans developing applications with netbeans ide [online]. 2013. Available from: https://docs.oracle.com/cd/E40938_01/doc.74/e40142/gs_nbeans.htm [cited 10 September 2016]. 16

[OSI06] National defense authorization act for fical year 2006 [online]. 2006. Available from: https://www.gpo.gov/fdsys/pkg/PLAW-109publ163/html/PLAW-109publ163.htm [cited 9 October 2016]. 1

[PuT16] PuTTY. Introduction to putty [online]. 2016. Available from: https://the.earth.li/~sgtatham/putty/0.60/htmldoc/Chapter1.html [cited 10 September 2016]. 17

[SD11] K. Thirugnana Sambanthan S.S. Dhenakaran. Web crawler - an overview. *International Journal of Computer Science and Communication*, 2(1):265–267, 2011. 8

[Sof16] Softpedia. Putty key generator (puttygen) [online]. 2016. Available from: http://www.softpedia.com/get/Network-Tools/Misc-Networking-Tools/PuTTY-Key-Generator.shtml [cited 10 September 2016]. 17

[SS11] J.P.Gupta Shruti Sharma, A.K.Sharma. A novel architecture of a parallel web crawler. *International Journal of Computer Applications*, 14(4), 2011. xxv, 11

[Ste07] R Steele. Open source intelligence. *Handbook of Intelligence Studies*, 42(5):129–147, 2007. 1

[Twi16a] Twitter. Developer agreement and policy [online]. 2016. Available from: https://dev.twitter.com/overview/terms/agreement-and-policy [cited 16 June 2016]. 30

[Twi16b] Twitter. Rate limits: Chart [online]. 2016. Available from: https://dev.twitter.com/rest/public/rate-limits [cited 31 August 2016]. 15, 33

[Vyb16] VybeOn. Api [online]. 2016. Available from: http://vybeon.com/Skillz/Technology/API [cited 31 August 2016]. 12

[Wan10] Alex Hai Wang. Don't follow me spam detection in twitter. *International Conference on Security and Cryptography (SECRYPT)*, 2010. 15

[Yam07] Yamamoto, Y. Java library for the twitter api [online]. 2007. Available from: http://www.twitter4j.org/ [cited 16 June 2016]. 29

[ZL14] Abbasi M. Zafarani, R. and H. Liu. Social media mining. ISBN: 9781107018853. Cambridge University Press, 2014. 24

Appendix A

Human Classification of Posts

The tables contained in this annex summarize the results obtained during the subjective analysis to the performance of the algorithm for classification of posts. Table A.1 concerns a subset of 50 posts that the system considered to be of moderate relevancy, while Table A.2 compiles similar results for 50 posts that the system considered to be of high relevancy. These tables are mentioned and discussed in chapter 5.

News Number	Likes	Retweets	Classification (application)	Post Relevancy (human analysis)	Affected Community (human analysis)	Classification (human analysis)
N1	0	2	1	13	14	0.667
N2	0	392	1	12	18	0.700
N3	0	7	1	18	16	0.867
N4	0	21	1	14	17	0.750
N5	0	8	1	14	18	0.767
N6	0	5	1	15	14	0.733
N7	0	11	1	10	11	0.517
N8	0	39	1	17	14	0.800
N9	1	3	1	12	13	0.617
N10	0	7	1	14	19	0.783
N11	39	25	1	1	19	0.350
N12	3	8	1	16	18	0.833
N13	0	2	1	17	16	0.833
N14	0	8	1	14	12	0.650
N15	0	3	1	14	16	0.667
N16	0	2	1	2	5	0.150
N17	1	3	1	16	14	0.767
N18	0	2	1	16	15	0.783
N19	0	2	1	12	13	0.650
N20	0	2	1	0	20	0.333
N21	0	15	1	16	12	0.733
N22	1	1	1	16	18	0.833
N23	0	14	1	1	3	0.083
N24	0	3	1	10	5	0.417
N25	0	8	1	15	17	0.783
N26	0	2	1	2	1	0.083
N27	0	4	1	17	10	0.733
N28	0	4	1	3	3	0.150
N29	0	2	1	2	2	0.100
N30	0	29	1	0	0	0.000
N31	2	2	1	15	12	0.700
N32	0	2	1	16	15	0.783
N33	0	2	1	5	2	0.200
N34	0	25	1	15	17	0.783
N35	0	2	1	18	18	0.900
N36	2	3	1	0	2	0.033
N37	0	2	1	18	17	0.883
N38	0	9	1	13	15	0.683
N39	1	1	1	0	0	0.000
N40	2	3	1	10	6	0.433
N41	1	4	1	0	0	0.000
N42	0	2	1	10	12	0.533
N43	3	1	1	0	0	0.000
N44	1	1	1	15	13	0.717
N45	0	8	1	16	15	0.783
N46	0	4	1	1	18	0.300
N47	0	1	1	11	14	0.600
N48	0	2	1	13	11	0.617
N49	0	2	1	14	14	0.700
N50	0	3	1	0	1	0.017

Table A.1: Results for the classification of posts with metrics obtained by the system and scores from the human evaluators (subjective analysis). The system considered these posts to be of moderate relevance.

News Number	Likes	Retweets	Classification (application)	Post Relevancy (human analysis)	Affected Community (human analysis)	Classification (human analysis)
N1	4	11	2	17	18	0.867
N2	9	13	2	13	10	0.600
N3	0	4	2	6	17	0.483
N4	0	42	2	16	16	0.800
N5	7	6	2	7	19	0.550
N6	39	25	2	16	18	0.833
N7	0	29	2	13	15	0.683
N8	5	10	2	14	14	0.700
N9	3	8	2	12	10	0.567
N10	0	34	2	19	18	0.933
N11	6	12	2	17	17	0.638
N12	10	6	2	16	18	0.833
N13	0	93	2	7	10	0.400
N14	0	146	2	1	19	0.35
N15	4	3	2	17	15	0.613
N16	0	85	2	15	16	0.767
N17	2	4	2	16	19	0.850
N18	0	131	2	5	13	0.383
N19	2	6	2	18	18	0.900
N20	4	14	2	17	14	0.800
N21	0	84	2	10	9	0.483
N22	6	16	2	18	16	0.800
N23	15	32	2	14	12	0.667
N24	0	70	2	16	15	0.783
N25	0	98	2	10	11	0.517
N26	0	23	2	16	16	0.800
N27	0	46	2	13	19	0.750
N28	5	3	2	16	12	0.733
N29	0	86	2	17	18	0.867
N30	0	7	2	5	2	0.200
N31	0	104	2	10	6	0.433
N32	0	3	2	0	20	0.333
N33	0	117	2	16	18	0.833
N34	6	6	2	15	13	0.717
N35	0	111	2	17	16	0.833
N36	0	64	2	15	14	0.733
N37	51	45	2	16	16	0.800
N38	0	38	2	15	13	0.717
N39	0	74	2	18	18	0.900
N40	4	7	2	10	6	0.433
N41	14	11	2	15	18	0.800
N42	0	11	2	3	3	0.150
N43	0	58	2	7	15	0.483
N44	0	72	2	2	7	0.183
N45	22	22	2	18	16	0.800
N46	22	10	2	14	17	0.750
N47	0	144	2	10	5	0.417
N48	0	139	2	15	18	0.800
N49	0	61	2	14	14	0.7
N50	0	125	2	12	17	0.683

Table A.2: Results for the classification of posts with metrics obtained by the system and scores from the human evaluators (subjective analysis). The system considered these posts to be of high relevance.