E-MAIL HEADER INFORMATION

Joshua Clark

Fourth Year Project Report for the Final Honour
School of Computer Science

May 2016

Abstract

After extensive public education, fewer people are now clicking on links in e-mails that are disguised as phishing attacks, though the threat still remains, and considerable amounts of work has gone into exploring the demographics most likely to be targeted. As the number of technically literate people grows, this sort of attack is increasingly unlikely to be successful. Therefore, malicious entities are more likely to attempt to attack people based on the information leaked in their emails, and more specifically, the header, which most people are less likely to have some degree of control over.

The risks are not just limited to individual users, and at a corporate level, the risks posed by leaking information through e-mails could be even greater: e-mail headers can reveal the internal network structure of a company's computer systems as well as the different pieces of software that are running inside the system. Extracting the social information could be of great value for executing a phishing atack, however, there is also value in determining the specific weaknesses in a system. This can be aided through the use of vulnerability databases.

This report discusses the existing research into the information leaked by e-mail headers and presents a tool to extract such information.

Acknowledgements

I want to thank my supervisor, Dr Jason R.C. Nurse for his assistance throughout the year; my tutor, Professor Peter Jeavons, for his unfailing help and support throughout my time at Oxford. I would like to acknowledge the support from my family and partner, Agata, for encouraging me.

Contents

List of Tables

List of Figures

List of Algorithms

1 Introduction

1.1 Motivation

E-mail systems are now so integrated into our modern lives that we struggle to cope without them. E-mails ubiquity is also one of its largest weaknesses, a fact recognised very early on. The first spam email was sent in 1978, as documented by Templeton n.d. After spam came phishing, first described by Felix and Hauck 1987, with the first-real world use being against the customers of America Online, an ISP. However, this still relies on the targets providing their data for malicious purposes. One of the first e-mail viruses to spread was the Happy99 virus, which, other than propagating itself, had no other effect on infected systems. Later viruses would target credit-card and banking information. However, all of these techniques rely on the malicious email being received and its contents being opened. There are fewer instances recorded, however, of the information flow being sent the other way. A more subtle attack will focus on the information being sent from a legitimate user to an attacker. It is easy enough for an individual to read an e-mail header and identify interesting elements, however, on a large scale, this quickly becomes more difficult.

1.2 Aims and Objectives

This project aims to develop a tool that can be used as described above to automatically extract information from e-mail headers and analyse its results to display the personal information continaed within an e-mail's header, as well as information about the software configurations that may be found on a user's computer, or the servers used to send their e-mail.

The program would be expected to satisfy the following minimal requirements in order for it to be considered successful:

Accuracy — any information produced by the parser should be reflective of the input e-mail

Representation — the produced visualisation should be intuitive to read: each element should be presented separately from the others, and clearly labelled.

Portability — the visual output produced by the program should be available to the user in a variety of formats.

Interactivity — the program should produce sensible warnings when an e-mail that is not possible to parse has been entered.

1.3 Typical Use

On starting the application, the user will provide an e-mail that they wish to have analysed. This will then be parsed, and some relevant information presented in a table.

Lastly, an option is available to view the information about security vulnerabilities in a separate webpage, forming the main output of the program.

Email Header Information		
Sender Information (Name, originating domain)	Sender Software	Sender Usernames (Presented as a list with likely organisation)
Graphical representation of devices used to deliver the e-mail		
List of derived information including found software and similar information		
Histograms for vulnerability scores, separated by product		
Searchable and filterable table of discovered CVEs		

Table 1.1: Format of presented data found in e-mail header

The resultant webpage will be structured as in table 1.1. It will then be possible for the user to click on the representations of the devices to find out more information. It will also be possible to search within the vulnerability list to find more information, as well as filter by impact and availability details.

1.4 Document Conventions

By convention, when class diagrams are used, ovals will represent traits/abstract classes, with rectangles representing concrete implementations. Objects are indicated using bold lines in the diagram, and are similar in behaviour to statically declared objects in many languages.

Whenever `this font face` is used, the text is referring to either some implementation level object or class, or text entered into the application or used for testing.

1.5 Structure

Chapter 2 begins by discussing the existing research on the subject as well as existing publicly-available tools to analyse headers. I then use these as a basis to discuss features that would be expected to appear in a header analyser looking for leaked information and vulnerabilities.

The implementation's high-level structure and details will be discussed in Chapter 3, and algorithms presented in pseudo-code where necessary. A full listing will be presented at the end of this document in an appendix. The results of the analysis of the headers will be discussed in Chapter 4, beginning with the methodology used, and presenting a number of results. Finally, Chapter 5 will discuss my conclusions and areas of further improvement.

2 Literature Review

In this chapter, we will discuss the nature of existing threats to data, and the ongoing research in this area. We will then consider the specific threats posed by e-mail.

2.1 General Data Leakage

The importance of data leakage is gaining more importance as the amount of information stored about entities increases, and the risks are being considered more carefully. From the obvious ramifications for businesses discussed in Papadimitriou and Garcia-Molina 2011: the loss of trust and legal action resulting from the discovery of leaking data, to the more personal issues discussed in Irani et al. 2011: the possibility of using the discovered data to discover passwords or to physically identify them. 53% of Americans can be uniquely identified by their birthdate, gender and location (city/town), with the number jumping 87% when using birthdate, gender and zip code.

2.1.1 Personal Data Leakage

From a persona perspective, there are a number of risks. There are a significant number of social networks available, with an estimated 1.65 billion monthly active users, with a significantly higher proportion used in developed countries. Irani et al. 2011 showed the rate at which the information gathered from social networks can be used to uniquely identify anindividual, with only 9 sites required before there's approximately a 70% chance that both a person's hometown and name could be recovered. A similar number of sites can give an aggregate normalised attribute leakage of 1, where the leakage is defined as

$$\Psi(F_a, P) = \frac{\sum_{f_a \in F_a} \phi(f_a, P)}{|F_a|} \text{ where } \phi(f_a, P) = [f_a \in P]$$

2.1.2 Corporate Data Leakage

When companies receive user data, they often have a legal obligation to ensure that the data is protected and treated as confidential and sensitive. When this trust is broken, there are often severe consequences, both from regulators and consumers moving their business to competitors. Squicciarini, Sundareswaran, and Lin 2010 considers one way that data may be leaked, despite care being taken to ensure that it is properly encrypted and stored. By failing to protect against the indices for databases being stored insecurely, customer data may be leaked. Order-preserving encryption schemes, as decribed in Agrawal et al. 2004, is one way of solving this problem, to an extent.

2.2 Data Leakage from E-Mails

2.2.1 E-Mail Headers

All e-mails include additional information about the sender and receiver, some of which is used by an e-mail client in order to display more information about the message that is currently being viewed, such as its original sender, reply-to addresses and the time it was sent. Additional fields allow senders to authenticate themselves using public-key methods.

The format of e-mail headers was first defined in RFC 822, and further refined in subsequent RFCs. The standard for e-mails was then formalised precisely in RFC 5322.

2.2.2 Example Header and Pertinent Data

In the example below, and text highlighted with red, like so is information about the receiver. Information about the sender, their hardware or software is highlighted in green, like so; and information gathered about intervening devices is highlighted in blue, like so.

```
Delivered-To: joshuaclark94@gmail.com
Received: by 10.25.150.146 with SMTP id y140csp543137lfd;
       Sat, 6 Feb 2016 08:49:56 -0800 (PST)
X-Received: by 10.112.12.2 with SMTP id u2mr8302831lbb.145.1454777396580;
       Sat, 06 Feb 2016 08:49:56 -0800 (PST)
Return-Path: <agatabor@poczta.onet.pl>
Received: from smtpo75.poczta.onet.pl (smtpo75.poczta.onet.pl. [141.105.16.25])
       by mx.google.com with ESMTPS id o199si12255636lfb.94.2016.02.06.08.49.56
       for <joshuaclark94@gmail.com>
       (version=TLS1_2 cipher=ECDHE-RSA-AES128-GCM-SHA256 bits=128/128);
       Sat, 06 Feb 2016 08:49:56 -0800 (PST)
Received-SPF: pass (google.com: domain of agatabor@poczta.onet.pl designates
              141.105.16.25 as permitted sender) client-ip=141.105.16.25;
Authentication-Results: mx.google.com;
spf=pass (google.com: domain of agatabor@poczta.onet.pl designates 141.105.16.25
              as permitted sender) smtp.mailfrom=agatabor@poczta.onet.pl
Received: from [10.26.196.156] (client-8-32.eduroam.oxuni.org.uk [192.76.8.32])
(Authenticated sender: agatabor@poczta.onet.pl)
by smtp.poczta.onet.pl (Onet) with ESMTPA id 3pyKNH4ffyzT6tkv8
for <joshuaclark94@gmail.com>; Sat, 6 Feb 2016 17:49:50 +0100 (CET)
Date: Sat, 06 Feb 2016 16:49:07 +0000
Subject: Test e-mail
Message-ID: <j66i9tkyhy3l4v77erlw1gne.1454777347191@email.android.com>
Importance: normal
From: Agata <agatabor@poczta.onet.pl>
To: Joshua Clark <joshuaclark94@gmail.com>
MIME-Version: 1.0
Content-Type: multipart/alternative;
       boundary="--_com.android.email_1892258509098440"

----_com.android.email_1892258509098440
```

```
Content-Type: text/plain; charset=utf-8
Content-Transfer-Encoding: base64

CgoKClNlbnQgZnJvbSBteSBTYW1zdW5nIEdhbGF4eSBzbWFydHBob25lLg==

----_com.android.email_1892258509098440
Content-Type: text/html; charset=utf-8
Content-Transfer-Encoding: base64

PGh0bWw+PGhlYWQ+PG1ldGEgaHR0cC1lcXVpdj0iQ29udGVudC1UeXBlIiBjb250ZW50PSJ0ZXh0
L2h0bWw7IGNoYXJzZXQ9VVRGLTgiPjwvaGVhZD48Ym9keT48ZGl2IHN0eWxlPSJ3b3JkLWJyZWFr
O2tlZXAtYWxsOyI+PGJyPjxicj48YnI+PGJyPlNlbnQgZnJvbSBteSBTYW1zdW5nIEdhbGF4eSBz
bWFydHBob25lLjxicj48L2Rpdj48L2JvZHk+PC9odG1sPg==

----_com.android.email_1892258509098440--
```

The particularly interesting portions of the e-mail header include the IP adresses of the various servers the message has travelled through, allowing their approximate location to be determined. Additionally, the information on the protocol being used and the software being run allows for anyone with access to mail headers to find more information about the attacks a device and its software may be vulnerable to.

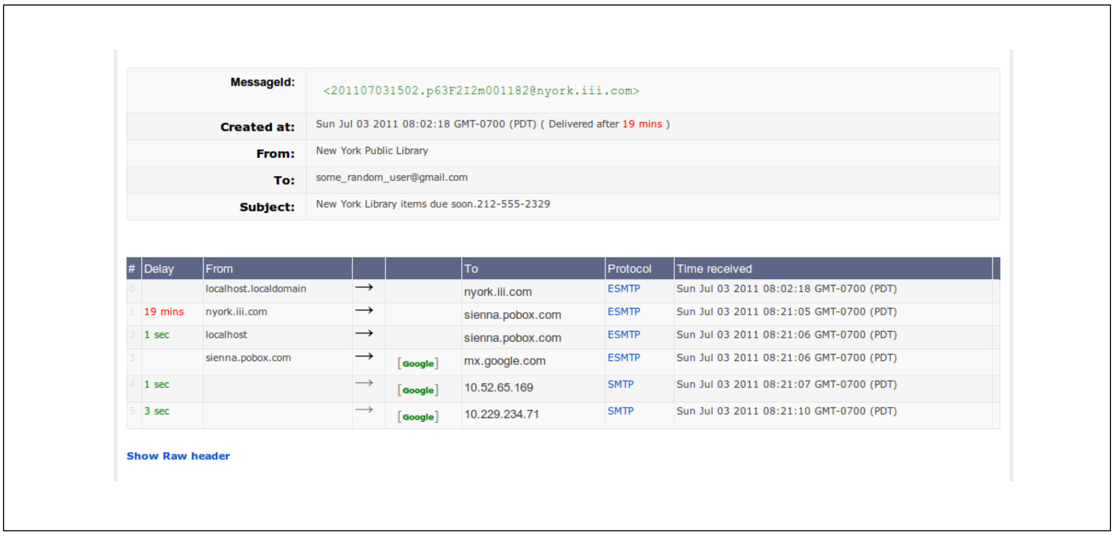
Further examples from headers that may be particularly interesting include the following examples:

2.2.3 Existing Research

In Nurse et al. 2015, the idea of using the information available in an email header was mooted, turning the previously standard threat of malware and phishing contained in received e-mails on its head, and instead presenting the threat in outgoing emails, and the personally identifying information (PII) contained therein. Many emails leaked information about employers, e-mail services and applications used, and IP address. Initial examination of a variety of e-mail headers found within my own inbox also revealed a plethora of information, including phone carriers, preferred languages, and system usernames. It is conceivable therefore, that it is possible to automate at least part of this, and present the information that can be extracted, in a white-hat tool to allow people to audit the information that they are revealing. The obvious malicious use-case involves using such information as part of a spear-phishing exercise.

An alternative vulnerability presents itself in the information about systems that may be revealed. Many email clients embed identifying information, and there are multiple databases available to allow specific threats to be identified. This could allow a malicious entity to compromise the security of a target machine, and gain access to the data stored on that machine and available on any connected network devices. Work started in Joshi, Lal, and Finin 2013 discusses the need to aggregate data about vulnerabilities from multiple sources to present a more complete and coherent picture, which is also likely to then contain more accurate data.

Al-zarouni 2004 presents an alternative set of results, describing how an individual can seek to protect themselves against malicious e-mails, using the contents of e-mail headers. Various discrepancies between forged e-mail addresses and legitimate messages are described.



MessageId:	<201107031502.p63F2I2m001182@nyork.iii.com>
Created at:	Sun Jul 03 2011 08:02:18 GMT-0700 (PDT) (Delivered after 19 mins)
From:	New York Public Library
To:	some_random_user@gmail.com
Subject:	New York Library items due soon.212-555-2329

#	Delay	From			To	Protocol	Time received
0		localhost.localdomain	→		nyork.iii.com	ESMTP	Sun Jul 03 2011 08:02:18 GMT-0700 (PDT)
1	19 mins	nyork.iii.com	→		sienna.pobox.com	ESMTP	Sun Jul 03 2011 08:21:05 GMT-0700 (PDT)
2	1 sec	localhost	→		sienna.pobox.com	ESMTP	Sun Jul 03 2011 08:21:06 GMT-0700 (PDT)
3		sienna.pobox.com	→	[Google]	mx.google.com	ESMTP	Sun Jul 03 2011 08:21:06 GMT-0700 (PDT)
4	1 sec		→	[Google]	10.52.65.169	SMTP	Sun Jul 03 2011 08:21:07 GMT-0700 (PDT)
5	3 sec		→	[Google]	10.229.234.71	SMTP	Sun Jul 03 2011 08:21:10 GMT-0700 (PDT)

Figure 2.1: Google Apps Toolbox E-mail header output

2.3 Existing Tools

Several tools already exist online to display the information that is found in e-mail headers. Tools from Microsoft and Google exist to analyse the contents of e-mail headers. These tools clearly display the information displayed in the header, showing the key-value pairs, and the set of servers the message transferred through and the protocols used.

2.3.1 Google

The Google Apps Toolbox features an e-mail header analyser[1]. An example of the output of the utility is found in Figure 2.3.1.

One of the most useful features from the Google Apps Toolbox is the information provided about the servers the message travelled through. This tool shows the details of the time taken for each hop, and the protocol used.

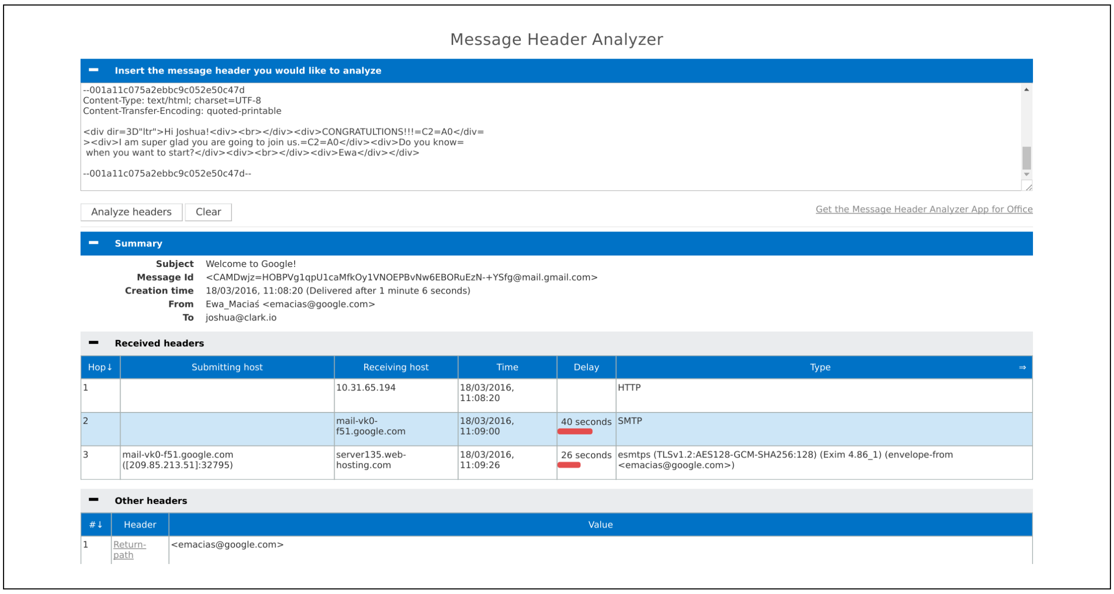
2.3.2 Microsoft

The Microsoft Message Header Analyzer[2] and showing sample results in Figure 2.3.2 is of a similar nature to the Google tool, discussed in Subsection 2.3.1.

In addition to the information presented by Google, this tool also produces a set of "Other Headers", highlighting fields that may be of interest. However, little additional context is provided as to their relevance.

[1] Found at `https://toolbox.googleapps.com/apps/messageheader/`

[2] Fount at `https://testconnectivity.microsoft.com/MHA/Pages/mha.aspx`


Message Header Analyzer
Insert the message header you would like to analyze
Analyze headers
Clear
Get the Message Header Analyzer App for Office
Summary
Received headers
Other headers

```
--001a11c075a2ebbc9c052e50c47d
Content-Type: text/html; charset=UTF-8
Content-Transfer-Encoding: quoted-printable

<div dir=3D"ltr">Hi Joshua!<div><br></div><div>CONGRATULTIONS!!!=C2=A0</div=
><div>I am super glad you are going to join us.=C2=A0</div><div>Do you know=
 when you want to start?</div><div><br></div><div>Ewa</div></div>

--001a11c075a2ebbc9c052e50c47d--
```

Subject Welcome to Google!
Message Id <CAMDwjz=HOBPVg1qpU1caMfkOy1VNOEPBvNw6EBORuEzN-+YSfg@mail.gmail.com>
Creation time 18/03/2016, 11:08:20 (Delivered after 1 minute 6 seconds)
From Ewa_Maciaś <emacias@google.com>
To joshua@clark.io

Hop↓	Submitting host	Receiving host	Time	Delay	Type ⇒
1		10.31.65.194	18/03/2016, 11:08:20		HTTP
2		mail-vk0-f51.google.com	18/03/2016, 11:09:00	40 seconds	SMTP
3	mail-vk0-f51.google.com ([209.85.213.51]:32795)	server135.web-hosting.com	18/03/2016, 11:09:26	26 seconds	esmtps (TLSv1.2:AES128-GCM-SHA256:128) (Exim 4.86_1) (envelope-from <emacias@google.com>)

#↓	Header	Value
1	Return-path	<emacias@google.com>

Figure 2.2: Microsoft E-mail header output

2.4 Vulnerabilities

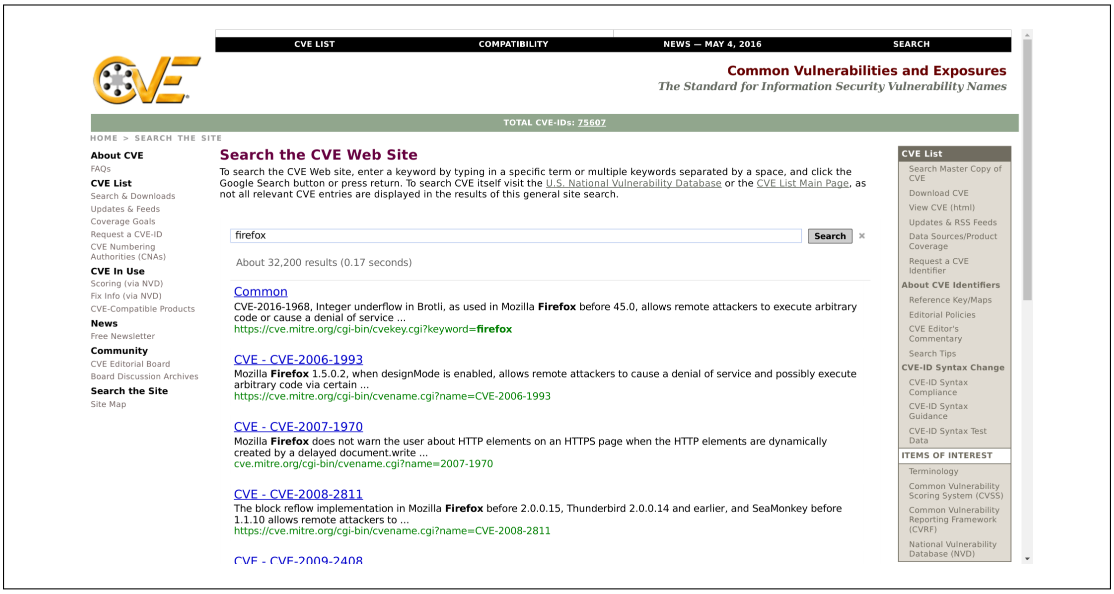
> Beware of bugs in the above code; I have only proved it correct, not tried it.
>
> Donald Knuth

Problems in software are nothing new, and seeking to exploit these issues is almost as old. As the security implications behind flawed software became more widely recognised, reducing their impact wherever possible became the next most important step. The MITRE Corporation operates the National Cybersecurity Federally Funded Research and Development Centre, which exists to maintain a database of these vulnerabilities, which are referred to as Common Vulnerabilities and Exposures (CVE).

2.4.1 CVE Mitre Lookup

There are a number of tools to look up CVEs[3] and showing sample results in figure 2.4.1. There are a number of limitations to the results returned by the CVE Mitre tool. Firstly, little context is returned: information about scores, the impact and access information are omitted, for example. Additionally, the process of finding relevant vulnerabilities is further slowed down by the necessity to search for specific terms one at a time. Additionally, automated tools exist at a consumer and enterprise level that will automatically scan a computer or network to detect installed software configurations and show the results.

[3] Fount at `https://www.cve.mitre.org/find/index.html`



Search the CVE Web Site

To search the CVE Web site, enter a keyword by typing in a specific term or multiple keywords separated by a space, and click the Google Search button or press return. To search CVE itself visit the U.S. National Vulnerability Database or the CVE List Main Page, as not all relevant CVE entries are displayed in the results of this general site search.

firefox

About 32,200 results (0.17 seconds)

Common
CVE-2016-1968, Integer underflow in Brotli, as used in Mozilla **Firefox** before 45.0, allows remote attackers to execute arbitrary code or cause a denial of service ...
https://cve.mitre.org/cgi-bin/cvekey.cgi?keyword=**firefox**

CVE - CVE-2006-1993
Mozilla **Firefox** 1.5.0.2, when designMode is enabled, allows remote attackers to cause a denial of service and possibly execute arbitrary code via certain ...
https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2006-1993

CVE - CVE-2007-1970
Mozilla **Firefox** does not warn the user about HTTP elements on an HTTPS page when the HTTP elements are dynamically created by a delayed document.write ...
cve.mitre.org/cgi-bin/cvename.cgi?name=2007-1970

CVE - CVE-2008-2811
The block reflow implementation in Mozilla **Firefox** before 2.0.0.15, Thunderbird 2.0.0.14 and earlier, and SeaMonkey before 1.1.10 allows remote attackers to ...
https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2008-2811

CVE - CVE-2009-2408

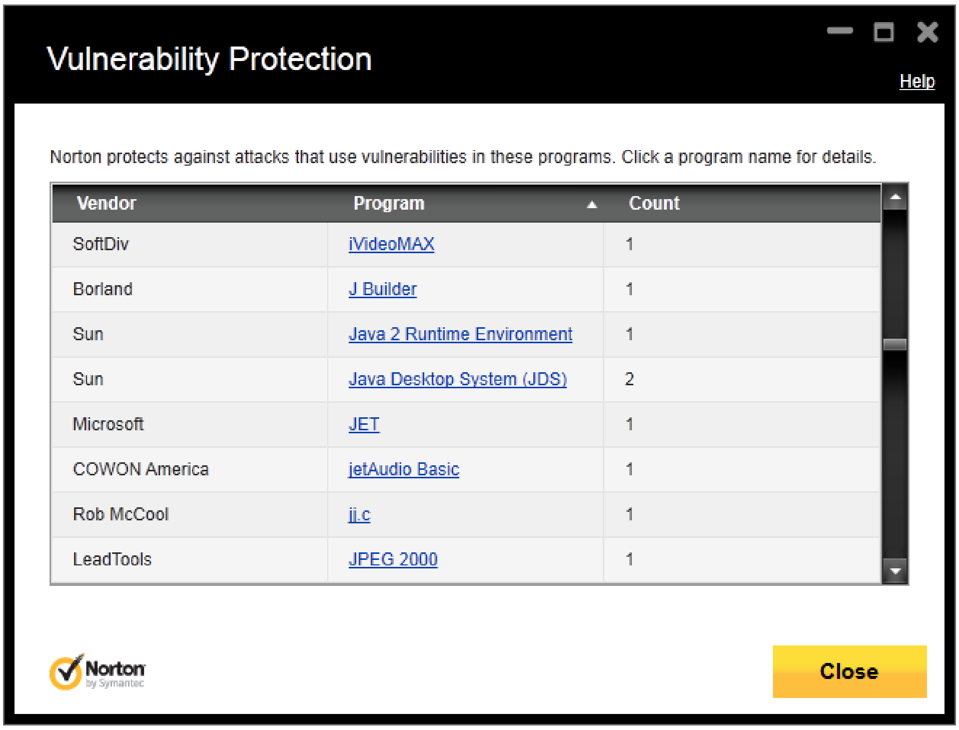
Figure 2.3: CVE Search Results

2.4.2 Norton Vulnerability Protection

For example, the now deprecated Norton Vulnerability Protection tool, as shown in Figure 2.4.2[4] lists the programs and the total number of vulnerabilities found, providing more information on each program. This method has the advantage of indicating the specific programs that have vulnerabilities, with the aim of allowing a user to update their vulnerable applications, however it does not allow for more fine-grained information.

2.5 Overview

[4] Available at community.norton.com



Norton protects against attacks that use vulnerabilities in these programs. Click a program name for details.

Vendor	Program ▲	Count
SoftDiv	iVideoMAX	1
Borland	J Builder	1
Sun	Java 2 Runtime Environment	1
Sun	Java Desktop System (JDS)	2
Microsoft	JET	1
COWON America	jetAudio Basic	1
Rob McCool	jj.c	1
LeadTools	JPEG 2000	1

Figure 2.4: Norton Vulnerability Protection Results

3 Implementation

3.1 Overview

The analysis is implemented as a series of stages, firstly, the e-mail header is parsed, to extract important information to a predefined set of Java objects. This is followed by the analysis phase, where the resultant data is passed to a set of analyser modules, each running separately. Finally, this information is presented to the user. This chapter presents each of these stages in detail.

3.1.1 Program Overview

The program is split up into three main stages: textual analysis and parsing; header contents analysis, and visualisation, as shown in Figure 3.1.

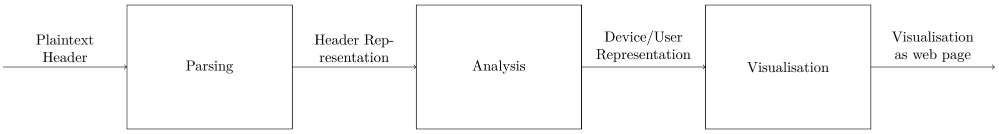


Figure 3.1: Simplified Control Flow of Application

3.2 Definitions

The following covers the essential definitions required for the notation and concepts that will be discussed in this document.

3.2.1 Parsing

In order to aid the parsing of the e-mail header, a combination of regular expressions and context-free grammars are needed, and defined as follows.

Alphabets and Languages A set of symbols, usually denoted as Σ. A language is a subset of $\mathcal{P}(\Sigma)$.

The following special classes are provided as part of the Perl-Compatible Regular Expression library, and are subsets of the alphabet of Unicode characters, defined in Group et al. n.d.

alnum — letters and digits

alpha — letters

ascii — the set of ASCII characters (character codes 0 — 127)

blank — tabs or blank spaces

cntrl — control characters

digit — decimal digits

graph — printing characters (excluding spaces)

lower — lower-case letters

print — printing characters (including spaces)

punct — punctuation marks (printing characters excluding letters and spaces)

space — white space

upper — upper case letters

word — "word" characters (same

xdigit — hexadecimal digits

Regular Languages

Regular languages are defined as follows:

- $\emptyset$ and $\{\epsilon\}$ are regular languages
- for each $a \in \Sigma$, $\{a\}$ is a regular language
- if A and B are both regular, $A \cup B$, $A \cdot B$ and A^* are regular languages.

  $A \cup B$ is the union of two languages. $A \cup B = \{s : s \in A \vee s \in B\}$

  $A \cdot B$ is the concatentation of two languages. $A \cdot B = \{ab : a \in A, b \in B\}$

  A^* is the Kleene star of a language.

$$\begin{aligned} A_0 &= \{\epsilon\} \\ A_1 &= A \\ A_{i+1} &= \{aa' : a \in A_i, a' \in A\} \\ A^* &= \bigcup_{i \in \mathbb{N}} A_i \end{aligned}$$

Context-Free Grammars

A context-free grammar G is defined as $G = (V, \Sigma, R, S)$ where:

- V is a variable.
- Σ is the alphabet of symbols.
- R is a relation defined over $V \rightarrow (V \cup \Sigma)^*$
- S is the start symbol

For example, $\langle \mathrm{S} \rangle$ is the field name with the associated productions $\langle \mathrm{T} \rangle \, \langle \mathrm{U} \rangle$, where T and U are productions.

$$\langle \mathrm{S} \rangle \quad \models \quad \langle \mathrm{T} \rangle \, \langle \mathrm{U} \rangle$$

For example, $\langle \mathrm{S} \rangle$ is the field name with the associated productions $a \, \langle \mathrm{U} \rangle$, where a is a terminal symbol.

$$\langle \mathrm{S} \rangle \quad \models \quad a \, \langle \mathrm{U} \rangle$$

This is then extended in the following ways used in the RFC syntax.

The square brackets are used to indicate an optional element.

$$\langle \text{field} \rangle \quad \models \quad \langle \text{field-name} \rangle\text{: } [\langle \text{field-body} \rangle] \text{ CRLF}$$

The asterisk is used to indicate an element that appears 0 or more times. $n*$ is used to indicate a component that repeats n or more times.

$$\langle \text{fields} \rangle \quad \models \quad \langle \text{dates} \rangle \, \langle \text{source} \rangle \, 1* \langle \text{destination} \rangle \; * \langle \text{optional-fields} \rangle$$

The hash-symbol is used to indicate an element that appears a certain number of times. $m * n$ is used to indicate a component that repeats at least m times and at most n times.

$$\langle \text{fields} \rangle \quad \models \quad \langle \text{dates} \rangle \, \langle \text{source} \rangle \, 1\#\langle \text{destination} \rangle \; * \langle \text{optional-fields} \rangle$$

The | is used to indicate a selection between a pair of elements.

$$\langle \text{fields} \rangle \quad \models \quad \text{a} \;|\; \text{b}$$

3.2.2 Database Queries

The following notations will be used for the CVE database queries.

Set-Theoretic Operators The operators $F \cup G$, $F \cap G$, $F \setminus G$ behave as is expected for these operators, resulting in the union, intersection and difference of the sets. The only proviso being that the atrribute names must match.

Selection

$$\sigma_{\text{product=thunderbird}} D$$

The above notation is used to indicate a search over the attribute named “product” for the string “thunderbird” in the database table D. As a single database is only being used, this may be occasionally elided. The output of this function is another object of the same type as D.

Projection

$$\pi_{\text{product}} D$$

The above notation is used to indicate a projection on the attribute named "product" in the database table D. The output of this function is another object of the same type as D.

Composition The above functions results can be coposed repeatedly to produce more specific search queries.

3.2.3 Data Structures

These wil be drawn using square boxes to represent single objets that are encapsulated within an object. Square boxes with an inner square box indicate some collection of objects.

Thin arrows will be used to denote the encapsulation relation, with thicker arrows being used to list relevant public methods.

Where the type of an object can be expressed simply, the following conventions are used:

Functions — $\alpha \to \beta$ `Str` is a function from α to β stored using a `Str` object in Java.

Tuples — (α, β) represents a two-tuple containing objects of type α and β, respectively. This extends naturally for arbitrarily many objects.

Lists — $[\alpha]$ represents a list or array of objects, all with type α.

3.3 Data Extraction and Parsing

The parser's operation completes in a number of stages, following RFC822 (Crocker 1982). The header is divided up into two disjoint sections, the routing information (`Received from...`) and the key-value map of other pertinent information.

3.3.1 Received fields

The received fields are the most complicated part of the e-mail header to parse, as they are described by a non-trivial grammar, presented below.

$$\begin{aligned}
\langle\text{message}\rangle &\models \langle\text{fields}\rangle \; *(\text{CRLF} \; *\textit{text}) \\
\langle\text{fields}\rangle &\models \langle\text{dates}\rangle\ \langle\text{source}\rangle\ 1*\langle\text{destination}\rangle \; *\langle\text{optional-fields}\rangle \\
\langle\text{field}\rangle &\models \langle\text{field-name}\rangle\text{: }[\langle\text{field-body}\rangle]\text{ CRLF} \\
\langle\text{field-name}\rangle &\models \textit{any word consisting of CHAR, excluding CTLs, SPACE, and ``":"} \\
\langle\text{field-body}\rangle &\models \langle\text{field-body-contents}\rangle\ [\text{CRLF } \textit{LWSP-char}\ \langle\text{field-body}\rangle] \\
\langle\text{field-body-contents}\rangle &\models \textit{ASCII characters} \\
\langle\text{source}\rangle &\models [\langle\text{trace}\rangle]\ \langle\text{originator}\rangle[\langle\text{resent}\rangle] \\
\langle\text{trace}\rangle &\models \langle\text{return}\rangle\ 1*\langle\text{received}\rangle \\
\langle\text{return}\rangle &\models \text{Return-path: }\langle\text{route-addr}\rangle \\
\langle\text{recieved}\rangle &\models \text{Received:} \\
\langle\text{cont.}\rangle &\models [\text{from }\langle\text{domain}\rangle] \\
\langle\text{cont.}\rangle &\models [\text{by }\langle\text{domain}\rangle]
\end{aligned}$$

$$\begin{array}{rcl}
\langle\text{cont.}\rangle & \models & [\text{via } \langle\text{atom}\rangle] \\
\langle\text{cont.}\rangle & \models & *(\text{with } \langle\text{atom}\rangle) \\
\langle\text{cont.}\rangle & \models & [\text{id } \langle\text{msg-id}\rangle] \\
\langle\text{cont.}\rangle & \models & [\text{for } \langle\text{addr-spec}\rangle] \\
\langle\text{cont.}\rangle & \models & ;\ \langle\text{date-time}\rangle \\
\langle\text{msg-id}\rangle & \models & <\langle\text{addr-spec}\rangle> \\
\langle\text{addr-spec}\rangle & \models & \langle\text{local-part}\rangle\ @\ \langle\text{domain}\rangle \\
\langle\text{local-part}\rangle & \models & \langle\text{word}\rangle\ *(.\ \langle\text{word}\rangle) \\
\langle\text{word}\rangle & \models & \langle\text{atom}\rangle \mid \langle\text{quoted-string}\rangle \\
\langle\text{domain}\rangle & \models & \langle\text{sub-domain}\rangle * (.\langle\text{sub-domain}\rangle) \\
\langle\text{sub-domain}\rangle & \models & \langle\text{domain-ref}\rangle \mid \langle\text{domain-literal}\rangle \\
\langle\text{domain-ref}\rangle & \models & \langle\text{atom}\rangle \\
\langle\text{date-time}\rangle & \models & [\textit{day,}]\ \textit{date time} \\
\langle\text{atom}\rangle & \models & 1*\ \textit{any character excluding specials, SPACE and CTLs}
\end{array}$$

An example field is as follows:

```
Received: from relay12.mail.ox.ac.uk (129.67.1.163)
    by HUB05.ad.oak.ox.ac.uk (163.1.154.231)
    with Microsoft SMTP Server id 14.3.169.1;
    Sat, 14 Nov 2015 10:55:35 +0000
```

3.3.2 Other fields

These are read by a Python script and output to STDOUT to be read by the Java parser in a consistent format. These are then loaded into a hashmap to allow quick lookup.

3.3.3 Input Data Structures

The raw string of the message header is the only input to this module.

3.3.4 Output Data Structures

The data structure presented in Figure 3.2 shows the output of the parsing and textual analysis module, which is then provided as an input to the analysis modules.

3.4 Analysis

After completing the parsing of the fields, it is then ready to be analysed for different features. All of the analysers implement the HeaderAnalyser interface, requiring information about the header to be analysed, and the currently running application. All of these then implement the Runnable interface, allowing the class to be run asynchronously.

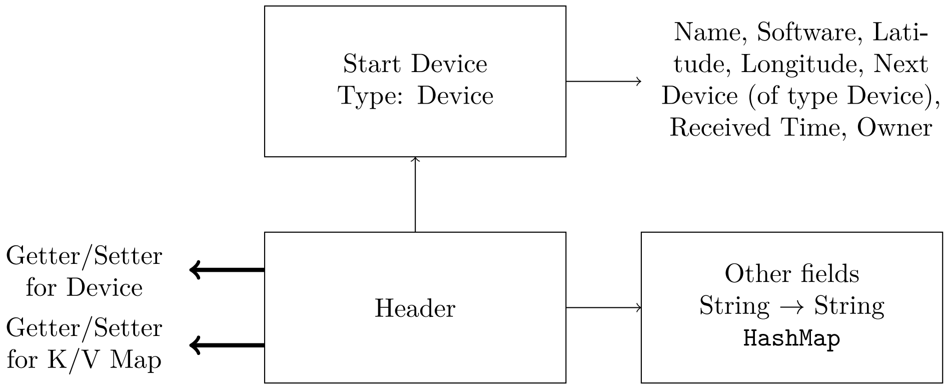

Start Device
Type: Device
Name, Software, Latitude, Longitude, Next Device (of type Device), Received Time, Owner
Getter/Setter for Device
Getter/Setter for K/V Map
Header
Other fields
String → String
HashMap

Figure 3.2: Header Data Structure Format

3.4.1 Text-Based

The fields from the header are analysed in different modules, with searches being performed for specific strings. Of particular interest to Oxford Nexus users is the "X-Oxford-Username" string, containing the username of the individual that sent the message. As confirming the username is a fairly standard security procedure for an IT support technician, having access to this information could allow a phisher in a later stage of an attack to increase their credibility.

In some cases, the likely keys that are being searched for are known in advance, and can then be checked against the hash-map of entries.

An example of this approach is for the specific check for an Oxford username, as shown in Algorithm 1.

```
Input: Header
Output: Any Oxford-based username that is found
if X-Oxford-Username ∈ Header.KvMap then
    return Header.KvMap(X-Oxford-Username);
end
```

Algorithm 1: Lookup based on a known key

Alternatively, we may be interested in properties of the keys, necessitating a search over the keys, as shown in Algorithm 2.

```
Input: Header
Output: Any information relating to Microsoft Exchange that is found
foreach Key k ∈ Header.KvMap do
    if k starts-with X-MS-Exchange then
        return Header.KvMap(k);
    end
end
```

Algorithm 2: Lookup based on a key property

3.4.2 Client Inferrence

In Nurse et al. 2015, a number of different e-mail clients were identified based on the header tags that were present. By identifiying these pieces of software likely to be found on a user's machine, we gain a significant amount of information from them, and should therefore devote some effort to correctly identifying them. Using a number of e-mail samples provided, I have been able to extract examples for a number of different e-mail clients, and use them to infer the client being used. Using a sample of e-mails, it is possible to find information for additional e-mail clients and senders, such as (but not limited to) PHPMailer and Foxmail. A number of these approaches are shown in Algorithm 3.

```
Input: Header
Output: The name of the software that is likely being used, and necessary CVE
        Product name (elided for brevity)
if "Message-ID" $\in$ Header.KvMap then
    if Header.KvMap("Message-ID") contains "email.android.com" then
        return "Android Device";
    end
else if "X-Mailer" $\in$ Header.KvMap then
    switch Header.KvMap("X-Mailer") contains do
        case "iPhone" do
            return "iPhone"
        end
        case "Outlook Express" do
            return "Microsoft Outlook Express"
        end
        ...
    end
else if "User-Agent" $\in$ Header.KvMap then
    return "Thunderbird";
else if Header.KvMap $\cap$ OutlookKeywords $\neq \emptyset$ then
    if "X-Mailer" $\in$ Header.KvMap then
        return Apple Mail
    else
        return Outlook
    end
end
```

Algorithm 3: Client Inferrence Technique

3.4.3 Database Queries

Using the results gathered from the text-based queries and analysis of the received fields, relevant software configurations are extracted and queried against results in the CVE database. These are then parsed and collated in preparation for displaying the outputs. Specifically, the queries are limited to those matching the product name, and vulnerabilities that can be remotely executed.

As more information is found, more details of products used will also become available. These are added asynchronously.

```
Input: Header product name $p$
Output: CVE Entries
cve-list ← $\emptyset$;
foreach $s \in \sigma_{vector \neq LOCAL}\sigma_{product=p}D$ do
    cve-builder←blank cve;
    cve-builder.id← $\pi_{\text{CVE-ID}}s$;
    . . . – extract other features;
    cve-list ← cve-list $\cup$ make(cve-builder);
end
return cve-list;
```

Algorithm 4: Extracting CVE entries

3.4.4 Analysis Modules and Data Flow

The following modules are used in the analysis of e-mail headers. Via `HeaderAnalyser`, they all subclass `Callable<Object>`, allowing them to return values to calling classes when side-effects are undesirable.

HeaderAnalyser — the base interface for all analysis modules.

ClientInferrer — as described in Algorithm 3, this analyses the entire header, looking for specific indicators relating to e-mail clients.

DeviceAnalyser — Extracts the hostname, and then IP address, or IP address for each device, allowing a lookup of its co-ordinates.

ExchangeHeaderAnalyser — determines if a Microsoft Exchange server has handled the message.

GeoIPAnalyser — Given an IP address, this module looks up the latitude and longitude for said IP address. This search will fail for local IP addresses (commonly found in the 192.168.0.0/16 subnet).

OxfordHeaderAnalyser — This module searches for the specific `X-Oxford-Username` field, one of the most common fields in my collection of e-mail headers.

SenderInformationExtractor — This is used to lookup an individual's name from the set of fields, if it is available.

VulnerabilityAnalyser — For each entry from the database as a `String`, this analyser returns a `VulnerabilityDisclosure` object.

VulnerabilityFinderManager — this is an interface for other classes to implement. A reference implementation is provided in `VulnerabilityFinderManagerImpl`, with a simple implementation found in `NoopVulnerabilityFinderManager` for systems that lack the CVE database.

WhoIsAnalyser — Using a hostname, this looks up the relevant owning organisation for a server, if it is available.

Figure 3.3 shows the direct interaction of the analysis modules. Unless noted, if an arrow goes from α to β, it is used to indicate that α calls β, passing data from α, and/or receiving data from β.

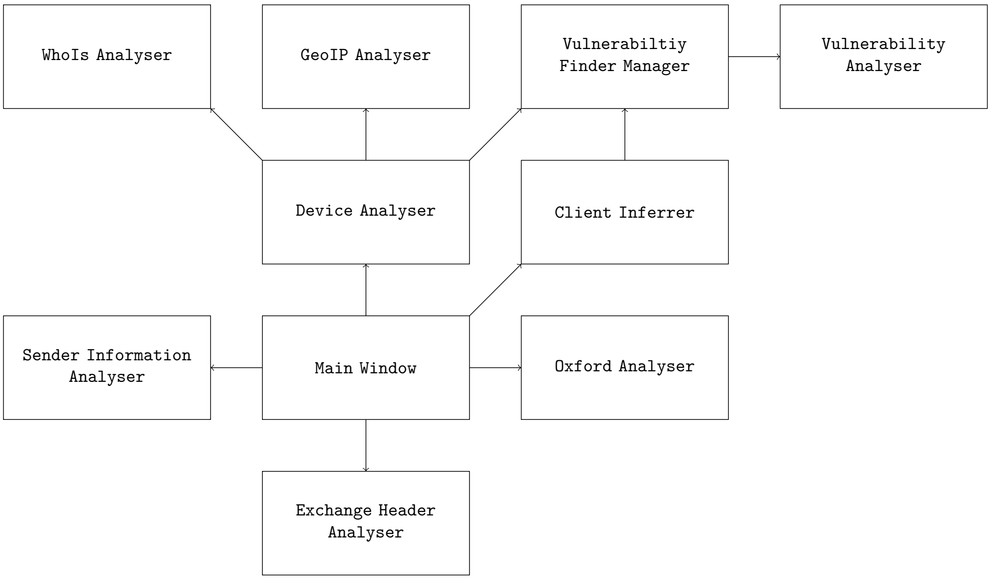


Figure 3.3: Information flow between analysis modules

3.4.5 Output Data Structures

The output of the analysis modules is compiled into the `FoundInformation` class, which represents the list of facts, servers and other information that has been gathered from an e-mail.

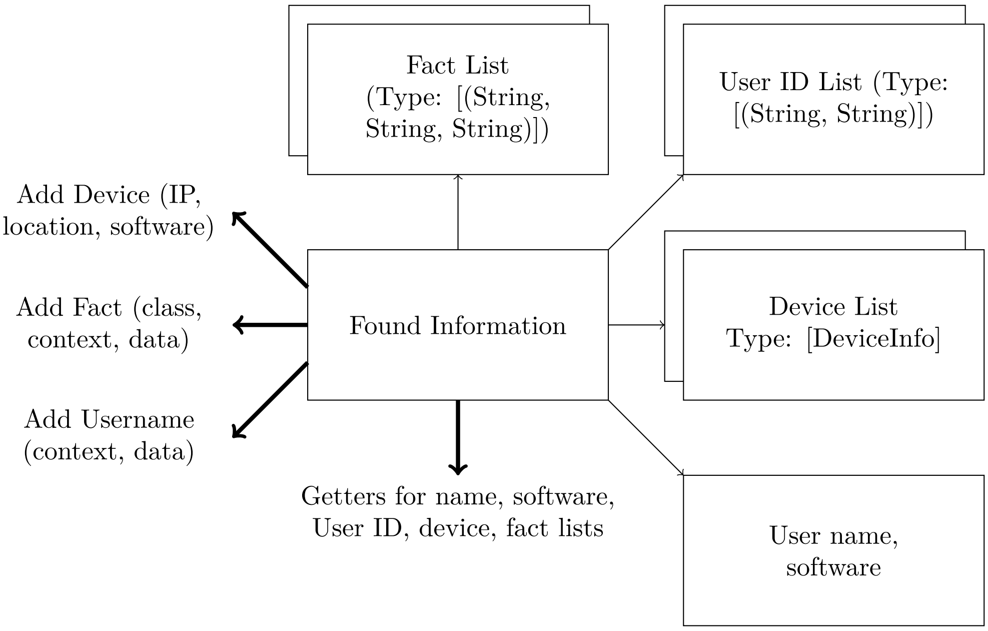

Fact List (Type: [(String, String, String)])
User ID List (Type: [(String, String)])
Add Device (IP, location, software)
Add Fact (class, context, data)
Add Username (context, data)
Found Information
Device List Type: [DeviceInfo]
Getters for name, software, User ID, device, fact lists
User name, software

Figure 3.4: Found Information Data Structure Format

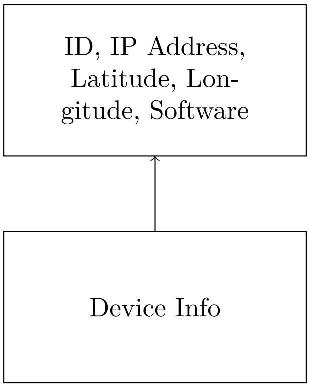


Figure 3.5: Device Info Data Structure

3.5 Visualising the Results

Using a pre-existing template, the results from the e-mail analysis will be presented in a temporary webpage, which can then be saved independently. Other than the referenced JavaScript libraries, the document requires no additional information or database access, allowing it to be quickly shared.

4 Evaluation

After producing the application described in Chapter 3, referring back to the design specifications to determine its performance against the stated criteria was the next step.

4.1 Methodology

Using a sample of e-mails provided by my supervisor, each of them was run through the final version of the program, and scored based on the following attributes:

- Let M be the total number of received fields.
- Let N be the total number of other fields.
- 1 point is added to the total R for each piece of information found in a Received field for the following:

  One of device name *or* IP address

  Software *or* protocol used

  Vulnerabilities found for the relevant piece of software

  Location Data
- 1 point is addded to F for each piece of information found in other fields.

The final score for an e-mail is given as

$$\frac{1}{2}\left(\frac{R}{4M}+\frac{F}{N}\right)$$

to give a value between 0 and 1 for each e-mail.

The e-mails received have been numbered from 1 up to 70, and a random sample of size 30 was selected using the following code:

```
>>> random.seed()
>>> random.sample(list(range(1,70)), 30)
[64, 48, 11, 21, 63, 68, 27, 69, 29,  8, 28,
 34, 13, 57, 10,  3, 22, 32, 23, 49, 26, 45,
 19,  1, 36, 46, 41, 18, 20, 17]
```

Thus giving a sorted list of the following e-mails: 1, 3, 8, 10, 11, 13, 17, 18, 19, 20, 21, 22, 23, 26, 27, 28, 29, 32, 34, 36, 41, 45, 46, 48, 49, 57, 63, 64, 68, 69.

Header	Total Fields	Received Fields (M)	Other Fields (N)
1.txt	26	7	19
3.txt	32	6	26
8.txt	34	8	26
10.txt	22	3	19
11.txt	29	6	23
13.txt	17	4	13
17.txt	27	6	21
18.txt	20	6	14
19.txt	20	6	14
20.txt	22	7	15
21.txt	22	6	16
22.txt	22	6	16
23.txt	26	6	20
26.txt	19	6	13
27.txt	21	1	20
28.txt	22	6	16
29.txt	20	6	14
32.txt	24	6	18
34.txt	23	5	18
36.txt	26	6	20
41.txt	21	6	15
45.txt	23	6	17
46.txt	26	5	21
48.txt	25	6	19
49.txt	21	6	15
57.txt	28	6	22
63.txt	23	5	18
64.txt	36	9	27
68.txt	21	6	15
69.txt	22	6	16

Table 4.1: M and N values for chosen headers

4.2 Results

Table 4.1 gives the M and N values for the different sampled e-mails. These were sampled using standard Unix tools: `grep`ping for fields and counting the output.

The results of the testing are contained within Table **??**, and give a breakdown for each e-mail's values of R and F, as well as the final score.

During the testing, the following trends were noticed. As many of the e-mails passed through the same set of servers, as they had been received by an Oxford e-mail address, the same set of servers were frequently seen, all of which had associated IP addresses, geolocation data and (except for one server running Microsoft SMTP Server) CVE data for the running software. For an e-mail sent within the University Nexus system, this gives an inflated score, as very few of the servers are missing information.

The score for the information gathered from fields does not take into account the number of fields needed to determine or infer a piece of information. For example, the presence, or absence of multiple fields is required to determine a piece of information. Nor does it consider the relative value of a piece of information, failing to rate the presence of a username above the presence of a particular piece of software also giving false positives.

However, very few e-mails in the testing population contained fields relating to usernames (`X-Oxford-Username`, `X-...-User`, `X-Authenticated-User`) compared to the result in Nurse et al. 2015, which found 14% of e-mails to contain usernames as opposed to the 8% found in the population.

5 Conclusions and Future Work

5.1 Conclusions

5.2 Future Work

During the late stages of development and testing, a number of missing features quickly became apparent. Due to the limited information available, and the differences in version numbering, a decision was made to search for all available vulnerabilities for an application, allowing the user to discern which were most relevant. Subsequent versions could focus on the different pieces of version data available. For example, `esmtp` frequently references its version number in the “Received” field frequently.

Alternatively, a better picture may be presented by accumulating multiple e-mails. For example, using the information provided from multiple members of single organisation, a better picture may be built up of the software used by the servers, as well as the network configuration.

Bibliography

[1] Rakesh Agrawal et al. “Order preserving encryption for numeric data”. In: *Proceedings of the 2004 ACM SIGMOD international conference on Management of data*. ACM. 2004, pp. 563–574.

[2] D. Crocker. *STANDARD FOR THE FORMAT OF ARPA INTERNET TEXT MESSAGES*. STD 11. RFC Editor, Aug. 1982.

[3] Jerry Felix and Chris Hauck. “System security: a hacker’s perspective”. In: *Interex Proceedings* 1 (1987), pp. 6–6.

[4] PHP Group et al. *PHP: Character classes - Manual*. URL: `https://secure.php.net/manual/en/regexp.reference.character-classes.php`.

[5] Danesh Irani et al. “Modeling unintended personal-information leakage from multiple online social networks”. In: *Internet Computing, IEEE* 15.3 (2011), pp. 13–19.

[6] Akanksha Joshi, Ravendar Lal, and Tim Finin. “Extracting cybersecurity related linked data from text”. In: *Semantic Computing (ICSC), 2013 IEEE Seventh International Conference on*. IEEE. 2013, pp. 252–259.

[7] Jason RC Nurse et al. “Investigating the leakage of sensitive personal and organisational information in email headers”. In: *Journal of Internet Services and Information Security (JISIS)* 5.1 (2015), pp. 70–84.

[8] Panagiotis Papadimitriou and Hector Garcia-Molina. “Data leakage detection”. In: *Knowledge and Data Engineering, IEEE Transactions on* 23.1 (2011), pp. 51–63.

[9] Anna Squicciarini, Smitha Sundareswaran, and Dan Lin. “Preventing information leakage from indexing in the cloud”. In: *Cloud Computing (CLOUD), 2010 IEEE 3rd International Conference on*. IEEE. 2010, pp. 188–195.

[10] Brad Templeton. *Reaction to the DEC Spam of 1978*. URL: `http://www.templetons.com/brad/spamreact.html`.

[11] Marwan Al-zarouni. *Tracing E-mail Headers*. 2004.