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.

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, Jason Nurse for his assistance throughout the year; my tutor, 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

1 Introduction

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.

1.1 Motivation

1.2 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.

Chapter 3 sets out essential terminology that will be used in this report and is of particular relevance when discussing the implementation, which is covered in chapter 4. The implementation's high-level structure and details will be discussed, 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 5, beginning with the methodology used, and presenting a number of results.

Finally, chapter 6 will discuss my conclusions and areas of further improvement.

1.3 Aims and Objectives

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.

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

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

1.4 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.

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.

1.5 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.

2 Literature Review

2.1 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.

2.2 Existing Tools

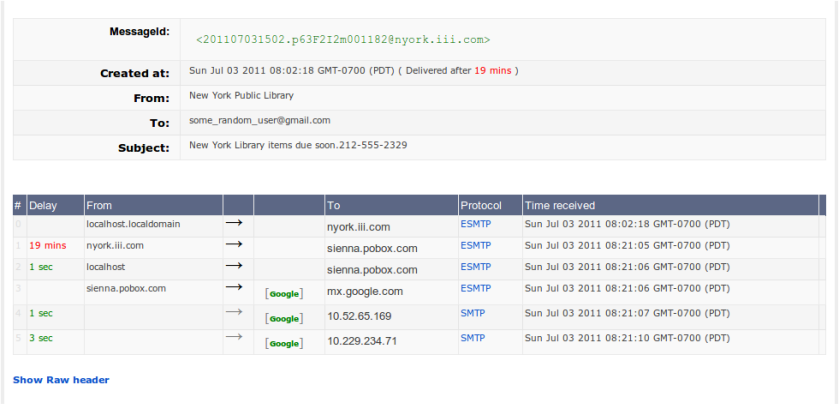
Several tools already exist online to display the information that is found in e-mail headers. Tools from Microsoft and Google exist to analyse the contents of e-mail headers. These tools clearly display the information displayed in the header, showing the key-value pairs, and the set of servers the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

2.2.1 Google

The Google Apps Toolbox features an e-mail header analyser[1]. An example of the output of the utility is found in figure 2.2.1.

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.

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



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)

Show Raw header

Figure 2.1: Google Apps Toolbox E-mail header output

2.2.2 Microsoft

The Microsoft Message Header Analyzer [2] and showing sample results in figure 2.2.2

2.2.3 CVE Mitre Lookup

There are a number of tools to look up CVEs[3] and showing sample results in figure 2.2.3. 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.

For example, the now deprecated Norton Vulnerability, as shown in figure 2.2.3[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]Fount at `https://testconnectivity.microsoft.com/MHA/Pages/mha.aspx`
[3]Fount at `https://www.cve.mitre.org/find/index.html`
[4]Available at `community.norton.com`

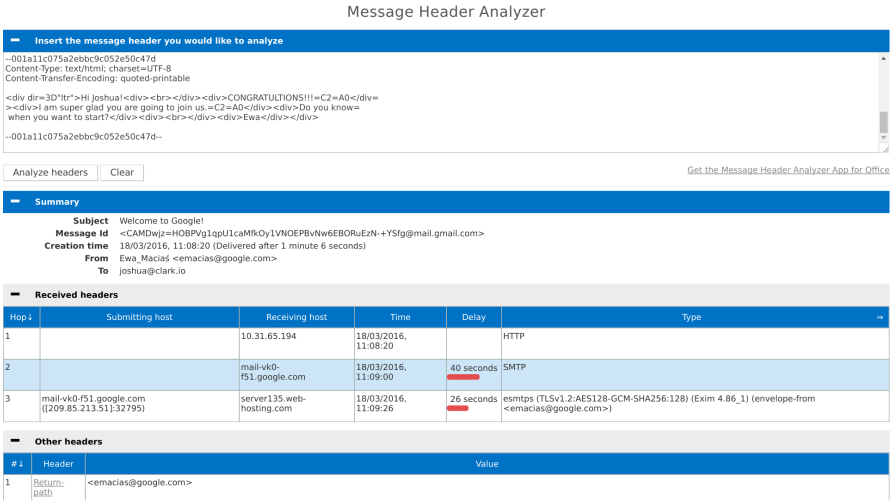


Figure 2.2: Microsoft E-mail header output

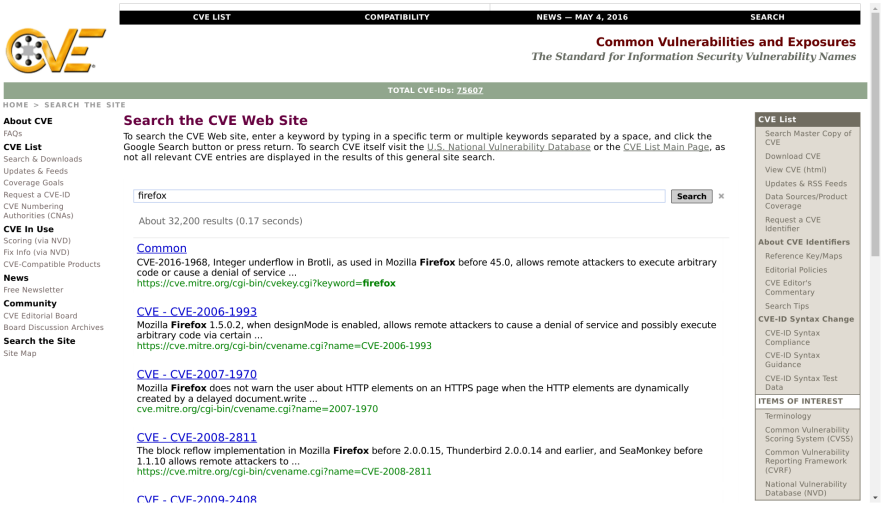


Figure 2.3: CVE Search Results

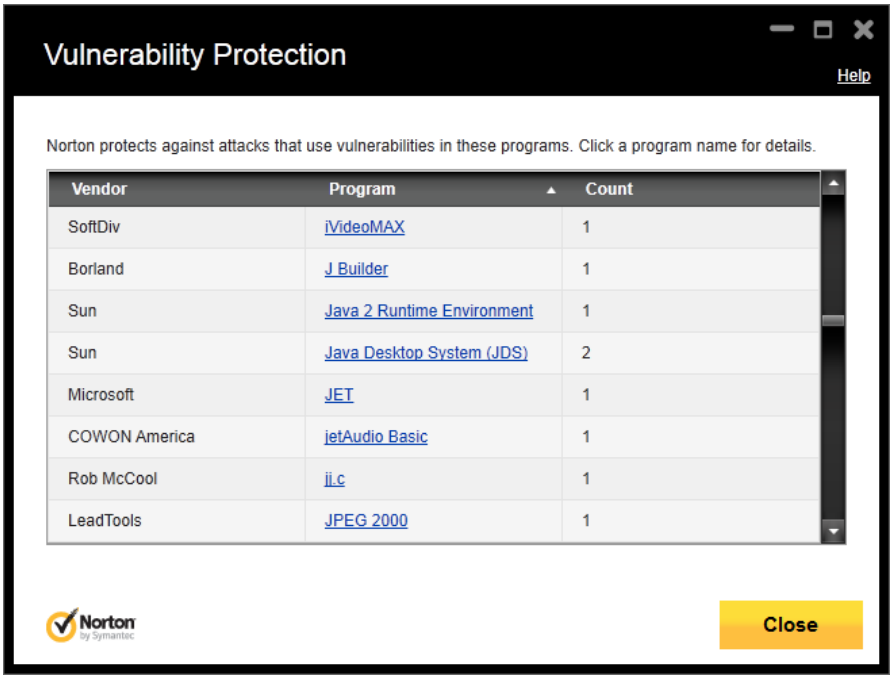


Figure 2.4: Norton Vulnerability Protection Results

3 Definitions

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

3.1 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.

3.1.1 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

3.1.2 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.

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

3.1.3 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 $V \to (V \cup \Sigma)^*$
- S is the start symbol

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

$$\langle \text{S} \rangle \quad \models \quad \langle \text{T} \rangle \langle \text{U} \rangle$$

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

$$\langle \text{S} \rangle \quad \models \quad a \langle \text{U} \rangle$$

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.

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

4 Implementation

4.1 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.

4.2 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.

4.2.1 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.

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

$$
\begin{aligned}
\langle \text{addr-spec} \rangle &\models \langle \text{local-part} \rangle \ @ \ \langle \text{domain} \rangle \\
\langle \text{local-part} \rangle &\models \langle \text{word} \rangle \ * \ (. \ \langle \text{word} \rangle) \\
\langle \text{word} \rangle &\models \langle \text{atom} \rangle \ | \ \langle \text{quoted-string} \rangle \\
\langle \text{domain} \rangle &\models \langle \text{sub-domain} \rangle * (.\langle \text{sub-domain} \rangle) \\
\langle \text{sub-domain} \rangle &\models \langle \text{domain-ref} \rangle \ | \ \langle \text{domain-literal} \rangle \\
\langle \text{domain-ref} \rangle &\models \langle \text{atom} \rangle \\
\langle \text{date-time} \rangle &\models [\textit{day,}] \ \textit{date time} \\
\langle \text{atom} \rangle &\models 1* \textit{ any character excluding specials, SPACE and CTLs}
\end{aligned}
$$

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
```

4.2.2 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.

4.3 Analysis

After completing the parsing of the field, 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.

4.3.1 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.

4.3.2 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.

4.4 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.

Input: Input
Output: Output
foreach *alkasdfj* **do**
  asdkf;
end

Algorithm 1: Words

5 Testing

5.1 Methodology

5.2 Results

6 Conclusions and Future Work

6.1 Conclusions

6.2 Future Work

Bibliography

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

[2] Jerry Felix and Chris Hauck. "System security: a hacker's perspective". In: *Interex Proceedings* 1 (1987), pp. 6–6.

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

[4] 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.

[5] 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.

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

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