E-jection fraction - Tracking how your website pumps out E-Mails

Ben Trovato[*]
Institute for Clarity in Documentation
1932 Wallamaloo Lane
Wallamaloo, New Zealand
trovato@corporation.com

G.K.M. Tobin[†]
Institute for Clarity in Documentation
P.O. Box 1212
Dublin, Ohio 43017-6221
webmaster@marysville-ohio.com

Lars Thørväld[‡]
The Thørväld Group
1 Thørväld Circle
Hekla, Iceland
larst@affiliation.org

Lawrence P. Leipuner
Brookhaven Laboratories
Brookhaven National Lab
P.O. Box 5000
lleipuner@researchlabs.org

Sean Fogarty
NASA Ames Research Center
Moffett Field
California 94035
fogartys@amesres.org

Charles Palmer
Palmer Research Laboratories
8600 Datapoint Drive
San Antonio, Texas 78229
cpalmer@prl.com

ABSTRACT

E-Mail header injection vulnerability is a class of vulnerability that can occur in web applications that use user input to construct e-mail messages. E-Mail injection is possible when the mailing script fails to check for the presence of e-mail headers in user input (either form fields or URL parameters). The vulnerability exists in the reference implementation of the built-in "mail" functionality in popular languages like PHP, Java, Python, and Ruby. With the proper injection string, this vulnerability can be exploited to inject additional headers and/or modify existing headers in an E-Mail message, allowing an attacker to completely alter the content of the e-mail.

This thesis develops a scalable mechanism to automatically detect E-Mail Header Injection vulnerability and uses this mechanism to quantify the prevalence of EMail Header Injection vulnerabilities on the Internet. Using a black-box testing approach, the system crawled 21,675,680 URLs to find URLs which contained form fields. 6,794,917 such forms were found by the system, of which 1,132,157 forms contained e-mail fields. The system used this data feed to discern the forms that could be fuzzed with malicious payloads. Amongst the 934,016 forms tested, 52,724 forms were found to be injectable with more malicious payloads. The system tested 46,156 of these, and was able to find 496 vulnerable URLs across 222 domains, which proves that the threat is widespread and deserves future research attention.

CCS Concepts

•**Computer systems organization** → **Embedded systems;** *Redundancy;* Robotics; •**Networks** → Network reliability;

Keywords

ACM proceedings; LaTeX; text tagging

1. INTRODUCTION

The World Wide Web has single-handedly brought about a change in the way we use computers. The ubiquitous nature of the Web has made it possible for the general public to access it anywhere and on multiple devices like phones, laptops, personal digital assistants, and even on TVs and cars. This has ushered in an era of responsive web applications which depend on user input. While this rapid pace of development has improved the speed of dissemination of information, it does come at a cost. Attackers have an added incentive to break into user's e-mail accounts more than ever. E-Mail accounts are usually connected to almost all other online accounts of a user, and e-mails continue to serve as the principal mode of official communication on the web for most institutions. Thus, the impact an attacker can have by having control over the e-mail communication sent by websites to users is of an enormous magnitude.

Since attackers typically masquerade themselves as users of the system, if user input is to be trusted, then developers need to have proper sanitization routines in place. Many different injection attacks such as SQL injection or cross-site scripting (XSS) [10] are possible due to improper sanitization of user input.

Our research focuses on a lesser known injection attack known as E-Mail Header Injection. E-Mail Header Injection can be considered as the e-mail equivalent of HTTP Header Injection vulnerability [5]. The vulnerability exists in the reference implementation of the built-in "`mail`" functionality in popular languages like PHP, Java, Python, and Ruby.

[*]Dr. Trovato insisted his name be first.
[†]The secretary disavows any knowledge of this author's actions.
[‡]This author is the one who did all the really hard work.

WOODSTOCK '97 El Paso, Texas USA

DOI: 10.475/123_4

With the proper injection string, this vulnerability can be exploited to inject additional headers and/or modify existing headers in an e-mail message — with the potential to alter the contents of the e-mail message — while still appearing to be from a legitimate source.

E-Mail Header Injection attacks have the potential to allow an attacker to perform e-mail spoofing, resulting in phishing attacks that can lead to identity theft. The objective of our research is to study the prevalence of this vulnerability on the World Wide Web, and identify whether further research is required in this area.

We performed an expansive crawl of the web, extracting forms with e-mail fields, and injecting them with different payloads to infer the existence of E-Mail Header Injection vulnerability. We then audited received e-mails to see if any of the injected data was present. This allowed us to classify whether a particular URL was vulnerable to the attack. The entire system works in a black-box manner, without looking at the web application's source code, and only analyzes the e-mails we receive based on the injected payloads.

Structure of document

This thesis document is divided logically into the following sections:

- Chapter 2 discusses the background of E-Mail Header Injection, a brief history of the vulnerability, and enumerates the languages and platforms affected by this vulnerability.
- Chapter 3 discusses the System design, the architecture, and the components of the system.
- Chapter 4 describes the experimental setup and sheds light on how we overcame the issues and assumptions discussed in Chapter 3.
- Chapter 5 presents our findings and our analysis of the results.
- Chapter 6 continues the discussion of the results; the lessons learned over the course of the project, limitations, and a suitable mitigation strategy to overcome the vulnerability.
- Chapter 7 explores related work in the area.
- Chapter 8 concludes this thesis, with ideas to expand the research in this area.

We hope that our research sheds some light on this relatively less well-known vulnerability, and find out its prevalence on the World Wide Web. In summary, we make the following contributions:

- A black-box approach to detecting the presence of E-Mail Header Injection vulnerability in a web application.
- A detection and classification tool based on the above approach, which will automatically detect such E-Mail Header Injection vulnerabilities in a web application.
- A quantification of the presence of such vulnerabilities on the World Wide Web, based on a crawl of the Web, including 21,675,680 URLs and 6,794,917 forms.

2. E-MAIL HEADER INJECTION BACKGROUND

This chapter goes into the background of the problem at hand and gives a brief history of E-Mail Header Injection. It then describes the languages affected by this vulnerability and discusses the overall impact E-Mail Header Injection can have, and the attacks that can result from this vulnerability.

2.1 Problem Background

E-Mail Header Injection belongs to a broad class of vulnerabilities known simply as injection attacks. However, unlike its more popular siblings, SQL injection [1], [3], [13], Cross-Site Scripting (XSS) [6], [8] or even HTTP Header Injection [7], relatively little research is available on E-Mail Header Injection.

As with other vulnerabilities in this class, E-Mail Header Injection is caused due to improper sanitization (or lack thereof) of user input. If the script that constructs e-mails from user input fails to check for the presence of e-mail headers in the user input, a malicious user — using a well-crafted payload — can control the headers set for this particular e-mail. This can be leveraged to enable malicious attacks, including, but not limited to, spoofing, phishing, etc.

2.2 History of E-Mail Injection

E-Mail Header Injection seems to have been first documented over a decade ago, in a late 2004 article on phpsecure.info [15] accredited to user tobozo@phpsecure.info describing how this vulnerability existed in the reference implementation of the "`mail`" function in PHP, and how it can be exploited. More recently, a blog post by Damon Kohler [9] and an accompanying wiki article [2] describe the attack vector and outline a few defense measures for the same.

As this vulnerability was initially found in the *mail()* function of PHP, E-Mail Header Injection can be traced to as early as the beginning of the 2000's, present in the *mail()* implementation of PHP 4.0.

The vulnerability was also described briefly (less than a page) by Stuttard and Pinto in their widely acclaimed book, "*The Web Application Hacker's Handbook: Discovering and Exploiting Security Flaws*" [14]. A concise timeline of the vulnerability is presented in Table 1.

An example of the vulnerable code written in PHP is shown in Listing 1. This code takes in user input from the PHP superglobal "`$_REQUEST['email']`", and stores it in the variable "`$from`", which is later passed to the "`mail`" function to construct and send the e-mail.

```
$from = $_REQUEST['email'];
$subject = "Hello Sai Pc";
$message = "We need you to reset your
    password";
$to = "schand31@asu.edu";

// example attack string to be injected as
    the value for
// $_REQUEST['email'] => 'sai@sai.com\nCC:
    spc@spc.com'
$retValue = mail($to, $subject, $message, "
    From: $from");
// E-Mail gets sent to both schand31@asu.
    edu AND spc@spc.com
```

Listing 1: PHP program with e-mail header injection vulnerability.

When this code is given the malicious input "`sai@sai.com \nBCC:spc@spc.com`" as the value of the "`$_REQUEST['email']`", it generates the SMTP Headers shown in Listing 2. It can be seen that the 'CC' (carbon copy) header that we injected appears as part of the resulting SMTP message. This will make the e-mail get sent to the e-mail address specified as part of the 'CC' as well.

```
Received: from mail.ourdomain.com
    ([62.121.130.29])
        by sai.com (Postfix) with ESMTP id
            5A08E52C0154
        for <sai@sai.com>; Sun, 20 Mar 2016
            13:56:58 -0700 (MST)
To: sai@sai.com
Subject: Hello Sai Pc
CC: spc@spc.com
Date: Sun, 20 Mar 2016 13:56:58 -0700 (MST)

We need you to reset your password
```

Listing 2: SMTP headers generated by a PHP mailing script.

Year	Notes
Early 2000's	PHP 4.0 is released, along with support for the tion against E-Mail Header Injection.
Jul 2004	Next Major version of PHP - Version 5.0 relea
Dec 2004	First known article about the vulnerability su
Oct 2007	The vulnerability makes its way into a text by
Dec 2008	Blog post and accompanying wiki about the examples.
Apr 2009	Bug filed about email.header package to fix th
Jan 2011	Bug fix for Python 3.1, Python 3.2, Python 2 to older versions not available.
Sep 2011	The vulnerability is described with an exam Stuttard and Pinto.
Aug 2013	Acunetix adds E-Mail Header Injection to th part of their Enterprise Web Vulnerability Sca
May 2014	Security Advisory for JavaMail SMTP Heade written by Alexandre Herzog.
Dec 2015	PHP 7 releases, mail function still unpatched.

Table 1: A brief history of e-mail header injection.

2.3 Languages Affected

This section describes the popular languages which exhibit this type of vulnerability. This section is not intended as a complete reference of vulnerable functions and methods, but rather as a guide that specifies which parts of the language are known to have the vulnerability.

2.3.1 *PHP*

PHP was one of the first languages found to have this vulnerability in its implementation of the *mail()* function. The early finding of this vulnerability can be attributed in part to the success and popularity of the language for creating web pages. According to w3techs [16], PHP is used by 81.9% of all the websites in existence, thereby creating the possibility of this vulnerability to be widespread.

PHP's low barrier to entry and lack of developer education about the existence of this vulnerability have contributed to the vulnerability continuing to exist in the language. After 13 further iterations of the language since the 4.0 release (the current version is 7.1), the *mail()* function is yet to be fixed after 15 years. However, it is specified in the PHP documentation [12] that the *mail()* function does not protect against this vulnerability. A working code sample of the vulnerability, written in PHP 5.6 (latest well-supported version), is shown in Listing 1.

2.3.2 *Python*

A bug was filed about the vulnerability in Python's implementation of the *email.header* library and its header parsing functions allowing newlines in early 2009, which was followed up with a partial patch in early 2011.

Unfortunately, the bug fix was only for the *email.header* package, and thus is still prevalent in other frequently used packages such as *email.parser*, where both the classic *Parser()* and the newer *FeedParser()* exhibit the vulnerability even in the latest versions - *2.7.11* and *3.5*. The bug fix was also not backported to older versions of Python. There is no mention of the vulnerability in the Python documentation for either library. A working code sample of the vulnerability, written in Python 2.7.11, is shown in Listing 3.

```
from email.parser import Parser
import cgi
form = cgi.FieldStorage()
to = form["email"] # input() exhibits the
    same behavior
msg = """To: """ + to + """\n
From: <user@example.com>\n
Subject: Test message\n\n
Body would go here\n"""

f = FeedParser() # Parser.parsestr() also
# contains the same vulnerability
f.feed(msg)
headers = FeedParser.close(f)

# attack string => 'sai@sai.com\nBCC:
    spc@spc.com'
# for form["email"]

# both to:sai@sai.com AND bcc:spc@spc.com
# are added to the headers
print 'To: %s' % headers['to']
print 'BCC: %s' % headers['bcc']
```

Listing 3: Python program with e-mail header injection vulnerability.

2.3.3 *Java*

Java has a bug report about E-Mail Header Injection filed against its JavaMail API. A detailed write-up by Alexandre Herzog [4] is complete with a proof of concept program that exploits the API to inject headers.

2.3.4 *Ruby*

From our preliminary testing, Ruby's built-in Net::SMTP library has this vulnerability. This is not documented on the library's homepage. A working code sample of the vulnerability, written in Ruby 2.0.0 (the latest stable version at the time of writing), is shown in Listing 4.

```
require 'sinatra'
require 'net/smtp'

get '/hello' do
```

Server Side Language	% of Usage
PHP	81.9
ASP.NET	15.8
Java	3.1
Ruby	0.6
Perl	0.5
JavaScript	0.2
Python	0.2

Table 2: Language usage statistics compiled from w3techs [16].

```ruby
email = params[:email]

message = """
From: Sai <schand31@asu.edu>
Subject: SMTP e-mail test
To: #{email}

This is a test e-mail message.
"""
# construct a post request with email set to attack_string
# attack_string => sai@sai.com%0abcc:spc@spc.com%0aSubject:Hello
Net::SMTP.start('localhost', 1025) do |smtp|
smtp.send_message message, 'schand31@asu.edu',
'to@todomain.com'
end
# Headers get added, and Subject field changes to what we set.
end
```

Listing 4: Ruby program with e-mail header injection vulnerability.

2.4 Potential Impact

The impact of the vulnerability can be pretty far-reaching. Table 2 shows the current server-side language usage statistics on the Web [16].

PHP, Java, Python, and Ruby (combined) account for over 85% [1] of the websites measured. The vulnerability can be exploited to do potentially any of the following:

- Phishing and Spoofing Attacks
  Phishing [11] (a variation of spoofing [17]) refers to an attack where the recipient of an e-mail is made to believe that the e-mail is a legitimate one. The e-mail usually redirects them to a malicious website, which then steals their credentials.

  E-Mail Header Injection gives attackers the ability to inject arbitrary headers into an e-mail sent by a website and control the output of the e-mail. This adds credibility to the generated e-mail, as it is sent right from the websites and people are more ready to trust e-mail that is received from the website directly and can thus result in more successful phishing attacks.

- Spam Networks
  Spam networks can use E-Mail Header Injection vulnerabilities on the ability to send a large amount of e-mail from servers that are trusted. By adding additional "`cc`" or "`bcc`" headers to the generated e-mail, attackers can easily achieve this effect.

  Due to the e-mails being from trusted domains, recipient e-mail clients might not flag them as spam. If they do flag them as spam, then that can lead to the website being blacklisted as a spam generator.

- Information Extraction of legitimate users
  E-Mails can contains sensitive data that is meant to be accessed only by the user. Due to E-Mail Header Injection, an attacker can easily add a "`bcc`" header, and send the e-mail to himself, thereby extracting important information. User privacy can thus be compromised, and loss of private information can by itself lead to other attacks.

- Denial of service by attacking the underlying mail server
  Denial of service attacks (DoS), can also be aided by E-Mail Header Injection. The ability to send hundreds of thousands of e-mails by just injecting one header field can result in overloading the mail server, and cause crashes and/or instability.

It is evident that E-Mail Header Injection is a critical vulnerability that web applications must address.

3. THE *BODY* OF THE PAPER

Typically, the body of a paper is organized into a hierarchical structure, with numbered or unnumbered headings for sections, subsections, sub-subsections, and even smaller sections. The command `\section` that precedes this paragraph is part of such a hierarchy.[2] LaTeX handles the numbering and placement of these headings for you, when you use the appropriate heading commands around the titles of the headings. If you want a sub-subsection or smaller part to be unnumbered in your output, simply append an asterisk to the command name. Examples of both numbered and unnumbered headings will appear throughout the balance of this sample document.

Because the entire article is contained in the **document** environment, you can indicate the start of a new paragraph with a blank line in your input file; that is why this sentence forms a separate paragraph.

3.1 Type Changes and *Special* Characters

We have already seen several typeface changes in this sample. You can indicate italicized words or phrases in your text with the command `\textit`; emboldening with the command `\textbf` and typewriter-style (for instance, for computer code) with `\texttt`. But remember, you do not have to indicate typestyle changes when such changes are part of the *structural* elements of your article; for instance, the heading of this subsection will be in a sans serif[3] typeface, but that is handled by the document class file. Take care with the use of[4] the curly braces in typeface changes; they

[1] A website may use more than one server-side programming language

[2] This is the second footnote. It starts a series of three footnotes that add nothing informational, but just give an idea of how footnotes work and look. It is a wordy one, just so you see how a longish one plays out.

[3] A third footnote, here. Let's make this a rather short one to see how it looks.

[4] A fourth, and last, footnote.

Table 3: Frequency of Special Characters

Non-English or Math	Frequency	Comments
Ø	1 in 1,000	For Swedish names
π	1 in 5	Common in math
\$	4 in 5	Used in business
Ψ_1^2	1 in 40,000	Unexplained usage

Figure 1: A sample black and white graphic.

mark the beginning and end of the text that is to be in the different typeface.

You can use whatever symbols, accented characters, or non-English characters you need anywhere in your document; you can find a complete list of what is available in the *LaTeX User's Guide*[**?**].

3.2 Tables

Because tables cannot be split across pages, the best placement for them is typically the top of the page nearest their initial cite. To ensure this proper "floating" placement of tables, use the environment **table** to enclose the table's contents and the table caption. The contents of the table itself must go in the **tabular** environment, to be aligned properly in rows and columns, with the desired horizontal and vertical rules. Again, detailed instructions on **tabular** material is found in the *LaTeX User's Guide*.

Immediately following this sentence is the point at which Table 1 is included in the input file; compare the placement of the table here with the table in the printed dvi output of this document.

To set a wider table, which takes up the whole width of the page's live area, use the environment **table*** to enclose the table's contents and the table caption. As with a single-column table, this wide table will "float" to a location deemed more desirable. Immediately following this sentence is the point at which Table 2 is included in the input file; again, it is instructive to compare the placement of the table here with the table in the printed dvi output of this document.

3.3 Figures

Like tables, figures cannot be split across pages; the best placement for them is typically the top or the bottom of the page nearest their initial cite. To ensure this proper "floating" placement of figures, use the environment **figure** to enclose the figure and its caption.

This sample document contains examples of **.eps** files to be displayable with LaTeX. If you work with pdfLaTeX, use files in the **.pdf** format. Note that most modern TeX system will convert **.eps** to **.pdf** for you on the fly. More details on each of these is found in the *Author's Guide*.

As was the case with tables, you may want a figure that spans two columns. To do this, and still to ensure proper "floating" placement of tables, use the environment **figure*** to enclose the figure and its caption. and don't forget to end the environment with figure*, not figure!

Figure 2: A sample black and white graphic that has been resized with the `includegraphics` command.

3.4 Theorem-like Constructs

Other common constructs that may occur in your article are the forms for logical constructs like theorems, axioms, corollaries and proofs. There are two forms, one produced by the command `\newtheorem` and the other by the command `\newdef`; perhaps the clearest and easiest way to distinguish them is to compare the two in the output of this sample document:

This uses the **theorem** environment, created by the `\newtheorem` command:

THEOREM 1. *Let f be continuous on $[a,b]$. If G is an antiderivative for f on $[a,b]$, then*

$$\int_a^b f(t)dt = G(b) - G(a).$$

The other uses the **definition** environment, created by the `\newdef` command:

Definition 1. If z is irrational, then by e^z we mean the unique number which has logarithm z:

$$\log e^z = z$$

Two lists of constructs that use one of these forms is given in the *Author's Guidelines.*

There is one other similar construct environment, which is already set up for you; i.e. you must *not* use a `\newdef` command to create it: the **proof** environment. Here is a example of its use:

PROOF. Suppose on the contrary there exists a real number L such that

$$\lim_{x\to\infty} \frac{f(x)}{g(x)} = L.$$

Then

$$l = \lim_{x\to c} f(x) = \lim_{x\to c} \left[gx \cdot \frac{f(x)}{g(x)} \right] = \lim_{x\to c} g(x) \cdot \lim_{x\to c} \frac{f(x)}{g(x)} = 0 \cdot L = 0,$$

which contradicts our assumption that $l \neq 0$. □

Complete rules about using these environments and using the two different creation commands are in the *Author's Guide*; please consult it for more detailed instructions. If you need to use another construct, not listed therein, which you want to have the same formatting as the Theorem or the Definition[**?**] shown above, use the `\newtheorem` or the `\newdef` command, respectively, to create it.

A *Caveat* for the TeX Expert

Because you have just been given permission to use the `\newdef` command to create a new form, you might think you can use TeX's `\def` to create a new command: *Please refrain from doing this!* Remember that your LaTeX source code is primarily intended to create camera-ready copy, but may be converted to other forms – e.g. HTML. If you inadvertently omit some or all of the `\def`s recompilation will be, to say the least, problematic.

4. CONCLUSIONS

This paragraph will end the body of this sample document. Remember that you might still have Acknowledgments or Appendices; brief samples of these follow. There is

Table 4: Some Typical Commands

Command	A Number	Comments
`\alignauthor`	100	Author alignment
`\numberofauthors`	200	Author enumeration
`\table`	300	For tables
`\table*`	400	For wider tables

Figure 3: A sample black and white graphic that needs to span two columns of text.

still the Bibliography to deal with; and we will make a disclaimer about that here: with the exception of the reference to the LaTeX book, the citations in this paper are to articles which have nothing to do with the present subject and are used as examples only.

5. ACKNOWLEDGMENTS

This section is optional; it is a location for you to acknowledge grants, funding, editing assistance and what have you. In the present case, for example, the authors would like to thank Gerald Murray of ACM for his help in codifying this *Author's Guide* and the **.cls** and **.tex** files that it describes.

References

[1] S. W. Boyd and A. D. Keromytis. Sqlrand: Preventing sql injection attacks. In *Applied Cryptography and Network Security*, pages 292–302. Springer, 2004.

[2] 2010.

[3] W. G. Halfond, J. Viegas, and A. Orso. A classification of sql-injection attacks and countermeasures. In *Proceedings of the IEEE International Symposium on Secure Software Engineering*, volume 1, pages 13–15. IEEE, 2006.

[4] A. Herzog. Full Disclosure: JavaMail SMTP Header Injection via method setSubject [CSNC-2014-001], 2014.

[5] H. H. Injection. HTTP header injection — Wikipedia, the free encyclopedia. http://en.wikipedia.org/w/index.php?title=HTTP%20header%20injection&oldid=713295668, 2016. [Online; accessed 18-April-2016].

[6] T. Jim, N. Swamy, and M. Hicks. Defeating script injection attacks with browser-enforced embedded policies. In *Proceedings of the 16th International Conference on World Wide Web*, WWW '07, pages 601–610, New York, NY, USA, 2007. ACM.

[7] M. Johns and J. Winter. Requestrodeo: Client side protection against session riding. In *Proceedings of the OWASP Europe 2006 Conference*, 2006.

[8] A. Klein. [DOM Based Cross Site Scripting or XSS of the Third Kind] Web Security Articles - Web Application Security Consortium, 2005.

[9] D. Kohler. damonkohler.com: Email Injection, 2008.

[10] OWASP. OWASP Top Ten Project, 2013.

[11] Phishing. Phishing — Wikipedia, the free encyclopedia. http://en.wikipedia.org/w/index.php?title=Phishing&oldid=706223617, 2016. [Online; accessed 27-February-2016].

[12] PHP-Manual. PHP mail - Send mail, 2016.

[13] A. Sadeghian, M. Zamani, and A. A. Manaf. A taxonomy of sql injection detection and prevention techniques. In *Informatics and Creative Multimedia (ICICM), 2013 International Conference on*, pages 53–56. IEEE, 2013.

[14] D. Stuttard and M. Pinto. *The Web Application Hacker's Handbook: Finding and Exploiting Security Flaws.* John Wiley & Sons, 2011.

[15] Tobozo. Mail headers injections with PHP, 2004.

[16] W3techs. Usage Statistics and Market Share of PHP for Websites, February 2016, 2016.

[17] Wikipedia. Spoofing attack — Wikipedia, the free encyclopedia. http://en.wikipedia.org/w/index.php?title=Spoofing%20attack&oldid=711407565, 2016. [Online; accessed 23-April-2016].

APPENDIX

A. HEADINGS IN APPENDICES

The rules about hierarchical headings discussed above for the body of the article are different in the appendices. In the **appendix** environment, the command **section** is used to indicate the start of each Appendix, with alphabetic order designation (i.e. the first is A, the second B, etc.) and a title (if you include one). So, if you need hierarchical structure *within* an Appendix, start with **subsection** as the highest level. Here is an outline of the body of this document in Appendix-appropriate form:

A.1 Introduction

A.2 The Body of the Paper

A.2.1 Type Changes and Special Characters

A.2.2 Math Equations

Inline (In-text) Equations.

Display Equations.

A.2.3 Citations

A.2.4 Tables

A.2.5 Figures

A.2.6 Theorem-like Constructs

A Caveat for the TEX Expert

A.3 Conclusions

A.4 Acknowledgments

A.5 Additional Authors

This section is inserted by LATEX; you do not insert it. You just add the names and information in the `\additionalauthors` command at the start of the document.

A.6 References

Generated by bibtex from your .bib file. Run latex, then bibtex, then latex twice (to resolve references) to create the .bbl file. Insert that .bbl file into the .tex source file and comment out the command `\thebibliography`.

B. MORE HELP FOR THE HARDY

The sig-alternate.cls file itself is chock-full of succinct and helpful comments. If you consider yourself a moderately experienced to expert user of LATEX, you may find reading it useful but please remember not to change it.