E-Mail Header Injections

An Analysis of the World Wide Web

by

Sai Prashanth Chandramouli

A Thesis Presented in Partial Fulfillment
of the Requirement for the Degree
Master of Science

Approved April 2016 by the
Graduate Supervisory Committee:

Dr. Adam Doupe, Chair
Dr. Gail-Joon Ahn
Dr. Ziming Zhao

ARIZONA STATE UNIVERSITY

May 2016

ABSTRACT

E-mail header injection vulnerability is a class of vulnerability that has been around for a long time but has not made its way to popular literature. It can be considered as the email equivalent of HTTP Header Injection Vulnerability. Email injection is possible when the mailing script fails to check for the presence of email headers in the form fields that take in email addresses. The vulnerability exists in the reference implementation of the builtin mail functionality in popular languages like PHP, Java and python.With the proper injection string, this vulnerability can be exploited to inject additional headers and/or modify existing headers in an E-mail message.

To understand and quantify the prevalence of E-Mail Header Injection vulnerabilities, we used a black-box testing approach, where we crawled 'x' URLs in order to find the URLs which contained form fields. Our system used this data feed to classify the forms which had e-mail fields which could be fuzzed with malicious payloads. Amongst the 's' forms fuzzed, our system was able to find 'y' vulnerable URLs among 'z' domains, which proves that the threat is/isn't widespread and deserves future research attention.

To my mother and father, for giving me the life I dreamt of,

To my sister, who constantly made me do better just to keep up with her,

To my family in Phoenix, for always being there,

To God, for making me so lucky, for letting me be strong when I had nothing, and making me believe when no one else would have.

ACKNOWLEDGEMENTS

A project of this size is never easy to complete without the help and support of other people. I would like to take this opportunity to thank some of them.

This thesis would not have been possible without the help and guidance of my thesis advisor, and committee chair - the brilliant Dr. Adam Doupe. This project was his brainchild, and he held my hand through the entire project. Thank you for everything you did, Adam.

I would like to thank Dr. Gail-Joon Ahn, for being part of the committee, for all his help, and his valuable input on the changes to be made to make the project more impactful.

I would also like to thank Dr. Ziming Zhao, for being a part of my committee, and for the constant motivation.

I would like to thank the members of the SEFCOM, for their support, and would like to especially thank Mike Mabey, for helping set up the infrastructure for this project, and Marthony Taguinod, for helping me document this project.

TABLE OF CONTENTS

CHAPTER Page

LIST OF TABLES

Table Page

LIST OF FIGURES

Chapter 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 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 taking over just a single E-Mail account of an unsuspecting end user is of a large magnitude.

Since attackers are typically 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 like the ever popular SQL injection or cross-site scripting (XSS) OWASP (2013) 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. The vulnerability exists in the reference implementation of the built-in mail functionality in popular languages like PHP, Java and Python. With the proper injection string, this vulnerability can be exploited to inject additional headers and/or modify existing headers in an E-Mail message.

E-Mail Header Injection attacks have the potential to allow an attacker to perform E-Mail spoofing, resulting in vicious Phishing attacks that can lead to identity theft. The objective of our research is to find if this vulnerability is widespread on the World Wide Web, and whether further research is required in this area.

In order to do this, we performed an expansive crawl of the web, extracting forms that had E-Mail fields in them, and then injecting them with different payloads. We then audited the received emails to see if any of the injected data was present. This allowed us to classify whether a particular URL was vulnerable to the attack. This entire system works in a black-box manner, without looking at the web application's source code, and only analyzing the emails 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 proceeds onto enumerate the languages and platforms affected by this vulnerability.

- Chapter 3 discusses the System design, and enunciates the architecture and the components of the system, along with a detailed test plan to validate the system. It also enumerates the issues faced, and the assumptions made.

- Chapter 4 briefly describes the experimental setup and sheds light on how we overcame the issues and assumptions discussed in the previous section.

- Chapter 5 presents our findings, and our analysis of the said findings.

- 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, and clearly shows how and why our research is different.
- Chapter 8 wraps up the document, with ideas to expand the research in this area.

We hope that our research sheds some light on this relatively less popular 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, that 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 expansive crawl across the Web, including 'x' URLs and 'y' forms.

Chapter 2

E-MAIL HEADER INJECTION BACKGROUND

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 (Halfond *et al.* (2006), Boyd and Keromytis (2004), Sadeghian *et al.* (2013)), cross-site scripting (XSS) (Jim *et al.* (2007) Klein (2005)) or even HTTP Header Injection (Johns and Winter (2006)), 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 mailing script fails to check for the presence of E-Mail headers in the form fields that take in user input to send E-Mails, a malicious user, using a well-crafted payload, can control the headers set for this particular E-Mail. Suffice it to say, that this can be leveraged to do a host of 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 (Tobozo (2004)) accredited to user tobozo@phpsecure.info describing how this vulnerability existed on the reference implementation of the mail function in PHP, and how it can be exploited. More recently, a blog post by Damon Kohler (Kohler (2008)) and an accompanying wiki article (Email Injection - Secure PHP Wiki (2010)) describe the attack vector, and outline a few defense measures for the same.

Since this vulnerability was initially found in the *mail()* function of PHP, E-Mail Header Injection can be traced to as early as early 2000's, present in the *mail()* implementation of PHP 4.0. It is to be noted that after 13 further iterations of the language (the current version is 7.1), the *mail()* function is yet to be fixed.

The vulnerability was also described very briefly (less than a page) by Stuttard and Pinto in their widely acclaimed book, "*The Web Application Hacker's Handbook: Discovering and Exploiting Security Flaw*" (Stuttard and Pinto (2011)). The closest published works on this topic are a whitepaper for MBSD by Terada [6], which describes a similar vulnerability, namely SMTP Injection, to attack underlying SMTP servers, and by Teodoro et al. [7], where the vulnerability is briefly discussed along with other vulnerabilities. A timeline of the vulnerability is presented in Table 2.1.

Year	Notes
Early 2000's	PHP 4.0 gets released, along with support for the mail() function, which has no protection against E-Mail Header Injection.
Jul 2004	Next Major version of PHP - Version 5.0 releases
Dec 2004	First known article about the vulnerability surfaces on phpsecure.info

Table 2.1: A brief history of E-Mail Header Injection

2.3 Languages Affected

This section describes the popular languages which exhibit this type of vulnerability.

- PHP - Describe which functions/params are affected
- Java - Describe which functions/params are affected
- Python - Describe which functions/params are affected

2.4 Potential Impact

This section describes the impact of the vulnerability, and how wide/far-reaching the effects could be.

Chapter 3

SYSTEM DESIGN

3.1 Our Approach to the Problem

This section will describe the approach we have taken. Will discuss about blackbox testing, and why we chose it.

3.2 System Architecture

This will have a diagram of our architecture, including all 8 components.

3.3 System Components

This will discuss in detail about the components of the system, like the following:

3.3.1 Crawler

Describe the functionality of the Crawler

3.3.2 Form Parser

Describe the functionality of the Form Parser

3.3.3 E-Mail Field Checker

Describe the functionality of the E-Mail Field Checker

3.3.4 Fuzzer

Describe the functionality of the Fuzzer

Non-Malicious Payload Describes what the regular payload is.

Malicious Payload Describes what the malicious payloads are.

3.3.5 E-Mail Analyzer

Describe the functionality of the E-Mail Analyzer

3.3.6 Database

3.4 Test Plan

This section will describe the test plan for the project, and will explain what was tested, and how our system conforms to the requirements.

3.5 Design Issues

This section will describe the issues we might face with the approach that we have chosen, and the design decisions.

3.6 Assumptions

This discusses the assumptions that we have made while building the system, examples include:

1. Crawler is not blocked by the firewalls.
2. The Crawler feed is an ideal representation of the World Wide Web.

Chapter 4

EXPERIMENTAL SETUP

4.1 System Configuration

Will briefly describe the servers used for the experiments.

4.2 Platforms and Software

Will briefly describe the platform, (ie) Ubuntu 14.04, and the softwares that were used for the experiments. (eg) Postfix, Apache, MySQL, etc.

4.3 Languages used

Will very briefly (maybe one paragraph) describe what we used to create the system. (Python 2) Will also describe the limitation of Python, (GIL basically), and point to next section.

4.4 Celery Queues

Will briefly describe how Celery and rabbitMQ help us to overcome the GIL, and do tasks in parallel.

Chapter 5

DATA ANALYSIS AND RESULTS

This section will have tables, images and charts.

5.1 Data

Will display a table/graph with the data, then go on to explain what the fields/-graphs mean.

5.1.1 URLS crawled

5.1.2 Forms collected

5.1.3 Forms with E-Mail Fields

5.1.4 E-Mail received from Forms

5.1.5 Fuzzed Forms

Chapter 6

DISCUSSION

6.1 Lessons Learned

Describes what we learned from this particular project.

6.2 Limitations of the Project

Describes what limitations were present, stuff like:

- CAPTCHAs
- JavaScript Apps
- Blogs powered by WordPress/Drupal
- Mail libraries

6.3 How to prevent this attack

Describes how to prevent this attack, stuff like:

- Use Mail Libraries
- CMS
- Input Validation

Chapter 7

RELATED WORK

This will be a detailed section on the papers that are related to our work, *but* important thing is to show why our work is different from prior work in this area. Also, can/will add references to the blogs and books that describe this attack :)

Chapter 8

CONCLUSION

Conclude with what the results were, whether the vulnerability was widespread or not, and how (if needed) this can be alleviated.

REFERENCES

Beizer, B., *Black-box Testing: Techniques for Functional Testing of Software and Systems* (John Wiley & Sons, Inc., New York, NY, USA, 1995).

Boyd, S. W. and A. D. Keromytis, "Sqlrand: Preventing sql injection attacks", in "Applied Cryptography and Network Security", pp. 292–302 (Springer, 2004).

Christey, S. and R. A. Martin, "Vulnerability type distributions in cve", Mitre report, May (2007).

Doupé, A., B. Boe, C. Kruegel and G. Vigna, "Fear the EAR : Discovering and Mitigating Execution After Redirect Vulnerabilities", in "Computer and Communications Security (CCS)", CCS '11, pp. 251–261 (ACM Press, 2011).

Email Injection - Secure PHP Wiki, URL `http://securephpwiki.com/index.php/EmailInjection` (2010).

Franklin, J., A. Perrig, V. Paxson and S. Savage, "An inquiry into the nature and causes of the wealth of internet miscreants.", in "ACM conference on Computer and communications security", pp. 375–388 (2007).

Halfond, W. G., J. Viegas and A. Orso, "A classification of sql-injection attacks and countermeasures", in "Proceedings of the IEEE International Symposium on Secure Software Engineering", vol. 1, pp. 13–15 (IEEE, 2006).

Herzog, A., "Full Disclosure: JavaMail SMTP Header Injection via method setSubject [CSNC-2014-001]", URL `http://seclists.org/fulldisclosure/2014/May/81` (2014).

Hodges, J., C. Jackson and A. Barth, "Http strict transport security (hsts)", URL: http://tools. ietf. org/html/draft-ietf-websec-strict-transport-sec-04 (2012).

Jim, T., 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, pp. 601–610 (ACM, New York, NY, USA, 2007), URL `http://doi.acm.org/10.1145/1242572.1242654`.

Johns, M. and J. Winter, "Requestrodeo: Client side protection against session riding", in "Proceedings of the OWASP Europe 2006 Conference", (2006).

Klein, A., "[DOM Based Cross Site Scripting or XSS of the Third Kind] Web Security Articles - Web Application Security Consortium", URL `http://www.webappsec.org/projects/articles/071105.shtml` (2005).

Kohler, D., "damonkohler.com: Email Injection", URL `http://www.damonkohler.com/2008/12/email-injection.html` (2008).

Murray, D., "Email.header.Header too lax with embedded newlines", URL `http://bugs.python.org/issue5871` (2009).

OWASP, "OWASP Top Ten Project", URL https://www.owasp.org/index.php/OWASP_Top_10 (2013).

Payet, P., A. Doupé, C. Kruegel and G. Vigna, "EARs in the Wild: Large-Scale Analysis of Execution After Redirect Vulnerabilities", in "Proceedings of the ACM Symposium on Applied Computing (SAC)", (Coimbra, Portugal, 2013).

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

PHP-Manual, "PHP mail - Send mail", URL http://php.net/manual/en/function.mail.php (2016).

Pietraszek, T. and C. V. Berghe, "Defending against injection attacks through context-sensitive string evaluation", in "Recent Advances in Intrusion Detection", pp. 124–145 (Springer, 2005).

Pope, A., "Prevent Contact Form Spam Email Header Injection — Storm Consultancy Web Design Bath", URL https://www.stormconsultancy.co.uk/blog/development/dev-tutorials/secure-your-contact-form-against-spam-email-header-injection/ (2008).

Python, "email - an email and mime handling package", URL https://docs.python.org/2/library/email-examples.html (2016).

Sadeghian, A., 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", pp. 53–56 (IEEE, 2013).

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

Terada, T., "SMTP Injection via recipient email addresses", MBSD White Paper (2015).

Tobozo, "Mail headers injections with PHP", URL http://www.phpsecure.info/v2/article/MailHeadersInject.en.php (2004).

Zanero, S., L. Carettoni and M. Zanchetta, "Automatic detection of web application security flaws", Black Hat Briefings (2005).

APPENDIX A

CODE SNIPPETS