Measuring E-Mail Header Injections on the World Wide Web

Sai Prashanth Chandramouli[1], Pierre-Marie Bajan[2], Christopher Kruegel[3], Giovanni Vigna[3], Ziming Zhao[1], Adam Doupé[1], and Gail-Joon Ahn[1]

[1] Arizona State University
e-mail{saipc, zzhao30, doupe, gahn}@asu.edu
[2] IRT SystemX
e-mailpierre-marie.bajan@irt-systemx.fr
[3] University of California, Santa Barbara
e-mail{chris, vigna}@cs.ucsb.edu

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 Header Injection is possible when the script fails to check for the presence of e-mail headers in user input. We discovered that the vulnerability exists in the built-in e-mail functionality of the popular languages PHP, Java, Python, and Ruby. With the proper injection string, this vulnerability can be exploited to allow an attacker to inject additional headers, modify existing headers, and alter the content of the e-mail.

While E-mail Header Injection vulnerabilities are known by the community, and some commercial vulnerability scanners claim to discover E-mail Header Injection vulnerabilities, they have never been studied by the academic community. This paper presents a scalable mechanism to automatically detect E-mail Header Injection vulnerabilities and uses this mechanism to quantify the prevalence of E-mail Header Injection vulnerabilities on the web. Using a black-box testing approach, the system crawled 23,553,796 URLs to identify web pages which contained form fields. 7,354,425 such forms were found by the system, of which 1,228,774 forms contained e-mail fields. We then fuzzed 1,012,530 forms to see if they would send us an e-mail, and 74,192 forms sent us an e-mail. Of these, 64,510 forms were tested with E-mail Header Injection payloads and, of these, we found 994 vulnerable URLs across 414 domains. Then, to measure if E-mail Header Injection vulnerabilities are actively being exploited to create a spamming platform, we found evidence that 157 IPs that were vulnerable to E-mail Header Injection are on spamming blacklists. We also performed a virus check on the received e-mails and found 265 domains to be sending malicious content. This work shows that E-mail Header Injection vulnerabilities are widespread and deserve future research attention.

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 made it possible for any-

one to access information and services anywhere and on multiple devices such as phones, laptops, TVs, and cars. This access has ushered in an era of web applications which depend on user input. While the rapid pace of development has improved the speed of information dissemination, it comes at a cost. As users move more and more of their personal and financial information to web applications, attackers are responding by using web application vulnerabilities to steal lucrative data.

Many common and well-known web application vulnerabilities, such as SQL Injection and Cross-Site Scripting [2], are command injection vulnerabilities [4], where malicious user input is used to alter the structure of a command (SQL query and JavaScript code respectively). Developers of web applications must use the correct sanitization routine in all paths leading from user input to a command.

E-mail Header Injection vulnerabilities are a lesser-known command injection vulnerability. We verified that this vulnerability exists in the implementation of the built-in e-mail functionality in the popular languages PHP, Java, Python, and Ruby. The format of e-mail messages is defined by the Simple Mail Transfer Protocol (SMTP) [3]. Each e-mail message is represented by a series of headers separated by newlines, followed by the body content (separated from the headers by two newlines). Some of these headers are mandatory (From, To, Date), but the headers could also include other information such as the Subject, CC, BCC, etc.

With the proper injection string, E-mail Header Injection vulnerabilities can be exploited by an attacker to inject additional headers, modify existing headers, or alter the contents of the e-mail—while still appearing to be from a legitimate source. E-mail Header Injection exploits allow an attacker to perform e-mail spoofing, resulting in phishing attacks *that are sent from the actual e-mail server*.

While some command injection vulnerabilities have received extensive attention from the research community, E-mail Header Injection vulnerabilities have received little focus. In fact, the Acunetix vulnerability scanner contains an AcuMonitor component which claims to detect E-mail Header Injection vulnerabilities while scanning [1]. Unfortunately, as a commercial product, little is known about how AcuMonitor detects E-mail Header Injection vulnerabilities.

To shed light on this little-studied vulnerability class, we describe E-mail Header Injection vulnerabilities and measure E-mail Header Injection vulnerabilities. To perform this measurement, we crawled the web, extracted forms with e-mail fields, and injected them with different payloads to infer the existence of an 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. Our automated system works in a black-box manner, without looking at the web application's source code, and only analyzes the payloads in the e-mails.

In summary, we make the following contributions:

– We develop a black-box approach to detect E-mail Header Injection vulnerabilities in a web application.

– We develop an open-source system to crawl the web and automatically detect E-mail Header Injection vulnerabilities.
– We use our system to crawl 23,553,796 URLs, and we find 994 URLs vulnerable to E-mail Header Injection across 414 domains.

2 Other Sections

Other sections are here.

3 Conclusion

Conclusions are here.

Acknowledgments

Authors would like to thank YYYYY.

4 Footnotes

Footnotes within the text should be coded:

`\footnote{Text}`

Sample Input

Text with a footnote`\footnote{The footnote is automatically numbered.}` and text continues ...

Sample Output

Text with a footnote[4] and text continues ...

5 Tables

Table captions should be treated in the same way as figure legends, except that the table captions appear *above* the tables. The tables will be numbered automatically.

5.1 Tables Coded with LaTeX

Sample Output

Table 1. Critical N values

$M_\odot$	β_0	T_{c6}	γ	N_{crit}^{L}	N_{crit}^{Te}
30	0.82	38.4	35.7	154	320
60	0.67	42.1	34.7	138	340
120	0.52	45.1	34.0	124	370

References

1. Acunetix: AcuMonitor: For detecting Email Header Injection, Blind XSS and SSRF - Acunetix. `http://www.acunetix.com/vulnerability-scanner/acumonitor-blind-xss-detection/`
2. OWASP: `https://www.owasp.org/index.php/OWASP_Top_10`
3. Resnick, P.W.: Internet Message Format - RFC 5322 (2008), `https://tools.ietf.org/html/rfc5322`
4. Su, Z., Wassermann, G.: The essence of command injection attacks in web applications. In: ACM SIGPLAN Notices. vol. 41, pp. 372–382. ACM (2006)

[4] The footnote is automatically numbered.