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
\memberThree
\memberFour

ARIZONA STATE UNIVERSITY

May 2016

ABSTRACT

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.

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

LIST OF TABLES

Table Page

LIST OF FIGURES

Chapter 1

INTRODUCTION

Franklin *et al.* (2007)

Chapter 2

E-MAIL HEADER INJECTION BACKGROUND

2.1 Problem Background

This section describes the background of the vulnerability.

2.2 History of E-Mail Injection

This section describes the history of the vulnerability.

2.3 Languages Affected

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

- PHP
- Java
- Python

2.4 Potential Impact

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

Chapter 3

WEB APPLICATIONS

3.1 Web Applications: Individual and Commercial Front Doors

Importance of Security Area Web applications continue to remain as the most popular method for businesses to conduct services over the Internet. As the number of web applications that are accessible increase, so too does the amount of sensitive business and user data that is managed and processed by web applications. Because of their continuously increasing popularity and their inherent nature, vulnerabilities that are present in these web applications put both businesses and end-users' security and privacy at risk.

This is not an abstract risk, as the JPMorgan Chase breach in 2014 affected 76 million US households **?**. Bloomberg reported that the hackers "exploited an overlooked flaw in one of the bank's websites" **?**. Therefore, web applications serve as the "front door" for many companies and ensuring their security is of paramount importance.

Current approaches to address security Current techniques and tools focus primarily on prevention and discovery of these vulnerabilities. For instance, many techniques and tools using static analysis (white-box) or dynamic analysis (black-box) approaches have been proposed and developed to discover the vulnerabilities of web applications **?????**, so that the vulnerabilities can be removed before attackers discover and exploit them. However, the efforts of discovering and fixing vulnerabilities are not enough to protect web applications for many reasons:

1. The increasing complexity of modern web applications brings inevitable risks that cannot be fully mitigated in the process of web application development and deployment

2. Attackers are able to take their time in understanding the target web application's functionality and underlying technology stack before executing an attack.

Proposed approach We believe that a defense-in-depth approach is best in securing web applications. Therefore, to complement the aforementioned vulnerability analysis techniques, we propose to use the ideas of Moving Target Defense to create a novel and proactive approach that adds an additional layer of defense to web applications. At a high level, a Moving Target Defense dynamically configures and shifts systems over time to increase the uncertainty and complexity for attackers to perform probing and attacking **??**. While a system's availability is preserved to legitimate users, the system components are changed in unpredictable ways to the attackers. Therefore, the attacker's window of attack opportunities decrease and the costs of attack increase. Even if an attacker succeeds in finding a vulnerability at one point, the vulnerability could be unavailable as the result of shifting the underlying system, which makes the environment more resilient against attacks.

To best apply the MTD ideas to protect web applications, there are two high-level decisions:

- Deciding what web application component to move

- Choosing the optimal frequency of randomization of the chosen components

To assist in answering these questions, we first dissect the architecture of a modern web application - both client and server as well as their running environments, in order to explore the possible application of MTD at different layers. We hope our

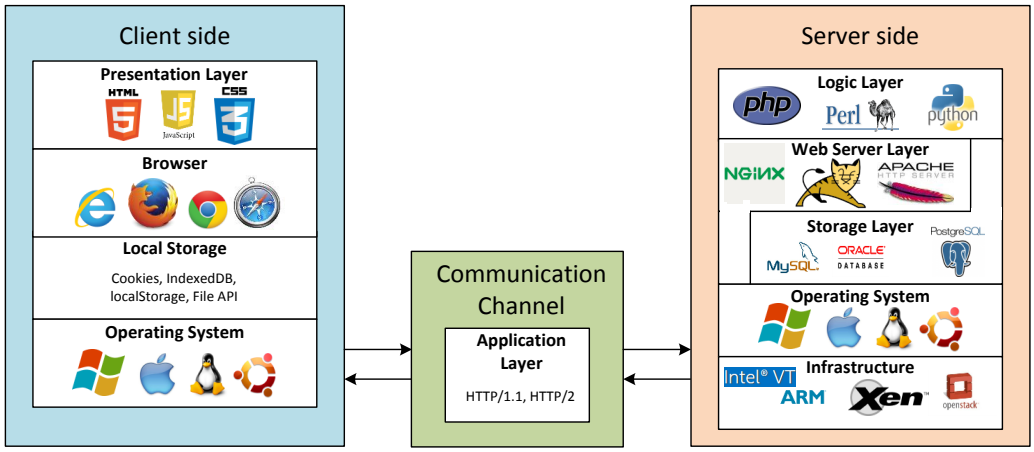

Figure 3.1: A Modern Web Application Architecture and Its Running Environments.

analysis provides insights into the trade-offs among the different places to apply MTD to web applications.

We also discuss our first steps in applying MTD techniques to protect web applications. The first technique changes the server-side language used in a web application by automatically translating server-side web application code to another language in order to prevent Code Injection exploits. The second technique shifts the database used in a web application by transforming the backend SQL database into different implementations that speak different dialects in order to prevent SQL Injection exploits.

3.2 Dissecting Modern Web Applications

Architecture overview for web applications In order to properly understand how to apply the ideas of moving target defense to web applications, we first describe a typical web application followed by a discussion on the ideas behind using moving target defense. As shown in Figure 3.1, a web application follows a distributed application structure, with components running on both server and the client systems.

Web application workflow and background When requesting a web resource, the client first issues a request to the server-side component over its communication channels - this is typically the HTTP protocol and its derivative protocols such as HTTPS, SPDY, and HTTP/2. The server receives the request and processes it using the application's logic and returns the requested resource. If data stored in an external database is requested, the server passes the relevant user-input as a query and processes the result. The server side typically includes the following layers from top to bottom [1] :

- The server-side logic layer implements the application business logic using high-level programming languages, such as Java, PHP, or Python.
- The web server layer receives the HTTP request from the client, parses the HTTP request, and passes the request to the appropriate server-side program. Examples include Apache web server, Windows IIS, or Nginx.
- The data storage layer stores the web application state and user data. Popular data storage systems are traditional SQL databases, which include MySQL, PostgreSQL, or MSSQL.
- The operating system layer that provides the running environment for the web server layer and database storage layer.
- The infrastructure layer that runs the operating systems. An infrastructure could be a physical machine or a virtualization platform which manages multiple virtual machines.

The client receives the HTTP response from the server-side component and converts

[1] Of course, modern web application stacks can become increasingly complex, with caches, external requests, or other services, however we restrict our discussion to this abstracted model.

the HTML contained in the HTTP response into a graphical interface for the user. The client consists of the following components:

- The client-side logic layer, usually known as the presentation layer. The logic code here is usually composed of a combination of HTML, CSS, and JavaScript, with JavaScript providing a way for the server-side code to execute application logic on the client.
- The browser, which retrieves the presentation layer code from the server (typically HTML), interprets it, and presents it as a graphical interface to the user.
- The storage layer, that the presentation layer code uses to store data. Available storage methods include `cookies`, `localStorage`, `IndexedDB`, and File APIs.
- The operating system layer, which the browser runs on.

3.3 Current Security Issues and Considerations

Based on our proposed definition of a web application's structure 3.1, if a layer is compromised, the upper layers are not trustworthy. For instance, if the server's operating system is compromised, then the data storage, web server, and server-side logic are also compromised due to the interconnected nature of web applications. In addition to this, if the communication channel is also compromised - i.e. Man in the Middle attack; the client side presentation layer also gets affected, as attackers are able to manipulate the information being seen by users. In order to attack a layer in Figure 3.1, adversaries often utilize interfaces exposed to the upper layers. For instance, in a heap spraying attack executed on the client browser layer **?**, an attacker allocates malicious objects using JavaScript in the presentation layer in order to coerce the browser into spraying objects in the heap, increasing the success rate

of an exploit where a vulnerability is exploited by jumping to the location within the heap. In this example, the attacker leverages a vulnerability located in the presentation layer - lack of input validation; to exploit a vulnerability in the browser layer that leads to arbitrary code execution in the browser's address space. The arbitrary code that was injected can in turn exploit a vulnerability found on the client operating system in order to escalate a malicious user's privilege and further infect the client machine. Furthermore, vulnerabilities are not isolated within each system. For example, malicious JavaScript code could be delivered by an attacker by exploiting a vulnerability in the server-side logic layer, using a reflected or stored cross-site scripting (XSS) vulnerability.

REFERENCES

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).

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).

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

APPENDIX A

RAW DATA