Attack Surface Reduction Through Software Debloating

Babak Amin Azad

Department of Computer Science

Adviser: Nick Nikiforakis

July 2019

Abstract

As software becomes increasingly complex, its attack surface expands enabling the exploitation of a wide range of vulnerabilities. Web applications are no exception since modern HTML5 standards and the ever-increasing capabilities of JavaScript are utilized to build rich web applications, often subsuming the need for traditional desktop applications. One possible way of handling this increased complexity is through the process of software debloating, i.e., the removal not only of dead code but also of code corresponding to features that a specific set of users do not require. Even though debloating has been successfully applied on operating systems, libraries, and compiled programs, its applicability on web applications has not yet been investigated.

In this report, we present the first analysis of the security benefits of debloating web applications. We focus on four popular PHP applications and we dynamically exercise them to obtain information about the server-side code that executes as a result of client-side requests. We evaluate two different debloating strategies (file-level debloating and function-level debloating) and we show that we can produce functional web applications that are 46% smaller than their original versions and exhibit half their original cyclomatic complexity. Moreover, our results show that the process of debloating removes code associated with tens of historical vulnerabilities and further shrinks a web application's attack surface by removing unnecessary external packages and abusable PHP gadgets.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

Despite its humble beginnings, the web has evolved into a full-fledged software delivery platform where users increasingly rely on web applications to replace software that traditionally used to be downloaded and installed on their devices. Modern HTML5 standards and the constant evolution of JavaScript enable the development and delivery of office suites, photo-editing software, collaboration tools, and a wide range of other complex applications, all using HTML, CSS, and JavaScript and all delivered and rendered through the user's browser.

This increase in capabilities requires more and more complex server-side and client-side code to be able to deliver the features that users have come to expect. However, as the code and code complexity of an application expands, so does its attack surface. Web applications are vulnerable to a wide range of client-side and server-side attacks including Cross-Site Scripting [2, 15, 29], Cross-Site Request Forgery [1, 14, 5], Remote Code Execution [3], SQL injection [4, 9], and timing attacks [7, 8]. All of these attacks have been abused numerous times to compromise web servers, steal user data, move laterally behind a company's firewall, and infect users with malware and cryptojacking scripts [16, 30, 11].

One possible strategy of dealing with ever-increasing software complexity is to customize software according to the environment where it is used. This idea, known as *attack-surface reduction* and *software debloating*, is based on the assumption that not all users require the same features from the same piece of software. By removing the features of different deployments of the same software according to what the users of each deployment require, one can reduce the attack surface of the program by maintaining only the features that users utilize and deem necessary. The principle of software debloating has been successfully tried on operating systems (both to build unikernel OSs [20] and to remove unnecessary code from the Linux kernel [19, 18]) and more recently on shared libraries [21, 22] and compiled binary applications [10].

In this report, we present the first evaluation of the applicability of software debloating for web applications. We focus on four popular open-source PHP applications (phpMyAdmin, MediaWiki, Magento, and WordPress) and we map the CVEs of 69 reported vulnerabilities to the source code of each web application. We utilize a combination of tutorials (encoded as Selenium scripts), monkey testing, web crawling, and vulnerability scanning to get an *objective* and *unbiased* usage profile for each application. By using these methods to stimulate the evaluated web applications in combination with dynamically profiling the execution of server-side code, we can precisely identify the code that was executed during this stimulation and therefore the code that should be retained during the process of debloating.

Equipped with these server-side execution traces, we evaluate two different debloating strategies (file-level debloating and function-level debloating) which we use to remove unnecessary code from the web applications and quantify the security benefits of this procedure. Among others, we discover an average reduction of the codebase of the evaluated web application of 33.1% for file-level debloating and 46.8% for function-level debloating, with comparable levels of reduction in the applications' cyclomatic complexity. In terms of known vulnerabilities, we remove up to 60% of

known CVEs and the vast majority of PHP gadgets that could be used in Property Oriented Programming attacks (the equivalent of Return-Oriented Programming attacks for PHP applications).

Overall, our contributions are the following:

- We encode a large number of application tutorials as Selenium scripts which, in combination with monkey testing, crawling, and vulnerability scanning, can be used to objectively exercise a web application. Similarly, we map 69 CVEs to their precise location in the applications' source code to be able to quantify whether the vulnerable code could be removed during the process of debloating.

- We design and develop an end-to-end analysis pipeline using Docker containers which can execute client-side, application stimulation, while dynamically profiling the executing server-side code.

- We use this pipeline to precisely quantify the security benefits of debloating web applications, finding that debloating pays large dividends in terms of security, by reducing a web application's source code, cyclomatic complexity, and vulnerability to known attacks.

To motivate further research into debloating web applications and to ensure the reproducibility of our findings, we are releasing *all* data and software artifacts.

Chapter 2

Related Work

Over the years, different approaches that target very different parts of the software stack have been studied in the context of software debloating.

2.0.1 Debloating for the web

Despite the importance of the web platform, there has been very little work that attempts to apply debloating to it. Snyder et al. investigated the costs and benefits of giving websites access to all available browser features through JavaScript [27]. The authors evaluated the use of different JavaScript APIs in the wild and proposed the use of a client-side extension which controls which APIs any given website would get access to, depending on that website's level of trust. Schwarz et al. similarly utilize a browser extension to limit the attack surface of Chrome and show that they are able to protect users against microarchitectural and side-channel attacks [25]. These studies are orthogonal to our work since they both focus on the client-side of the web platform, whereas we focus on the server-side web applications.

Boomsma et al. performed dynamic profiling of a custom web application (a PHP application from an industry partner) [6]. The authors measured the time it takes for their dynamic profile system to get complete coverage and the percentage of files

that they could remove. Since the application was a custom one, the authors were not able to report specifics in terms of the reduction of the programs attack surface, as that relates to CVEs. Contrastingly, by focusing on popular web applications, and utilizing function-level as well as file-level debloating, we were able to precisely quantify the reduction of vulnerabilities, both in terms of known CVEs as well as gadgets for PHP object-injection attacks.

2.0.2 Debloating in other platforms

Regehr et al. developed *C-Reduce* which is a tool that works at the source code level [24]. It performs reduction of C/C++ files by applying very specific program transformation rules. Sun et al. designed a framework called *Perses* that utilizes the grammar of any programming language to guide reduction [28]. Its advantage is that it does not generate syntactically invalid variants during reduction so that the whole process is made faster.

Heo et al. worked on *Chisel* whose distinguishing feature is that it performs fine-grained debloating by removing code even on the functions that are executed, using reinforcement learning to identify the best reduced program [10].

All three aforementioned approaches are founded on Delta debugging [31]. They reduce the size of an application progressively and verify at each step if the created variant still satisfies the desired properties.

Sharif et al. proposed *Trimmer*, a system that goes further than simple static analysis [26]. It propagates the constants that are defined in program arguments and configuration files so that it can remove code that is not used in that particular execution context. However, their system is not particularly well suited for web applications where we remove complete features. Our framework goes beyond this contextual analysis by mapping what is actually executed by the application.

Other works include research that revolves mainly around static analysis to remove dead code. Jiang et al. looked at reducing the bloat of Java applications with a tool called *JRed* [12]. Jiang et al. also designed *RedDroid* to reduce the size of Android applications with program transformations [13]. Quach et al. adopted a different approach by bringing dead-code elimination benefits of static linking to dynamic linking [22].

Rastogi et al. looked at debloating a container by partitioning it into smaller and more secure ones [23]. They perform dynamic analysis on system-call logs to determine which components and executables are used in a container, in order to keep them. Koo et al. proposed configuration-driven debloating [17]. Their system removes unused libraries loaded by applications under a specific configuration. They test their system on Nginx, VSFTPD, and OpenSSH and show a reduction of 78% of code from Nginx libraries is possible based on specific configurations.

Chapter 3

Design & Methodology

To start, in your main .tex file, use this class as your main documentclass instead of 'report' or 'book'. For example:

$$\backslash documentclass[12pt, lot, lof]\{puthesis\}$$

In this example, we setup our document to use the PU Thesis style, with 12pt font for body text, and to include a List of Tables and List of Figures in the front matter. You could instead set an 11 point or 10 point font by changing the first option. You can also add 'los' to include a list of symbols.

To use single spacing, add the option 'singlespace'. This is a special option for the `puthesis` documentclass, which sets single spacing for both the front matter and for the document itself. Additional parameters should be set in your main .tex file, and are described in detail in Section 3.1.

The template itself declares two other options, to be set immediately after the `documentclass` command. First is 'printmode', declared with the command:

$$\backslash newcommand\{\backslash printmode\}\{\}$$

This command, used later in the thesis.tex file, turns off the `hyperref` package and all internal links in the PDF file. This removes any colored links and highlighting that

would not be appropriate in a printed and bound thesis. Instead the `url` package is loaded, so that
url commands in your document will continue to work and urls will break properly across multiple lines.

When 'printmode' is not specified, the hyperref package is included. It creates colored links for citations, footnotes, and internal references, which can be used to navigate the PDF document more easily. It also adds bookmarks to the PDF file, mirroring the table of contents. By default, it is set to use colored links. For the PDF file that you will submit electronically to ProQuest, this may not be desirable since some copies may be printed, while others will be used electronically. Thus another option, 'proquestmode', is defined that keeps hyperref but disables colored links:

$\newcommand{\proquestmode}{}$

This mode has no effect when used in combination with 'printmode'.

3.1 Options

In this section, we describe the options you can set when using this thesis class.

Table 3.1: List of options for the puthesis document class and template

Option	**Description**
12pt	Specify the font size for body text as a parameter to `documentclass`. The Mudd Library requirements [**?**] state that 12pt is preferred for serif fonts (e.g., Times New Roman) and 10pt for sans-serif fonts (e.g., Arial).
letterpaper	If your document is coming out in a4paper, your LaTeX defaults may be wrong. Set this option as a parameter to `documentclass` to have the correct 8.5"x11" paper size.
lot	Set this option as a parameter to `documentclass` to insert a List of Tables after the Table of Contents.

(Continued on next page)

Table 3.1: (continued)

Option	Description
lof	Set this option as a parameter to `documentclass` to insert a List of Figures after the Table of Contents and the List of Figures.
los	Set this option as a parameter to `documentclass` to insert a List of Symbols after the Table of Contents and the other lists.
singlespace	Set this option as a parameter to `documentclass` to single space your document. Double spacing is the default otherwise, and is required for the electronic copy you submit to ProQuest. Single spacing is permitted for the printed and bound copies for Mudd Library.
draft	Set this option as a parameter to `documentclass` to have LaTeXmark sections of your document that have formatting errors (e.g., overfull hboxes).
newcommand{*printmode*}{}	Insert this command after the `documentclass` command to turn off the hyperref package to produce a PDF suitable for printing.
newcommand{*proquestmode*}{}	Insert this command after the `documentclass` command to turn off the 'colorlinks' option to the hyperref package. Links in the pdf document will then be outlined in color instead of having the text itself be colored. This is more suitable when the PDF may be viewed online or printed by the reader.
makefrontmatter	Insert this command after the *begin*{*document*}* command, but before including your chapters to insert the Table of Contents and other front matter.
title	Set the title of your dissertation. Used on the title page and in the PDF properties.
submitted	Set the submission date of your dissertation. Used on the title page. This should be the month and year when your degree will be conferred, generally only January, April, June, September, or November. Check the Mudd Library rules [**?**] for the appropriate deadlines.
copyrightyear	Set the submission year of your dissertation. Used on the copyright page.
author	Your full name. Used on the title page, copyright page, and the PDF properties.

(Continued on next page)

Table 3.1: (continued)

Option	Description
$\backslash adviser$	Your adviser's full name. Used on the title page.
$\backslash departmentprefix$	The wording that precedes your department or program name. Used on the title page. The default is "Department of", since most people list their department and can leave this out (e.g., Department of Electrical Engineering), however if yours is a program, set $\backslash departmentprefix\{Programin\}$
$\backslash department$	The name of your department or program. Used on the title page.
$\backslash renewcommand$ $\{\backslash maketitlepage\}\{\}$	Disable the insertion of the title page in the front matter. This is useful for early drafts of your dissertation.
$\backslash renewcommand*$ $\{\backslash makecopyrightpage\}\{\}$	Disable the insertion of the copyright page in the front matter. This is useful for early drafts of your dissertation.
$\backslash renewcommand*$ $\{\backslash makeabstract\}\{\}$	Disable the insertion of the abstract in the front matter. This is useful for early drafts of your dissertation.

I've seen other people print their dissertations using $\backslash pagestyle\{headings\}$, which places running headings on the top of each page with the chapter number, chapter name, and page number. This documentclass is not currently compatible with this option – the margins are setup to be correct with page numbers in the footer, placing them 3/4" from the edge of the paper, as required. If you wish to use headings, you will need to adjust the margins accordingly.

Chapter 4

Results & Conclusion

In this work, we explain how to use the puthesis.cls class file and the accompanying template.

4.1 Future Work

Future work should include options in the template for a masters thesis or an undergraduate senior thesis. It should also support running headings in the headers using the 'headings' pagestyle. The print mode and proquest mode included in the template might also be candidates to include in the class itself.

Appendix A

Implementation Details

Appendices are just chapters, included after the *appendix* command.

A.1 Switching Formats

When switching `printmode` on and off (see Section 3.1), you may need to delete the output .aux files to get the document code to compile correctly. This is because the hyperref package is switched off for `printmode`, but this package inserts extra tags into the contents lines in the auxiliary files for PDF links, and these can cause errors when the package is not used.

A.2 Long Tables

Long tables span multiple pages. By default they are treated like body text, but we want them to be single spaced all the time. The class therefore defines a new command, *tablespacing*, that is placed before a long table to switch to single spacing when the rest of the document is in double spacing mode. Another command, *bodyspacing*, is placed after the long table to switch back to double spacing. Normal

tables using `tabular` automatically use single spacing and do not require the extra commands.

When the documentclass is defined with the 'singlespace' option, these commands are automatically adjusted to stay in single spacing after the long table.

Make sure there is always at least one blank line after the *\bodyspacing* command before the end of the file.

Some times long tables do not format correctly on the first pass. If the column widths are wrong, try running the LaTeXcompiler one or two extra times to allow it to better calculate the column widths.

If you want your long table to break pages at a specific point, you can insert the command *\pagebreak*[4], to tell LaTeXthat it really should put a page break there. *\pagebreak*[2] gives it a hint that this is a good place for a page break, if needed. If there's a row that really should not be broken across a page, use \\∗, which will usually prevent a pagebreak.

A.3 Booktabs

The booktabs package is included to print nicer tables. See the package documentation [**?**] for more details and motivation. Generally, all vertical lines are removed from the tables for a better visual appearance (so don't put them in), and better spacing and line thicknesses are used for the horizontal rules. The rules are defined as *\toprule* at the top of the table, *\midrule* in between the heading and the body of the table (or between sections of the table), and *\bottomrule* at the end of the table. *\cmidrule* can be used with the appropriate options to have a rule that spans only certain columns of the table.

A.4 Bibliography and Footnotes

The bibliography and any footnotes can also be single spaced, even for the electronic copy. The template is already setup to do this.

Bibliography entries go in the .bib file. As usual, be sure to compile the LaTeXcode, then run BibTeX, and then run LaTeXagain.

To cite websites and other electronically accessed materials, you can use the '@electronic' type of BibTeX entry, and use the 'howpublished' field to include the URL of the source material.

The formatting of bibliography entries will be done automatically. Usually the titles are changed to have only the first word capitalized. If you'd prefer to have your original formatting preserved, place the title in an extra set of curly braces, i.e., "title = {{My title has an AcroNyM that should stay unchanged}},".

A.5 Figures and Tables

The captions of figures and tables take an optional parameter in square brackets, specifying the caption text to be used in the Table of Contents. The regular caption in curly braces is used for the table itself.

Generally captions for tables are placed above the table, while captions for figures are placed below the figure.

Bibliography

[1] Cross-Site Request Forgery (CSRF) - OWASP. https://www.owasp.org/index.php/Cross-Site_Request_Forgery_(CSRF).

[2] Cross-site Scripting (XSS) - OWASP. https://www.owasp.org/index.php/Cross-site_Scripting_(XSS).

[3] Remote Code Execution Vulnerability — Netsparker. https://www.netsparker.com/blog/web-security/remote-code-evaluation-execution/.

[4] SQL Injection: OWASP. https://www.owasp.org/index.php/SQL_Injection.

[5] Adam Barth, Collin Jackson, and John C. Mitchell. Robust Defenses for Cross-site Request Forgery. In *Proceedings of the 15th ACM Conference on Computer and Communications Security*, CCS, NY, USA, 2008. ACM.

[6] H. Boomsma, B. V. Hostnet, and H. Gross. Dead code elimination for web systems written in PHP: Lessons learned from an industry case. In *2012 28th IEEE International Conference on Software Maintenance (ICSM)*, Sept 2012.

[7] David Brumley and Dan Boneh. Remote Timing Attacks Are Practical. In *Proceedings of the 12th Conference on USENIX Security Symposium - Volume 12*, SSYM'03, Berkeley, CA, USA, 2003. USENIX Association.

[8] Tom Van Goethem, Wouter Joosen, and Nick Nikiforakis. The Clock is Still Ticking: Timing Attacks in the Modern Web. In *Proceedings of the 22nd ACM Conference on Computer and Communications Security (CCS)*, 2015.

[9] William G Halfond, Jeremy Viegas, Alessandro Orso, et al. A classification of SQL-injection attacks and countermeasures. In *Proceedings of the IEEE International Symposium on Secure Software Engineering*. IEEE, 2006.

[10] Kihong Heo, Woosuk Lee, Pardis Pashakhanloo, and Mayur Naik. Effective Program Debloating via Reinforcement Learning. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*. ACM, 2018.

[11] Geng Hong, Zhemin Yang, Sen Yang, Lei Zhang, Yuhong Nan, Zhibo Zhang, Min Yang, Yuan Zhang, Zhiyun Qian, and Haixin Duan. How You Get Shot in the Back: A Systematical Study About Cryptojacking in the Real World. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, CCS '18, 2018.

[12] Y. Jiang, D. Wu, and P. Liu. JRed: Program Customization and Bloatware Mitigation Based on Static Analysis. In *2016 IEEE 40th Annual Computer Software and Applications Conference (COMPSAC)*, volume 1.

[13] Yufei Jiang, Qinkun Bao, Shuai Wang, Xiao Liu, and Dinghao Wu. RedDroid: Android Application Redundancy Customization Based on Static Analysis. In *Proceedings of the 29th IEEE International Symposium on Software Reliability Engineering (ISSRE18)*, 2018.

[14] Nenad Jovanovic, Engin Kirda, and Christopher Kruegel. Preventing cross site request forgery attacks. In *Securecomm and Workshops*. IEEE, 2006.

[15] Engin Kirda, Christopher Kruegel, Giovanni Vigna, and Nenad Jovanovic. Noxes: A Client-side Solution for Mitigating Cross-site Scripting Attacks. In *Proceedings of the 2006 ACM Symposium on Applied Computing*, SAC '06, New York, NY, USA, 2006. ACM.

[16] Radhesh Krishnan Konoth, Emanuele Vineti, Veelasha Moonsamy, Martina Lindorfer, Christopher Kruegel, Herbert Bos, and Giovanni Vigna. MineSweeper: An In-depth Look into Drive-by Cryptocurrency Mining and Its Defense. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, CCS '18, 2018.

[17] Hyungjoon Koo, Seyedhamed Ghavamnia, and Michalis Polychronakis. Configuration-driven software debloating. In *Proceedings of the 12th European Workshop on Systems Security*, EuroSec '19, New York, NY, USA, 2019. ACM.

[18] Anil Kurmus, Alessandro Sorniotti, and Rüdiger Kapitza. Attack surface reduction for commodity os kernels: Trimmed garden plants may attract less bugs. In *Proceedings of the Fourth European Workshop on System Security*, EUROSEC '11, 2011.

[19] Anil Kurmus, Reinhard Tartler, Daniela Dorneanu, Bernhard Heinloth, Valentin Rothberg, Andreas Ruprecht, Wolfgang Schröder-Preikschat, Daniel Lohmann, and Rüdiger Kapitza. Attack Surface Metrics and Automated Compile-Time OS Kernel Tailoring. In *Proceedings of Network and Distributed Systems Security (NDSS)*, 2013.

[20] Anil Madhavapeddy and David J Scott. Unikernels: Rise of the virtual library operating system. *Queue*, 11(11), 2013.

[21] Shachee Mishra and Michalis Polychronakis. Shredder: Breaking Exploits through API Specialization. In *Proceedings of the Annual Computer Security Applications Conference (ACSAC)*, 2018.

[22] Anh Quach, Aravind Prakash, and Lok Kwong Yan. Debloating Software through Piece-Wise Compilation and Loading. *Proceedings of USENIX Security*, 2018.

[23] Vaibhav Rastogi, Drew Davidson, Lorenzo De Carli, Somesh Jha, and Patrick McDaniel. Cimplifier: Automatically Debloating Containers. In *Proceedings of the 2017 11th Joint Meeting on Foundations of Software Engineering*, ESEC/FSE 2017, New York, NY, USA, 2017. ACM.

[24] John Regehr, Yang Chen, Pascal Cuoq, Eric Eide, Chucky Ellison, and Xuejun Yang. Test-case Reduction for C Compiler Bugs. In *Proceedings of the 33rd ACM SIGPLAN Conference on Programming Language Design and Implementation*, PLDI '12, New York, NY, USA, 2012. ACM.

[25] Michael Schwarz, Moritz Lipp, and Daniel Gruss. JavaScript Zero: Real JavaScript and Zero Side-Channel Attacks. *Ndss*, (February), 2018.

[26] Hashim Sharif, Muhammad Abubakar, Ashish Gehani, and Fareed Zaffar. TRIMMER: Application Specialization for Code Debloating. In *Proceedings of the 33rd ACM/IEEE International Conference on Automated Software Engineering*, ASE 2018, NY, USA, 2018. ACM.

[27] Peter Snyder, Cynthia Taylor, and Chris Kanich. Most Websites Don'T Need to Vibrate: A Cost-Benefit Approach to Improving Browser Security. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, CCS '17, New York, NY, USA, 2017. ACM.

[28] Chengnian Sun, Yuanbo Li, Qirun Zhang, Tianxiao Gu, and Zhendong Su. Perses: Syntax-guided Program Reduction. In *Proceedings of the 40th International Conference on Software Engineering*, ICSE '18, New York, NY, USA, 2018. ACM.

[29] Philipp Vogt, Florian Nentwich, Nenad Jovanovic, Engin Kirda, Christopher Kruegel, and Giovanni Vigna. Cross Site Scripting Prevention with Dynamic Data Tainting and Static Analysis. In *NDSS*, volume 2007, 2007.

[30] Wenhao Wang, Benjamin Ferrell, Xiaoyang Xu, Kevin W Hamlen, and Shuang Hao. SEISMIC: SEcure In-lined Script Monitors for Interrupting Cryptojacks. In *European Symposium on Research in Computer Security*. Springer, 2018.

[31] Andreas Zeller and Ralf Hildebrandt. Simplifying and Isolating Failure-Inducing Input. *IEEE Trans. Softw. Eng.*, 28(2), February 2002.