FINAL REPORT – May 7, 2021

Development of Software Content Guard for Filtering PDF Files

Ervin Canigur
University of Houston
ekcanigu@cougarnet.uh.edu

Parisa Gholampour
University of Houston
pmehdigh@cougarnet.uh.edu

Will Glaser
University of Houston
whglaser@cougarnet.uh.edu

Steven Triplett
University of Houston
smtriple@cougarnet.uh.edu

Prepared for:
MIT Lincoln Laboratory
May 7th, 2021

Area: Software De-Bloating

Keywords: Robustness Principle, Postel's Law, .pdf, PDF format, ISO 32000

Description:

A content filter targeting the PDF file format based on ISO 32000 (PDF 2.0) could reduce the code base of a PDF application that an attacker could exploit while maintaining the core functionalities of PDF to ensure robust communication within an organization.

Development of Software Content Guard for Filtering PDF Files

Ervin Canigur, University of Houston, ekcanigu@cougarnet.uh.edu
Parisa Gholampour, University of Houston, pmehdigh@cougarnet.uh.edu
Will Glaser, University of Houston, whglaser@cougarnet.uh.edu
Steven Triplett, University of Houston, smtriple@cougarnet.uh.edu

Keywords: Robustness Principle, Postel's Law, .pdf, PDF format, ISO 32000

Executive Summary:

"TCP implementations should follow a general principle of robustness: be conservative in what you do, be liberal in what you accept from others."[1] In one sentence Jon Postel outlined what he dubbed the Robustness Principle. The benefits of easy interoperability between systems would outweigh any downsides. New systems and protocols could be implemented more swiftly.[2] The Internet could grow more rapidly. The downsides at that time were few. Not many protocols existed for communicating between machines. Only a few file formats existed for distributing and consuming content. Maintaining adherence to the Robustness Principle was relatively easier to achieve. With fewer edge cases for your application to cover, fewer lines of code had to be written and maintained.

The cost of Postel's Law has grown alongside the Internet. File formats and protocols are now numerous. For an application to adhere to Postel's Law means writing code to handle all possible inputs, or all possible interactions. This increases the code base and with it the possibility of the introduction of bugs. The application's attack surface grows. This project aims to examine the Robustness Principle and invert it by implementing a filtering content guard. The goal of this content guard would be to reduce the amount of code that a protocol implementation could access within a given network. This result could certainly reduce the capabilities of a protocol, but it should also reduce the attack surface.

This project targeted the Portable Document Format (PDF) ISO 32000 standard due to its immense popularity as a document format. A literature review assisted in the identification of relevant research, PDF specifications, PDF implementations, and current tools for PDF metadata extraction. A combination of a whitelisting/blacklisting rule-based approach was implemented in C++ to limit the PDF feature-set available and thereby reduce source lines of code available as an attack surface. Testing of a PDF corpus was performed using the open source view Okular and the built-in GCC tool Gcov to acquire the resulting source lines of code (SLOC) executed to render each PDF file. A second test was performed with the implementation of our content guard to acquire the SLOC executed for only those PDFs allowed to pass. Finally, statistical hypothesis testing was performed using the two-sample Welch's t-test to conclude at a 99% confidence level (alpha = 0.01) that our guard reduced the SLOC executed.

Table of Contents

A. Introduction
    a. Problem Statement
    b. Purpose Statement
    c. Motivation
    d. Broader Impact
B. Literature Review
C. Methods & Procedures
D. Findings & Analysis
E. Conclusions & Future Work
F. Bibliography
G. Team
    a. Biographies

Introduction

Problem Statement: The current implementation of the Robustness Principle (Postel's Law) has long been a concern for security as well as a cause for software bloat. Currently, it is common practice for those in network administration to implement guards/firewalls to filter traffic that doesn't meet certain rules in order to reduce the attack surface of the network. However, this type of guard is not typically available for specific file types. Furthermore, if integrated with a software application for example, would a filetype-specific content guard reduce or increase the overall attack surface of the code libraries that are used to process the file?

Purpose Statement: Inverting the Robustness Principle will lead to a measurable increase in robustness of an organization's security. By implementing a content filter for the PDF file format, based on ISO 32000, we will be able to measurably reduce lines of code activated in corresponding PDF applications. This reduction will also affect a corresponding reduction in an application's attack surface by making vulnerable features unavailable.

Motivation:

"TCP implementations should follow a general principle of robustness: be conservative in what you do, be liberal in what you accept from others."

Fewer downsides do not equal no downsides, however. Any bug which existed in the output of some other application had to be accommodated. In this way bugs became features. An example of this is HTML. Syntactically HTML consists of tags wrapping some text. It is common enough for developers to forget to add a closing tag here or there. Browsers in turn will generally take a guess at where that closing tag was meant to go and render the resulting HTML. [4] In this way imperfect input is now an accepted feature in most web browsers. Conversely, for the sake of Postel's Law, if a feature exists regardless of its popularity, you still must maintain compatibility. This is what happened in the case of the Heartbleed vulnerability. A little used feature in the TLS specification led to the introduction of a serious vulnerability.

Novel solutions are sometimes relatively simple solutions. The HeartBleed vulnerability affected hundreds of thousands of websites, web servers, and individual devices vulnerable as recent as 2017. However, Amazon's S2N implementation of the TLS reduced the LOC to 6,000, compared to 500,000 LOC in OpenSSL which contained the HeartBleed vulnerability as well. Amazon thereby reduced their attack surface by simply removing code that was not necessary for their network.

Broader Impact:

With the recent disclosures of CVE-2021-21057 through CVE-2021-21063 this project is especially timely. CVE-2021-21057 through CVE-2021-21063 affect recent versions of Adobe Acrobat Reader DC. By using a specially crafted PDF document malicious actors can trigger arbitrary code execution, disclose sensitive information, or even a denial of service.[8] An unsuspecting user just has to open the file with the affected software. This is not so uncommon as PDF is one of a handful of common email attachments used to piggyback malicious code onto

a system leading to an initial foothold for an attacker. Firewalls are more commonly configured to allow email through, with the assumption that a host machine's antivirus and email client will filter out malicious content; the latter mechanisms are only able to discard content that matches known malicious signatures. There are gaps in our current defenses, and some implementations of current defensive systems increase the attack surface of an organization. By inverting Postel's Law and reducing acceptable file formats to an organization specified subset, we can measurably reduce attack surface.

The research we are doing into the PDF format could lead to similar research into other file formats. If a content filter such as the one we aim to develop can significantly reduce an organization's attack surface we could see a cultural shift in security postures overall; limit internal usage of file formats to a prearranged subset suitable for an organization's given domain, and enforce that limit through the usage of metadata-based content filters.

Literature Review

[3] Avenatti et al. tackled inverting Postel's law project by designing a content guard that filtered out JPEG image files that are malformed in their database. In their research they narrowed down the type of JPEG file they were working on to JPEG ISO/IEC 10918-1. They claimed that the content filter reduced the overall attack surface. In our work, we are planning to target another type file, PDF, which is extremely complicated compared to JPEG files. Each PDF file consists of texts, different types of images, etc and they are built on a variety of standards. We must take into account these pieces when parsing PDF files to be able to detect malformation of the data. Another consideration is our content filter will have a totally different structure in terms of filtering out another category of files compared to previous work.

The PDF format itself has been subject to much research both in the Academic and Security communities. The complex structure of the PDF files makes them a unique payload which can reach a large variety of platforms. In this paper [4] the author has examined the attack surface of PDF over a year and he has discovered almost 150 vulnerabilities independently in the most popular PDF readers consisting of Adobe Acrobat and reader, foxit reader, google chrome, Windows PDF library, OS X preview and Adobe Digital Editions. Over 100 of them have been authorized with CVEs and fixed by vendors.

In another work [5] security issues regarding PDF documents are provided at PDF code level. Their approach has targeted the code execution capabilities of the PDF language that can be exploited by attackers. The authors were able to design two proof-of-concepts to assess the level of risk of the possible attack. Furthermore, they developed their own tool to analyze and examine PDF documents.

PDF has been a popular topic on security blogs as well. Phil Stokes writing for SentinelOne's blog [9] dissected the techniques attackers use to weaponize PDF files. One of the most popular, uses embedded Javascript. Typically used to make documents interactive Javascript can be used to contact a tracking site, transmit personal information and more. Also popular are Adobe XFA forms, essentially embedded XML typically used for dynamically resizing fields. An attacker can instead cause your PDF reader to attempt an authenticated connection to an SMB share which they control leading you to expose your NTLM password hash.

A context-aware approach for detection and confinement of malicious Javascript in PDF in this paper [6] has been proposed. They argued by extracting static features and applying context monitoring into a PDF document, it is possible to detect malicious documents. They were able to implement a prototype to test their method by using 18623 benign PDF and 7370 malicious one. Their approach was able to detect and confine malicious Javascript in PDF files with high accuracy.

Attack surface is difficult to measure. There is no broadly agreed upon metric by which to quantify the size of an application or organization's attack surface. Manadhata et al. [10] attempt to define a qualitative metric. They identify three types of resources which they use to measure system attack surface; methods, channels and data. Important to their work is distinguishing between actual vulnerabilities present in a system and likelihood of an attacker to target that vulnerability. A system which has a large number of vulnerabilities does not necessarily have a large attack surface. Likewise, a system which has very few vulnerabilities is

not guaranteed to have a small attack surface. Therefore their metric attempts to measure properties of a system.

Prior to this last paper, Manadhata et al. [11] had tried to define a qualitative metric for attack surface in terms of a systems actions which it exposes to users and those resources which the system actions access or modify. This work is an attempt to improve upon two commonly used metrics: Number of bugs present at the code level, and number of CERT advisories, Microsoft Security Bulletins and CVE currently associated with the system. With their proposed metric, this team attempts to capture how exposed a system is to attack. Intriguingly their proposed procedure for measuring attack surface requires comparing two systems, making it a measurement of relative security rather than absolute security.

More broadly, there has been research done into reducing the attack surface of an application through debloating. In this paper [7] Koo et al. focused on tighter configuration of applications to reduce application feature sets, thereby debloating the application. They found that compared to the standard configurations they were able to reduce code base 77% for Nginx, 53% for VSFTPD and 20% for OpenSSH. They argue this is a significant attack surface reduction.

Postel's Law inherently is about initial robustness of a system, not long term. Protocols, applications and systems, when initially implemented follow Postel's Law. There seems to be a small subset however, which over time tightens their specification and implementations. Other systems go the opposite route, adding features and increasing the laxity for allowable inputs. There seems to be a dearth of literature pertaining to either scenario. It would be interesting to see work which shows what happens to attack surface or even just system vulnerability when a system matures away from Postel's Law.

While qualitative metrics for measuring attack surface have real merit, they seem to have not yet overthrown the status quo. It seems that CVE, system bugs and application code base still dominate conversations around a system's attack surface. Part of this may stem from practicalities involved with measuring attack surface using qualitative metrics proposed.

We justify our use of source lines of code as a metric for measuring attack surface on the following bases. First, reducing source lines of code is an existing approach that current organizations take to reduce overall attack surface. Second, reducing source lines of code will satisfy both traditional metrics for attack surface as well as qualitative ones. A system should experience fewer bugs and CVE's over its lifetime with fewer source lines of code, and fewer source lines of code exposes fewer features, or 'actions,' to a user or attacker. This will narrow down approaches left to an adversary to exploit a system and possibly remove any low hanging fruit as well. The trade-off is that source lines of code can't ultimately be a universal metric. Proprietary software is common, especially in the enterprise setting. This can be circumvented somewhat when a proprietary software utilizes open source projects for implementation of some modules. The scope of this project allows us to choose an open source PDF viewer for our tests. For work going beyond this project, a more global solution may be necessary.

Methods & Procedures

Our team employed empirical research methods for this project. We propose a guard written in C++ to limit the PDF feature-set available and thereby reduce source lines of code available as an attack surface. We used code coverage tools and a large corpus of PDF to test and measure the number of lines of code executed when examining a PDF with our guard and opening a PDF in a popular open source PDF viewer. With these measurements we could then compare the overall lines of code executed by the guard and the PDF viewer.

Given the large set of features one can find in PDF metadata, we initially planned on writing a guard which would whitelist desired features. There are however distinct categories within PDF metadata which led us to determine a simpler approach of blacklisting some features and whitelisting others. We also needed to generate a set of PDFs which would pass our guard. This proved more difficult than finding a corpus which would fail. Ultimately, we were able to generate some PDf which were simple enough and could be stripped of compression resulting in a final document which would pass.

The guard is designed to only pass whitelisted PDF files that have no compression, no JavaScript, email links, URLs and in general no embedded action function in their meta data. Furthermore, the type of font and image are also limited to a few types. By reading PDF as a simple text file in the guard code, the meta data details are accessible. We read the file line by line and look for the specific types of font, image, compression filters, encryption filters and the action function tags. JPEG images are stored with /XObject tag in a decompressed PDF file. "ASCIIHexDecode", "ASCII85Decode", "LZWDecode", "FlateDecode", "RunLengthDecode", "CCITTFaxDecode", "JBIG2Decode", "DCTDecode", "JPXDecode", "Crypt" and "Encrypt" is the list of compression and encryption tags that were are looking for in the body of metadata to be rejected if the guard detects them. Action functions are represented as "URI" "Launch", "JavaScript", "GoTo", " GoToR", "GoToE" tags. The PDF files containing them will not pass by guard as well.

To measure code coverage, you must compile an application from source; due to this we looked at open source PDF viewers. We initially attempted to use PDFium, which is developed by Google and used in the Chrome browser. PDFium however proved difficult to configure and compile for code coverage. We chose to use Okular instead. Okular is developed by KDE and utilizes the Poppler library for PDF rendering. Most importantly, Okular compiles with GCC allowing us to compile it for code coverage and generate coverage reports with the GCC tool Gcov. Lastly, Bash was used to automate the testing process as much as possible, and generate coverage reports and source lines of code executed totals.

While we were able to measure a large portion of code coverage, there are still some effects or features of PDF which we could not measure. One of the features we aimed to block with our guard are XFA style sheets; essentially embedded XML in a PDF document. Okular does not support XFA style sheets so for documents in our testing corpus which included this feature there weren’t any source lines of code executed, and so nothing to measure. Using a different PDF viewer would remedy this. PDFium is used in the Chrome browser and likely would support XFA style sheets.

We also couldn't capture fully the effects of URL and Email addresses included in documents in our test corpus. URL when clicked open in the systems default Web Browser. Likewise, clicking on an Email address opens the systems default Email application. We were able to measure the source lines of code executed in Okular from these actions, but as neither Email application or Web Browser were compiled from source, for code coverage, once the chain of execution left Okular, we could not measure it.

We were able to successfully measure much of the relevant code which executed when opening a PDF. The data we captured however is quantitative. It shows that significantly fewer source lines of code are executed when utilizing the guard and a minimal PDF document vs no guard and allowing all PDF features available. The data however does not show which lines of code executed. It could be very interesting to see which lines of code overlap for both the allowed set of PDF and the disallowed PDFs.

Findings & Analysis

In short, our work set out with the hypothesis that the implementation of our content guard would reduce the attack surface by requiring input to be highly regulated. The measure through which this was accomplished was SLOC executed for a corpus of PDF files. Based upon our testing and analysis of the resulting data, we can conclude that our guard did in fact reduce the average SLOC executed for our test corpus and therefore the attack surface as well.

To state these findings with some degree of confidence, we approached the analysis of our data from a statistics perspective. A common method of comparing two samples of data is to inspect and compare the corresponding means of the samples. At a high level, we used a boxplot (Figure 1) to visualize the effect of our content guard on the mean SLOC executed. This effect can be clearly seen by the significant reduction in mean SLOC executed from 14682 (standard deviation = 2861, n = 62) before the guard to 10038 (standard deviation = 769, n = 8) after the implementation of the guard.

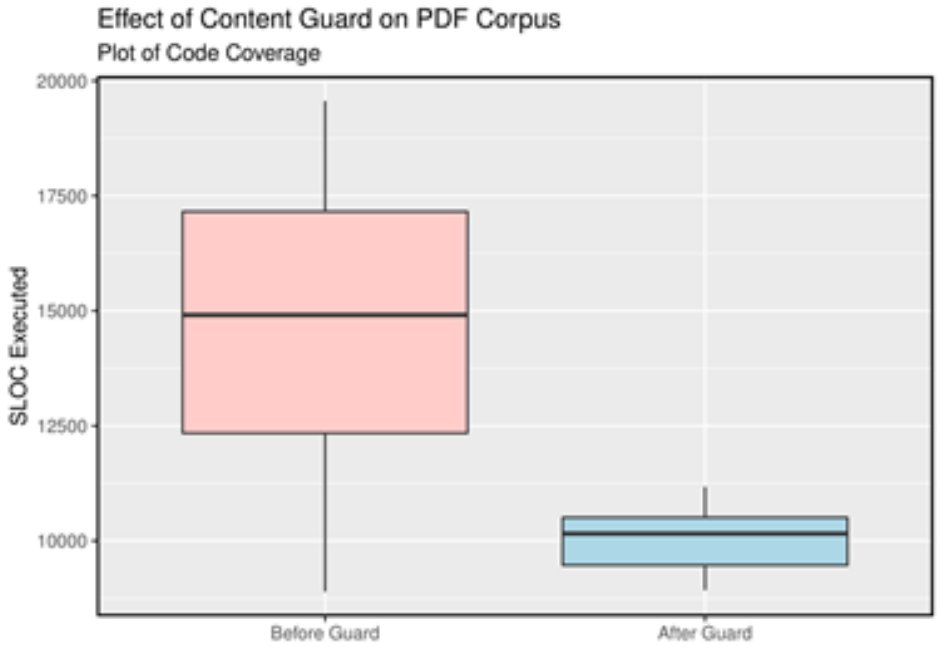

Figure 1. Effect of content guard on SLOC executed

We sought to provide a slightly more rigorous analysis of the resulting data, and a common method for comparing the means of two samples is a two-sample t-test. The most common variation of the two-sample t-tests is the Student's t-test. However, this requires two underlying assumptions to hold: the data must be normally distributed and the variance of both samples must be approximately equal. To tackle the first assumption of normality, a common tool is the Shapiro-Wilk test which tests the null hypothesis that a data sample came from a normally distributed population. The resulting Shapiro-Wilk test p-values of 0.0987 and 0.801 for the before-guard implementation and after-guard implementation respectively allow us to

conclude at 95% confidence (alpha = 0.05) that both samples are normally distributed. The second assumption of equal variances between two samples is commonly investigated though an F-test in which the null hypothesis is that the two samples have equal variances. The resulting F-statistic and degrees of freedom of 61 and 7 respectively correspond to a p-value of 0.001297. Therefore, the null hypothesis can be rejected even at a 99% confidence level (alpha = 0.01) that the samples have equal variances. Although our data does not satisfy both assumptions of the Student's t-test, the alternative Welch's t-test allows for a comparison of means which does not rely on the assumption of equal variance and does not use a pooled variance in the test statistic calculation. In this test our null hypothesis is that the implementation of our guard does not affect the SLOC executed from our dataset, and the resulting t-value of 10.23 and p-value of 1.076e-12 provides significant evidence that the null hypothesis can be rejected at a 99% confidence level. Therefore, our conclusion is that the implementation of our guard does affect the SLOC executed from the dataset (and the resulting attack surface).

Conclusions & Future Work

The results of this work provide evidence that the content guard approach to inverting Postel's Law can successfully do so while reducing attack surface, and these findings indicate that the PDF file specification can be analyzed in combination with the content guard approach to restrict the input to an application (such as the PDF view Okular) while also maintaining the core functionality of the document.

Due to the limited time of only one semester we had to limit the scope of our guard to reach our goals. Because of the nature of PDFs and their support for a wide variety of uses as well as devices. We were only able to focus on filtering out text, images as well as certain PDF actions. With our SLOC execution and code coverage reports we were able to determine that a guard would be effective. If we had more time we would have liked to get a report on exactly which functions are being called. The actual function name, thus allowing for further analysis. Another thing would be to test with more than just one PDF viewer and PDF render. Then it could be possible to draw some conclusions about the PDF spec posted in the ISO definition. We believe that there are many more possibilities with this guard, it can be expanded and implemented into the PDF viewer to give the user a safer PDF experience.

Bibliography:

[1] DOD STANDARD TRANSMISSION CONTROL PROTOCOL. (n.d.). Retrieved from https://tools.ietf.org/html/rfc761#section-2.10

[2] The Harmful Consequences of the Robustness Principle. (n.d.). Retrieved January 31, 2021, from https://tools.ietf.org/html/draft-iab-protocol-maintenance-04

[3] Avenatti, B., Benson, M. and Jiles, R., 2020. Inverting Postel's Law: Reducing Attack Surface by Combining Content Guards with Applications at Runtime.

[4] https://www.blackhat.com/docs/asia-17/materials/asia-17-Liu-Dig-Into-The-Attack-Surface-Of-PDF-And-Gain-100-CVEs-In-1-Year-wp.pdf

[5] https://www.blackhat.com/presentations/bh-europe-08/Filiol/Presentation/bh-eu-08-filiol.pdf

[6] Liu, D., Wang, H. and Stavrou, A., 2014. Detecting Malicious Javascript in PDF through Document Instrumentation. *2014 44th Annual IEEE/IFIP International Conference on Dependable Systems and Networks*,.

[7] Koo, H., Ghavamnia, S. and Polychronakis, M., 2019. Configuration-Driven Software Debloating. *Proceedings of the 12th European Workshop on Systems Security - EuroSec '19*,.

[8] https://cve.mitre.org/cgi-bin/cvekey.cgi?keyword=PDF

[9]Malicious PDFs | Revealing the Techniques Behind the Attacks https://www.sentinelone.com/blog/malicious-pdfs-revealing-techniques-behind-attacks/

[10]An Approach to Measuring A System's Attack Surface http://www.cs.cmu.edu/~wing/publications/CMU-CS-07-146.pdf

[11]Measuring a System's Attack Surface https://www.cs.cmu.edu/~wing/publications/tr04-102.pdf

Final Schedule:

Gathered Initial Test Dataset	April 2nd
Completed Testbed Environment	April 9th
Tested/Refined Guard and Dataset	April 12th – April 23rd
Biweekly Progress Meetings with Technical Director	April 2nd – April 30th
Biweekly Progress Meetings with Project Instructor	April 1st – April 29th
Evaluated Data and Draw Conclusions	April 26th – May 3rd
Final Submission	May 7th

Team Biography:

Evrin Canigur is a senior undergraduate student in Computer Science at University of Houston. He has working experience with a variety of languages and currently works as a software engineer. He has experience with servers, ios development, a variety of java projects, as well as a variety of web applications. With over 7 years of development experience, Ervin has had a chance to work with a variety of teams and a variety of projects. He hopes to soon also pursue a masters in computer science in the future.

Parisa Gholampour is currently a Master's student in Computer Science at University of Houston. She previously obtained her undergraduate degree in biomedical engineering and master degree in electrical engineering. In the latter, her main dissertation focus was on industrial networking solutions for control and automation challenges which provided insight about network systems and tools. In recent years she has decided to transition to the computer science field in depth and gained a lot of knowledge in this area. She has extensive programming knowledge in Python, C++ and R specifically in data science applications.

Will Glaser is a senior undergraduate student in Computer Science at University of Houston. He has significant experience in developing programs in several languages such as Javascript, Java, C, ARMv7 assembly language, and C++. His interest in security related work stems from personal studies of Cyber Security and Penetration Testing which results in exposure to HTML and Javascript. He is experienced with the Linux and Arch Linux operating systems, working from the command line, and scripting with Python. He has experience with security tools such as Wireshark for packet capture and analysis, and using various networking command line utilities such as ip link, ip route, ip neigh, nslookup, drill and iptables.

Steven Triplett is currently a Master's student in Computer Science at University of Houston. Throughout his education and career, he has been involved in numerous research and professional projects which provided him the opportunity to lead, or work as part of a team. Additionally, these projects encompassed a wide variety of topics ranging from the design and analysis of chemical plant to video game design. While his previous academic background is not focused on computer science, his extensive knowledge of mathematics, statistics, and modeling from his education in chemical and biomedical engineering will prove helpful in understanding current research and developing their own models.