RUHR-UNIVERSITÄT BOCHUM

RUB

RUHR-UNIVERSITÄT BOCHUM

Browsing PDFs for Fun and Profit

Tobias Friese

Master's Thesis – July 20, 2026
Chair for Network and Data Security.

Supervisor: Vladsilav Mladenov
Advisor: Lukas Knittel

hgi Lehrstuhl für Netz- und Datensicherheit

1 Introduction

PDFs are a widely used format for sharing digital documents on the web, including in business and governmental contexts. Today, all major browsers support viewing PDFs directly inside the browser. Previous research shows that the PDF specification contains a lot of functionality, such as URL invocation or JavaScript execution, which can be especially dangerous in a web context [9]. Due to this, browser developers have taken steps to isolate the PDF viewer from the rest of the browser, and limited the amount of supported PDF features. In this master thesis, we want to examine if these PDF viewers are still vulnerable to previously identified attacks targeting PDF applications, despite their limited feature set. We also want to examine the interaction between the PDF viewer and web technologies to identify new attack paths abusing PDF features, or ways to increase the impact of known dangerous functionality.

1.1 Motivation

Previous research in the field of PDF security has mostly concentrated on dedicated PDF viewers, as they offer the broadest support of PDF features. The attack scenarios were developed with these powerful PDF viewers in mind, and then *also* tested against the less powerful PDF viewers in browsers. However, due to their unique context of being inside a web browser, additional attack paths might arise from the interaction of the PDF viewer with web technologies. In this master thesis, we want to examine these potential attack paths by taking a closer look at how the PDF viewers are implemented in the browsers, in which way they can interact in a web context, and whether new vulnerabilities arise from that. We will also re-examine the previously identified attacks which also affected web browsers, and see if we can escalate their impact.

We specifically want to answer the following research questions:

- How do modern browsers integrate PDF viewing? What origin and process isolation model is used to seperate potentially dangerous PDF functionality from the browser?
- Which web APIs are reachable from inside PDFs in this isolation model?

- Which communication channels exist between the opened PDF in the PDF viewer and the surrounding browser? How are messages relayed between PDF and browser?
- Does the PDF viewer receive special handling as a built-in browser component that weakens standard web security policies, e.g. by disabling the Same-Origin-Policy (SOP), granting access to cookies and web storage, or by removing navigation restrictions? Is it possible to exploit the weakend security policies by abusing dangerous PDF functionality?
- Can PDF-contained actions and scripts escape the PDF viewer sandbox or gain capabilities beyond those of a normal webpage in a browser?

1.2 Related Work

While there has already been research on the topic of PDF security, it mostly focused on attacks targeting dedicated PDF applications.

A significant part of this research focused on breaking PDF signatures, e.g. by abusing the signature verification process with non-compliant documentes as shown in "1 trillion dollar refund: How to spoof pdf signatures" [8] by Mladenov et al., or by introducing changes which were not deemed important enough by PDF viewers to invalidate the signature, as shown in "Shadow Attacks: Hiding and Replacing Content in Signed PDFs." [7] by Mainka, Mladenov, and Rohlmann. In the bachelor thesis *Visual Signature Spoofing in PDFs* [12], Tobias Friese also showed that it is possible to spoof signature indicators in these applications using only specified PDF features.

While there are some browser-based PDF toolkits which offer functionality to digitally sign PDFs and verify these signatures, the browser-embedded PDF viewers do not offer such functionality. Instead, they simply display the signature and offer no information about the validity of it. For this reason, the aforementioned signature attacks do not apply to the built-in PDF viewers, and are not the focus of this thesis.

Another way to target PDF viewers is trying to find memory corruption vulnerabilites in the PDF viewers parsing and rendering processes. As a part of the browser, the embedded PDF viewers are regularly tested for such vulnerabilities using automated fuzzing. For PDFium, which is Chromium's embedded PDF viewer, NIST lists more than 100 security vulnerabilities, with most of them being related to memory corruption [10]. We therefore leave this attack vector to other researchers, and instead focus on specification-driven attacks.

Regarding this focus, the paper "Processing dangerous paths" [9] by Müller et al. is more interesting to us. In it the authors reviewed potentially dangerous features in

the PDF specification, and developed attack vectors to abuse these features for denial of service, disclosure of information, data manipulation, and code execution. This research again focused on dedicated PDF viewers, but also included major browsers, namely Chrome, Firefox, Safari, Opera, and Edge.

While the browser PDF viewers were not vulnerable to data manipulation or code execution, it was possible to cause denial of service and invoke URLs in Chrome, Firefox, and Opera. The authors also mention the increased impact caused by URL invocation in browsers, but do not examine this path further. This is where we want to extend this previous research.

The PDF specification allows PDFs to execute limited JavaScript [6, 1]. In [9], the authors also developed a crawler to map the available JavaScript functionality in the PDF viewers, which allows the PDF to fingerprint the used PDF viewer. At the time of publishing (2021), Chrome and Opera offered a much more limited subset of JavaScript than other PDF viewers, Firefox's PDF viewer did not support JavaScript at all. We want to get an updated look at the JavaScript support of the browser's PDF viewers by rerunning the experiment using an updated version of the crawler.

In "Beyond Traditional PDF Exploits: Security Risks in XFA Documents" [2], Borgstedt et al. additionally evaluated the XML Forms Architecture (XFA), which allows more feature-rich forms in PDFs, for dangerous functionality, similar to [9]. The researchers identified multiple dangerous features in the XFA specification which could allow remote code execution, URL invocation, file inclusion and denial of service. These features were tested against 7 PDF viewers, all of which proved vulnerable against at least one vulnerability class. In particular, all but one PDF viewer were vulnerable to URL invocation, which could prove even more harmful in a browser's PDF viewer.

In their paper, the authors mention that Chromium's PDF viewer does not support XFA at all, which would mean that it is not a risk from these attacks. Firefox's PDF.js however does implement a subset of the XFA specification. The researchers excluded PDF.js from the evaluation, which means that the potential of XFA attacks on PDF.js is not fully explored.

In "When Documents Rewrite Themselves: Security Analysis of Self-Modifying PDFs" [13], Vollbracht et al. examined possibilities how PDFs can automatically modify their appearance when specific conditions are met. Since the PDF viewers in browsers do not support digital signatures, the identified attacks which self-modify in order to display different content for a signed PDF are not applicable to these viewers.

The authors also looked at self-modification to change the visible content during printing, and found both Firefox's and Chrome's built-in PDF viewer to be partially vulnerable: it was possible to change the visible content of the PDF during printing, but the changes are visible in the print preview. This means that smaller changes

like changing “$100.00” to “$10000” in a multi-page contract might go unnoticed, while larger changes would likely be detected.

In a similar vein, Fedotkin published a blogpost *Fickle PDFs: exploiting browser rendering discrepancies* [3] about how a PDF can display different content, depending on whether the PDF was opened in Safari and Preview on MacOS, or Firefox and Chrome's PDF viewers. The attack abuses the missing support for widget annotation appearances in Safari and Preview to inject content which is only shown in Firefox and Chrome. In [5, 2], a similar attack is described, this time exploiting the fact that PDFs containing XFA display different content if XFA is not supported by the PDF viewer, leading to different content between Chrome and Firefox.

Based on this previous research [13, 3, 5], it is likely that more such differences between the rendering in the browsers' PDF viewers exist, which can be exploited to show different content to users.

Over the years, there have also been several attacks abusing PDF viewers in browsers to bypass browser defenses. In the blogpost *Adobe Reader Same-Origin Policy Bypass* [11], Rios, Lanusse, and Gentile found that by exploiting the Adobe Reader browser plugin, it was possible to read data from other origins, effectively leading to a bypass of the browsers Same-Origin-Policy. Franken, Van Goethem, and Joosen examined the handling of third-party cookies, i.e. cookies which are sent cross-site, in “Who Left Open the Cookie Jar? A Comprehensive Evaluation of Third-Party Cookie Policies” [4]. They found that PDFium, used in Chrome and Opera, allowed sending POST-requests with cross-site cookies, even when the browser-configuration was set to block third-party cookies. These examples show that abusing PDFs to bypass browser defenses is not a fully new idea and proved successful. In this thesis, we want to explore this attack surface further.

2 Work Packages

In my master thesis, I will tackle the following work packages:

WP 1: PDF Viewers in Browsers In WP1, I want to collect the most commonly used PDF viewers in browsers, like built-in PDF viewers, popular PDF viewer extensions, or projects for rendering and editing PDFs in the browser. From these PDF viewers, I will select candidates for further evaluation, and evaluate their feature support in preparation for WP2.

Task 1.1: First, I will collect the most commonly used PDF viewers in browsers. This will include the built-in PDF viewers in browsers, but also popular libraries with which websites render PDFs, or browser extensions which change and replace the built-in PDF viewer. The goal is to get an overview over how PDFs are viewed in browsers, and select the candidates for further evaluation based on their popularity. I expect for this task to take one week.

Task 1.2: Next, I want to evaluate the feature support of the selected PDF viewers to map their attack surface. I will review the PDF viewer's support for the potentially dangerous PDF features identified in [9], but also look for PDF features which were either added in PDF 2.0, or potentially pose a risk only in a web context. I will also try to map the supported JavaScript functionality of the PDF viewers as a graph using an updated version of the crawler from [9] to find out whether the supported JavaScript functionality has increased since 2021. I plan to spend five weeks on this task.

The result of WP1 will be an overview over popular PDF viewer libraries for browsers, and their supported features.

WP 2: Re-Evaluation of Known PDF Attacks In this workpackage, I want to re-evaluate the selected PDF viewers on whether they are vulnerable to known PDF attacks, e.g. those discussed in Section 1.2.

Task 2.1: First, I will collect known attacks, and create an overview over the required PDF features needed for the attacks. As discussed in Section 1.2, we will leave memory corruption and signature attacks out of the scope of this thesis, and concentrate on attacks targeting dangerous features in the PDF specification. I expect this task to take a week.

Task 2.2: Next, I will compare the required features for the attacks with the available features of the selected PDF viewers which were identified in Task 1.2. If the required features are available, I will create a Proof-of-Concept for the attack to see if the viewer is indeed vulnerable. Any identified vulnerabilities will be communicated in a responsible disclosure process. I expect this task to take four weeks.

The result of WP2 will be an overview over the required PDF features for known PDF attacks, and whether the selected PDF viewers are vulnerable to these attacks.

WP 3: Examining the Impact of PDF Attacks on Browsers In WP3, I want to test for additional attack surface offered by the browsers. I want to evaluate whether the position inside the browser makes new attacks feasible, or can be used to escalate the impact of known attacks.

Task 3.1: First, I want to examine how the PDF viewers are embedded inside the browsers. I will examine how the potentially dangerous PDF functionality is isolated from the rest of the browser, and how the communictation between the PDF in the PDF viewer and the browser works. I will also examine whether the PDF viewers receive special handling due to being a browser component. If some security policies like the SOP or Content-Security-Policy (CSP) are ignored by the PDF viewers, it might be possible to abuse dangerous PDF functionality to gain capabilities beyond those of a normal webpage. I expect this task to take 6 weeks.

Task 3.2: Next, I will try to turn the identified browser behavior into feasible attacks. I will first try to escalate the impact of the attacks identified in WP2, e.g. by abusing weakend security policies such as the SOP. I also want to take note of the different impact dangerous behavior like URL invocation can have in a browser context, and will look for new PDF attacks which can only exist in a browser context. For the identified attacks, I will create Proof-of-Concepts to demonstrate their impact. Any identified vulnerabilities will be communicated in a responsible disclosure process. While the required time for this task is largely dependent on the results in the previous tasks, I plan to spend three weeks on this.

WP 4: Summarizing my Findings In this final WP, I want to organize my findings, and present them in my thesis. I will also organize the artifacts created during the thesis, namely the PDFs testing the viewers capabilities, and the PoC's for the identified PDF attacks. I will work on this work-package for the full duration of the thesis, and fully focus on it in the final 6 weeks.

3 Organization of this Thesis

I plan to structure my thesis as follows:

Chapter 1: Introduction will introduce the topic of my thesis, give a motivation and describe the work done during this thesis.

Chapter 2: Background will cover the basics of PDFs which are required to understand the attacks found in the thesis. It will also introduce web technologies and security features which were (successfully) attacked using malicious PDFs.

Chapter 3: PDF Viewers in Browsers will contain the results of WP 1 and WP 3.1. It will include a description of the PDF viewers selected for evaluation and the used methodology. It will also list the PDF features supported by the PDF viewers, and illustrate the security architecture used to isolate potentially dangerous PDF functionality from the rest of the browser.

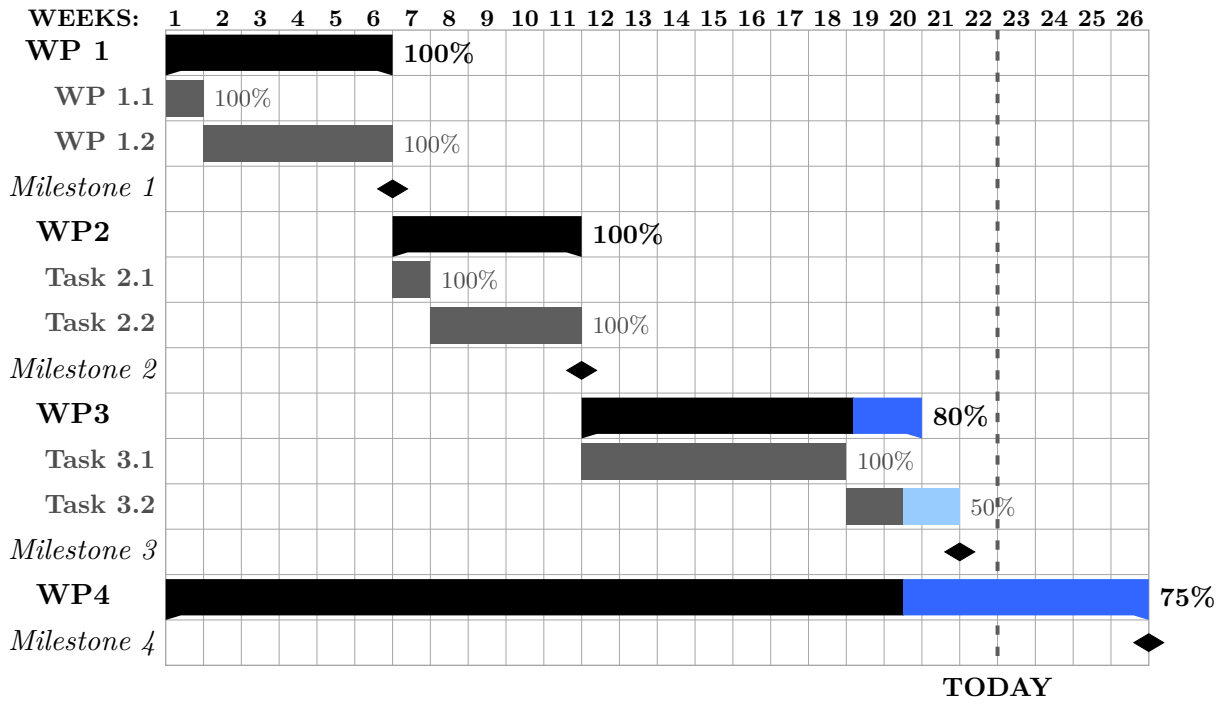
Chapter 4: Re-Evaluation of Known PDF Attacks will contain the results of the reevaluation of the known attacks, which are done in WP2. It will contain information about the tested attacks, their required features, and an overview over which PDF viewers support these features and were vulnerable.

Chapter 5: Impact of PDF Attacks on Browsers will include the results of WP 3.2, i.e. whether the impact of previously known attacks could be increased due to the browser context, and any additional attacks which might only be possible in PDF viewers in browsers.

Chapter 6: Conclusion will conclude my thesis, summarizing the previous findings, and showcasing the security landscape of PDF viewers in browsers.

4 Timeline

My master thesis will be following this general timeline:



4.1 Milestones

I plan to end each work package with a "Milestone" result, which are described in the following:

Milestone 1: Overview of Selected PDF Viewers and Their Supported Features
Milestone 1 will conclude the first workpackage, by which I plan to have an overview over PDF viewers used in browsers, a selection of the relevant candidates for further evaluation, and a list of the features supported by these PDF viewers. This overview should allow me to identify viable known attacks in work package 2.

Milestone 2: Overview of Vulnerable PDF Viewers and Proof-of-Concept Exploits By the end of work package 2, I want to have evaluated the selected PDF viewers on whether they are vulnerable to known PDF attacks. This will give an updated look at the security of the PDF viewers, and allow me to identify vectors to attack the PDF viewers further in work package 3. I will first test the vulnerability of the PDF viewers to these attacks theoretically, based on the supported features, and then create practical Proof-of-Concept PDFs for the viewers deemed vulnerable.

Milestone 3: Research Results on Browser-Based Vulnerabilities With milestone 3, I will conclude work package 3. By the end of this workpackage, I want to have found ways to escalate the impact of the known vulnerabilities previously identified in the PDF viewers, and new attack vectors, based on the viewers position in the browser. For the identified attacks, I will have created Proof-of-Concepts showcasing their impact. I want to have examined the research questions specified in Section 1.1, and any additional research questions that came up during further study of the PDF viewers in the browser.

Milestone 4: Written Thesis and Organized Results This final milestone concludes work package 4, and my master thesis as a whole. I want to have written my thesis, and organized my findings so that they can be easily used during further research, or be made available as artifacts.

Bibliography

[1] Adobe Systems Incorporated. *JavaScript for Acrobat API Reference*. 2021.

[2] Sören Borgstedt, Titus Vollbracht, Christian Mainka, and Vladislav Mladenov. "Beyond Traditional PDF Exploits: Security Risks in XFA Documents". 2025.

[3] Zakhar Fedotkin. *Fickle PDFs: exploiting browser rendering discrepancies*. 2024. URL: https://portswigger.net/research/fickle-pdfs-exploiting-browser-rendering-discrepancies.

[4] Gertjan Franken, Tom Van Goethem, and Wouter Joosen. "Who Left Open the Cookie Jar? A Comprehensive Evaluation of Third-Party Cookie Policies". In: *27th USENIX Security Symposium (USENIX Security 18)*. 2018, pp. 151–168.

[5] Toni Huttunen. *Two Faces of a Same PDF Document*. 2022. URL: https://blog.fraktal.fi/two-faces-of-the-same-pdf-document-17e7a15522a0.

[6] International Organization for Standardization (ISO). *ISO 32000-2:2020 Document management — Portable document format — Part 2: PDF 2.0*. 2020.

[7] Christian Mainka, Vladislav Mladenov, and Simon Rohlmann. "Shadow Attacks: Hiding and Replacing Content in Signed PDFs." In: *NDSS*. 2021.

[8] Vladislav Mladenov, Christian Mainka, Karsten Meyer zu Selhausen, Martin Grothe, and Jörg Schwenk. "1 trillion dollar refund: How to spoof pdf signatures". In: *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*. 2019, pp. 1–14.

[9] Jens Müller, Dominik Noss, Christian Mainka, Vladislav Mladenov, and Jörg Schwenk. "Processing dangerous paths". In: *Network and Distributed Systems Security Symposium*. NDSS. 2021.

[10] NIST. *NVD Vulnerability Search Results for "PDFium"*. URL: %5Curl%7Bhttps://nvd.nist.gov/vuln/search#/nvd/home?keyword=pdfium&resultType=records%7D (visited on 01/13/2026).

[11] Billy Rios, Federico Lanusse, and Mauro Gentile. *Adobe Reader Same-Origin Policy Bypass*. 2013. URL: https://www.sneaked.net/adobe-reader-same-origin-policy-bypass.

[12] Tobias Friese. *Visual Signature Spoofing in PDFs*. 2022.

[13] Titus Vollbracht, Sören Borgstedt, Lilian Hildebrandt, Christian Mainka, and Vladislav Mladenov. "When Documents Rewrite Themselves: Security Analysis of Self-Modifying PDFs". 2025.