Fine-Grained Isolation via the Endokernel in a Rust Web Browser

Aman Shanbhag
as212@rice.edu

Byron Harris
bbh3@rice.edu

Shaquille Que
stq1@rice.edu

1 Introduction

Browsers share the contradictory goals of being able to run general workloads while also maintaining security and performance for the user [3, 13]. Because of the very diverse content it must be able to process and render from the Web, the browser must utilize many libraries specialized for these tasks. These libraries are often low-level in nature in order to process large volumes of data. As a result, a vulnerability in any included library may compromise the security of the browser and thus the user [2, 10]. Several models have been proposed to mitigate this threat including RLBox on Firefox and process isolation on Chrome, but most lack in either security or performance.

However, modern browsers are increasingly being hardened against zero-day attacks. Doing so requires isolation of untrusted code, either from the outside or from libraries within. For example, Mozilla has implemented RLBox in Firefox, which is a framework that provides sandboxing via WebAssembly for libraries that Firefox uses for rendering. RLBox also supports isolation via NaCl (Google Native Client) modified to support SFI and Process Isolation [11].

On the other hand, Chrome implements Site Isolation that puts each site in their own process, but this incurs the performance overhead of IPC and does not prevent the renderer from compromising the entire browser [14].

We argue that this problem is well-suited for the Endokernel [8], an embedded security monitor within a single process that provides complete memory isolation via the creation of two privilege levels. Moreover, we argue that this model is best complemented with a type- and memory-safe language to provide a more enhanced security.

In this paper, we introduce Bowser, a Rust-based minimal browser that uses the Endokernel model to sandbox external libraries and isolate tabs. Our goal is to (1) demonstrate the feasibility and usability of a browser that uses the Endokernel model, and (2) demonstrate a fine-grained isolation approach to building browsers that isolate individual libraries and components of the browser.

Our contributions are the following: (1) a browser purpose-built for fine-grained isolation of libraries, and (2) the first practical use-case of the Endokernel model in running Rust code.

2 Background

While process-level isolation is traditionally used to guarantee that components running in separate processes are well-isolated, this usually incurs a performance overhead. Moreover, a browser implemented in an unsafe language, such as C or C++, are still vulnerable to bugs that can trigger sandbox escape [4, 12]. For example, JailbreakMe 3.0 escapes iOS sandboxing and gains access to the underlying operating system by exploiting the PDF parser in Safari.

On the other hand, Rust features a rich type system and ownership model that guarantees memory-safety and thread-safety, which eliminates classes of bugs entirely at compile-time [9]. Thus, a browser written in Rust can address many common security issues while maintaining performance [1]. For example, Mozilla reports that on a total of 70 vulnerabilities (with 43 being security related), Rust code resulted in fewer CVEs [5].

To further enhance security, we integrate the Endokernel model into our Rust-based browser to provide fine-grained isolation. The Endokernel functions as a monitor, embedded within individual processes, that defines the functionality for isolation and information passing between domains [8].

3 Methods and Plan

To evaluate our idea, we built Bowser, a minimal browser, from scratch in Rust. Bowser currently features HTML fetching, parsing, and rendering via HTTP and HTTPS. In the two weeks we plan to implement the Document Object Model (DOM), styling via Cascading Style Sheets (CSS), Hyperlinks, interactive scripts via JavaScript, Task Scheduling, and a Cookie Store.

In the current model, we import OpenSSL [7] for HTTPS and Druid [6] for rendering the graphical user interface (GUI). These crates (Rust libraries), along with other future dependencies, will be isolated via the Endokernel.

We aim to be feature complete by November 18th and have isolation implemented by November 25th.

4 Evaluation and Results

This section will contain performance benchmarks for Bowser both with and without isolation enabled. This is not yet applicable, since we are still developing the browser.

5 Discussion and Lessons Learned

We aim to discuss the architecture of the browser, the challenges of using the Endokernel to isolate rust code (specifically comparing crates vs C libraries), and the performance of Endokernel in our system. This is not yet applicable, since we are still developing the browser.

6 Conclusion

Bowser is an incredibly exciting example of the possibilities of the Endokernel. With minimal overhead, the Endokernel practically provided both fine-grain isolation to imported libraries and tabular isolation. While Bowser was not built to match the scale of enterprise web browsers, we hope that this project empowers researchers in those organizations (or at other large projects) to explore the Endokernel as a viable isolation technique.

References

[1] Brian Anderson, Lars Bergstrom, Manish Goregaokar, Josh Matthews, Keegan McAllister, Jack Moffitt, and Simon Sapin. Engineering the servo web browser engine using rust. In *Proceedings of the 38th International Conference on Software Engineering Companion*, ICSE '16, page 81–89, New York, NY, USA, 2016. Association for Computing Machinery.

[2] Adam Barth, Collin Jackson, Charles Reis, TGC Team, et al. The security architecture of the chromium browser. In *Technical report*. Stanford University, 2008.

[3] Xinshu Dong, Hong Hu, Prateek Saxena, and Zhenkai Liang. A quantitative evaluation of privilege separation in web browser designs. In Jason Crampton, Sushil Jajodia, and Keith Mayes, editors, *Computer Security – ESORICS 2013*, pages 75–93, Berlin, Heidelberg, 2013. Springer Berlin Heidelberg.

[4] Mozilla Firefox. Mozilla foundation security advisory 2020-01 security vulnerabilities fixed in firefox 72. https://www.mozilla.org/en-US/security/advisories/mfsa2020-01/, 2020.

[5] Mozilla Firefox. Mozilla foundation security advisory 2020-01 security vulnerabilities fixed in firefox 72. https://www.mozilla.org/en-US/security/advisories/mfsa2020-01/, 2020.

[6] Rust Foundation. Crate druid. https://docs.rs/druid/latest/druid/.

[7] Rust Foundation. Crate openssl. https://docs.rs/openssl/latest/openssl/.

[8] Bumjin Im, Fangfei Yang, Chia-Che Tsai, Michael LeMay, Anjo Vahldiek-Oberwagner, and Nathan Dautenhahn. The endokernel: Fast, secure, and programmable subprocess virtualization. arXiv, 2021.

[9] Steve Klabnik and Carol Nichols. *The Rust Programming Language (Covers Rust 2018)*. No Starch Press, 2019.

[10] Jin Liu and Chong Xu. Pwning microsoft edge browser: From memory safety vulnerability to remote code execution, 2018.

[11] Shravan Narayan, Craig Disselkoen, Tal Garfinkel, Nathan Froyd, Eric Rahm, Sorin Lerner, Hovav Shacham, and Deian Stefan. Retrofitting fine grain isolation in the firefox renderer. In *Proceedings of the 29th USENIX Conference on Security Symposium*, SEC'20, USA, 2020. USENIX Association.

[12] National Institute of Standards and Technology. Cve-2021-38013 detail. https://nvd.nist.gov/vuln/detail/CVE-2021-38013#vulnCurrentDescriptionTitle, 2021.

[13] Erica Weistrand and Sophie Ryrberg. *Security-performance tradeoffs in HTTPS implementations of browsers*. PhD thesis, 2020.

[14] Alexander Yip, Neha Narula, Maxwell Krohn, and Robert Morris. Privacy-preserving browser-side scripting with bflow. In *Proceedings of the 4th ACM European Conference on Computer Systems*, EuroSys '09, page 233–246, New York, NY, USA, 2009. Association for Computing Machinery.