FEAST'24: Sixth Workshop on Forming an Ecosystem Around Software Transformation

Ryan Craven
Office of Naval Research
Arlington, Virginia, USA
ryan@rcraven.net

Matthew Mickelson
MITRE
McLean, Virginia, USA
mmickelson@mitre.org

Abstract

The Sixth Workshop on Forming an Ecosystem Around Software Transformation (FEAST) revives the series, with the original five events taking place from 2016-2020. FEAST is concerned with all aspects of achieving effective, robust, and appraisable late-stage transformation of software for security. Late-stage transformations allow third parties to deeply tailor existing software to their mission, customizing it with little to no access to source code or support from the original developer.

Research has shown that late-stage software customization is of particular benefit to security-conscious software consumers who must use closed-source or source-free binary software components in mission-critical settings, or who must harden software against newly emerging attacks not anticipated during the software's original design and development. However, there is still a long way to go toward achieving sound and robust transformations whose holistic benefits to deployed software are fully appraisable. Motivated by these outstanding challenges, FEAST continues in its goal to form an active ecosystem of strategies and tools for accomplishing source-free binary code transformation reliably and on-demand.

CCS Concepts

• **Security and privacy → Software and application security**; • **Software and its engineering → Software post-development issues**.

Keywords

binary software; software debloating; software de-layering; software security hardening; binary rewriting; software transformation

ACM Reference Format:
Ryan Craven and Matthew Mickelson. 2024. FEAST'24: Sixth Workshop on Forming an Ecosystem Around Software Transformation. In *Proceedings of the 2024 ACM SIGSAC Conference on Computer and Communications Security (CCS '24), October 14–18, 2024, Salt Lake City, UT, USA.* ACM, New York, NY, USA, 2 pages. https://doi.org/10.1145/3658644.3691553

1 Introduction

In 2024, software is more bloated than ever [8]. Since the most recent FEAST in 2020, software size has continued to climb at an exponential rate. The proliferation of "smart" features into everyday consumer products has carried along with it a need for manufacturers across all industries to rapidly ship complex software functionality out to a host of cost-sensitive products. Nowhere has the change been more dramatic than the auto industry, where even low-end vehicles can have 100 ECUs and 100 million of lines of code, and advances in autonomous driving functions hold the potential for some new cars to push the 500 million lines of code mark [4]. That would place an amount of code equal to one quarter the size of all the software running Google's *entire* internet services catalog ten years ago [11], into every family vehicle.

While economics have always driven developers to ship software with lots of features for broad appeal, it is the advancement of efficient and easy-to-use code reuse practices along with the powerful gains they deliver to programmer productivity that enable the explosive growth we see. We find it important to stress that maximizing code reuse is not itself a bad thing: Consumers and businesses benefit from a rapid pace of less expensive, more feature-rich products, and developers spend their limited time more efficiently. But there is a growing problem that security-conscious consumers know all too well: Most of the code in any modern system is unnecessary or even potentially undesirable to its users [12]. One study found that, on average, only 10% of the functions in the most frequently used shared libraries in Ubuntu are ever invoked by common programs [13].

Identifying when unneeded or undesirable code is being added to a system is difficult, due in large part to an enormous amount of complexity that gets hidden behind slick abstractions: frameworks, middlewares, container orchestration, and so many libraries that today's popular languages all need their own package managers just to make things manageable [7]. We expect these conditions will persist, as the cost of the complexity seems cheap compared to the value gained. The costs, however, are only thought to be cheap because the market measures them at *development time*, where all the benefits (increased productivity resulting from the code reuse and abstractions) are being gained.

The follow-on costs (increased maintenance, features that later become bugs, expanded attack surfaces, and opaque software supply chains) are difficult to capture, and varyingly affect different segments of consumers. FEAST is a recognition of these costs. Rather than fight market forces directly, which is unlikely to be effective, the FEAST Workshop is devoted to improving the feasibility and effectiveness of *late-stage software transformation*. Late-stage transformations modify low-level software after it has been designed, developed, and compiled into a distributable product. Such technologies offer consumers the ability to customize software to their particular requirements, such as by removing unneeded features, stripping out unnecessary complexity, or adding hardened security

CCS '24, October 14–18, 2024, Salt Lake City, UT, USA

ACM ISBN 979-8-4007-0636-3/24/10
https://doi.org/10.1145/3658644.3691553

defenses against dangerous attacks. Source-free software transformation challenges of particular interest include:

- **software debloating**, which concerns the removal of software behaviors, code, or data that is unneccessary for a given consumer's needs;
- **software de-layering**, which removes levels of indirection or abstraction layers that impede efficiency;
- **software security hardening**, which concerns adding extra security checks and other defenses to code in order to thwart attacks;
- **post-deployment patching**, which allows binary code to be more easily modified to replace or remove functionalities;
- **attack surface discovery and reduction**, which discovers and mitigates potential opportunities for abuse and compromise of binary software products;
- **software self-healing**, which transforms software to detect and remediate faults unanticipated by its authors;
- **transformation-aware reverse-engineering**, which lifts low-level software to a higher-level form amenable to analysis, transformed, and then lowered back to executable form without sacrificing efficiency; and
- **low-level formal methods**, which extend automated theorem proving, model-checking, and type-based verification typically used at the source level for high assurance code down to executable binaries.

The goal of FEAST is to cultivate a robust ecosystem of these and other technologies relevant to practical, effective customization of binary software without the aid of source code or developer support.

1.1 The Continuing Need for FEAST

Since the most recent FEAST in 2020, numerous major events continue to motivate our vision. In 2021, an obscure feature added to the Log4j library almost eight years prior triggered a global cybersecurity emergency [5]. The cause was due to one user of the library adding a feature (support for JNDI lookups) to make their life more convenient, and their patch was committed by the maintainers less than 24 hours later [1].

Seeing the lack of scrutiny going into widespread code reuse, attackers began more heavily targeting overworked package maintainers [6], leading to a coordinated multi-year operation against the maintainer of XZ Utils compression library [2]. The Sunburst malware was discovered to have been hiding in Solarwinds Orion, a software product intended to *improve* security, for months undetected [14]. And unnecessary support for obscure image formats led to a zero-click vulnerability in iMessage [3].

The rapid and pervasive adoption of AI coding assistants is already having broad effects on how software is built and deployed [9]. One possibility is that complexity and opacity increase another level as the new technology leads to changes in behaviors. For instance, LLM hallucinations of package library names were observed being used in the wild [10]. As we progress toward FEAST's goal of practical and effective customization of binary software, consumers that bear a larger burden from the offset costs of increasing size and complexity will benefit as they become empowered to exert more rigor over and reshape the software they deploy.

2 Sixth Workshop Program

The sixth FEAST workshop consists of four full paper presentations, and ten talk proposal presentations. Only full papers were entered into official proceedings. The talk proposal was a lighter-lift submission type we created to incorporate more diverse perspectives. The acceptance rate was 93%, with many submissions being of high quality and relevant to scope. The presentations are organized into four sessions, each with a different theme tied to a property about late-stage transformations that we seek to improve: Soundness, Robustness, Appraisability, and Enabling Technology.

3 Workshop Organization

The following program committee members helped organize the 2024 FEAST Workshop:

- Ryan Craven (Office of Naval Research)
- Matthew Mickelson (MITRE)
- Jason Li (Trusted ST)
- Sukarno Mertoguno (Georgia Tech)
- Daniel Koller (Pennsylvania State University)
- Nathan Burow (MIT Lincoln Laboratory)

4 Acknowledgments

The workshop chairs for FEAST'24 wish to thank all authors for submitting papers, as well as past chairs Taesoo Kim, Dinghao Wu, Yan Shoshitaishvili, Mayur Naik, Adam Doupé, Zhiqiang Lin, Long Lu, and Kevin Hamlen for their stewardship of the FEAST community over the years.

References

[1] Apache. 2013. JNDI Lookup plugin support. ASF JIRA, https://issues.apache.org/jira/browse/LOG4J2-313.
[2] Fred Bals. 2024. What is the Xz Utils Backdoor: Everything you need to know about the supply chain attack. Synopsys Blog, https://www.synopsys.com/blogs/software-security/xz-utils-backdoor-supply-chain-attack.html.
[3] Ian Beer and Samuel Gross. 2021. A deep dive into an NSO zero-click iMessage exploit: Remote Code Execution. *Google Project Zero* (2021).
[4] Robert Charette. 2021. How Software is Eating the Car. *IEEE Spectrum* (2021).
[5] CISA. 2021. Mitigating Log4Shell and Other Log4j-Related Vulnerabilities. Cybersecurity Advisory, https://www.cisa.gov/news-events/cybersecurity-advisories/aa21-356a.
[6] Ruian Duan, Omar Alrawi, Ranjita Pai Kasturi, Ryan Elder, Brendan Saltaformaggio, and Wenke Lee. 2021. Towards Measuring Supply Chain Attacks on Package Managers for Interpreted Languages. In *NDSS*.
[7] Johannes Düsing and Ben Hermann. 2022. Analyzing the Direct and Transitive Impact of Vulnerabilities onto Different Artifact Repositories. *Digital Threats* 3, 4 (2022).
[8] Bert Hubert. 2024. Why Bloat is Still Software's Biggest Vulnerability: A 2024 plea for lean software. *IEEE Spectrum* 61, 4 (2024), 22–50.
[9] Jan H. Klemmer, Stefan Albert Horstmann, Nikhil Patnaik, Cordelia Ludden, Cordell Burton Jr, Carson Powers, Fabio Massacci, Akond Rahman, Daniel Votipka, Heather Richter Lipford, Awais Rashid, Alena Naiakshina, and Sascha Fahl. 2024. Using AI Assistants in Software Development: A Qualitative Study on Security Practices and Concerns. In *ACM CCS 2024*.
[10] Bar Lanyado. 2024. Diving Deeper into AI Package Hallucinations. Lasso Security Blog, https://www.lasso.security/blog/ai-package-hallucinations.
[11] Rachel Potvin. 2015. Why Google Stores Billions of Lines of Code in a Single Repository. Systems @Scale, https://youtu.be/W71BTkUbdqE.
[12] Anh Quach, Rukayat Erinfolami, David Demicco, and Aravind Prakash. 2017. A Multi-OS Cross-Layer Study of Bloating in User Programs, Kernel and Managed Execution Environments. In *Proceedings of the 2017 Workshop on Forming an Ecosystem Around Software Transformation (FEAST '17)*.
[13] Anh Quach, Aravind Prakash, and Lok Yan. 2018. Debloating software through Piece-Wise compilation and loading. In *27th USENIX security symposium (USENIX Security 18)*.
[14] Kim Zetter. 2023. The Untold Story of the Boldest Supply-Chain Hack Ever. *Wired* (2023).