Universität des Saarlandes
MI Fakultät für Mathematik und Informatik
Department of Computer Science

Bachelorthesis

Capabilities as a Solution against Tracking Across Android Apps

submitted by

Tim Christmann

on November 17, 2025

Reviewers

Dr. Sven Bugiel

Noah Mauthe

Eidesstattliche Erklärung

Ich erkläre hiermit an Eides statt, dass ich die vorliegende Arbeit selbständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel verwendet habe.

Statement in Lieu of an Oath

I hereby confirm that I have written this thesis on my own and that I have not used any other media or materials than the ones referred to in this thesis.

Saarbrücken, November 17, 2025,

(Tim Christmann)

Einverständniserklärung

Ich bin damit einverstanden, dass meine (bestandene) Arbeit in beiden Versionen in die Bibliothek der Informatik aufgenommen und damit veröffentlicht wird.

Declaration of Consent

I agree to make both versions of my thesis (with a passing grade) accessible to the public by having them added to the library of the Computer Science Department.

Saarbrücken, November 17, 2025,

(Tim Christmann)

Write dedication here

Abstract

Trusted Web Activities and Custom Tabs enable Android developers to seamlessly integrate web content into native applications, offering a powerful tool for features such as Single Sign-On and in-app monetization. However, as shown by HyTrack, this integration also introduces severe privacy risks by blurring the boundary between web and app contexts, allowing persistent tracking through the browser's shared cookie storage.

In this work, we propose a novel mitigation framework that applies capability-based access control to browser cookie handling. Cookie access is encapsulated in fine-grained, identity-bound capabilities, ensuring that only trusted first-party or explicitly authorized third-party web servers – defined by a developer-controlled policy – can access the shared browser state. All other untrusted third-party servers are confined to isolated, in-app cookie jars. This empowers well-meaning developers to continue leveraging third-party libraries while preventing them from performing unauthorized cross-app tracking. At the same time, essential features such as Single Sign-On and personalized content delivery remain fully functional. Our approach balances privacy and usability, allowing tracking-resistant web-app integration without degrading the user experience.

Acknowledgements

I would deeply like to thank Dr. Sven Bugiel and Noah Mauthe for their support and guidance.

Contents

Abstract **vii**

Acknowledgements **ix**

List of Figures **xiii**

List of Tables **xv**

1 Introduction **1**

2 Background **3**

3 Method: Preventing Cross-App Tracking in CTs and TWAs via Capabilities **5**
3.1 The Threat Model 5
3.2 Root Causes of HyTrack and Our Mitigation Strategy 6
3.3 Proposed Solution: Capability-Based Cookie Isolation 7
3.4 Benefits 9
3.5 Hypotheses 9
3.6 Components 10

4 Implementation **11**
4.1 Custom Installer 12
4.2 Fenix Browser 12
4.3 Libraries 12
4.3.1 Byetrack/Mitigation 12
4.3.2 AndroidX Browser 12
4.4 HyTrack Demo Apps 12
4.5 TrackerOne 12
4.6 Launcher 12
4.7 Test App 12
4.8 Evil App 12

5 Evaluation **13**

6 Discussion **15**

7 Related Work **17**

8 Future Work **21**

9 Conclusion **23**

Bibliography **25**

Appendix **27**

List of Figures

List of Tables

Chapter 1

Introduction

In recent years, Android applications have increasingly leveraged web content within their interfaces to enhance user experience and streamline features such as authentication and monetization. To enable this, developers often use Custom Tabs (CTs) and Trusted Web Activities (TWAs), technologies that provide seamless, browser-backed web integration while maintaining native-like performance and features. This approach allows web-based functionality like Single Sign-On (SSO), such as login via Facebook or embedded advertising, without forcing users to switch between app and browser.

However, these benefits come at a cost. CTs and TWAs share the browser's cookie storage across all apps, enabling continuity of web sessions – but also opening serious privacy vulnerabilities. Recent research by Wessels et al. introduced HyTrack [1], a novel tracking technique that exploits this shared browser state to persistently track users across different applications and the web, even surviving device changes, cookie clearing, or browser switching. HyTrack works by embedding a third-party library into multiple unrelated apps. Each app, unaware of the library's true purpose, opens a CT or TWA to the same tracking domain. This domain sets a unique identifier in a cookie, stored in the browser's shared cookie jar. When another app using the same library loads content from the same domain, the cookie is sent, enabling the tracker to correlate activity across apps and even into regular browser use. Due to Android's backup mechanisms, the tracking ID can be restored even after a factory reset, rendering it more persistent than the evercookie [2].

This thesis explores whether capabilities, a fine-grained access control model, can be used to limit or prevent these privacy issues without breaking legitimate use cases of CTs and TWAs. Specifically, we aim to design and evaluate a framework that allows developers to retain the benefits of third-party libraries (e.g., SSO or monetization) without exposing users to invisible, cross-app tracking. The framework should be simple to integrate,

practical in real-world deployments, and minimize interference with already established app workflows.

Chapter 2

Background

Chapter 3

Method: Preventing Cross-App Tracking in CTs and TWAs via Capabilities

Custom Tabs (CTs) and Trusted Web Activities (TWAs) enable a seamless integration between apps and the web by sharing browser state – most notably, session cookies. While this enhances user experience, it also introduces a significant privacy risk: third-party libraries can exploit the shared browser state to track users across multiple apps in which they are embedded. For instance, HyTrack demonstrates that a single shared third-party library used across multiple apps is sufficient to persistently identify and track users, bypassing conventional browser and OS sandboxing mechanisms.

3.1 The Threat Model

The developer of an Android application unknowingly includes a third-party library that uses the HyTrack technique for their own purposes, such as advertising. We want to prevent this library from tracking the apps user across multiple apps and empower the app developer to use any third-party library without risking user privacy in regards to cross-app tracking via HyTrack.

For this, we assume that the app developer is not malicious and does not intend to violate user privacy. Otherwise, they could simply choose to not use our mitigation framework and directly use the HyTrack library on their will.

...

3.2 Root Causes of HyTrack and Our Mitigation Strategy

The feasibility of HyTrack relies on the following underlying weaknesses in the CT and TWA security model:

- **Implicit and Persistent Cookie Sharing**: All apps using CTs or TWAs inadvertently access a single, persistent global browser cookie jar, regardless of whether such sharing is intended by the developer. This cookie state persists across app launches and can survive typical user efforts to clear tracking data.
- **Lack of App Context in Browser**: The browser is unaware of which app initiated a given request, and therefore cannot enforce app-specific isolation or developer-defined policies. As a result, all apps using CTs or TWAs share the same cookie state, even if their purposes and trust levels differ.
- **Unrestricted Third-Party Inclusion**: Third-party libraries embedded in multiple apps gain access to the shared browser cookie jar, allowing them to identify users across app boundaries.

Our approach addresses these issues by introducing explicit, developer-defined policies and browser-enforced **capability tokens**:

- **Explicit Cookie Isolation**: Cookies are stored only if the domain is explicitly declared as trusted or untrusted in the app's policy, i.e. a capability for the domain exists. Based on the token(s) received by the app, the browser either stores the wrapped cookie in the shared cookie jar or returns it to the app for isolated local storage. This reverses the implicit sharing model and ensures that even persistent HyTrack identifiers cannot be used for cross-app tracking, as each app maintains a distinct, app-scoped identity – assuming the policy is correctly configured.
- **App-Aware Browser Context**: Each capability encodes an *App ID*, allowing the browser to enforce per-app cookie policies and prevent unintended delegation between apps. Additionally, capabilities include an *App Version Number*, enabling the browser to detect app updates and invalidate tokens that no longer reflect the current policy.
- **Capability-Scoped Access Control**: Malicious third-party domains are confined to app-local storage and cannot access the shared cookie jar, thereby blocking cross-app tracking via embedded libraries. At the same time, legitimate use cases, such as Single Sign-On (SSO), remain supported by granting trusted domains the necessary capabilities to access the shared cookie jar.

By making browser state access explicit, app-aware and scoped, our solution enforces cookie isolation across apps – particularly against embedded third-party libraries – thereby neutralizing HyTrack's core tracking mechanisms while preserving the seamless user experience of CTs and TWAs and supporting legitimate app-web integrations.

3.3 Proposed Solution: Capability-Based Cookie Isolation

To mitigate cross-app tracking caused by shared browser state, we propose a developer-defined policy mechanism, combined with cryptographically enforced capabilities, to control how cookies are managed and isolated on a per-app basis.

During app installation, the installer extracts and transmits the policy to the browser. This policy – declared by the app developer in the manifest and config file – defines the following:

- A list of trusted domains (e.g., the developer's own domains or explicitly permitted third parties).
- A list of untrusted domains (e.g., third-party ad libraries).
- Optionally, a set of predefined cookie names expected to be used by the respective servers.

According to this policy, the browser creates capabilities for the app wrapping important metadata into a secure structure, encoded and signed, similar to JSON Web Tokens (JWT) [3]. The capability structure includes the following fields:

- **Signature:** Ensures the capability was issued by the browser and has not been tampered with.
- **App ID:** Identifies the origin of the request, ensuring that only authorized apps can hold a given capability. This field is essential to prevent implicit delegation – without it, a receiving app could exploit the capabilities granted to the sending app and potentially gain unauthorized access.
- **Domain:** Specifies the designated destination web server, ensuring the capability is only sent to the correct endpoint.
- **App Version Number**: Indicates the version of the app to detect potential policy changes and ensure consistency between app and policy. In case of a version mismatch, the browser rejects these capabilities.

- **Rights:** Define the scope of access granted to the app – for example, whether it may request the browser to read cookies. This restriction is essential to prevent libraries from exploiting browser access to extract capability values, which could otherwise be used for tracking.
- **Global Jar Flag:** Determines whether the shared cookie jar or an app-specific cookie jar should be used.

Consequently, these capabilities act as **authorization tokens** for cookie access and fall into one of the following categories:

- **Wildcard Capabilities:** Allow a web server to define arbitrary cookie names and values by leaving these fields unspecified in the capability, which are later filled by the browser when it receives a cookie from the web server. This mechanism effectively transforms conventional cookies into structured capability tokens that encapsulate metadata in addition to the cookie data. Hence, the app receives such a capability for each domain registered in the policy, with the difference that wildcards for trusted domains have the global jar flag set to true, while those for untrusted domains have it set to false.
- **Predefined Capabilities:** Define fixed cookie names as specified in the policy and can be stored either in the app-specific cookie jar or the browser's shared cookie jar, depending on the configuration of the global jar flag. This provides fine-grained control over cookie scoping and allows developers to isolate their app from the browser's state when desired, while still enabling seamless web-app integration.
- **Ambient Capabilities:** Instruct the browser to revert to default behavior by storing all received cookies in the global cookie jar and including them in subsequent requests to the corresponding web server. These have the global jar flag set by default and augment a wildcard capability with the domain name, cookie name, and corresponding value.

  Ambient capabilities provide no security guarantees and serve solely as a fallback mechanism for maintaining functionality when no explicit developer policy is provided.

When the app opens a URL via CT or TWA, it includes a list of capabilities along with the regular intent containing the target URL. The browser parses each capability and verifies its authenticity by validating its signature and associated fields.

The communication between browser and web server remains unchanged, i.e. the browser sends Request and Set-Cookie headers as usual and the web server responds with a

(customized) Response and possibly new Cookies. Upon receiving cookies, the browser matches them against the capabilities provided by the app:

1. If the cookie name is predefined, only the value is updated in the capability.
2. If it is new and a matching wildcard capability exists, the browser creates a new capability by duplicating the wildcard, setting the name and value accordingly and storing it in the corresponding cookie jar. The initial wildcard capability is then returned to the app.
3. If the cookie is new and no wildcard capability is available, the browser discards it to prevent unauthorized or unexpected state persistence.

3.4 Benefits

Beyond eliminating cross-app tracking through HyTrack and its postulated goals, we identify the following additional benefits offered by this approach:

B1) **Fine-Grained Control**: Developers can specify which cookies are shared and which are isolated, allowing for a more tailored approach to privacy.

B2) **No Browser State for Apps**: The browser is stateless with respect to app-specific capabilities, as these are retained and transmitted by the app with each intent invocation.

B3) **No Third-Party Code Changes**: The web server does not need to be aware of the capabilities or make any changes to its code, as the browser handles the capability management.

B4) **Backwards Compatibility**: In case the developer does not implement its own policy, it will default to allowing any domain using the shared browser state, consequently behaving like the state-of-the-art cookie management.

3.5 Hypotheses

The evaluation is based on the following hypotheses:

H1) Passing capabilities with each intent enables the browser to validate and manage cookies securely and effectively. It also ensures the browser does not need to store app states.

H2) The custom installer reliably transmits developer-defined policies to the browser.

H3) The proposed mechanism is compatible with the original three goals of HyTrack as outlined in Section 5.

H4) The mechanism significantly mitigates cross-app tracking by isolating browser state based on trusted policy definitions.

H5) The use of *Ambient Capabilities* ensures backwards compatibility with existing systems.

H6) In the case of invalid or missing capabilities, the browser by default discards the received Cookie, preventing fallback to shared state.

H7) The proposed approach requires no modifications to existing third-party web server code.

3.6 Components

In summary, implementing the proposed solution requires a combination of new components and modifications to existing ones:

- **Manifest** and **config file** for the developers to define which origins should retain shared browser state.
- A **custom installer** to extract and send the developer-defined policy to the browser upon app installation.
- The **browser** to function as the Policy Enforcement Point. For this, new functionality for validating the received policy, creating capabilities accordingly and validating them is necessary. In addition, the browser needs to be capable of storing a secret key for signing capabilities and verifying the signature. Of course, functionality for making decisions based on the capability's context is essential, i.e. where and how to store the capability.
- The **app** must send capabilities as an additional parameter with each intent and store the capabilities returned by the browser for future use. It must also support functionality to send specific capabilities to the browser.
- **Capabilities** to serve as secure wrappers for cookie data, including the app ID, target domain, global jar flag, and associated access rights.

Chapter 4

Implementation

To demonstrate the feasibility of our proposed mitigation strategy, we developed a proof-of-concept installer application, a new library that ships the changes and modified the androix browser library to inject our capability tokens for each call to launch a Custom Tab (CT) or Trusted Web Activity (TWA).

In this proo-of-concept, we chose the firefox mobile browser (Fenix) to act as the policy enforcement point, but it could have been any other browser the authors of HyTrack [1] found out to be vulnerable to their attack, such as Chrome or Brave.

We modified the proof-of-concept apps provided by the authors of HyTrack and provide a test application for more insight into the framework's behavior. For completeness, we also provide a "evil" acting app that demonstrates what the HyTrack library included in tan app could do to circumvent the mitigation.

Put a diagram here that shows the components and their interactions!!!

4.1 Custom Installer

4.2 Fenix Browser

4.3 Libraries

4.3.1 Byetrack/Mitigation

4.3.2 AndroidX Browser

4.4 HyTrack Demo Apps

4.5 TrackerOne

4.6 Launcher

4.7 Test App

4.8 Evil App

Chapter 5

Evaluation

To assess the effectiveness of our proposed mitigation strategy, we adopt the three primary goals identified by the authors of HyTrack as essential for any viable defense:

1) **Support all features of the web platform:** The solution must allow applications to display fully functional web content, including support for cookies, JavaScript, and modern APIs.

2) **Preserve seamless integration:** The user experience must remain uninterrupted. This includes avoiding obtrusive permission dialogs and maintaining smooth transitions between native and web content.

3) **Enable controlled access to shared browser state:** While isolation is required to prevent cross-app tracking, legitimate scenarios such as Single Sign-On (SSO) must remain functional.

These criteria reflect the fact that HyTrack exploits standard Android behavior – specifically, the shared browser state exposed through Custom Tabs and Trusted Web Activities – rather than relying on unauthorized access or system vulnerabilities. Therefore, naive approaches like disabling shared cookies entirely would break common use cases and are not acceptable.

To validate these hypotheses, we will build on the open-sourced measurement tooling and proof-of-concept applications provided by the authors of HyTrack. Specifically, we plan to:

- Replicate the original HyTrack experiments under controlled conditions using two unrelated Android apps that embed the tracking library (similar to the HyTrack demo).

- Apply the mitigation framework and compare observed behavior against the baseline.

We will collect and analyze the following metric:

- Number of Capabilities created and used by the browser.

In doing so, we aim to demonstrate that the proposed solution effectively blocks HyTrack's cross-app tracking channel while preserving compatibility with existing web features and maintaining seamless user experience. Additionally, we will show that this strategy can be used by developers to control cookie transmission in a fine-grained and policy-driven manner to enable safer and more transparent integration of third-party web content.

Chapter 6

Discussion

Chapter 7

Related Work

Tracking mechanisms are typically divided into two broad categories: stateful and stateless tracking. Stateless tracking, also known as fingerprinting, infers a user's identity based on a combination of device-specific attributes. Consequently, this method is hard to detect and block, but is also inherently less reliable, as small system changes may alter the fingerprint and disrupt identification.

Instead, stateful tracking relies on storing unique identifiers on the client device, most commonly through cookies or local storage. When a user revisits a site or interacts with embedded third-party content across domains, these identifiers are sent along with requests, allowing persistent recognition. While straightforward and highly effective, stateful tracking has become increasingly restricted through browser policies (e.g., third-party cookie blocking) and mobile platform changes such as the ability to disable the Google Advertising ID (GAID) on Android.

This problem not only affects the web, but also extends into the mobile ecosystem, as recently demonstrated by the Facebook Localhost Scandal [4] that exposed a covert tracking method used by Meta and Yandex on Android. In this case, their apps (e.g., Instagram) silently listened on localhost ports to receive browser tracking data – such as mobile browsing sessions and web cookies – sent from websites embedding Meta Pixel or Yandex scripts. This allowed the apps to link web activity to logged-in users, bypassing the browser's and Android's privacy protections. Although the practice was discontinued shortly after public disclosure, it highlighted a critical privacy gap between web content and native apps on mobile platforms.

HyTrack [1] demonstrates a novel cross-app and cross-web tracking technique in the Android ecosystem by exploiting the shared cookie storage between Custom Tabs (CTs) and Trusted Web Activities (TWAs). This allows persistent tracking of users across

multiple applications and the browser, even surviving user efforts to reset or sanitize their environments. The need to address HyTrack becomes even more critical in light of additional research on Custom Tabs. Beer et al. [5] conducted a comprehensive security analysis of CTs and revealed that they can be exploited for state inference, SameSite cookie bypass, and UI-based phishing attacks. Their work further shows that Custom Tabs are widely adopted, with over 83% of top Android apps using them, often via embedded libraries. These findings reinforce that CTs are a high-value attack surface and that the shared browser state – central to HyTrack – has broader security implications. As TWAs are a specialized form of CTs, they are similarly affected, further enabling the tracking to be fully disguised.

While HyTrack highlights a serious privacy vulnerability, no concrete mitigation has been proposed that balances privacy with the legitimate need for seamless web integration – such as Single Sign-On or ad delivery – within mobile apps. This can be seen by taking a closer look at the two possible mitigation strategies discussed by the authors, namely Browser State Partitioning and Forced User Interaction. Modern browsers prominently adopt state partitioning to combat third-party tracking. Firefox's Total Cookie Protection (TCP) [6] and Safari's Intelligent Tracking Prevention (ITP) [7] both enforce per-site cookie jars, thereby limiting cookie-based cross-site tracking.

However, both approaches introduce significant drawbacks. Browser state partitioning would allow each app to use its own cookie storage and hence prevent cross-app tracking. The seamless integration of web content remains intact, as no changes to the UI are necessary, but by completely removing the browser's shared state, benign uses like Single Sign-On (SSO) or ad personalization would be broken. Google is actively working on a similar mechanism under the name CHIPS (Cookies Having Independent Partitioned State) [8]. CHIPS allows third-party cookies to be partitioned by the top-level site with an optional *Partitioned* flag, enabling legitimate services like SSO to maintain function while avoiding broad tracking vectors. However, CHIPS is not applicable to Android's embedded web contents like CTs or TWAs, as the top-level site is the tracker itself. Our solution can be seen as extending this paradigm to the app level.

In contrast, Forced User Interaction avoids these problems by allowing the browser to use its shared cookie storage. But this introduces a significant usability issue, as the user is forced to interact with the browser every time a web content is loaded, which not only degrades user experience but also breaks seamless integration of web content into the app. Furthermore, this approach hands control and responsibility to the user, which is not ideal from a security perspective, as the user might be unaware of the consequences of their actions and may inadvertently enable tracking by failing to interact with the browser as required. Other strategies, such as limiting CTs and TWAs to First-Party

Domains or disabling them for specific domains via browser options ultimately reflect the aforementioned approaches, relying on either browser state partitioning or forced user interaction. Therefore, these are not effective countermeasures against HyTrack.

This work addresses this gap by proposing a capability-based access control framework for Android applications using CTs and TWAs. By wrapping Cookies into fine-grained capability tokens – created by the browser according to the app developer's policy –, the browser decides which cookies are stored in the shared cookie jar and which are stored in the app-specific storage, depending on a flag analogous to CHIPS' *Partitioned* attribute. This ensures that there is no cross-app tracking possible for untrusted third-party libraries, as each app stores its own tracking cookie. As the shared cookie storage still exists for domains declared as first-party or trusted by the app developer, legitimate uses of the shared browser state (e.g. SSO) are preserved. Seamless integration of web content is also preserved, as there is no need for user interaction or changes to the UI. Thus, in contrast to prior discussed mitigation strategies, this approach provides developers with a practical and enforceable way to render cross-app tracking infeasible.

Our interpretation of capability tokens is inspired by JSON Web Tokens (JWTs) [3], which are widely used in web authentication to encode claims about a user or a session in a secure, verifiable manner. Instead of storing user information directly on the server upon receiving a POST request, JWTs allow the server to issue a signed token that contains the necessary claims, which the client can then present in subsequent requests. As a result, the server does not need to maintain session state, as the token itself carries all the information needed and can verify via the signature that the token has not been tampered with. For this purpose, JWTs consist of three components separated by dots: a header that specifies the token type and algorithm for encoding and decoding it, a payload for the actual data, and a signature of the first two parts after base64 encoding that ensures the integrity of the token. Our approach extends this idea by including the cookie information and other metadata in the token's payload, and by establishing a communication channel between the browser and the app: the browser issues these tokens according to the app's policy, and the app presents them in subsequent requests to either access the browser's shared cookie jar or store cookies in its own app-local storage.

The work of Georgiev et al. titled "Breaking and Fixing Origin-Based Access Control in Hybrid Web Applications" [9] highlights critical failures in how hybrid apps enforce origin boundaries. Specifically, they show that WebViews and hybrid frameworks often bypass or misapply the Same-Origin Policy (SOP), enabling attackers to inject or reuse authentication tokens across apps and domains. Their proposed mitigation involves reintroducing stricter origin enforcement tied to app identities. Our approach builds

on this idea by using capability tokens to encode both the origin and the app context explicitly, thereby preventing unauthorized reuse or delegation.

Chapter 8

Future Work

Chapter 9

Conclusion

Bibliography

[1] M. Wessels, S. Koch, J. Drescher, L. Bettels, D. Klein, and M. Johns, "Hytrack: Resurrectable and persistent tracking across android apps and the web," in *34th USENIX Security Symposium (USENIX Security 25)*. Seattle, WA: USENIX Association, Aug. 2025.

[2] S. Kamkar, "Evercookie," *URl: http://samy. pl/evercookie*, 2010.

[3] M. B. Jones, J. Bradley, and N. Sakimura, "JSON Web Token (JWT)," RFC 7519, May 2015. [Online]. Available: https://www.rfc-editor.org/info/rfc7519

[4] LocalLeaks, "Tracking users with localhost: Facebook's covert redirect abuse," https://localmess.github.io/, 2023.

[5] P. Beer, M. Squarcina, L. Veronese, and M. Lindorfer, "Tabbed out: Subverting the android custom tab security model," in *2024 IEEE Symposium on Security and Privacy (SP)*, 2024, pp. 4591–4609.

[6] Mozilla, "Firefox's total cookie protection," 2021, https://developer.mozilla.org/en-US/docs/Web/Privacy/State_Partitioning.

[7] Apple, "Intelligent tracking prevention," 2020, https://webkit.org/blog/10218/full-third-party-cookie-blocking-and-more/.

[8] Google, "Cookies having independent partitioned state (chips)," https://github.com/privacycg/CHIPS, 2023.

[9] M. Georgiev, S. Jana, and V. Shmatikov, "Breaking and fixing origin-based access control in hybrid web/mobile application frameworks," in *NDSS symposium*, vol. 2014, 2014, p. 1.

Appendix

TODO

Use of Generative Digital Assistants

used Claude Sonnet Model 4.0 embedded in Visual Studio Code exclusively for understanding the firefox codebase and to help linking my code additions together.

Models like ChatGpt and Claude also used for debugging purposes (copy paste and let it try to fix the code or to explain obscure error messages).

...