Attributing Open-Source Contributions is Critical but Difficult: A Systematic Analysis of GitHub Practices and Their Impact on Software Supply Chain Security

Jan-Ulrich Holtgrave[*], Kay Friedrich[*], Fabian Fischer[*], Nicolas Huaman[†], Niklas Busch[*], Jan H. Klemmer[*], Marcel Fourné[‡], Oliver Wiese[*], Dominik Wermke[§], and Sascha Fahl[*]
[*]CISPA Helmholtz Center for Information Security, Germany, {jan-ulrich.holtgrave,kay.friedrich, fabian.fischer,niklas.busch,jan.klemmer,oliver.wiese,sascha.fahl}@cispa.de
[†]Leibniz University Hannover, Germany, huaman@sec.uni-hannover.de
[‡]Paderborn University, Germany, marcel.fourne@uni-paderborn.de
[§]North Carolina State University, USA, dwermke@ncsu.edu

Abstract*—Critical open-source projects form the basis of many large software systems. They provide trusted and extensible implementations of important functionality for cryptography, compatibility, and security. Verifying commit authorship authenticity in open-source projects is essential and challenging. Git users can freely configure author details such as names and email addresses. Platforms like GitHub use such information to generate profile links to user accounts. We demonstrate three attack scenarios malicious actors can use to manipulate projects and profiles on GitHub to appear trustworthy. We designed a mixed-research study to assess the effect on critical open-source software projects and evaluated countermeasures. First, we conducted a large-scale measurement among 50,328 critical open-source projects on GitHub and demonstrated that contribution workflows can be abused in 85.9% of the projects. We identified 573,043 email addresses that a malicious actor can claim to hijack historic contributions and improve the trustworthiness of their accounts. When looking at commit signing as a countermeasure, we found that the majority of users (95.4%) never signed a commit, and for the majority of projects (72.1%), no commit was ever signed. In contrast, only 2.0% of the users signed all their commits, and for 0.2% of the projects all commits were signed. Commit signing is not associated with projects' programming languages, topics, or other security measures. Second, we analyzed online security advice to explore the awareness of contributor spoofing and identify recommended countermeasures. Most documents exhibit awareness of the simple spoofing technique via Git commits but no awareness of problems with GitHub's handling of email addresses.

I. Introduction

Open-source software (OSS) is the backbone of the modern digital world and the software supply chain [1]. This importance of open-source software makes it a lucrative target for attackers. Attackers can impact many dependent projects and users by compromising a single OSS package. In February 2024, security researchers identified over 100,000 malicious repositories on GitHub that turned out to be fake clones of original repositories and contained obfuscated malicious code [2]. Additional incidents such as *Solarwinds*, the fake *VMConnector* on PyPi, and most recently *XZ* further highlight the threat potential [3]–[5]. Hence, Software Supply Chain (SSC) security is receiving more and more attention to protect vulnerable components, which is also reflected by the 2021 U.S. Presidential *Executive Order 14028* emphasizing the criticality of the software supply chain and striving for massive investment in its security [6]. As a recent follow-up in March 2024, CISA published its Secure Software Development Attestation Form (SSDAF), naming measures like regular *Trust Auditing* and *Software Provenance* as prerequisites for a secure SSC [7].

In summary, security is becoming more and more critical for open-source software. Technical tools such as the *OpenSSF Scorecard* [8] can help as an indicator for security practices employed in an OSS project. Still, previous research shows that trust in the other contributors is sometimes more important [9]. In his Turing Award lecture in 1984, Ken Thompson illustrated the need to trust contributors with a famous statement: *"To what extent should one trust a statement that a program is free of Trojan horses? Perhaps it is more important to trust the people who wrote the software."* [10]. However, trusting the wrong people led to disastrous security incidents (e. g., XZ [5]).

When looking at the reasons why people trust other developers in the decentralized and often anonymous open-source environment, the social coding platforms on which the code is hosted move to the center of attention [9], [11]. Social coding platforms, such as GitHub or GitLab, act as Version Control System (VCS) servers that host code for OSS projects and as social networks for OSS developers [11]–[13]. OSS developers can create user profiles, depict their contributions, earn badges, present their projects, and follow each other to not miss out on new endeavors of their favorite OSS developers [14]. A lot of the information used to generate user profiles and project sites is based on easily manipulable Git metadata, e. g. the username, and email addresses for Git commits. Both can

Network and Distributed System Security (NDSS) Symposium 2025
24 - 28 February 2025, San Diego, CA, USA
ISBN 979-8-9894372-8-3
https://dx.doi.org/10.14722/ndss.2025.240613
www.ndss-symposium.org

be freely configured by developers. GitHub uses this metadata to generate links from commits to user profiles; exploiting this behavior, attackers can spoof commit authorship and OSS project contributions [15]. As a countermeasure, GitHub and organizations like the Open Source Security Foundation (OpenSSF) and the Linux Foundation recommend using Git's built-in commit signing mechanism to add authenticity to a commit and prevent impersonations [16]–[19]. However, in the light of previous experiences with the adoption of signatures like digital signatures for email with OpenPGP or S/MIME [20], the question arises whether signatures are adopted by users and if GitHub's handling helps against the attacks. This is backed by a statement of Scott Chacon, co-founder of GitHub and co-author of the most widely used Git documentation `git-scm`, who confessed on his blog that his article about commit signing was flawed (*"I may have misled a generation"*) [16], [21]. While GitHub acknowledges the risk stemming from their handling of Git metadata [17], our work illustrates a false assessment.

We shed light on the security issues created by social coding platforms' interaction with untrustworthy Git metadata and demonstrate how trust-creating user interfaces could be manipulated. Furthermore, we investigate whether current countermeasures such as signed commits help to address this problem and how they are currently used by critical OSS projects and covered by security advice for developers.

RQ1 *How credible are open-source code contributions displayed on social coding platforms?* GitHub user profiles are fueled by Git metadata and illustrate contributors' activities, and participation in OSS projects is a critical source of contributors' reputations [22]–[24]. We demonstrate that this metadata is not robust against manipulation, which allows attackers to impersonate contributors and hijack their commits. We illustrate widespread implications for critical OSS projects.

RQ2 *How vulnerable are critical open-source projects to these weaknesses, and what countermeasures are in place?* Showing the weaknesses under lab conditions is one thing, but we measure the risk for real projects. We investigate the handling of the Git metadata weaknesses for 50,628 critical OSS projects and 578,897 associated contributors on GitHub. We also measure how GitHub's advice on signing commits is used in these projects.

RQ3 *How does online advice for open source contributors address the challenges of unreliable Git metadata related to contributor attribution?* Git metadata can easily be manipulated and might mislead users of social coding platforms into making false assumptions about the reputation and activities of contributors. We analyze 55 pieces of security advice for OSS contributors, focusing on problem awareness and proposed countermeasures.

RQ4 *What can Git and Git-based social coding platforms do to provide stronger protection against impersonation attacks, and how can the reliability of authorship attributions for Git commits be improved?* Based on our findings, we propose several changes for Git and GitHub and offer advice for maintainers and OSS developers.

We conducted a mixed-methods study to address our research questions. First, we demonstrate three attack scenarios for manipulating Git metadata that can be used to forge representations on the official GitHub website and refer to previous attacks that have been utilizing them. Second, we took the source code repositories of the top 50,628 OSS packages by dependents count and analyzed their 26,170,564 commits to estimate the attack surface. We also evaluated the repositories for present countermeasures like commit signing and assessed the added protection. Lastly, we leveraged a Gray Literature Analysis (GLA) to determine whether the weaknesses are known and what countermeasures are recommended, e.g., by social coding platforms. We analyzed 55 online security advice resources to investigate sources OSS contributors might consult on this topic.

With our paper, we make the following contributions:

- **Contributor Impersonations in OSS Projects.** We demonstrate three ways to tamper with Git's metadata to manipulate attributions on GitHub: *Contributor Spoofing*, *Reputation Hijacking*, and *Contribution Hijacking*.
- **Measuring Weaknesses in Critical OSS Projects.** We measure which and how many critical OSS projects and their contributors are vulnerable to these weaknesses. We also show what is done to protect projects.
- **Commit Signing in Critical OSS Projects.** Although we see a small trend towards more signed commits, about 50% of commits are not protected by a signature in 2023. However, there are significant differences in the programming languages and environments. GitHub's effort to increase adoption also seems to affect the commit signing rate positively.
- **Analyzing Authorship Metadata Spoofing in Advice for OSS Developers.** We show that online security advice for OSS developers addresses the risk of contributor spoofing. Still, no source mentions the problem of unattributed emails in GitHub repositories (*Contribution Hijacking*). Almost all sources recommend signed commits as a solution, but some also mention problems with commit signing.
- **Consequences for the OSS Software Supply Chain.** We emphasize that commit signing can play an essential role in the transparency and traceability of open-source software components. That said, only a few projects and contributors achieve a level of coverage with signatures that allow the signed Git history to be considered a trustworthy source for authenticity.
- **Changes to Git and GitHub to Reduce the Attack Surface of Authorship Metadata Spoofing.** We provide recommendations for OSS developers, projects, GitHub and Git to reduce the attack surface and improve the usability and effect of countermeasures.

Responsible Disclosure: We disclosed our findings to GitHub. We report the process timeline and details in Appendix B.

II. Related Work

We introduce related work on SSC security, security issues utilizing Git, and work on adopting cryptographic signatures.

A. Software Supply Chain Security

The SSC is vulnerable from various angles, and its protection must include more than one measure [13], [25]–[27]. Enck and Williams mention updating vulnerable dependencies, leveraging the concept of *Software Bill of Materials (SBOM)* for security purposes, choosing trusted supply chain dependencies, securing the build process, and getting industry-wide participation as core defensive measures, but those measures pose significant challenges for organizations [27]. SBOM is becoming increasingly widespread as an instrument for validating software and its dependencies [28]. However, not all components, such as Git commits [29], can be mapped in SBOM, and there are high error rates when generating reports with different tools [30]. Lamb and Zacchiroli propose *reproducible builds* of software packages [31], which in the context of SSC has been highlighted by Fourné *et al.* as necessary [32] to solve [33] the *Trusting Trust* attack [10]. Torres-Arias *et al.* introduced the *in-toto* framework, that uses chained signatures from the Git commit throughout the build and packaging process to the package manager [34]. Both, *reproducible builds* and *in-toto*, aim to prevent tampering with third parties' SSC. Okafor *et al.* shed light on the costs of protective measures. They propose three security dimensions for software projects involved in the SSC: *Transparency*, *Validity*, and *Separation* [25]. Different security properties are applied to artifacts, operations, and actors, and various security mechanisms are considered. Git's *commit signing* is named as a protection mechanism for actors at the *Transparency* level and for artifacts at the *Validity* level [25]. Ohm *et al.* and Ladisa *et al.* analyzed previous attacks on the open-source SSC and the vectors exploited in these incidents [13], [26]. Forged packages or compromised accounts were significant vectors in the SSC. Ladisa *et al.* found that most developers oppose using signatures for utility and cost reasons [13]. When selecting trustworthy packages, Zimmermann *et al.* noted that maintaining packages is becoming increasingly difficult. For NPM, an increasing number of unmaintained OSS packages exist, and security patches are released less and less frequently [35]. Expired domains are another problem. If they belong to email addresses of contributors. they can be taken over [36]. The security of OSS is directly connected to SSC issues, as its components are used by many other software projects [11], [13]. In contrast to proprietary software, the often decentralized and open contribution possibilities of the OSS community represent a different development environment [22], [37], [38]. In interviews, Wermke *et al.* found that open-source projects are very heterogeneous in their setup and processes. The security levels of projects also vary, and the decentralized and impersonal structure makes trust an essential factor [9], [23], [39].

> **Our Contributions.** We extend current research with the perspective of how Git's metadata can be abused to manipulate representations on GitHub and to exploit trust structures within OSS projects.

B. Git-related Security Issues

There has been some research on security issues of Git and how to tackle them. Using manipulated metadata, Torres-Arias *et al.* showed an attack on how to get malicious states into a Git history [40]. They propose a cryptographically signed log of developer actions to detect irregularities [40]. Alarcon *et al.* also propose using commit signing as a countermeasure [24]. To detect similar attacks, tools are proposed, such as *Anomalicious*, which uses commit logs and repository metadata to detect potentially malicious commits, achieving a 53.33% detection rate on malware-infected repositories while maintaining a low false-positive rate [41]. Afzali *et al.* propose `le-git-imate` to extend the audit trail towards social code platforms [42].

> **Our Contributions.** Previous academic research emphasized commit signing to deter malicious contributions and increase auditability. Our analysis of security advice explores recommendations and countermeasures from the developer community on this topic.

C. Research on PGP and Digital Signatures

Git, GitHub, and GitLab allow commit signing with PGP. PGP has been studied over the last decades but primarily for end-to-end encrypted (E2EE) emails. Whitten and Tygar studied the usability of an early PGP version [43]. They found several usability issues, e. g., participants sending clear text secrets instead of encrypted email or not publishing the public key. Various researchers proposed user interfaces and user workflows to improve the usability of E2EE emails [44]–[51]. Atwater *et al.* suggested users prefer encryption tools with tight integration into their system [49]. Others proposed user interfaces for E2EE messengers. Most of them focus on (public key) encryption [52]–[56]. Only Garfinkel *et al.* focus on digital signatures and the authenticity of emails. They showed that minor UI improvements can increase security [57], [58]. Gutmann proposed key continuity management as a trust model for public keys where users trust the first public key for others' identities [59]. It does not require a Public Key Infrastructure (PKI) and is also known as *trust on first use* [60], [61]. Garfinkel and Miller argue that users can hardly decide to trust a key for an unknown email address [62]. Other PKIs suffer from the same problem. PGP's original trust model is the Web of Trust (WoT) [63]. In this model, users sign others' public keys if they trust them. Users can upload their signed public keys to keyservers. A user can trust an unknown public key based on the signatures and the trust chain. Email communication is an everyday use case, and various researchers have studied this model [64]–[66]. Ulrich *et al.* studied published PGP keys from 2009 [64]. However, of about 2.7 million public keys, only about 325,000 were signed by other keys and, hence, part of the WoT. Lerner *et al.* used another trust model for their email encryption prototype [67]. They followed the *KeyBase* approach, linking public keys to social media or other online accounts [68]. Users can trust public keys based on these accounts. Commit signing with PGP on platforms like GitHub or GitLab follows the same approach and can be a trust model for developers as a subgroup of PGP users. Lerner *et al.* showed that automatic public key management via KeyBase mitigates critical user errors, e. g., forgetting to import or distribute public keys. In 2022, Stransky *et al.* studied E2EE emails [20]. Their analysis indicates that E2EE emails are rare, but digitally signed emails were slightly more common. Other researchers showed that specific user groups, e. g., activists or investigative journalists, use PGP for email encryption [69], [70]. For software development, Schorlemmer *et al.* investigated the prevalence of signatures for package/software signing. They highlight the role of package managers who must actively mandate signatures and emphasize the role of helpful tooling [71]. To maintain anonymity and guarantee zero-knowledge properties, Merrill *et al.* propose *Speranza* to bridge the gap between usable developer-managed keys and certificate-based systems [72].

> **Our Contributions.** Previous research has mainly focused on PGP for encrypting emails, and digital signatures are primarily examined in the same context or when signing software packages. Our study examines PGP and digital signatures when used for Git commits and explores their use with a tech-savvy population of OSS contributors under real-world conditions.

III. The Flow of Git's Metadata

Version Control System (VCS) are the infrastructure backbone of the OSS ecosystem. Developers can collaborate, track changes, try out ideas via branching, and revert unsuccessful changes. 96.65% of professional developers on StackOverflow use Git [73]. It is the *de facto* standard VCS for OSS projects, and through features like pull requests and forks, it indirectly defines contributor roles and interactions [73], [74]. Below, we illustrate Git's workflows, show how metadata represents contributor interactions, and describe how GitHub uses this metadata.

A. Local Generation of Git Contributor Metadata

Linus Torvald developed Git focusing on a simple design, efficiency, and scalability [75]. Users locally configure their Git environment with their name and email address, and they can set arbitrary names and email addresses. Git uses this information to add authorship information to commits [75]. Git defines three major roles for submitting source code to a remote repository [75]. They are depicted in Figure 1.

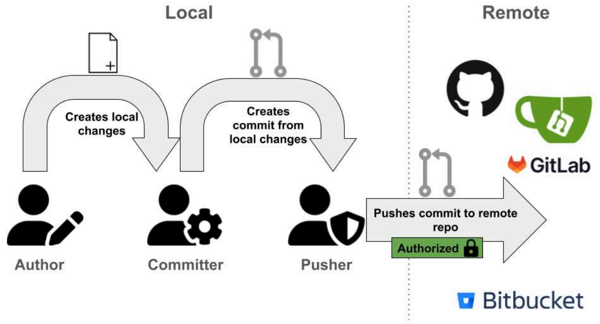


Fig. 1. The different role interactions for a contribution with Git.

Author: The author is the developer who modifies, adds, or deletes files using `git add` and `git rm` in a Git repository. Git then generates a patch file that contains all current changes in the repository. This role is represented by the `author` field of a commit, and it contains the name and email address from the local Git environment [75].

Committer: The committer stages and bundles a patch's changes into a single commit, which is then added to the local Git history. A commit contains the version history of the patch file linked to its parent commit [75]. While creating a commit, the committer can add a signature generated by signing a temporary content hash over the patch file, the commit message, the author, the committer, and the tree state. The committer for a commit can be found in the `committer` field [75]. Typically, the committer and author are the same user, but in some cases, they differ. For instance, code changes for the Linux Kernel are discussed and handled via email [76]. Only a few contributors can make commits; consequently, the author of the change and the committer can be different users.

Pusher: To make a commit appear upstream, it has to be pushed to a remote repository [75]. The pusher needs permissions for the remote repository, e. g., on GitHub. Before pushing commits from local to remote, the pusher must authenticate their local Git client for the upstream repository [77]. Git supports authentication via SSH [78] and HTTP [75].

B. Git's Commit Signing

To add authenticity and non-repudiation to Git commits, users can sign commits using digital signatures with a private key. Git supports the public key schemes PGP, SSH, and S/MIME [75]. For this process, a temporary file is created that contains the patch file, the corresponding Merkle tree status and the contributors. A local signature agent then signs a hash of the file. A third party can verify a signed commit using a given public key. The remote server, project maintainer, or other contributors can link signed commits to a private key owner and, after a key verification, to a certain user.

C. Fueling GitHub's Social Network with Git's Metadata

Since GitHub is the most widely used social coding platform, it is a particularly critical component in the OSS ecosystem. GitHub user profiles are often used as a source for assessing a developer's skills and reputation [79]–[81] and contribute to establishing trust between stakeholders in the OSS community [23], [24]. Hence, GitHub user profiles are often used in technical recruitment processes, are part of job applications [82], [83], and are used by OSS maintainers to assess the trustworthiness of contributors [9], [24]. Since Git's authorship metadata are locally configured, they are untrustworthy. However, GitHub uses this metadata to fuel their social network. GitHub uses the `author` field of a Git commit as the source for attributing a commit's authorship [14]. It creates a clickable hyperlink to user profiles for all email addresses in Git commits registered for GitHub accounts—for both primary and secondary account email addresses [84]. Thus, the contributions displayed for user accounts and project pages on GitHub are generated using untrustworthy email addresses. GitHub also supports the custom Git tag `Signed-off-by: NAME <EMAIL>`, which can be added to the commit message. This indicates that a commit was not made by a single author alone but by other contributors as well. This happens, for example, with squash merges, which merge changes from multiple commits into a single commit. GitHub adds all authors of the squashed commits as co-authors to the new commit. In GitHub's UI, they are displayed as co-authors. While this practice helps GitHub improve its social network features, it allows malicious users to spoof commit authorship and hijack contributions and reputation (see Section IV below).

D. GitHub's Perspective: From Untrustworthy Metadata to Trust Signals

GitHub acknowledges the risk of using untrustworthy commit authorship metadata [17], [21]. They argue that the attribution of commits is not a severe security issue, and changing their attribution system would result in an incorrect display of commit histories: *"It would be incorrect to assign* *all of the commits to the person doing the push, so we use the commit log email addresses to assign attribution on GitHub.com*” [17]. However, GitHub also states:

> *“[. . .][We] consider impersonation an account abuse issue that we take very seriously. If someone is wrongfully impersonating you, please let us know, and we will investigate the matter and deal with it as quickly as possible*”
> — GitHubSecurity [17]

As a countermeasure, GitHub recommends Git’s built-in commit signing feature (*“check out the* `git commit -S` *command* ”[16]). However, the proof that the committer with a private key is the account owner is still missing, and uploading a public key to the account does not imply that the account owner knows the corresponding private key.

GitHub added the so-called *Vigilant Mode* for users. If users activate it, an unsigned commit or a commit with an unfamiliar signature in which they occur will be marked as *Unverified* on GitHub. If the committer signs a commit, but the author differs from the committer and has the *Vigilant Mode* activated, GitHub displays a *Partly Verified* badge. Only if the committer and author are the same individual of a verifiable signed commit, GitHub displays the *Verified* badge [85].

GitHub discusses the lack of email address verification for commit attribution [17]. This seems to be an intentional design decision, and it is recommended that affected users resolve related issues via GitHub’s customer support:

> *“Any email address that is not already associated with an account on GitHub may be claimed, and this will give commit attribution to the claiming user. While we allow this attribution without requiring email address verification, any disputes around emails on accounts can be resolved by contacting our support team.*” — GitHub Security [17]

In summary, GitHub acknowledges the security drawbacks of its lax commit attribution strategy but prioritizes its social networking features over in-depth defense of the OSS ecosystem: “*We consider a handful of reports ineligible, either because the feature is working as intended or we accept the low risk as a security/usability tradeoff*” [17].

IV. Abusing GitHub’s Representation of Git

The author, committer, co-authors, and other contributor attributes are unauthenticated metadata in Git commit objects. Before adding a commit to the tree, the pusher of a commit can modify the metadata freely and without restrictions, making it untrustworthy information. However, GitHub uses this untrustworthy information to fuel their social network components and link GitHub user profiles to commits and projects. Consequently, malicious pushers can impersonate legitimate contributors and hijack project contributions in multiple ways. Below, we illustrate how the use of Git’s untrustworthy metadata as trust signals by platforms like GitHub can be exploited for impersonation attacks on OSS packages. We analyze the prevalence of these weaknesses in critical OSS projects and make a risk assessment.

A. Selecting Critical OSS Projects

We aimed to select critical OSS projects that are part of the SSC and hosted on GitHub. We collected metadata and content properties to investigate the prevalence of our impersonation attacks and how projects handle authenticity. Figure 2 illustrates our selection process and the selected data.

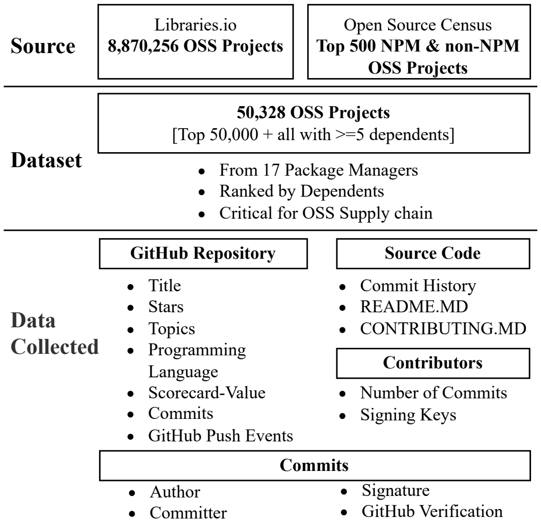


Fig. 2. Overview of projects and data collected.

Selected Projects: We aimed to analyze projects that play a critical role in the SSC. Therefore, like previous work [86], [87], we use the count of dependent packages as an indicator for the criticality of a project. The result is a ranked set sampling (RSS) approach that uses the number of dependents as the ranking characteristic. RSS typically provides a fitting representation of a population [88]. We based the OSS project selection on the Linux Foundation’s Open Source Census 2022 [86] and the Libraries.io [89] dataset. These sources, in combination with our criticality indicator from the RSS, allow us to narrow down our selection to software projects that are part of the SSC. As an additional benefit of our sampling approach, repositories that are not part of the SSC, e. g., awesome lists or tutorials, are not considered by design. Like Nagle *et al.*, we included critical OSS projects from 17 OSS package managers, including NPM, PyPi, Maven, Homebrew, Cargo, and CRAN. We aimed to analyze the 50,000 most critical OSS projects based on their number of dependents. We cleaned our dataset as follows: First, we removed duplicate entries from the Libraries.io dataset that pointed to the same repository. Second, we excluded forks and projects whose commit histories were subsets of other projects in our sample. We decided against repositories’ archival status, age, or activity as further cleaning criteria because their commits can still be exploited for contribution hijacking. Ultimately, our sample encompassed 50,328 GitHub projects with at least five dependents.[1] Our replication package contains a list of all projects (cf. Appendix A).

Collected Data: We collected the *title, topics, stars, primary programming language, dependents, OpenSSF Scorecard values*, the *source code*, the *history of all commits*, and the output of the GitHub Events-API for all repositories.

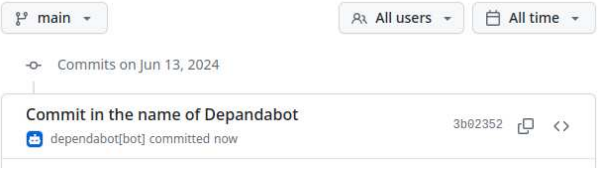
For each commit, we collected the *author’s and committer’s*

[1] Since more than 50,000 projects had five dependents or more, we included the surplus of 328 more packages.

GitHub account ID, name, and *email address*. If available, we fetched commit signatures, including GitHub's verification results. We also downloaded the SSH and PGP signing keys for all GitHub accounts that authored commits in our dataset. We collected their public key IDs from the Ubuntu and OpenPGP keyservers wherever possible. Our replication package describes all data sources (cf. Appendix A).

B. Tampering with Commit Metadata

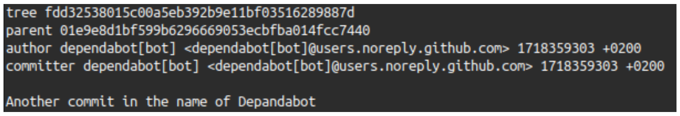
When pushing a commit to a remote repository with respective push permissions, a malicious pusher can freely set the `author` and `committer` fields of a commit to the email address of an arbitrary GitHub account. While arbitrary email addresses can be used, addresses of popular, active or longstanding OSS contributors and maintainers are particularly attractive. After pushing such a malicious commit with modified metadata to a repository, GitHub tries to match the email address to a registered GitHub user account and lists this account as the committer, author, or co-author in the respective project.



Commits on Jun 13, 2024

Commit in the name of Depandabot
dependabot[bot] committed now
3b02352

Fig. 3. A spoofed commit pushed to a GitHub repository by one of the authors in the name of GitHub's dependabot.



```
tree fdd32538015c00a5eb392b9e11bf03516289887d
parent 01e9e8d1bf599b6296669053ecbfba014fcc7440
author dependabot[bot] <dependabot[bot]@users.noreply.github.com> 1718359303 +0200
committer dependabot[bot] <dependabot[bot]@users.noreply.github.com> 1718359303 +0200

Another commit in the name of Depandabot
```

Fig. 4. Output of running `git cat-file [HASH]` for the spoofed commit.

Figure 3 illustrates a spoofed commit made in the name of GitHub's *dependabot*. In this case, one of the researchers pushed this commit with the modified email address of the dependabot account. GitHub also does not notify users if another GitHub user pushes commits with their email addresses. Figure 4 shows that detecting a spoofed commit in the Git history is challenging. The GitHub UI displays this commit just like other legit (dependabot) commits. Hence, assessing the authenticity and trustworthiness of commits' authorship information is hard for GitHub users when there is no signature. This behavior results in two abuse scenarios:

1) **Contributor Spoofing:** Malicious actors can disguise themselves to sneak commits containing malicious code into third-party repositories. The actor can create pull requests with commits pretending to originate from trusted contributors and leverage the effect of trust (e. g., in an attack with GitHub's dependabot account [15]).
2) **Reputation Hijacking:** Malicious pushers can add popular OSS contributors with GitHub accounts, e.g., Linus Torvalds, to their potentially malicious repositories. This can make repositories look more legitimate and affect signals from third parties like the OpenSSF Scorecards [8]. GitHub was affected by mass repository confusion attacks in February 2024, which also included new malicious commits in the name of former contributors [2].

TABLE I. COLLECTED DATA FOR PUSH EVENTS.

Description	**Mean**	**Std.**
Commits per Repository	57.9	110.3
Commits per Push Event	3.3	6.0
Commits per Pusher	65.8	1122.7

Detecting cases of these two attacks is not easy. However, assessing the disparity of pushers and authors in commits can help to estimate an upper bound for projects at risk. Contribution practices in which all commits have the same person in all roles are less at risk, as GitHub always authenticates the pusher. However, pushing commits in the name of others is a common practice for mirrored projects and projects using bots [90], [91]. The commit authors are part of our collected metadata (cf. Figure 2). Still, we had to collect the associated push events using GitHub's Events-API to get the identity of the pusher. However, this API endpoint is restricted to 90 days and 300 events for a project [92], limiting our analysis.

Results: We collected 132,281 push events for 7,546 GitHub repositories between April and July 2024. These events were triggered by 6,649 GitHub accounts and contained 437,211 commits. Table I gives more insight into the commits' distribution among repositories, push events and pushers.

The share of commits pushed by the same account that GitHub also attributes to the committer and the author is 30.1%. This is the most benign case since all three roles point to the same account, which is authenticated by GitHub and authorized to push commits to a repository. However, this leaves 305,454 commits with different committers, authors, and pushers. While this commit behavior reflects typical project behavior, such as the code review process or utilization of bots [93], it can indicate abuse. 1,067 (14.1%) of the repositories in our dataset only contained commits with equality in all three roles, while 2,368 (31.4%) of the repositories only contained commits with inequality in the three roles. Most of the repositories (6,479; 85.9%) could be at risk, as they have a contributing workflow that allows inequality in these roles. We also found repositories for which all the commits in our dataset differ in all roles, such as JetBrains IntelliJ Community[2] and Drupal[3]. Though both share the same characteristic, the origin is different. For the JetBrains project, this stems from their use of the `intellij-monorepo-bot`, which shows up as both committer and pusher but not as the author. As indicated by GitHub, Drupal is a mirror that uses the `hubot` account for pushing code. These two examples illustrate legitimate scenarios where pusher, committer, and author differ.

When inspecting the data for pushers, we find that some accounts only push commits that list someone else as the committer or author. Some, like the aforementioned `intellij-monorepo-bot`, indicate themselves as bots via their name, and others carry the official `[bot]` suffix, like `dependabot[bot]`. However, others like the account `zeke` only use the email

[2] https://github.com/jetbrains/intellij-community

[3] https://github.com/drupal/drupal

`github-actions@users.noreply.github.com` as both committer and author addresses for all their commits. This address is not mapped to any GitHub account. One possible benign explanation is that these commits are part of an automated process. Privacy considerations might be another reason for using a different email address since these addresses appear in a public GitHub repository.

> **Contributor Spoofing is challenging to detect.**
>
> - 85.9% of projects have at least one push event where the pushing GitHub account is not the committer (or author) account.
> - 2,368 repositories only contain commits where author, contributor, and pusher are not identical. There are benign cases when this behavior is expected.

C. Contribution Hijacking

GitHub users can link multiple email addresses to their GitHub accounts: a primary address, backup addresses, and committing addresses [94]. GitHub uses all of these email addresses for commit attribution, i.e., GitHub links commits that include any of a user's email addresses to the user's GitHub profile. While users need to verify their primary address via email-based authentication, users can add any email address as a backup or committing address without verification [95], [96]. When users add email addresses to their accounts, GitHub sends the following email to the added email account: *"You're receiving this email because you recently created a new GitHub account or added a new one. If this wasn't you, please ignore this email."* This suggests that users do not have to act if someone else claims their address, and they are still secure. However, GitHub keeps the link between the unverified email address and the account that added the address. The account claims the unverified address; others cannot reclaim the same address, GitHub links the account to all commits that contain the address, and generates a link to the corresponding user profile. An attacker can exploit this by claiming email addresses mentioned in commits that are not linked to any GitHub account. Successful attackers are listed as contributors with all attributed commits on the project page on GitHub and can misuse this to increase their account's reputation, e. g., by being listed as a contributor in a popular open-source project. Attackers might also use longstanding commit histories in popular open-source projects to find unlinked commits and claim email addresses to add trustworthiness to their account, e. g., infiltrating other open-source projects by leveraging perceived reputation.

To estimate the potential effect of contribution hijacking, we analyzed how many commits are not linked to a GitHub account and what entries are set in the `author` and `committer` fields. We checked each entry for valid email addresses and unregistered domains that could be hijacked [36]. If an attacker takes over an abandoned domain, they can verify all email addresses under said domain. This allows attackers to add signing keys, in addition to taking over existing commits. Future commits by an attacker can thus also be signed, which will make commits appear more trustworthy.

Results: In all 25,889,629 commits, we found 578,897 unique email addresses. Of those, 5,854 had an invalid format and could not be parsed. We extracted 138,743 unique domain names from the 573,043 valid addresses. We used the Start of Authority (SOA) Domain Name System (DNS) resource records to analyze the domains [97]. A domain is registered when a valid SOA record exists in the DNS. The domain is considered available if it does not have a valid SOA entry. We found 4,107 unregistered domains. 13 addresses were linked to multiple GitHub accounts or had distinct commits that were reported as both linked and unlinked by GitHub's API during the crawling. Those email addresses could be transferred between accounts, or their commits were pushed before an account for the address was registered or after its deletion.

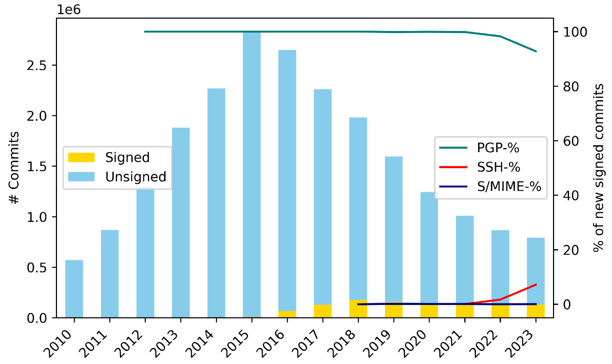


Fig. 5. The chart shows the total commits introduced per year between 2010 and 2024, separated into signed and unsigned commits. The line charts show the percentage of newly signed commits per signature type.

> **Contribution Hijacking is Feasible.**
>
> - We identified 3,013,817 unlinked commits of which 573,043 contain valid email addresses that can be claimed.
> - Based on these 573,043 email addresses, 4,107 domains can be hijacked by the attackers to build an OSS legend.

V. Commit Signing in Critical OSS Projects

In an initial regression analysis, we examined factors influencing signature behavior in our dataset. The age of a repository had a significant but small negative influence on the overall signature behavior. Still, it was outweighed by other characteristics such as the used programming languages (see regression table in Appendix D-A). Hence, because old commits are also vulnerable to impersonations, we examined the entire dataset from Section IV-A for signatures. Below, we illustrate and evaluate the usage of commit signing.

A. Commit Signatures and Signing Technologies

Overall, we found 4,063,376 (15.7%) signed commits among all 25,889,629 analyzed commits. Commits that are made via the GitHub UI set the *web-flow* account as the committer and the acting GitHub user. GitHub's web-flow account signed 2,961,704 commits; users signed 1,101,672 (5.0%) commits. Since we are interested in the users' signing practices, we exclude the web-flow commits from the following reports unless stated otherwise.

1,089,270 of the users' commit signatures used PGP, 11,864 SSH, and 538 S/MIME. Figure 5 illustrates the development of commit signing over time, separated by signing technologies. We added GitHub's web-flow account, which uses PGP for commit signatures, to compare it to user commits. GitHub began signing web-flow commits in 2017.

TABLE II. A COMPARISON OF SIGNATURE VERIFICATION RESULTS BETWEEN USERS AND GITHUB AND ACROSS SIGNATURE TECHNOLOGIES. GITHUB PROVIDES A DETAILED ERROR CODE DESCRIPTION [98].

Result	PGP		SSH		S/MIME		GitHub PGP	
`Valid`	890,171	82%	10,660	90%	2	0%	2,960,651	100%
`Unknown key`	104,760	10%	810	7%	–	–	1	0%
`No user`	53,973	5%	119	1%	157	29%	0	0%
`Bad email`	19,918	2%	–	–	0	0%	0	0%
`Unverified email`	19,690	2%	273	2%	0	0%	0	0%
`Invalid`	212	0%	2	0%	1	0%	941	0%
`GPG error`	384	0%	–	–	–	–	111	0%
`Bad certificate`†	–	–	–	–	363	67%	–	–
`No signing key`	162	0%	–	–	–	–	0	0%
`OCSP revoked`*	–	–	–	–	15	3%	–	–

We also found 5 instances of unknown signature types we could not process.
* Entry in the certificate chain was revoked. † Certificate could not be verified.

Figure 5 shows a peak of new commits in 2015, which means our dataset is skewed toward older repositories. This could result from our sampling process, which favors older projects due to the required number of dependents. While the number of new commits has declined since then, the number of signed commits remains steady, leading to a relative increase in signed commits. SSH signatures appeared before August 2022 when GitHub announced their support [99]. Before that, PGP signatures were dominant, with S/MIME never reaching any relevant adoption rate. The GitHub API reports the verification status of commit signatures [98]. 82% of all PGP and 90% of all SSH signed commits were valid. The most common error was a missing public key to verify the signature (10% of PGP commits and 7% of SSH commits). Table II illustrates these verification results across different technologies.

Apart from one commit with an unknown key, the 941 commits marked as invalid, and the 111 errors due to GPG verification, all web-flow commit signatures were verified. The GPG errors are transient and the result of a failure in GitHub's backend. The unknown key is the result of a commit where the user supplied the email address (*noreply@github.com*) used for the web-flow account as the committer email address but tried to sign the commit with a different key than the one used by GitHub. The invalid entries are inexplicable.

Most of the signatures provided by users are valid. The most common reason for failed verification is missing keys that were not uploaded to GitHub by the users (*Unknown key*). Other reasons for failure are associated with the committer's email address: Either it is not associated with any GitHub account (*No user*), it is not part of the PGP key that was used (*Bad email*), or the email address was added to the GitHub account but not verified (*Unverified email*). There also exist 5 commits in our dataset that did not contain valid signatures at all (*Unknown signature*).

B. Signing Keys

Of GitHub accounts that committed to one of the projects in our dataset, 48,123 (22.7%) users either uploaded a PGP or SSH key for signing commits; 41,733 (19.7%) uploaded PGP keys, 3,246 (1.5%) uploaded SSH keys, and 3,144 (1.5%) uploaded keys of both types. Apart from GitHub, roughly one-third of the PGP keys can be found on public keyservers, with 10.0% on *keys.openpgp.org*, 3.3% on *keyserver.ubuntu.com* and 19.4% on both.

Table V in the Appendix shows the different algorithms used for PGP and SSH signing keys. For PGP, RSA-4096 is the dominant algorithm (61% of PGP keys), followed by ED25519 (12%). For SSH, the popularity of the top two algorithms is reversed (ED25519: 73%, RSA-4096: 8%). Overall, more PGP keys were uploaded by users, which could be a result of the longer support for PGP by GitHub, with SSH only being recently adopted. There also exist 626 DSA-1024 keys, whose algorithm was already considered insecure by the time GitHub announced its *Verified* badge in 2016 [100]. An expiry date is set for 36.6% of PGP keys, with an average time span of 4.8 years between their creation and expiration; 20.3% were marked as expired. There is no expiration date for SSH keys.

C. Repositories

Our dataset contains 50,328 repositories. On average, a repository contains around 100 commits (see Figure 11). 72.1% of repositories do not have any signed commits. The prevalence of signed commits increases slightly when looking at repositories that leverage commit signing. The repositories with signed commits have an average signing frequency of around 5%.

The GitHub API reports 105 different primary programming languages for these repositories. With almost 30,000 repositories, JavaScript is the most used language. Rust and TypeScript have the highest percentage of signed commits at over 8%, while Java shows less commit signing at under 4%. Figure 8 in the Appendix shows the distribution of signed commits across these languages and their overall share of our dataset. We also found that the signing frequency diminishes if a project has more committers or commits (Figure 9). This could be because in a project with many committers, most do not sign their commits unless mandated by policy. The OpenSSF Scorecard [8] is a value that aims to represent whether repositories follow security best practices. We found that if the contributors to a repository sign their commits, the repository has a Scorecard value above 1 (Figure 10). The Scorecard documentation does not mention commit signing as a measure considered when calculating the score. Therefore, this observation could result from an omission in the documentation. Figure 10 also shows the commit signing frequency concerning dependents, as calculated by the GitHub API and measured by the number of dependent repositories and packages. The dependencies show a similar influence as the number of commits and contributors, primarily dependent packages. Repositories with many dependents are probably more popular, inviting more contributors.

D. Committers

The commits in our dataset show 348,662 distinct committer email addresses. The average committer contributed less than ten signed commits to the repositories in our dataset (see Figure 12). 95.4% of all committers do not have any signed commit. Those who sign their commits show an increase in contributed commits over time.

71.0% of the committer's email addresses can be linked to a GitHub account. Their signing behavior differs when comparing contributors linked to a GitHub account and those without a corresponding account. GitHub account users issued 19,087,501 commits, with 1,047,475 (5.5%) of the commits

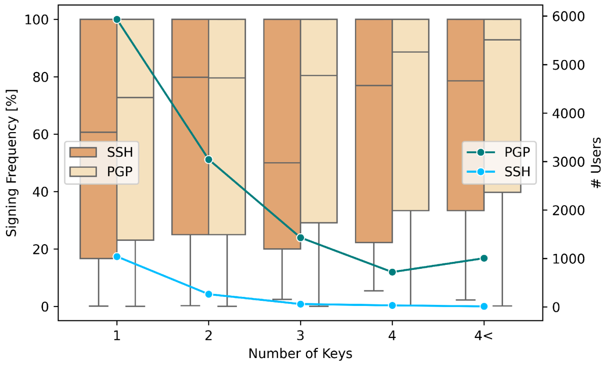


Fig. 6. The boxplots display the number of GitHub account users and the keys they uploaded, while the line chart illustrates the signing behavior per the number of keys uploaded.

being signed. On the other hand, non-GitHub committers are responsible for 3,013,817 commits, 54,202 (1.8%) of them signed. Users can upload multiple keys to their PGP and SSH accounts. This is the case for 48,123 (22.7%) of accounts in our dataset, of which 41,733 (19.7%) uploaded PGP keys, 3,246 (1.5%) uploaded SSH keys, and 3,144 (1.5%) uploaded both. In 25 cases, keys are shared between GitHub accounts.

Figure 6 illustrates the number of keys uploaded by users and how that influences the users' signing behavior. Most users uploaded one or two keys to their accounts. Adding a second key shows an increase in commit signing frequency. Accounts with three SSH keys show a decrease in their signing frequency. This could result from a statistical outlier due to our dataset's number of samples, especially because the signing frequency recovers and remains at the same value with more keys. PGP signing frequency remains steady after two keys and even increases toward higher key counts.

> **Low Prevalence and Challenging to Verify.** We find that only 1,101,672 (5.0%) of commits are signed. Of those 1,089,270 (98.9%) use PGP keys for their signatures, but only 11,864 (1.1%) use SSH. Only 0.2% of the commits are signed on the repository level; beyond that, we could not identify signing policies. Moreover, verifying signatures (e.g., keys not available on GitHub and keyservers) is challenging.

VI. Analyzing Online Security Advice on GitHub Impersonations for OSS Developers

After demonstrating the vulnerability of Git metadata to impersonations and the impact on critical OSS projects, we analyze security advice for developers related to the threats presented above. This analysis helps to reveal which of the attacks presented in Section IV are widely recognized and discussed by the community, along with the recommended protective measures to mitigate them. We were particularly interested in the similarities between contribution hijacking and previous dependency confusion attacks that resulted in mitigation strategies. Additionally, looking at the results of the 2020 FOSS Contributor Survey [12], we found a discrepancy between the measures taken by developers and the recommendations from large organizations, including GitHub, to use commit signing as a defense against impersonation [16]–[19]. Analyzing security advice can reveal whether a similar discrepancy exists.

A. Method

We aligned our analysis with the work on conducting a Gray Literature Analysis (GLA) in the software engineering community [101]–[104]. The following describes the practical ramifications of sampling, categorization, and analysis of security advice.

1) Sampling: We elaborate on identifying relevant sources of advice and outline our sampling procedure.

Search Terms: We first determined search terms that return potential security advice for our analysis. We used the terms *commit spoofing*, *github contributor spoofing*, *author impersonation git*, and *commit signing* as a starting point. We followed the approach of Wang *et al.*, utilizing ChatGPT for creating additional and related terms for achieving a broader variance of search terms [105]. Our final list contains **16 search terms** and can be found in Appendix C.

Search Process: We used Google and Bing to search for security advice using the search terms (Appendix C), as search engines are a primary resource for developers seeking solutions to security issues [106]. We used a clean, non-personalized browser setup for each search to eliminate distortions due to previous personalized search histories. Combining search terms and search engines, we used **32 individual search queries**.

As a **stopping rule**, we used a combination of *Theoretical Saturation* and *Effort Bounding* [101]. For each search term, we always evaluated the first ten search results (effort bounding). We continued our evaluation until three consecutive search results were already part of our security advice corpus or out of scope (theoretical saturation).

Selection Criteria: According to Garousi *et al.*, different types of sources are suitable for analyzing gray literature, including online sources for security advice. For our sampling, we primarily focused on blogs, technical documentation, opinion pieces, tutorials, and official recommendations. We added or excluded search results based on several criteria.

As **inclusion criteria**, we used the reliability metric and the evaluation of the authorship proposed by Garousi *et al.*, considering aspects like the outlet type, the author, and possible back linking [101]. This serves to better assess the validity and potential bias of sources that are typically not peer-reviewed. For example, documents from sources like 'Company Blogs' sometimes pitch their products or services as reasonable solutions to impersonation attacks.

As **exclusion criteria**, we primarily used the length of a document and our RQs. Hence, we excluded short documents that made only one or two statements or that only peripherally addressed our RQs. Since our study is mainly concerned with GitHub impersonations, contributor spoofing, and possible countermeasures, we focused on documents that addressed these aspects. The initial dataset was created by one researcher and cross-checked by a second researcher against the criteria. We excluded nine advice documents that did not fit the topic or criteria in that step. We also checked all advice documents for AI-generated content using the tool GPTzero [107] to

TABLE III. Overview of analyzed documents based on outlet type with associated IDs of sources, distribution, and category description.

Outlet Type *[IDs]*	$\sum$	**Description**
CVE Report *[D1]*	1	Mitre CVE Report
Company Blog *[D2–D14]*	13	Blogs frequently hosted on company websites serve as platforms where employees share their expertise through articles
Institutional Report *[D15, D16]*	2	Reports issued by governmental and private institutions
Learning Platform *[D17, D18]*	2	Contributions to websites that are digital hubs for education
Online Magazine *[D19–D34]*	16	Private Publishers of (non-peer-reviewed) articles
Personal Blog or Website *[D35–D51]*	17	Personal blogs authored by (often) experienced developers, exploring topics they are knowledgeable or passionate about
Technical Documentation *[D52–D55]*	4	Websites tied to specific applications, providing technical assistance within these applications

exclude security advice written by generative AI. As GPTzero considered all texts as "mostly human written," we excluded no further documents. We provide our selection criteria, all examined documents, and the AI reports in our replication package (cf. Appendix A).

2) Analysis: We applied qualitative coding to analyze all security advice documents in our sample, creating themes and patterns to structure their content. We double-coded all documents using a semi-open approach: Two co-authors coded all documents independently, using ATLAS.ti. One author started by coding the documents, using a deductive approach and predefined codes that emerged from our RQs. Thus, we searched for segments in the documents resembling those codes, including: *Contributor Spoofing*, *Reputation Hijacking*, *Contribution Hijacking*, *Motivations*, *Countermeasures*, and *Obstacles*. In the second phase, we examined the documents for new codes, using thematic analysis to enhance the initial codebook [108]. After the initial coding, the second researcher also coded all documents. We resolved all emerging code conflicts through discussion. As our methodology is purely qualitative, we follow previous work and omit the reporting of inter-rater reliability (IRR) [109]–[111].

B. Results

We analyzed 55 documents that give security advice on contributor impersonations and potential countermeasures. Table III gives an overview of the document types and the associated source identifiers used. The complete codebook is provided in the replication package (cf. Appendix A). For this section, given codes are reported in **bold**, with the number of documents in which the code occurs at least once in brackets.

1) Impersonation Attacks: When looking at potential exploits or attacks via Git, we found that some documents mention the potential to **fabricate metadata (12)**, e.g., timestamps, *"to make a repository appear older than it is"* (D26). Some sources also mention GitHub's behavior to **link user profiles to commits (9)** based on the commit header, defined by the local Git configuration when the commit is created. **Contributor spoofing (30)** is the most named impersonation attack in our documents. It is often stated in combination with the example that *"someone is able to push a change with malicious content to a repository under a name of a regular contributor"* (D9). In contrast to contributor spoofing, a **reputation hijacking (10)** attack tries to make the attackers' repository look more trustworthy: *"To make their project look reliable, attackers can use this technique once or multiple times and populate their repository's contributors section with known reliable contributors"* (D25). Interestingly, no document identifies the possibility of **contribution hijacking (0)**.

Numerous documents emphasize the **importance of trust (17)** as a higher-level security concern in the context of impersonation attacks: *"It is important that the sources used are trustworthy. This also applies to source code"* (D40). The question of trust is crucial, considering the **risks of unverified code (6)**: *"maintainers and developers should only trust those contributors that are known to them and have an extensive and verifiable commit history"* (D29).

2) Commit Signing as Primary Countermeasure: Commit signing is the most considered countermeasure against impersonation attacks in Git-based software development. This led us to analyze how commit signing is described in our documents.

Due to the tutorial-centric nature of many documents in our sample, the bulk covers **GPG (28)** as a signing method, as opposed to **SSH (14)**. Some documents state that **SSH familiarity simplifies setup (4)** of commit signing. According to them, the widespread ownership and utilization of SSH keys among developers is an advantage over GPG, highlighting a potential easier usage in commit signing. Only a few documents mention **S/MIME (5)**, and only one (D13) provides practical guidance on its utilization. A common recommendation for commit signing is to use the **autosign (19)** function, which configures the local Git agent to sign all commits automatically.

Linked to the importance of trust is the question of how to verify trust. One approach is the PGP-based Web of Trust. Interestingly, while referenced several times, the **Web of Trust (9)** is often glossed over, sometimes even dismissed, without detailed explanation: *"No one uses PGP's web-of-trust, so anyone checking the signatures has to rely on the public keys uploaded to the GitHub account [...]"* (D23). Moreover, only two documents mention uploading keys to public keyservers. Consequently, the majority of documents tend to rely on GitHub as a *de facto* **centralized trust entity (11)** for storing, managing, and verifying signature keys, by linking them to GitHub profiles: *"At this point GitHub becomes one massive* `allowed_signers` *file, arguably with much better security and trust than the one on your computer"* (D48). The **vigilant mode (11)** is named as an essential anti-spoofing tool for GitHub users. Eleven documents recommend the vigilant mode to *"[make] it more accessible for potential users to spot impersonation attempts"* (D27). Eight documents recommend to **enforce signing for contributions (8)** using GitHub' branch protection rule to enforce signed commits.

3) Contributions to Software Supply Chain Security: The security of the **Software Supply Chain (16)** is a main motivation in our documents: *"As a core committer, you have the ability to affect over a million websites"* (D52). For many of the analyzed documents, the **proof of authenticity (16)** is the central contribution that commit signing provides. *"The receiver of the data can verify that the signature is authentic,* *and therefore must've come from the signatory"* (D17). Given its role as the main countermeasure against contributor spoofing and reputation hijacking attacks, it is unsurprising that authenticity receives considerable attention. Furthermore, it is often closely linked to **proof of authorship (10)**, despite emphasizing the **difference between author and committer (6)**. *"when you see a verified commit, the author has nothing to do with the verified status"* (D17). Eleven documents mention the positive impact of commit signing on the trustworthiness of commits and projects. This link between commit signing and trustworthiness is based on the perception that commit signing provides **proof of project/commit integrity (8)**. Related to authenticity, some documents consider a history of signed commits a good way to ensure **accountability and non-repudiation (7)**. For twelve documents, this seems important since *"signed commits could act as a reliable source for an audit trail"* (D6).

4) Alternative Countermeasures: Since account-based social coding platforms, like GitHub, play an essential role in the OSS ecosystem, one countermeasure against impersonation attacks is account-bound accessibility rights. Scott Chacon writes in a document: *"At GitHub we were never really majorly concerned with spoofing because you can't push into a repository you don't have access to"* (D48). The biggest security risk in that case are **account/infrastructure compromises (4)**, which are countered by **GitHub/GitLab account hygiene (4)**. In this case, the mentioned trust is mostly centralized on GitHub.

Some documents discuss alternatives or extensions to current Git commit signing, such as Git push signing, Gitsign by Sigstore and OpenPubkey. D35 mentions there is no way to dual-sign a commit, representing a design aspect of Git itself. Further, three documents commented on the fact that there is no notification when someone else refers to one's GitHub identity. One of our more comprehensive documents (D50) specifies three expectations towards a good commit signing solution: *"[1] short-lived cryptographic identities [...][2] clear revocation procedures (backed by trusted timestamps) [...][3] straightforward local verification (no web UI badges)"* (D50).

> **Security Advice on Contributor Impersonation.** While *Contribution Hijacking* is not discussed, *Contributor Spoofing* and *Reputation Hijacking* are known attacks and considered to be best counter-measured by commit signing. Moreover, a signed commit history is believed to offer authenticity and a reliable audit trail, thereby enhancing SSC security.

VII. Ethics

We were committed to upholding high ethical standards and protecting individuals' rights and privacy. Thus, we aligned our studies with the Menlo Report for research in the field of computer science [112]. We responsibly disclosed our findings to GitHub (still ongoing, see Appendix B) and conducted no active attacks on projects. We only tested our exploits in our setups according to the GitHub BugBounty program [17]. Our measurement of the threat to OSS projects was also only carried out passively so that no projects were at risk. All data we used was taken from publicly available sources, including commit histories in OSS projects and cryptographic keys that must be publicly accessible as part of the WoT. For datasets that allow the identification of individuals, e. g., by an email address linked to cryptographic keys, we only report information in aggregated form to protect the privacy of these individuals.

VIII. Limitations

For our repository analysis, evaluating all commits of all repositories on GitHub or even beyond GitHub is infeasible. Without a reference set we could use, we had to sample our own, which automatically imposes restrictions on our dataset. Our initial method of sampling projects required them to be marked as the dependency of multiple other projects. Since that can only be the case for projects that have seen some adoption in the past, they must have been around for some time. This leads to our dataset being more influenced by older projects, showing commit signing behavior in the past. A project's age also influences the ratio between signed and unsigned commits because unsigned commits in the past will remain unsigned and, therefore, be an obstacle to a fully signed commit history. Commit metadata, such as the commit time and email address, are self-reported by users. While working with these, e.g., to establish trends over time or to distinguish users, we have to assume that most users set them in good faith without modifying them with an ulterior motive. The analysis of the SOA records can also only be considered a lower bound since resellers have presumably registered further claimable domains and thus have a SOA entry. Furthermore, forks of projects with a high number of signed or unsigned commits in their shared history can bias the previously shown results in either direction, leading to over- or under-representation of the respective characteristic. Ultimately, our approach also depends on the quality of the data obtained from Libraries.IO. We excluded projects because their Git repository could no longer be tracked. As a result, some projects may not be part of our sample.

While the qualitative analysis of online sources is valuable for synthesizing diverse perspectives and insights beyond traditional academic sources, it is subject to inherent limitations [101]. These limitations stem primarily from the sources' quality, depth, and representativeness variability. As a result, interpretations derived from online sources may be subject to bias, lack generalizability, and require cautious interpretation within the study's objectives and methodological framework. We address this through our approach. In line with Garousi *et al.* and Giustini, we chose a general methodology applicable to the SE community at large [101], [102]. We used common search terms and selected and categorized articles for different stakeholders. Therefore, we believe that our sources represent advice developers find and consult for their work. Another critical factor is that our collection of online sources only provides the current state of online sources. To make our analysis understandable and comparable to future studies, we provide PDF versions of all the sources examined in our replication package in Section A.

IX. Discussion

Below, we answer our research questions and discuss our findings. We also provide a blueprint for project categorizations and describe future work.

A. The Research Questions

We start by presenting the implications of our results for each research question.

[RQ1] Trust Signals from Untrustworthy Origins: Previous research on the security of the SSC [13], [25]–[27], recommendations from large open-source organizations [18], [19], and the provenance requirements from CISA [7] indicate a need for verifiable authenticity and integrity of software products and their development. In this context, trust becomes a relevant aspect. Previous research has shown that trust is often the basis for decisions in code reviews [23], for accepting changes [24], or even for assigning project roles and access rights [9], e.g., as happened in the XZ incident [5]. However, we showed that the link between Git and GitHub for the attribution of contributions is weak and can be abused. Our three attack scenarios (*Contributor Spoofing*, *Reputation Hijacking*, *Contribution Hijacking*) show that attackers can either spoof contributors or build up a fake legend for more sophisticated attacks leveraging signals used for trust on GitHub.

We point out that these interactions not only affect the UI of GitHub, but the resulting data is also used by other projects via the official GitHub API. For example, the OpenSSF Scorecards consider the affiliation of contributors to organizations as a positive sign, and projects like `djangopackages.org` list contributors and commit frequency as a health metric for a package. In 2015, GitHub posted their security statement about contributor spoofing, and they are aware that attackers can fool users [113]. We are doubtful whether the previous risk analysis is still correct under these circumstances, and we argue that GitHub should consider a reevaluation.

[RQ2] Threats and Countermeasures: Our measurement study of critical OSS projects indicates that many projects and their contributors are vulnerable to our attacks due to not signing and assigning their commits. For contributor spoofing and reputation hijacking, we show for 6,479 GitHub repositories that it is difficult to distinguish between legitimate push events and malicious ones, i.e., malicious contributions in the name of a project maintainer. GitHub, mirrors, and bots use different roles when creating and pushing commits, so the author, committer, and pusher are only identical for a minority of commits (Section IV-B). This depicts the threat model: Malicious actors can hide their contributions in these commits, and it is challenging to distinguish between evil and legitimate ones. We also identified 3,013,817 commits in critical OSS projects that are not attributed to any GitHub account and can, therefore, be hijacked. For 4,107 email domains, we could also show that malicious actors can reacquire the corresponding domains. Thus, a complete takeover of the respective contributors' current and historical identity on GitHub is possible. This also includes new signed commits because these email addresses can be successfully verified on GitHub when the malicious actor takes over and controls the domain and uploads a corresponding public key to their GitHub account.

Commit signing seems to be the most prominent protection against impersonation. However, the commit histories in our measurement study show a low prevalence of signed commits in the repositories. We could only find 15% signed commits in the Git histories, of which two-thirds came from the GitHub *web-flow* account. Only if a user signs all commits is it possible to identify commits that do not originate from them (i.e., a potential malicious impersonation) because those do not have a signature or an invalid signature. At the individual committer level, our analysis shows that while many of the security advice documents recommend the `autosign` function (19/55), only 2.0% of all committers in our dataset sign all commits. Our analysis further shows that only about one-third of PGP keys can be found outside GitHub, rendering local verification without GitHub unreliable. This observation also extends to SSH-based commit signing, where verification is only viable through keys uploaded to GitHub, as there is no public keyserver—resulting in a complete dependence on the platform.

[RQ3] Analyzing Security Advice: Surveying online security advice shows that many documents acknowledge the untrustworthiness of Git metadata but mostly warn people about contributor spoofing. A few sources also mention that malicious repositories can be filled with renowned open-source contributors to look more legitimate. Still, we have not seen any reference to the issue of old unattributed email addresses in commit histories on GitHub. Thus, *contribution hijacking* appears to be unknown so far. However, as shown in Section IV, many projects are vulnerable to this threat.

Commit signing is frequently mentioned as a countermeasure, and eight documents also recommend using branch protection rules for projects to enforce signed commits. This recommendation, however, is not reflected in our measurement results. The documents also emphasize the value of improving the authenticity of contributions in an OSS project to promote its integrity for the SSC, with twelve considering signed commits as a good source for a reliable audit trail. However, some also mention several problems with the current signing procedure, mainly criticizing the *"awful user experience"* (D42) of GPG and the issues with several external tools. The added support of SSH keys by GitHub resulted in many new articles on commit signatures, emphasizing their usefulness due to the widespread use of SSH. In summary, there is much demand for the authenticity and protection that commit signatures can offer. Still, existing methods are only used reluctantly because they encompass severe user challenges.

[RQ4] Improvements for Git & GitHub: We see several actions different stakeholders can undertake to improve the current situation and provide authenticity for the SSC. We argue that Git should add a signature field for the `author` to their commit objects. As both `committer` and `author` can be spoofed, providing authenticity for both roles via separate signatures would be beneficial. We suggest several improvements for GitHub dealing with Git metadata:

1) **Email Validation.** Our findings illustrate that relying on email addresses for functionalities such as commit attribution without user authentication of email submission and retrieval can be a security risk—especially if legit users cannot claim their email address once another (malicious) user has added it. Furthermore, the current text in the verification email is misleading because it advises users to ignore malicious claims. We urge GitHub to rethink the email verification process for improved security and not rely on unverified email addresses for authorship contribution anymore.
2) **Adoption of Novel Commit Signing Technologies.** Modern signature schemes such as Sigstore are currently not supported by GitHub; their signed commits appear as *Unverified*. Supporting these technologies could help their adoption and authenticity of signatures.

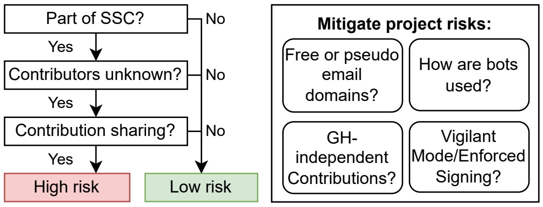


Fig. 7. A decision tree for project risk categorization and the proposed mitigation techniques.

3) **User Interface.** The current design of the GitHub UI may create misconceptions about the origin, authenticity, and security of the displayed information. Based on our findings, we propose the following UI changes to match the technical implications:
   a) **Distinguish GitHub and Developer Signatures:** We suggest adding a different display of commits signed by GitHub as the authenticity is based on various security principles, namely authentication versus public key cryptography.
   b) **Display Pusher:** Only the committer and author of a commit are shown in the UI. We suggest adding a notice about the pushing account to display all involved actors.
   c) **Limit Attributions:** The visual linking of untrustworthy email metadata to GitHub user profiles is a root cause of our impersonation scenarios. Limiting the visual attribution to commits without disparity and introducing the same interaction requirements as for the profiles' heatmap would streamline functionalities and ensure the display of only trustful information.
4) **Transparency Log for Signing Keys.** Our findings suggest that GitHub has become the sole PKI for commit verification in the OSS ecosystem. To fulfill this role, we suggest the introduction of a transparency log for signing keys, such as CONIKS [114] or one of its further refinements like SEEMless [115] or Parakeet [116], in which key material is publicly available. Ideally, this service would be independent of GitHub's API and UI, enabling independent and local signature verification.

B. *Recommendations for Open Source Projects*

Based on our results, we propose three questions for project risk assessment and four mitigation techniques for high-risk projects, as shown in Figure 7. We also discuss the importance of code reviews and how the issue of forged trust signals can be addressed.

Defining Project Scope: The three questions for risk assessment determine whether a project is in the scope of our attack scenarios.

1) **Part of SSC**: Projects that are not part of the open-source SSC are not of particular interest for either Contributor Spoofing or Contribution Hijacking, as only limited reputation can be gathered and malicious contributions are limited in their impact.
2) **Contributors Unknown:** The project structures are important. Suppose a project is only developed by a single person or a group of developers who know each other, and all the commits come from them. In that case, it is difficult for attackers to infiltrate these social structures.
3) **Contribution Sharing:** Contribution sharing is a practice that is a root cause for the weaknesses. If the pusher, committer, and author are identical in all commits in the project, the authenticated pusher guarantees authenticity, and no unassigned commits should exist.

Mitigating Project Risk: If a project is in the scope of our attack scenarios, we define four mitigation techniques that projects should evaluate.

1) **Free or Pseudo Emails:** Project maintainers should check whether the commit history contains free or pseudo email addresses and whether these are assigned to the correct GitHub accounts. If such email addresses exist, the affected contributors should be asked to add them to their GitHub account.
2) **How are Bots Used:** Project maintainers should evaluate how bots are used in the project. The attack with the fake `dependabot` showed that bots are used for spoofing, and our findings illustrate that bots are a dominant source for the disparity in commit data. GitHub supports the signing of bot commits to provide authenticity, and for projects working with bots, it should be checked whether this is configured [85].
3) **Vigilant Mode/Enforced Signing:** The signing behavior within the project should be investigated. Although signed commits are not perfect, signed commits with verifiable signatures are currently the best way to provide authenticity for a contribution. Hence, the introduction of contribution rules for signed commits should be considered, and contributors can use the *Vigilant Mode*.
4) **GitHub-independent Contributions:** Not all contributions are made on GitHub but are, for example, represented by mirrors, or the project was imported to GitHub at some point. This may result in many unassigned commits and also complicates signing. It should be evaluated whether there are legacy contributions in the project and whether they can be exploited, e. g., because not all contributors are linked on GitHub.

Importance of Code Reviews: While the profile of an OSS contributor might be considered an indicator of whether a contribution is trustworthy or secure, this is not the only indicator. In OSS projects, there is typically a review of the code of pull requests before the merge—often including running test suites, static (security) analysis tools (SAST), and alike using automated CI pipelines. While OSS contributions are not merged solely based on the contributor's identity, prior research found that trust in individuals and their identity benefits these decisions [9], [24]. This emphasizes the potential impact of the weaknesses we described and that GitHub's reliance on unauthenticated Git metadata might be dangerous—especially as it is likely that only a tiny fraction of Git and GitHub users are aware of the risks. An adversary might use this to create the illusion of reputation and increase the chances of getting pull requests merged that contain malicious code.

All in all, this makes a critical code review all the more important. Ideally, it would not matter *who* is the author or committer of some contribution to an OSS project. Only *what* is contributed, i.e., the source code should be checked to determine whether a contribution gets merged. The maintainers responsible for merging pull requests should especially be aware of the weaknesses we describe and focus solely on the contribution. We call this an ideal state, but in practice, this requires a lot of time from OSS contributors who are volunteers, and many projects already lack time and resources.

C. Future Work

An interview study with OSS developers on that topic would be interesting to understand the awareness of the problem better and also learn about practices and countermeasures that are not recognizable in our data. Furthermore, a more detailed examination of legend building in the OSS ecosystem could help estimate the overall criticality.

X. Conclusion

Our work illustrates three scenarios that can be used for impersonations using GitHubs' user and project profiles. While GitHub describes their handling of Git's metadata as not critical for security, we think this perspective should be reassessed. We show how our attack scenarios can generate critical trust signals that malicious actors could abuse. We analyzed repositories on GitHub for 50,328 critical OSS packages and measured their vulnerability to these weaknesses. Additionally, we investigated whether countermeasures against these weaknesses were taken in projects, and we measured how widespread GitHub's recommendation to sign commits is in practice. We further analyzed security advice for developers for dealing with impersonations on GitHub. Our results show that GitHub's current presentation makes it hard for users to attribute commits correctly. We also identified 3,013,817 commits with no linked GitHub account that malicious attackers can use to attribute open-source contributions. Although security advice recommends signing commits, it is rare in critical OSS projects.

XI. Acknowledgments

This work was co-funded by the VolkswagenStiftung Niedersächsisches Vorab – ZN3695, the European Union as part of the program *Digital Europe*, and by NSF grant CNS-2207008. Any findings and opinions expressed in this material are those of the authors and do not necessarily reflect the views of the funding agencies. Neither the European Union nor the other granting authorities can be held responsible for them. We also thank the anonymous reviewers and our shepherd for their valuable feedback.

References

[1] N. Harutyunyan, *"Managing your open source supply chain-why and how?"* *Computer*, vol. 53, no. 6, pp. 77–81, 2020.

[2] M. Giladi and G. David. *"Over 100,000 Infected Repos Found on GitHub."* (Feb. 28, 2024), [Online]. Available: https://apiiro.com/blog/malicious-code-campaign-github-repo-confusion-attack/ (visited on 11/18/2024).

[3] NIST. *"CVE-2023-23839."* (Aug. 3, 2023), [Online]. Available: https://nvd.nist.gov/vuln/detail/CVE-2023-23839 (visited on 11/18/2024).

[4] A. Sharma. *"Malicious PyPI package 'VMConnect' imitates VMware vSphere connector module."* (Aug. 3, 2023), [Online]. Available: https://blog.sonatype.com/malicious-pypi-package-vmconnect-imitates-vmware-vsphere-connector-module (visited on 11/18/2024).

[5] S. James. *"FAQ on the xz-utils backdoor (CVE-2024-3094)."* (2024), [Online]. Available: https://gist.github.com/thesamesam/223949d5a074ebc3dce9ee78baad9e27 (visited on 11/18/2024).

[6] Executive Office of the President, *Executive Order 14028: Improving the Nation's Cybersecurity*, https://www.federalregister.gov/documents/2021/05/17/2021-10460/improving-the-nations-cybersecurity, 2021.

[7] Cybersecurity and Infrastructure Security Agency (CISA). *" Secure Software Development Attestation Form."* (Mar. 18, 2024), [Online]. Available: https://www.cisa.gov/resources-tools/resources/secure-software-development-attestation-form (visited on 11/18/2024).

[8] Open Source Security Foundation. *"OpenSSF Scorecard."* (2023), [Online]. Available: https://scorecard.dev (visited on 11/18/2024).

[9] D. Wermke, N. Wöhler, J. H. Klemmer, M. Fourné, Y. Acar, and S. Fahl, *"Committed to trust: A qualitative study on security & trust in open source software projects,"* in *2022 IEEE Symposium on Security and Privacy (SP)*, IEEE, 2022, pp. 1880–1896.

[10] K. Thompson, *"Reflections on trusting trust,"* *Communications of the ACM*, vol. 27, no. 8, pp. 761–763, 1984.

[11] L. Dabbish, C. Stuart, J. Tsay, and J. Herbsleb, *"Social coding in GitHub: transparency and collaboration in an open software repository,"* in *Proc. ACM 2012 Conference on Computer Supported Cooperative Work*, ser. CSCW '12, ACM, 2012, pp. 1277–1286.

[12] F. Nagle, D. A. Wheeler, H. Lifshitz-Assaf, H. Ham, and J. L. Hoffman, *"Report on the 2020 FOSS Contributor Survey,"* Dec. 10, 2020.

[13] P. Ladisa, H. Plate, M. Martinez, and O. Barais, *"SoK: Taxonomy of Attacks on Open-Source Software Supply Chains,"* in *2023 IEEE Symposium on Security and Privacy (SP)*, 2023, pp. 1509–1526.

[14] GitHub. *"Viewing contributions on your profile."* (2024), [Online]. Available: https://docs.github.com/en/account-and-profile/setting-up-and-managing-your-github-profile/managing-contribution-settings-on-your-profile/viewing-contributions-on-your-profile (visited on 11/18/2024).

[15] C. Jones. *"Hackers are spoofing themselves as GitHub's Dependabot to steal user passwords."* (Sep. 27, 2023), [Online]. Available: https://www.itpro.com/security/cyber-attacks/hackers-are-spoofing-themselves-as-githubs-dependabot-to-steal-user-passwords (visited on 11/18/2024).

[16] S. Chacon and B. Straub, *"7.4 Git Tools - Signing Your Work,"* in *Pro Git*, 2nd ed. Nov. 14, 2024, Version 2.1.438.

[17] GitHub Security. *"Bug Bounty Programm: Ineligible submissions."* (2024), [Online]. Available: https://bounty.github.com/ineligible.html (visited on 11/18/2024).

[18] Open Source Security Foundation (OpenSSF) Best Practices Working Group. *"Source Code Management Platform Configuration Best Practices."* (Aug. 29, 2023), [Online]. Available: https://best.openssf.org/SCM-BestPractices/ (visited on 11/18/2024).

[19] I. Haddad, *"Recommended Practices for Hosting and Managing Open Source Projects on GitHub,"* The Linux Foundation - Research, Mar. 2023.

[20] C. Stransky, O. Wiese, V. Roth, Y. Acar, and S. Fahl, *"27 Years and 81 Million Opportunities Later: Investigating the Use of Email Encryption for an Entire University,"* in *43rd IEEE Symposium on Security and Privacy, IEEE S&P 2022, May 22-26, 2022*, IEEE Computer Society, May 2022.

[21] S. Chacon. *"Signing Commits in Git, Explained."* (Sep. 25, 2023), [Online]. Available: https://blog.gitbutler.com/signing-commits-in-git-explained/ (visited on 11/18/2024).

[22] A. Bosu and J. C. Carver, *"Impact of developer reputation on code review outcomes in OSS projects: an empirical investigation,"* in *Proc. 8th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement*, ser. ESEM '14, ACM, 2014.

[23] M. Syeed, J. Lindman, and I. Hammouda, *"Measuring Perceived Trust in Open Source Software Communities,"* in *13th IFIP International Conference on Open Source Systems (OSS)*, ser. Open Source Systems: Towards Robust Practices, Part 2: Posters and Tools, vol. AICT-496, Springer, May 2017, pp. 49–54.

[24] G. M. Alarcon, A. M. Gibson, C. Walter, R. F. Gamble, T. J. Ryan, S. A. Jessup, B. E. Boyd, and A. Capiola, *"Trust Perceptions of Metadata in Open-Source Software: The Role of Performance and Reputation,"* *Systems*, vol. 8, no. 3, 2020.

[25] C. Okafor, T. R. Schorlemmer, S. Torres-Arias, and J. C. Davis, *"SoK: Analysis of Software Supply Chain Security by Establishing Secure Design Properties,"* in *Proc. 2022 ACM Workshop on Software Supply Chain Offensive Research and Ecosystem Defenses*, ser. SCORED'22, New York, NY, USA: ACM, 2022, pp. 15–24.

[26] M. Ohm, H. Plate, A. Sykosch, and M. Meier, *"Backstabber's knife collection: A review of open source software supply chain attacks*," in *Detection of Intrusions and Malware, and Vulnerability Assessment: 17th International Conference, DIMVA 2020, Lisbon, Portugal, June 24–26, 2020, Proceedings 17*, Springer, 2020, pp. 23–43.

[27] W. Enck and L. Williams, *"Top Five Challenges in Software Supply Chain Security: Observations From 30 Industry and Government Organizations*," *IEEE Security & Privacy*, vol. 20, no. 2, pp. 96–100, 2022.

[28] M. Mirakhorli, D. Garcia, S. Dillon, K. Laporte, M. Morrison, H. Lu, V. Koscinski, and C. Enoch, *A Landscape Study of Open Source and Proprietary Tools for Software Bill of Materials (SBOM)*, 2024. arXiv: 2402.11151 [cs.CL].

[29] B. Xia, T. Bi, Z. Xing, Q. Lu, and L. Zhu, *"An empirical study on software bill of materials: Where we stand and the road ahead*," in *2023 IEEE/ACM 45th International Conference on Software Engineering (ICSE)*, IEEE, 2023, pp. 2630–2642.

[30] M. Balliu, B. Baudry, S. Bobadilla, M. Ekstedt, M. Monperrus, J. Ron, A. Sharma, G. Skoglund, C. Soto-Valero, and M. Wittlinger, *"Challenges of Producing Software Bill of Materials for Java*," *IEEE Security & Privacy*, vol. 21, no. 6, pp. 12–23, Nov. 2023.

[31] C. Lamb and S. Zacchiroli, *"Reproducible builds: Increasing the integrity of software supply chains*," *IEEE Software*, vol. 39, no. 2, pp. 62–70, 2021.

[32] M. Fourné, D. Wermke, W. Enck, S. Fahl, and Y. Acar, *"It's like flossing your teeth: On the Importance and Challenges of Reproducible Builds for Software Supply Chain Security*," in *2023 IEEE Symposium on Security and Privacy (SP)*, 2023, pp. 1527–1544.

[33] D. Wheeler, *"Countering trusting trust through diverse double-compiling*," in *21st Annual Computer Security Applications Conference (ACSAC'05)*, 2005, 13 pp.–48.

[34] S. Torres-Arias, H. Afzali, T. K. Kuppusamy, R. Curtmola, and J. Cappos, *"in-toto: Providing farm-to-table guarantees for bits and bytes*," in *28th USENIX Security Symposium (USENIX Security 19)*, Santa Clara, CA: USENIX, Aug. 2019, pp. 1393–1410.

[35] M. Zimmermann, C.-A. Staicu, C. Tenny, and M. Pradel, *"Small World with High Risks: A Study of Security Threats in the npm Ecosystem*," in *28th USENIX Security Symposium (USENIX Security 19)*, Santa Clara, CA: USENIX, Aug. 2019, pp. 995–1010.

[36] N. Zahan, T. Zimmermann, P. Godefroid, B. Murphy, C. Maddila, and L. Williams, *"What are weak links in the npm supply chain?"* In *Proc. 44th International Conference on Software Engineering: Software Engineering in Practice*, ser. ICSE-SEIP '22, Pittsburgh, Pennsylvania: ACM, 2022, pp. 331–340.

[37] J. Coelho and M. T. Valente, *"Why modern open source projects fail*," in *Proc. 2017 11th Joint Meeting on Foundations of Software Engineering*, ser. ESEC/FSE 2017, ACM, 2017, pp. 186–196.

[38] K. Constantino, M. Souza, S. Zhou, E. Figueiredo, and C. Kästner, *"Perceptions of open-source software developers on collaborations: An interview and survey study*," *Journal of Software: Evolution and Process*, vol. 35, no. 5, e2393, Oct. 2021.

[39] M. Zahedi, M. Ali Babar, and C. Treude, *"An Empirical Study of Security Issues Posted in Open Source Projects*," in *Hawaii International Conference on System Sciences (HICSS)*, 2018, pp. 5504–5513.

[40] S. Torres-Arias, A. K. Ammula, R. Curtmola, and J. Cappos, *"On omitting commits and committing omissions: Preventing git metadata tampering that (re) introduces software vulnerabilities*," in *25th USENIX Security Symposium (USENIX Security 16)*, 2016, pp. 379–395.

[41] D. Gonzalez, T. Zimmermann, P. Godefroid, and M. Schäfer, *"Anomalicious: automated detection of anomalous and potentially malicious commits on GitHub*," in *Proc. 43rd International Conference on Software Engineering: Software Engineering in Practice*, ser. ICSE-SEIP '21, Virtual Event, Spain: IEEE, 2021, pp. 258–267.

[42] H. Afzali, S. Torres-Arias, R. Curtmola, and J. Cappos, *"le-git-imate: Towards Verifiable Web-based Git Repositories*," in *Proc. 2018 on Asia Conference on Computer and Communications Security*, ser. ASIACCS '18, Incheon, Republic of Korea: ACM, 2018, pp. 469–482.

[43] A. Whitten and J. D. Tygar, *"Why Johnny Can't Encrypt: A Usability Evaluation of PGP 5.0*," in *Proc. 8th Usenix Security Symposium (SEC'99)*, USENIX, 1999.

[44] S. Ruoti, N. Kim, B. Burgon, T. W. van der Horst, and K. E. Seamons, *"Confused Johnny: when automatic encryption leads to confusion and mistakes*," in *Symposium On Usable Privacy and Security, SOUPS '13, Newcastle, United Kingdom, July 24-26, 2013*, 2013, 5:1–5:12.

[45] S. Ruoti, J. Andersen, S. Heidbrink, M. O'Neill, E. Vaziripour, J. Wu, D. Zappala, and K. Seamons, *""We're on the Same Page" A Usability Study of Secure Email Using Pairs of Novice Users*," in *Proc. CHI Conference on Human Factors in Computing Systems (CHI'16)*, 2016.

[46] S. Ruoti, J. Andersen, T. Hendershot, D. Zappala, and K. E. Seamons, *"Private Webmail 2.0: Simple and Easy-to-Use Secure Email*," in *Proc. 29th Annual Symposium on User Interface Software and Technology, UIST 2016*, ACM, 2016, pp. 461–472.

[47] S. Ruoti, J. Andersen, T. Monson, D. Zappala, and K. Seamons, *"A Comparative Usability Study of Key Management in Secure Email*," in *Proc. Fourteenth USENIX Conference on Usable Privacy and Security*, ser. SOUPS '18, Baltimore, MD, USA: USENIX, 2018, pp. 375–394.

[48] V. Roth, T. Straub, and K. Richter, *"Security and usability engineering with particular attention to electronic mail*," *International Journal of Human-Computer Studies*, vol. 63, no. 1, pp. 51–73, 2005.

[49] E. Atwater, C. Bocovich, U. Hengartner, E. Lank, and I. Goldberg, *"Leading Johnny to Water: Designing for Usability and Trust*," in *Eleventh Symposium On Usable Privacy and Security (SOUPS 2015)*, Ottawa: USENIX, Jul. 2015, pp. 69–88.

[50] W. Bai, M. Namara, Y. Qian, P. G. Kelley, M. L. Mazurek, and D. Kim, *"An Inconvenient Trust: User Attitudes toward Security and Usability Tradeoffs for Key-Directory Encryption Systems*," in *Twelfth Symposium on Usable Privacy and Security (SOUPS 2016)*, Denver, CO: USENIX, Jun. 2016, pp. 113–130.

[51] W. Bai, D. Kim, M. Namara, Y. Qian, P. G. Kelley, and M. L. Mazurek, *"Balancing Security and Usability in Encrypted Email*," *IEEE Internet Computing*, vol. 21, no. 3, pp. 30–38, 2017.

[52] C. Stransky, D. Wermke, J. Schrader, N. Huaman, Y. Acar, A. L. Fehlhaber, M. Wei, B. Ur, and S. Fahl, *"On the Limited Impact of Visualizing Encryption: Perceptions of E2E Messaging Security*," in *Seventeenth Symposium on Usable Privacy and Security (SOUPS 2021)*, USENIX, Aug. 2021, pp. 437–454.

[53] O. Akgul, W. Bai, S. Das, and M. L. Mazurek, *"Evaluating In-Workflow Messages for Improving Mental Models of End-to-End Encryption*," in *30th USENIX Security Symposium (USENIX Security 21)*, USENIX, Aug. 2021, pp. 447–464.

[54] R. Abu-Salma, M. A. Sasse, J. Bonneau, A. Danilova, A. Naiakshina, and M. Smith, *"Obstacles to the Adoption of Secure Communication Tools*," in *2017 IEEE Symposium on Security and Privacy, SP 2017, San Jose, CA, USA, May 22-26, 2017*, 2017, pp. 137–153.

[55] R. Abu-Salma, E. M. Redmiles, B. Ur, and M. Wei, *"Exploring User Mental Models of End-to-End Encrypted Communication Tools*," in *8th USENIX Workshop on Free and Open Communications on the Internet (FOCI 18)*, Baltimore, MD: USENIX, 2018.

[56] S. Fahl, M. Harbach, T. Muders, M. Smith, and U. Sander, *"Helping Johnny 2.0 to Encrypt His Facebook Conversations*," in *Proc. 8th Symposium on Usable Privacy and Security (SOUPS'12)*, ACM, 2012.

[57] S. L. Garfinkel, J. I. Schiller, E. Nordlander, D. Margrave, and R. C. Miller, *"Views, Reactions and Impact of Digitally-Signed Mail in e-Commerce*," in *Financial Cryptography and Data Security*, Springer, 2005, pp. 188–202.

[58] S. L. Garfinkel, D. Margrave, J. I. Schiller, E. Nordlander, and R. C. Miller, *"How to Make Secure Email Easier to Use*," in *Proc. SIGCHI Conference on Human Factors in Computing Systems*, ser. CHI '05, Portland, Oregon, USA: ACM, 2005, pp. 701–710.

[59] P. Gutmann, *"Why isn't the Internet secure yet, dammit*," in *AusCERT Asia Pacific Information Technology Security Conference*, 2004.

[60] N. Unger, S. Dechand, J. Bonneau, S. Fahl, H. Perl, I. Goldberg, and M. Smith, *"SoK: secure messaging*," in *2015 IEEE Symposium on Security and Privacy*, IEEE, 2015, pp. 232–249.

[61] W. Bai, M. Namara, Y. Qian, P. G. Kelley, M. L. Mazurek, and D. Kim, *"An Inconvenient Trust: User Attitudes toward Security and Usability Tradeoffs for Key-Directory Encryption Systems*," in *Twelfth Symposium on Usable Privacy and Security (SOUPS 2016)*, 2016, pp. 113–130.

[62] S. L. Garfinkel and R. C. Miller, *"Johnny 2: a user test of key continuity management with S/MIME and Outlook Express*," in *Proc. 1st Symposium on Usable Privacy and Security (SOUPS'05)*, ACM, 2005.

[63] A. Ulrich, R. Holz, P. Hauck, and G. Carle, *"Investigating the OpenPGP Web of Trust*," in *European Symposium on Research in Computer Security*, Springer, 2011, pp. 489–507.

[64] A. Ulrich, R. Holz, P. Hauck, and G. Carle, *"Investigating the openpgp web of trust*," in *Computer Security–ESORICS 2011: 16th European*

Symposium on Research in Computer Security, Springer, 2011, pp. 489–507.

[65] A. Barenghi, A. Di Federico, G. Pelosi, and S. Sanfilippo, *“Challenging the trustworthiness of pgp: Is the web-of-trust tear-proof?”* In *Computer Security–ESORICS 2015: 20th European Symposium on Research in Computer Security*, Springer, 2015, pp. 429–446.

[66] B. Schacht and P. Kieseberg, *“An Analysis of 5 Million OpenPGP Keys.*,” *J. Wirel. Mob. Networks Ubiquitous Comput. Dependable Appl.*, vol. 11, no. 3, pp. 107–140, 2020.

[67] A. Lerner, E. Zeng, and F. Roesner, *“Confidante: Usable Encrypted Email: A Case Study with Lawyers and Journalists*,” in *2017 IEEE European Symposium on Security and Privacy (EuroS&P)*, IEEE Computer Society, Apr. 2017, pp. 385–400.

[68] M. Krohn and C. Coyne. *“Keybase.io.*” (2014), [Online]. Available: https://keybase.io/ (visited on 12/02/2024).

[69] S. E. McGregor, E. A. Watkins, M. N. Al-Ameen, K. Caine, and F. Roesner, *“When the Weakest Link is Strong: Secure Collaboration in the Case of the Panama Papers*,” in *26th USENIX Security Symposium (USENIX Security 17)*, USENIX, Aug. 2017, pp. 505–522.

[70] S. Waters, *“The effects of mass surveillance on journalists’ relations with confidential sources: A constant comparative study*,” *Digital Journalism*, vol. 6, no. 10, pp. 1294–1313, 2018.

[71] T. R. Schorlemmer, K. G. Kalu, L. Chigges, K. M. Ko, E. A.-M. A. Isghair, S. Baghi, S. Torres-Arias, and J. C. Davis, *Signing in Four Public Software Package Registries: Quantity, Quality, and Influencing Factors*, 2024. arXiv: 2401.14635 [cs.CR].

[72] K. Merrill, Z. Newman, S. Torres-Arias, and K. Sollins, *Speranza: Usable, privacy-friendly software signing*, 2023. arXiv: 2305.06463 [cs.CR].

[73] Stack Overflow. *“Stack Overflow Developer Survey 2022.*” (2022), [Online]. Available: https://survey.stackoverflow.co/2022/#section-version-control-version-control-systems.

[74] git-scm. *“3.4 Git Branching - Branching-Workflows.*” (2014), [Online]. Available: https://git-scm.com/book/de/v2/Git-Branching-Branching-Workflows.

[75] S. Chacon and B. Straub, *Pro git*. Springer Nature, 2014.

[76] R. Love, *Linux kernel development*. Pearson Education, 2010.

[77] GitHub. *“About authentication to GitHub.*” (2024), [Online]. Available: https://docs.github.com/en/authentication/keeping-your-account-and-data-secure/about-authentication-to-github (visited on 12/02/2024).

[78] OpenBSD Foundation. *“OpenSSH Release Notes.*” (2024), [Online]. Available: https://www.openssh.com/releasenotes.html.

[79] K. Blincoe, J. Sheoran, S. Goggins, E. Petakovic, and D. Damian, *“Understanding the popular users: Following, affiliation influence and leadership on GitHub*,” *Information and Software Technology*, vol. 70, pp. 30–39, 2016.

[80] J. Xavier, A. Macedo, and M. de Almeida Maia, *“Understanding the popularity of reporters and assignees in the Github.*,” in *SEKE*, 2014, pp. 484–489.

[81] A. S. Badashian and E. Stroulia, *“Measuring user influence in GitHub: the million follower fallacy*,” in *Proc. 3rd International Workshop on CrowdSourcing in Software Engineering*, 2016, pp. 15–21.

[82] A. Carpenter. *“How to Use GitHub to Strengthen Your Resume.*” (Sep. 27, 2024), [Online]. Available: https://www.codecademy.com/resources/blog/how-to-use-github-to-strengthen-your-resume/.

[83] Linkedin. *“How do you use GitHub to source and assess technical talent?*” (2024), [Online]. Available: https://www.linkedin.com/advice/0/how-do-you-use-github-source-assess-technical (visited on 12/02/2024).

[84] GitHub. *“Verifying Your Email Adress.*” (2024), [Online]. Available: https://docs.github.com/en/account-and-profile/setting-up-and-managing-your-personal-account-on-github/managing-email-preferences/verifying-your-email-address (visited on 12/02/2024).

[85] GitHub. *“Signing commits.*” (2024), [Online]. Available: https://docs.github.com/en/authentication/managing-commit-signature-verification/signing-commits.

[86] F. Nagle, J. Dana, J. Hoffman, S. Randazzo, and Y. Zhou, *“Census II of Free and Open Source Software—Application Libraries*,” *Linux Foundation, Harvard Laboratory for Innovation Science (LISH) and Open Source Security Foundation (OpenSSF)*, vol. 80, 2022.

[87] A. Decan, T. Mens, and P. Grosjean, *“An empirical comparison of dependency network evolution in seven software packaging ecosystems*,” *Empirical Software Engineering*, vol. 24, no. 1, pp. 381–416, 2019.

[88] D. A. Wolfe, *“Ranked set sampling*,” *Wiley Interdisciplinary Reviews: Computational Statistics*, vol. 2, no. 4, pp. 460–466, 2010.

[89] Tidelift, Inc. *“libraries.io.*” (2024), [Online]. Available: https://libraries.io/.

[90] M. Wessel, B. M. De Souza, I. Steinmacher, I. S. Wiese, I. Polato, A. P. Chaves, and M. A. Gerosa, *“The power of bots: Characterizing and understanding bots in oss projects*,” *Proc. ACM on Human-Computer Interaction*, vol. 2, no. CSCW, pp. 1–19, 2018.

[91] E. Kalliamvakou, G. Gousios, K. Blincoe, L. Singer, D. M. German, and D. Damian, *“The promises and perils of mining GitHub*,” in *Proc. 11th Working Conference on Mining Software Repositories*, 2014, pp. 92–101.

[92] GitHub. *“REST API endpoints for events.*” (2024), [Online]. Available: https://docs.github.com/en/rest/activity/events?apiVersion=2022-11-28 (visited on 12/02/2024).

[93] GitHub. *“Best Practices for Pull Requests.*” (2024), [Online]. Available: https://docs.github.com/en/pull-requests/collaborating-with-pull-requests/getting-started/best-practices-for-pull-requests (visited on 12/02/2024).

[94] GitHub. *“Managing email preferences.*” (2024), [Online]. Available: https://docs.github.com/en/account-and-profile/setting-up-and-managing-your-personal-account-on-github/managing-email-preferences (visited on 11/18/2024).

[95] GitHub. *“Changing your primary email address.*” (2024), [Online]. Available: https://docs.github.com/de/account-and-profile/setting-up-and-managing-your-personal-account-on-github/managing-email-preferences/changing-your-primary-email-address (visited on 12/02/2024).

[96] GitHub. *“Setting a backup email address.*” (2024), [Online]. Available: https://docs.github.com/de/account-and-profile/setting-up-and-managing-your-personal-account-on-github/managing-email-preferences/setting-a-backup-email-address (visited on 12/02/2024).

[97] Internet Engineering Task Force, *Domain names - implementation and specification*, RFC 1035, Nov. 1987.

[98] GitHub. *“REST API endpoints for commits.*” (2022), [Online]. Available: https://docs.github.com/en/rest/commits/commits?apiVersion=2022-11-28 (visited on 12/02/2024).

[99] GitHub, *SSH commit verification now supported*, https://github.blog/changelog/2022-08-23-ssh-commit-verification-now-supported/, Aug. 23, 2022.

[100] GitHub. *“GPG signature verification.*” (2016), [Online]. Available: https://github.blog/security/application-security/gpg-signature-verification/ (visited on 12/02/2024).

[101] V. Garousi, M. Felderer, and M. V. Mäntylä, *“Guidelines for including grey literature and conducting multivocal literature reviews in software engineering*,” *Information and software technology*, vol. 106, pp. 101–121, 2019.

[102] D. Giustini, *“Retrieving grey literature, information, and data in the digital age*,” *The handbook of research synthesis and meta-analysis*, pp. 101–126, 2019.

[103] L. Neil, E. Bouma-Sims, E. Lafontaine, Y. Acar, and B. Reaves, *“Investigating Web Service Account Remediation Advice*,” in *Seventeenth Symposium on Usable Privacy and Security (SOUPS 2021)*, USENIX, Aug. 2021, pp. 359–376.

[104] J. H. Klemmer, M. Gutfleisch, C. Stransky, Y. Acar, M. A. Sasse, and S. Fahl, *““Make Them Change it Every Week!”: A Qualitative Exploration of Online Developer Advice on Usable and Secure Authentication*,” in *2023 ACM SIGSAC Conference on Computer and Communications Security (CCS ’23)*, ACM, Nov. 2023.

[105] S. Wang, H. Scells, B. Koopman, and G. Zuccon, *“Can ChatGPT write a good boolean query for systematic review literature search?*” In *Proc. 46th International ACM SIGIR Conference on Research and Development in Information Retrieval*, 2023, pp. 1426–1436.

[106] Y. Acar, M. Backes, S. Fahl, D. Kim, M. L. Mazurek, and C. Stransky, *“You get where you’re looking for: The impact of information sources on code security*,” in *2016 IEEE symposium on security and privacy (SP)*, IEEE, 2016, pp. 289–305.

[107] GPTZero, *Quantify AI with Advanced Scan*, https://gptzero.me/, 2024.

[108] V. Clarke and V. Braun, *“Thematic Analysis*,” in *Encyclopedia of Critical Psychology*, T. Teo, Ed. New York, NY: Springer New York, 2014, pp. 1947–1952.

[109] N. McDonald, S. Schoenebeck, and A. Forte, *“Reliability and Inter-Rater Reliability in Qualitative Research: Norms and Guidelines for CSCW and HCI Practice*,” *ACM on Human-Computer Interaction*, vol. 3, no. CSCW, 2019.

[110] A. McDonald, C. Barwulor, M. L. Mazurek, F. Schaub, and E. M. Redmiles, *“"It’s stressful having all these phones": Investigating Sex Workers’ Safety Goals, Risks, and Practices Online*,” in *30th USENIX Security Symposium*, USENIX, 2021, pp. 375–392.

[111] X. Bouwman, H. Griffioen, J. Egbers, C. Doerr, B. Klievink, and M. van Eeten, *“A different Cup of TI? The Added Value of Commercial Threat Intelligence*,” in *29th USENIX Security Symposium (USENIX Security 20)*, USENIX, Aug. 2020, pp. 433–450.

[112] M. Bailey, D. Dittrich, E. Kenneally, and D. Maughan, *“The Menlo Report*,” *IEEE Security & Privacy*, vol. 10, pp. 71–75, Mar. 2012.

[113] GitHub Security. *“GitHub Bounty Program: Ineligible Submissions*.” (2014), [Online]. Available: https://web.archive.org/web/20150401000000*/https://bounty.github.com/ineligible.html%5C#impersonating_a_user_through_git_email_address.

[114] M. S. Melara, A. Blankstein, J. Bonneau, E. W. Felten, and M. J. Freedman, *“CONIKS: Bringing Key Transparency to End Users*,” in *24th USENIX Security Symposium (USENIX Security 15)*, Washington, D.C.: USENIX, Aug. 2015, pp. 383–398.

[115] M. Chase, A. Deshpande, E. Ghosh, and H. Malvai, *“Seemless: Secure end-to-end encrypted messaging with less trust*,” in *Proc. 2019 ACM SIGSAC Conference on Computer and Communications Security*, 2019, pp. 1639–1656.

[116] H. Malvai, L. Kokoris-Kogias, A. Sonnino, E. Ghosh, E. Oztürk, K. Lewi, and S. Lawlor, *“Parakeet: Practical key transparency for end-to-end encrypted messaging*,” in *2023 Network and Distributed Systems Security Symposium*, 2023.

[117] Z. Newman, J. S. Meyers, and S. Torres-Arias, *“Sigstore: Software Signing for Everybody*,” in *Proc. 2022 ACM SIGSAC Conference on Computer and Communications Security*, ser. CCS ’22, New York, NY, USA: ACM, 2022, pp. 2353–2367.

[118] Gitsign Team. *“gitsign*.” (2024), [Online]. Available: https://github.com/sigstore/gitsign.

Appendix A
Availability

Additional materials can be found at our OSF repository: https://doi.org/10.17605/OSF.IO/9RVSX

Appendix B
Responsible Disclosure

We disclosed our findings to GitHub via HackerOne:

- **July 15, 2024:** We submitted a vulnerability report via HackerOne to GitHub, which included our PoCs with videos, our publication, and an impact assessment.
- **July 15, 2024:** Hubot replied with an automated message thanking for the report, which will now be validated in a triage process before being forwarded to a team (Quote: “Once validated, we will let you know and triage this issue to the appropriate team.”).
- **August 23, 2024:** A member of the GitHub Security team contacted us via HackerOne, stating that their investigation is still ongoing. They asked when we plan to publish our findings (Quote: “Thanks again for the submission! We are still investigating and hope to have more information for you soon. Do you have a date in mind for publication? Thank you!”).
- **August 26, 2024:** We replied that we plan to publish our findings in February 2025 (NDSS).
- **September 16, 2024:** We asked for an update for NDSS’ interactive rebuttal phase.
- **October 21, 2024:** GitHub closed our report as an *informative disclosure*. The security team confirmed that our PoCs are correct, but the behaviors we have highlighted are working as intended. However, GitHub might consider making this functionality more strict in the future.

Appendix C
Search Terms

Used search terms for finding online security advice and the analysis of project guidelines: *commit spoofing*, *github contribution spoofing*, *author impersonation git*, *github contribution hijacking*, *github false contributor*, *tampered git commits*, *git author spoofing*, *commit identity spoofing*, *github reputation hijacking*, *commit signing*, *trustworthy source code commits*, *commit signature*, *commit signing guidelines*, *commit authentication*, *version control system signatures*, and *digital signatures in source code*.

Appendix D
Measurement Study

A. Results of the Linear Regression Analysis

We modeled the proportion of user-signed commits (dependent variable) in a project based on: project age, programming language, number of contributors, project topic, Scorecard value and number of dependents. For statistical stability, programming languages that occur in less than 1% of the data set were excluded. `score` refers to the Scorecard value of a repository, `100k_dependent_repositories` is true for projects that have more than 100,000 dependent projects, and `1k_stars` is true for projects with more than 1000 stars on GitHub. Table IV shows the results for the regression.

TABLE IV. Results of the Regression Analysis.

Factor	Coef.	C.I.	p-value
Intercept	0.09	[0.08, 0.09]	<0.001*
language_Ruby	0.03	[0.02, 0.03]	<0.001*
language_TypeScript	0.02	[0.01, 0.02]	<0.001*
language_PHP	0.007850	[0.00, 0.01]	<0.001*
language_Rust	0.02	[0.01, 0.02]	<0.001*
language_HTML	-0.01	[-0.02, -0.00]	0.015*
language_CoffeeScript	-0.008494	[-0.02, 0.00]	0.052
age	-0.01	[-0.01, -0.01]	<0.001*
score	0.02	[0.02, 0.02]	<0.001*
100k_dependent_repositories	0.0001682	[0.00, 0.00]	<0.001*
1k_stars	-0.0002891	[-0.00, -0.00]	0.048*

B. GitHub’s Verification

We found two undocumented verification reasons when working with the GitHub API [98].

`ocsp_revoked` are signatures that are no longer valid because a certificate in their chain has been revoked. Unverified commits with the `bad_cert` status also include commits signed with Sigstore and gitsign [117], [118]. As GitHub does not accept Sigstore as a certificate authority, they are shown as *Unverified*. This is only shown in the GitHub UI and not in the GitHub API. This means we can only report an upper bound of 363 for Sigstore since the `bad_cert` verification reason appears frequently in our dataset. The documentation also states that the `unknown_signature_type` error occurs if the verification encounters a non-PGP signature. However, this is outdated, as shown by the data in Table II.

Apart from that, we found a few commits that are still shown as verified, even though the commit was made after the key expired. GitHub displays commits signed with this key as *Verified* with an additional *key expiration* note, independently of the commit or upload date. A few keys GitHub specified as expired were renewed by users on keyservers.

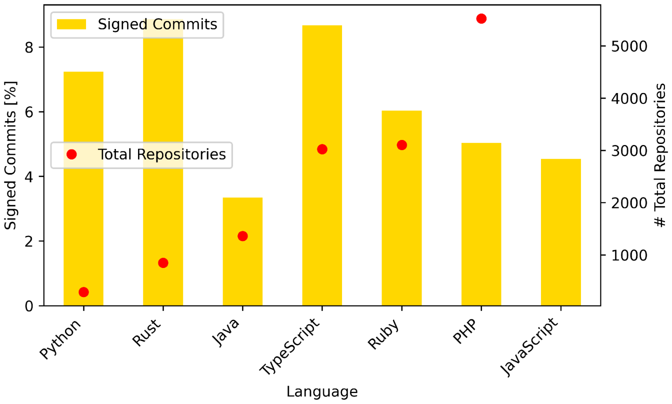


Fig. 8. Share of signed commits and the number of repositories per programming language. JavaScript's number of repositories (29,082) was omitted for better readability.

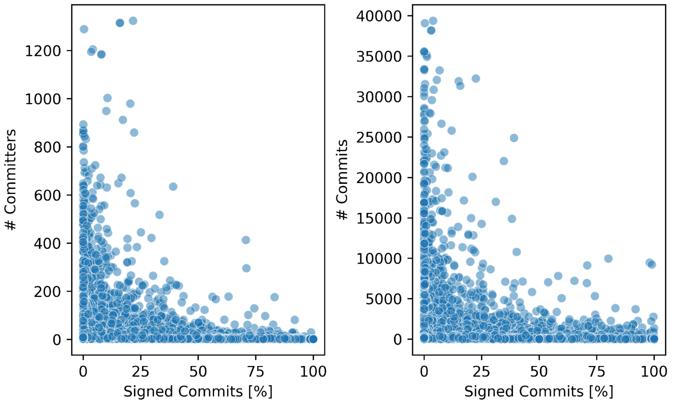


Fig. 9. Signing behavior diminishes with more commits or committers. Contributors were cut off at 1500, leaving out 6 repositories with an average signing of 3.8%. Commits were cut off at 40000, leaving out 28 repositories with an average signing of 6.3%.

C. Key Algorithms

TABLE V. ALGORITHMS USED FOR PGP AND SSH KEYS.

PGP			SSH		
Algorithm	Count	%	Algorithm	Count	%
RSA-4096	47,472	61%	ED25519	6,042	73%
ED25519	9,224	12%	RSA-4096	658	8%
RSA-2048	7,438	9%	RSA-2048	537	6%
RSA-3072	5,074	6%	RSA-3072	467	6%
DSA-1024	626	1%	ECDSA-256	278	3%
Other	8,550	11%	Other	299	4%
Total	78,384			8,281	

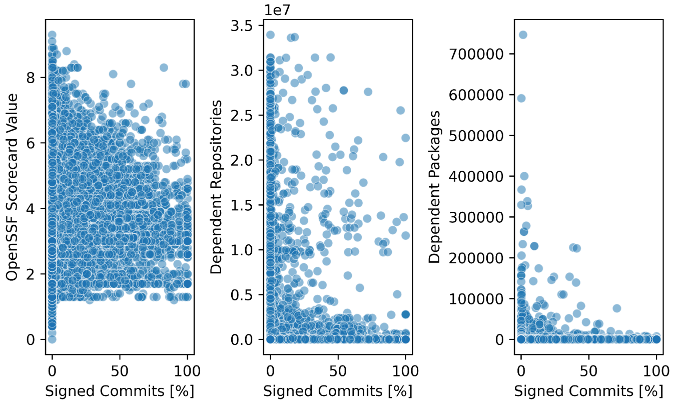


Fig. 10. Signing behavior in relation to OpenSSF Scorecard value and number of dependents, either measured as repositories or packages.

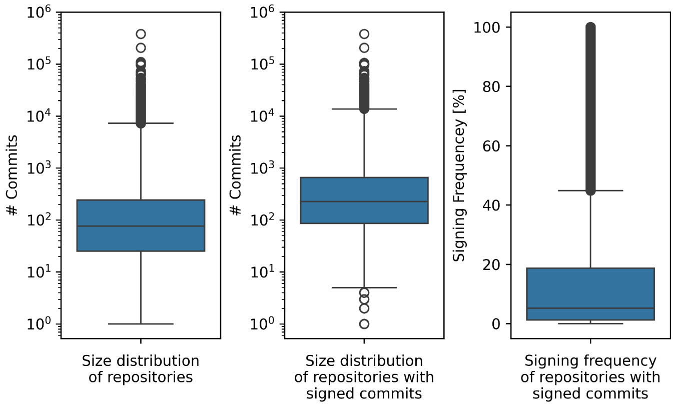


Fig. 11. Distribution of commit quantity for all repositories (left), those with at least one signed commit (middle), and the distribution of signing frequency for repositories with at least one signed commit (right).

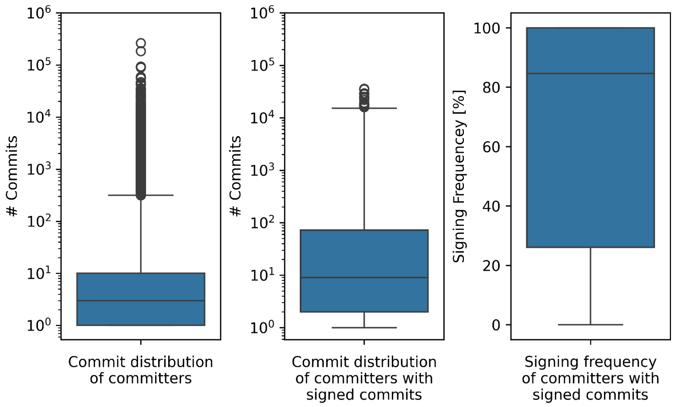


Fig. 12. The distribution of commit quantity for all committers (left), those having at least one signed commit (middle), and the distribution of signing frequency for committers having at least one signed commit (right).

Appendix A
Artifact Appendix

Our research focused on potential impersonation attacks on GitHub, which utilize manipulated Git metadata and GitHub's interpretation of this metadata. After we defined our attack scenarios, we took the top 50,328 critical open-source packages and estimated their risk and vulnerability. We also investigated potential countermeasures which were used by projects and discussed in online security advice.

A. Description & Requirements

Our artifacts consist of three parts, each of which supports a different aspect of our study.

- **Proof of Concept Videos:** For each of our three attacks, we provide a proof of concept video to show our attack techniques and what kind of representation they create in GitHub's UI.
- **Project List and Analysis Code:** For the large-scale measurement study, we provide the complete list of the examined projects and scripts used in their analysis. This includes scripts for crawling the GitHub API for projects and contributors, our crawling of PGP keyservers, the exploration of hijackable top-level domains (TLDs), and jupyter-notebook containing some of our analysis steps.
- **Materials for the Analysis of Online Advice:** We provide all investigated online sources, our sampling materials, the AI evaluation results and our codebook for the analysis of the online sources.

All artifact parts contain a separate `README.MD` file describing the content and how to interact with it.

1) How to access: We host our artifacts on OSF. They are available via this link: https://doi.org/10.17605/OSF.IO/9RVSX

2) Hardware dependencies: None.

3) Software dependencies: The following packages are required as prerequisites to access the content of our package:

- **Media Player (like VLC)** (https://www.videolan.org/vlc/)
- **npm** (https://docs.npmjs.com/downloading-and-installing-node-js-and-npm)
- **yarn** (https://classic.yarnpkg.com/lang/en/docs/install/#debian-stable)
- **Python(3.8+)** (https://www.python.org/downloads/)
- **poetry** (https://python-poetry.org/docs/)
- **jupyter-notebook** (https://jupyter.org/install)

Additionally, a **developer token** is required to access the **GitHub API**. This means that the user needs a GitHub account and has to create a classic token without any assigned scopes using the developer settings[4].

4) Benchmarks: Our artifacts include our crawling scripts to create our dataset. No further external models/datasets were used.

B. Artifact Installation & Configuration

To generate the dataset for the measurement study, several prerequisites must be installed for our scripts (cf. Section A-A3). In addition, the GitHub API token must be copied to the `gh_keys.json` file so that it can be accessed by the scripts. The format of the API keys is described in `github-api/README.md`.

C. Experiment Workflow

First, we show the feasibility of our attacks using the videos as a proof of concept. Secondly, we conduct a risk assessment on critical open-source packages that can be considered the main victims of such attacks. To achieve this, a dataset for the projects and their contributors is first created using the sources described in the `Dataset_Description_GitHub.pdf` (e.g., GitHub API, keyservers, domain registrations). Using this dataset, we conduct a number of analyses, such as searching for unassigned commits and the usage of commit signing as a countermeasure.

The analysis of online sources took place independently and deals with discussions about impersonation on GitHub and possible countermeasures.

D. Major Claims

Our artifacts are supporting several claims from our paper:

- (C1): We demonstrate three ways to tamper with Git's metadata to manipulate attributions on GitHub: Contributor Spoofing, Reputation Hijacking and Contribution Hijacking. These attacks are shown by our Proof of Concepts (PoC) on real GitHub repositories. The videos are included in the `Impersonation Techniques` directory and referenced in Sections IV-B and IV-C.
- (C2): We measure which and how many critical open-source projects and their contributors are vulnerable to these exploits. This claim is proven by our experiments E1-E4 which results are reported in Section IV.
- (C3): We show that only a minority of contributors signs their commits. Our utilized procedure is shown in E3 and the results are displayed in Figures 5 & 6 and Tables 2 & 3.
- (C4): We show that online security advice for developers addresses the risk of contributor spoofing, but no source mentions the problem of unattributed emails in GitHub repositories (Contribution Hijacking). Almost all sources recommend signed commits as a solution, but some also mention problems with their use. All materials for this analysis are included in directory `Gray Literature Analysis` and key findings of the codebook are referenced in Section VI-B.

E. Evaluation

We present 4 experiments to display the functionality of our approach. E1 and E2 show how we collect our dataset for projects and contributors. E3 describes the analyzing steps for our major findings. E4 explains how we checked for vulnerable top-level domains with the commit data that can be taken over.

Important Note: We only provide a small sample of 5 projects for the functionality evaluation for

[4] https://docs.github.com/en/authentication/keeping-your-account-and-data-secure/managing-your-personal-access-tokens

E1-3 and a random sample of 49 real domains for E4. The sampled projects are available in `ae-repositories.txt` and the complete projects list is provided in `github_repositories_cleaned.txt`.

1) Experiment (E1): **[Creation of Repository Dataset]** [15 human-minutes + 0.5 compute-hour]: We create a dataset of critical projects and their commits.

[Preparation] Open the `github-api` directory with your console. The file `gh_keys.json` must contain a valid GitHub developer key, and all prerequisites must be installed (see `github-api/README.md` installation steps).

[Execution] Follow the instructions in the `README.md` and run the first two commands (e.g., `download-repo-information.ts` and `augment-repo-info.ts`)

[Results] Our scripts will create a JSON for each project from `ae-repositories.txt` that contains all contributor information about the project.

2) Experiment (E2): **[Creation of Contributor Dataset]** [15 human-minutes + 3 compute-hour]: We create a dataset for all contributors in our projects. We also check whether they have any public keys available on GitHub or on PGP keyservers.
Important notice: It may happen that the keyservers slow down or block the requests after a while. Restarting the script usually solves this issue.

[Preparation] Open the `github-api` directory with your console. The file `gh_keys.json` must contain a valid GitHub developer key, and all prerequisites must be installed (see `github-api/README.md` installation steps).

[Execution] Execute the complete last three commands from the `README.md` in the correct order. For reference, these are `collect-user-information.ts`, `download-user-keys.ts` and `/download-from-keyserver.ts`.

[Results] Our scripts will create two directories and one JSON file. The first (`userKeys`) contains all signing keys that GitHub has from a user, and the second (`keyIDsPerServer`) contains three sub folders with information from each keyserver. The JSON file (`user-info.json`) contains a list of all found user accounts from the GitHub API and their contribution behavior.

3) Experiment (E3): **[Analyze commits & contributors]** [10 human-minutes + 1 compute-hour]: In this experiment, we conduct a number of analyzing steps that were also referenced in the paper.

[Preparation] Open the `github-api` directory with your console.

[Execution] Start the contained jupyter-notebook or run the last command from the `README.md` to get a HTML representation. You can take a look at all analyses steps and also re-run the whole notebook to check its functionality.

[Results] The results from the notebook shows how we check for unassigned commits, signed commits and the general signing behavior within a repository and from a certain user.

4) Experiment (E4): **[Check Free Domains]** [10 human-minutes + 0.5 compute-hour]: In this experiment, we investigate whether found domains from the commit data are available for hijacking.

[Preparation] Open the `Measurement Study` directory with your console. Let poetry create a virtual environment by following the installation steps in the `README.md` file.

[Execution] Execute the script by following the description from the `README.md` file.

[Results] The script takes the domains from `domains.txt` and checks all domains for their *Start of Authority* (SOA) DNS resource records. The results are listed in 5 different files depending on the SOA response. The vulnerable domains from the commits can be found in `free_domains.txt`.

F. Notes

With these artifacts, we only want to demonstrate the functionality of our scripts on a small dataset because the entire analysis took several days with multiple nodes on a HPC infrastructure. It involves more than 50 GB of compressed data. For this reason, it is rather difficult to reproduce our concrete results, but we want to be transparent about our scientific process. Therefore, we only applied for the *Available* and *Functional* artifact badges.