Spilling the Tea: Uncovering TEA Token Abuse in npm

Elizabeth Wyss
University of Kansas
Lawrence, KS, USA
ElizabethWyss@ku.edu

Lorenzo De Carli
University of Calgary
Calgary, CA
Lorenzo.DeCarli@ucalgary.ca

Drew Davidson
University of Kansas
Lawrence, KS, USA
DrewDavidson@ku.edu

Abstract

TEA tokens–an experimental cryptocurrency designed to incentivize and reward high-impact contributions to open-source package ecosystems–launched an incentivized testnet in early 2024 and is now preparing for a full release. At this moment, the repository-wide impacts of TEA on open-source package ecosystems are critically understudied.

In this work, we conduct a measurement study on TEA token-related packages across the entire npm repository, involving the analysis of more than 3.8 million unique packages. From this study, we identify 50,953 packages that are directly registered to claim TEA tokens–in addition to 484,595 dependent packages, which indirectly influence the reward mechanism of TEA–thus encompassing **13.8%** of the entire npm ecosystem.

However, further investigation revealed large swathes of TEA token-related packages that serve no functional purpose–but rather appear designed solely to illegitimately manipulate the reward mechanism of TEA. Through heuristic-based filtering, we estimate that up to **207,101** TEA token-related packages are illegitimate, including **98.6%** of the packages that are directly registered to claim TEA tokens. It is our hope that this research raises broader awareness and action against TEA token-related spam and abuse–across open-source package ecosystems and the implementation of TEA itself.

ACM Reference Format:
Elizabeth Wyss, Lorenzo De Carli, and Drew Davidson. 2025. Spilling the Tea: Uncovering TEA Token Abuse in npm. In *Proceedings of the 2025 Workshop on Software Supply Chain Offensive Research and Ecosystem Defenses (SCORED '25), October 13–17, 2025, Taipei, Taiwan.* ACM, New York, NY, USA, 9 pages. https://doi.org/10.1145/3733827.3765527

1 Introduction

Open-source package ecosystems play a crucial role in modern software development. By enabling developers to quickly fetch and install modular software dependencies, package repositories assist in streamlining the overall process of building software. As a result, large communities of open-source developers and security practitioners [1, 19, 20] have invested substantial time and effort into the creation and maintenance of freely available and open-source package software. Managing the health of these ecosystems–and contending with package spam and malware at scale–is an active research problem.

In an effort to reward package maintainers for high-impact projects, the TEA Association [28] (stylized as 'tea') is designing an experimental cryptocurrency, based on their novel "Proof of Contribution" consensus mechanism, which is aimed at quantifying the impacts of open-source packages [11]. In essence, *TEA tokens* aim to financially incentivize and compensate impactful contributions to open-source package ecosystems. Such rewards are determined primarily by a package's total number of *dependents* (i.e., how many other packages transitively depend on it) [27]. As of March 2025, the tea protocol released its final incentivized testnet to the public and is preparing for its full launch [29].

In spite of best intentions, the tea protocol [28] may inadvertently yield perverse incentives for spammers to manipulate package ecosystems for ill-gotten financial gains. Such spam and abuse harms not only the tea protocol–but also the open-source ecosystems that become flooded with significant waves of package spam. This, in turn, poses negative security consequences for the broader software supply chains that depend on these ecosystems–as package spam is demonstrated to contribute to issues such as package confusion [16, 26]–in addition to burdening the workload of repository crawlers and indexers that perform critical security analyses.

We note that the tea Association directly acknowledges this potential for spam and abuse, stating that they are actively tuning defensive mechanisms aimed at preventing spammers from illegitimately claiming rewards [27]. However, regardless of whether the tea protocol's reward mechanism effectively excludes spammers (and regardless of whether the tea protocol is even adopted at large) we emphasize that open-source package ecosystems–and the software supply chains they enable–are still adversely impacted by any package spam resulting from the tea protocol.

In this work, we seek to critically explore the repository-wide impacts of the tea protocol [28], through a detailed quantitative analysis spanning the entire npm repository–the largest open-source package ecosystem currently supported by the tea protocol [21]. However, realizing this research goal poses two key challenges. First, is the massive scale of npm, which as of March 2025, hosts more than three million unique packages and thus requires substantial resources and effort to analyze at scale. The second challenge is that legitimate packages need to be distinguished from package spam that is intended to manipulate the tea protocol. As such, careful data analysis and targeted classification techniques are needed to fully understand the repository-wide impacts of the tea protocol.

We overcome these challenges through large-scale data analysis over a synchronous mirror of npm that we have actively maintained for more than five years–archiving packages in real time as they are uploaded to the repository. Not only does this mirror enable us to identify how the tea protocol has impacted the npm ecosystem over time, it also allows us to catalog historical package takedowns to assess whether repository maintainers are taking action against any tea-related spam and abuse.

Utilizing our mirror of npm, we conduct a detailed analysis of more than 3.8 million unique packages. This analysis uncovered **207,101** packages that appear to be illegitimately manipulating the tea protocol, of which **93.8%** are still live on the official npm package registry and costing npm hosting resources. However, we also record **328,447** seemingly legitimate packages that either directly registered to claim TEA tokens, or indirectly influence the tea protocol by depending on one or more tea-registered packages. Notably, we find only **7** highly popular packages[1] that are directly registered to claim TEA tokens; however, the recursive dependent trees of these 7 packages account for the overwhelming majority of seemingly legitimate tea-related packages–highlighting the immensely interdependent nature of npm.

From our data, we construct a timeline of the tea protocol's influence across npm, and we explore characteristic features for differentiating legitimate packages from spam aimed at manipulating the tea protocol.

Overall, the contributions of our work are as follows:

- We conduct a measurement study of the tea protocol's influence across the entire npm repository.
- We demonstrate metrics and heuristics that effectively distinguish tea-related spam and abuse from legitimate packages.
- We derive a timeline of releases and takedowns for packages designed to artificially manipulate the tea protocol.

2 Background

This section provides key details that are essential to understanding the tea protocol [28] and how it interacts with the npm repository.

2.1 The tea Protocol

Here, we describe the basic functionality of the tea protocol. To register a package with tea and thus make it eligible to collect TEA tokens, the package must first be registered on the official tea web application [27]. This registration process requires one or more tea accounts, connected to associated GitHub account(s) that own the package being registered[2]. Upon successful registration, a *tea.yaml* file is generated, which must then be added into the package's file tree–and validated on the official tea web application–before TEA tokens may start being awarded to the package. As such, packages that are eligible to generate TEA tokens are able to be identified by the presence of a *tea.yaml* file.

For registered packages, TEA tokens are distributed on a daily basis, proportional to a custom metric referred to as 'teaRank' [11]. Although the precise implementation details of the teaRank algorithm are not publicly known, the tea Association has stated that it utilizes a logarithmic scale and is largely based on a package's number of dependents [27]. Effectively, this means that every package that (transitively) depends on one or more registered tea packages increases its teaRank and thus indirectly influences the tea protocol.

[1]defined as garnering at least 100,000 weekly downloads

[2]Although package ownership on npm is controlled by an *npm* account, packages may be linked to an associated GitHub repository via a field in their *package.json* metadata file, which the tea protocol utilizes determine GitHub ownership for npm packages.

2.2 Spam Resistance

To defend against spam and abuse, the tea Association states that they have implemented and are actively refining several spam-resistance mechanisms, which are included in the teaRank algorithm [11]. Once again, although precise implementation details are not made public, the official tea documentation provides some high-level information regarding the function of these defensive mechanisms [27]. The first mechanism relates to *self-influence attacks*, wherein the owner of a high teaRank package creates superfluous dependencies for their package in order to artificially inflate the teaRank of those dependencies. To address self-influence attacks, teaRank employs a parameter, κ, which encapsulates the percentage of total influence that can be transferred from a package to its dependencies, and then distributes that fraction of total influence evenly across its dependencies.

The next mechanism concerns *tree attacks*, wherein a spammer publishes large quantities of packages that transitively depend on a tea-registered package in order to illegitimately boost its teaRank. To combat tree attacks, the tea Association records and monitors changes to the dependent tree width and depth for every tea-registered package on a parameterized time period, represented as δ [27]. However, it is unclear the extent to which a dependent tree change is required to trigger this mechanism, nor the effectiveness of this mechanism against an adaptive adversary who controls their flow of package releases to emulate organic growth. Nevertheless, the tea protocol, and its spam resistance measures, are still being developed and refined in anticipation of the tea protocol's full release [29].

For the aims of this work, we emphasize that tea-related spam impacts not only the tea protocol–but also the npm package ecosystem itself–as large quantities of spam packages occupy storage resources, contribute to possible package confusion [16, 26], and increase the workload of repository crawlers and indexers.

3 Overview

In this section, we outline the methodology of our repository-wide investigation into the impacts of the tea protocol [28] across npm.

3.1 Description of tea Spam

This subsection provides a qualitative description of tea-related spam that we encounter across the npm repository. We emphasize that it is impossible to perfectly infer the intent behind tea-related spam packages, as it may be the case that certain packages manipulating the tea protocol are intended as stress tests, rather than attempts at cryptocurrency exploitation. However, the focus of this work lies in measuring tea-related spam–and its impacts across the npm repository–regardless of intent.

For the purpose of our analysis, we consider tea-related spam packages to be those devoid of any novel functionality (often any functionality altogether), yet manipulate the teaRank algorithm [11] via their dependency structure. Many of these packages appear to be procedurally generated, employing common template file structures and naming conventions. Listing 1 shows an excerpt of a procedural package generation script, which we found residing within a tea-related spam package identified by our analysis.

```
function publishWithDelay() {
  ...
  let pkgJson = fs.readFileSync("package.json");
  let pkgData = JSON.parse(pkgJson);
  let randomFruit = uniqueNamesGenerator({
    dictionaries: [adjectives, animals, colors],
    length: 2,
  });

  pkgData.name = `${randomFruit}_z3n`;

  fs.writeFileSync("package.json",
    JSON.stringify(pkgData));
  ...
  exec("npm publish --access public")
  ...
}
```

Listing 1: Excerpt of procedural package generation script identified in tea-related spam package `beneficial_cougar_z3n`

Broadly, we find two types of tea-related spam packages. First, are the packages that directly contain a *tea.yaml* file, which are used to claim TEA tokens. These packages tend to be first created shortly after the release of the initial incentivized tea testnet in early 2024 [28]. Notably, we find these packages to have suspiciously high dependent counts relative to their weekly download counts, often accruing thousands of dependents despite having weekly download counts below 350, which is the npm maintainers' upper-bound estimate of weekly downloads that can be attributed to bots and crawlers alone [17].

The second category of tea-related spam packages are those that depend on one or more of the packages that directly contain a *tea.yaml* file. These packages often list dozens of package dependencies, which are typically composed of a mixture of highly popular npm packages and other tea-related spam packages. Some of these packages also have additional tea spam dependents or are interdependent with other tea spam packages. We also find that tea-related spam dependent trees are often distributed across numerous distinct npm accounts (which we further discuss in Section 4.3), likely functioning as Sybil accounts for the authors of the packages that are registered to claim TEA tokens. Such features demonstrate deliberate attempts to manipulate the teaRank algorithm by artificially inflating package dependent trees.

3.2 Research Questions

Based on our initial observations on the prevalence of tea-related spam packages with no legitimate functionality, we initiate a repository-wide measurement study focused on the following research questions:

- RQ1: How many npm packages interact with the tea protocol, either directly or indirectly? To what extent are these packages legitimate, or tea-related spam and abuse?
- RQ2: What are the key characteristics of tea-related spam, and how do these packages differ from legitimate ones?
- RQ3: Are communities of spammers and/or Sybil accounts responsible for tea-related spam and abuse?

4 Results

This section presents the findings of our repository-wide investigation into the impacts of the tea protocol [28] across npm, as directed by our research questions. First, Section 4.1 explores the overall scope of npm packages interacting with the tea protocol and demonstrates techniques for distinguishing legitimate packages from spam and abuse. Then, in Section 4.2, we investigate key characteristics of packages identified as likely tea spam and contrast them against those identified as likely legitimate. Finally, Section 4.3 examines the authors behind packages identified as likely tea spam, from which we derive distinct communities of spammers and their Sybil accounts.

4.1 Classifying Tea Packages

In this subsection, we answer RQ1–by identifying npm packages that interact with the tea protocol [28] and demonstrating techniques for classifying these packages as either likely spam or likely legitimate.

Our analysis is based on a snapshot of npm, captured on 3/17/2025, which spans a total of 3,884,532 unique packages. Across this snapshot, we identify 50,953 packages that directly contain a *tea.yaml* file and are thus eligible to generate TEA tokens. We note that the total scope of the tea protocol across npm is much larger than just this set of packages that directly interact with it–since the tea protocol rewards packages based on the number of other packages that depend on them. To identify these dependent packages, we start from our initial set of direct *tea.yaml* packages and recursively crawl the package reverse dependency trees of our entire snapshot of npm until the complete dependent trees of all tea-incorporating packages are extracted. This process uncovered an additional 484,595 packages that eventually depend on one or more *tea.yaml* packages, bringing the total scope of the tea protocol across npm to 535,548 unique packages.

To accurately characterize the impacts of the tea protocol across npm, it is crucial to distinguish legitimate packages using tea from spammers seeking to manipulate the tea protocol for financial gain. Figure 1a depicts the distribution of release dates for every npm package in the complete tea dependent tree. The large spike of package releases in early 2024 coincides with the initial launch of the incentivized tea testnet [28] and appears to largely consist of spammers attempting to exploit the tea protocol. To separate spam and abuse from seemingly legitimate packages, we employ a heuristic-based filter that captures key defining characteristics of tea abuse. Our filtering criteria are as follows:

(1) Tea spam packages must be initially created on or after 2024. This criterion serves primarily to filter out packages that were created well before the initial release of the incentivized tea testnet and thus are overwhelmingly likely to be legitimate.
(2) Tea spam packages must accrue no more than 350 weekly downloads. This figure represents the npm maintainers'

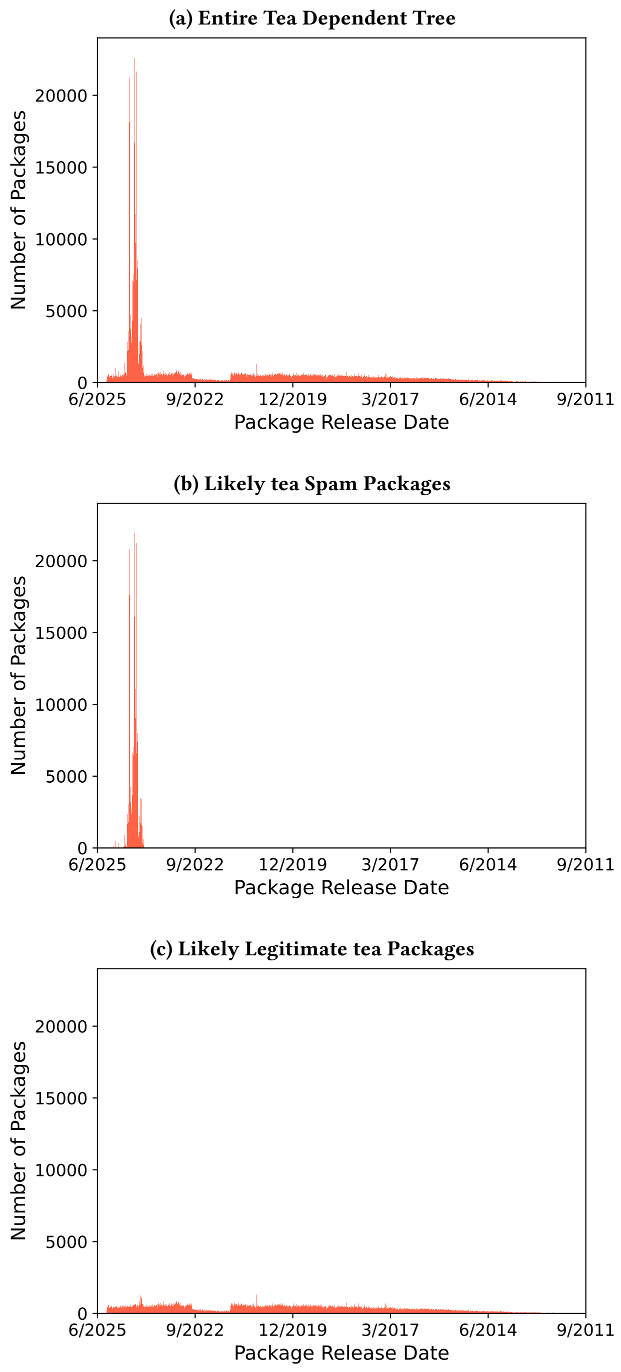

Figure 1: Histograms depicting the distribution of package creation dates for (a) all packages in the entire tea dependent tree across npm, (b) the subset of packages identified as likely tea spam, and (c) the subset of tea packages identified as likely legitimate.

upper-bound estimate of weekly downloads that can be attributed to bots and mirrors alone [17]. Hence, packages below this threshold are likely never downloaded by real users and extremely unlikely to have legitimate packages depend on them.

(3) Tea spam packages either directly contain a *tea.yaml* file or reside in the dependent trees of other tea spam packages.

To be classified as tea-related spam, a package must meet all three of the filtering criteria. We apply this filter in two passes. First, we recursively propagate our filter up the dependent trees of tea-related packages (i.e., from a package to those that depend on it), starting from the packages that directly contain a *tea.yaml* file and match our filtering criteria. From this first pass, we flag 50,251 out of 50,953 packages that directly contain a *tea.yaml* file as likely spam, and 149,106 out of 484,595 packages that eventually depend on one or more *tea.yaml* packages as likely spam.

In our second pass, we reapply our filtering criteria to all tea-dependent packages that share package authorship with any tea-related spam packages identified in the first pass, as this secondary pass helps to catch any edge cases in which dependency trees may have been stealthily manipulated (e.g., via dependency aliasing [14]). From this second pass, we flag an additional 7,744 packages that eventually depend on one or more *tea.yaml* packages as likely spam. This brings the total quantity of likely spam packages identified to 207,101 out of 535,548 tea-related packages.

To visualize this data, see Figure 1. The distribution of package creation dates for all tea-related packages is presented in Figure 1a, the distribution for likely tea spam in Figure 1b, and the distribution for likely legitimate tea-related packages in Figure 1c. We note that the relatively constant distribution of likely legitimate packages in Figure 1c is expected for typical package releases, which we obtain by removing likely tea spam packages from the distribution of all tea-related packages (Figure 1a). In other words, the tea-related spam packages identified by our filter almost entirely account for the anomalous spike of package releases observed in early 2024. Thus, we believe our filter to be largely effective in separating waves of tea-related spam from legitimate npm packages.

Summary of RQ1: We record 535,548 distinct npm packages that interact with the tea protocol, representing a notable portion of the npm repository. We estimate that 207,101 tea-related packages are spam, which includes 98.6% of all packages that are directly registered to claim TEA tokens. Despite this, a small number of high-profile packages also appear to utilize the tea protocol as intended, and these packages account for 327,745 unique dependents across npm.

4.2 Characterizing Tea Spam

This subsection, guided by RQ2, explores key characteristics of tea spam packages and contrasts them against legitimate packages, so as to provide a deeper characterization of tea spam and abuse across npm.

Dependency Trees: Dependency trees, and their overall structure, provide useful insight into how spammers attempt to exploit the tea protocol, and how this differs from typical package use. Figure 2 presents the dependency tree depth distribution for likely tea spam

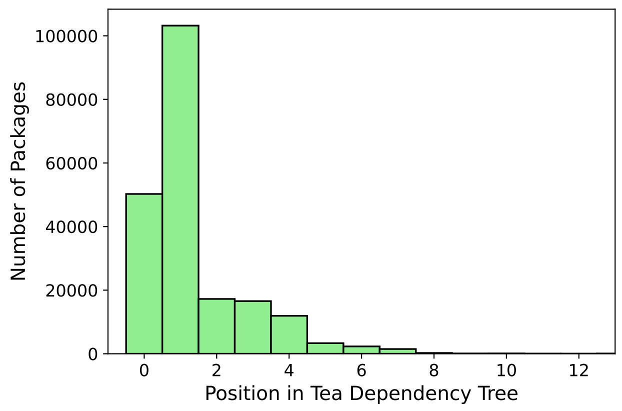

Figure 2: Histogram of dependency tree positioning for likely tea spam packages. Note: a position of zero represents a package that directly contains a *tea.yaml* file, and a position of n represents a package that is at most n dependency relationships removed from any package at position zero.

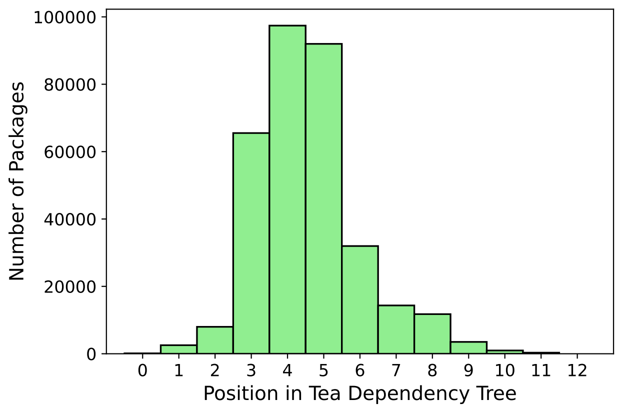

Figure 3: Histogram of dependency tree positioning for for likely legitimate tea packages. Note: a position of zero represents a package that directly contains a *tea.yaml* file, and a position of n represents a package that is at most n dependency relationships removed from any package at position zero.

packages, and Figure 3 depicts the dependency tree depth distribution for likely legitimate tea packages. For likely tea spam packages, there is a large number of packages that directly incorporate a *tea.yaml* file, and most of their dependents directly depend on them. We also observe a small number of long dependency chains (for dependent tree depths of 15 and beyond, we find at most 13 distinct packages per layer), the longest of which being a single chain of dependent packages that runs 96 dependency relationships deep.

As for likely legitimate packages, we observe a small number of popular packages that directly contain a *tea.yaml* file, with most of their dependents centered around a tree depth of 4 dependency relationships apart. Regarding the popularity of these packages, we find only 7 npm packages that contain a *tea.yaml* file and garner more than 100,000 weekly downloads (a threshold that represents the top 16,300 npm packages, which together accrue more than 99% of all package downloads across npm [26]). We note that the overall tree depth distribution of likely legitimate tea packages appears to be caused by this small number of very popular package dependencies, such as `dotenv-expand` and `brace-expansion` that reside a few layers deep in the dependency trees of other highly popular npm packages.

The characteristic differences across these distributions may serve as a useful, but not definitive signal in distinguishing legitimate packages from spam, as we find that observed spam dependent trees are tightly focused on direct dependencies, whereas observed legitimate dependency trees trend more towards slightly deeper and seemingly normal distributions.

Code Complexity: Next, we explore code complexity as a metric for characterizing tea spam versus legitimate packages–as we expect tea-related spam packages to contain lower-effort code, which should be reflected in code complexity metrics. For our analysis, we quantify code complexity using cyclomatic complexity [10], a software metric that encapsulates the total number of linearly independent paths through a program, starting at a value of one and increasing with program loops and branches.

To extract package-level cyclomatic complexity data, we utilize the `complexity-report` package to recursively analyze the directories of every package in the entire tea dependent tree and measure their total cyclomatic complexity. Figure 4 depicts the distribution of cyclomatic complexity observed in likely tea spam packages, and Figure 5 depicts the distribution of cyclomatic complexity observed in likely legitimate tea packages. We find that likely tea spam packages overwhelmingly exhibit a cyclomatic complexity of one and show little variation or deviation from this value. Meanwhile, we find that likely legitimate packages appear to exhibit a power-law-like distribution, starting at a cyclomatic complexity of one and quickly tapering off into larger values. Although the differences across these cyclomatic complexity distributions are interesting at scale (and potentially distinguishing for values greater than two, which are almost exclusively exhibited by likely legitimate packages), we find that in most instances, cyclomatic complexity is unreliable in distinguishing legitimate packages from spam, as both groups are dominated by packages with a cyclomatic complexity of one.

While the focus of our analysis is on tea-related spam, previous work notes significant quantities of near-empty, placeholder, template, and trivial packages [2, 3] across the npm repository, many of which are effectively unused. As such, the large quantity of likely legitimate packages with a cyclomatic complexity of one is an expected result of the extensive trivial packages observed in the npm ecosystem.

File Tree Uniqueness: The file tree structure of packages may also provide insights into the mass generation of spam packages. Within the set of likely tea spam packages, we find that only 2.68% have a unique file tree structure, whereas within the set of likely legitimate tea packages, we measure that 73.64% have a unique file

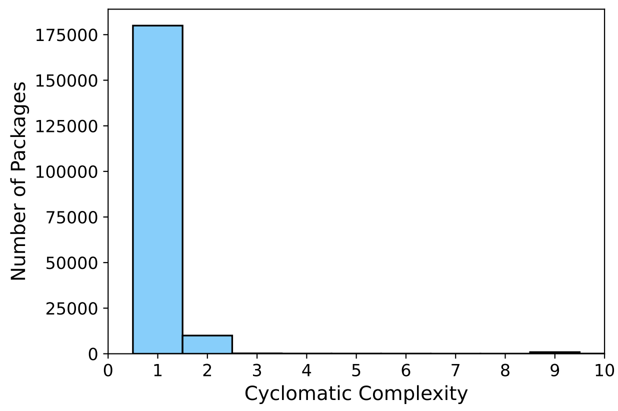

Figure 4: Distribution of package-level cyclomatic complexity for likely tea spam packages.

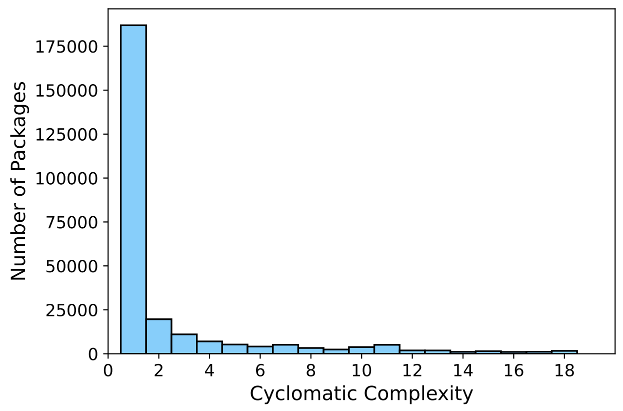

Figure 5: Distribution of package-level cyclomatic complexity for likely legitimate tea packages.

tree structure. Many of these non-unique file trees across likely legitimate packages are once again the result of trivial packages [2, 3] observed across the npm repository, which are often composed of just one or two JavaScript files spanning a handful of common naming conventions and/or an assortment of common metadata files. To further contextualize these percentages, we collect a control sample of 500,000 randomly selected packages across our snapshot of npm, and find that about half of these packages (49.00%) have a unique file tree structure. Within our control sample, we observe a lower percentage of unique file trees compared to our set of likely legitimate packages. We find this smaller percentage to be a result of various forms of package spam–as well as typical packages–both being present within the control sample, in addition to the overall larger size of the control sample increasing the chance odds of collisions occurring. For comparison, every package in the complete tea dependent tree, including both spam and legitimate packages, makes for a comparable sample size of 535,548, and we observe that only 47.21% of packages within this complete set have a unique file tree structure. Ultimately, these results demonstrate significant differences between the file tree structure of legitimate npm packages and those identified as likely tea spam, which we believe to be potentially useful in validating and/or tuning filtering criteria for identifying tea-related spam and abuse.

Summary of RQ2: Numerous characteristics distinguish tea-related spam and abuse from legitimate packages at a population level, including dependent tree structure, code complexity, and file tree uniqueness. For classifying individual packages, we find that dependent tree depth and file tree structure can provide useful but not definitive insight, whereas cyclomatic complexity is only situationally useful in identifying a small subset of legitimate packages that are highly complex.

4.3 Spammer Communities

In this subsection, we explore characteristics pertaining to the authorship of tea packages and identify communities of tea-related spammer accounts and their Sybils, as guided by RQ3.

First, we explore package author networks across our set of likely tea spam packages and our set of likely legitimate tea packages. We find a total of 5,442 unique author accounts are responsible for the 207,101 identified likely tea spam packages. Conversely, we find a total of 154,921 unique author accounts are responsible for the 328,447 identified likely legitimate tea packages. Hence, likely tea spam package authors publish, on average, nearly 18 times more packages per npm account than likely legitimate package authors. Regarding inter-group overlap, we find the sets of likely tea spam package authors and likely legitimate tea authors to be nearly bipartite, with only 30 npm accounts publishing packages marked in both categories. While this small overlap may represent some noise in our data, the overwhelmingly disjoint nature of tea spam versus legitimate package authors provides reassurance that our data is highly consistent.

Next, we investigate the presence of author communities and their Sybil accounts behind packages identified as likely tea spam. To accomplish this, we employ the well-established Louvain community detection algorithm [4], a graph-based heuristic that iteratively identifies non-overlapping communities of nodes by optimizing for maximal connectedness within communities and maximal disjointness across communities. To utilize this approach, we construct a package author relation graph over our set of likely tea spam packages, where nodes represent individual npm accounts and edges represent the total number of package-to-package dependency relationships between different accounts. Effectively, this means that employing the Louvain algorithm on this graph will identify the most strongly connected groups of likely tea spam package authors, based on whose packages depend on each other's. From this graph, the Louvain algorithm identifies 227 author communities and 637 isolated authors. Table 1.1 presents the ten largest of these communities sorted by total authors, and Table 1.2 presents the ten largest of these communities sorted by total packages. We observe significant variation across community sizes and total packages per community; however, when measured in isolation, each metric appears to follow a power-law-like distribution. Overall, our results demonstrate the presence of several large communities of tea-related spammers and Sybil accounts that are behind a significant portion of likely tea spam packages identified by our filter.

1.1 Sorted by Total Authors

Rank	Authors	Likely Tea Spam Packages
10	90	194
9	106	2,015
8	135	591
7*	137	14,981
6*	157	38,064
5	164	1,865
4	168	1,421
3	186	1,153
2*	401	15,088
1	408	817

1.2 Sorted by Total Packages

Rank	Authors	Likely Tea Spam Packages
10	5	4,523
9	6	6,649
8	10	6,790
7	4	7,224
6	6	7,225
5	41	7,509
4*	137	14,981
3*	401	15,088
2	18	16,815
1*	157	38,064

Table 1: Top ten largest communities of likely tea spam package authors, sorted by total authors in the community (Table 1.1) and total number of packages published by the community (Table 1.2). Note:* indicates a community that is present in both top tens.

Summary of RQ3: Communities of spammers and Sybil accounts are responsible for a large majority of identified tea-related spam and abuse, with 88.3% of such packages belonging to a community of npm accounts as found by the Louvain community detection algorithm.

5 Discussion

In this section, we discuss auxiliary findings and implications of this work.

5.1 Adoption of tea

Here, we discuss the current state of the npm repository, with respect to the adoption of the tea protocol [28]. In its current testnet stage, we find that only 0.04% of packages that garner more than 100,000 weekly downloads are directly registered with the tea protocol. These high-impact packages represent notable targets for tea to attract, as they simultaneously are capable of generating the most rewards and encouraging legitimate use of the tea protocol. Despite the current number of highly popular packages adopting tea being small, the transitive dependents of these packages are wide-reaching and span hundreds of thousands of packages indirectly influencing the tea protocol and demonstrating its effectiveness in benign settings.

Meanwhile, seemingly illegitimate packages account for 98.6% of all npm packages that are directly registered with the tea protocol. Although the contrast between legitimate and illegitimate adoption may seem alarming, we emphasize that TEA is still in a testnet stage, and the data gathered from this testnet is extremely valuable for resolving issues prior to a full launch.

In spite of this, we highlight that independent of the tea protocol itself, tea-related spam packages still pose concerns regarding the overall health of the npm ecosystem, as they occupy storage capacity, contribute to possible package confusion [16, 26], and increase the bandwidth of crawlers and indexers scanning the repository.

5.2 Spam Resistance

In this subsection, we discuss potential mechanisms for hardening against spam, both within protocols such as tea and across open-source package ecosystems. In its current form, dependent trees alone do not provide enough information to accurately flag all forms of spam and abuse–particularly considering adaptive adversaries that are capable of manipulating package publication traffic to emulate natural dependent growth.

Rather, we recommend that a wider range of metrics be employed to flag potential spam and abuse. Download counts may serve as a simple sanity check when compared against transitive dependent counts; however, we note that instances of download count manipulation have also been observed in the npm repository [17]. Despite this, adversarial emulation of natural growth is made significantly more difficult as more and more metrics are monitored. Further, associated GitHub repository metrics, such as stars, watchers, and forks [5] may additionally serve as cross-references for indicating real use, considering that the tea protocol requires an associated GitHub repository to register a package in the first place.

Outside of popularity metrics, package code itself may assist in identifying non-meaningful projects. Empty, template, and/or procedurally generated packages that are registered to claim TEA tokens could be caught and flagged at scale–perhaps via existing code clone detection [33] or spam detection [23] techniques.

Moreover, we recommend that protocol designers, such as tea, work directly with open-source ecosystem maintainers to report flagged spam and mitigate negative externalities on package repositories. Expanding upon existing package metrics, protocol designers would have access to additional information regarding their userbase, linked repository accounts, and their registered packages. Such data could be highly valuable, both in identifying spammers and in propagating spam reports to relevant repository authorities.

In total, different spam resistance approaches constrain adversarial behavior in different ways, and we ultimately recommend that an ensemble of metrics and detection strategies be employed to flag tea-related spam and abuse–both for securing protocols like tea–and keeping open-source package ecosystems clean from spam.

6 Related Work

This section explores related works and contextualizes our findings alongside existing research.

6.1 Characterizing Package Ecosystems

Investigating the characteristics and impacts of open-source package ecosystems forms a large and active body of research [3, 7, 15, 32, 33, 36, 37]. Notable features identified by these works include extensive interdependence between packages [13] and package popularity distributions that are heavily skewed towards a tiny fraction of packages that account for the overwhelming majority of all package downloads [15, 26, 37]. Both of these features pose implications for the tea protocol [28], as the dependent trees of highly adopted packages exhibit exponential growth; moreover, popularity metrics may assist in augmenting existing data sources to identify spam and abuse.

Other works in this domain detail the existence of repository-wide phenomena, such as the widespread presence and adoption of trivial packages [2, 3], stealthily-forked packages that obscure provenance information [33], or discrepancies between code published to package ecosystems versus their associated GitHub repositories [31].

Our work is unique in that it, to the best of our knowledge, is the first study within academic literature to characterize and measure repository-wide impacts of the tea protocol [28]. Further, we believe that tea-related spam and abuse are a prominent, yet critically under-studied, threat to the health and authenticity of open-source package ecosystems.

6.2 Software Supply Chain Security

Also related to this work is the field of software supply chain security, which explores the security implications of vulnerable and/or malicious software dependencies [20]. Works in this domain often operate from a software development perspective–focusing on the identification of key attack vectors [6, 8, 16, 22, 30, 38]and/or proposing tools to solve or mitigate package security issues [9, 12, 13, 18, 24, 25, 34, 35].

Although we do not observe directly vulnerable or malicious code being distributed by tea-related spam packages, abuse of the tea protocol [28] represents a software supply chain-related financial security issue, which may constitute cryptocurrency fraud. As such, this work provides a unique contribution to the greater understanding of open-source software supply chains and their widespread security implications.

7 Conclusion

This work explored the impacts of the tea protocol–and the experimental cryptocurrency it distributes based on package contributions [28]–across the entire npm repository. We found that even a small number of highly utilized packages adopting the tea protocol expands its influence to hundreds of thousands of dependent packages. Further, we characterized spam and abuse of the tea protocol across npm, uncovering more than 200,000 packages that likely serve no functional utility, yet illegitimately manipulate the reward mechanism of the tea protocol. It is our hope that this work raises broader awareness and efforts to combat spam and abuse across open-source package ecosystems as the tea protocol nears its full launch.

References

[1] 2022. OpenSSF Scorecard. https://github.com/ossf/scorecard.

[2] Rabe Abdalkareem, Olivier Nourry, Sultan Wehaibi, Suhaib Mujahid, and Emad Shihab. 2017. Why Do Developers Use Trivial Packages? An Empirical Case Study on Npm. In *ESEC/FSE 2017* (Paderborn, Germany). Association for Computing Machinery, New York, NY, USA. doi:10.1145/3106237.3106267

[3] Rabe Abdalkareem, Vinicius Oda, Suhaib Mujahid, and Emad Shihab. 2020. On the impact of using trivial packages: an empirical case study on npm and PyPI. *Empirical Software Engineering* 25 (03 2020). doi:10.1007/s10664-019-09792-9

[4] Vincent D Blondel, Jean-Loup Guillaume, Renaud Lambiotte, and Etienne Lefebvre. 2008. Fast unfolding of communities in large networks. *Journal of Statistical Mechanics: Theory and Experiment* 2008, 10 (Oct. 2008), P10008. doi:10.1088/1742-5468/2008/10/p10008

[5] Hudson Borges and Marco Túlio Valente. 2018. What's in a GitHub Star? Understanding Repository Starring Practices in a Social Coding Platform. *CoRR* abs/1811.07643 (2018). arXiv:1811.07643 http://arxiv.org/abs/1811.07643

[6] Kyriakos Chatzidimitriou, Michail Papamichail, Themistoklis Diamantopoulos, Michail Tsapanos, and Andreas Symeonidis. 2018. Npm-miner: An infrastructure for measuring the quality of the npm registry. In *MSR 2018*. IEEE, 42–45.

[7] Tapajit Dey and Audris Mockus. 2020. Deriving a usage-independent software quality metric. *ESE* 25 (2020). doi:10.1007/s10664-019-09791-w

[8] Ruian Duan, Omar Alrawi, Ranjita Pai Kasturi, Ryan Elder, Brendan Saltaformaggio, and Wenke Lee. 2021. Towards Measuring Supply Chain Attacks on Package Managers for Interpreted Languages. In *NDSS 2021*. Internet Society.

[9] Aurore Fass, Michael Backes, and Ben Stock. 2019. JStap: a static pre-filter for malicious JavaScript detection. In *Proceedings of the 35th Annual Computer Security Applications Conference* (San Juan, Puerto Rico, USA) *(ACSAC '19)*. Association for Computing Machinery, New York, NY, USA, 257–269. doi:10.1145/3359789.3359813

[10] William Fetzner. 2021. What Is Code Complexity? What It Means and How to Measure It. https://linearb.io/blog/what-is-code-complexity/

[11] Max Howell and Timothy Lewis. 2025. A Decentralized Protocol for Open-Source Developers to Capture the Value They Create. https://docs.tea.xyz/tea-white-paper/white-paper

[12] Cheng Huang, Nannan Wang, Ziyan Wang, Siqi Sun, Lingzi Li, Junren Chen, Qianchong Zhao, Jiaxuan Han, Zhen Yang, and Lei Shi. 2024. DONAPI: Malicious NPM Packages Detector using Behavior Sequence Knowledge Mapping. *arXiv preprint arXiv:2403.08334* (2024).

[13] Igibek Koishybayev and Alexandros Kapravelos. 2020. Mininode: Reducing the Attack Surface of Node.js Applications. In *RAID 2020*. USENIX Association. https://www.usenix.org/conference/raid2020/presentation/koishybayev

[14] Mike McCready, Kyle Mitchell, Santoshraj2, Mottle, Hong Xu, s100, Uiolee, Christian Oliff, , Daniel Kaplan, Jan Sott, Gar, Francesco Sardone, P-Chan, Davide, Darryl Tec, Rohan Mukherjee, and Luke Karrys. 2024. package.json | npm Docs. https://docs.npmjs.com/cli/v10/configuring-npm/package-json

[15] Suhaib Mujahid, Rabe Abdalkareem, and Emad Shihab. 2022. What are the characteristics of highly-selected packages? A case study on the npm ecosystem. doi:10.48550/ARXIV.2204.04562

[16] Shradha Neupane, Grant Holmes, Elizabeth Wyss, Drew Davidson, and Lorenzo De Carli. 2023. Beyond Typosquatting: An In-depth Look at Package Confusion. In *32nd USENIX Security Symposium (USENIX Security 23)*. USENIX Association, Anaheim, CA, 3439–3456. https://www.usenix.org/conference/usenixsecurity23/presentation/neupane

[17] npmjs.org. 2014. numeric precision matters: how npm download counts work (accessed 02/2021). https://blog.npmjs.org/post/92574016600/numeric-precision-matters-how-npm-download-counts-work.

[18] Marc Ohm, Felix Boes, Christian Bungartz, and Michael Meier. 2022. On the Feasibility of Supervised Machine Learning for the Detection of Malicious Software Packages. In *Proceedings of the 17th International Conference on Availability, Reliability and Security* (Vienna, Austria) *(ARES '22)*. Association for Computing Machinery, New York, NY, USA, Article 127, 10 pages. doi:10.1145/3538969.3544415

[19] Marc Ohm and Charlene Stuke. 2023. SoK: Practical Detection of Software Supply Chain Attacks. In *Proceedings of the 18th International Conference on Availability, Reliability and Security* (Benevento, Italy) *(ARES '23)*. Association for Computing Machinery, New York, NY, USA, Article 33, 11 pages. doi:10.1145/3600160.3600162

[20] Chinenye Okafor, Taylor R. Schorlemmer, Santiago Torres-Arias, and James C. Davis. 2022. SoK: Analysis of Software Supply Chain Security by Establishing Secure Design Properties. In *SCORED*. doi:10.1145/3560835.3564556

[21] OpenJS Foundation. 2024. npm. https://www.npmjs.com/.

[22] Brian Pfretzschner and Lotfi ben Othmane. 2017. Identification of Dependency-based Attacks on Node.Js. In *ARES*.

[23] Mohammed Rasol Al Saidat, Suleiman Y. Yerima, and Khaled Shaalan. 2024. Advancements of SMS Spam Detection: A Comprehensive Survey of NLP and ML Techniques. *Procedia Computer Science* 244 (2024), 248–259. doi:10.1016/j.procs.2024.10.198 6th International Conference on AI in Computational Linguistics.

[24] Adriana Sejfia and Max Schäfer. 2022. Practical automated detection of malicious npm packages. In *Proceedings of the 44th International Conference on Software*

Engineering (Pittsburgh, Pennsylvania) *(ICSE '22)*. Association for Computing Machinery, New York, NY, USA, 1681–1692. doi:10.1145/3510003.3510104

[25] Raphael J. Sofaer, Yaniv David, Mingqing Kang, Jianjia Yu, Yinzhi Cao, Junfeng Yang, and Jason Nieh. 2024. RogueOne: Detecting Rogue Updates via Differential Data-flow Analysis Using Trust Domains. In *Proceedings of the IEEE/ACM 46th International Conference on Software Engineering* (Lisbon, Portugal) *(ICSE '24)*. Association for Computing Machinery, New York, NY, USA, Article 101, 13 pages. doi:10.1145/3597503.3639199

[26] Matthew Taylor, Ruturaj Vaidya, Drew Davidson, Lorenzo De Carli, and Vaibhav Rastogi. 2020. Defending Against Package Typosquatting. In *NSS 2020*.

[27] tea Association. 2024. Docs. https://docs.tea.xyz/tea

[28] tea Association. 2025. tea. https://tea.xyz/.

[29] tea Association. 2025. Unlock the Value of Open Source: Join the Tea Revolution. https://tea-xyz.webflow.io/sepolia

[30] Ruturaj K. Vaidya, Lorenzo De Carli, Drew Davidson, and Vaibhav Rastogi. 2019. Security Issues in Language-based Sofware Ecosystems. *CoRR* abs/1903.02613 (2019). arXiv:1903.02613 http://arxiv.org/abs/1903.02613

[31] Duc-Ly Vu, Fabio Massacci, Ivan Pashchenko, Henrik Plate, and Antonino Sabetta. 2021. LastPyMile: Identifying the Discrepancy between Sources and Packages. In *ESEC/FSE 2021*. doi:10.1145/3468264.3468592

[32] Erik Wittern, Philippe Suter, and Shriram Rajagopalan. 2016. A look at the dynamics of the JavaScript package ecosystem. In *MSR*.

[33] Elizabeth Wyss, Lorenzo De Carli, and Drew Davidson. 2022. What the Fork? Finding Hidden Code Clones in npm. In *ICSE 2022*. doi:10.1145/3510003.3510168

[34] Elizabeth Wyss, Alexander Wittman, Drew Davidson, and Lorenzo De Carli. 2022. Wolf at the Door: Preventing Install-Time Attacks in Npm with Latch. In *ASIA CCS '22*. doi:10.1145/3488932.3523262

[35] Nusrat Zahan, Thomas Zimmermann, Patrice Godefroid, Brendan Murphy, Chandra Maddila, and Laurie Williams. 2022. What are Weak Links in the npm Supply Chain?. In *ICSE-SEIP 2022*. doi:10.1145/3510457.3513044

[36] Ahmed Zerouali, Eleni Constantinou, Tom Mens, Gregorio Robles, and Jesus Gonzalez-Barahona. 2018. An Empirical Analysis of Technical Lag in npm Package Dependencies. doi:10.1007/978-3-319-90421-4_6

[37] Ahmed Zerouali, Tom Mens, Gregorio Robles, and Jesus M. Gonzalez-Barahona. 2019. On the Diversity of Software Package Popularity Metrics: An Empirical Study of npm. In *SANER 2019*. doi:10.1109/SANER.2019.8667997

[38] Markus Zimmermann, Cristian-Alexandru Staicu, Cam Tenny, and Michael Pradel. 2019. Small world with high risks: A study of security threats in the npm ecosystem. In *USENIX Security 19*.