Investigating Package Related Security Threats in Software Registries

Yacong Gu[*], Lingyun Ying[*][✉], Yingyuan Pu[*][†], Xiao Hu[*],
Huajun Chai[*], Ruimin Wang[*][‡], Xing Gao[§], Haixin Duan[¶][‖]
[*]QI-ANXIN Technology Research Institute, [†]Ocean University of China, [‡]Southeast University,
[§]University of Delaware, [¶]Tsinghua University, [‖]Tsinghua University-QI-ANXIN Group JCNS
[*]{guyacong, yinglingyun, puyingyuan01, huxiao, chaihuajun, wangruimin}@qianxin.com,
[§]xgao@udel.edu, [¶][‖]duanhx@tsinghua.edu.cn

Abstract—**Package registries host reusable code assets, allowing developers to share and reuse packages easily, thus accelerating the software development process. Current software registry ecosystems involve multiple independent stakeholders for package management. Unfortunately, abnormal behavior and information inconsistency inevitably exist, enabling adversaries to conduct malicious activities with minimal effort covertly. In this paper, we investigate potential security vulnerabilities in six popular software registry ecosystems. Through a systematic analysis of the official registries, corresponding registry mirrors and registry clients, we identify twelve potential attack vectors, with six of them disclosed for the first time, that can be exploited to distribute malicious code stealthily. Based on these security issues, we build an analysis framework, *RScouter*, to continuously monitor and uncover vulnerabilities in registry ecosystems. We then utilize *RScouter* to conduct a measurement study spanning one year over six registries and seventeen popular mirrors, scrutinizing over 4 million packages across 53 million package versions. Our quantitative analysis demonstrates that multiple threats exist in every ecosystem, and some have been exploited by attackers. We have duly reported the identified vulnerabilities to related stakeholders and received positive responses.**

1. Introduction

Software registries play an essential role in the open source software supply chain. By providing repositories for maintaining software packages, registries enable developers to share rich libraries and add-on packages, and thus accelerate the entire software development process. Millions of packages are actively maintained in software registries for almost all popular programming languages. In 2021, the top four software registry ecosystems (i.e., Maven [1], npm [2], PyPI [3], NuGet [4]) contained a combined 37 million different versions of packages, attracting more than 2.2 trillion downloading requests [5].

Meanwhile, the number of attacks against the software registry ecosystem has also substantially increased by 650% in 2021 [5]. The popularity of software registries inevitably attracts unwanted attention from adversaries, as serious security threats can be posed with compromised packages. For example, the *ua-parser-js* package [6], which attracts millions of weekly downloads in npm, was hijacked to install a cryptocurrency miner and harvest credential information, posing significant security threats to developers and end-users. Unfortunately, the software registry ecosystem involves multiple independent stakeholders for package management, and one insecure stakeholder can potentially affect the whole architecture. The desynchronized data and inconsistent information among different stakeholders might allow attackers to conduct malicious activities with minimal effort covertly.

In this paper, we systematically study potential security threats to package management in the software registry ecosystem. We primarily focus on vulnerabilities that can be exploited without compromising any parties, including registries, registry mirrors, code hosting platforms, and software build pipelines. We analyze multiple stakeholders in the ecosystem following the lifecycle of a package, from the publishing and maintenance (e.g., upgrade, unpublishing) process, to the distribution of a package. In total, we identify twelve potential attack vectors that can lead to severe consequences, such as covertly distributing malicious packages or stealthily injecting malicious code. Particularly, attackers can exploit (1) *reused resources*, including package names, registered accounts, and code hosting platform locations; (2) *inconsistency* between the official registry and its mirrors; and (3) *confused resources*, such as case sensitivity, package version, and dependency, to mount multiple attacks. We also study potential vulnerabilities related to *typosquatting*.

Our work greatly complements previous research efforts on understanding security threats in software registries [7] [8] [9]. To the best of our knowledge, we reveal six of these potential attack vectors for the first time. We have also conducted an in-depth study on registry mirrors. In particular, we identify several new approaches for attackers to hijack packages in both upstream registries and registry mirrors. For example, we find that code hosting platforms (e.g., GitHub) automatically redirect renamed users [10] and transferred projects [11] from an old repository location to a new location. While both links can be used to download the package, attackers may preempt the previous repository location, and GitHub will immediately cut off the redirection. If the repository location is not updated in registries, users will download the malicious package (denoted as *Package*

[✉]Lingyun Ying is the corresponding author.

Redirection Hijacking Attack). Some registry mirrors may fail to properly handle letter cases, such as *Buffer* and *buffer*. Users will always get the same package no matter what case is used in the request. Attackers can simply publish a malicious package with the same name but different cases as the victim package and then potentially hijack all requests (named as *Case Sensitivity Confusion Attack*). Meanwhile, we report new findings (e.g., new exploitation approaches) and results for other known attacks. For example, in addition to hijacking accounts in registries via expired domain names [8], we discover that a similar attack can be mounted by exploiting deleted third-party website (e.g., GitHub) accounts. We also find that mirror synchronization failures could amplify the damage of package reference attacks [7].

To investigate the existence of the above threats and understand the status quo, we conduct a large-scale measurement study on six popular software registry ecosystems, including Maven [1], PyPI [3], npm [2], NuGet [4], Cargo [12], and Go [13]. We develop *RScouter* to carefully collect various information (e.g., the daily published and unpublished packages, package metadata such as account and version) from both official registries and the corresponding registry mirrors, and identify vulnerabilities using differential analysis. Specifically, *RScouter* combines multiple approaches to collect maintainer accounts from different registries, such as extracting accounts from the source code repository, contact information, and their web portal. During our measurement spanning one year over six registries and seventeen popular mirrors, we have collected over 4 million unique packages across more than 53 million distinct package versions.

We find that 16,807 maintainer accounts and 40,327 packages can be immediately taken over in both centralized registries and de-centralized registries (e.g., Go). In particular, 12,121 packages in *Go* are threatened by the newly disclosed *Package Redirection Hijacking Attack*. 85,192 and 38,496 packages in *PyPI* and *npm*, respectively, are vulnerable to the *Package Use-After-Free Attack*. We also find that many widely-used registry mirrors have resource confusion and synchronization problems, allowing attackers to hijack packages. For example, the *Aliyun npm mirror* [14], which serves more than 2 billion downloads per month, suffers from the *Case Sensitivity Confusion Attack*: 2,038 packages can be hijacked immediately, and some popular victim packages attract more than 1 million downloads in one month. Furthermore, we discover eight *real Package Reference Attacks* against PyPI, where malicious packages are referenced in popular packages, affecting many mirror users. Finally, we find that a large number of malicious packages (e.g., officially marked by npm) have similar package names to existing benign packages, with the Damerau-Levenshtein distance one, indicating that attackers have heavily abused *typosquatting*.

We have disclosed our results to all affected registries and mirrors and received positive feedback. The npm, PyPI, and Go teams have confirmed multiple issues that we reported, and the npm team has awarded us with a $2,000 bug bounty. Aliyun and Tsinghua mirror maintainers have confirmed the reported vulnerabilities and taken action to fix these issues by retrofitting their repository mirroring software. We have been working together with them to fix/remove vulnerable packages and minimize the impact on both users and package maintainers in the software registry ecosystem.

In summary, the major contributions of this work include:

- We perform a systematic study on package related security threats in software registry ecosystems, including upstream registries and their mirrors. We disclose six new potential threats and report new findings/results for six other attacks.
- We developed the *RScouter* tool and conducted a large-scale measurement study on the six popular software registries and seventeen popular mirrors. Our measurement study covers more than 4 million unique packages.
- Our measurement results indicate that many registries are indeed threatened by our disclosed attacks, and tens of thousands of packages are at the risk of being hijacked.

2. Background

2.1. Primary Stakeholders

Registries. Registries host packages and provide users with services such as searching and downloading. Most existing registries are centralized, such as Maven Central for Java, PyPI for Python, npm for JavaScript, Cargo for Rust, and NuGet for .NET. Centralized registries typically maintain a website so that package maintainers can register a unique account (denoted as a *maintainer account*) to publish and manage packages. According to the way of registration, maintainer accounts can be divided into two categories: email and a third-party account sign-in (e.g., a GitHub account). Centralized registries also store package files and metadata, such as description, maintainer information, and code link.

By contrast, decentralized registries, such as Go, do not maintain a centralized website for package management. Package maintainers typically use code hosting platforms like GitHub to manage packages, and they only need to publish the information of their packages to the registry index. For instance, Go officially maintains an index website (i.e., *pkg.go.dev*) for users to search and view the Go packages. Meanwhile, Go also operates an official centralized mirror caching its package files.

Generally, a tuple ⟨repository, package name, version, hash value⟩ uniquely identifies a package. The repository is the code hosting location of the package. The package name can be customized by the package maintainer as long as it follows the naming standards and does not conflict with the existing packages in the registry. The version should be unique for the package's each publishing, and the hash value is calculated according to the package's content.

Registry Mirrors. Registry mirrors provide a data mirroring service for registries, aiming at network acceleration or bypassing Internet censorship. Usually, a mirror contains the same content as the upstream registry, and multiple mirrors might be separately maintained by different organizations. Typically, mirror maintainers should not directly add or remove packages on mirrors but only synchronize with the upstream registry through tools developed by themselves or off-the-shelf repository management tools such as *Nexus* [15] and *Artifactory* [16]. Almost all popular registries have mirror sites, which are used worldwide. For example, the Aliyun npm mirror has more than 2 billion monthly downloads [14].

Registry Clients. Finally, package maintainers use registry clients (provided by the corresponding registries or third parties) for developing, maintaining, and managing packages, and package users also use registry clients for searching, downloading and installing packages. In particular, for compiled programming languages (e.g., Java), registry clients pull dependent packages at the package compiling time, and all dependent package code is compiled into object binaries. While for interpreted programming languages (e.g., Python), dependent packages are pulled at the package installing time, and all dependent package code is saved as separate files.

2.2. Package Lifecycle

Generally, a package maintainer first registers an account on the registry's web portal. Then she/he uses this account as a credential to conduct all operations (e.g., publish, update, unpublish). Different registries have different *unpublish* policies. Some registries (e.g., PyPI, npm) allow maintainers to delete a package completely so users can no longer use the package. The second policy is to unlist a package (e.g., Cargo, NuGet), i.e., remove the package from the package index so that users cannot find it easily. However, unlisted packages can still be downloaded and installed by using an exact version number. Other registries (e.g., Maven Central) prohibit maintainers from unpublishing a package. For decentralized registries, package maintainers use their code hosting platform accounts as the credential and publish packages on the code hosting platform.

Registry mirrors then synchronize package data from the upstream registry, including the acquisition of newly published packages and synchronization of unpublished packages. Some mirrors run in the full synchronization mode and regularly synchronize the changes of all packages by parsing the registry index file. Users might be allowed to force synchronization to obtain the latest version of a package. Others operate in the proxy mode, where packages will be cached upon user requests.

Package users (usually software developers) integrate packages into their software using a *registry client*, which resolves and installs the target package and its dependencies from registries or mirrors, based on its configuration. If the package version is not specified, registry clients may adopt different strategies to choose the most suitable version. For example, NuGet 2.8x always uses the latest version, while NuGet 3.x gets the oldest eligible version by default [17].

2.3. Overview of Registry Abuse

Security breaches have become prevalent in software registries in recent years. Thousands of malicious packages such as ransomware [18] and cryptojacking [19] have been removed from different types of registries [2] [3]. Among those attacks, typosquatting has been widely used to trick users into downloading and installing malicious packages [7]. Attackers exploit typographical mistakes made by victims by publishing packages with similar names to the legitimate package. This also includes squatting popular package names across multiple registries [7], where attackers intentionally upload packages with the same names as popular packages on other platforms.

Package hijacking is another popular attack: attackers attempt to take over existing packages via various methods, including social engineering and exploiting weak passwords [9]. *Resource reuse*, which is a common scenario on today's Internet, has been widely exploited to hijack packages. For example, emails might be recycled by email providers and thus reused by other users [20]. Private email domain names might be expired and be purchased by others [21]. Similarly, package and account names might also be reused by other users. Attackers can then utilize legitimate operations such as password recovery via email to control the account and all associate packages [8], [22]. Those issues have been raised several times before [23] [24] and received some attention from the press [25] [26].

Finally, package distribution systems can also be abused by attackers as they can publish a malicious package with the same name as the private package to the official registry to launch a dependency confusion attack [27] [28]. Attackers can compromise the registry web service to manipulate existing package binaries [29] [30]. They can tamper with package contents through man-in-the-middle (MITM) attacks when victims download packages through insecure channels [31].

3. Threats on Software Registries

This work aims to highlight the existence of many potential vulnerabilities in the current software registry ecosystem, which can be exploited to distribute malicious code to users stealthily. Different from previous research efforts, we expand the scope of *resource reuse* to include third-party website accounts and disclose several new hijacking attacks. Also, we present an in-depth study on registry mirrors and find that *inconsistency* might exist between registries and their mirrors, causing security threats such as package overriding. Finally, existing software registries and mirrors might confuse about particular resources, such as case sensitivity, package version, and dependency, enabling attackers to hijack packages or stealthily distribute malicious code.

The exploitation can be considered in two types: (1) to bait users into obtaining software packages maintained by adversaries, and (2) to introduce malicious code or vulnerabilities to users in a stealthy manner. We consider the scenario that all attacks can be mounted without compromising any package management stakeholders (e.g., hacking their software systems). To exploit the vulnerabilities described in this paper, adversaries can continuously monitor key events (e.g., package unpublishing, account deletion, and synchronization failure) in registries, mirrors, and third-party code hosting platforms. All these events can be collected from public sources by many approaches, such as parsing the package index file provided by the registry, obtaining through web APIs, and crawling. For instance, the Maven Central registry provides a complete index file, including

TABLE 1. Overview of potential threats on each registry ecosystem. The columns highlighted in gray color indicate novel attacks. The ✓ means vulnerable.

Registry Ecosystem	U_1	U_2	U_3	M_1	M_2	T_1	R_1	R_2	C_1	C_2	O_1	O_2
Maven		✓		✓		✓		✓	✓*			✓
PyPI	✓	✓		✓		✓	✓	✓	✓	✓		✓
npm	✓	✓		✓	✓	✓	✓	✓			✓	✓
Cargo		✓				✓						
NuGet						✓		✓	✓			✓
Go		✓	✓			✓						

* Gradle [38] tool is vulnerable to Dependency Confusion Attack (C_1).

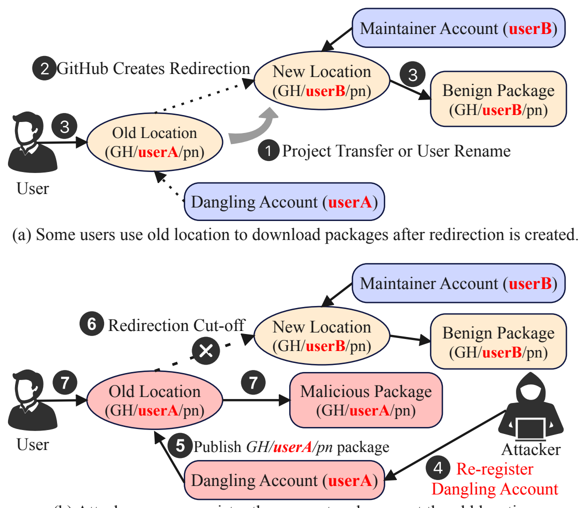
all packages' versions and release time; the npm registry provides CouchDB APIs [32], which can be used to obtain package addition and deletion events. Adversaries can also find expired domain names by analyzing DNS records (e.g., via the NXDOMAIN status [33] using the *whois* command [34]).

In most attacks, adversaries need to publish packages to the registries. These packages could contain malicious code to compromise victims (once the packages are obtained by victims). Thus, it requires the adversaries to have the capability to circumvent the target registries' checking mechanisms. While the creation of such malicious code is outside the scope of this paper, it has been shown that existing registries already contain many malicious or vulnerable packages [35] [36] [37] [6]. Once victims are tricked into downloading and installing malicious packages, the damage could be severe: adversaries can (1) implant a backdoor so that they can remotely control the victim's system [37]; (2) steal the victim's sensitive data [35]; (3) generate financial profits (such as mining cryptocurrency using the victim's machine) [6]; and (4) harvest computing resources of the host machine for launching advanced attacks (e.g., denial-of-service) [36].

In the remainder of this section, we discuss twelve potential attack vectors related to different stakeholders, belonging to six categories. In each category, we introduce the specific threat model and attackers' motivation. We mark six attacks disclosed by us for the first time (to the best of our knowledge) as *novel*. We report our new findings for other attacks, including new exploitation approaches. All vectors combined form the basis of our scanning tool, *RScouter*, for a large-scale measurement study in Section 4. A summary of investigated registry ecosystems with potential vulnerabilities is listed in Table 1.

3.1. Package Hijacking in Upstream Registries

Threat Model and Motivation. Attackers attempt to bait users into obtaining software packages maintained by adversaries. They can hijack (e.g., override or take over) existing packages through normal activities (such as publishing a package). Once the target package is hijacked, future users will get the attacker's package instead of the original one. The victim package of current users will also be replaced if they upgrade their packages. Generally, this type of attack requires attackers to publish new packages or purchase expired domain names. For publishing a new package, attackers need to


Maintainer Account (userB)
2 GitHub Creates Redirection
New Location (GH/userB/pn)
3
Benign Package (GH/userB/pn)
3
User
Old Location (GH/userA/pn)
1 Project Transfer or User Rename
Dangling Account (userA)
Maintainer Account (userB)
6 Redirection Cut-off
New Location (GH/userB/pn)
Benign Package (GH/userB/pn)
7
User
Old Location (GH/userA/pn)
7
Malicious Package (GH/userA/pn)
Attacker
5 Publish GH/userA/pn package
Dangling Account (userA)
4 Re-register Dangling Account

(a) Some users use old location to download packages after redirection is created.

(b) Attackers can re-register the account and preempt the old location.

Figure 1. Overview of Package Redirection Hijacking Attack (U_3).

register a new account on the registry web portal or code hosting platforms (e.g., GitHub), which typically requires a unique and verifiable email address. For some hijacking attacks, attackers need to purchase an expired domain name. As domain registration generally costs \$2~\$20 per year [39], we consider that the cost is low.

- **Package Use-After-Free (UAF) Attack (U_1).** Users rely on a unique identifier to retrieve a package. A package UAF attack can occur if a registry allows package maintainers to reuse the name of a deleted package. Attackers can continuously monitor the registry's package list and preempt those deleted packages with malicious ones. Then, users who attempt to use the previous packages will download the new (malicious) packages published by the attackers. Besides, if a project has integrated the previous packages, the update process will replace the benign packages with the malicious ones.
- **Package Maintainer Account Hijacking Attack (U_2).** If attackers can compromise a package maintainer's account, they can inject malicious code into existing packages or release new malicious packages under the compromised account. In particular, existing registries primarily rely on third-party identity provider services (e.g., email providers) to register accounts. The resource reuse on the identity provider side is unknown to software registries. If the email domain name for registering the maintainer account is expired, attackers can purchase this domain name and then take over the account on the registry side through password recovery. Similarly, attackers can register any deleted GitHub accounts and take over the associated maintainer accounts if a registry relies on GitHub accounts to sign in.
- **Package Redirection Hijacking Attack (U_3 – *Novel*).** In this type of attack, attackers can hijack a package *without* compromising the current maintainer account for the target package, as illustrated in Figure 1. For decentralized registries, such as *Go*, there are no web portals for package maintainers to manage packages, but they utilize code hosting platforms (e.g., GitHub and GitLab) to publish and manage packages. Instead, Go only maintains an official website (i.e.,

pkg.go.dev) for users to search and view the Go packages. Code hosting platforms support users to change their username [10] or transfer their project to another account [11] (step ❶). For these cases, GitHub will automatically redirect the package from the old repository location to the new location (via the HTTP response status code `301 Moved Permanently`), and both links can be used to download the package (steps ❷❸). The problem is that the registry's website (e.g., *pkg.go.dev*) is unaware of the changes and does not update the *Repository* field of the package with the new location accordingly. As shown in Figure 1, if the GitHub account corresponding to the old repository location is deleted, attackers can re-register the account and preempt the previous repository location (steps ❹❺), and GitHub will immediately cut off the redirection (step ❻). As a result, users will obtain the malicious package when downloading or upgrading the victim package if they use the registry's official website (step ❼).

3.2. Package Hijacking in Registry Mirrors

Threat Model and Motivation. This type of adversary attempts to hijack packages in registry mirrors. However, they do not have the capability to directly publish a package to the mirror (to potentially hijack a package). Instead, they can monitor package information in the target mirror and publish packages to the corresponding upstream registry.

- **Mirror Package Override Attack (M_1 – *Novel*).** Registry mirrors should maintain exactly the same content as the official registry. However, in practice, a mirror might still store packages that do not exist in the upstream registry. For example, a mirror may not synchronously delete packages that have been deleted in the registry. Another example is that mirror maintainers might publish internal packages. If a mirror stores a package that does not exist in the official registry, it may allow attackers to override the package on the mirror. Attackers can compare the package lists between the official registry and the target mirror to locate those packages. Then they can publish a package to the registry with the same name, causing the package in the mirror to be overwritten during the synchronization process. In particular, for mirrors that use the proxy mode or support forced synchronization, attackers can accelerate the attacking process by forcing a package sync.
- **Case Sensitivity Confusion Attack (M_2 – *Novel*).** Some registries support package names with mixed letter cases. For example, in the npm registry, *Buffer* and *buffer* are two distinct packages. However, a registry mirror may fail to handle the case logic correctly. In this situation, no matter what the download request is (e.g., *Buffer* or *buffer*), users will always get the same package (e.g., *buffer*). Thus, the downloaded package by users will not match the package name they entered. Attackers can exploit registry mirrors by simply publishing a malicious package with a different case of the benign package name.

3.3. Typosquatting (T_1) in Registries

Threat Model, Motivation, and Attacks. Typosquatting has been a widespread tactic for abusing many Internet services [40] [41] [42] [43]. This type of adversary attempts to exploit potential typos made by users in software registries. Attackers can generate many typosquatting package names similar to existing popular and benign packages. If users accidentally misspell the package name (e.g., by making a typo) or omit the specifier of a package name, they will obtain an incorrect package instead of the intended one. In particular, some registry ecosystems like Maven, npm, and Go use a segmented package name scheme, which consists of a *scope segment* and a *name segment*: the former should be unique in the registry, and the latter can be duplicated. In this case, attackers can carry out typosquatting attacks by constructing a package with the exact same *name segment* and a similar *scope segment* as the victim package.

We also consider a cross-registry scenario [44] where multiple registries contain packages with the same package name. Users might naively think that these packages across different registries provide the same functionalities. Adversaries can exploit this to launch a cross-registry squatting attack by intentionally publishing lure packages whose names exist in other registries but not in the target registry. As a result, users might download the squatting package.

3.4. Reference Attack and Variants

Threat Model and Motivation. In this type of attack, we assume users have already downloaded one normal package controlled by adversaries (e.g., via the above-mentioned hijacking or typosquatting attacks). Meanwhile, although adversaries have the capabilities to create packages containing vulnerabilities in registries, it is challenging to create a large amount of them without being noticed. We assume adversaries may only successfully deploy a limited number of malicious packages in registries. Also, those malicious packages may not be popular (users will not download them voluntarily). Thus, adversaries have the motivation to attract victims to use their malicious packages.

The normal package controlled by attackers looks unsuspicious to users, as the malicious logic is not directly implemented in this package. Instead, adversaries attempt to utilize this normal package to further direct victims to their malicious packages through the methods described below. The motivation is to mount the attack stealthily (by avoiding publishing many packages that directly contain malicious code/vulnerability) but to affect more victim users. Adversaries can (1) reduce the number of uploaded packages directly containing malicious logic so that they can reduce the possibility of being detected; and (2) attract victims to download unpopular malicious packages.

- **Package Reference Attack (R_1).** Instead of injecting malicious code directly into existing packages, attackers can reference malicious packages in a normal package. Users will then automatically download malicious packages when using the normal package. Such an attack is subtle as malicious and trusted code is stored separately in different packages. It has been used in practice: the maintainer imports a dependent malicious package (i.e., the *peacenotwar* package) in the recent npm package *node-ipc* [45] incident (CVE-2022-23812), and attracts tens of thousands of users to download.

Dangling References. In particular, we find the existence of many dangling references in published packages. A dangling reference is a non-existent package (potentially caused by package unpublishing) referenced by benign packages. Attackers can exploit these dangling references to launch package reference attacks by hijacking the dangling package (e.g., via publishing a package with the same name). For this package, they can directly publish malicious packages or further reference to other malicious packages (i.e., nested dependency). Thus, for safety concerns, registries should be cautious about these references. For example, they should remove any normal packages that reference known malicious and dangling packages.

- **Ghost Package Attack in Mirrors (R_2 – *Novel*).** The package reference threat can be even worse in registry mirrors if they fail to unpublish malicious packages synchronously. The problem is that registry mirrors may not handle package synchronization properly, especially for package unpublishing. A registry can unpublish a package for security or maintenance reasons. Depending on its policy, a registry may (1) entirely delete a package's content and metadata, (2) remove it from the index file while keeping package content, or (3) only change package status. If a registry mirror cannot precisely synchronize with the official registry in the package unpublishing operation, some packages that have been unpublished in the official registry will still be accessible in the mirror. We refer to these packages as *ghost packages* in mirrors. If the package reference attack happens to the ghost package (i.e., referencing a ghost package), mirror users will still suffer the threat even when the original malicious package has been deleted from the registry.
  Creating Ghost Packages. Attackers could deliberately create a ghost package in these vulnerable mirrors. First, they can publish a malicious package to the registry and then wait for (or force) the mirror to synchronize the package. After that, they can actively delete it from the registry. Because the mirror cannot precisely synchronize with the official registry, this package becomes a ghost package in this mirror. Also, if automated malware detection systems [46] [47] only monitor official registries but not mirrors, it might be difficult to detect these malicious ghost packages. Attackers can intentionally reference a malicious ghost package (in a normal package) to attack mirror users.

3.5. Registry Client Related Vulnerabilities

We further report two vulnerabilities in existing registry clients. We find that, in certain conditions, some registry clients cannot correctly handle different versions of the same package. While exploiting them requires strong threat models (described individually below), they could still potentially cause security threats to users.

- **Dependency Confusion Attack (C_1).** It is common for users to use multiple sources (e.g., public and private registries) to download packages. This hybrid configuration might automatically pick the source with the highest version number for a package with multiple different versions. A dependency confusion attack (DCA) occurs when attackers discover a private package that does not exist in public registries [48]. Attackers can simply publish a malicious substitute with the same name but a higher version number to public registries, forcing the users to download the malicious version unknowingly. This attack differs from the aforementioned hijacking attacks, as attackers do not override any packages in registries. However, it requires the attacker to gain internal information about private packages. Previous works have studied this problem in the *virtual repository* mode supported by many popular package management tools (e.g., *Nexus* [15] and *Artifactory* [16]), referred to as virtual repository-side DCA. Both public and private registries are logically integrated into a virtual repository. Thus, users using these virtual repositories potentially suffer from this attack. We further show that some registry clients can be similarly exploited (referred to as registry client-side DCA). Users configure multiple upstream registries for registry clients *without* using the virtual repository. Both cases will automatically pick the package with the highest version number for users. We illustrate the details of both attacks in Figure 9 in the Appendix.
- **Package Version Downgrade Attack (C_2 – *Novel*).** Packages can set constraints on their allowed versions of all dependent packages (i.e., the maximum and minimum package versions) in package configurations, to avoid using buggy versions of a particular package (named as P_{vul}). We find that these constraints might be broken by package references in some registry clients. If another package (used by the user) references a lower version of P_{vul}, the registry client will replace the existing version of P_{vul} with the specific lower version, which the constraint prohibits. Attackers can exploit C_2 to break version constraints of other packages, if the victim user has already downloaded one normal package controlled by adversaries, as well as a benign version of package P_{vul}. Attackers can then reference a low-version of P_{vul} (which contains vulnerabilities) in the normal package. As a result, the benign version of package P_{vul} in the victim's client will be replaced by the vulnerable low-version. Such an attack is unobtrusive for users as the current version of all used packages has no malicious code.

3.6. Miscellaneous Vulnerabilities

We finally present two miscellaneous vulnerabilities. Exploiting them might be difficult in practice, as it requires attackers to have extra capabilities. However, we believe registries and mirrors should prevent them from happening.

- **Package Version Reuse Attack (O_1 – *Novel*).** If the registry allows version reuse (i.e., packages that use the same version number but have modified code), it potentially allows attackers to inject malicious code stealthily. Instead of directly publishing a malicious package (which users might notice), attackers can first publish a normal package to attract users. After victim users have downloaded this normal package, they can update it with malicious code while keeping the version number unchanged. Users might be unaware of the change with the same version number. Furthermore, it can attack users with *version pinning* enabled, a feature supported by all registry clients. Users can use version pinning to indicate the specific version of the package, ensuring the same codebase is used (and potentially avoiding vulnerable versions). For the above reasons, registries usually have disabled version reuse, such as the PyPI and npm registries [49] [50].

- **Package Tampering Attack (O_2).** A package generally contains an integrity checking file (e.g., SHA1 checksum) to avoid using tampered packages during the downloading process. Registry mirrors should not remove this integrity checking file; otherwise, users have to turn off the registry client's package integrity checking function to use the mirror normally. It might cause damage if adversaries can intercept the user's network traffic. For example, if registry mirrors support the insecure HTTP channel, adversaries can modify packages by implementing a MITM attack [31].

4. Measurement Methodology

To investigate the current security status of the software registry ecosystem, we design *RScouter* to monitor popular registries and their mirrors continuously. Overall, *RScouter* first collects a broad spectrum of package information (e.g., package content and metadata) and monitors various events (e.g., package addition, deletion, and modification). It then conducts a differential analysis of the contents between an official registry and its mirrors to capture any inconsistency. With the collected data, *RScouter* adopts multiple vulnerability detectors to find potential vulnerabilities. The overall workflow of *RScouter* is illustrated in Figure 2. This section details all components of *RScouter*. We also discuss ethical concerns and limitations.

4.1. Data Source

Registries. We strive to cover all types of registries, including both compiled and interpreted programming languages, centralized and decentralized registries, and different login methods. We thus select official registries for the six popular programming languages based on the TIOBE [51] and PYPL [52] popularity indices, including Maven Central, PyPI, npm, NuGet, Cargo, and Go. Among them, Maven Central, PyPI, and npm utilize emails to register package maintainer accounts. NuGet uses Microsoft accounts, and Cargo uses GitHub accounts for sign-in. Finally, Go has no centralized package hosting site, and packages are directly maintained on code hosting platforms. We treat the code hosting platform's account as the package maintainer account.

Registry Mirrors. Since there is no official registry mirror list, we manually find several widely used mirrors for each registry through the GitHub and Google search functions. While most threats studied in this paper are general problems in mirrors, we primarily focus on mirrors in China, as mirrors are widely used there to accelerate and bypass the Great Firewall [69]. We choose seventeen different target mirrors from all six registries (details in Table 2).

4.2. Crawler and Package Parser

We first develop a crawler to collect package information and observe all related activities (e.g., releasing a new version or removing a package). All six target registries provide package index files maintaining the package name, version, and other information. When a package changes, the corresponding package index file changes accordingly. We thus regularly record package information by parsing the package index file daily. We also download and record the package and its metadata for every version of all packages. Particularly, we extract references between packages and build a dependency graph.

TABLE 2. Overview of potential threats on each registry mirror. The columns highlighted in gray color indicate novel attacks. The ✓ means vulnerable.

Registry	Mirrors	M_1	M_2	R_2	O_2
Maven Central	Aliyun [53]			✓	
	Huawei [54]			✓	✓
	NJU [55]	✓			
PyPI	Aliyun [56]				✓
	Douban [57]	✓		✓	✓
	NJU [58]	✓			
	Tsinghua [59]	✓		✓	
npm	Aliyun [14]	✓	✓	✓	✓
	Huawei [60]	✓		✓	
	Tencent [61]				✓
	NJU [62]				
Cargo	Tsinghua [63]				
	STJU [64]				
	USTC [65]				
NuGet	Huawei [66]			✓	✓
	Azure China [67]				
Go	Qiniu [68]				

We use three different strategies for registry mirrors. (1) For mirrors that provide package index files, we use the processing method similar to the above; (2) for mirrors that allow traversal of their file structure on the web, we collect data through a web crawler; (3) for other mirrors, we collect packages and their metadata using the mirrors' download interface. Notably, for the last method, we consider that a deleted package (from the registry) is removed from the mirror if it cannot be downloaded.

We extract each package's repository information (e.g., the registry or mirror information), package name, version, publish time (i.e., the package's release time), maintainers, contact information, dependency packages, and file hash values. The maintainer's information includes the login account (i.e., maintainer account) for managing the package on the registry web. The contact information represents the package developers' contact information (typically emails).

Maintainer Accounts. We use several approaches to extract the maintainer accounts. First, *npm* and *Maven Central* allow us to obtain account information directly from public sources. npm provides the package maintainer account information, including username and email, in the package metadata, from which we successfully extracted 595,508 unique accounts. Maven Central uses JIRA [70] to manage package publish requests [71]. When a maintainer applies for a new package release, she/he needs to register an account on JIRA and create an issue claiming the requested package name. Then

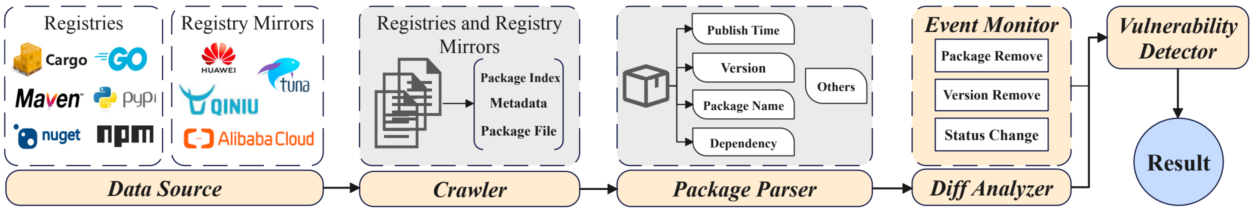


Figure 2. Workflow of *RScouter*.

Maven Central will use the same login email and password as the applicant's JIRA account to create a maintainer account on the web portal. Anyone with a JIRA account can browse all issues, including the creator's login email and requesting package name. We use this method to successfully extract 21,213 independent accounts in Maven Central.

For *Cargo* and *Go*, we extract their account information through the source code repository URL. Cargo uses the GitHub account sign-in function to log in to the registry maintainer account. Meanwhile, most packages in Cargo expose the source code repository URL on GitHub, where we extract the GitHub account identifier. Among the 74,441 packages in Cargo, we find that 56,795 (76.3%) packages expose GitHub links, and a total of 16,441 independent accounts are collected. Similarly, we analyze the vast majority of Go packages hosted on GitHub (91.8%) and GitLab (1.2%), and extract a total of 206,220 independent accounts.

Next, for *PyPI*, we infer maintainer accounts through contact information. PyPI does not disclose the maintainer's account in package metadata by default, but 290,911 out of 346,000 (84.1%) packages provide a total of 138,039 independent *contact* emails. We infer that most contact emails are the same as the maintainer accounts' login emails. For example, for Maven Central and npm that provide both information, 46.5% of 109,385 Maven Central accounts and 66.3% of 577,000 npm accounts use the same email for both *contact* and *maintainer account*. Thus, we conduct an analysis using the contact information on PyPI.

Finally, we find that *NuGet* does not expose the maintainer account or contact information. If the user wants to contact the package maintainer, NuGet utilizes an email forwarding function that only exposes the email when the maintainer replies. Thus, we are unable to collect the account information on NuGet.

4.3. Event Monitor and Diff Analyzer

RScouter monitors several events in registries, including the addition, removal, and modification of packages. Particularly, registries or package maintainers might remove a specific version or the whole package (i.e., all versions of a package). Instead of removing a package, they can also change the access status of a package from downloadable to not-downloadable. *RScouter* also records if the content of a specific version of the package file has been changed.

RScouter further conducts a differential analysis of the contents between a registry mirror and its corresponding official registry. As registry mirrors are supposed to maintain the same content as the official registry, any differences can potentially be used to identify problems in mirrors. In particular, we focus on the following differences: (1) mirrors do not synchronously remove the deleted package in the official registry; (2) mirrors do not synchronously remove some versions of packages deleted from the official registry; (3) mirrors do not synchronize a package's access status change in the official registry; (4) packages only exist in mirrors but not in the official registry; and (5) mirrors modify the contents of the package file.

4.4. Vulnerability Detector

Based on the logic of each attack, we implement corresponding detectors to identify potential vulnerabilities in the existing software registry ecosystem. Since most implementations are trivial, we list some details below. We adopt a similar method as [20] to identify vulnerable email accounts that attackers can potentially hijack. We use the WHOIS lookup service to find expired domain names by checking the NXDOMAIN status (which indicates a non-registered domain), whose accounts are then considered vulnerable. We also utilize the APIs provided by GitHub [72] and GitLab [73] to find deleted accounts. We detect version reuse if the hash value is the only difference for two tuples ⟨repository, package name, version, hash value⟩.

In mirrors, we first collect and group all packages with the same name but different cases in a registry. For each group (which includes at least two packages), we use the registry client to download each package and observe whether or not the downloaded package is inconsistent with the requesting name. For attacks related to references, we focus on two situations: malicious references and dangling references. The former means a benign package refers to a known malicious package (e.g., identified by the registry). The latter means a benign package refers to a non-existent package that attackers can exploit.

4.5. Ethical Considerations

We take ethical considerations very seriously. To assess security issues in software registries, we constantly monitor and measure package status in target registries and mirrors. Also, we intentionally register several accounts and upload multiple packages to the registry ecosystems. For all these experiments, we have conducted them in legitimate manners. First, we never compromise any accounts in any involved parties but register all accounts legally through the web interface. Meanwhile, all crawled information is public and can be obtained through legitimate approaches. Second, we never intentionally trick users. Szurdi et al. [74] studied email typosquatting by intentionally registering typosquatting domain names, and Liu et al. [75] studied container typosquatting by intentionally publishing docker images. We have not conducted similar experiments, but only reported the current status (e.g., potential threats). Third, we have never hijacked any account. For most threats, we verify them on our own package, which is set as private when possible. Finally, at the end of our study, we have manually deleted all of our published packages/accounts and disclosed our findings to the corresponding parties via emails.

The only exception is for the *Mirror Package Override Attack* (M_1). The verification step of M_1 is semi-automated. It includes an automated process to identify potentially vulnerable packages in mirrors based on the Diff Analyzer of *RScouter* (e.g., by finding packages that only exist in mirrors) and a manual process to further confirm the vulnerability. However, we cannot verify the problem with our own package as we are unable to directly publish a package to a mirror. We have conducted the following steps to minimize the potential impact. First, for packages that only exist in mirrors, we publish a package with *the exact same name and content* to the official registry. We then observe whether the package in the mirrors is overridden. If the package is overridden, only the package description will be changed, which has no (or negligible) impact on users. For the first tested mirror, we only tested two packages in total (the first one to verify the vulnerability and the second one to avoid measurement errors). We then immediately contact the mirror maintainers to inform them of the vulnerability. For the rest of the mirrors, we conducted the experiments *after* the corresponding mirror maintainers having granted our permission. All our uploaded packages are removed immediately after the verification. In fact, all mirror maintainers have confirmed that no real users were affected during the experiment. Thus, we argue that the ethical impact is negligible.

4.6. Limitations

Registries and registry mirrors are black boxes for us; we can only detect potential threats through data analysis and testing. Thus, our approach might cause false positives/negatives. First, we can only find contact emails for PyPI when collecting the maintainer accounts, and it may generate false positives as contact emails are not necessarily the same as maintainer accounts. Similarly, because we cannot find a complete list of malicious packages deleted by the PyPI registry, whose names are not allowed to re-register, it also may cause false positives. Second, we utilize the NXDOMAIN status to identify expired email domain names. This method cannot cover expired domains registered by domain name registrars or brokers for sale. Thus, our result might contain false negatives, indicating that the actual situation might be even worse than our results. Third, the account hijacking attack may not succeed if the account is protected by multi-factor authentication (MFA). However, we find that all registries either do not provide MFA functionalities (i.e., Maven Central, Cargo) or do not enable MFA by default (i.e., PyPI, npm, NuGet). Thus, the impact of MFA might be limited. Note that npm has forced MFA login authentication from March 1st, 2022, later than the end time of our evaluation. Fourth, according to GitHub policy [76], the deleted account name becomes available after 90 days. Some vulnerable accounts we uncovered may still be in the 90-day time window, making registration impossible. Finally, we cannot pull all packages from the registry mirrors that do not provide a package index file and do not allow us to traverse the file structure. As a result, we cannot determine whether these mirrors contain packages that do not exist in their corresponding registries (as M_1).

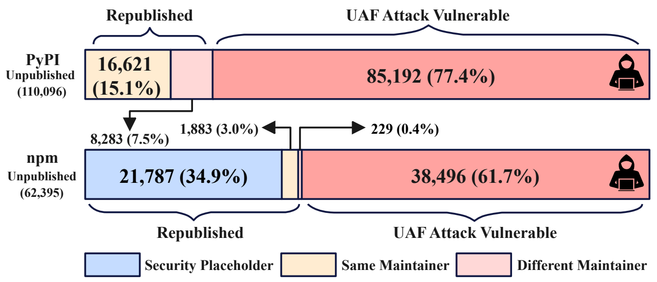


Figure 3. PyPI and npm unpublishing statistics.

5. Measurement Results

We adopt *RScouter* to conduct the evaluation for twelve months, starting from January 2021, and have collected 4,061,330 unique packages across 53,282,905 distinct package versions. We identify that many registries and mirrors monitored by *RScouter* are vulnerable to the threats mentioned in this paper. In this section, we report our measurement results in detail.

5.1. Package Hijacking in Upstream Registries

Package Use-After-Free Attack (U_1). We find that PyPI and npm are vulnerable to this attack. In PyPI, we use its XML-RPC APIs [77] to gather package updating activities, while in npm, we obtain the list of unpublished packages by continuously monitoring the changes in the package index. Our results indicate that the unpublishing operation happens frequently: 110,096 packages in PyPI and 62,395 packages in npm (since August 2021) have been unpublished. Any of these packages that are not republished are subject to this attack. Specifically, the number of vulnerable packages is 85,192 (77.3%) in PyPI and 38,496 (61.7%) in npm, as shown in Figure 3.

We further analyze whether the U_1 risk already existed in these two registries. Among 24,904 republished packages in PyPI, 8,283 (33.2%) were republished by *different* maintainers. If a package is identified as malicious and deleted by the registry, PyPI does not allow re-registration of the package with the same name. As for npm, 229 out of 23,899 republished packages were republished by *different* maintainers. Surprisingly, the npm web portal displays the download count of the previously unpublished package on the republished one, making it more difficult for users to identify a package UAF attack.

Package Maintainer Account Hijacking Attack (U_2). We find that five registries (except NuGet) are vulnerable to

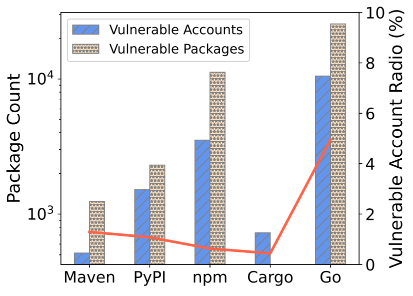


Figure 4. Package maintainer accounts takeover measurement results.

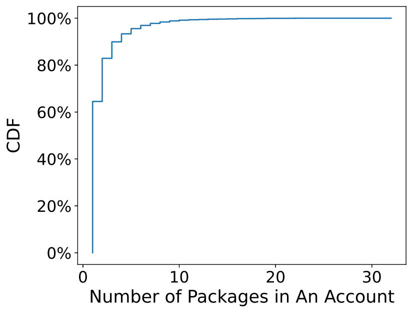


Figure 5. Package redirection results.

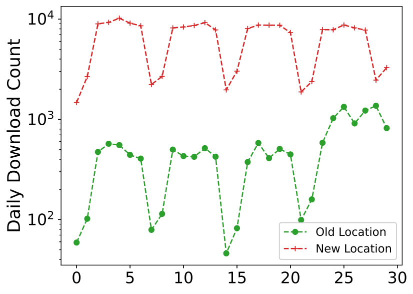


Figure 6. Vulnerable Go packages' daily download count from the Qiniu proxy (2022-02-12~2022-03-13).

U_2. As shown in Figure 4, all registries have over 16,807 accounts and 40,327 vulnerable packages in total. The ratios of vulnerable accounts to all accounts in each registry vary from 0.44% (Cargo) to 4.96% (Go). We should mention that, even after taking over the maintainer account, Cargo does not allow attackers to modify packages published by the previous owner. However, attackers hold control of all other resources and can publish new packages under this account.

Package Redirection Hijacking Attack (U_3). Go is vulnerable to U_3. From 597,340 Go packages hosted on GitHub, we find 11,788 (2.0%) vulnerable packages with redirection, for which the accounts corresponding to the old location have been deleted. Among 8,323 packages hosted on GitLab, we detect 333 (4.0%) vulnerable Go packages. Figure 5 shows the distribution of the number of packages affected by those deleted accounts. 64.4% of those accounts affect one project, and 6.7% of them affect more than five projects. In particular, the account *s**io* (name hidden due to ethical concerns), which has been deleted on GitHub, has affected 32 Go packages.

Since Go is a decentralized registry, there is no download count for each package. We thus use a Go proxy [68] that provides download statistics to evaluate the impact if these Go packages are under attack. Figure 6 shows the number of downloads using both the old and the new location for vulnerable Go packages in the time window from February 12th, 2022, to March 13th, 2022. We can observe that the old location still attracts many downloads. Some packages even have the old location dominated for downloads. For example, all 6,066 downloads of the package *github.com/0LuigiCode0/library* are from the old location *github.com/000mrLuigi000/Library*. Note that our data is collected from merely one proxy, and the actual download count should be much larger.

5.2. Package Hijacking in Registry Mirrors

Mirror Package Override Attack (M_1). We find a large number of packages in PyPI, npm, and Maven mirrors can be overridden immediately. Specifically, for PyPI mirrors, 51,387 packages in Douban, 32,616 packages in Tsinghua, and two packages in NJU are vulnerable to this attack. For npm mirrors, we find that 11,763 vulnerable packages exist in the Aliyun mirror and 2,363 packages in the Huawei mirror. Finally, the NJU Maven mirror exposes 11 packages.

TABLE 3. Incomplete list of case sensitivity problems in the Aliyun npm mirror. The packages highlighted in gray color are the *winners*.

Packages with Uppercase			Packages with All Lowercase		
Name	**DL***	**Update**†	**Name**	**DL***	**Update**†
Buffer	7,219	2011-08-01	buffer	173M	2020-11-23
Express	574	2016-12-07	express	95M	2022-02-17
Promise	1,111	2016-09-13	promise	55M	2020-03-02
D	23,390	2017-08-11	d	42M	2019-06-14
MD5	24,008	2015-07-15	md5	31M	2020-08-02
Validator	1,761	2021-08-01	validator	28M	2021-11-01
JSONStream	24M	2018-10-14	jsonstream	12,114	2015-04-30
FileList	17,294	2014-10-24	filelist	22M	2021-02-06
jQuery	13,279	2012-07-01	jquery	18M	2021-03-02
jStat	79,393	2019-11-04	jstat	499,271	2021-08-10

* Download count from 2022-02-04 to 2022-03-04. *M = Million*.
† Last update time of a package. The date format is yyyy-MM-DD.

Just like normal packages on the mirror, these packages are all listed on the index page and can be accessed by public users. We have further confirmed that none of these packages is malicious by antivirus software detection and manual verification. After collaborating with mirror maintainers, we find that the vulnerable packages in both NJU PyPI and Maven mirrors are private (e.g., added by internal users). The vulnerable packages in all other mirrors are packages that have been unpublished in official registries but are not synchronously deleted in mirrors.

Case Sensitivity Confusion Attack (M_2). We find npm contains package names with mixed cases. Although npm has banned newly released packages from including any uppercase letters since 2017 [78], there are still 793 groups, a total of 1,588 packages, with the same name but different cases. Some packages are very popular, as shown in Table 3. For example, the package *buffer* has more than 170 million downloads per month, and *Buffer* has more than 7,000 downloads per month. These packages were published before the uppercase ban, but they can still be updated and downloaded.

We then find that the Aliyun npm mirror, which has more than 2 billion monthly downloads, does not handle these mixed-case packages properly. Our testing results show that users will always get the last updated package (i.e., the package with the latest version release time). For example, the *buffer* is updated after *Buffer*; no matter which package is requested by the user (*Buffer* or *buffer*), the final downloaded package is always the *buffer*.

The case sensitivity confusion problem might cause a

TABLE 4. Top 10 downloaded packages affected by M_2.

Package Name	Downloads*
objectFitPolyfill	1,106,785
CSSselect	162,742
CSSwhat	155,559
jxLoader	137,346
diveSync	83,180
canvas-toBlob	41,734
wProto	30,874
wConsequence	30,850
wCopyable	30,672
wColor	30,404

* From 2022-02-04 to 2022-03-04.

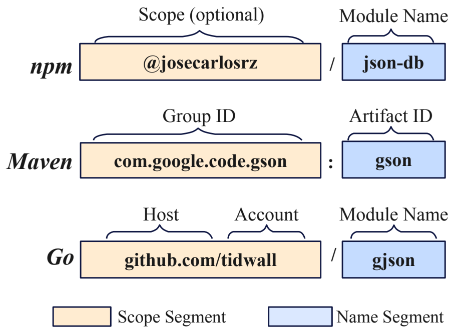


Figure 7. Package name schemes.

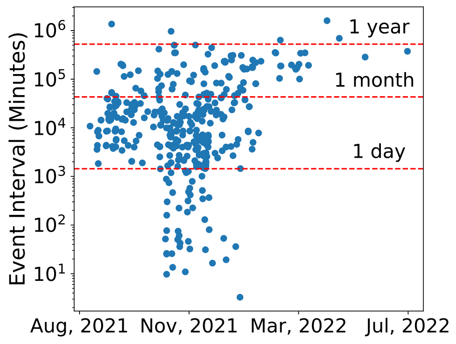


Figure 8. Version reuse results.

TABLE 5. Typosquatting measurement results.

Registry	# Pkgs	# DL-1 Pkgs	# Deleted	# DL-1 Pkgs	# Malicious	# DL-1 Pkgs	Identical Pkgs Segment	
							# DL-1 Pkgs	# Total
Maven Central	457,827	81,149 (17.7%)	-	-	-	-	353 (3.5%)	10,202
PyPI	363,320	116,369 (32.0%)	110,096	89,121 (80.9%)	-	-	-	-
npm	1,973,657	132,068 (6.7%)	40,608	35,792 (88.1%)	21,787	21,108 (96.9%)	34,792 (15.0%)	231,416
Cargo	79,510	18,304 (23.0%)	-	-	-	-	-	-
NuGet	521,201	47,439 (9.1%)	-	-	-	-	-	-
Go	644,256	73,249 (11.4%)	-	-	-	-	372 (0.1%)	268,921

more severe security threat. For a benign package containing uppercase letters in the npm registry, if its corresponding package with all lowercase letters does not exist, the attacker can publish a malicious package with all lowercase letters to the registry. As long as the attacker ensures that the last update time of the malicious package is newer than the benign one (which is very easy to achieve), any user using the Aliyun npm mirror to download the benign package (with uppercase) will end up with the malicious package (with lowercase), even though the user has entered the correct benign package name. We count the number of such vulnerable packages and find this attack threatens 2,038 packages. The top 10 most-downloaded packages among them are shown in Table 4. Particularly, the package *objectFitPolyfill* has more than 1 million downloads in one month, potentially affecting a significant number of users.

5.3. Typosquatting (T_1) in Registries

It has been shown that people are more likely to make typos that involve a one-character distance, also called as Damerau-Levenshtein distance one (i.e., DL-1) [79]. We hence measure the package identifier pairs of DL-1 for all six registries. The result is shown in Table 5. Among all existed packages in Maven Central, PyPI, npm, Cargo, NuGet, and Go, the packages with at least one similar DL-1 package account for 81,149 (17.7%), 116,369 (32.0%), 132,068 (6.7%), 18,304 (23.0%), 47,439 (9.1%) and 73,249 (11.4%), respectively.

We have two interesting observations: (1) on PyPI and npm, which allow deleting packages, the deleted packages with at least one similar DL-1 package are 81.0% and 88.1%, respectively; and (2) 96.9% of malicious packages (officially marked by npm) are similar to existing benign packages (i.e., DL is 1). The result indicates that typosquatting is already heavily abused by attackers.

We also analyze registries using the segmented package name, such as Maven Central, npm, and Go. As shown in Figure 7, typically, the package name contains two parts: a *scope segment* (i.e., a specifier used for namespace isolation) and a *name segment* (i.e., a label used to identify a package in the scope). As Table 5 shows, in Maven Central, there are 353 (3.5%) pairs of similar packages whose name segments (i.e., `Artifact ID`) are exactly the same, and scope segments (i.e., `Group ID`) are DL-1. This number is 372 (0.14%) in Go, and even higher in npm, up to 34,792 (15.0%). These results show that the segmented package name also potentially suffers the typosquatting threat.

In npm, we further find a *scope-related typosquatting* threat, because the *scope* in npm is optional [80]. There are 266,465 (13.50%) npm packages with scope segments, but their corresponding non-scoped package names are available for registering. For example, there are 462 different scopes for package names *@{scope}/lo**nt* (name hidden due to ethical concerns), but attackers can register the name *lo**nt* without any scope. If users forget to enter the scope segment (since it is optional), they will download the *lo**nt* package without any scope. In total, we identify 233,922 such package names, affecting 80,013 scopes.

Cross-Registry Squatting. We evaluate the cross-registry squatting risk across four registries with non-scoped (or optional-scope) package names, including PyPI, npm, Cargo and NuGet. Among the top 10k PyPI packages downloaded in 2021, only 363 package names exist in all four registries simultaneously, and 747 package names exist in three registries. Thus, the last registry might be vulnerable to this type of typosquatting risk. For example, we find a package *r**x* (name hidden due to ethical concerns), with more than 20 million downloads per month in PyPI, more than 67 million downloads in total in Cargo, and thousands of downloads per week in npm, does not yet exist in NuGet. Attackers can preempt this in NuGet and potentially attract victims.

5.4. Reference Attack and Variants

Package Reference Attack (R_1). During our monitoring, we discover eight *real* package reference attacks against PyPI. One is the package *pydspace* released on December 18th, 2021. This package does not contain malicious code itself but refers to the malicious package *matlabengineforpython*. It installs the malicious package synchronously during the installation process. Note that *matlabengineforpython* was officially removed by PyPI on March 1st, 2021, before the release of *pydspace*. Another is *AAmiles*, published on May 18th, 2020, which references the malicious package *request*. PyPI removed the malicious package *request* on August 4th, 2020. As a result, when the user uses the PyPI registry to install *pydspace* and *AAmiles*, the installation is terminated due to the failure of dependency resolution.

However, we find that two widely used mirrors (i.e., Douban and Tsinghua) do not delete the malicious package synchronously. Therefore, users using these mirrors will still download malicious packages when installing *pydspace* and *AAmiles*. Note that *pydspace* references a deleted malicious package, representing a real example of mounting reference attack on ghost packages.

Dangling References. Moreover, after analyzing more than 120 million dependency information of 1.9 million packages, we find that 228 packages in npm have the dangling reference problem: the referenced package does not exist in the registry and can be taken over by attackers immediately. This is interesting, as npm explicitly states that a package cannot be unpublished if other packages depend on it [50]. Installing those packages will be terminated due to dependency resolution failure. Still, once attackers reuse these package names, their dependency resolution and installation process will succeed, which means malicious code will be executed. Similarly, we find 561 packages in PyPI that have the dangling reference problem.

In addition, when handling a malicious package, npm will unpublish all versions of the package and then publish a security placeholder. The placeholder's name is the same as the malicious package, but the version is always `0.0.1-security`. We find that 803 npm packages with at least one version reference the security placeholder, indicating that these packages reference a deleted malicious package at these versions. Note that attackers cannot exploit these placeholder-related dangling packages directly.

Ghost Package Attack in Mirrors (R_2). We further find that many mirrors have synchronization issues. First, some mirrors do not synchronously delete packages that have been deleted in the upstream registry. These ghost packages are also vulnerable to Mirror Package Override Attack (M_1). Vulnerable mirrors include Douban and Tsinghua PyPI mirrors, and Aliyun and Huawei npm mirrors, with the specific numbers of vulnerable packages presented in M_1.

Second, some mirrors do not synchronize the access status of the package when the corresponding one in the registry changes. In particular, Maven Central does not delete any known malicious package, but only changes its access status to inaccessible. When a user attempts to download the package, the registry returns an HTTP status code `403 Forbidden`, preventing the user from obtaining it. However, we find that both Huawei mirror and Aliyun mirror have not updated the access status of packages correctly, resulting in 233 and 222 ghost packages, respectively. For example, `com.github.codingandcoding:servlet-api` [81] is a known malicious package but still available for downloading at the above mirrors. Similarly, the Huawei NuGet mirror also has this problem. We find that the *listed* status of 2,358 packages in the Huawei NuGet mirror is inconsistent with the status of corresponding packages in the official registry. Note that these ghost packages are *not* vulnerable to Mirror Package Override Attack (M_1) since their binary files still exist in registries and thus cannot be republished by attackers.

Finally, some mirrors do not synchronously delete a specific version of the package already deleted in the registry. The npm registry allows maintainers to unpublish a specific version without unpublishing the whole package. In particular, since npm handles malicious packages by modifying the package version (i.e., `0.0.1-security` mentioned in R_2), any version synchronization errors will leave packages (that have been officially marked as malicious) to still accessible to mirror users. In particular, we discover that 1,210 packages in the Aliyun npm mirror have different versions from the npm registry. We have verified that they are malicious packages marked by the npm registry.

5.5. Registry Client Related Vulnerabilities

Dependency Confusion Attack (C_1). While virtual repository-side DCA is a well-known attack vector, this paper mainly analyzes registry client-side DCA. Two conditions are required to mount C_1 on registry clients: (1) the registry client supports specifying multiple upstream registries; and (2) when the registry client tries to download a package from one (usually private) registry, it actually downloads the package from another (usually public) registry. We test each registry client by building two upstream registries and checking their official documents. For official clients of all registries, we find that PyPI and NuGet are vulnerable. Go and npm do not meet the first condition; Maven and Cargo do not meet the second. However, we find that Gradle [38], which is a popular third-party registry client for Maven and used by 49% of Java developers [82], is vulnerable.

Particularly, we find that Java developers often use multiple registries (e.g., Maven Central, JCenter [83], and JitPack [84]) at the same time. So Gradle users may suffer this threat even using multiple public registries. On GitHub, we find that 485,000 projects use both Maven Central and JCenter, and 123,000 projects use both Maven Central and JitPack. For example, we find that 435,513 (95.1%) package names in Maven Central do not exist in JitPack. Attackers can publish a malicious package with the same name and higher version number to JitPack, thereby attacking Gradle users who use both Maven Central and JitPack.

Package Version Downgrade Attack (C_2). We find that packages installed by pip (the official registry client of PyPI) are vulnerable to C_2. If two packages pull different versions of a package (denoted as P_{dep}) as dependencies, the later installed version will overwrite the earlier installed version. According to the CVE database [85], 328 packages (denoted as P_{vul}; examples including *requests* and *numpy*) in PyPI have known vulnerabilities that existed in some specific versions. Developers widely use these P_{vul} packages: we find that 21,977 (6%) packages in PyPI directly depend on them. Attackers can publish a lure package, which references a vulnerable version of a P_{vul} package. If attackers can trick users into installing the lure package, the vulnerable version of P_{vul} will overwrite the benign one (if it existed), regardless of any constraint. As a result, this vulnerable version of P_{vul} will be further used by all other packages on the victim host.

5.6. Miscellaneous Vulnerabilities

Package Version Reuse Attack (O_1). We find that PyPI used to allow version reuse, but this was in 2015 [49]. Similarly, npm banned version reuse in 2014 [50]. However, during the evaluation, we find that 559 packages in npm have experienced 562 version reuse events. These packages are republished with the same version numbers, but their contents have been changed. In particular, we find that 557 (99.1%) version reuse events happen within one year, and five events have a time interval greater than one year. The package `karyon@1.0.0` has the longest interval (1,109 days), which occurred on April 29th, 2022. We show the details of version reuse events in Figure 8 and list some packages with version reuse in Table 7 in the Appendix. However, we are not able to reproduce the version reuse case, as the checking logic is implemented on the server side. We have contacted several maintainers for those version-reused packages. They all mentioned that the publishing process of those packages is as usual without any special operations.

Package Tampering Attack (O_2). We observe two phenomena that could lead to package tampering attacks. First, some mirrors modify the package content, and thus users cannot verify the integrity of the package. In particular, the Huawei Maven mirror will delete the checksum file of all packages and modify/add content to the package metadata (i.e., the maven-metadata.xml) file. In addition, we find that the Aliyun npm mirror not only makes necessary changes to the metadata (i.e., changes the package's download address), but also modifies the created and modified time in the metadata. These changes make it hard for users to notice if a package has been tampered with. Second, we find that many mirrors, including Huawei Maven, Aliyun PyPI, Douban PyPI, Tencent npm, and Huawei NuGet, can still be accessed through HTTP. It is insecure and vulnerable to many well-known attacks, such as a MITM attack.

6. Countermeasures and Disclosure

6.1. Defense Practices

We propose several defensive practices to reduce the occurrence of discovered threats. While fully addressing them requires coordination among different stakeholders, the proposed countermeasures here are primitive and preliminary. We believe the affected registries or mirrors should enforce appropriate defensive mechanisms that best fit their systems.

Registries should adopt MFA to reduce account hijacking related attacks. They can also enforce code signing mechanisms so that users can realize potential attacks if the code signature is different. In addition, registries should add more scrutiny to automatically filter out suspicious packages. For example, they should be more careful when handling package name/version reuse to reduce the risk of a UAF attack. For registries that rely on code hosting platforms, they should regularly check user account creation time and project redirecting conditions on those platforms. Moreover, we suggest registries provide package operation information (e.g., package unpublish) via public APIs so that mirrors can synchronize consistently.

Registry mirrors should ensure that all information in the mirror (e.g., packages, metadata files, status) are consistent with the upstream registry, especially for registries that support package deletion or version deletion. They can also regularly check and compare with the upstream registry to detect and avoid any inconsistencies. Mirrors should update/upgrade their repository mirroring software in time and they should avoid exposing private packages.

Registry clients should inform users if attacks related to package version confusion and dependency confusion could occur. They should also compare the input package name with popular packages to detect and alert potential typos.

Finally, *Package maintainers* should enable MFA and code signing for their accounts and published packages when possible. If they decide to delete accounts/packages or transfer projects, they should inform users through package metadata, GitHub readme file, and others.

6.2. Disclosure and Response

We have initiated responsible disclosure with five relevant official registries (note that the official NuGet registry is not vulnerable to most threats) and twelve mirrors maintained by five entities to help them mitigate the detected threats. In detail, we have reported all issues found in this paper and the mitigation practices proposed in Section 6.1 to these entities. We have received multiple positive feedback. The npm team has confirmed multiple issues, including package use-after-free (U_1), package maintainer account hijacking (U_2), dangling references (R_1), and republished package inheritance problem (e.g., download count). At the same time that the npm team attempts to fix the above issues, we are still working with them to find the cause for version reuse. Furthermore, they rewarded us with a bug bounty of \$2,000. The Go team has confirmed package maintainer account hijacking (U_2) and package redirection hijacking (U_3) issues and is currently working on mitigation measures for them. Moreover, the PyPI team has confirmed package maintainer account hijacking (U_2) and package use-after-free (U_1) issues. For the former, PyPI has introduced the un-verify email mechanism to alleviate it partially. The Aliyun mirror and Tsinghua mirror have acknowledged our report and confirmed that those issues are caused by their outdated repository mirroring software. The Aliyun mirror has fixed the case

TABLE 6. The comparison of the attacks from our paper and prior works.

Attack Vector	Prior Works	Our Contributions
U_1 - Package UAF	Mentioned by Ohm et al. [22].	Measure the problem in PyPI and npm on a large scale.
U_2 - Package Maintainer Account Hijacking	Duan et al. [7] indicated that account hijacking is the second most exploited vectors. Zimmermann et al. [9] studied traditional methods (e.g., weak/compromised passwords and social engineering); Zahan et al. [8] studied expired email domain for npm.	Disclose a new attack caused by deleted third-party website accounts, affecting Go and Cargo.
T_1 - Typosquatting	Duan et al. [7] uncovered many malicious packages with typosquatting package names.	Analyze and measure scoped package names and typosquatting across ecosystems.
R_1 - Package Reference	Zimmermann et al. [9] found that package dependencies grows rapidly in npm. Duan et al. [7] showed that reverse dependencies can amplify the impact of malicious packages.	Uncover several real attacks on malicious packages; study dangling references in registries and ghost packages in mirrors.
C_1 - Dependency Confusion (DCA)	*Virtual repository-side* DCA has occurred several times [27] [28].	Focus on *registry client-side* DCA, affecting PyPI, NuGet, and Gradle in Maven.
O_2 - Package Tampering	Ohm et al. [22] mentioned the attack on registries.	Discover some registry mirrors are vulnerable.
Mirror Related	Cappos et al. [86] studied security impact of malicious mirrors.	Disclose several new vulnerabilities in benign mirrors.

sensitivity confusion attack (M_2) and ghost package issues. Both Huawei and NJU mirror maintainers confirmed and acknowledged our findings on their corresponding mirrors.

7. Related Work

Software Registry Ecosystem. The software registry ecosystems face many security threats, such as typosquatting [87], package vulnerability [88] [89], account takeover [20] [90] and infrastructure compromise [91]. Many empirical studies have been conducted on different software ecosystems [92] [93] [94] [95]. Particularly, Duan et al. [7] designed a comparative framework to identify security threats in PyPI, npm, and RubyGems. They detected hundreds of malicious packages and found that typosquatting and account hijacking are the two most exploited vectors. They also studied package reference attacks and found that package dependency may amplify attack impact. We extend their study by considering registry mirrors (including package hijacking and reference attacks) and decentralized registries. We also analyze scoped packages typosquatting in registries. Zimmermann et al. [9] studied the account take over attack (U_2) through traditional methods, such as using weak passwords and via social engineering; and Zahan et al. [8] further analyzed this problem in npm caused by expired email domain names. In contrast, we disclose a new account take over attack by exploiting deleted third-party website (e.g., GitHub) accounts.

Meanwhile, with the growing software ecosystem, package dependency is getting more complex and causes potential security issues [96] [97] [98]. Several studies [99] [100] demonstrated that vulnerable packages might be exploited to affect dependent packages further, impacting a large part of the ecosystem. Our research discloses several actual incidents that exploit reference attacks on malicious packages. Finally, little research has been carried out on the security of registry mirrors. Cappos et al. [86] found that attackers can build their own (malicious) mirror to compromise clients, while we disclose several new vulnerabilities in benign mirrors. A detailed comparison with previous works is listed in Table 6.

Resource Reuse and Typosquatting Attack. Resource reuse attacks are dangerous as they are stealthy for users, and thus attract extensive research in recent years, including reused domain names [21] [101] [102] [23], shared TLS certificates [103], reused email addresses [20] [104], recycled phone numbers [105] [106], and reused passwords [107] [108] [109]. Liu et al. [24] analyzed the threat in DNS posed by dangling DNS records, which adversaries can exploit to hijack domains. Lee et al. [105] found that a large fraction of recycled phone numbers is linked to leak login credentials on the web, which could enable account hijacking. Hanamsagar et al. [110] reported that 98% of users reuse their passwords with no changes. In our work, we further extend the resource reuse problems on popular registry ecosystems and identify several new threats that widely exist in the wild.

Many research studies have been conducted on typosquatting, which is a common software supply chain attack [41] [40] [111] [42]. Prior works mainly focus on domain names [112] [113] [114] or a single software ecosystem [115] [9]. In this paper, we collect an enormous number of package name information from multiple software ecosystems and analyze the possible typosquatting attack in these ecosystems. Our experimental result shows that typosquatting remains a significant security threat across different ecosystems.

8. Conclusion

This paper systematically analyzes package-related security vulnerabilities that existed in different package stakeholders of the software registry ecosystems. We identified twelve security threats, including eight new security issues disclosed for the first time. We developed a scanning tool to continuously monitor software registry ecosystems for six popular programming languages, covering millions of packages in six official registries and seventeen corresponding mirrors over a one-year measurement study. Our experimental results show that many packages are vulnerable to identified security threats. We have discussed potential mitigation, reported all security issues to corresponding stakeholders, and received positive responses.

Acknowledgments

We would like to thank the anonymous reviewers for their insightful comments. We also thank Chaoyi Lu for assisting in editing this paper.

This work is partially supported by the National Natural Science Foundation of China (U1836213, U19B2034).

References

[1] "Maven Central Repository Search," https://search.maven.org/.

[2] "npm," https://www.npmjs.com/.

[3] "PyPI · The Python Package Index," https://pypi.org/.

[4] "NuGet Gallery," https://www.nuget.org/.

[5] "Sonatype's 2021 State of the Software Supply Chain," https://www.sonatype.com/resources/state-of-the-software-supply-chain-2021.

[6] "Popular npm Library Hijacked to Install Password-stealers, Miners," https://www.bleepingcomputer.com/news/security/popular-npm-library-hijacked-to-install-password-stealers-miners/.

[7] R. Duan, O. Alrawi, R. P. Kasturi, R. Elder, B. Saltaformaggio, and W. Lee, "Towards Measuring Supply Chain Attacks on Package Managers for Interpreted Languages," in *NDSS*, 2021.

[8] N. Zahan, T. Zimmermann, P. Godefroid, B. Murphy, C. Maddila, and L. Williams, "What are Weak Links in the npm Supply Chain?" in *IEEE/ACM ICSE-SEIP*, 2022.

[9] 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 *USENIX Security*, 2019.

[10] "Changing Your GitHub Username," https://docs.github.com/en/account-and-profile/setting-up-and-managing-your-github-user-account/managing-user-account-settings/changing-your-github-username.

[11] "Transferring a Repository," https://docs.github.com/en/repositories/creating-and-managing-repositories/transferring-a-repository.

[12] "crates.io: Rust Package Registry," https://crates.io/.

[13] "Go Packages," https://pkg.go.dev/.

[14] "Aliyun npm Mirror," https://registry.npmmirror.com/.

[15] "Nexus Repository - Software Component Management," https://www.sonatype.com/products/repository-pro.

[16] "JFrog Artifactory - Universal Artifact Repository Manager," https://jfrog.com/artifactory/.

[17] "NuGet Package Version Reference," https://docs.microsoft.com/en-us/nuget/concepts/package-versioning.

[18] "Malicious NPM Libraries Caught Installing Password Stealer and Ransomware," https://thehackernews.com/2021/10/malicious-npm-libraries-caught.html.

[19] "Cryptocurrency Clipboard Hijacker Discovered in PyPI Repository," https://bertusk.medium.com/cryptocurrency-clipboard-hijacker-discovered-in-pypi-repository-b66b8a534a8.

[20] D. Gruss, M. Schwarz, M. Wübbeling, S. Guggi, T. Malderle, S. More, and M. Lipp, "Use-after-freemail: Generalizing the Use-after-free Problem and Applying It to Email Services," in *AsiaCCS*, 2018.

[21] J. A. Reed and J. Reed, "Potential Email Compromise via Dangling DNS MX," 2020.

[22] M. Ohm, H. Plate, A. Sykosch, and M. Meier, "Backstabber's Knife Collection: A Review of Open Source Software Supply Chain Attacks," in *Springer DIMVA*, 2020.

[23] E. Alowaisheq, "Cracking Wall of Confinement: Understanding and Analyzing Malicious Domain Takedowns," in *NDSS*, 2019.

[24] D. Liu, S. Hao, and H. Wang, "All Your DNS Records Point to Us: Understanding the Security Threats of Dangling DNS Records," in *ACM CCS*, 2016.

[25] "Millions of Phone Numbers Are Being Reused Every Year," https://medium.com/codex/two-thirds-of-mobile-phone-numbers-are-reused-7fbf03e1b88f.

[26] "Recycled Bank Accounts Can Mean Sending Money to the Wrong Person," https://www.theguardian.com/money/2016/dec/17/recycled-bank-accounts-send-money-wrong-person.

[27] "Dependency Confusion: How I Hacked Into Apple, Microsoft and Dozens of Other Companies," https://medium.com/@alex.birsan/dependency-confusion-4a5d60fec610.

[28] "This week in malware—400+ npm packages target Azure, Uber, Airbnb developers," https://blog.sonatype.com/this-week-in-malware-400-npm-packages-target-azure-uber-airbnb-developers.

[29] "Remote code execution on rubygems.org," https://justi.cz/security/2017/10/07/rubygems-org-rce.html.

[30] "Remote code execution on packagist.org," https://justi.cz/security/2018/08/28/packagist-org-rce.html.

[31] "How to take over the computer of any Java (or Clojure or Scala) developer," https://max.computer/blog/how-to-take-over-the-computer-of-any-java-or-clojure-or-scala-developer/.

[32] "The CouchDB API," https://docs.couchdb.org/en/stable/api/index.html.

[33] "NXDOMAIN: There Really Is Nothing Underneath," https://datatracker.ietf.org/doc/html/rfc8020.

[34] "Whois - Client for the WHOIS Directory Service," https://manpages.debian.org/stretch/whois/whois.1.en.html.

[35] "Large-scale npm Attack Targets Azure Developers with Malicious Packages," https://jfrog.com/blog/large-scale-npm-attack-targets-azure-developers-with-malicious-packages/.

[36] "Sonatype Stops Software Supply Chain Attack Aimed at the Java Developer Community," https://blog.sonatype.com/malware-removed-from-maven-central.

[37] "Malicious PyPI Packages with over 10,000 Downloads Taken Down," https://www.bleepingcomputer.com/news/security/malicious-pypi-packages-with-over-10-000-downloads-taken-down/.

[38] "Gradle Build Tool," https://gradle.org.

[39] "How Much Does a Domain Name Cost? Find Out!" https://www.godaddy.com/garage/how-much-domain-name-cost/.

[40] J. Szurdi, B. Kocso, G. Cseh, J. Spring, M. Felegyhazi, and C. Kanich, "The Long 'Taile' of Typosquatting Domain Names," in *USENIX Security*, 2014.

[41] P. Agten, W. Joosen, F. Piessens, and N. Nikiforakis, "Seven Months' Worth of Mistakes: A Longitudinal Study of Typosquatting Abuse," in *NDSS*, 2015.

[42] N. Nikiforakis, M. Balduzzi, L. Desmet, F. Piessens, and W. Joosen, "Soundsquatting: Uncovering the Use of Homophones in Domain Squatting," in *Springer ICS*, 2014.

[43] K. Tian, S. T. Jan, H. Hu, D. Yao, and G. Wang, "Needle in a Haystack: Tracking Down Elite Phishing Domains in the Wild," in *IMC*, 2018.

[44] "Package phishing," http://blog.fatezero.org/2017/06/01/package-fishing/.

[45] "node-ipc npm Package Sabotage to Protest Ukraine Invasion," https://securityaffairs.co/wordpress/129174/hacking/node-ipc-npm-package-sabotage.html.

[46] "Meet the Developers Behind Sonatype's Automated Malware Detection System Securing Open Source Supply Chains," https://blog.sonatype.com/meet-the-developers-behind-sonatypes-automated-malware-detection-system-securing-open-source-supply-chains.

[47] "New npm scanning tool sniffs out malicious code," https://portswigger.net/daily-swig/new-npm-scanning-tool-sniffs-out-malicious-code.

[48] A. Birsan, "Dependency Confusion: How I Hacked Into Apple, Microsoft and Dozens of Other Companies," https://medium.com/@alex.birsan/dependency-confusion-4a5d60fec610, 2021.

[49] D. Stufft, "[Distutils] Closing the Delete File + Re-upload File Loophole," https://mail.python.org/pipermail/distutils-sig/2015-January/025683.html, 2015.

[50] "Npm-Unpublish | Npm Docs," https://docs.npmjs.com/cli/v8/commands/npm-unpublish.

[51] "TIOBE Index," https://www.tiobe.com/tiobe-index/.

[52] "PYPL PopularitY of Programming Language," https://pypl.github.io/PYPL.html.

[53] "Aliyun Maven Repository Central," https://maven.aliyun.com/repository/central.

[54] "Huawei Cloud Maven Public," https://repo.huaweicloud.com/repository/maven/.

[55] "NJU Maven Repository Central," https://doc.nju.edu.cn/books/35f4a/page/maven.

[56] "Aliyun Python Package Index," https://mirrors.aliyun.com/pypi/simple/.

[57] "Douban Python Package Index," http://pypi.doubanio.com/simple.

[58] "NJU Python Package Index," https://mirrors.tuna.tsinghua.edu.cn/pypi/web/simple.

[59] "Tsinghua Python Package Index," https://mirrors.tuna.tsinghua.edu.cn/pypi/web/simple.

[60] "Huawei npm Mirror," https://repo.huaweicloud.com/repository/npm/.

[61] "Tencent npm Mirror," https://mirrors.cloud.tencent.com/npm/.

[62] "NJU npm Mirror," https://doc.nju.edu.cn/books/35f4a/page/npm.

[63] "Tsinghua Cargo Mirror," https://mirrors.tuna.tsinghua.edu.cn/git/crates.io-index.git.

[64] "SJTU Cargo Mirror," https://mirrors.sjtug.sjtu.edu.cn/git/crates.io-index/.

[65] "USTC Cargo Mirror," https://mirrors.ustc.edu.cn/crates.io-index.

[66] "Huawei NuGet Mirror," https://repo.huaweicloud.com/repository/nuget/v3/index.json.

[67] "Microsoft Azure NuGet Mirror for China," https://nuget.cdn.azure.cn/v3/index.json.

[68] "Qiniu Cloud - Goproxy.cn," https://goproxy.cn/.

[69] N. P. Hoang, A. A. Niaki, J. Dalek, J. Knockel, P. Lin, B. Marczak, M. Crete-Nishihata, P. Gill, and M. Polychronakis, "How Great is the Great Firewall? Measuring China's DNS Censorship," in *USENIX Security*, 2021.

[70] "Sonatype JIRA," https://issues.sonatype.org.

[71] "OSSRH Guide - The Central Repository Documentation," https://central.sonatype.org/publish/publish-guide/.

[72] "Users API | GitHub," https://docs.github.com/en/rest/reference/users#get-a-user.

[73] "Users API | GitLab," https://docs.gitlab.com/ee/api/users.html.

[74] J. Szurdi and N. Christin, "Email Typosquatting," in *IMC*, 2017.

[75] G. Liu, X. Gao, H. Wang, and K. Sun, "Exploring the Unchartered Space of Container Registry Typosquatting," in *USENIX Security*, 2022.

[76] "Deleting Your User Account," https://docs.github.com/en/account-and-profile/setting-up-and-managing-your-github-user-account/managing-user-account-settings/deleting-your-user-account.

[77] "PyPI's XML-RPC methods," https://warehouse.pypa.io/api-reference/xml-rpc.html.

[78] "New Package Moniker Rules," https://blog.npmjs.org/post/168978377570/new-package-moniker-rules.html.

[79] V. I. Levenshtein, "Binary Codes Capable of Correcting Deletions, Insertions, and Reversals," in *Soviet Physics Doklady*, 1966.

[80] "Scoped Packages," https://docs.npmjs.com/cli/v8/using-npm/scope.

[81] S. S. R. Team, "Sonatype Stops Software Supply Chain Attack Aimed at the Java Developer Community," https://blog.sonatype.com/malware-removed-from-maven-central.

[82] "JetBrains 2021 Dev Ecosystem Survey," https://www.jetbrains.com/lp/devecosystem-2021/java/.

[83] "JCenter Repository," https://mvnrepository.com/repos/jcenter.

[84] "JitPack - Publish JVM and Android libraries," https://jitpack.io/.

[85] "MITRE CVE List," https://cve.mitre.org/cve/search_cve_list.html.

[86] J. Cappos, J. Samuel, S. Baker, and J. H. Hartman, "A Look in the Mirror: Attacks on Package Managers," in *ACM CCS*, 2008.

[87] N. P. Tschacher, "Typosquatting in Programming Language Package Managers," Ph.D. dissertation, 2016.

[88] J. Wetter and N. Ringland, "Understanding the Impact of Apache Log4j Vulnerability," https://security.googleblog.com/2021/12/understanding-impact-of-apache-log4j.html.

[89] "[CVE-2019-15224] Version 1.6.13 published with malicious backdoor. · Issue #713 · rest-client/rest-client," https://github.com/rest-client/rest-client/issues/713.

[90] P. Doerfler, M. Marincenko, J. Ranieri, Y. Jiang, A. Moscicki, D. McCoy, and K. Thomas, "Evaluating Login Challenges as a Defense Against Account Takeover," in *WWW*, 2019.

[91] "Postmortem for Malicious Packages Published on July 12th, 2018," https://eslint.org/blog/2018/07/postmortem-for-malicious-package-publishes.

[92] J. Kabbedijk and S. Jansen, "Steering Insight: An Exploration of the Ruby Software Ecosystem," in *Springer ICSOB*, 2011.

[93] D. M. German, B. Adams, and A. E. Hassan, "The Evolution of the R Software Ecosystem," in *IEEE CSMR*, 2013.

[94] A. Decan, T. Mens, and E. Constantinou, "On the Impact of Security Vulnerabilities in the npm Package Dependency Network," in *MSR*, 2018.

[95] K. Manikas, "Revisiting Software Ecosystems Research: A Longitudinal Literature Study," *Journal of Systems and Software*, 2016.

[96] C.-A. Staicu, M. Pradel, and B. Livshits, "SYNODE: Understanding and Automatically Preventing Injection Attacks on NODE.JS," in *NDSS*, 2018.

[97] J. C. Davis, E. R. Williamson, and D. Lee, "A Sense of Time for JavaScript and Node.js: First-Class Timeouts as a Cure for Event Handler Poisoning," in *USENIX Security*, 2018.

[98] C.-A. Staicu and M. Pradel, "Freezing the Web: A Study of ReDoS Vulnerabilities in JavaScript-based Web Servers," in *USENIX Security*, 2018.

[99] J. Hejderup, *In Dependencies We Trust: How Vulnerable Are Dependencies in Software Modules?* Delft University of Technology, 2015.

[100] A. Decan, T. Mens, and P. Grosjean, "An Empirical Comparison of Dependency Network Evolution in Seven Software Packaging Ecosystems," *Empirical Software Engineering*, 2019.

[101] M. Squarcina, M. Tempesta, L. Veronese, S. Calzavara, and M. Maffei, "Can I Take Your Subdomain? Exploring Same-Site Attacks in the Modern Web," in *USENIX Security*, 2021.

[102] T. Lauinger, A. Chaabane, A. S. Buyukkayhan, K. Onarlioglu, and W. Robertson, "Game of Registrars: An Empirical Analysis of Post-Expiration Domain Name Takeovers," in *USENIX Security*, 2017.

[103] M. Zhang, X. Zheng, K. Shen, Z. Kong, C. Lu, Y. Wang, H. Duan, S. Hao, B. Liu, and M. Yang, "Talking with Familiar Strangers: An Empirical Study on HTTPS Context Confusion Attacks," in *ACM CCS*, 2020.

[104] H. Hu, P. Peng, and G. Wang, "Characterizing Pixel Tracking through the Lens of Disposable Email Services," in *IEEE S&P*, 2019.

[105] K. Lee and A. Narayanan, "Security and Privacy Risks of Number Recycling at Mobile Carriers in the United States," in *IEEE eCrime*, 2021.

[106] A. McDonald, C. Sugatan, T. Guberek, and F. Schaub, "The Annoying, the Disturbing, and the Weird: Challenges with Phone Numbers as Identifiers and Phone Number Recycling," in *ACM CHI*, 2021.

[107] Q. Li, *A Survey Study of Password Setting and Reuse*. University of Delaware, 2020.

[108] A. Das, J. Bonneau, M. Caesar, N. Borisov, and X. Wang, "The Tangled Web of Password Reuse," in *NDSS*, 2014.

[109] K. C. Wang and M. K. Reiter, "How to End Password Reuse on the Web," in *NDSS*, 2019.

[110] A. Hanamsagar, S. Woo, C. Kanich, and J. Mirkovic, "How Users Choose and Reuse Passwords," *Information Sciences Institute*, 2016.

[111] M. T. Khan, X. Huo, Z. Li, and C. Kanich, "Every Second Counts: Quantifying the Negative Externalities of Cybercrime via Typosquatting," in *IEEE S&P*, 2015.

[112] N. Nikiforakis, S. Van Acker, W. Meert, L. Desmet, F. Piessens, and W. Joosen, "Bitsquatting: Exploiting Bit-flips for Fun, or Profit?" in *WWW*, 2013.

[113] P. Kintis, N. Miramirkhani, C. Lever, Y. Chen, R. Romero-Gómez, N. Pitropakis, N. Nikiforakis, and M. Antonakakis, "Hiding in Plain Sight: A Longitudinal Study of Combosquatting Abuse," in *ACM CCS*, 2017.

[114] T. Moore and B. Edelman, "Measuring the Perpetrators and Funders of Typosquatting," in *Springer FC*, 2010.

[115] D.-L. Vu, I. Pashchenko, F. Massacci, H. Plate, and A. Sabetta, "Typosquatting and Combosquatting Attacks on the Python Ecosystem," in *IEEE EuroS&PW*, 2020.

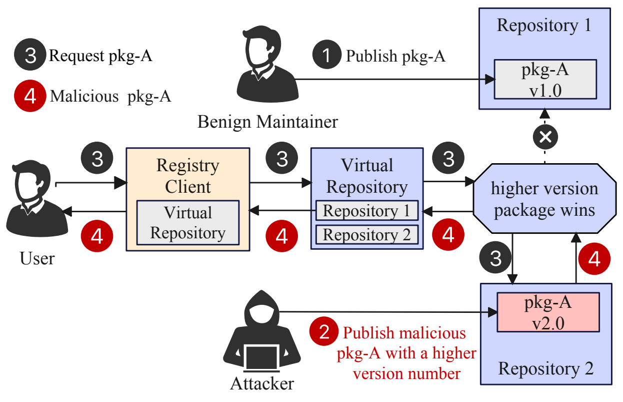
Appendix

Additional Tables and Figures

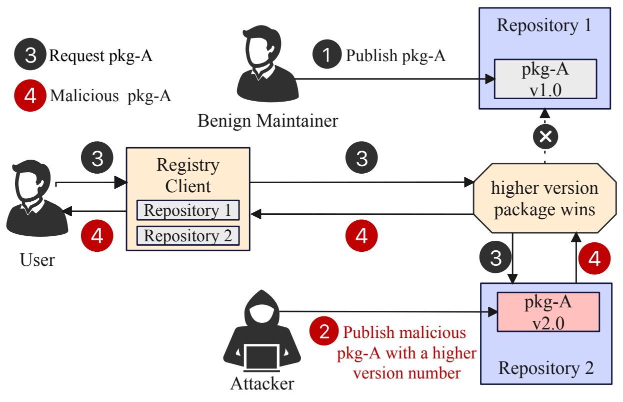
TABLE 7. Five pairs of package experience the version reuse problem in npm. The table shows their first and second publish times as well as hash values.

Package Name	Previous Version			Current Version		
	Version	Publish Time	Hash Value*	Version	Publish Time	Hash Value*
tinyburg	0.0.1	2021-04-26 18:44:52	969de60ee5	0.0.1	2021-09-15 16:29:29	60e14cc05a
ladder-ui	0.1.1	2021-07-15 09:28:04	2d721db168	0.1.1	2021-10-26 01:40:50	40831fff7c
bmyc	0.0.1	2021-09-03 13:25:41	41491a3f9a	0.0.1	2021-09-16 20:18:15	a5db491024
cjkui	1.0.3	2021-09-09 12:00:09	aa54c2dcca	1.0.3	2021-10-11 03:47:49	4f81261eaf
03-custom	1.0.0	2020-11-30 09:43:31	ed1edb443e	1.0.0	2021-11-13 02:20:59	a68b33a11e

* Showing the first 10 characters of hex representation of each hash value.



(a) Virutal repository-side dependency confusion attack. *Virutal repository* decides which version to use.



(b) Registry client-side dependency confusion attack. *Registry client* decides which version to use.

Figure 9. Overview of two scenarios of Dependency Confusion Attack (C_1). The figure shows what roles registry clients and virtual repositories play in the attack.