HOW EFFECTIVELY DO CURRENT OPEN SOURCE TOOLS DEAL WITH PACKAGE CONFUSION ATTACKS?

Humaira Fasih Ahmed Hashmi
hfhashmi@ucdavis.edu
University of California
Davis, California, USA

Sarah Yuniar
sayuniar@ucdavis.edu
University of California
Davis, California, USA

Parnian Shabani Kamran
pkamran@ucdavis.edu
University of California
Davis, California, USA

Abstract

The open-source software (OSS) ecosystem provides developers with access to a vast number of third-party repositories that can be easily integrated into their projects. Package managers such as npm (JavaScript), PyPI (Python), and RubyGems (Ruby) facilitate the seamless discovery and installation of third-party packages, enhancing development efficiency. However, this convenience also increases the risk of software supply chain security threats, such as package confusion attacks. In such attacks, adversaries upload malicious packages with names similar to popular ones, misleading developers into installing unintended and potentially harmful dependencies.

In this work, we first evaluate the efficacy of TypoMind, a research tool designed for detecting typosquatting attacks, within the PyPI ecosystem. We then enhance OSSGadget, an open-source tool developed by Microsoft, by integrating new typosquatting detection rules inspired by TypoMind. Finally, we assess the effectiveness of these improvements in strengthening OSSGadget's ability to detect malicious typosquatting packages.

Keywords

Typosquatting, Package Confusion, Open Source Software Security (OSS)

1 Introduction

Package confusion attacks are a type of software supply chain attack where a person is misled into installing an unintended package, which may be malicious. Such packages are also sometimes called *typosquatting* packages. Typosquatting packages are intentionally named to closely resemble the names of more popular packages, misleading users who may not notice the differences during installation. Since the package management ecosystems are open-source, launching this attack is convenient, which makes effective detection and countermeasures even more important. Previous research [2][3] has identified instances of typosquatting packages in npm, PyPI, and RubyGems ecosystems, and more recently, there has been an example [5] in the Rust ecosystem as well.

Our work presents an updated study that focuses on analyzing the effectiveness of open source tools that deal with such attacks on the pypi ecosystem, and improves one of them. We found OSSGadget [1], Typomind [2], TypoGard [3] or SpellBound [4] , and shortlisted OSSGadget and Typomind for our study. OSSGadget is a Microsoft tool; Typomind and TypoGard are research tools for detecting and flagging typosquatted packages to developers. We norrow down our analysis to Typomind, as it is an improved version of TypoGard and some authors are the same across both papers. Although Neupane et al. [2] comparatively studied across the three tools, their comparison is limited to the npm ecosystem only. Furthermore, OSSGadget has been improved by the maintainers since their work was published. Overall, our contribution involves the following:

1) We reproduce the study performed by Neupane et al. [2] on the current-state of the PyPI ecosystem, 1.5 years after their work was published.

2) We improve OSSGadget typosquatting package detection in cases where the package has a prefix/suffix added to or removed from its name by 38.6% by adapting logics used by Typomind for OSS-Gadget, resulting in a reduction of the total amount of false positives.

3) We speed up OSSGadget detection by creating and using a fixed dataset of popular/unpopular Python packages.

2 Dataset

The Python Package Index, or PyPI, is an online repository for Python packages with over 10,000,000 weekly package downloads. Tools in the wild exist, such as PyPI Stats, that are able to fetch information about the PyPI ecosystem, such as metadata of individual packages and top recent downloads. But PyPI Stats explicitly states that it cannot fetch data for every Python package due to the large volume of packages. Indeed, with almost 600,000 total packages on PyPI, for large data processing like our purposes, PyPI and PyPI Stats recommend accessing PyPI's public datasets via BigQuery, a Google Cloud service that stores large datasets and allows queries over them.

To gather a list of all Python packages with their weekly download count, we constructed an SQL query that aggregated all downloads on PyPI over the past week. Initially, we used the `file.project` field to represent a package's name, but we discovered that the dataset replaced all periods (.) in package names with dashes (-), e.g. `zope.interface` became `zope-interface`. To get an accurate list of package names, therefore, we instead extracted the package name from the `file.filename` field instead, e.g. `zope.interface` from `zope.interface-7.2.whl`.

Despite our best attempts at sanitizing the resulting list, however, duplicates still arose. This is due to inconsistent naming conventions with various versions of the same package; for example, earlier versions of `zope.interface` are uploaded under the filename `zope_interface-VERSION.whl` while more recent versions are uploaded under the filename `zope.interface-VERSION.whl`. However, we felt this was an acceptable compromise.

In total we collected a list of 575,745 packages along with their download counts. As with *Typomind* [2], we bisected this list into popular packages, those with more than 15,000 weekly downloads,

and unpopular packages, those with 15, 000 or less weekly downloads. Within these groups, there were 11, 241 and 564, 504 packages, respectively.

Due to hardware limitations, however, we were only able to process the top 1, 000 popular packages out of the 11, 241 using *Typomind*. To keep our analysis consistent, we therefore also used the same 1, 000 popular packages in OSSGadget. For both tools, we compared these 1, 000 popular packages against the complete set of 564, 504 unpopular packages.

3 Methodology and Results

We analyzed the efficacy of *OSSGadget* and *Typomind* using the dataset described in 2. *OSSGadget* was modified by implementing the prefix/suffix augmentation rule introduced by Neupane et al. [2], and its ability to detect typosquatting packages was evaluated after the modification.

To address hardware resource limitations, a fixed dataset of popular and unpopular Python packages was used, which improved *OSSGadget*'s runtime. The impact of adding new detection rules on *OSSGadget*'s precision was evaluated using *Typomind*'s results as the ground truth, as *Typomind* can flag more typosquatting packages than *OSSGadget*'s. We also report the overlap percentage of Vanilla OSSGadget and our modified OSSGadget version (after adding the PrefixSuffix Augmentation rule) with Typomind and find a huge improvement of 38.6%.

We also examined the overlap in packages detected by *Typomind* and *OSSGadget*. Specifically, typosquatting packages flagged by *OSSGadget* were compared against those reported by *Typomind*. For matches, the signal types were reviewed and labeled consistently with *Typomind*'s conventions. Details of how *Typomind*'s rule was customized and integrated with *OSSGadget* are explained in section 3.1.

Typomind's results indicated that the *OSSGadget* version used in their experiments could not detect scope confusion in the npm ecosystem. However, during the reproducibility phase, it was observed that the current version of *OSSGadget* supports package confusion detection. This feature is discussed in section 3.2.

3.1 Prefix/Suffix Augmentation Rule

To modify *OSSGadget* and incorporate the prefix/suffix augmentation rule, we adapted the approach used by *Typomind* with some changes. First, each token in the package name is sanitized by replacing dots with dashes (e.g., *zope.publisher* becomes *zope-publisher*). Following this, the package name is split into segments using spaces as delimiters instead of dashes (e.g., *mkdocs-material* becomes *mkdocs, material*).

Next, a segmentation process is applied to each segment in the list. This process splits each segment into meaningful parts based on a list of tokens generated by running an NLP lemmatizer on all available package names in the PyPI ecosystem. Segmentation is performed in both forward and backward directions, returning the most meaningful tokens from both passes (e.g., *{mkdocs, material}* becomes *{mk, docs, material}*).

Finally, the list of tokens is permuted, and the resulting names are checked against a list of unpopular PyPI packages. If a match is found, the package is flagged as a potential typosquat. Using this method, we improved OSSGadget's overlap with Typomind from 267 in the *Vanilla OSSGadget* [as seen in **1a**] to 2, 276 in the modified version [**1b**] in the Prefix/Suffix Augmentation rule/category. This can be observed by looking at the orange stacked bar titled **OG Overlap** in both figures for the prefix/suffix augmentation rule. Furthermore, since multiple signals/rules can be triggered by a typosquated package, we observe that our modification increases the overlap from 30 in *Vanilla OSSGadget* to 243 in our modified OSSGadget version within the **nonsemantic substitution** category as well.

To present the improvement more clearly, we compared the results against *Typomind*, our ground truth, which flagged 5, 206 packages. We use **eq.1** for measuring the overlap percentage. Plugging in the numbers given above for both OSSGadget versions, we find that our modified OSSGadget has an overlap of 43.7% with Typomind Prefix/Suffix Augmentation rule, while the Vanilla OSSGadget version only has an overlap of 5.1%.

$$\text{overlap}_{\text{signal}} = \frac{\text{No. of packages}_{\text{detected by OSSGadget}}}{\text{No. of packages}_{\text{detected by Typomind}}} \times 100 \quad (1)$$

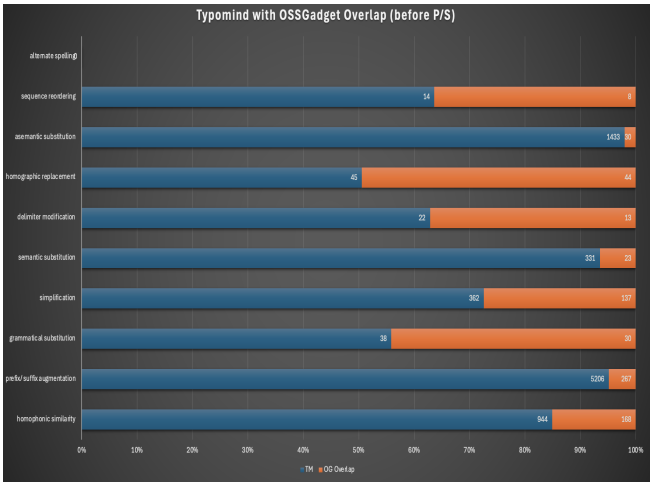
Despite this improvement, the overlap percentage of our modified *OSSGadget* remains lower than that of *Typomind*. This is primarily because *Typomind* applies an NLP lemmatizer to each segment during segmentation, a step we omitted to reduce runtime overhead and accommodate hardware limitations. We anticipate that reintroducing this lemmatization step for matching segments of package names could further enhance the accuracy of *OSSGadget*.

Another point to note, however, is that the total number of typosquatting packages that are detected by the newly added prefix/suffix augmentation rule in *OSSGadget* [as shown in **Table 1** in the rule labelled "Mutated package with prefix/suffix generated"] is 5, 969, which is more than *Typomind's* detection rate of 5, 206 [reported in **Table 2**].

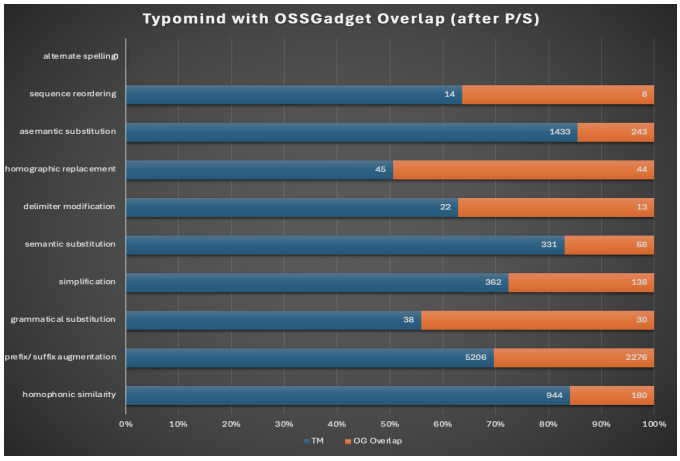
3.2 Scope Confusion Rule

As part of our contributions, we initially planned to implement scope confusion in OSSGadget, as defined by Neupane et al. [2]. However, during our investigation, we discovered that OSSGadget had already been updated to include two new mutators implementing this exact functionality. This update occurred while Neupane et al.'s [2] work was under review for the 32nd USENIX Security Symposium, as evidenced by OSSGadget's Git history and the paper's review/acceptance timeline.

Additionally, the concept of *scope* does not exist in Python as implemented in Typomind [2] and OSSGadget. Both tools support the style enforced in npm i.e `@<scope>/<package-name>` whereas Python package names are typically flat. while namespaces can be simulated using namespace packages (e.g., mycompany.mypackage), these are more about directory structure and import organization rather than a scoping mechanism enforced at the package manager level (e.g., PyPI). Thus, scope confusion is not relevant for pypi as a possible package confusion attack vector, and we omit its evaluation from our results.



(a) Vanilla OSSGadget



(b) OSSGadget with our improvement for Prefix-Suffix Augmentation applied

Figure 1: Typomind matches alongside OSSGadget overlapping matches on a dataset of top $1,000$ popular packages from PyPI. "Match" refers to each `<target package, confuser/typosquatting package>` pair flagged by a tool.

Table 1: OSSGADGET Data Summary

Type of Mutation	Count
Character Added	1203
Containing Character Added	273
Letter Duplicated	77
Mutated package with prefix/suffix generated	5969
Character Replaced	2463
Close Letters on QWERTY Keyboard	547
Letter Duplicated and Replaced	59
Double hit character	99
Character Removed	411
Suffix Removed	52
Ascii Homoglpyh	213
Prefix Added	40
Swap Vowel	262
Letters Swapped	61
Bit Flips	132
Separated Section Removed	193
Separator Removed -	16
Suffix Added	114
String Substituted	10
Separator Changed	11
Total	**12205**

Table 2: TYPOMIND Data Summary

Type of Mutation	Count
Homophonic Similarity	944
Prefix/Suffix Augmentation	5206
Grammatical Substitution	38
Simplification	362
Semantic Substitution	331
Delimiter Modification	22
Homographic Replacement	45
Nonsemantic Substitution	1433
Sequence Reordering	14
Total	**8395**

3.3 Data Collection

We ran both tools on an AMD Ryzen 5 5500 Processor after making modifications to OSSGadget and Typomind for the sake of performance and data collection, while ensuring their base functionality remained unchanged.

To speed up the performance of OSSGadget, we modified the code that confirms the existence of a package to query a local list of Python packages rather than the PyPI website; this is due to the fact that confirming the existence of a package in a list of packages in memory can be reduced to $O(1)$ time when using a hashed structure—in our case, a HashSet in C#. This is significantly faster than OSSGadget's original function, which checked every possible permutation of a base package by checking for a valid HTTP response from its corresponding PyPI webpage (if it exists). This same optimization did not apply to Typomind since it already uses a local file to check for package existence.

With these optimizations, we were able to reduce OSSGadget's performance from several hours to less than 20 minutes. However, we were unable to reduce Typomind's runtime, hence why we opted for taking only the most popular $1,000$ packages, which still took approximately 10 hours.

We also modified Typomind's logic for outputting its results. In OSSGadget's case, it has a built-in feature to write its output into a file, while for Typomind, we had to modify the main script to write its output to a file; we did this without changing the underlying functionality of Typomind.

We then used Python in combination with Regex to parse the resulting output files, extracting the base package name, the confuser package name, and the signals triggered that led to the typosquat detection. We aggregated these results into a standardized format in *csv* files that could then be processed and counted using spreadsheet programs or other Python scripts.

4 Limitations

In this section we discuss the limitations of our current experiment.

Hardware resources: One of the main limitations of our current experiment is that we had limited hardware resources that could not support running our experiment with all the 600, 000 packages in PyPi. As a result, we had to limit our experiment to the top 1, 000 popular packages considering popularity with a similar threshold of 15, 000 downloads per week, as introduced by the authors of *TypoGard* [3]. The reason is that package name typosquatting occurs for the most popular packages.

Restriction on using NLP tools: As we discussed in section 3.1,*Typomind* uses an NLP tool for segmentation and tokenization; However, due to the runtime overhead and limited resources we used pre-processed data generated by the same NLP tool for lemmatizing package names in PyPI ecosystem and skipped using another NLP tool during the segmentation step. Removing this step causes a decrease in the number of packages that can be detected by our version compared to *Typomind*. As a result, in our approach, we do not use a lemmatizer for each meaningful segment of the tokens in a package name.

False positives in detecting popular packages: While one of our intuitions in this experiment was to reduce the number of false positives detected by *OSSGadget* due to the time limitation, we could not add this section to all mutators implemented in *OSSGadget*.

Using fixed list of popular and unpopular packages: Due to hardware resource limitations, we used a fixed set of popular and unpopular packages. While this fixed list improves the runtime and assist us in performing the tools on our current experimental setup and also reduces the number of false positives in OSSGadget (as the tool only goes through the unpopular packages list), it does not use an updated list of the packages available on PyPI ecosystem.

5 Data/Artifact Release

Our data preprocessor and analyzer for the evaluation of the typosquat tools is available on https://github.com/sarahayu/typosquat-project.git and our version of *OSSGadget* with the results are available on https://github.com/Parniaan/oss-gadget/tree/main

6 Conclusion

In this work, we expand the study by Neupan et al. [2] to another popular package management ecosystem - pypi. We evaluated the efficacy of the updated OSSGadget with Typomind. We investigated that OSSGadget have improved over the years (it currently detects some cases of Sequence Reordering category, which did not exist in the original study). Our results suggest that there is still room for OSSGadget [1] improvement. However, the question of whether the additional cases that Typomind detects are genuinely confusing to users needs to be explored more.

Acknowledgments

This report was conducted as the final project for ECS 235A: Computer and Information Security, taught by Professor Matt Bishop at UC Davis in Fall 2024. The authors thank Professor Bishop for his valuable guidance and support throughout this project.

References

[1] Microsoft Inc. 2020. Ossgadget: Collection of tools for analyzing open source packages. https://github.com/microsoft/OSSGadget. Accessed: 12-13-2024.

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

[3] Matthew Taylor, Ruturaj Vaidya, Drew Davidson, Lorenzo De Carli, and Vaibhav Rastogi. 2020. Defending Against Package Typosquatting. In *Network and System Security*, Mirosław Kutyłowski, Jun Zhang, and Chao Chen (Eds.). Springer International Publishing, Cham, 112–131.

[4] Matthew Taylor, Ruturaj Vaidya, Drew Davidson, Lorenzo De Carli, and Vaibhav Rastogi. 2020. SpellBound: Defending Against Package Typosquatting. doi:10.48550/arXiv.2003.03471 arXiv:2003.03471 [cs.CR]

[5] Phylum Research Team. 06-24-2023. Rust malware staged on crates.io. https://blog.phylum.io/rust-malware-staged-on-crates-io/. Accessed: 12-13-2024.