Uncovering Similar but Different Packages in PyPI and Potential Security Threats

Sunha Park, Soojin Han, Seunghoon Woo

Korea University

FSE 2026

KOREA UNIVERSITY

SSP Lab
Software Security and Privacy Laboratory

Package Replication

What is package replication?

Code reuse

Reusing code snippets

Common in OSS to improve efficiency

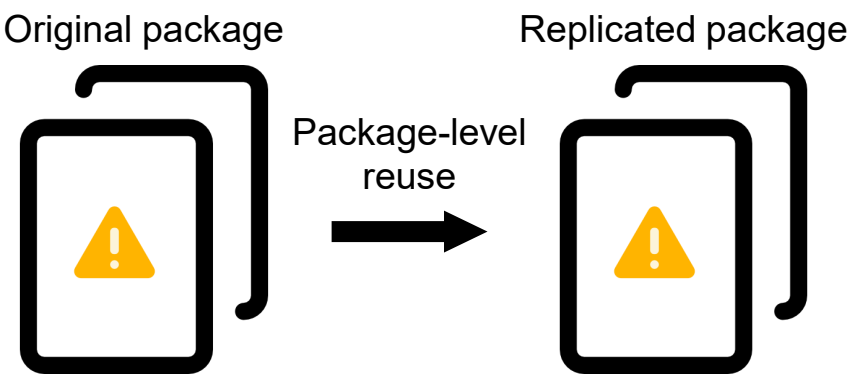
Package replication

Reusing substantial parts of another package

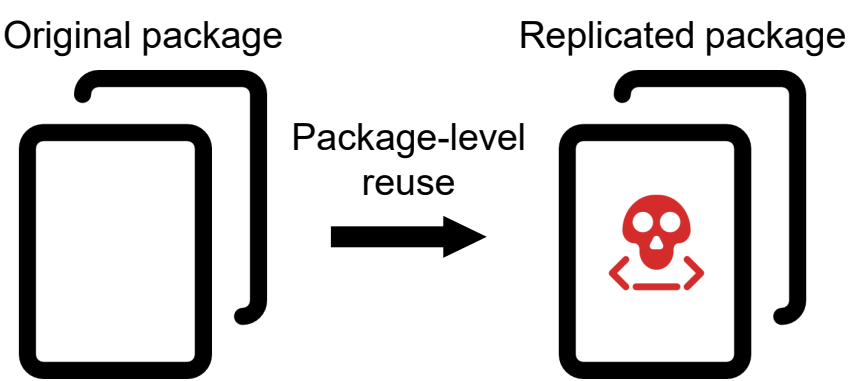
Publishing as a new package

A practical form of package-level reuse

Package Replication



Vulnerability propagation



Malicious code injection

Package replication can introduce **security risks**

Python Package Index

Large and rapidly growing ecosystem

670K packages as of Aug 2025

Package replication remains **underexplored** in Python

Research Goal

A large-scale, in-depth study of package replication in PyPI and its security implications

- **RQ1 - Popularity analysis**

How prevalent is package replication in PyPI?

- **RQ2 - Vulnerability analysis**

How widespread are vulnerabilities in replicated packages?

- **RQ3 - Malware analysis**

What are the characteristics of replicated malicious packages?

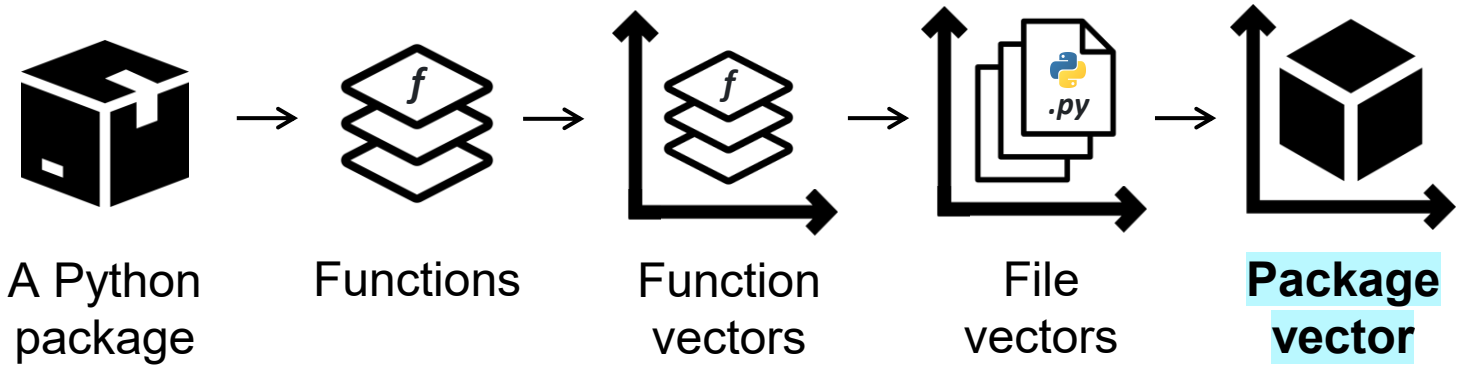
Detection Pipeline

Identify replicated packages using package embedding and clustering, then analyze their characteristics

P1. Package Embedding

Representing each package as a vector

- **Preprocess:** Remove comments and extract functions
- **Embedding:** Generate 256-dimensional function vectors with **CodeT5+**
- **Aggregation:** function → file → package via **mean pooling**

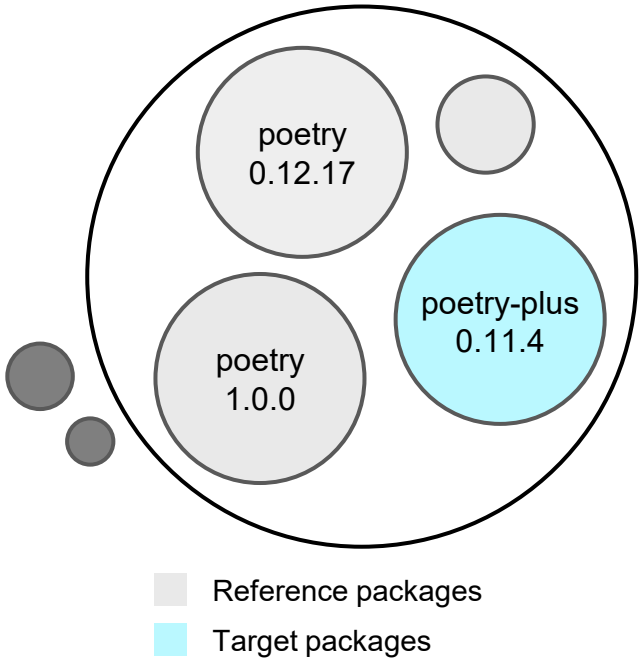


P2. Clustering

Finding package pairs for replication verification

- Reduce dimension of package vectors using **UMAP**
  - Preserves neighborhood structure in a lower-dimensional space
- Group similar packages using **HDBSCAN**
  - Finds dense clusters without predefining the number of clusters
- Extract **package pairs** for file-level verification

Example cluster: Poetry

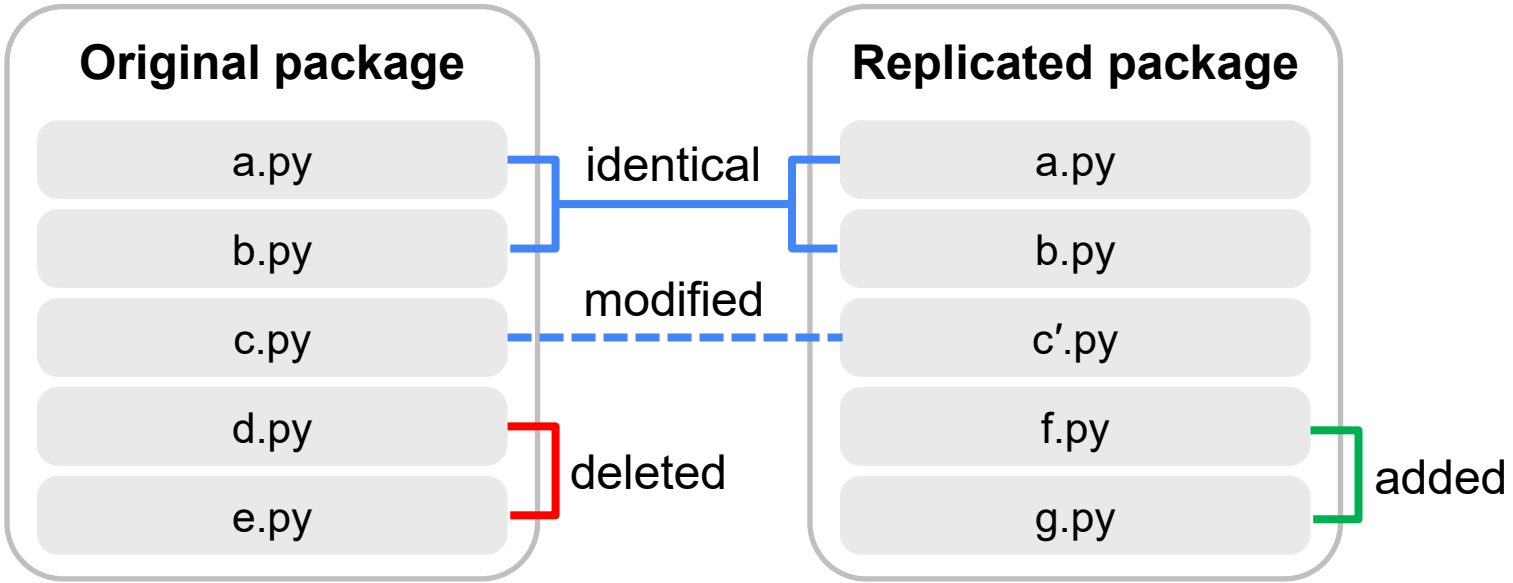


UMAP: McInnes et al., JOSS 2018 · HDBSCAN: McInnes et al., JOSS 2017

P3. Replication Identification

Classifying file relationships

Compare **file paths** and **content hashes** between two packages



P3. Replication Identification

Confirming replication through similarity scoring

Code similarity score:

$$\mathbf{sim}(X, Y) = \frac{\#\mathbf{identical} + \mathbf{0.5} \times \#\mathbf{modified}}{\#\mathbf{total\ files}}$$

#total files: #identical + #modified + #added + #deleted

Replication threshold:

- **≥ 0.9: Highly replicated** - *Near-identical* package-level reuse
- **0.5–0.9: Partially replicated** - *Substantial* package-level reuse
- **< 0.5**: Excluded - Likely weak overlap or fragment-level clones

P3. Replication Identification

Evaluation of detection reliability

No ground truth for package-level replication

→ Compared with SourcererCC and manual labels

Performance of our approach:

Precision 91.58% | Recall 77.02% | **F1-score 83.68%**

- Higher precision and F1-score than SourcererCC
- Validated true positives are used for RQ1 prevalence analysis

For details on ground-truth construction and experimental setup, refer to Section 3

Dataset Collection

Datasets for multi-perspective replication analysis

Dataset	Description	#Packages	#Versions
Popularity	Most downloaded Python packages	2,767	160,317
Vulnerability	Packages with known vulnerabilities	1,072	70,280
Malware	Known malicious packages	2,656	3,883
Recent	Newly uploaded packages	25,407	36,008
Candidate	Randomly sampled PyPI packages	200,737	200,737

Dataset Collection

Datasets for multi-perspective replication analysis

Dataset pairs by analysis goal

Analysis goal	Dataset pair
RQ1. Popularity analysis	Popularity vs. Candidate
RQ2. Vulnerability analysis	Vulnerability vs. Candidate
RQ3. Malware analysis	Popularity vs. Malware / Recent

Key Findings

1,361

Replicated packages from popular PyPI packages

256

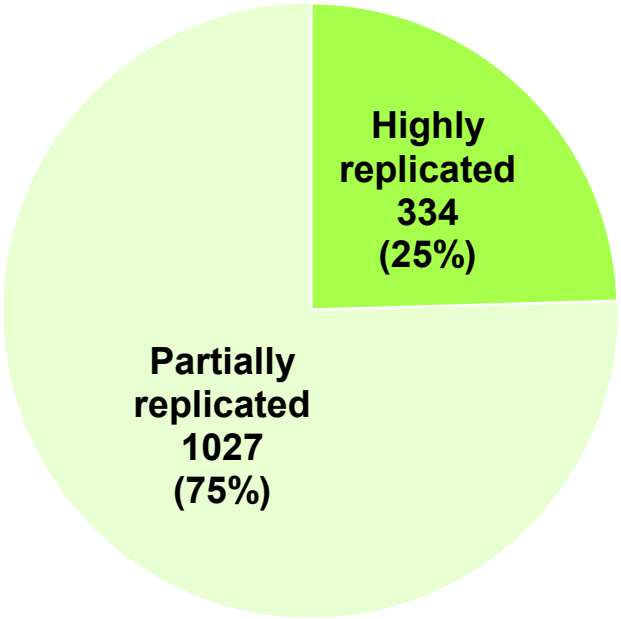
Vulnerable replicated packages preserving known vulnerabilities

186 + 7

Malicious replicated packages, known and newly discovered

RQ1. Prevalence of Replication

Popularity analysis



1,361 replicated packages

Package replication is not rare in PyPI

200,737 candidate packages
compared against 2,767 popular packages
→ **1,361** replicated packages identified

Observed purposes:
customization, vendoring, compatibility, malicious use, and others

RQ1. Replication Characteristics

Replicated packages preserve both code and identity signals

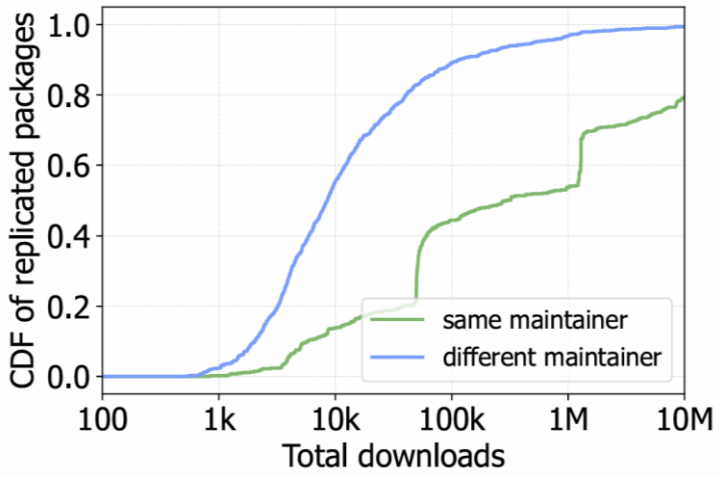
Original package vs. Replicated package

Maintainer	Different account
Name	Similar
Metadata	Shared fields (≥1 matched)
Code	Substantial reuse

- **85%** of categorized files identical
- **60%** (817) under different maintainers
  - **338** with highly similar names
  - **403** with ≥1 shared metadata field

Similar identity signals make replicated packages hard to distinguish from the original

RQ1. Ecosystem Exposure



Replication reaches real users in the PyPI ecosystem

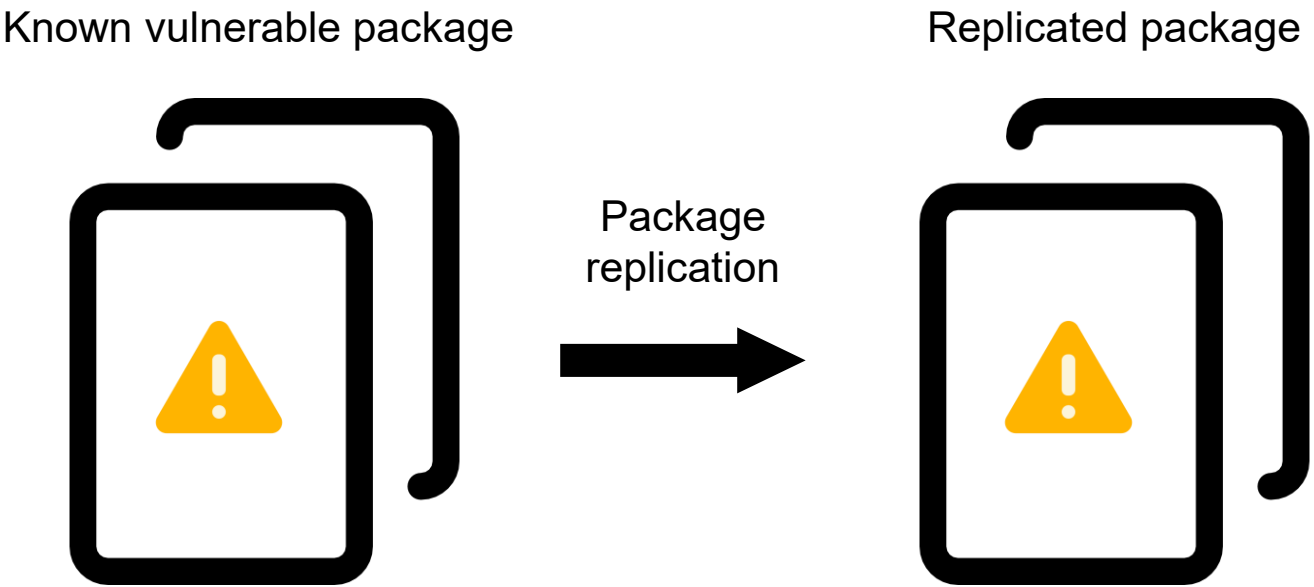
Same maintainer

- **303** over 100K downloads

Different maintainer

- **89** over 100K downloads
- **26** over 1M downloads

RQ2. Vulnerability Analysis

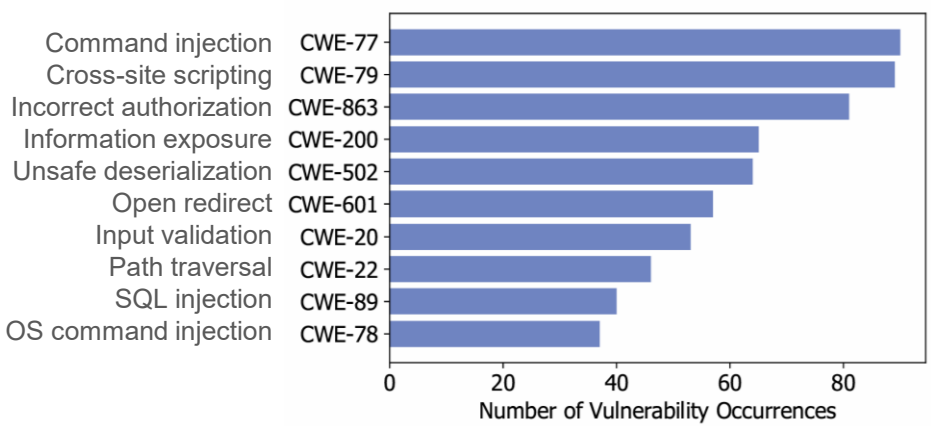


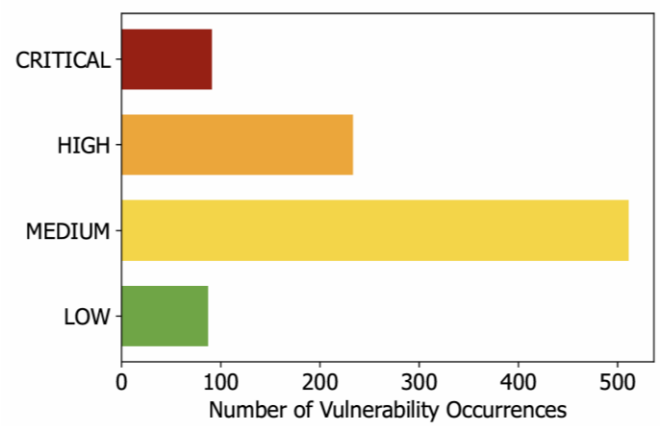
RQ2. Vulnerability Preservation

256 Replicated packages contained known vulnerabilities

1,025 vulnerability occurrences

370 unique CVEs

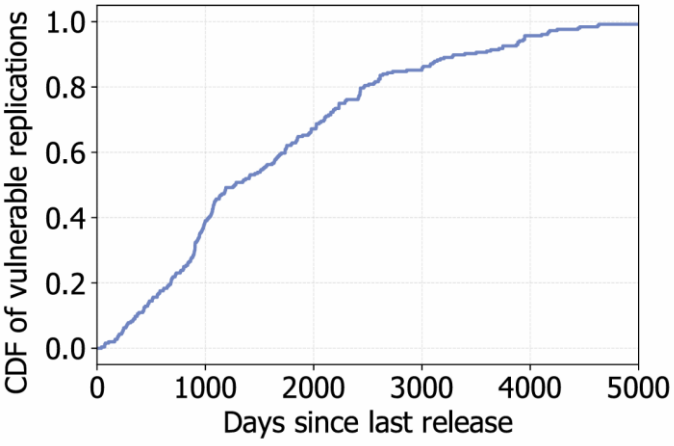


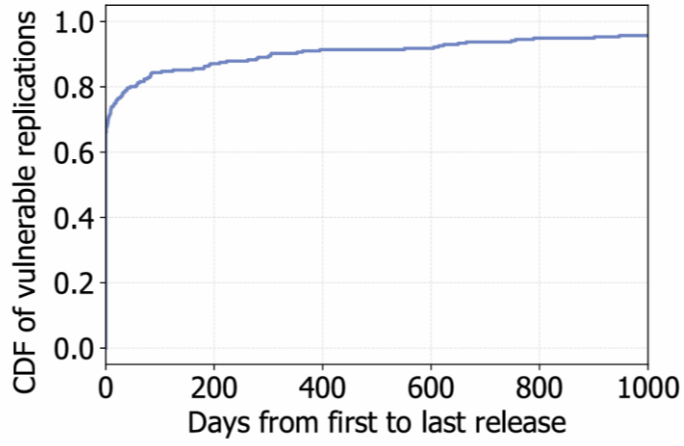


RQ2. Maintenance Status

Vulnerable replicated packages are often left unmaintained

- **90%** not updated for over one year
- **1,630** average days since last release (median: 1,274)
- **92** poorly maintained packages had over 10K downloads





RQ2. Detection Gap

Most vulnerable replicated packages were missed by existing tools

Safety: 4 / 256
Dependency-Track: 1 / 256

Metadata-based scanners rely on original package information

Same vulnerable code under different names can create invisible security risks

RQ2. Example of Vulnerability Preservation

CVE-2025-48889: Unauthorized file copy in Gradio

```
1  def _copy_to_dir(self, dir: str) -> FileData:
2    pathlib.Path(dir).mkdir(exist_ok=True)
3    new_obj = dict(self)
4    if not self.path:
5      raise ValueError("Source file path is not set")
6    new_name = shutil.copy(self.path, dir) # vulnerable sink
7    new_obj["path"] = new_name
8    return self.__class__(**new_obj)
```

Upstream Gradio
patched

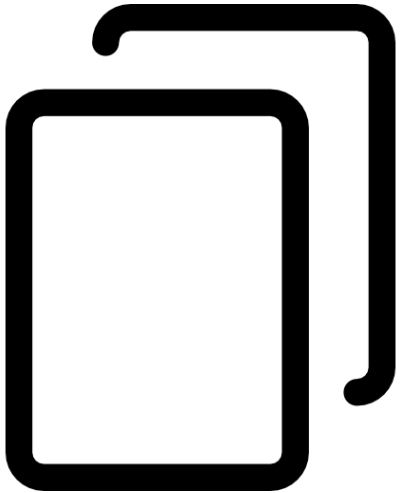
Replicated package
vulnerable flow remained

Patch did not propagate to the replicated package

Metadata-based scanners did not report it

RQ3. Malware Analysis

Popular package



Package replication

+

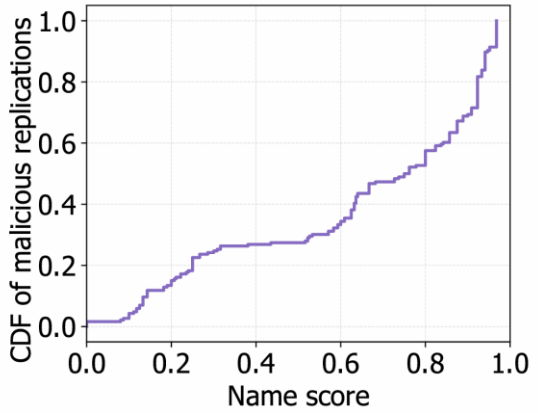
Inject suspicious APIs

Replicated package

RQ3. Malware Analysis

186 known malicious packages reused popular package code

- **67%** reused ≥ 90% of original code
- **88** had name similarity ≥ 0.8
- **158** shared at least one metadata field



Malicious packages replicate both code and metadata to enhance credibility or evade detection

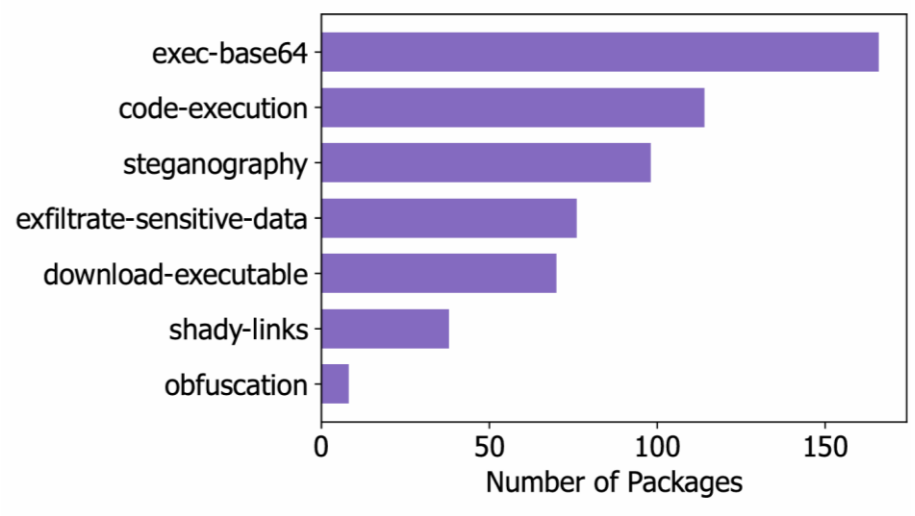
RQ3. Suspicious API Injection

Malicious packages injected suspicious APIs into reused code

180 packages injected **796** unique suspicious APIs

APIs often appeared in combination, suggesting sophisticated malicious intent

Typical malicious behaviors
(GuardDog-based)
- Payload execution
- Concealment
- Data theft / external fetch



RQ3. Example of Malicious Replication

Near-identical package with a small malicious modification

- aliyun-python-sdk-core-2.13.36 → *python-aliyun-sdk-core-2.13.36*
- 144 / 146 files reused unchanged, 4 metadata fields identical
- Injected code collected access keys and stored them for exfiltration

```
1   resolve_request.endpoint_regional = request.endpoint_regional
2 + try:
3 +   sessions = Session()
4 +   data = {"ak": self._ak, "secret": self._secret}
5 +   sessions.close()
6 + except:
7 +   pass
8   return self._endpoint_resolver.resolve(resolve_request)
```

RQ3. Real-world Detection

7 previously unknown malicious packages found in recent uploads

25,407 recent registered packages (May–Aug 2025)

↓

227 package versions replicated popular packages

↓

67 added suspicious APIs

↓

7 previously unknown malicious packages

Replication is an attack vector in recent PyPI uploads

Conclusion

A large-scale, in-depth study of package replication in PyPI

- Replication is not rare

  1,361 packages reused popular package code
- Preserves vulnerabilities

  256 vulnerable replicated packages were identified
- Enables malware distribution

  186 known and **7** previously unknown malicious packages found

Package replication analysis can uncover hidden security blind spots beyond simple code reuse

Q & A

Thank you for your attention!

Repository: https://github.com/sunha21/pypi-replication-analysis

Contact 

- Sunha Park (sunhap@korea.ac.kr)
- Software Security & Privacy Lab. (https://ssp.korea.ac.kr)