

An Empirical Study of Malicious Code In PyPI Ecosystem

Wenbo Guo*, Zhengzi Xu^{†‡}, Chengwei Liu[†], Cheng Huang*, Yong Fang^{*‡}, Yang Liu[†]
honywenair@gmail.com, zhengzi.xu@ntu.edu.sg

[†]School of Computer Science and Engineering, Nanyang Technological University, Singapore

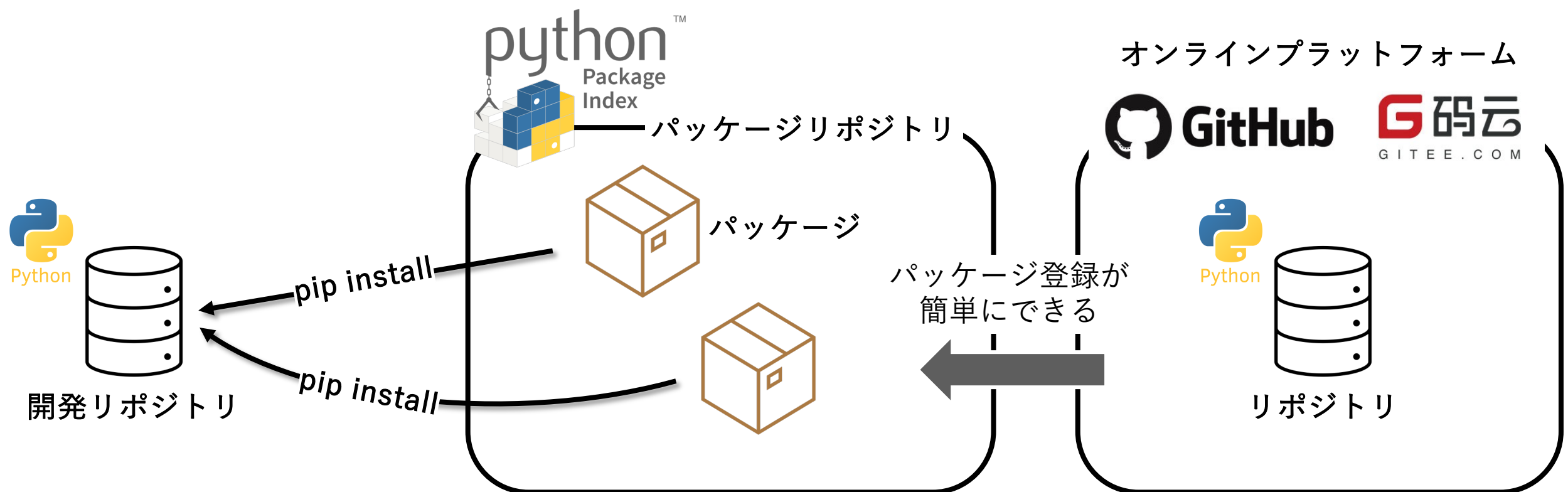
*School of Cyber Science and Engineering, Sichuan University, China

Proceedings of the 38th IEEE/ACM International Conference on Automated Software Engineering (ASE 2023)

PyPIエコシステムとMalicious Code

Pythonではさまざまな開発が可能であり、**再利用可能なパッケージ**として提供されることが多い

- 再利用可能なパッケージはPyPIやCondaに登録されている
- “pip” や “conda” で再利用可能なパッケージをインストール



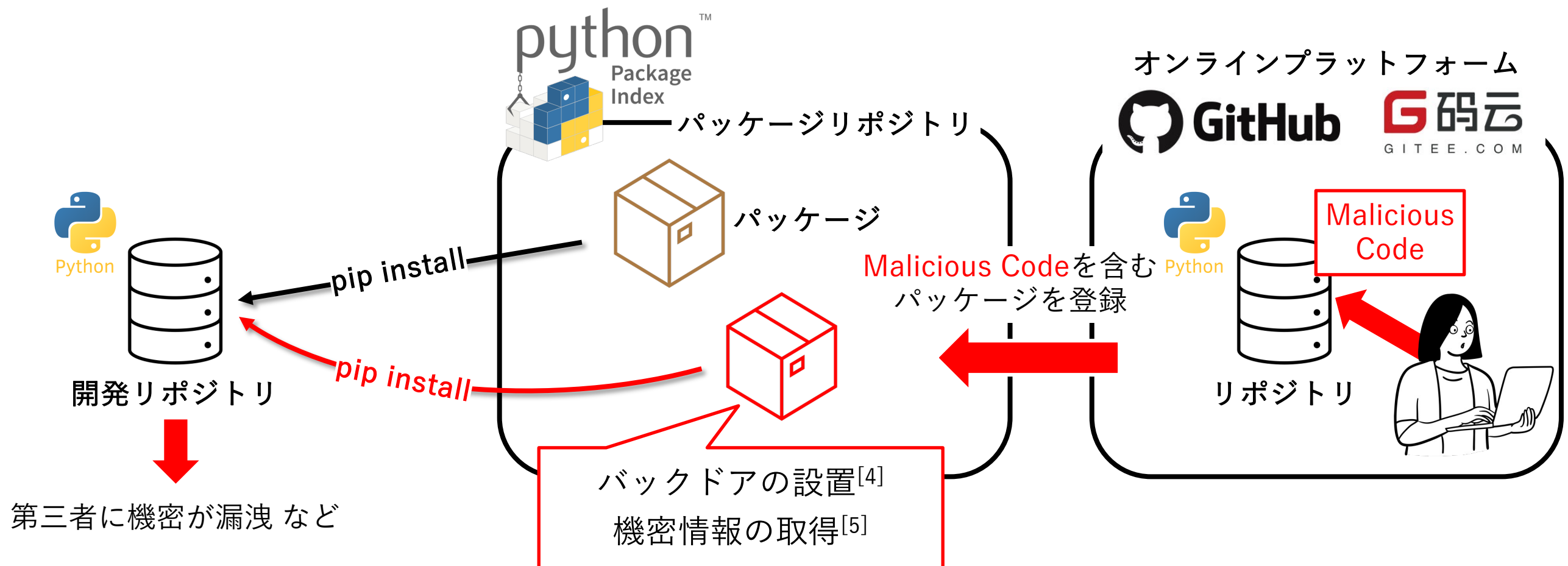
PyPIエコシステムとMalicious Code

OSSには**Malicious Code**が含まれている可能性がある

Malicious Code

何らかの害意がある不正コードの総称

- プラットフォームがセキュリティ保証しているわけではない
 - GitHubだから大丈夫というわけではない



既存研究 | PyPIとマルウェア

マルウェアパッケージレベルの調査^[7, 8, 9]

- マルウェア伝播パターン, 影響要因, 起こりうる問題 など

課題

- ソースコードレベルでの問題が十分に調査されていないこと
 - Malicious Codeの特徴が理解できない
- Malicious Codeに関連するデータセットが不十分であること
 - 検出手法の構築には実際のコードサンプルなどが必要

[7] B. Kaplan and J. Qian, “A survey on common threats in npm and pypi registries,” in Deployable Machine Learning for Security Defense: Second International Workshop, MLHat 2021, Virtual Event, August 15, 2021, Proceedings 2, pp. 132–156, Springer, 2021.

[8] M. Alfadel, D. E. Costa, and E. Shihab, “Empirical analysis of security vulnerabilities in python packages,” Empirical Software Engineering, vol. 28, no. 3, p. 59, 2023.

[9] M. Valiev, B. Vasilescu, and J. Herbsleb, “Ecosystem-level determinants of sustained activity in open-source projects: A case study of the pypi ecosystem,” in Proceedings of the 2018 26th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering, pp. 644–655, 2018.

研究目的・Research Question

研究目的

- 先述の課題を克服するために、PyPIエコシステムにおける Malicious Codeの実証的に調査する
 - サンプル付きデータセットの提供, 具体的な攻撃戦略 など

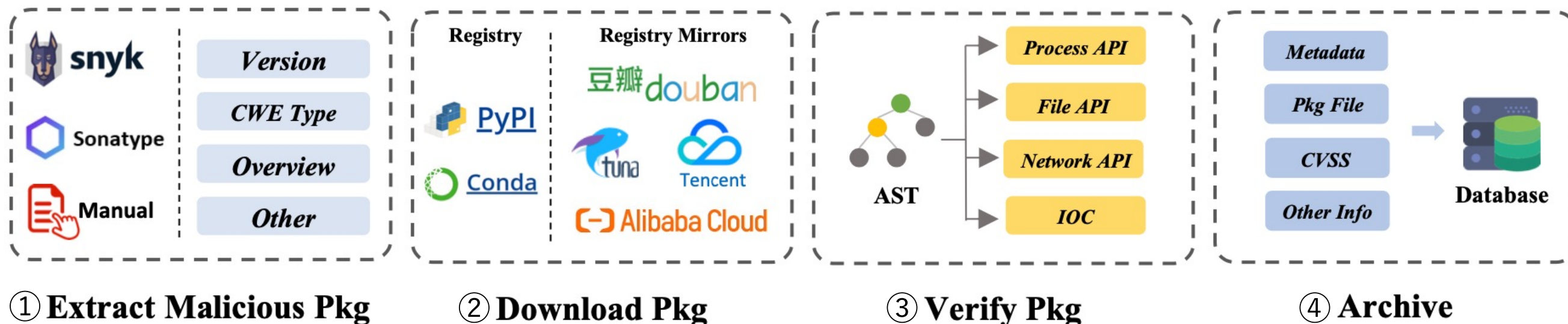
Research Question

- RQ1：コードの特徴
- RQ2：攻撃戦略
- RQ3：検出回避技術
- RQ4：影響と侵入

データセット

作成手順

- ① セキュリティサイトからマルウェアパッケージ情報をウェブスクレイピング
- ② 公式サイトやミラーサイトからマルウェアパッケージをダウンロード
- ③ パッケージファイルをASTに変換あと，APIやIOCを含むコードを抽出しMalicious Codeかどうか検証
- ④ 収集したデータをデータベースに保存



RQ1 | コードの特徴

TABLE II: Unique properties of malicious code

Data Source	Pkg C	Py C	Total L	Avg L	Max D
PyPI	2,035	3,355	1,119,707	334	3
GitHub	2,355	54,460	10,685,811	196	17
Exploit DB	25	25	806	32	1
StackOverflow	116	116	2,283	20	1
VX Heavens	17	64	14,684	229	4
Attck Tools	1	1	1	1	1
Hackerforums	1	1	159	159	1

Pkg C : パッケージ数

Py C : パッケージに含まれるファイル数

Total L : 合計コード行数

Avg L : 1 ファイルあたりのコード行数

Max D : ファイルの深さ

PyPI : 規模が小さくあまり複雑ではない

- 1 パッケージあたり2ファイル ($3,355/2,035$)
- ファイルの深さが最大でも3

GitHub : 規模が大きく複雑

- 1 パッケージあたり23ファイル ($54,460/2,355$)
- ファイルの深さが最大で17

RQ1 | コードの特徴

| Malicious Code Density (LOM/LOC)

- Malicious Codeが含まれる割合
 - LOM : 1 ファイルあたりのMalicious Code行数
 - LOC : 1 ファイルあたりのコード行数
 - MCD値0.5以上のパッケージは52%
 - **MCD値0.9以上のパッケージは43%**
- 多くのパッケージはシンプルな設計となっている
- 複雑な場合は、**少ないコードかつ間接的に不正コードが実行されるように設計されている**

RQ3 | 検出回避技術

既存の検出ツールでは検出できないもしくは偽陽性を多く含む (Fig. 5 参照)

回避方法

- Indirect Import
間接的にインストールさせる

Listing 1: Indirect Import in secrevtwo-0.0.3.

```
1 setup(install_requires=['secrevtwo']
2 try:
3     p = subprocess.Popen(["python3", "-c", \
4         "from_secrevtwo_import_dist_util"], close_fds=True)
```

- Image Steganography
画像に不正コードを埋め込む

Listing 2: Image Steganography in colorsapi-6.6.7.

```
1 with open(f'{os.getenv("TEMP")}\\a.png', 'wb') as f:
2     f.write(r.content)
3 exec(lsb.reveal(f'{os.getenv("TEMP")}\\a.png'))
```

- Sandbox Escape
Sandboxから実領域に侵入

Listing 3: Python Sandbox Escape in colorwin-6.6.7.

```
1 __import__('builtins').exec(__import__('builtins').
2 compile(__import__('base64').b64decode("..."),
3 '<string>', 'exec'))
```

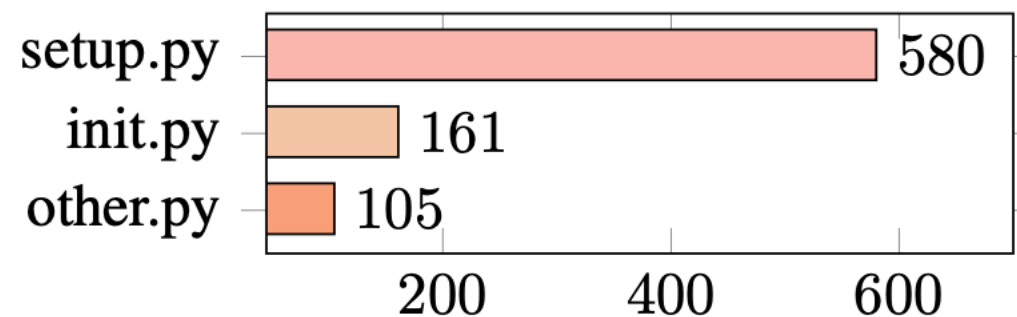
RQ4 | 影響と侵入

攻撃タイミングは3通り

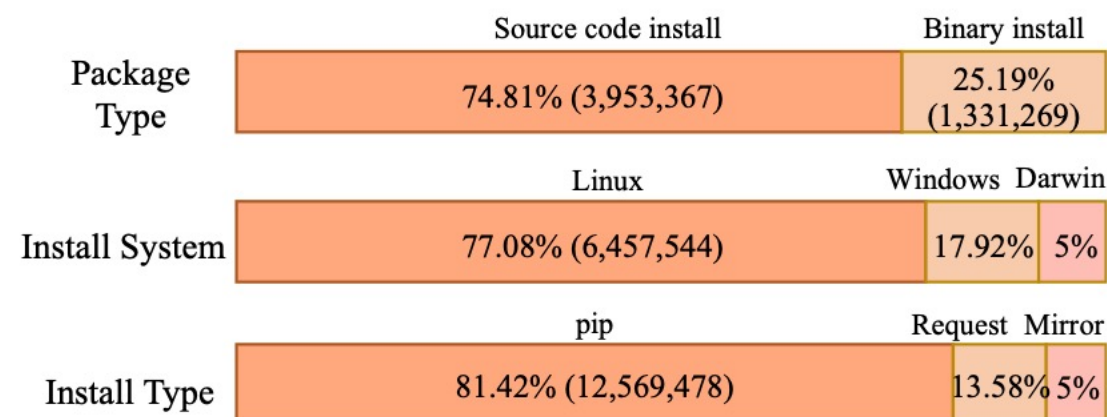
- インストール (68.6%) , インポート (19%) , 実行 (12.4%)

インストールを介した攻撃について

- setup.pyを介して実行されることが多い



- Linuxを対象としていることが多い



まとめ・所感

Malicious Codeについて実証的調査を実施

- Malicious Codサンプル付きのデータセットの提供
- ソースコードレベルでの実態を明らかにした

所感

- 具体的に用いられる攻撃など勉強になった
 - RQ3の回避方法などは衝撃的だった
- セキュリティに関して知識をつけておくのはこれからの時代では必要だと思う