

A Time Series Analysis of Malware Uploads to Programming Language Ecosystems

Jukka Ruohonen^[0000–0001–5147–3084] and Mubashrah Saddiqa
`{juk, msad}@mmmi.sdu.dk`

University of Southern Denmark, Sønderborg, Denmark

Abstract. Software ecosystems built around programming languages have greatly facilitated software development. At the same time, their security has increasingly been acknowledged as a problem. To this end, the paper examines the previously overlooked longitudinal aspects of software ecosystem security, focusing on malware uploaded to six popular programming language ecosystems. The dataset examined is based on the new Open Source Vulnerabilities (OSV) database. According to the results, records about detected malware uploads in the database have recently surpassed those addressing vulnerabilities in packages distributed in the ecosystems. In the early 2025 even up to 80% of all entries in the OSV have been about malware. Regarding time series analysis of malware frequencies and their shares to all database entries, good predictions are available already by relatively simple autoregressive models using the numbers of ecosystems, security advisories, and media and other articles as predictors. With these results and the accompanying discussion, the paper improves and advances the understanding of the thus far overlooked longitudinal aspects of ecosystems and malware.

Keywords: software ecosystems, malware, vulnerabilities, dependencies, security risks, typo-squatting, security scanning, sweeps, autoregression, lags, CRA

1 Introduction

Software ecosystems—understood in the present context as programming language specific repositories from which software packages can be downloaded and updated—have greatly facilitated software development and the general software design principles, among them particularly reusability [6]. This facilitation has correlated with an enormous growth of the ecosystems, many of which contain hundreds of thousands of software packages. However, over again, software ecosystems have also been shown to be risky in terms of software security. Indeed, the security aspects of practically all major software ecosystems have been examined in recent years. The examples include, but are not limited to, PyPI for Python [9, 11, 16, 21], CRAN for R [7], npm for JavaScript [21, 22], Maven for Java [13], RubyGems for Ruby [7], and Packagist for PHP [17]. The overall

conclusion from this already vast but still growing literature branch is the general insecurity of the software ecosystems for individual developers and software development organizations, whether companies or open source software projects.

Among the primary reasons for the security risks is that many of the packages distributed in the ecosystems are of poor quality, containing various unverified security issues or already identified vulnerabilities. The security risks also increase due to a heavy use of dependencies in the ecosystems [7, 13, 22]. Both direct and transitive dependencies contribute to the risks, which are also related to the presence of many outdated, unmaintained, and abandoned packages distributed in the ecosystems. Also the operational security of software developers using the ecosystems has been seen as a risk factor [21, 22]. Furthermore, the problems are made worse by the increasing presence of malware in some ecosystems [9, 11, 21]. The paper aligns with and contributes to the last mentioned genre of software ecosystem research, the empirically motivated malware-specific research domain.

A traditional attack vector with the malware uploads has been so-called typo-squatting; an attacker uploads a malware package with a name resembling an existing, legitimate package, trying to fool people into downloading and installing the malware-ridden package [2]. Such typo-squatting belongs to a broader class of name confusion attacks [19]. For instance, it has recently been argued that hallucinated package names by large language models might make the problem worse in the nearby future [5]. Regardless, the problem is already intensified by dependencies because it essentially may only take one misstep by one developer somewhere in a dependency network to compromise the whole network [11]. For this reason alone, it is important to gain better knowledge on the longitudinal aspects of malware uploads to popular programming language ecosystems. With this point in mind, the following three research questions (RQs) are examined:

RQ.1: How much detected malware uploads have popular programming language ecosystems seen compared to traditional vulnerability reports?

RQ.2: Which ecosystems have been particularly prone to malware uploads?

RQ.3: Can time series analysis provide insights into malware uploading trends?

To the best of the authors' knowledge, the RQs, or something analogous, have not been asked and examined in previous research. The paper thus fills a small but notable knowledge gap. The first RQ.1 also reflects a recently proposed distinction between supply chain attacks and vulnerabilities; the former can be seen to involve also an insertion of malware into an ecosystem, whereas the latter is about vulnerabilities in third-party components propagating into a software using the components [6]. Though, as was noted, also malware may propagate through dependencies. If such a propagation reaches commercial software vendors or important open source software projects, including distributors such as Linux distributions, the consequences can be catastrophic in many ways. In any case, before continuing to the means to answer to the three RQs, the paper's topic is further motivated from a new and different angle in the opening Section 2. The subsequent Section 3 presents the materials and methods. Results are presented in Section 4, and a conclusion follows in the final Section 5.

2 Motivation

The security risks involved are easy to demonstrate. Most—if not all—malware recently discovered from the npm ecosystem come with the following warning:

“Any computer that has this package installed or running should be considered fully compromised. All secrets and keys stored on that computer should be rotated immediately from a different computer. The package should be removed, but as full control of the computer may have been given to an outside entity, there is no guarantee that removing the package will remove all malicious software resulting from installing it.”¹

This serious warning is not intensified only by the noted risk with dependencies. Among other things, many malware-ridden packages in both the npm and PyPI ecosystems have relied on installation-time infections made possible by the execution of scripts during installation or even downloading [21]. Many of the malware uploads recorded in the OSV are further referenced with CWE-506, which refers to embedding of malicious code in the Common Weakness Enumeration (CWE) framework. A similar warning is available from this framework.

The many security risks have been recognized also by policy-makers in recent years. In terms of new regulations, particularly important to acknowledge is the Cyber Resilience Act (CRA) recently enacted in the European Union (EU) [20]. This regulation is noteworthy for motivating also the present work. Among other things, the CRA’s new essential cyber security requirements for most information technology products with a network functionality contain an obligation to only ship products without known vulnerabilities. This requirement applies also to vulnerabilities in dependencies distributed in software ecosystems. Although the CRA does not mention malware explicitly, its further essential requirements to ensure confidentiality, integrity, and availability [15] can be seen to cover also malware—as the above quotation also testifies. Accidentally embedding malware to a product is thus likely to face also regulatory sanctions at least in severe cases.

The CRA is important to mention also from a perspective of not software products and producers but also from a perspective of regulators. In particular, the regulation’s Article 60 obliges European market surveillance authorities to conduct coordinated sweeps of particular products for checking compliance and detecting potential infringements. Regarding the paper’s time series analysis, the note in the CRA’s recital 114 about justifying sweeping particularly when “market *trends*, consumer complaints or *other indications* suggest that certain categories of products with digital elements are often found to present cybersecurity *risks*” (italics added). While the CRA is not meant to regulate all the world’s software, sweeping software ecosystems, possibly together with other stakeholders, might improve the cyber security for everyone. To this end, it could be also argued that the new obligations placed upon regulators themselves might enhance and improve the existing tracking and monitoring infrastructures, among

¹ <https://osv.dev/vulnerability/MAL-2024-226>

them the OSV database. After all, in the context of cyber security, the concept of a sweep, which originates from the EU’s product safety laws [4], is rather close to security scanning, security audits, security monitoring, and related concepts and techniques already used to improve also the security of software ecosystems.

3 Materials and Methods

3.1 Data

The dataset examined was assembled from a bulk snapshot obtained in April 2025 from the OSV database.² Although OSV curates data from various publicly available sources, the dataset assembling was restricted to CRAN, Go, Maven, npm, PyPI, and RubyGems. Then, the following time series were constructed:

1. $MalFreq_t$ counts the number of malware entries reported for all of the six ecosystems sampled at t . The identification of malware entries was done by including those files whose names started with a **MAL-** character string. Even though this simple identification technique is not perfect, searching for a string **malware** from the other files indicates no major concerns.
2. $MalShare_t$ is a percentage share of malware entries to all entries in the six ecosystems at a given t . If $VulnFreq_t$ would be a total count of software vulnerabilities in the six ecosystems at the given t , an approximation $MalShare_t \simeq MalFreq_t / (MalFreq_t + VulnFreq_t) \times 100$ would hold.
3. Eco_t is a count of the given ecosystems that contributed to $MalFreq_t$ at t . It follows that $\max(Eco_t) = 6$ and $\min(Eco_t) = 0$ hold for all t .
4. Adv_t is a count of security advisories curated in the OSV at t for malware entries in the six ecosystems. Given the OSV’s JavaScript Object Notation (JSON) schema, the parsing was done by searching and counting the **ADVISORY** entries in the schema’s **references** field.
5. Art_t is a count of media articles, blog posts, and related information sources recorded in the OSV database at t for malware entries in the six ecosystems. The parsing was analogous to Adv_t but by using the **ARTICLE** entries.

These five time series were operationalized into daily, weekly, and monthly aggregates for which the lengths are $T = 1195$, $T = 168$, and $T = 39$, respectively. The starting periods were restricted to the first day, first week, and first month (January) of 2022. The reason for this restriction is that only a few malware entries have been recorded in the OSV database prior to 2022. Regarding the daily aggregates, $MalShare_t$ was manually set to zero in case an amount of all entries at a day t was zero. Given the date of the data collection, the end periods are March 2025 and its last day and week.

² <https://osv.dev/>

3.2 Methods

The following autoregressive distributed lag (ARDL) model is used:

$$\begin{aligned} f(y_t) = & \alpha + \sum_{j=1}^{p_1} \beta_j f(y_{t-j}) + \sum_{j=0}^{p_2} \gamma_j f(Eco_{t-j}) \\ & + \sum_{j=0}^{p_3} \phi_j f(Adv_{t-j}) + \sum_{j=0}^{p_4} \rho_j f(Art_{t-j}) + \varepsilon_t, \end{aligned} \quad (1)$$

where y_t refers to either $MalFreq_t$ or $MalShare_t$, $t = 1, \dots, T$, α is a constant, β_j , γ_j , ϕ_j , and ρ_j are regression coefficients, and ε_t is a normally distributed residual term with a zero mean and a variance σ_ε^2 . If y_t is $MalFreq_t$, $f(x) = \ln(x + 1)$; for $MalShare_t$ it is an identity function, $f(x) = x$. The immediate effects of a unit change in Eco_t , Adv_t , and Art_t upon y_t are given by γ_0 , ϕ_0 , and ρ_0 , respectively. If the unit changes are sustained, the effects are given by so-called long-run multipliers (LRMs). To use Eco_t as an example, such a multiplier is given by

$$LRM_{Eco_t} = \frac{\sum_{j=0}^{p_2} \gamma_j}{1 - \sum_{j=1}^{p_1} \beta_j}. \quad (2)$$

The interpretation of these LRMs is similar to standard regression coefficients. If $MalShare_t$ and Eco_t are considered as an example, a sustained increase by one ecosystem in the Eco_t series will increase $MalShare_t$ by LRM_{Eco_t} percentage points, all other things being constant. In addition, dynamic multipliers (DMs) are useful for evaluating the dynamics of a given effect [3, 12]. To again use Eco_t as an example, for some integer $k > 1$ the DMs for it are given by

$$(DM_{Eco_t,1}, \dots, DM_{Eco_t,k}) = \left(\frac{\partial y_t}{\partial Eco_t}, \dots, \frac{\partial y_{t+k}}{\partial Eco_t} \right), \quad t + k \leq T. \quad (3)$$

For a sufficiently large k , it follows that

$$\sum_{i=1}^k DM_{Eco_t,i} \simeq LRM_{Eco_t}. \quad (4)$$

Finally, there is the tricky problem of selecting the orders p_1 , p_2 , p_3 , and p_4 . On one hand, selecting too short orders may lead to the omitted variable bias because relevant information is excluded. Too short orders often lead to also other problems, including remaining autocorrelation in the residual term ε_t . On the other hand, selecting too long orders encounters the overfitting problem.

By inspecting automatic order selection algorithms in two different implementations [14, 18], it can be concluded that both implementations yield extremely long orders both with the Akaike information criterion (AIC) and the Bayesian information criterion (BIC). Therefore, a manual but still systematic three-step procedure was used. In the first step the orders were uniformly increased, $p_1 = p_2 = p_3 = p_4$, until no notable autocorrelation was present in the

residual terms. In the second step p_1 , as obtained from the first step, was held constant but the remaining orders, $p_2 = p_3 = p_4$, were uniformly decreased until either autocorrelation was present or any of the coefficients γ_{p_2} , ϕ_{p_3} , ρ_{p_4} were statistically significant at the conventional 95% confidence level. In the third and final step p_4 , p_3 , and p_2 , in the order of listing, were consecutively and individually decreased by using the same stop criterion as in the second step.

4 Results

4.1 Descriptive Statistics

The presentation of the results can be started by taking a look at the OSV’s malware entries across the six ecosystems; a basic breakdown is shown in Table 1. As can be seen, npm and PyPI have garnered the most entries—as well as the most malware entries. In fact, as much as about 84% and 57% of all entries for these two ecosystems have recently (from 2022 onward) been about malware. Although it is impossible to say how many have fallen victim to these malware uploads, the observation is still quite alarming in a sense that vetting of new uploads seems to be either working poorly or absent altogether. Interestingly, furthermore, RubyGems takes only the fifth place in terms of total entries but the third place in terms of malware uploads. At the moment, CRAN, Go, and Maven seem to have not been particular targets of malware uploads, or they have countermeasures in place, but the situation may change in the future.

Table 1. Entries Across the Six Ecosystems

Ecosystem	Frequency		Malware share
	All entries	Malware entries	
CRAN	10	0	0.00
Go	4,145	8	0.19
Maven	5,461	1	0.02
npm	24,837	20,481	82.46
PyPI	15,929	8,966	56.29
RubyGems	1,727	813	47.07

The weekly $MalFreq_t$ and $MalShare_t$ time series shown in Fig. 1 further indicate the persistence of malware uploads to the three ecosystems. The former series contain three large spikes, which are likely due to specific sweeps or more general clean-up operations. Even when keeping these spikes in mind, the median is as high as 37 malware entries per week, which is again a rather striking number on its own. Then, the $MalShare_t$ time series fluctuates a lot, partially due to the spikes in $MalFreq_t$ but also otherwise. This observation reflects the operationalization of the time series (see Subsection 3.1). The moving average

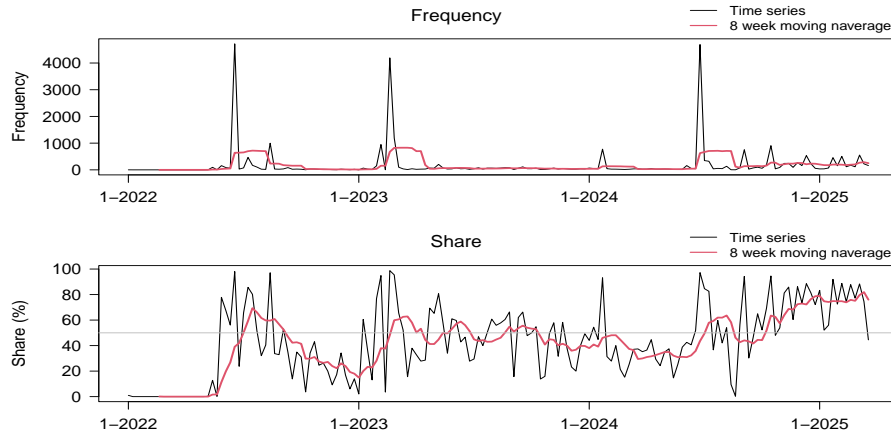


Fig. 1. The Two Malware Time Series (weekly aggregates)

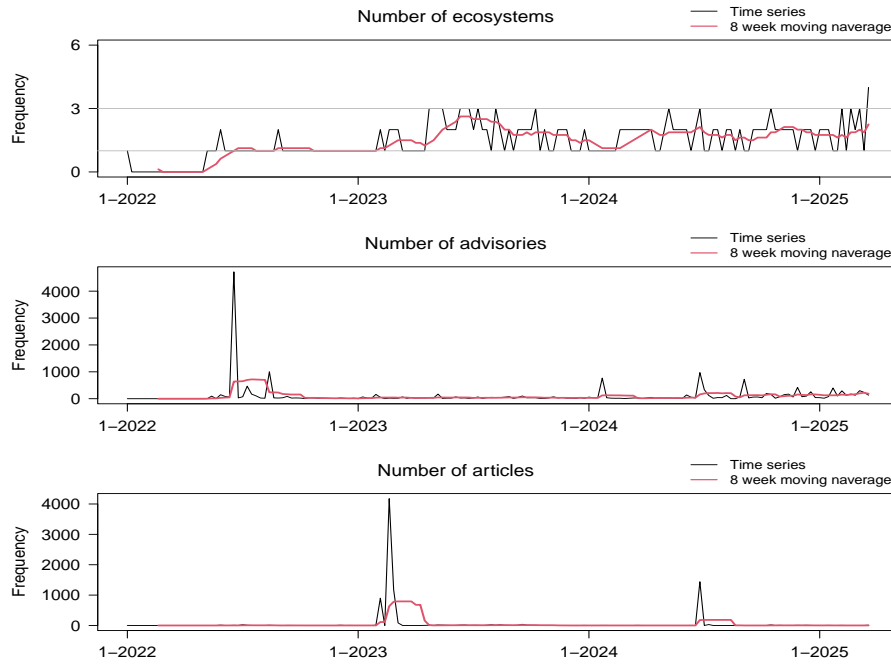


Fig. 2. The Three Explanatory Time Series (weekly aggregates)

shown was around 50% during 2023 and most of 2024, but in the early 2025 it had increased even up to 80%. This amount is almost twice the median of 41%.

Regarding the three explanatory time series shown in Fig. 2, Eco_t fluctuates mainly between the values one and three, meaning that malware uploads into npm, PyPI, and RubyGems have sometimes been reported individually in a given week but some other times weekly malware reports have been made for all three ecosystems. However, a human eye cannot see whether the fluctuations in Eco_t correspond with the fluctuations in $MalFreq_t$ and $MalShare_t$. This point justifies the formal ARDL modeling soon disseminated in Subsection 4.2.

With respect to the two other explanatory time series, the three notable spikes in $MalFreq_t$ seem to correspond with one large spike in Adv_t and two visible spikes in Art_t . Also this observation motivates the formal time series modeling because it seems that to some extent publicity correlates with reported malware uploads. When taking a peek at the sources behind the articles counted by Art_t , many of these have been either media articles and blogs about open source software supply-chain security or malware discovery announcements from cyber security companies and others scanning the programming language ecosystems. Given the ARDL context, it can be hypothesized that publicity may not only correlate simultaneously with the spikes but past publicity may influence a current or a future discovery rate. If there is a lot of publicity, as has been the case in the past three years, it may be that more and more companies, open source software developers, cyber security professionals, and others pay attention to the malware uploading problem. That is, it may be that $Adv_{t-1}, \dots, Adv_{t-p_3}$ and $Art_{t-1}, \dots, Art_{t-p_4}$, not necessarily Adv_t and Art_t alone, influence $MalFreq_t$ and $MalShare_t$. A similar reasoning applies to Eco_t . The rising trends in $MalFreq_t$ and $MalShare_t$ from late 2024 onward, and the persistence of the $[1, 3]$ range fluctuations in Eco_t , may—or should—motivate further security scans and sweeps—or, alternatively, these should not at least motivate stopping existing efforts.

4.2 Regression Analysis

The ARDL model in (1) is estimated for both $MalFreq_t$ and $MalShare_t$ by using the daily, weekly, and monthly aggregates. Thus, in total six models are estimated. Before continuing, it can be noted that both series are stationary, as also confirmed by formal Dickey-Fuller tests [8]. Another preliminary point is about the manual order selection procedure described in Subsection 3.2. As can be seen from Table 2, the procedure resulted relatively, but not substantially, large orders for the daily aggregates, as could be expected. Somewhat unexpectedly, however, rather short orders were suitable for the weekly aggregates. A minimal model with $p_1 = 1$ and $p_2 = p_3 = p_4 = 0$ was suitable for the monthly $MalShare_t$ series. Despite these points, the procedure worked well in ensuring that no remaining autocorrelation is present. As can be seen from Fig. 3, all autocorrelation functions (ACFs) remain below the 95% confidence intervals.

As is typical in applied problems, the normality assumptions about ε_t is a small problem. For instance, the Jarque-Bera test [10], which is based on skewness and kurtosis, rejects the null hypothesis of normality for all residual

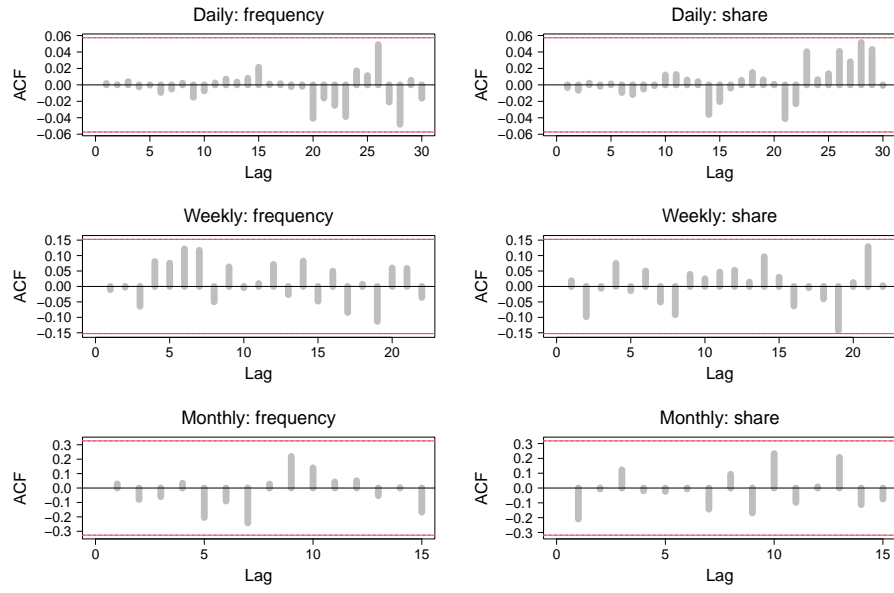


Fig. 3. Autocorrelation Functions of the Residual Terms from the Six ARDL Models (95% confidence intervals; maximum lag lengths determined by $\lfloor 10 \times \log_{10}(T) \rfloor$)

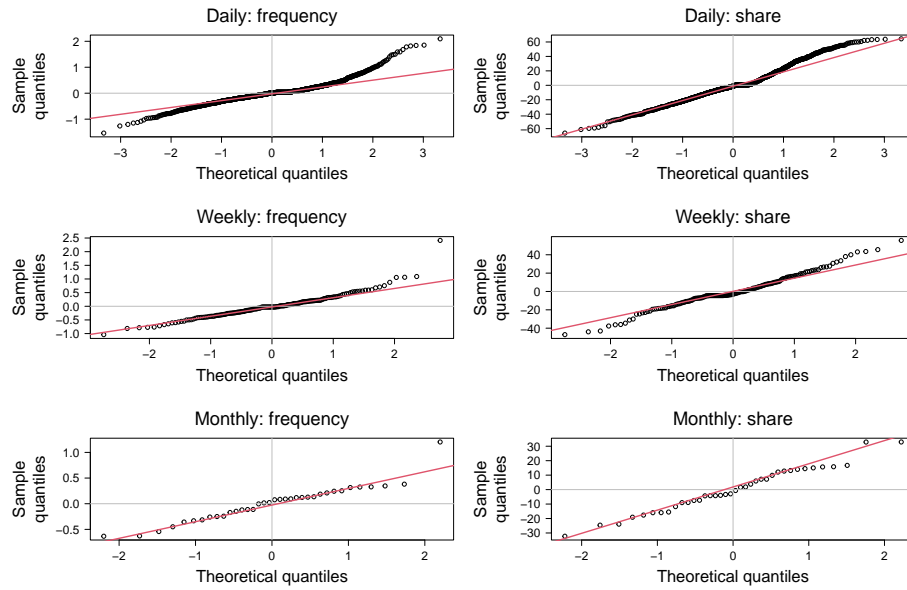


Fig. 4. Normal Quantile-Quantile Plots

Table 2. ARDL(p_1, p_2, p_3, p_4) Orders Selected

	Daily	Weekly	Monthly
Frequency	(27, 26, 22, 18)	(3, 2, 2, 2)	(3, 2, 3, 2)
Share	(23, 21, 21, 20)	(3, 0, 1, 0)	(1, 0, 0, 0)

series except those coming from the weekly and monthly estimates for $MalShare_t$. When taking a visual look at Fig. 4, however, the situation is hardly as bad as the formal test results would indicate. In other words, all residual series resemble the normal distribution; some more, some less, but all still sufficiently for proceeding.

Heteroskedasticity is also a slight problem. As can be seen from Fig. 5, particularly the residuals from the two models for the daily aggregates indicate non-random patterns. There is also a related problem: the plain ARDL model is not optimal for $MalShare_t$ because $\max(MaxShare_t) = 100$, which is exceeded by some of the estimated values. However, such exceedances are rather small: the percentage shares of estimated values exceeding one hundred are only 1.7, 1.8, and 2.6 for the three models using the daily, weekly, and monthly aggregates of $MalShare_t$, respectively. All in all, it must be acknowledged that some diagnostic problems are present, as is often the case in applied time series regression analysis, but none of the problems are severe enough to prevent proceeding into the actual results. Against this backdrop, the LRMs are shown in Table 3.

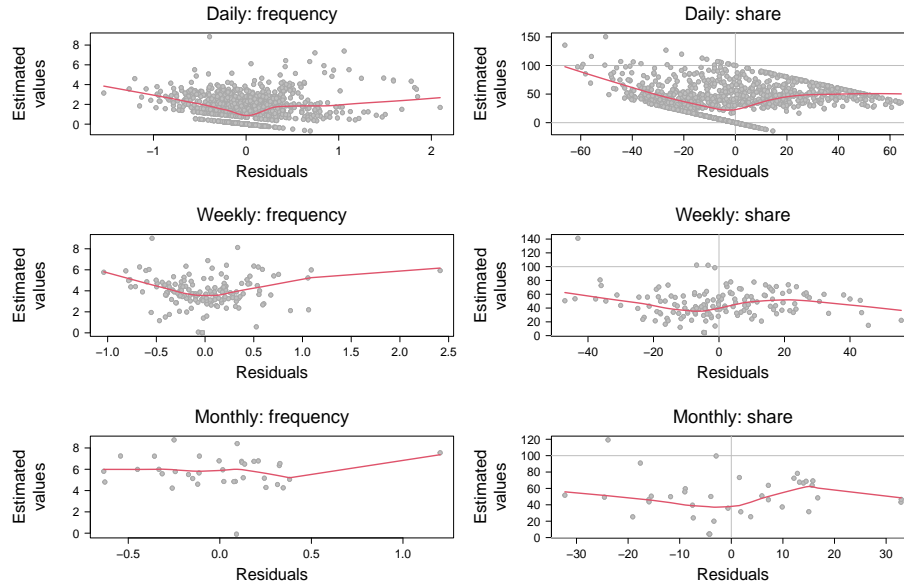
**Fig. 5.** Estimated Values and Residuals

Table 3. Long-Run Multipliers¹

	Daily		Weekly		Monthly	
	Frequency	Share	Frequency	Share	Frequency	Share
Eco_t	0.812	46.501	0.437	19.413	0.160	13.060
Adv_t	0.840	0.113	0.886	0.027	0.963	0.033
Art_t	0.519	0.061	0.344	0.041	0.169	0.015

¹ Colored entries denote statistical significance at the conventional 95% level.

All three explanatory time series indicate long-run effects, irrespective whether daily, weekly, or monthly aggregates are used. Furthermore, only three of the LRMs are not statistically significant at the conventional 95% confidence level. Having said that, the effects from Adv_t and Art_t are much lower than those from the ecosystem count time series, which indicate substantial long-run impacts. All other things being constant, a unit increase in Eco_t increases $MalShare_t$ by 46.5 percentage points daily, 19.4 percentage points weekly, and 13.1 percentage points monthly. When keeping the maximum in mind, these long-run effects are substantial but hardly surprising as such due to the concentration of malware uploads to npm, PyPI, and RubyGems. The DMs shown in Fig. 6 further indicate that the effects are not merely immediate shocks but persist relatively long before eventually dampening. Though, a similar observation applies to the other series as well. The Adv_t and Art_t time series indicate particularly disturbing dynamic shocks upon $MalShare_t$ in the model using the daily aggregates. Although the corresponding effects, as seen from the y -axes in Fig. 6 and the LRMs in Table 3, are still small in magnitude, these persistent but fluctuating shocks could be interpreted to support a conclusion that the publicity theorization is not entirely without a basis. As for the earlier speculation in the previous Subsection 4.1 about the potential impact of the past values of the ecosystem time series, it can be concluded that—and despite for the selection of $p_2 = 0$ for two series (see Table 2)—the effects are not entirely simultaneous. Finally, the empirical exposition can be ended by noting that the ARDL models yield generally good statistical performance; the lowest and highest coefficients of determination are 0.59 and 0.95. While forecasts can be left for further work, such values hint that even simple time series model could be used also in practical foresight about programming language ecosystem security in the nearby future.

5 Conclusion

The paper examined recent (from 2022 to early 2025) malware uploads to six popular programming language ecosystems: CRAN, Go, Maven, npm, PyPI, and RubyGems. Regarding the three research questions specified, the answer to RQ.1 is simple but alarming: malware uploads have surpassed the reporting of traditional software vulnerabilities in packages distributed in the ecosystems. With respect to RQ.2, npm (over twenty thousand malware uploads), PyPI (nearly

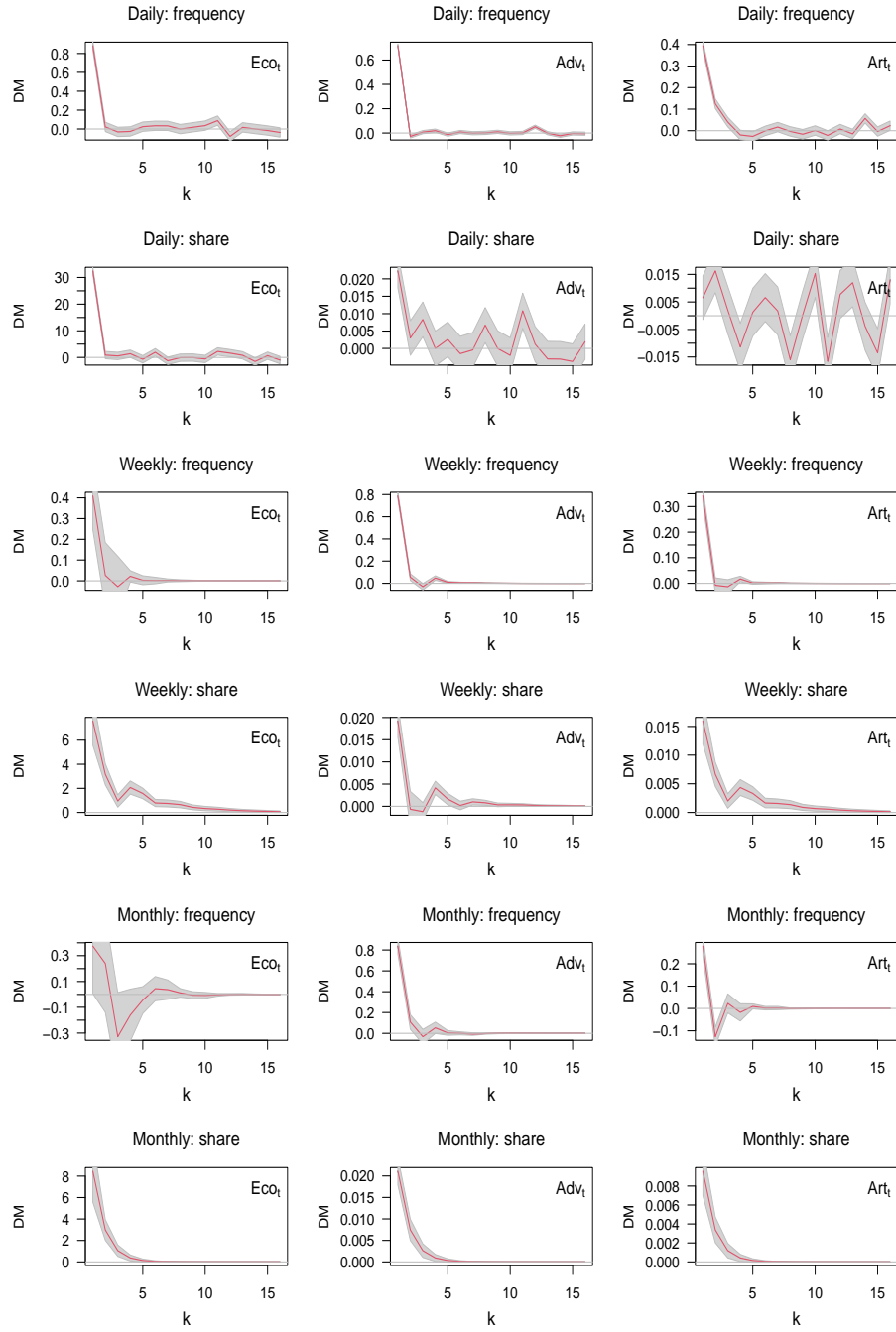


Fig. 6. Dynamic Multipliers

nine thousand malware uploads), and RubyGems (about eight hundred malware uploads) have been particularly prone to malware uploads, whereas CRAN, Go, and Maven have seen less than ten malware uploads in total. The answer to the third and final RQ.3 is that time series analysis can reveal insights about malware uploads and their trends. The decent statistical performance obtained—the average coefficient of determination is 0.79—indicates that forecasting could be used also in practical foresight. Although such forecasts were left for further work, it can be hypothesized that the increasing trend of malware uploads continues also in the nearby future. As could be expected, the number of ecosystems provides particularly good predictive power; the more there are ecosystems, the more malware uploads are also reported. Smaller but still visible effects are present for security advisories and media and other articles; publicity seems to also affect the malware upload trends. Rather analogously to recent arguments about reported vulnerabilities in open source software projects [17], the explanation might be that increasing publicity about malware uploads prompts more companies and security professionals to scan and monitor the ecosystems.

The research on ecosystems and malware has often recommended improving monitoring and detection capabilities [2, 9]. In addition to publicity and awareness, it may be that this recommendation also implicitly and partially explains the answer to RQ.2. In other words, it may be that detection and monitoring capabilities might have already improved, such that more malware uploads have been detected, removed, and reported—possibly irrespective whether malware uploads have actually increased *per se*. While improving the capabilities further may improve the situation somewhat, a probability of bad apples slipping through is also dependent on the sizes of the ecosystems. When there are hundreds of thousands of packages, it is probable that some malware will slip through even with highly accurate detection engines. Against this backdrop, it is interesting to see whether the future will see curated lists for safe and secure packages. A recommendation to improve code signing [6] aligns with such curated lists. Curating has also been what Linux distributions have always done, but somewhere in recent history this quality gating function was forgotten or overridden by the emergence of programming language software ecosystems.

The results have also implications for research. For instance, reflecting a lack of data sharing and a lack of good benchmark datasets in malware research [1], some studies have attempted to verify a true positiveness of a malware sample by checking that the corresponding packages are absent in PyPI [11]. Clearly, such a check is misleading or at least a poor choice due to the prevalence of malware in PyPI too. Instead, a starting point for further research might be to evaluate the performance of existing commercial malware detection engines in the programming ecosystem context.³ It may well be that detection accuracy is not as good as with other, more conventional malware variants usually distributed as binaries. There is also room for more practice-oriented work regarding takedown efficiency, which seems to be suboptimal according to existing research [2]. Though, takedowns and clean-ups reiterate also the point about curated lists and signing—it is debatable to which areas future efforts should be allocated.

³ <https://www.virustotal.com/>

References

- [1] Botacin, M., Ceschin, F., Sun, R., Oliveir, D., Grégio, A.: Challenges and Pitfalls in Malware Research. *Computers & Security* 106, 102287 (2021)
- [2] Cao, A., Dolan-Gavitt, B.: What the Fork? Finding and Analyzing Malware in GitHub Forks. In: *Proceedings of the Workshop on Measurements, Attacks, and Defenses for the Web (MADWeb 2022)*. The Internet Society, San Diego (2022)
- [3] Cheng, M., Liu, B.: Analysis on the Influence of China's Energy Consumption on Economic Growth. *Sustainability* 11, 3982 (2019)
- [4] Chiara, P.G.: The Cyber Resilience Act: the EU Commission's Proposal for a Horizontal Regulation on Cybersecurity for Products with Digital Elements: An Introduction. *International Cybersecurity Law Review* 3, 255–272 (2022)
- [5] Claburn, T.: LLMs Can't Stop Making Up Software Dependencies and Sabotaging Everything: Hallucinated Package Names Fuel 'Slopsquatting' (2025), *The Register*, available online in April 2025: https://www.theregister.com/2025/04/12/ai_code_suggestions_sabotage_supply_chain/
- [6] Cox, R.: Fifty Years of Open Source Software Supply Chain Security. *ACM Queue* 23(1), 84–107 (2025)
- [7] Decan, A., Mens, T., Claes, M.: An Empirical Comparison of Dependency Issues in OSS Packaging Ecosystems. In: *Proceedings of the IEEE 24th International Conference on Software Analysis, Evolution and Reengineering (SANER 2017)*. pp. 2–12. IEEE, Klagenfurt (2017)
- [8] Dickey, D.A., Fuller, W.A.: Distribution of the Estimators for Autoregressive Time Series With a Unit Root. *Journal of the American Statistical Association* 74(366), 427–431 (1979)
- [9] Guo, W., Xu, Z., Liu, C., Huang, C., Fang, Y., Liu, Y.: An Empirical Study of Malicious Code in PyPI Ecosystem. In: *Proceedings of the 38th IEEE/ACM International Conference on Automated Software Engineering (ASE 2023)*. pp. 166–177. IEEE, Luxembourg (2023)
- [10] Jarque, C.M., Bera, A.K.: Efficient Tests for Normality, Homoscedasticity and Serial Independence of Regression Residuals. *Economics Letters* 6(3), 255–259 (1980)
- [11] Mehedi, S.T., Islam, C., Ramachandran, G., Jurdak, R.: DySec: A Machine Learning-Based Dynamic Analysis for Detecting Malicious Packages in PyPI Ecosystem (2025), archived manuscript, available online: <https://arxiv.org/abs/2503.00324>
- [12] Menegaki, A.N.: The ARDL Method in the Energy-Growth Nexus Field; Best Implementation Strategies. *Economies* 7, 105 (2019)
- [13] Nachuma, C., Hossan, M.M., Turzo, A.K., Zibran, M.F.: Decoding Dependency Risks: A Quantitative Study of Vulnerabilities in the Maven Ecosystem (2025), archived manuscript, available online: <https://arxiv.org/abs/2503.22134>
- [14] Natsiopoulou, K., Tzeremes, N.: ARDL: ARDL, ECM and Bounds-Test for Cointegration (2023), R package version 0.2.4, available online in April 2025: <https://cran.r-project.org/web/packages/ARDL/index.html>
- [15] Ruohonen, J., Hjerppe, K., Kang, E.Y.: A Mapping Analysis of Requirements Between the CRA and the GDPR (2025), archived manuscript, available online: <https://arxiv.org/abs/2503.01816>
- [16] Ruohonen, J., Hjerppe, K., Rindell, K.: A Large-Scale Security-Oriented Static Analysis of Python Packages in PyPI. In: *Proceedings of the 18th Annual International Conference on Privacy, Security and Trust (PST 2021)*. pp. 1–10. IEEE, Auckland (online) (2021)

- [17] Ruohonen, J., Ramadan, Q.: The Popularity Hypothesis in Software Security: A Large-Scale Replication with PHP Packages (2025), archived manuscript, available online: <https://arxiv.org/abs/2502.16670>
- [18] Seabold, S., Perktold, J.: statsmodels: Econometric and Statistical Modeling with Python. In: Proceedings of the 9th Python in Science Conference (SciPy 2010). Austin (2010), Autoregressive Distributed Lag (ARDL) Models, statsmodels 0.15.0, available online in April 2025: https://www.statsmodels.org/devel/examples/notebooks/generated/autoregressive_distributed_lag.html
- [19] Snyk Limited: Name Confusion Attacks (2025), available online in April 2025: <https://learn.snyk.io/lesson/name-confusion-attacks/>
- [20] The European Union: Regulation (EU) 2024/2847 of the European Parliament and of the Council of 23 October 2024 on Horizontal Cybersecurity Requirements for Products With Digital Elements and Amending Regulations (EU) No 168/2013 and (EU) 2019/1020 and Directive (EU) 2020/1828 (Cyber Resilience Act) (Text With EEA Relevance) (2024), available online in March 2025: <https://eur-lex.europa.eu/eli/reg/2024/2847/oj/eng>
- [21] Zhang, J., Huang, K., Huang, Y., Chen, B., Wang, R., Wang, C., Peng, X.: Killing Two Birds With One Stone: Malicious Package Detection in NPM and PyPI Using a Single Model of Malicious Behavior Sequence. *ACM Transactions on Software Engineering and Methodology* (Published online in November), 1–27 (2024)
- [22] Zimmermann, M., Staicu, C., Tenny, C., Pradel, M.: Small World with High Risks: A Study of Security Threats in the npm Ecosystem. In: Proceedings of the 28th USENIX Security Symposium. pp. 995–1010. USENIX, Santa Clara (2019)