



ΠΑΝΕΠΙΣΤΗΜΙΟ ΠΕΙΡΑΙΩΣ

Τμήμα Ψηφιακών Συστημάτων

Πρόγραμμα Μεταπτυχιακών Σπουδών

Κυβερνοασφάλεια και Τεχνολογίες Τεχνητής Νοημοσύνης

Διπλωματική Εργασία

SupplyGuard

Ένα Policy-Driven Πλαίσιο Προεγκαταστασιακής Άμυνας
για την Ασφάλεια της Python Software Supply Chain

*A Policy-Driven Pre-Installation Defense Framework
for Python Package Supply-Chain Security*

Φοιτητής: Ανδρέας Βανικιώτης

Αριθμός Μητρώου: mte24005

Επιβλέπων Καθηγητής: Κωνσταντίνος Λαμπρινουδάκης

Τμήμα: Ψηφιακών Συστημάτων

Πειραιάς

2026

Περίληψη

Η παρούσα διπλωματική εργασία παρουσιάζει το *SupplyGuard*, ένα policy-driven πλαίσιο προεγκαταστασιακής άμυνας για την ασφάλεια της Python software supply chain. Το πρόβλημα αφορά την εγκατάσταση κακόβουλων ή ύποπτων Python packages από οικοσυστήματα όπως το PyPI, όπου μια συνηθισμένη εντολή `pip install` μπορεί να εισαγάγει κώδικα με obfuscation, ύποπτη επικοινωνία δικτύου, install-time λογική ή άλλες επικίνδυνες ενδείξεις.

Το SupplyGuard υλοποιείται ως PyPI-compatible gateway/proxy. Κάθε package artifact αναλύεται πριν εγκατασταθεί, με static-only διαδικασία: δεν εκτελείται κώδικας, δεν γίνεται `import` του package και δεν ενεργοποιούνται build-time ή runtime hooks. Η ανάλυση οργανώνεται σε πέντε engines: static rules, machine learning risk scoring, gateway static scanning, provenance/typosquatting/dependency-confusion checks και artifact integrity checks. Τα επιμέρους outputs συνδυάζονται από fusion policy, το οποίο παράγει τελική ενέργεια εγκατάστασης και αντίστοιχο evidence/audit record.

Στο frozen benchmark της εργασίας, με 700 assumed-clean και 3,701 malicious-labeled artifacts, το πλήρες A-E fusion policy σταμάτησε πριν από την εγκατάσταση 3,358 από τα 3,701 malicious-labeled artifacts (90.73%). Το clean false-403 rate ήταν 114/700 (16.29%), ενώ το permanent block σε clean δείγματα ήταν 1/700 (0.14%). Το Engine B, βασισμένο σε Random Forest, πέτυχε στο εσωτερικό held-out test set accuracy 0.9466, macro-F1 0.9465 και ROC-AUC 0.9838. Σε frozen shape-matched external αξιολόγηση, το ίδιο ML signal διατήρησε μέτρια διάκριση με ROC-AUC 0.8427, χωρίς όμως να υποστηρίζει ισχυρισμό πλήρους source-independent generalization.

Η συνεισφορά της εργασίας είναι ένα εξηγήσιμο pre-install risk-reduction framework και όχι ένας απόλυτος malware detector. Το σύστημα δείχνει ότι ο συνδυασμός πολλαπλών ατελών σημάτων, οργανωμένων από policy-driven fusion, μπορεί να μειώσει τον κίνδυνο πριν από την εγκατάσταση Python packages, διατηρώντας παράλληλα ορατότητα σε evidence, review και audit διαδικασίες.

Λέξεις-κλειδιά: Software Supply Chain Security, PyPI, Python Packages, Static Analysis, Machine Learning, Random Forest, Pre-install Defense, Fusion Policy, Dependency Confusion, Typosquatting

Abstract

This thesis presents *SupplyGuard*, a policy-driven pre-installation defense framework for Python software supply-chain security. The addressed problem is the installation of malicious or suspicious Python packages from ecosystems such as PyPI, where a normal `pip install` command may introduce code containing obfuscation, suspicious network indicators, install-time logic, or other risk signals.

SupplyGuard is implemented as a PyPI-compatible gateway/proxy. Each package artifact is inspected before installation using a static-only process: package code is not executed, the package is not imported, and build-time or runtime hooks are not triggered. The analysis is organized around five engines: static rules, machine-learning risk scoring, gateway static scanning, provenance/typosquatting/dependency-confusion checks, and artifact integrity checks. Their outputs are combined by a fusion policy that produces the final installation action and an evidence/audit record.

In the frozen benchmark used in this thesis, consisting of 700 assumed-clean and 3,701 malicious-labeled artifacts, the full A-E fusion policy stopped 3,358 of 3,701 malicious-labeled artifacts before installation (90.73%). The clean false-403 rate was 114/700 (16.29%), while the permanent clean block rate was 1/700 (0.14%). Engine B, based on a Random Forest classifier, achieved 0.9466 accuracy, 0.9465 macro-F1 and 0.9838 ROC-AUC on the internal held-out test set. In a frozen shape-matched external evaluation, the same ML signal retained moderate discrimination with 0.8427 ROC-AUC, without supporting a full source-independent generalization claim.

The contribution of this thesis is an explainable pre-install risk-reduction framework rather than an absolute malware detector. SupplyGuard demonstrates that multiple imperfect risk signals, organized through policy-driven fusion, can reduce risk before Python packages are installed while preserving evidence, review and auditability.

Keywords: Software Supply Chain Security, PyPI, Python Packages, Static Analysis, Machine Learning, Random Forest, Pre-install Defense, Fusion Policy, Dependency Confusion, Typosquatting

Περιεχόμενα

Περίληψη	i
Abstract	ii
1 Problem Definition	1
1.1 Εισαγωγή στο πρόβλημα	1
1.2 Python Package Supply-Chain Attacks	1
1.3 Περιορισμοί παραδοσιακών προσεγγίσεων	2
1.4 Ανάγκη για Pre-Installation Defense	2
1.5 Στόχος της εργασίας	2
1.6 Research Questions	2
1.7 Contributions	3
2 Related Work and Background	4
2.1 Software Supply-Chain Security	4
2.2 Python/PyPI Package Ecosystem	4
2.3 Malicious Packages, Typosquatting and Dependency Confusion	5
2.4 Static Analysis and Machine Learning	6
2.5 Pre-Install Controls and Research Gap	6
3 Methodology	8
3.1 Methodology Overview	8
3.2 Pre-Installation Analysis Pipeline	8
3.3 Static-Only Scope	8
3.4 Artifact Collection and Preparation	9
3.5 Feature Extraction	9
3.6 Multi-Engine Analysis Methodology	9
3.7 Fusion, Enforcement and Evidence Methodology	9
3.8 Experimental Methodology	9
4 System Architecture	11
4.1 Architecture Overview	11
4.2 PyPI-Compatible Gateway	11

4.3	Backend API	11
4.4	Frontend / SOC Dashboard	12
4.5	Persistence Layer	12
4.6	Review Workflow	12
4.7	Governance and Auditability	12
5	Engine Descriptions	13
5.1	Overview	13
5.2	Engine A -- Static Rules	13
5.3	Engine B -- Sentinel v4.6 Machine Learning	13
5.4	Engine C -- Gateway Static Scanner	14
5.5	Engine D -- Provenance, Typosquatting and Dependency Confusion	14
5.6	Engine E -- Artifact Integrity	14
5.7	Comparative Summary	15
6	Fusion Policy	16
6.1	Motivation	16
6.2	Why Multiple Signals Are Required	16
6.3	Decision Categories	16
6.4	Engine Roles	16
6.5	Fusion Rules / Decision Logic	16
6.6	Example Fusion Decision	17
6.7	REVIEW and HOLD as Human-in-the-Loop States	17
6.8	False Positives and Policy Trade-offs	17
6.9	Gateway Enforcement Mapping	18
7	Experimental Setup	19
7.1	Environment	19
7.2	Dataset / Artifact Universe	19
7.3	Frozen Benchmark	19
7.4	ML Experimental Setup	19
7.5	Algorithm Comparison Method	19
7.6	Engine-Level Evaluation Method	19
7.7	Fusion / Ablation Method	20
7.8	External / Shape-Matched Evaluation Method	20
7.9	Controlled Synthetic Packages and Live Gateway Setup	20

7.10	Requirements File Intake	20
7.11	Evaluation Metrics	20
7.12	Experimental Evaluation Overview	20
7.13	Scope and Exclusions	20
8	Experimental Results and Discussion	22
8.1	Dataset Composition	22
8.2	Machine Learning Results	22
8.3	Algorithm Comparison	22
8.4	Engine-Level Results	23
8.5	Fusion Policy Results	24
8.6	Ablation Study	24
8.7	External / Shape-Matched Evaluation	25
8.8	Controlled Red-Team / Open-Set Results	27
8.9	Live Gateway Demonstration	27
8.10	Discussion of Research Questions	27
9	Conclusion	29
9.1	Summary of the Work	29
9.2	Main Findings	29
9.3	Practical Value	29
9.4	Limitations	29
9.5	Future Work	29
9.6	Final Conclusion	29
	Βιβλιογραφία	30

Κατάλογος σχημάτων

3.1	Pre-installation analysis pipeline implemented by SupplyGuard.	8
4.1	High-level architecture of the SupplyGuard pre-installation defense framework.	11
6.1	Multi-engine signal fusion and policy-driven decision flow in SupplyGuard. .	17
7.1	Overview of the experimental evaluation methodology used for SupplyGuard.	21
8.1	Σύγκριση Weighted F1 και ROC-AUC των machine-learning algorithms που αξιολογήθηκαν.	23
8.2	Ablation comparison of malicious-labeled artifacts stopped before installation and clean false-403 rate across engine configurations.	25

1. Problem Definition

1.1 Εισαγωγή στο πρόβλημα

Η ανάπτυξη λογισμικού βασίζεται όλο και περισσότερο σε εξωτερικά packages, βιβλιοθήκες και αυτοματοποιημένους μηχανισμούς εγκατάστασης. Στην Python, το `pip` και το `PyPI` αποτελούν βασικό κανάλι διανομής κώδικα[5, 4]. Αυτή η ευκολία δημιουργεί και σημείο κινδύνου: ένα package μπορεί να εγκατασταθεί γρήγορα σε developer workstation, virtual environment ή CI/CD pipeline χωρίς ουσιαστικό έλεγχο του artifact που τελικά εισάγεται στο περιβάλλον.

Σε ένα τέτοιο μοντέλο εμπιστοσύνης, ο επιτιθέμενος δεν χρειάζεται απαραίτητα να παραβιάσει άμεσα το σύστημα του χρήστη. Μπορεί να δημοσιεύσει παραπλανητικό package, να εκμεταλλευτεί όνομα που μοιάζει με νόμιμο package, να εισαγάγει κακόβουλη έκδοση σε dependency ή να αξιοποιήσει λάθος ρύθμιση package index. Η ζημιά μπορεί να ξεκινήσει από μια τυπική εντολή:

```
pip install package-name
```

Το κρίσιμο σημείο είναι ο χρόνος του ελέγχου. Αν ο έλεγχος γίνει μετά την εγκατάσταση ή μετά το `import`, πιθανή install-time ή runtime λογική μπορεί να έχει ήδη ενεργοποιηθεί.

1.2 Python Package Supply-Chain Attacks

Οι επιθέσεις σε package ecosystems εκμεταλλεύονται την εμπιστοσύνη ανάμεσα σε developers, package repositories και package managers. Στην περίπτωση της Python, σημαντικές κατηγορίες κινδύνου είναι το typosquatting, το dependency confusion, η malicious package publication, τα encoded payloads, η dynamic execution και η install-time ή import-time συμπεριφορά. Η βιβλιογραφία για package managers ερμηνευόμενων γλωσσών έχει δείξει ότι τέτοιες επιθέσεις αξιοποιούν αδυναμίες στο dependency resolution, στη διαχείριση ονομάτων και στην ανεπαρκή αξιολόγηση του κώδικα πριν από την εγκατάσταση[15, 8].

Το dependency confusion έγινε γνωστό ως πρακτική επίθεση όταν δημόσια packages με ονόματα εσωτερικών dependencies εγκαταστάθηκαν σε περιβάλλοντα μεγάλων οργανισμών[16]. Παρόμοια, το typosquatting βασίζεται στη μικρή απόκλιση ονομάτων και στην πιθανότητα ανθρώπινου λάθους. Αυτές οι επιθέσεις είναι ιδιαίτερα επικίνδυνες επειδή συχνά χρησιμοποιούν φυσιολογικές ροές ανάπτυξης λογισμικού.

1.3 Περιορισμοί παραδοσιακών προσεγγίσεων

Παραδοσιακές λύσεις όπως signature-based detection, post-install scanning ή γενικά antivirus controls δεν καλύπτουν πλήρως το συγκεκριμένο σημείο κινδύνου. Οι υπογραφές εξαρτώνται από γνωστά μοτίβα. Η ανάλυση μετά την εγκατάσταση ενδέχεται να γίνει αργά. Ένας μεμονωμένος detector μπορεί να δώσει χρήσιμο σήμα, αλλά δεν αρκεί για όλες τις κατηγορίες κινδύνου.

Η ασφάλεια της software supply chain απαιτεί συνδυασμό τεχνικών και οργανωτικών ελέγχων. Οδηγίες όπως το NIST SSDF και το NIST SP 800-204D τονίζουν την ανάγκη για ασφαλείς πρακτικές ανάπτυξης, ελέγχους dependencies και ενσωμάτωση ασφάλειας σε CI/CD pipelines[10, 11]. Το SupplyGuard τοποθετείται σε αυτό το πλαίσιο ως τεχνικό pre-install control point για Python packages.

1.4 Ανάγκη για Pre-Installation Defense

Ο στόχος του pre-install defense είναι να μεταφερθεί ο έλεγχος πριν από την είσοδο του package στο περιβάλλον. Αντί ο χρήστης να εγκαθιστά απευθείας από το PyPI ή από άλλο index, το request δρομολογείται μέσω gateway. Το gateway αναλύει το artifact και επιβάλλει policy πριν σερβίρει το αρχείο στο `pip`.

Αυτή η λογική δεν ισχυρίζεται ότι αποδεικνύει απόλυτα αν ένα package είναι κακόβουλο. Στόχος είναι η μείωση κινδύνου πριν από την εγκατάσταση, με σαφή evidence και δυνατότητα ανθρώπινης αξιολόγησης όταν τα σήματα είναι αβέβαια.

1.5 Στόχος της εργασίας

Η εργασία στοχεύει στη σχεδίαση, υλοποίηση και αξιολόγηση ενός policy-driven gateway που αναλύει Python package artifacts πριν εγκατασταθούν. Το σύστημα πρέπει να ανακτά artifacts, να εκτελεί static-only ανάλυση, να εξάγει risk signals, να εφαρμόζει fusion policy και να καταγράφει evidence για review/audit. Η κύρια αξιολόγηση εστιάζει στη συμπεριφορά του συστήματος σε frozen benchmark και σε συμπληρωματικά controlled demonstrations.

1.6 Research Questions

Τα ερευνητικά ερωτήματα είναι:

1. **RQ1:** Μπορεί ένα pre-installation gateway να μειώσει τις εγκαταστάσεις malicious-labeled Python packages;
2. **RQ2:** Προσφέρει το multi-engine fusion καλύτερη κάλυψη από μεμονωμένα engines;

3. **RQ3:** Ποιο είναι το λειτουργικό κόστος σε clean false positives και human review burden;
4. **RQ4:** Ποιοι περιορισμοί προκύπτουν από μια static-only προσέγγιση;

1.7 Contributions

Η εργασία συνεισφέρει: ένα PyPI-compatible pre-install gateway, static-only artifact analysis, πέντε συμπληρωματικά risk engines, policy-driven fusion, evidence/audit workflow και πειραματική αξιολόγηση με πραγματικά frozen metrics. Η συνεισφορά είναι αμυντικό framework μείωσης κινδύνου, όχι απόλυτο σύστημα ανίχνευσης malware.

2. Related Work and Background

2.1 Software Supply-Chain Security

Η software supply chain περιλαμβάνει τον κώδικα μιας εφαρμογής, τις third-party dependencies, τα build εργαλεία, τα package registries, τα CI/CD pipelines και τους μηχανισμούς deployment. Σε σύγχρονα έργα λογισμικού, μεγάλο μέρος της λειτουργικότητας προέρχεται από εξωτερικά packages. Αυτό δημιουργεί μεγάλα dependency graphs, όπου ένα πρόβλημα σε ένα upstream package μπορεί να μεταφερθεί downstream σε πολλά projects. Μελέτες σε language-based package ecosystems έχουν δείξει ότι η δομή και η εξέλιξη των dependency networks επηρεάζουν την έκταση της διάδοσης τεχνικού και security risk[6, 7].

Οι open-source supply-chain attacks δεν απαιτούν πάντα άμεση παραβίαση του τελικού οργανισμού. Ο επιτιθέμενος μπορεί να εισαγάγει κακόβουλο κώδικα σε package, να εκμεταλλευτεί παραπλανητική ονοματολογία, να στοχεύσει maintainer account ή να αξιοποιήσει automated dependency installation σε build pipelines. Η εργασία των Ohm et al. καταγράφει πραγματικά περιστατικά open-source supply-chain επιθέσεων και δείχνει ότι οι επιθέσεις αυτές συχνά εκτελούνται μέσα από κανονικά κανάλια διανομής λογισμικού[8]. Αντίστοιχα, η ENISA αναδεικνύει τη downstream φύση του κινδύνου: η παραβίαση ενός προμηθευτή ή ενός component μπορεί να επηρεάσει πολλούς καταναλωτές λογισμικού[9].

Σε επίπεδο πρακτικής άμυνας, frameworks και οδηγίες όπως το NIST SSDF, το NIST SP 800-204D, το OWASP Software Supply Chain Security Cheat Sheet και το SLSA δίνουν έμφαση σε secure development, dependency governance, provenance και CI/CD security controls[10, 11, 12, 14]. Το SupplyGuard τοποθετείται μέσα σε αυτό το ευρύτερο πλαίσιο, αλλά εστιάζει σε ένα συγκεκριμένο σημείο ελέγχου: την προεγκαταστασιακή αξιολόγηση Python package artifacts πριν αυτά περάσουν στο περιβάλλον του developer ή του CI.

2.2 Python/PyPI Package Ecosystem

Το PyPI αποτελεί βασικό registry για Python packages, ενώ το `pip` επιτρέπει εγκατάσταση από το PyPI ή από custom indexes που ακολουθούν τη λογική του Simple Repository API[5, 4, 1, 2, 3]. Αυτή η δυνατότητα είναι λειτουργικά χρήσιμη, αλλά σημαίνει ότι η εμπιστοσύνη στο package index και στο resolution path αποτελεί μέρος του threat model. Ένα package που εμφανίζεται σε registry μπορεί να εισαχθεί γρήγορα σε virtual environment, container build ή CI job, συχνά χωρίς manual inspection.

Η έρευνα των Duan et al. εξετάζει supply-chain attacks σε package managers ερμηνευόμενων γλωσσών και αναδεικνύει κενά εμπιστοσύνης και ασφάλειας στα οικοσυστήματα

αυτά[15]. Το ενδιαφέρον για το SupplyGuard βρίσκεται στο ότι οι package managers εγκαθιστούν artifacts με μεγάλη ευκολία, ενώ ο έλεγχος του πραγματικού artifact συχνά μεταφέρεται σε μεταγενέστερο στάδιο. Ένα PyPI-compatible gateway μπορεί να αξιοποιήσει τη νόμιμη δυνατότητα χρήσης custom index, παρεμβάλλοντας έλεγχο πριν την παράδοση του artifact στο pip.

2.3 Malicious Packages, Typosquatting and Dependency Confusion

Οι malicious packages σε registries όπως npm, PyPI και RubyGems έχουν μελετηθεί ως πρακτικός μηχανισμός εισαγωγής κακόβουλου κώδικα σε downstream projects. Τα περιστατικά που συγκεντρώνονται από τους Ohm et al. περιλαμβάνουν διαφορετικές τεχνικές injection και execution, ενώ οι Duan et al. αναφέρουν κατηγορίες συμπεριφορών όπως install-time execution, ύποπτη δικτυακή επικοινωνία και πρόσβαση σε ευαίσθητα δεδομένα[8, 15]. Στο npm ecosystem, η εργασία των Zimmermann et al. δείχνει ότι η κεντρικότητα packages και maintainers δημιουργεί σημαντικό systemic risk[7].

Το typosquatting βασίζεται σε lexical similarity και σε common typo patterns. Ο επιτιθέμενος δημιουργεί package με όνομα που μοιάζει με δημοφιλές package, στοχεύοντας σε ανθρώπινο λάθος ή σε απρόσεκτη επιλογή dependency. Οι Vu et al. μελετούν typosquatting και combosquatting ειδικά στο Python ecosystem και προτείνουν αυτοματοποιημένο εντοπισμό υποψήφιων παραπλανητικών ονομάτων[18]. Οι Taylor et al. παρουσιάζουν το SpellBound/TypoGard για εντοπισμό package typosquatting σε npm και PyPI, χρησιμοποιώντας lexical similarity και popularity-aware λογική[19]. Η νεότερη εργασία των Neupane et al. επεκτείνει το ζήτημα πέρα από απλά typos, δείχνοντας ότι package confusion μπορεί να προκύψει και από semantic ή άλλες μη καθαρά συντακτικές ομοιότητες[20].

Η βιβλιογραφία αυτή υποστηρίζει μια κρίσιμη σχεδιαστική επιλογή: ένα typosquatting ή package-confusion signal δεν πρέπει να θεωρείται από μόνο του απόδειξη malware. Η ομοιότητα ονόματος μπορεί να παράγει false positives, ειδικά όταν υπάρχουν forks, παρόμοια εργαλεία ή μικρά projects με νόμιμο σκοπό. Για αυτό το SupplyGuard αντιμετωπίζει το Engine D ως provenance/naming risk signal και όχι ως autonomous permanent blocker.

Το dependency confusion αφορά διαφορετικό αλλά συναφές πρόβλημα: ένα public package μπορεί να έχει ίδιο ή συμβατό όνομα με internal dependency και να επιλεγεί από τον resolver λόγω index precedence ή λάθος configuration. Το περιστατικό που περιέγραψε ο Birsan ανέδειξε πρακτικά τον κίνδυνο namespace collision μεταξύ private και public dependencies[16]. Το OWASP κατατάσσει το dependency chain abuse στα CI/CD supply-chain risks, επισημαίνοντας ότι ο έλεγχος dependencies πρέπει να συνδυάζεται με πολιτικές

χρήσης package sources[17, 13]. Το SupplyGuard δεν ισχυρίζεται πλήρη λύση dependency confusion, αλλά ενσωματώνει provenance checks ως ένα από τα signals του fusion.

2.4 Static Analysis and Machine Learning

Η static analysis χρησιμοποιείται ευρέως επειδή επιτρέπει έλεγχο κώδικα και metadata χωρίς εκτέλεση untrusted package. Για pre-install screening, αυτό είναι σημαντικό: ο έλεγχος μπορεί να γίνει γρήγορα και χωρίς να ενεργοποιηθούν install-time ή runtime hooks. Σε package security, στατικά σήματα μπορεί να είναι suspicious imports, shell/process execution, dynamic execution, network-related code, encoded strings, obfuscation, install scripts και metadata inconsistencies. Μεγάλης κλίμακας μελέτη στα Python packages του PyPI έχει δείξει ότι static analysis μπορεί να χρησιμοποιηθεί για security-oriented inspection του οικοσυστήματος, αλλά τα αποτελέσματα χρειάζονται προσεκτική ερμηνεία λόγω false positives και contextual ambiguity[21].

Οι περιορισμοί της static analysis είναι εξίσου σημαντικοί. Ένα στατικό finding δεν αποδεικνύει ότι η αντίστοιχη συμπεριφορά θα εκτελεστεί. Αντίθετα, runtime-only triggers, environment-dependent payloads, obfuscation και dynamic loading μπορούν να μειώσουν την ορατότητα. Επομένως, ένα static-only σύστημα πρέπει να παρουσιάζεται ως risk-reduction layer και όχι ως πλήρης behavioral sandbox.

Το machine learning χρησιμοποιείται συχνά για να συνδυάζει πολλά static ή metadata features σε classification ή ranking signal. Το Random Forest είναι καθιερωμένο ensemble μοντέλο για structured feature vectors[23, 24, 25]. Σε malicious package detection, η εργασία των Sejffia και Schäfer παρουσιάζει το Amalfi ως machine-learning-based approach για detection malicious npm packages, συνδυάζοντας classifiers με πρόσθετους ελέγχους για να περιορίσουν false positives και false negatives[22]. Η αξία για το SupplyGuard είναι μεθοδολογική: το ML μπορεί να βοηθήσει στη διάκριση ύποπτων packages, αλλά η πιθανότητα που παράγει δεν ισοδυναμεί με απόδειξη malware.

Συνεπώς, το Engine B του SupplyGuard παράγει probabilistic risk score και όχι τελική ετυμηγορία. Η χρήση του μέσα σε fusion policy απαντά στους γνωστούς περιορισμούς supervised learning: dataset bias, class representation, distribution shift και ανάγκη external validation.

2.5 Pre-Install Controls and Research Gap

Η σχετική βιβλιογραφία καλύπτει σημαντικές πλευρές του προβλήματος: dependency-network risk, malicious package corpora, typosquatting, package confusion, static analysis, ML-based detection, provenance και CI/CD security guidance. Ωστόσο, οι δυνατότητες αυτές

συχνά μελετώνται ως ξεχωριστοί detection ή governance μηχανισμοί. Το κενό που κινητοποιεί το SupplyGuard είναι η ενοποίησή τους σε ένα πρακτικό pre-install enforcement path.

Συγκεκριμένα, η εργασία εξετάζει τον συνδυασμό PyPI-compatible gateway, artifact-level static inspection, heterogeneous risk engines, policy-driven fusion, gateway enforcement πριν από την εγκατάσταση και evidence/review/audit workflow. Η θέση της εργασίας δεν είναι ότι δεν υπάρχει προηγούμενη έρευνα σε malicious package detection ή supply-chain security. Αντίθετα, η βιβλιογραφία δείχνει ότι υπάρχουν πολλές χρήσιμες αλλά μερικές τεχνικές, οι οποίες δημιουργούν κίνητρο για μια ενιαία pre-install αρχιτεκτονική με explainable policy decisions.

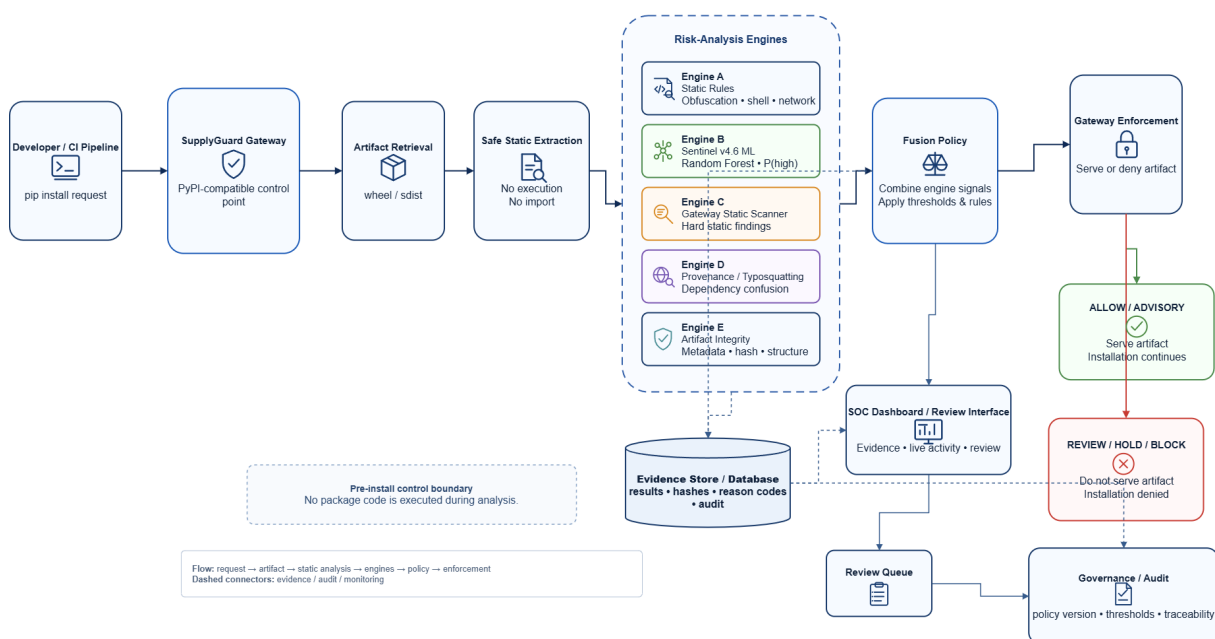
3. Methodology

3.1 Methodology Overview

Η μεθοδολογία αντιμετωπίζει κάθε package ως artifact που πρέπει να αξιολογηθεί πριν από την εγκατάσταση. Η ανάλυση δεν ξεκινά από το όνομα του package αλλά από το αρχείο που πρόκειται να δοθεί στο `pip`: wheel ή source distribution. Τα outputs των επιμέρους ελέγχων μετατρέπονται σε risk signals και καταλήγουν σε policy decision, όπως ορίζεται αναλυτικά στο Chapter 6.

3.2 Pre-Installation Analysis Pipeline

Το pipeline αποτελείται από λήψη request, ανάκτηση artifact, ασφαλή εξαγωγή, feature extraction, engine-level analysis, fusion policy, enforcement και evidence logging. Το Σχήμα 3.1 συνοψίζει τη ροή χωρίς να υπονοεί runtime εκτέλεση.



Σχήμα 3.1: Pre-installation analysis pipeline implemented by SupplyGuard.

3.3 Static-Only Scope

Η κύρια ανάλυση είναι static-only: δεν εκτελεί package code, δεν κάνει import, δεν ενεργοποιεί `setup.py`, build hooks ή runtime behavior, και δεν επιτρέπει network/system interaction από το package. Ο περιορισμός αυτός μειώνει το ρίσκο του analysis path, αλλά

δεν επιτρέπει επιβεβαίωση runtime-only συμπεριφορών. Επομένως, τα findings ερμηνεύονται ως risk signals.

3.4 Artifact Collection and Preparation

Κάθε artifact αποθηκεύεται, ταυτοποιείται με hash, εξάγεται σε προσωρινό χώρο και αναλύεται ως συγκεκριμένη έκδοση. Το σύστημα συλλέγει metadata, dependency declarations, Python files, configuration files και install-related αρχεία. Η artifact-level ανάλυση είναι απαραίτητη επειδή διαφορετικές εκδόσεις του ίδιου package μπορεί να έχουν διαφορετικό περιεχόμενο.

3.5 Feature Extraction

Τα features καλύπτουν suspicious imports, process/shell execution indicators, network-related patterns, encoded ή obfuscated content, dynamic execution, install-time files, credential indicators, metadata inconsistencies και naming/provenance signals. Τα features δεν θεωρούνται απόδειξη κακόβουλης συμπεριφοράς· χρησιμοποιούνται ως είσοδοι σε rules, ML και policy logic.

3.6 Multi-Engine Analysis Methodology

Η multi-engine προσέγγιση χρησιμοποιείται για να καλύψει διαφορετικές πλευρές του ίδιου artifact. Ένα engine μπορεί να είναι ισχυρό σε code-level indicators, άλλο σε provenance, και άλλο σε integrity. Οι τεχνικές λεπτομέρειες των Engines A--E παρουσιάζονται στο Chapter 5. Στο παρόν κεφάλαιο η βασική μεθοδολογική επιλογή είναι ο συνδυασμός συμπληρωματικών και όχι ανταγωνιστικών σημάτων.

3.7 Fusion, Enforcement and Evidence Methodology

Το fusion policy μετατρέπει τα engine outputs σε τελική ενέργεια εγκατάστασης. Το gateway εφαρμόζει την ενέργεια πριν σερβίρει το artifact. Παράλληλα, κάθε απόφαση αποθηκεύεται με engine outputs, reason codes, artifact metadata και review/audit πληροφορία. Η καταγραφή αυτή επιτρέπει επανεξέταση αποφάσεων και σύνδεση πειραματικών αποτελεσμάτων με συγκεκριμένα evidence.

3.8 Experimental Methodology

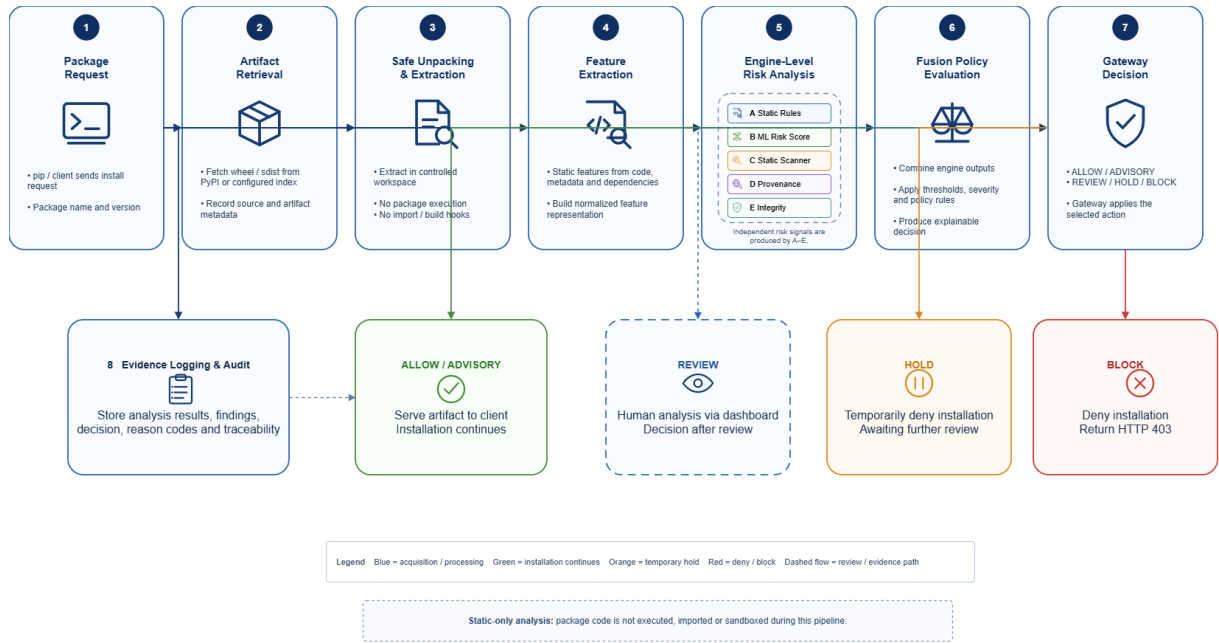
Η αξιολόγηση οργανώθηκε σε τέσσερα επίπεδα: ML evaluation, engine-level analysis, fusion/ablation evaluation και live gateway demonstration. Το κύριο benchmark είναι frozen:

model, thresholds και policy δεν τροποποιούνται κατά την τελική μέτρηση. Αυτό περιορίζει τον κίνδυνο εκ των υστέρων προσαρμογής στα δεδομένα.

4. System Architecture

4.1 Architecture Overview

Η υλοποίηση αποτελείται από PyPI-compatible gateway, backend API, risk engines, fusion layer, persistence layer και SOC-style dashboard. Το Σχήμα 4.1 δείχνει τη θέση κάθε υποσυστήματος στη ροή εγκατάστασης και στις παράλληλες λειτουργίες evidence, review και governance.



Σχήμα 4.1: High-level architecture of the SupplyGuard pre-installation defense framework.

4.2 PyPI-Compatible Gateway

Το gateway εκθέτει endpoints συμβατά με τη λογική του Simple Repository API και του `pip` index resolution[1, 2, 4]. Έτσι, ένα installation request μπορεί να δρομολογηθεί μέσω SupplyGuard με `--index-url`. Αν το policy επιτρέπει το artifact, το gateway το σερβίρει στο `pip`. Αν το policy παράγει blocking/review outcome, το artifact δεν παραδίδεται.

4.3 Backend API

Το backend συντονίζει artifact retrieval, static extraction, engine execution, fusion decision, persistence και δεδομένα dashboard. Η υλοποίηση δεν αντιμετωπίζει τα engines ως ανεξάρτητα scripts αλλά ως ενσωματωμένα services μέσα σε API-driven workflow.

4.4 Frontend / SOC Dashboard

Το frontend απευθύνεται σε administrator ή security analyst. Παρουσιάζει analyzed packages, decision history, engine findings, review queue, governance information και audit logs. Η λειτουργία του είναι να κάνει το enforcement εξηγήσιμο, όχι να αντικαταστήσει το gateway.

4.5 Persistence Layer

Για κάθε package record αποθηκεύονται package name, version, artifact hash, engine outputs, fusion decision, reason codes, review status και audit metadata. Η persistence layer υποστηρίζει αναπαραγωγικότητα, debugging και τεκμηρίωση αποτελεσμάτων.

4.6 Review Workflow

Οι αβέβαιες περιπτώσεις μπορούν να σταματήσουν προσωρινά και να οδηγηθούν σε ανθρώπινο review. Το workflow αυτό είναι σχεδιασμένο ως ελεγχόμενος συμβιβασμός: προτιμάται προσωρινό stop με evidence αντί για σιωπηλή εγκατάσταση όταν τα σήματα δεν είναι αρκετά καθαρά για οριστικό block.

4.7 Governance and Auditability

Η governance layer εκθέτει την ενεργή πολιτική, τη συμμετοχή engines και τις βασικές ρυθμίσεις απόφασης. Αυτό είναι σημαντικό για πειραματικό έλεγχο, διότι διαφορετική policy version ή διαφορετικά thresholds θα παρήγαγαν μη συγκρίσιμα αποτελέσματα.

5. Engine Descriptions

5.1 Overview

Το κεφάλαιο αυτό είναι η κύρια τεχνική περιγραφή των Engines A--E. Κάθε engine ορίζεται ως ανεξάρτητη πηγή evidence με συγκεκριμένο input, μηχανισμό ανάλυσης, output και policy role. Η τελική ενέργεια δεν παράγεται εδώ αλλά στο fusion policy.

5.2 Engine A -- Static Rules

Purpose. Το Engine A εντοπίζει στατικά suspicious indicators μέσα στον κώδικα και στη δομή του artifact.

Input. Εξαγμένα Python files, metadata, configuration files και install-related αρχεία.

Mechanism. Εφαρμόζει κανόνες για encoded payloads, high-entropy strings, suspicious names, dynamic execution, shell/process indicators, network indicators, sensitive path access και install-time patterns.

Output and role. Παράγει advisory findings και explainable evidence. Δεν χρησιμοποιείται ως αυτόνομος hard blocker, επειδή αρκετά patterns μπορεί να εμφανιστούν και σε νόμιμα packages.

Limitations. Τα rule findings έχουν θόρυβο και εξαρτώνται από το coverage των κανόνων. Δεν επιβεβαιώνουν runtime εκτέλεση.

5.3 Engine B -- Sentinel v4.6 Machine Learning

Purpose. Το Engine B παράγει πιθανότητα υψηλού κινδύνου $P(\text{high})$ από static feature vector.

Input. Structured static features που περιγράφουν imports, execution indicators, obfuscation, network patterns, metadata και install-related χαρακτηριστικά.

Mechanism. Χρησιμοποιεί Random Forest classifier[23, 25]. Το ensemble πολλών decision trees μειώνει την εξάρτηση από ένα μόνο rule path και ταιριάζει σε tabular/static features.

Output and role. Το output είναι risk score και thresholded signal. Στο active policy, υψηλό ML score δεν αρκεί από μόνο του για permanent block· χρησιμοποιείται ως ισχυρό input στο fusion.

Limitations. Το μοντέλο εξαρτάται από training data, feature design και distribution similarity. Η external shape-matched αξιολόγηση στο Chapter 8 δείχνει ότι το signal παραμένει χρήσιμο αλλά όχι απόλυτα γενικεύσιμο.

5.4 Engine C -- Gateway Static Scanner

Purpose. Το Engine C ελέγχει το artifact στο enforcement path, λίγο πριν σερβιριστεί στο `pip`.

Input. Το extracted artifact και τα gateway-level static findings.

Mechanism. Εντοπίζει ισχυρότερα indicators όπως `shell/process execution`, `encoded execution chains`, `install-time triggers`, `dynamic execution`, `suspicious network+execution combinations` και `native/system-level signals`.

Output and role. Μπορεί να συμμετέχει σε `hard enforcement`. Όταν συμφωνεί με άλλο ισχυρό signal, ειδικά με Engine B, η policy μπορεί να οδηγήσει σε `block`.

Limitations. Static findings μπορεί να επηρεάσουν `clean packages` με νόμιμη σύνθετη λειτουργικότητα. Γι' αυτό η policy δεν ερμηνεύει όλα τα findings ως ίδιας βαρύτητας.

5.5 Engine D -- Provenance, Typosquatting and Dependency Confusion

Purpose. Το Engine D καλύπτει κινδύνους ονοματολογίας και προέλευσης που δεν φαίνονται απαραίτητα στον κώδικα.

Input. Package name, normalized name, registry/source context, known package corpus και internal/public naming assumptions.

Mechanism. Εξετάζει `typosquatting`, `similarity` με γνωστά packages, `dependency-confusion context` και `provenance anomalies`.

Output and role. Παράγει `advisory/review signal`. Δεν λειτουργεί ως αυτόνομος `permanent blocker`, επειδή η ομοιότητα ονομάτων δεν αποδεικνύει από μόνη της κακόβουλη συμπεριφορά.

Limitations. Η αποτελεσματικότητα εξαρτάται από το corpus γνωστών ονομάτων και από το κατά πόσο είναι διαθέσιμο αξιόπιστο context για εσωτερικά dependencies.

5.6 Engine E -- Artifact Integrity

Purpose. Το Engine E ελέγχει τη συνέπεια και ακεραιότητα του artifact ως αντικείμενο εγκατάστασης.

Input. Artifact archive, file list, hashes, metadata και extracted structure.

Mechanism. Εξετάζει `metadata consistency`, `name/version mismatch`, `missing` ή `unexpected files`, `RECORD/hash mismatches` και δομικές ανωμαλίες.

Output and role. Σοβαρά integrity failures μπορούν να συμμετέχουν σε `hard enforcement`, επειδή μειώνουν την αξιοπιστία του artifact.

Limitations. Η ακεραιότητα δεν ισοδυναμεί με ασφάλεια. Ένα συνεπές artifact μπορεί να περιέχει επικίνδυνη λογική, ενώ μικρές ασυνέπειες μπορεί να μην είναι κακόβουλες.

5.7 Comparative Summary

Engine	Primary purpose	Main input	Output	Enforcement role
A	Static suspicious indicators	Extracted files and metadata	Advisory findings	Evidence only
B	ML risk scoring	Static feature vector	P(high) / risk zone	Fusion input
C	Gateway static scanner	Artifact at gateway path	Hard static findings	Hard-enforcement capable
D	Provenance and naming risk	Name, source and corpus	Advisory/review signal	Corroborating signal
E	Artifact integrity	Archive, metadata and hashes	Integrity result	Hard-enforcement capable

Πίνακας 5.1: Summary of the SupplyGuard risk-analysis engines.

6. Fusion Policy

6.1 Motivation

Το fusion policy είναι το μοναδικό επίπεδο που μετατρέπει engine outputs σε τελική ενέργεια. Ο διαχωρισμός αυτός περιορίζει την υπερβολική εξάρτηση από ένα μόνο signal και επιτρέπει διαφορετικό βάρος ανά engine.

6.2 Why Multiple Signals Are Required

Οι supply-chain επιθέσεις δεν αφήνουν πάντα το ίδιο είδος ίχνους. Άλλες φαίνονται στον κώδικα, άλλες στο όνομα, άλλες στα metadata και άλλες μόνο ως συνδυασμός ενδείξεων. Ένα isolated warning μπορεί να είναι νόμιμο· αντίθετα, η συμφωνία ανεξάρτητων signals αυξάνει την εμπιστοσύνη στην απόφαση.

6.3 Decision Categories

Το SupplyGuard χρησιμοποιεί πέντε τελικές κατηγορίες:

- **ALLOW**: το artifact σερβίρεται στο `pip` χωρίς blocking condition.
- **ADVISORY**: η εγκατάσταση επιτρέπεται, αλλά καταγράφονται προειδοποιητικά findings.
- **REVIEW**: η εγκατάσταση σταματά και απαιτείται ανθρώπινη αξιολόγηση.
- **HOLD**: η εγκατάσταση σταματά λόγω αυξημένου κινδύνου ή αβεβαιότητας.
- **BLOCK**: η εγκατάσταση απορρίπτεται βάσει ισχυρής policy condition.

6.4 Engine Roles

Το Engine A παρέχει context/evidence, το Engine B probabilistic risk score, το Engine C gateway hard signal, το Engine D provenance/naming signal και το Engine E integrity signal. Η policy εξετάζει τη βαρύτητα και τη συμφωνία τους.

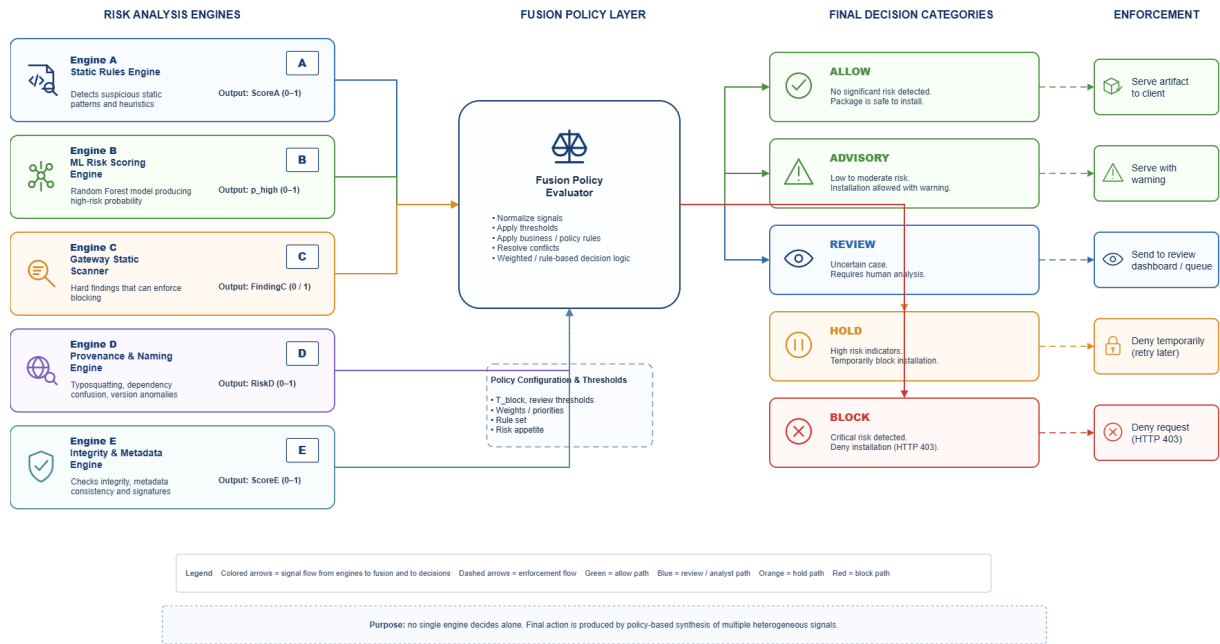
6.5 Fusion Rules / Decision Logic

Σε υψηλό επίπεδο, advisory-only findings οδηγούν σε καταγραφή χωρίς block, ενώ hard signals ή συνδυασμοί υψηλού κινδύνου οδηγούν σε REVIEW, HOLD ή BLOCK. Τα thresholds και οι κανόνες είναι frozen στην τελική αξιολόγηση, ώστε τα αποτελέσματα να είναι συγκρίσιμα.

6.6 Example Fusion Decision

B:block(p=0.975 >= Tblock=0.6267) + C:block -> BLOCK

Το παράδειγμα δείχνει συμφωνία μεταξύ probabilistic ML signal και hard gateway scanner finding. Η απόφαση δεν προκύπτει μόνο από το P(high) ούτε μόνο από ένα rule, αλλά από policy condition που απαιτεί συνδυασμό signals.



Σχήμα 6.1: Multi-engine signal fusion and policy-driven decision flow in SupplyGuard.

6.7 REVIEW and HOLD as Human-in-the-Loop States

REVIEW και HOLD είναι ενδιάμεσες αμυντικές καταστάσεις. Δεν σημαίνουν ότι το package αποδείχθηκε κακόβουλο. Σημαίνουν ότι το ρίσκο ή η αβεβαιότητα είναι αρκετά υψηλή ώστε η εγκατάσταση να μη συνεχιστεί χωρίς έλεγχο.

6.8 False Positives and Policy Trade-offs

Η αυστηρή policy αυξάνει την κάλυψη malicious-labeled artifacts αλλά μπορεί να αυξήσει clean false-403 outcomes. Η εργασία μετρά αυτό το trade-off στο Chapter 8, διαχωρίζοντας προσωρινά stops από permanent clean blocks.

6.9 Gateway Enforcement Mapping

ALLOW και ADVISORY αντιστοιχούν σε εγκατάσταση που επιτρέπεται. REVIEW, HOLD και BLOCK αντιστοιχούν σε άρνηση παράδοσης του artifact από το gateway, πρακτικά HTTP 403 στο installation path.

7. Experimental Setup

7.1 Environment

Η αξιολόγηση πραγματοποιήθηκε σε τοπικό εργαστηριακό περιβάλλον με backend API, gateway endpoints και frontend dashboard. Τα live gateway tests χρησιμοποιούν `pip` με `custom --index-url`, ενώ τα offline experiments εφαρμόζονται σε extracted package artifacts.

7.2 Dataset / Artifact Universe

Το κύριο frozen benchmark περιλαμβάνει 4,401 artifacts: 700 assumed-clean και 3,701 malicious-labeled. Κάθε artifact αξιολογείται ως συγκεκριμένο wheel ή source distribution. Τα labels ερμηνεύονται πειραματικά: assumed-clean δεν σημαίνει αποδεδειγμένα ασφαλές, και malicious-labeled δεν σημαίνει πλήρη κάλυψη όλων των malware μορφών.

7.3 Frozen Benchmark

Το model, οι thresholds και η policy έκδοση παρέμειναν frozen κατά την τελική μέτρηση. Το protocol καταγράφεται ως static-only, χωρίς install/execution/network/retraining/tuning στο analysis path.

7.4 ML Experimental Setup

Το Engine B αξιολογήθηκε με deduplicated development cohort και stratified 70/15/15 split. Το deployed frozen model είναι SupplyGuard Sentinel v4.6 Random Forest με 105 static features. Η έξοδος είναι `P(high)`.

7.5 Algorithm Comparison Method

Συγκρίθηκαν Random Forest, Gradient Boosting, Decision Tree, Logistic Regression, KNN και SVM με κοινό threshold 0.5 στο test split των 730 samples. Η επιλογή μοντέλου βασίστηκε σε weighted F1, ROC-AUC και σταθερότητα.

7.6 Engine-Level Evaluation Method

Για κάθε engine μετρήθηκε η ενεργοποίηση σε malicious-labeled και assumed-clean artifacts. Τα αποτελέσματα ερμηνεύονται με βάση τον ρόλο κάθε engine: advisory, risk scoring ή hard enforcement.

7.7 Fusion / Ablation Method

To ablation συγκρίνει single-engine και combined configurations με το full A--E policy. Ο σκοπός είναι να μετρηθεί αν το fusion προσθέτει operational value σε σχέση με μεμονωμένα signals.

7.8 External / Shape-Matched Evaluation Method

Για το Engine B χρησιμοποιήθηκε frozen shape-matched external cohort 230 ζευγών / 460 artifacts. Τα ζεύγη ταιριάχθηκαν σε package-shape χαρακτηριστικά. Η αξιολόγηση χαρακτηρίζεται U2 source-overlapping και δεν υποστηρίζει full source-independent claim.

7.9 Controlled Synthetic Packages and Live Gateway Setup

To controlled red-team/open-set set περιλαμβάνει inert malicious-intent packages και benign controls. Τα live demonstrations δείχνουν clean installation μέσω gateway και controlled suspicious denial με HTTP 403. Αυτά είναι demonstration evidence και όχι αντικατάσταση του frozen benchmark.

7.10 Requirements File Intake

To prototype υποστηρίζει και intake από dependency files όπως `requirements.txt`. Η λειτουργία αυτή επιτρέπει project-level visibility, αλλά δεν αντικαθιστά το gateway enforcement.

7.11 Evaluation Metrics

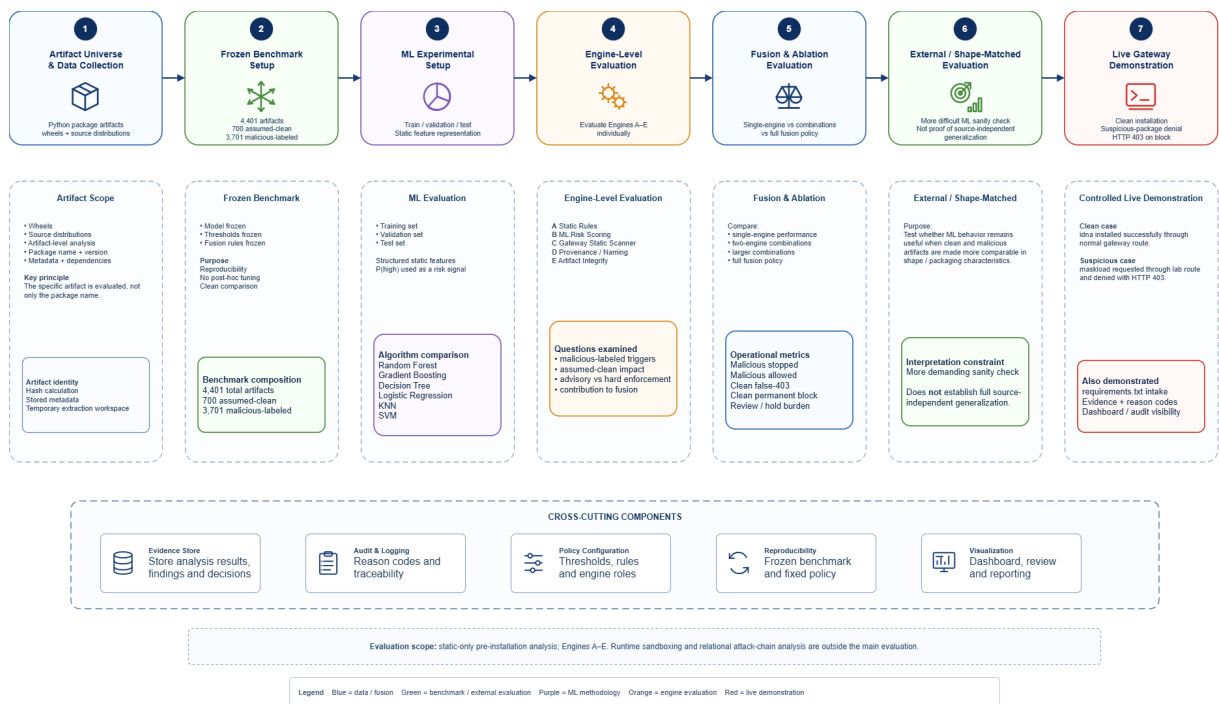
Χρησιμοποιούνται ML metrics όπως accuracy, precision, recall, F1, ROC-AUC και confusion matrix[26]. Για το συνολικό σύστημα χρησιμοποιούνται operational metrics: malicious-labeled pre-install stopped, malicious-labeled allowed, clean false-403, clean permanent block, review/hold burden και decision distribution.

7.12 Experimental Evaluation Overview

Το Σχήμα 7.1 συνοψίζει τη μεθοδολογία αξιολόγησης.

7.13 Scope and Exclusions

Η κύρια αξιολόγηση αφορά static-only A--E pipeline. Runtime sandboxing και relational attack-chain analysis αναφέρονται ως future work. Engine F δεν αποτελεί μέρος των official results.



Σχήμα 7.1: Overview of the experimental evaluation methodology used for SupplyGuard.

8. Experimental Results and Discussion

8.1 Dataset Composition

Class	Artifacts	Percentage
Assumed-clean	700	15.91%
Malicious-labeled	3,701	84.09%
Total	4,401	100.00%

Πίνακας 8.1: Composition of the frozen benchmark.

8.2 Machine Learning Results

To Engine B v4.6 Random Forest αξιολογήθηκε σε held-out test split $n = 730$. Τα αποτελέσματα στον Πίνακα 8.2 δείχνουν ισχυρό internal risk signal, όχι αυτόνομη απόδειξη κακόβουλης συμπεριφοράς.

Metric	Value
Accuracy	0.9466
Macro-F1	0.9465
High-class precision	0.9704
High-class recall	0.9188
High-class F1	0.9439
ROC-AUC	0.9838

Πίνακας 8.2: Internal held-out results for Engine B / Sentinel v4.6 Random Forest.

	Predicted clean	Predicted high-risk
Actual clean	363	10
Actual high-risk	29	328

Πίνακας 8.3: Confusion matrix for the Engine B held-out test split.

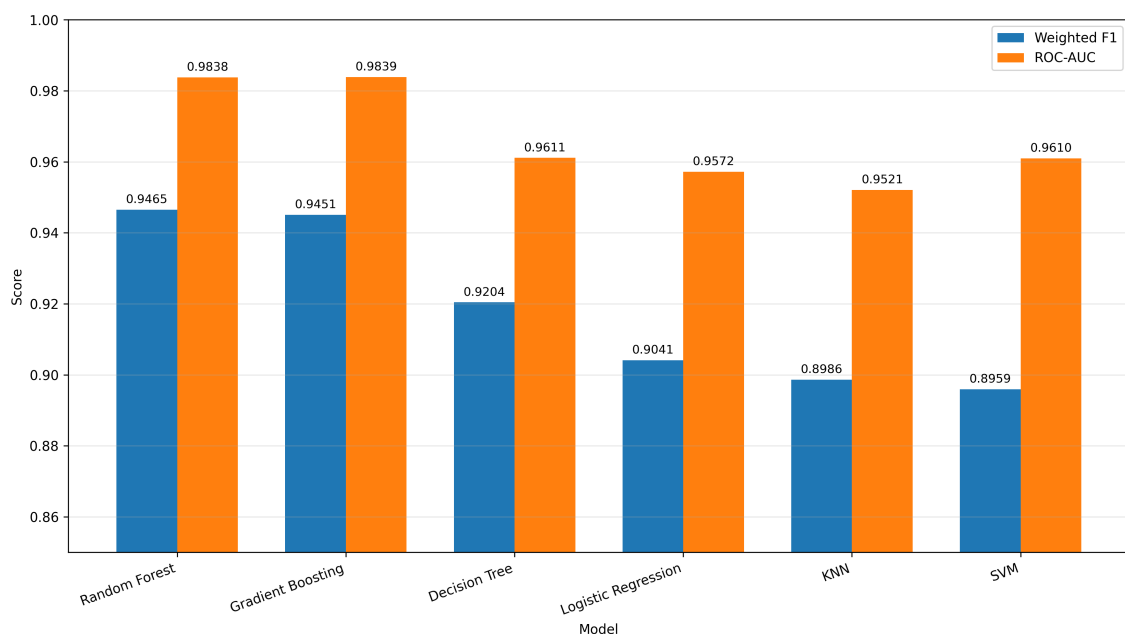
8.3 Algorithm Comparison

Η μικρή διαφορά μεταξύ Random Forest και Gradient Boosting αποτυπώνεται επίσης στο Σχήμα 8.1. Το Gradient Boosting έχει οριακά υψηλότερο ROC-AUC, ενώ το Random Forest

Model	Accuracy	Weighted F1	ROC-AUC	High recall
Random Forest	0.9466	0.9465	0.9838	0.9188
Gradient Boosting	0.9452	0.9451	0.9839	0.9188
Decision Tree	0.9205	0.9204	0.9611	0.8908
Logistic Regression	0.9041	0.9041	0.9572	0.9076
KNN	0.8986	0.8986	0.9521	0.8992
SVM	0.8959	0.8959	0.9610	0.8880

Πίνακας 8.4: Algorithm comparison on the test split.

διατηρεί ελαφρά καλύτερο Weighted F1. Η επιλογή Random Forest δεν παρουσιάζεται ως συντριπτική επικράτηση, αλλά ως ισορροπημένη επιλογή για το frozen structured-feature pipeline.



Σχήμα 8.1: Σύγκριση Weighted F1 και ROC-AUC των machine-learning algorithms που αξιολογήθηκαν.

8.4 Engine-Level Results

Το Engine A έχει υψηλή ενεργοποίηση και στις δύο κλάσεις, άρα είναι χρήσιμο κυρίως για context. Το Engine B έχει καλή malicious coverage με πολύ χαμηλό clean block-zone impact. Το Engine C φέρει σημαντικό enforcement burden και απαιτεί προσεκτική policy χρήση. Τα Engines D και E καλύπτουν ειδικές κατηγορίες κινδύνου.

Engine	Malicious coverage	Clean impact	Role
A	76.03%	82.57%	Advisory/evidence only; noisy but explainable.
B	71.90%	0.29%	ML block-zone signal; hypothetical standalone, not autonomous permanent block.
C	42.96%	16.00%	Gateway hard-enforcement capable.
D	4.65%	0.00%	Provenance/naming review signal.
E	26.61%	0.00%	Deterministic integrity hard signal.

Πίνακας 8.5: Engine-level behavior in the frozen benchmark.

8.5 Fusion Policy Results

To full A-E policy σταμάτησε πριν από την εγκατάσταση 3,358 από τα 3,701 malicious-labeled artifacts. Ταυτόχρονα, 114 από τα 700 assumed-clean artifacts έλαβαν false-403 outcome, με 1 permanent clean block.

Metric	Value	Rate
Malicious-labeled stopped pre-install	3,358 / 3,701	90.73%
Malicious-labeled install-allowed	343 / 3,701	9.27%
Clean false-403	114 / 700	16.29%
Clean permanent block	1 / 700	0.14%
Processing errors	0 / 4,401	0.00%

Πίνακας 8.6: Main frozen benchmark outcomes for the full A-E fusion policy.

Decision	Clean	Malicious-labeled	Total
ALLOW	586	305	891
ADVISORY	0	38	38
REVIEW	2	1,073	1,075
HOLD	111	1,799	1,910
BLOCK	1	486	487
Total	700	3,701	4,401

Πίνακας 8.7: Distribution of final fusion decisions in the frozen benchmark.

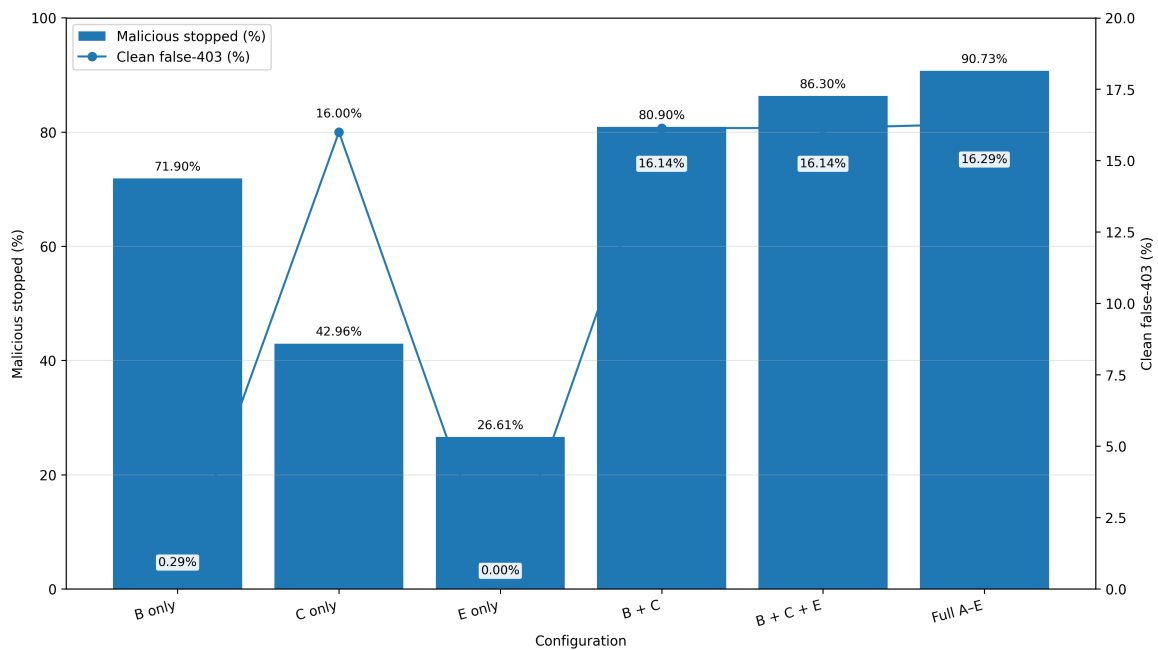
8.6 Ablation Study

Το Σχήμα 8.2 αποτυπώνει οπτικά το trade-off μεταξύ malicious coverage και clean false-403 rate καθώς προστίθενται engines στο fusion policy. Κανένα single-engine configuration δεν φτάνει την κάλυψη του full fusion. Το B-only signal έχει πολύ μικρό clean impact αλλά

Configuration	Malicious stopped	Clean false-403	Notes
B only (hypothetical)	2,661 / 3,701 (71.90%)	2 / 700 (0.29%)	ML alone, not production policy.
C only	1,590 / 3,701 (42.96%)	112 / 700 (16.00%)	Gateway scanner only.
E only	985 / 3,701 (26.61%)	0 / 700 (0.00%)	Integrity only.
B + C	2,994 / 3,701 (80.90%)	113 / 700 (16.14%)	Core ML/static agreement.
B + C + E	3,194 / 3,701 (86.30%)	113 / 700 (16.14%)	Adds integrity signal.
Full A--E	3,358 / 3,701 (90.73%)	114 / 700 (16.29%)	Final frozen policy.

Πίνακας 8.8: Ablation study comparing single-engine and combined policies.

χαμηλότερη malicious coverage, ενώ το C-only path αυξάνει το enforcement coverage με σημαντικότερο clean false-403 κόστος. Το B+C+E φτάνει 86.30%, ενώ το full A--E policy φτάνει 90.73%, με μικρό πρόσθετο operational cost.



Σχήμα 8.2: Ablation comparison of malicious-labeled artifacts stopped before installation and clean false-403 rate across engine configurations.

8.7 External / Shape-Matched Evaluation

Η external αξιολόγηση δείχνει ότι το ML signal διατηρεί μέτρια διάκριση όταν ελέγχονται βασικά package-shape χαρακτηριστικά. Όμως το cohort είναι U2 source-overlapping και δεν υποστηρίζει claim πλήρους source-independent generalization. Η μείωση από το internal ROC-AUC 0.9838 σε 0.8427 επιβεβαιώνει την ανάγκη να χρησιμοποιείται το ML μέσα σε fusion policy και όχι ως αυτόνομος detector.

Metric	Value
Matched pairs	230
Total artifacts	460
ROC-AUC	0.8427
PR-AUC	0.8219
Accuracy	0.7935
Macro-F1	0.7918
Malicious precision	0.8571
Malicious recall	0.7043
Clean hard-FP	0.0696
Pairs where malicious score > clean score	87.39%

Πίνακας 8.9: Frozen shape-matched external evaluation of Engine B.

Outcome	Count / Rate
Inert malicious-intent samples	16
Hard-blocked	10 (62.50%)
Advisory-only	3
Complete misses	3
Any reaction	81.25%
Benign controls	3
Benign false positives	0

Πίνακας 8.10: Controlled red-team/open-set evaluation summary.

8.8 Controlled Red-Team / Open-Set Results

To set είναι μικρό και controlled, άρα δεν αποτελεί κύριο benchmark. Είναι όμως χρήσιμο για να δείξει blind spots, όπως serialization/dynamic-attribute cases, και για να επιβεβαιώσει ότι provenance-only σενάρια μπορούν να εμφανιστούν ως advisory αντί για block.

8.9 Live Gateway Demonstration

Στο clean demonstration, package όπως το idna εγκαταστάθηκε μέσω gateway όταν δεν υπήρχε hard policy condition. Στο controlled suspicious demonstration, το maskload ζητήθηκε μέσω lab route και το gateway επέστρεψε HTTP 403 πριν από την εγκατάσταση.

```
python -m pip install --no-cache-dir \
  --index-url http://127.0.0.1:8000/simple \
  --trusted-host 127.0.0.1 \
  idna
```

```
python -m pip download --no-cache-dir --no-deps \
  --index-url http://127.0.0.1:8000/lab-simple \
  --trusted-host 127.0.0.1 \
  maskload==7.80
# Expected outcome in the controlled lab route: HTTP 403
  before installation.
```

To demonstration τεκμηριώνει τη λειτουργικότητα του gateway path. Τα στατιστικά συμπεράσματα παραμένουν βασισμένα στο frozen benchmark.

8.10 Discussion of Research Questions

RQ1

Question: Μπορεί ένα pre-installation gateway να μειώσει τις εγκαταστάσεις malicious-labeled packages;

Finding: Ναι, στο evaluated benchmark σταμάτησε 3,358/3,701 malicious-labeled artifacts.

Evidence: Πίνακας 8.6.

Interpretation: Το αποτέλεσμα υποστηρίζει risk reduction εντός του frozen static-only setup, όχι πλήρη malware detection.

RQ2

Question: Προσφέρει το fusion καλύτερη κάλυψη από single engines;

Finding: Το full A--E policy πέτυχε 90.73%, ενώ το καλύτερο single-engine signal ήταν 71.90%.

Evidence: Πίνακας 8.8.

Interpretation: Η συμπληρωματικότητα των engines προσθέτει μετρήσιμη αξία.

RQ3

Question: Ποιο είναι το κόστος σε clean false positives και review burden;

Finding: Το clean false-403 ήταν 16.29%, αλλά το clean permanent block 0.14%.

Evidence: Πίνακες 8.6 και 8.7.

Interpretation: Η policy είναι συντηρητική: προτιμά προσωρινό stop/review σε αβέβαιες περιπτώσεις.

RQ4

Question: Ποιοι περιορισμοί προκύπτουν από static-only analysis;

Finding: Η external αξιολόγηση και το controlled set δείχνουν μειωμένη γενίκευση και blind spots σε runtime/dynamic-only patterns.

Evidence: Πίνακες 8.9 και 8.10.

Interpretation: Το SupplyGuard είναι pre-install risk-reduction layer και χρειάζεται συμπληρωματικές τεχνικές για runtime-only behavior.

9. Conclusion

9.1 Summary of the Work

Η εργασία παρουσίασε το SupplyGuard, ένα PyPI-compatible, policy-driven pre-install framework για Python package supply-chain security. Το σύστημα αναλύει artifacts με static-only pipeline, συνδυάζει outputs από πέντε engines και εφαρμόζει gateway enforcement πριν από την εγκατάσταση.

9.2 Main Findings

Στο frozen benchmark, το full A-E fusion policy σταμάτησε 90.73% των malicious-labeled artifacts πριν από την εγκατάσταση. Το clean false-403 rate ήταν 16.29%, αλλά το clean permanent block παρέμεινε 0.14%. Το ablation study έδειξε ότι το fusion προσφέρει ουσιαστική βελτίωση έναντι single-engine configurations. Το ML component είχε ισχυρά internal metrics αλλά πιο περιορισμένη external shape-matched απόδοση.

9.3 Practical Value

Το prototype δείχνει ότι η ερευνητική ιδέα μπορεί να ενσωματωθεί σε πραγματική pipeline μέσω custom index URL. Το dashboard, η review queue και το evidence logging βοηθούν τον analyst να κατανοήσει γιατί ένα artifact επιτράπηκε ή σταμάτησε.

9.4 Limitations

Η εργασία δεν ισχυρίζεται πλήρη προστασία ούτε πλήρη ανίχνευση zero-day malware. Η static-only ανάλυση δεν επιβεβαιώνει runtime behavior. Τα labels είναι πειραματικά και το gateway προστατεύει μόνο requests που περνούν μέσα από αυτό.

9.5 Future Work

Μελλοντικές επεκτάσεις περιλαμβάνουν runtime sandbox analysis, relational attack-chain analysis, αυστηρότερα source-independent datasets, καλύτερο provenance intelligence και enterprise integration με private registries και CI/CD pipelines.

9.6 Final Conclusion

Συνολικά, το SupplyGuard τεκμηριώνει ότι ένας PyPI-compatible pre-installation gateway μπορεί να μειώσει τον κίνδυνο εγκατάστασης malicious-labeled packages όταν static-only analysis, multi-engine risk signals και policy-driven fusion συνδυάζονται με evidence και auditability. Εντός των ορίων του frozen benchmark²⁹, το αποτέλεσμα είναι αμυντικό risk-reduction framework και όχι απόλυτος malware oracle.

Βιβλιογραφία

- [1] Python Packaging Authority, ``Simple Repository API," Python Packaging User Guide. [Online]. Available: <https://packaging.python.org/en/latest/specifications/simple-repository-api/>. Accessed: 2026-08-24.
- [2] D. Holth and D. Stuft, ``PEP 503 -- Simple Repository API," Python Enhancement Proposals, 2015. [Online]. Available: <https://peps.python.org/pep-0503/>. Accessed: 2026-08-24.
- [3] Python Package Index, ``Index API," PyPI Documentation. [Online]. Available: <https://docs.pypi.org/api/index-api/>. Accessed: 2026-08-24.
- [4] Python Packaging Authority, ``pip install," pip documentation. [Online]. Available: https://pip.pypa.io/en/stable/cli/pip_install/. Accessed: 2026-08-24.
- [5] Python Software Foundation, ``Python Package Index." [Online]. Available: <https://pypi.org/>. Accessed: 2026-08-24.
- [6] R. Kikas, G. Gousios, M. Dumas, and D. Pfahl, ``Structure and Evolution of Package Dependency Networks," in *Proc. 2017 IEEE/ACM 14th International Conference on Mining Software Repositories (MSR)*, 2017, pp. 102--112, doi: 10.1109/MSR.2017.55.
- [7] M. Zimmermann, C.-A. Staicu, C. Tenny, and M. Pradel, ``Small World with High Risks: A Study of Security Threats in the npm Ecosystem," in *Proc. 28th USENIX Security Symposium*, 2019, pp. 995--1010.
- [8] M. Ohm, H. Plate, A. Sykosch, and M. Meier, ``Backstabber's Knife Collection: A Review of Open Source Software Supply Chain Attacks," in *Detection of Intrusions and Malware, and Vulnerability Assessment (DIMVA 2020)*, Lecture Notes in Computer Science, vol. 12223, Springer, 2020, pp. 23--43, doi: 10.1007/978-3-030-52683-2_2.
- [9] European Union Agency for Cybersecurity (ENISA), ``Threat Landscape for Supply Chain Attacks," 2021. [Online]. Available: <https://www.enisa.europa.eu/publications/threat-landscape-for-supply-chain-attacks>. Accessed: 2026-08-24.
- [10] National Institute of Standards and Technology, ``Secure Software Development Framework (SSDF) Version 1.1," NIST SP 800-218, 2022. [Online]. Available: <https://csrc.nist.gov/projects/ssdf>. Accessed: 2026-08-24.

- [11] National Institute of Standards and Technology, ``Strategies for the Integration of Software Supply Chain Security in DevSecOps CI/CD Pipelines," NIST SP 800-204D, 2023. [Online]. Available: <https://csrc.nist.gov/pubs/sp/800/204/d/final>. Accessed: 2026-08-24.
- [12] OWASP Foundation, ``Software Supply Chain Security Cheat Sheet." [Online]. Available: https://cheatsheetseries.owasp.org/cheatsheets/Software_Supply_Chain_Security_Cheat_Sheet.html. Accessed: 2026-08-24.
- [13] OWASP Foundation, ``OWASP Top 10 CI/CD Security Risks." [Online]. Available: <https://owasp.org/www-project-top-10-ci-cd-security-risks/>. Accessed: 2026-08-24.
- [14] Open Source Security Foundation, ``SLSA Specification." [Online]. Available: <https://slsa.dev/spec/>. Accessed: 2026-08-24.
- [15] R. Duan, O. Alrawi, R. P. Kasturi, R. Elder, B. Saltaformaggio, and W. Lee, ``Towards Measuring Supply Chain Attacks on Package Managers for Interpreted Languages," in *Proc. Network and Distributed System Security Symposium (NDSS)*, 2021, doi: 10.14722/ndss.2021.23055.
- [16] A. Birsan, ``Dependency Confusion: How I Hacked Into Apple, Microsoft and Dozens of Other Companies," 2021. [Online]. Available: <https://medium.com/@alex.birsan/dependency-confusion-4a5d60fec610>. Accessed: 2026-08-24.
- [17] OWASP Foundation, ``CICD-SEC-3: Dependency Chain Abuse," OWASP Top 10 CI/CD Security Risks. [Online]. Available: <https://owasp.org/www-project-top-10-ci-cd-security-risks/CICD-SEC-03-Dependency-Chain-Abuse/>. Accessed: 2026-08-24.
- [18] D.-L. Vu, I. Pashchenko, F. Massacci, H. Plate, and A. Sabetta, ``Typosquatting and Combosquatting Attacks on the Python Ecosystem," in *Proc. 2020 IEEE European Symposium on Security and Privacy Workshops (EuroS&PW)*, 2020, pp. 509--514, doi: 10.1109/EuroSPW51379.2020.00074.
- [19] M. Taylor, R. K. Vaidya, D. Davidson, L. De Carli, and V. Rastogi, ``Defending Against Package Typosquatting," in *Network and System Security: 14th International Conference (NSS 2020)*, Lecture Notes in Computer Science, vol. 12570, Springer, 2020, pp. 112--131, doi: 10.1007/978-3-030-65745-1_7.
- [20] S. Neupane, G. Holmes, E. Wyss, D. Davidson, and L. De Carli, ``Beyond Typosquatting: An In-depth Look at Package Confusion," in *Proc. 32nd USENIX Security Symposium*, 2023, pp. 3439--3456.

- [21] J. Ruohonen, K. Hjerpe, and K. Rindell, "A Large-Scale Security-Oriented Static Analysis of Python Packages in PyPI," in *Proc. 2021 18th International Conference on Privacy, Security and Trust (PST)*, 2021, pp. 1--10, doi: 10.1109/PST52912.2021.9647791.
- [22] A. Seifia and M. Schäfer, "Practical Automated Detection of Malicious npm Packages," in *Proc. 2022 ACM/IEEE 44th International Conference on Software Engineering (ICSE)*, 2022, pp. 1681--1692, doi: 10.1145/3510003.3510104.
- [23] L. Breiman, "Random Forests," *Machine Learning*, vol. 45, pp. 5--32, 2001, doi: 10.1023/A:1010933404324.
- [24] C. M. Bishop, *Pattern Recognition and Machine Learning*. Springer, 2006.
- [25] scikit-learn developers, "RandomForestClassifier." [Online]. Available: <https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html>. Accessed: 2026-08-24.
- [26] scikit-learn developers, "Model evaluation: quantifying the quality of predictions." [Online]. Available: https://scikit-learn.org/stable/modules/model_evaluation.html. Accessed: 2026-08-24.