УДК 004.056.5:004.41

Нуров Ф. С.

студент

Поволжский государственный технологический университет

Россия, Йошкар-Ола

АЛГОРИТМ ЗАПРЕТА ПОДМЕНЫ ЗАВИСИМОСТЕЙ В ГИБРИДНЫХ РЕЕСТРАХ ПАКЕТОВ НА ОСНОВЕ ПРИВЯЗКИ ПРОСТРАНСТВА ИМЕН К ИСТОЧНИКУ

Аннотация. Гибридная схема, в которой сборка одновременно использует внутренний и общедоступный реестры пакетов, создает риск подмены зависимости: совпадающее или похожее имя может разрешиться в пользу недоверенного источника. Цель исследования - разработать детерминированный алгоритм допуска зависимости, не зависящий от порядка опроса реестров и правил выбора версии конкретного менеджера пакетов. Метод объединяет нормализацию идентификатора, привязку пространства имен к разрешенному источнику, проверку полного графа транзитивных зависимостей, а также фиксацию конфигурации resolver, хеша артефакта и происхождения записи кэша. На модельном наборе из 30 случаев в экосистемах npm, NuGet и Python базовые политики закрыли 0, 9 и 12 из 24 опасных случаев соответственно, тогда как предложенный составной алгоритм закрыл 24 из 24; все четыре политики пропустили шесть заранее допустимых случаев. Полученные показатели характеризуют логическое покрытие сконструированного набора, а не статистику реальных атак или безопасность реализаций; практическая ценность работы состоит в едином контракте admission control, проверяемом в pull request и CI/CD до исполнения установочных скриптов.

Ключевые слова: цепочка поставки программного обеспечения, подмена зависимостей, гибридный реестр пакетов, пространство имен, привязка к источнику, транзитивная зависимость, admission control.

Nurov F. S.
Student
Volga State University of Technology
Russia, Yoshkar-Ola

ALGORITHM FOR PREVENTING DEPENDENCY SUBSTITUTION IN HYBRID PACKAGE REGISTRIES BASED ON NAMESPACE-TO-SOURCE BINDING

Abstract. A hybrid build that resolves packages from both private and public registries is exposed to dependency substitution: an identical or confusingly similar name may be selected from an untrusted source. This study develops a deterministic dependency-admission algorithm that does not depend on registry query order or a package manager's version-selection rules. The method combines identifier normalization, namespace-to-source binding, inspection of the complete transitive graph, and validation of the resolver configuration hash, artifact digest, and cache-record provenance. In a model set of 30 cases across the npm, NuGet, and Python ecosystems, the baseline policies rejected 0, 9, and 12 of 24 unsafe cases, respectively, whereas the proposed composite algorithm rejected all 24; all four policies admitted the six predefined legitimate cases. These figures characterize the logical coverage of the constructed set rather than real-world attack statistics or implementation security; the practical contribution is a uniform admission-control contract that can be checked in pull requests and CI/CD before package installation scripts are executed.

Keywords: software supply chain, dependency confusion, hybrid package registry, namespace, source binding, transitive dependency, admission control.

Введение

Современный репозиторий содержит не только исходный код, но и декларации зависимостей, lock-файлы и конфигурацию пакетного клиента. Сборка часто использует внутренние компоненты и открытые библиотеки одновременно. Если клиент опрашивает несколько реестров, имя

внутреннего пакета становится внешним идентификатором: регистрация совпадающего имени в публичном реестре, публикация большей версии или изменение порядка источников может привести к загрузке не того артефакта.

Исследования package confusion показывают, что проблема не ограничивается опечатками. Анализ более 1200 документированных атак выделил 13 механизмов семантической и синтаксической путаницы [1]. Экосистемные исследования npm выявляют высокую связанность и транзитивное распространение риска [2], а каталог реальных атак на цепочки поставки демонстрирует повторяемость вредоносных пакетов и учетных компрометаций [3]. Следовательно, контроль должен охватывать не только прямые зависимости и не только имя.

Официальные средства экосистем предлагают отдельные меры. npm рекомендует scoped packages для частных компонентов [6]. NuGet Package Source Mapping требует сопоставить каждый прямой и транзитивный идентификатор одному или нескольким источникам и предупреждает о влиянии глобального кэша [7]. Python Packaging User Guide разделяет зеркало, кэш и proxy-индекс; для изолированной сборки возможна установка без обращения к сети [8, 9]. Эти механизмы различаются, поэтому организациям с несколькими языками нужен общий проверяемый контракт.

Цель работы – разработать алгоритм допуска пакета в гибридной схеме частных и публичных реестров. Научная новизна состоит в том, что привязка имени к источнику рассматривается совместно с доказательством фактического разрешения, состоянием кэша и хешем конфигурации resolver. Алгоритм должен выдавать одинаковый verdict до выполнения установочного скрипта и сохранять объяснимый код отказа.

Методы и принципы исследования

Зависимость d представлена кортежем (e,n,v,h,s,p,c), где e – экосистема; n – нормализованное имя; v – разрешенная версия; h – криптографический хеш артефакта; s – фактический источник; p – проверяемые сведения о происхождении или подписи, если они доступны в экосистеме; c – запись кэша. Конфигурация разрешения r включает перечень реестров, правила приоритета, proxy/fallback, доверенные корни, сетевую политику и собственный хеш hr.

Политика владения M связывает шаблон имени с множеством разрешенных источников. Для внутреннего пространства имен M(n) содержит ровно один контролируемый источник; публичные имена разрешаются через утвержденный proxy или конкретный публичный реестр. Отсутствие записи означает deny-by-default. Несколько источников для одного точного имени считаются неоднозначностью и требуют отдельного исключения; порядок в конфигурационном файле не заменяет это правило.

Функция Admit(d,r,M) возвращает allow только при выполнении восьми условий: имя нормализовано по правилам экосистемы; имя покрыто M; фактический s принадлежит M(n); версия и h закреплены доверенным lock/manifest; hr совпадает с проверенной конфигурацией; каждый транзитивный узел прошел те же проверки; запись c содержит источник и хеш либо кэш очищен; direct URL и install scripts удовлетворяют отдельной политике. Для критического выпуска дополнительно проверяется доступное происхождение p, но его отсутствие не маскируется фиктивной подписью: политика явно фиксирует уровень доверия.

Алгоритм выполняется дважды. На этапе pull request анализируется декларация и ожидаемый граф; неизвестная зависимость блокирует слияние или создает заявку на владение именем. На этапе сборки

проверяется фактический граф после разрешения, но до исполнения скриптов пакета. Несовпадение ожидаемого и фактического графов, hr или источника блокирует сборку. Итоговый список (n,v,h,s,hr,verdict) сохраняется рядом с provenance артефакта.

Для аналитической проверки создано восемь классов опасных случаев: A1 – публичная версия выше внутренней; A2 – одинаковая версия в двух источниках; A3 – опечатка или семантически похожее имя; A4 – транзитивная коллизия; A5 – унаследованная конфигурация добавляет реестр; A6 – кэш не содержит подтвержденный источник; A7 – прямая URL-ссылка без хеша; A8 – подмена lock-файла вместе с зависимостью. Каждый класс представлен в npm, NuGet и Python, всего 24 случая. Дополнены шесть допустимых случаев: внутренний и публичный пакет в каждой экосистеме с корректным источником и хешем.

Сравниваются четыре политики: P0 – несколько реестров без привязки источника; P1 – только версия и хеш доверенного lock-файла; P2 – только namespace-to-source mapping; P3 – предложенный составной алгоритм. Статический оракул считает случай закрытым, если нарушение одного из обязательных предикатов гарантированно дает deny до выполнения кода пакета. Метод не моделирует вероятность атаки и не проверяет отсутствие уязвимостей в реализации клиента.

Таблица 1. Предикаты допуска и коды отказа

Предикат	**Проверка**	**Код отказа**
NORM	каноническое имя и допустимый scope/prefix	E-NAME
OWNER	для имени существует единственный владелец источника	E-OWNER
SOURCE	фактический реестр входит в M(n)	E-SOURCE
LOCK	версия и digest закреплены доверенным изменением	E-LOCK

Предикат	Проверка	Код отказа
RESOLVER	хеш конфигурации и proxy/fallback совпадают	E-CONFIG
TRANSITIVE	все узлы графа прошли тот же контракт	E-TRANSITIVE
CACHE	источник и digest кэша доказаны либо кэш пуст	E-CACHE
EXEC	URL и install script разрешены отдельной политикой	E-EXEC

Основные результаты

P0 не закрыла ни одного из 24 опасных случаев по строгому критерию. Реальный resolver может случайно выбрать корректный пакет, но политика не запрещает противоположный исход и потому не создает доказательства. P1 закрыла A1, A2 и A7 во всех трех экосистемах – 9 случаев: доверенный digest не совпадает с подставленным артефактом или отсутствует для прямой ссылки. Однако P1 не помогает, если атакующий изменил сам lock-файл в том же запросе, если имя ошибочно, либо если кэш скрывает источник.

P2 закрыла A1, A2, A4 и A5 – 12 случаев: разрешение из неправильного реестра блокируется независимо от версии и порядка. Она не обнаруживает похожее, но формально публичное имя A3; не гарантирует источник уже извлеченного кэша A6; не покрывает direct URL A7; и не связывает измененный lock-файл A8 с одобренной конфигурацией.

P3 закрыла все 24 случая в модельном наборе: A3 – правилом владения и заявкой на новое имя; A6 – обязательной метаданной источника кэша; A7 – запретом URL без digest и allowlist; A8 – проверкой независимого hr, владельца критического файла и повторным анализом фактического графа. Все политики пропустили шесть заранее определенных допустимых случаев, поскольку в них не было расхождений. Это не доказывает отсутствие ложных отказов на реальных

проектах: для такой оценки требуется промышленный корпус manifest и lock-файлов.

Таблица 2. Логическое покрытие модельного набора

Политика	Опасные случаи закрыты	Допустимые пропущены	Незакрытые классы
P0: multiregistry	0/24	6/6	A1–A8
P1: lock + digest	9/24	6/6	A3–A6, A8
P2: source mapping	12/24	6/6	A3, A6–A8
P3: составной алгоритм	24/24	6/6	нет в пределах набора

Обсуждение

Основной вывод состоит в различии между воспроизводимостью и происхождением. Lock-файл отвечает, какой байтовый объект ожидался, но без доверенной привязки не объясняет, откуда он был получен и кто разрешил изменение ожидания. Source mapping отвечает, откуда получать имя, но без digest не доказывает неизменность конкретной версии. Проверка обеих сторон и хеша resolver замыкает контракт.

Для npm внутренние пакеты целесообразно помещать в контролируемый scope, поскольку сама документация npm указывает этот механизм как защиту от подстановки частного пакета [6]. Для NuGet Package Source Mapping дает нативную основу, но требует охватить транзитивные зависимости и учитывать, что глобальный кэш может обойти поиск источника [7]. В Python использование extra-index-url без внешней политики оставляет неоднозначность; более предсказуемы единый proxy, предварительное wheelhouse и установка с no-index/find-links при проверенных хешах [8, 9]. Эти рекомендации приведены как варианты реализации, а не как утверждение абсолютной безопасности экосистем.

Алгоритм дополняет SBOM и provenance. SBOM, созданный после сборки, помогает инвентаризации, но не обязательно предотвращает исполнение вредоносного install script. Admission control должен сработать раньше. После успешной сборки список источников и digests включается в attestation; in-toto связывает шаги цепочки поставки [11], а Sigstore позволяет проверять подписи и записи прозрачности при наличии соответствующей инфраструктуры [12].

Для импортозамещения модель полезна тем, что не привязана к бренду хранилища. Репозиторная платформа должна защищать конфигурацию resolver и lock-файлы, запускать проверку до установки, изолировать runner и хранить доказательство. Частный реестр должен запрещать несанкционированное владение внутренним пространством имен, а proxy – сохранять исходный upstream. Отсутствие одной нативной функции можно компенсировать внешним шлюзом, но это увеличивает эксплуатационную ответственность и должно отражаться в оценке.

Ограничения: набор из 30 случаев сконструирован аналитически; не исследованы компрометация доверенного реестра, кража учетной записи владельца, вредоносное обновление ранее разрешенного пакета и коллизии хеша; значения покрытия зависят от выбранных классов. Эмпирическое продолжение должно использовать реальные проекты трех экосистем, фиксированные версии клиентов и локальные тестовые реестры. Измеряемые показатели – доля закрытых атак, ложные отказы, время проверки и расхождение ожидаемого и фактического графа.

Заключение

Предложен детерминированный алгоритм допуска зависимостей для гибридных реестров. Он объединяет нормализацию имени, владение пространством имен, проверку фактического источника, версии и digest, хеша resolver, транзитивного графа, кэша и исполняемых действий. На модельном наборе составной алгоритм логически закрыл 24 из 24 опасных случаев при сохранении шести допустимых.

Научный результат – формализация зависимости как объекта с источником и доказательством разрешения, а не только именем и версией. Практический результат – единый admission-контракт для npm, NuGet и Python, который можно выполнить до установочных скриптов и сохранить в provenance. Перед эксплуатационным выводом необходима стендовая проверка конкретных клиентов, реестров и политик кэша.

Использованные источники

1. Neupane S., Holmes G., Wyss E., Davidson D., De Carli L. Beyond Typosquatting: An In-depth Look at Package Confusion // 32nd USENIX Security Symposium. 2023. P. 4331-4348. URL: https://www.usenix.org/conference/usenixsecurity23/presentation/neupane (дата обращения: 04.09.2026).

2. Zimmermann M., Staicu C.-A., Tenny C., Pradel M. Small World with High Risks: A Study of Security Threats in the npm Ecosystem // 28th USENIX Security Symposium. 2019. P. 995-1010.

3. Ohm M., Plate H., Sykosch A., Meier M. Backstabber's Knife Collection: A Review of Open Source Software Supply Chain Attacks // DIMVA 2020. LNCS 12223. P. 23-43. DOI: 10.1007/978-3-030-52683-2_2.

4. Ladisa P., Plate H., Martinez M., Barais O. Taxonomy of Attacks on Open-Source Software Supply Chains // IEEE Security & Privacy. 2023. Vol. 21, no. 5. P. 86-94. DOI: 10.1109/MSEC.2023.3274656.

5. Chandramouli R., Kautz F., Torres-Arias S. Strategies for the Integration of Software Supply Chain Security in DevSecOps CI/CD Pipelines. NIST SP 800-204D. 2024. DOI: 10.6028/NIST.SP.800-204D.
6. npm Docs. Threats and Mitigations. URL: https://docs.npmjs.com/threats-and-mitigations/ (дата обращения: 04.09.2026).
7. Microsoft. Package Source Mapping // NuGet documentation. URL: https://learn.microsoft.com/en-us/nuget/consume-packages/package-source-mapping (дата обращения: 04.09.2026).
8. Python Packaging Authority. Package index mirrors and caches. URL: https://packaging.python.org/en/latest/guides/index-mirrors-and-caches/ (дата обращения: 04.09.2026).
9. Python Packaging Authority. Supporting downstream packaging: Do not use the Internet during the build process. URL: https://packaging.python.org/en/latest/discussions/downstream-packaging/ (дата обращения: 04.09.2026).
10. Souppaya M., Scarfone K., Dodson D. Secure Software Development Framework (SSDF) Version 1.1. NIST SP 800-218. 2022. DOI: 10.6028/NIST.SP.800-218.
11. Torres-Arias S., Afzali H., Kuppusamy T. K., Curtmola R., Cappos J. in-toto: Providing Farm-to-Table Guarantees for Bits and Bytes // 28th USENIX Security Symposium. 2019. P. 1393-1410.
12. Newman Z., Meyers J. S., Torres-Arias S. Sigstore: Software Signing for Everybody // Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security. 2022. P. 2353-2367. DOI: 10.1145/3548606.3560596.
13. Samuel J., Mathewson N., Cappos J., Dingledine R. Survivable Key Compromise in Software Update Systems // Proceedings of the 17th ACM Conference on Computer and Communications Security. 2010. P. 61-72. DOI: 10.1145/1866307.1866315.