脆弱性情報の上流は安定していない: 公開前検査による116日間の全数観測

篠原 俊一[1,a)]　中岡 典弘[1,b)]　神戸 康多[1,c)]

概要：脆弱性 DB が依存する外部フィードは，安定していない．データは一時的に消え，公開済みアドバイザリは遡って書き換えられる．本研究では，OSS 脆弱性スキャナ Vuls の DB 公開 CI に，候補と直前版を比較する公開前検査を実装した．検知結果と DB 内の検知条件構造を比較し，変動率が閾値を超えると公開を保留する．116 日間・943 run の全数調査では，閾値超過 418 run を 69 件の source-episode に集約し，上流由来 59 件 (平均約 2 日に 1 件)，自前の処理由来 10 件に帰属した．日常変動と構造的イベントは変動率で概ね桁が分離し，欠損 DB 2 件と異常な候補 1 件の公開を防いだ．以上から，全工程の履歴に基づく公開前検査が，上流の縦断観測と DB 公開の安全網を両立できることを示す．

キーワード：脆弱性データベース，データ品質，公開前検査，履歴管理，ソフトウェアサプライチェーン

Vulnerability Feeds Are Not Stable: A 116-Day Exhaustive Observation via Pre-Publication Inspection

Shinohara, Shunichi[1,a)]　Nakaoka, Norihiro[1,b)]　Kanbe, Kota[1,c)]

Abstract: External vulnerability feeds are not stable: data temporarily disappears, and published advisories are retroactively rewritten. We implemented pre-publication inspection in the database CI of Vuls, an open-source vulnerability scanner. It compares each candidate with the previous release through detection results and detection-criteria structures, withholding publication above a per-target change-rate threshold. Across all 943 runs over 116 days, 418 threshold-exceeding runs formed 69 source episodes: 59 upstream (roughly one every two days) and 10 internal. Everyday changes and structural events generally differed by an order of magnitude. The inspection prevented two incomplete databases and one anomalous candidate from being released. Backed by end-to-end history, it can serve both as a longitudinal observer and as a publication safety net.

Keywords: Vulnerability Database, Data Quality, Pre-Publication Inspection, History Management, Software Supply Chain

1. はじめに

脆弱性スキャナの検知結果は，ディストリビュータやベンダが公開するアドバイザリと脆弱性フィードに依存する．これらを集約した脆弱性データベース (以下，脆弱性 DB) から情報が欠ければ，スキャナの実装を 1 行も変えていなくても検知漏れが生じる．それにもかかわらず，DB の自動更新では，上流フィードは概ね安定しており，その変化は新規脆弱性の追加を中心とする，という仮定が暗黙に置かれやすい．この仮定はどの程度正しいのだろうか．

実際の上流フィードは，この仮定ほど単純ではない．月次データは丸ごと消え，過去のアドバイザリは予告なく再編される．本稿では，消失，縮退，無告知の再編と平常帯外の大変動を「不安定」と呼ぶ．変更が上流に由来するの

[1] フューチャー株式会社
Future Corporation
[a)] s.shinohara.yx@future.co.jp
[b)] n.nakaoka.46@future.co.jp
[c)] k.kanbe.r4@future.co.jp

か，収集・変換・DB 構築といった自前の処理に由来するのかも，完成した DB を眺めても分からない．従来研究は脆弱性 DB の品質を主に外部から静的に測定してきた．公開パイプラインの内部で経時変化を継続観測し，公開可否の判断に結び付けた実運用の報告は限られている．

我々は，OSS 脆弱性スキャナ Vuls[1] 向けの DB(vuls.db) を，40 超のデータソースから 6 時間ごとに構築して公開している．前稿 [2] では，収集 (fetch)，変換 (extract)，DB 構築 (db build) の全工程を履歴管理し，任意時点の入力と生成物を追跡して再現する基盤を報告した．本稿はその続編である．前稿が変化を**記録できる**ことを示したのに対し，本稿はその記録で，上流が実際に**どう変化しているか**を測り，公開品質管理へ実用化する．測定の装置は単純で，公開候補 DB と直前の公開版 (baseline) を自動比較する**公開前検査** (diff guard) である．検査は，変動率が閾値を超えた候補の自動公開を保留して人間の確認を強制するが，変更の正誤は判定しない．116 日分の発動履歴は副産物として，上流がいかに大きく，頻繁に，時に黙って動くかの縦断観測データとなった (図 1)．

本稿では，次の 3 つの研究質問に答える．

(1) **RQ1: 何が公開前検査を発動させるか．** 発動をイベント単位に集約し，上流と自前の処理に原因帰属する．

(2) **RQ2: 上流フィードはどのように不安定か．** 可用性，完全性，変動規模の観点から，代表的な変更とその影響を明らかにする．

(3) **RQ3: 差分検査は公開事故を防げるか．** 阻止できた事例だけでなく，承認判断を誤った事例と運用コストも評価する．

116 日間・943 run を全数調査した結果，閾値超過 418 run は 69 件の source-episode に集約され，59 件が上流由来，10 件が自前の処理由来であった (RQ1)．上流では，月次データの消失と復活，過去アドバイザリの縮退，更新情報を伴わないマッピング変更を観測した．日常変動とこれらの構造的イベントは，変動率で概ね桁が分離した (RQ2)．検査は欠損 DB の公開を 2 回，異常な候補 DB の公開を 1 回阻止した．一方，検査が発動しても人間が不適切に承認すれば事故は残るため，ソース単位の比較，削除方向の扱い，承認手続きが重要である (RQ3)．

本稿の貢献は，(1) 本番パイプラインの全発動を対象とした上流変化の縦断データ，(2) 全工程の履歴を使って変化の発生箇所を切り分ける再現可能な原因帰属手順，(3) 閾値の継続調整と監査証跡付き手動承認を含む公開前検査の設計，の 3 点である．

2. 対象システムと公開前検査

2.1 履歴管理された DB 公開パイプライン

vuls.db の生成は，**収集 (fetch)**，**変換 (extract)**，**DB 構築 (db build)** の 3 工程からなる [2], [3], [4], [5]．収集

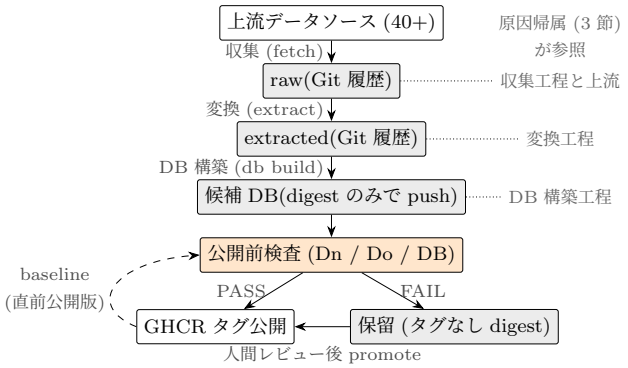

図 1 脆弱性 DB 公開パイプラインと公開前検査．全工程が履歴管理 [2] され，原因帰属 (3 節) は各工程の履歴と収集対象設定を参照する．

工程はデータソース (以下，ソース) からデータを取得し，原形をほぼ保った raw データとして Git で履歴管理する．変換工程は raw データを共通スキーマへ変換し，extracted データとして同様に履歴管理する．DB 構築工程は extracted データから BoltDB 形式の vuls.db を構築し，GitHub Container Registry(以下，GHCR) へ公開する．DB 内で OS の系統と版や CPE など検知対象を分ける区分を **ecosystem** と呼び，ソースとは多対多に対応する．本稿では，各工程の実装をそれぞれ**収集コード**，**変換コード**，**DB 構築コード**と呼ぶ．また，収集起点となる ID や URL と対象ソースを定める vuls-data-db 側の設定を**収集対象設定**と呼び，収集コードと区別する．

各 DB には入力データと DB 構築コードのバージョンが記録され，GHCR 上の生成物は digest で固定できる．したがって，任意時点の raw データまでコミット単位で遡り，同じ DB による検知結果を再現できる．公開前検査は，この追跡可能性を公開可否の判断に利用する (図 1)．

2.2 動機となった 2 つのインシデント

2026 年 3 月，DB 構築は成功したものの，検知結果が大きく誤る DB を 2 回公開した．

インシデント 1(Red Hat VEX，false negative): データ形式を変更した日に，上流から不正な VEX(Vulnerability Exploitability eXchange) データが配信された．変換処理はエラーとなり，データは旧形式のまま残ったが，後段のフィルタは新形式を前提としていたため空データを生成した．その結果，redhat:9 で本来検知される CVE の 51.9%(3,387 件) を「影響なし」とする DB が公開された．上流の異常と，工程間の形式整合を保証しない自前の実装が複合した事例である．

インシデント 2(Ubuntu CVE Tracker，false positive): DB に導入したマーカを旧バージョンのスキャナが解釈できず，ubuntu_2204 の検知件数が 5,968 件から 12,296 件へ増えた (+106.0%)．DB とスキャナはそれぞれ単体では正常であり，両者の組み合わせに互換性がな

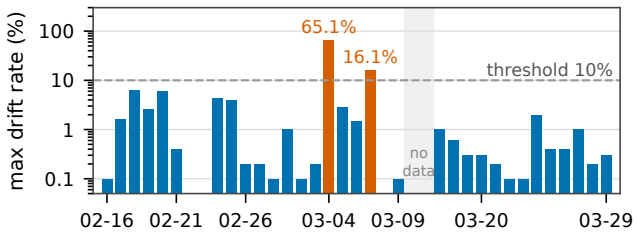

図 2 導入前ベンチマーク: 連続 1 日ペア 35 組の最大変動率 (対数軸，破線は DB 既定閾値 10%)．橙は実在の上流イベント (Red Hat VEX 65.1%, SUSE OVAL 16.1%) で，正常帯 (中央値 0.3%) と桁で分離する．

かった．

従来の CI が確かめるのはビルドの成否であって，生成物がデータとして正しいかではない．また，インシデント 1 は DB の構造から，インシデント 2 は DB と旧スキャナを組み合わせた検知結果から初めて分かる．そこで，DB 構造と実際の検知結果を相補的に比較する検査を設計した．

2.3 閾値設計の事前検証

脆弱性 DB に差分ゼロの日はない．差分の有無ではなく大きさで判定するには，日常変動と確認すべきイベントが変動率で分離している必要がある．

導入前に，GHCR に残る 37 日分の DB(2026-02-15 から 03-29，03-10〜15 は検証環境のディスク不足で欠測) から連続 1 日ペア 35 組を作り，DB 構造の変動率を測定した．日常変動は中央値 0.3%，最大 6.2%であった．10%を超えたのは，Red Hat VEX の構造変更 (redhat:6 で 65.1%) と SUSE OVAL の大規模再発行 (16.1%) という，実在する上流イベント 2 組であった (図 2)．前者はインシデント 1 と同一であり，この検査があれば公開前に捕捉できた．この桁の分離を根拠に DB 構造差分の既定閾値を 10%とした．一方，インシデント 2 は DB と旧スキャナの組み合わせで初めて現れるため，DB 構造の比較では捕捉できない．検知結果の比較 (2.4 節) を併置するのはこのためである．任意の DB ペアを比較できること自体，公開物が digest 付きで履歴化されている [2] ことの恩恵である．

2.4 3 つのチェック

公開 CI では，候補 DB の構築後，タグを付けて公開する前に次の 3 チェックを実行する．検知結果差分を Dn と Do，DB 構造差分を DB と略記する．

- **Dn(最新版による検知結果差分)**: baseline と候補 DB のそれぞれに対し，最新版のスキャナを同じ fixture 群で実行し，fixture ごとの検知 CVE 集合を比較する．fixture は実機スキャンに由来する 75 ファイルで，Windows を含む 12 OS ファミリと CPE 系 6 ソースを対象とする．
- **Do(旧版による検知結果差分)**: 観測期間中に継続利用されていた旧版を commit で固定し，同じ比較を行う．この旧版は旧版全体を代表するものではなく，運用上の互換性基準点である．DB と旧スキャナの組み合わせでのみ生じる非互換を検出する．
- **DB(DB 構造差分)**: BoltDB を直接読み，ecosystem，後にはソースごとに，検知条件木を leaf Criterion まで平坦化して比較する．バイト列ではなく Criterion を比較することで，JSON のキー順など意味を変えない差分を除外する．全体の分母は約 330 万 Criterion であり，日常的な小変更は小さな drift として現れる．一方，fixture に含まれない対象はこのチェックだけが観測する．

変動率は，追加数と削除数の和を baseline の要素数で割って求める．対象ごとに 1 行の結果を出力し，1 行でも閾値を超えれば run 全体を FAIL とする．baseline に存在しない対象の変動率は 100%とする．CI の Job Summary には，変動率，増減数，変化した CVE またはアドバイザリ ID を常に出力する．先に失敗したチェックが後続の情報を隠さないよう，3 チェックをすべて実行した後に結果を集約する．これは，前稿が課題として挙げた行ベースではない論理的な差分提示 [2] を，公開品質管理として実装したものである．

2.5 FAIL 時の運用: 監査証跡付き手動 promote

候補 DB は，検査前にタグを付けず digest だけで GHCR へ push する．検査が PASS した場合に限り公開タグを付ける．FAIL した候補も digest で取得できるため，公開せずに事後検証できる．

正当な変更と判断した場合，オペレータは専用 workflow で候補 digest にタグを付ける．この操作を **promote** と呼ぶ．実行者，digest，タグは workflow の履歴に，判断材料となったレポートは FAIL した run に残り，検査を手動で越える操作にも監査証跡がある．promote の判断と実行は人間に限定し，AI エージェントには許可していない．

2.6 閾値の per-target / per-source 化と grooming

既定閾値は Dn と Do が 5%，DB が 10%である．DB の値は事前検証に基づき，検知結果の値は経験的な初期値として定めたが，一般的な最適値を意味しない．新しいディストリビューションの一括 triage，rolling release，ベンダの月次更新など，正当で反復的な大変動には対象別の override を用意し，失敗履歴の「観測ピーク+ヘッドルーム」から再較正する．単発のスパイクは緩和せず，人間の確認対象として残す．

override が現状と乖離しないよう，約 2 か月ごとに全対象の値を再導出する．本稿ではこの作業を **grooming** と呼ぶ．また，運用後期には，ecosystem 全体を分母にすると大きなソースが小さなソースの異常を希釈することが分

かったため，レポートと閾値をソース単位に細分化した．なお，現在の変動率は追加と削除を対称に扱う．overrideを緩和すると，検知漏れに直結する削除への感度も下がるため，追加と削除を分けた閾値が必要である．

3. 評価方法

評価では，RQ1 に対して全 run をイベントへ集約し，工程履歴から原因を帰属する．RQ2 では，上流由来のイベントを可用性，完全性，変動規模の観点で分類する．RQ3 では，公開を阻止した事例，阻止できなかった事例，および運用コストを調べる．

3.1 観測期間と分析データ

観測期間は，検査を本番導入した 2026-04-23 から 08-16 までの 116 日間である．安定版と開発版の 2 系統で実行された全 943 run について，run ログ，関連 PR，promote 履歴を GitHub API から収集した．したがって，標本抽出ではなく期間内の全数調査である．

各 run から，実行系統，時刻，成否，失敗工程，チェック種別，対象，変動率，増減数を抽出した．後述の原因帰属には，DB 構築コード，収集コードと変換コード，収集対象設定のコミット履歴，raw および extracted データの Git 差分を用いた．

3.2 run から source-episode への集約

FAIL 中は公開版である baseline が更新されず，同じ差分が 6 時間ごとに繰り返し検出されるため，run 数は実イベント数より多くなる．この重複を除くため，まず同一系統の連続 FAIL を **fail sequence** として 104 件にまとめた．次に，安定版と開発版を統合し，一つの FAIL をソース (前半は推定区分) ごとに分解して，**source-episode** 69 件を得た．以後，原因分布を数える単位は source-episode とする．

同一ソースの発動に 24 時間以上の途切れがあれば，別の episode として扱う．この規則は時間的な集約であり，上流イベントの独立性を保証しない．分割間隔を 12 時間にすると 77 件，48 時間にすると 62 件となる．ソース名がない前半は target 名からソース区分を推定し，後半のレポートで対応を検証した．

3.3 工程履歴に基づく原因帰属

各 source-episode について，生成物から入力側へ遡り，次の順で原因候補を除外する．

(1) **DB 構築工程**: baseline と候補 DB のメタデータを比較する．DB 構築コードのバージョンが同じなら，同工程の変更を除外する．

(2) **収集工程と変換工程**: 対象期間のコード履歴を調べる．当該ソースの収集コードと変換コードに変更がなければ，両工程の変更を除外する．

(3) **収集対象設定**: DB 公開リポジトリの履歴を調べる．収集起点の ID や URL と対象ソースに変更がなければ，入力範囲の変更を除外する．

(4) **上流**: raw データのコミットを特定し，検査レポートと件数または ID が一致する変更実体を探す．例えば，アドバイザリ群の追加やディレクトリ全体の消失が直接の証拠となる．

これらの履歴がなければ，上流の一時障害として破棄すべき候補と，正当な変更として promote できる候補を根拠付きで区別できない．履歴管理は本測定の成立条件であり，本結果は前稿の基盤を実運用の原因帰属に利用した実証でもある [2]．全 69 episode を上流由来か自前の処理由来かへ帰属でき，「原因不明」は残らなかった．自前の処理由来 10 件は，いずれも原因となった変更まで確認した．上流由来 59 件では，自前の 3 工程と収集対象設定に関係する変更がないことを履歴から確認し，代表事例について raw 差分の直接証拠まで確認した．帰属手順は triage 手順書と AI エージェント用スキルにも記述した (役割分担は 5.4 節)．

3.4 本測定の限界

第一に，本稿が全数調査するのは run であり上流の全変更ではない．episode は閾値を超えた変更だけを含む打ち切り観測であるため，発動件数だけでなく原因の内訳，変動率の分布，日常変動も含む事前検証 (2.3 節) を併せて示す．第二に，episode の件数は 24 時間という集約規則に依存するため，12 時間と 48 時間の場合も感度分析として示した．第三に，観測期間中も検査を改良した (3 チェックの全実行化，fixture と override の追加，ソース単位化)．結果は固定した測定器による同質な標本ではなく，運用中に改善された検査の発動履歴である．第四に，Vuls 単一系のケーススタディであり，閾値と発動頻度は他の脆弱性 DB へ一般化できない．

3.5 再現用資料の公開

検査の workflow，閾値，override 設定は vuls-data-db[5] で，DB 構築と差分検査本体の実装は vuls2[4] で公開している．収集した run 表，fail sequence，source-episode ごとの判定表，集計コードは，本研究の公開リポジトリ[*1]に収録した．

4. 結果

4.1 RQ1: 発動規模と原因分布

全 943 run のうち 476 run が失敗した．内訳は，公開前検査 420 run(閾値超過 418，検査内のインフラ障害 2)，

[*1] https://github.com/vulsio/css2026-diff-guard

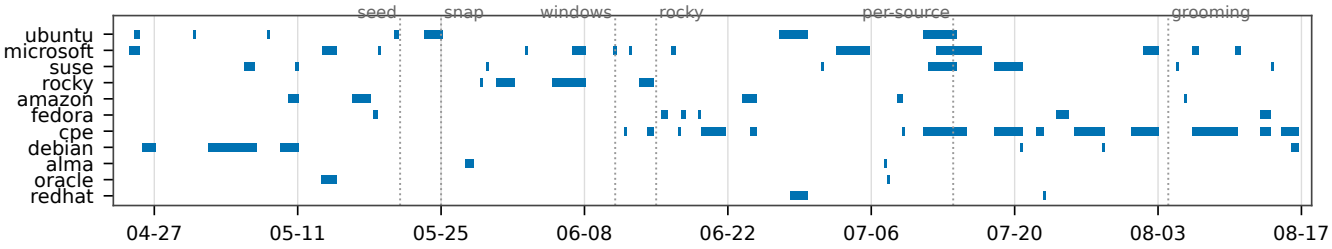

図 3 116 日間の source-episode 69 件．cpe 行は複数の CPE 系ソースを統合する．横棒は最初の FAIL から終端まで，点線は override，ソース単位化，grooming の導入時点を示す．

表 1 source-episode 69 件の原因分布．複数原因は公開判断を左右した原因で代表させた．「捕捉」は発動したチェック．

判定	件数	代表例	捕捉
上流: 正当な変更	55	月次パッチ等	混在
上流: 一時的障害	2	CVRF 消失	Dn
上流: 恒久的品質イベント	2	errata 再編	全て
変換工程由来 (コードバグ)	1	非決定的出力	DB
変換工程由来 (意図的変更)	7	CPE 系列等	DB
収集対象設定由来	2	収集起点 ID 追加	DB

DB 構築工程 39 run，runner 障害など 17 run である．閾値超過 run を集約すると 69 source-episode となった (図 3)．チェック別では，DB 単独が 23 件，Dn 単独が 16 件，複数チェックの同時発動が 30 件であり，Do 単独は 0 件であった (5.2 節)．

原因帰属の結果を表 1 に示す．上流由来は 59 件 (85.5%)，自前の処理由来は 10 件 (14.5%) であった．上流由来の大半は正当な変更だが，一時的な全消失 2 件と恒久的な品質イベント 2 件も含む．自前の処理由来は，変換コードのバグ 1 件，意図した変換コード変更 7 件，収集対象設定の変更 2 件であった．収集工程と DB 構築工程に帰属した episode はなかった．

自前の変更も十分に大きな発動を生じた．例えば，収集起点となる Windows 更新プログラム ID を追加した直後には microsoft/microsoft-msuc が 211.3%，新規データソースを追加した直後には microsoft/microsoft-servicing が 100.0%で発動した．いずれも上流や変換コードではなく，「何を収集するか」を定める収集対象設定の変更である．また，2012 年から 2022 年のアドバイザリへ検知を補完する変換コード変更では，検知を持つファイルが 643 件から 1,065 件に増え，fortinet-cvrf が 135.5%で発動した．検査は異常だけでなく，意図した大規模変更の影響範囲も可視化した．この 2 件を契機に，収集対象設定のコミット走査を原因帰属手順へ追加した．

4.2 RQ2-a: 可用性: 消え，戻り，また消える

Microsoft CVRF の月次データは消え，戻り，また消えた．2026 年 6 月分の全消失が 6 月と 7 月の 2 度起き，2 回目は 2 日間に消失と復活を 2 回ずつ繰り返した．API の応答が構文的に正しくても，収集した時点が欠損状態なら，パイプラインは正常終了したまま不完全な DB を作る．すなわち，収集処理の成功とデータの可用性は同義ではない．

対策は二段階ある．収集時に件数や既知 ID の消失を確認すれば不完全な入力の履歴化を早期に止められ，公開前に直前版と比較すれば想定外の欠損も利用者へ届く前に検出できる．

4.3 RQ2-b: 完全性: 遡及的に書き換わる

上流の変化は一時的な障害だけではなかった．過去データを意図的または暗黙に書き換える，次の 3 種類を観測した．表 1 では，AlmaLinux の 2 件を恒久的品質イベント，Cisco と VulnCheck を正当な上流変更とした．

- **errata の検知条件の縮退**: AlmaLinux では，errata.full.json から一部の errata が消失しただけでなく，残った errata でも対象パッケージが減少し，その内容は OSV(Open Source Vulnerabilities) データとアドバイザリ HTML でも一致した．上流からは，errata.full.json と updateinfo.xml は現行版リポジトリのパッケージだけを保持するとの回答を得た[*2]．上流が仕様どおりでも，過去バージョンを検査する下流にとっては情報の欠損になる．
- **サイレントなサーバ側再編**: Cisco openVuln API では，アドバイザリの更新日時とバージョンを変えずに，影響バージョンの対応だけが日次で書き換わった．93 アドバイザリで製品バージョン 2,022 件が削除，1,481 件が追加され (純減 541 件)，2014 年のあるアドバイザリでは製品リストが 149 件から 36 件になった．過剰な対応付けの修正とも解釈できるが，下流から見れば，告知も更新日時の変更もないまま生じる検知消失である．
- **一斉再生成の両義性**: VulnCheck による全年代 CVE への生成 CPE 一括付与は，情報量を増やす一方，CPE 語彙の変更による検知退行も含んだ．2 週間後には上流の修正による回復が再び大変動として現れた．変動

[*2] https://bugs.almalinux.org/view.php?id=644

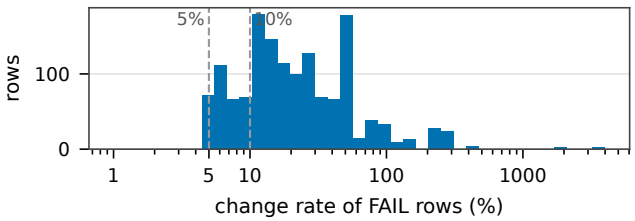

図 4 FAIL した 1,465 行の変動率分布 (対数軸)．5〜20%の帯に対し，100%を超える構造的イベントが裾を形成する．cron による重複を含み，閾値未満の更新を含まない．

率だけでは，拡充，退行，回復の意味を判定できない．

以上から，上流の変更を単純に正常か異常かへ二分することは難しい．公開前検査の役割は変更の意味を自動決定することではなく，平常帯から外れた変更を，履歴とともに人間へ提示することである．

4.4 RQ2-c: 変動規模: 正当でも桁違いに動く

変動率の頂点は新規リリース fedora:46 の 3433.3%で，AlmaLinux errata 再編の 428.3%，変換コードの非決定性バグの 274.5%(4.5 節) が続く．一方，閾値超過行だけを対象とし，baseline 停滞による累積を含む本番分布でも，日常変動の多くは 5〜20%にあり，構造的イベントとは概ね桁が分離した (図 4)．この結果は，単純な閾値を一次選別に利用できることを支持する．

ただし，変動率は比較条件に依存する．第一に，baseline が小さい新規対象は，少数の正当な追加でも 100%を超える．第二に，FAIL 中は baseline が更新されないため，比較期間が伸び，複数回の日常変動が累積する．第三に，大きなソースと小さなソースを同じ分母で集計すると，小さなソースの異常が希釈される．対象別閾値，迅速な承認，ソース単位の比較は，それぞれの影響を緩和するために必要である．

4.5 RQ3-a: 公開を阻止した 3 件

検査は，欠損 DB の公開を 2 回，異常な候補 DB の公開を 1 回阻止した．前者は上流データの一時的な全消失，後者は自前のコードに起因する異常である．

(1) **Microsoft CVRF 月次データの全消失 (2 件)**: 上流 API が 2026 年 6 月分のセキュリティ更新文書 700〜1,300 ファイルを返さなくなる事象が 6 月と 7 月に起き，Windows 系 13 対象で最大 53.6%の削除が生じた．2 件目では約 1,270 CVE の検知消失に相当する．候補を破棄し，上流回復後に再構築することで，false negative を含む DB の公開を防いだ．raw 履歴では，当月ファイル数が 1273 → 0 → 1273 → 0 → 1278 と変化しており，収集処理の単発失敗ではなく，上流配信のフラッピングであった．

(2) **変換コードの非決定性バグ (1 件)**: DB チェックが 274.5%の drift を検出した．extracted データの変更は 23,163 ファイルに及ぶ一方，入力の raw データは 266 ファイルにすぎず，この不均衡が変換コードのバグの発見に至った．原因は型の取り違えでソート処理が実行されず，出力順が非決定になったことである．候補を公開せず，変換コードを修正した．

利用者報告と事後分析の範囲では，検査が発動しないまま公開された破損は確認していない．ただし，未検知の破損がないことを網羅的に確認する手段はない．

4.6 RQ3-b: 防ぎ切れなかった 2 件

一方，検査が発動しても実害を防ぎ切れず，検査と運用の改良に帰結した事例が 2 件ある．検査は差分を検出して公開を保留するが，最終的な promote は人間が判断する．

(1) **AlmaLinux errata の縮退**: 1 週間に複数回の再編があり，アドバイザリ数は 279 件から一時 60 件へ減少した．削除されたアドバイザリが参照していた 434 CVE のうち 376 件 (87%) はフィード全体から消え，残った kernel アドバイザリも検知対象が 73 パッケージから 2 パッケージへ縮退した．検査は縮退時に 99.2%，巻き戻し時には最大 428.3%で発動した．しかし，滞留した FAIL を解消する際にオペレータが縮退候補を promote し，数時間だけ公開した．同日中に縮退前のコミットへ入力を固定して巻き戻し，後に履歴から消失分を復元する収集処理を実装した (4.3 節)．

(2) **VulnCheck 一斉ロールアウト**: 生成 CPE(Common Platform Enumeration) を一斉付与する単一コミットにより，145,672 ファイル，957 万行が追加された．当時の DB チェックは ecosystem 全体で cpe 28.9%と報告し，候補は promote された．事後にソース単位で再計算すると，当該ソースは 189.6%変化し，実検知が 60〜75%消失する退行を含んでいた．この事例を受け，レポートと閾値をソース単位へ変更した (2.6 節)．

2 件とも，差分の検出と公開保留までは成功した．残った問題は，レポートの分解能と人間の承認判断である．削除方向の自動ブロック，大規模消失時の複数名承認，承認時に確認すべき項目の明文化が必要である．

5. 考察

5.1 正当な変更を止める意味

発動 69 件のうち 55 件は正当な上流変更であった．発動は「異常」の宣告ではなく，「公開前に人間が一目見るべき規模」の合図であり，一律に誤報とは扱わない．例えば，822 CVE を含むカーネル更新や 5,312 検知条件の一括更新は，正当でも公開前に確認する価値がある．ただし各変更の確認価値は定量化しておらず，55 件すべてを有用な警告と主張するわけではない．示せたのは，正当な変更にも公開前確認に値する大変動が含まれ，反復性を確認したもの

(月次更新等) は override に反映して確認を未知の大変動へ集中できる，という運用上の分離である．

5.2 チェックの相補性と比較粒度

CVRF 全消失では，DB チェックが Windows 更新プログラムのキー集合だけを比較していたため，キーに紐付く CVE が消えても 0%となった．一方，Dn は実検知の減少を捉えて公開を止めた．逆に，fixture にない対象や，検知集合を変えない内部構造の大変化は DB だけが捉える．異なる表現を比較することで，一方の盲点をもう一方が補った．

Do だけが発動した episode は 0 件であり，Do だけが捉える互換性退行は観測されなかった．一方，スキャナのバイナリリリースの間にも変換コードの変更は DB へ直ちに反映されるため，DB と既存スキャナの更新周期は一致しない．Do はこの時間差に対し，継続利用される一つの旧版との互換性を公開境界で確認する，インシデント 2 型の非互換 (2.2 節) に対する回帰テストである．単独発動 0 件は本観測期間に当該退行が生じなかったことを示すだけで，チェックが不要であることは示さない．

比較粒度も結果を左右した．ソース単位化後，従来は cpe 全体だった変動を，vulncheck-nist-nvd2 の 110 行から nvd-feed-cve-v2 の 2 行まで 7 ソースへ分解できた．後半期間では，VulnCheck 事例と同型の希釈は観測していない．また，安定版と開発版で override が非対称だと，一方だけが慢性的に FAIL した．現在は grooming 時に両系統を同時に再評価している．

5.3 運用コストの実態

検査は 943 run 中 420 run(44.5%) を止めた．手動 promote は 116 日間で 92 unique digest，1 日当たり 0.79 件であった．ただし 92 件には同一イベントへの安定版と開発版それぞれの promote や再発分が含まれ，発動イベント数や誤報数とは一致しない．promote する場合の確認は 1 件あたり数分だが，promote できない場合は原因追及に時間を要する．

fail sequence の持続時間は中央値 11 時間，90 パーセンタイル 48 時間であった．run 単位の FAIL の 50%が土日に記録されたのは，上流イベントが週末に偏るからではなく，平日日中まで promote されず，同じ差分が繰り返し検出されるためである．したがって，持続時間は主にオペレータの応答時間を表す．また，保留中は直前版を配信し続けるため，慎重な保留は新規 CVE の反映遅延という別のリスクを生む．通知の改善，承認期限，削除量に応じた承認者数など，検出後の運用設計が必要である．

計算時間は DB チェック約 1 分，Dn と Do を含む検知チェック約 9 分で，6 時間周期の CI に常設できた．主要な制約は計算ではなく，検出後に人間の判断を要することであった．

5.4 出力検査は開発の最終ガードレールになる

工程単位のテストやコードレビューだけでは，集約後のデータが意味的に正しいことを保証できない．実際，型の取り違えによるソート漏れはコンパイルと既存テストを通過し，出力の大規模な構造差分として初めて見つかった．公開前検査は，実装方法ではなく生成物の振る舞いを検査する最後の境界として機能した．

開発版へ変更を先行投入して影響規模を測る，想定した対象だけが FAIL することを確認して混入を検出する，といった用途にも利用できる．本運用では一次調査の一部を AI エージェントに委ねる一方，根拠の検証と promote は人間に限定した (2.5 節)．生成も調査も速く，検証は出力で，承認は人間で，という役割分担である．

6. 関連研究

6.1 脆弱性 DB とフィードの品質測定

NVD をはじめとする脆弱性 DB については，影響バージョンの不整合 [6], 品質監査と自動修正 [7], 国際的な DB 間差 [8] が調査されている．また，スキャナや SCA(Software Composition Analysis) ツール間の検知差が，利用する DB の差に由来することも報告されている [9], [10]．NVD enrichment 停滞の継続観測 [11], [12] は，上流の運用状態が下流へ波及することを示す．

脅威インテリジェンスフィードでは，量，カバレッジ，遅延，正確性などをフィード間で比較し，重複の少なさや品質の見えにくさが示されている [13], [14], [15]．いずれも外部から複数 DB やフィードを比較する横断測定であり，本研究は同じ公開物の連続版を本番パイプライン内で比較して経時変化を公開判断へ結び付ける．

国内では，JVN の提案 [16] 以降，OSS の脆弱性修正状況 [17] や PSIRT 向けスクリーニング [18] が研究されてきた．JVN iPedia の登録状況レポート [19] は，2024 年の NVD 公開遅延によって国内 DB の登録数が減少したことを報告している．フィード内容の経時的不安定性を下流の公開パイプラインで全数観測した国内研究は，我々の調査範囲では見当たらない．

6.2 データパイプラインの品質検査

データ品質検査には，宣言的制約を用いる方法 [20], [21] と，過去の実行履歴から制約を導出する方法 [22], [23] がある．本研究に最も近いのは，新しいデータバッチが履歴から逸脱した場合に ML モデルの再学習を止めるゲートであり [24]，履歴との差分で下流での利用を保留する点が共通する．

本研究は列統計ではなく脆弱性の検知集合と検知条件を比較し，監査証跡付きの手動 promote を公開フローに組み込み，発動履歴を上流の縦断観測に利用する．直前公開版をオラクルとする differential testing[25] の時間軸方向へ

の適用でもある．

7. おわりに

上流は消え，戻り，遡及的に書き換わり，アナウンスなく再編される．しかも，その大半は正当な変更である．

RQ1 では，閾値超過 418 run を 69 件に集約し，全工程の履歴により，上流由来 59 件 (平均約 2 日に 1 件) と自前由来 10 件に帰属した．RQ2 では，月次データの全消失，過去アドバイザリの縮退，更新情報のない再編を観測した．上流が仕様どおりでも下流では検知消失になり得る．RQ3 では，欠損 DB 2 件と異常な候補 1 件の公開を防いだが，不適切な promote もあった．検査が保証するのは自動保留までであり，最終判断は人間に残る．

運用には，ソース単位の差分，履歴による閾値再導出，追加と削除の分離，監査証跡と判断基準を備えた手動承認が要る．上流の恒久的な劣化には，正常なコミットへのピン留めと収集履歴からの復元という出口が要る．全工程の履歴に基づく公開前検査は，上流の縦断観測と DB 公開の安全網を同じ仕組みで実現する．前稿が課題とした論理的な差分提示とパイプラインの堅牢化 [2] への，実運用を伴う回答でもある．

今後は，収集時の欠損検出，追加と削除の非対称閾値，baseline 停滞の補正，通知と承認の改善，観測期間の延長と他の脆弱性 DB 公開基盤との比較に取り組む．

参考文献

[1] Vuls: Vulnerability scanner. https://github.com/future-architect/vuls.

[2] 中岡典弘, 篠原俊一. 広範な脆弱性情報の統合管理と履歴追跡. コンピュータセキュリティシンポジウム 2025 (CSS 2025), 2025.

[3] vuls-data-update. https://github.com/MaineK00n/vuls-data-update.

[4] vuls2. https://github.com/MaineK00n/vuls2.

[5] vuls-data-db. https://github.com/vulsio/vuls-data-db.

[6] Ying Dong, Wenbo Guo, Yueqi Chen, Xinyu Xing, Yuqing Zhang, and Gang Wang. Towards the detection of inconsistencies in public security vulnerability reports. In *28th USENIX Security Symposium*, pp. 869–885, 2019.

[7] Afsah Anwar, Ahmed Abusnaina, Songqing Chen, Frank Li, and David Mohaisen. Cleaning the NVD: Comprehensive quality assessment, improvements, and analyses. *IEEE Transactions on Dependable and Secure Computing*, Vol. 19, No. 6, pp. 4255–4269, 2022.

[8] Juehao Lin, Xuanxiang William Wang, Manuel Egele, and Gianluca Stringhini. A comparative analysis of nvd, jvndb and cnvd: Insights into global and regional vulnerability reporting. In *Proceedings of the ACM Asia Conference on Computer and Communications Security (ASIA CCS)*, pp. 1323–1338, 2026.

[9] Nasif Imtiaz, Seaver Thorn, and Laurie Williams. A comparative study of vulnerability reporting by software composition analysis tools. In *ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM)*, 2021.

[10] Yekatierina Churakova, Mathias Ekstedt, and Larissa Schmid. Vexed by VEX tools: Consistency evaluation of container vulnerability scanners. arXiv:2503.14388, 2025.

[11] VulnCheck. The real danger lurking in the NVD backlog. https://www.vulncheck.com/blog/nvd-backlog-exploitation, 2024. (Accessed 2026-07-29).

[12] Anchore. The NVD enrichment crisis: One year later. https://anchore.com/blog/nvd-crisis-one-year-later/, 2025. (Accessed 2026-07-29).

[13] Vector Guo Li, Matthew Dunn, Paul Pearce, Damon McCoy, Geoffrey M. Voelker, Stefan Savage, and Kirill Levchenko. Reading the tea leaves: A comparative analysis of threat intelligence. In *28th USENIX Security Symposium*, pp. 851–867, 2019.

[14] Xander Bouwman, Harm Griffioen, Jelle Egbers, Christian Doerr, Bram Klievink, and Michel van Eeten. A different cup of TI? the added value of commercial threat intelligence. In *29th USENIX Security Symposium*, 2020.

[15] Harm Griffioen, Tim Booij, and Christian Doerr. Quality evaluation of cyber threat intelligence feeds. In *Applied Cryptography and Network Security (ACNS), LNCS 12147*, pp. 277–296, 2020.

[16] 寺田真敏, 高田眞吾, 土居範久. 脆弱性対策情報データベース JVN の提案. 情報処理学会論文誌, Vol. 46, No. 5, pp. 1256–1265, 2005.

[17] 葛野弘樹, 矢野智彦, 面和毅, 山内利宏. オープンソースソフトウェアを対象とした脆弱性修正状況の収集と調査. コンピュータセキュリティシンポジウム 2024 (CSS 2024), 2024.

[18] 金井遵, 上原龍也, 小池竜一, 鬼頭利之, 神戸康多, 木戸俊輔, 篠原俊一, 中岡典弘, 林優二郎. PSIRT 向け脆弱性スクリーニング技術と環境毎リスク評価技術. コンピュータセキュリティシンポジウム 2024 (CSS 2024), 2024.

[19] 情報処理推進機構. 脆弱性対策情報データベース JVN iPedia の登録状況. https://www.ipa.go.jp/security/reports/vuln/jvn/. 四半期レポート.

[20] Sebastian Schelter, Dustin Lange, Philipp Schmidt, Meltem Celikel, Felix Biessmann, and Andreas Grafberger. Automating large-scale data quality verification. *Proceedings of the VLDB Endowment*, Vol. 11, No. 12, pp. 1781–1794, 2018.

[21] Eric Breck, Marty Zinkevich, Neoklis Polyzotis, Steven Euijong Whang, and Sudip Roy. Data validation for machine learning. In *Proceedings of Machine Learning and Systems (MLSys)*, 2019.

[22] Dezhan Tu, Yeye He, Weiwei Cui, Song Ge, Haidong Zhang, Shi Han, Dongmei Zhang, and Surajit Chaudhuri. Auto-validate by-history: Auto-program data quality constraints to validate recurring data pipelines. In *29th ACM SIGKDD Conference on Knowledge Discovery and Data Mining (KDD)*, 2023.

[23] Sergey Redyuk, Zoi Kaoudi, Volker Markl, and Sebastian Schelter. Automating data quality validation for dynamic data ingestion. In *24th International Conference on Extending Database Technology (EDBT)*, pp. 61–72, 2021.

[24] Shreya Shankar, Labib Fawaz, Karl Gyllstrom, and Aditya G. Parameswaran. Moving fast with broken data. arXiv:2303.06094, 2023.

[25] William M. McKeeman. Differential testing for software. *Digital Technical Journal*, Vol. 10, No. 1, pp. 100–107, 1998.