

Propagation-Based Vulnerability Impact Assessment for Software Supply Chains

Bonan Ruan, Zhiwei Lin, Jiahao Liu, Chuqi Zhang, Kaihang Ji, Zhenkai Liang
School of Computing, National University of Singapore

40th IEEE/ACM International Conference on Automated Software Engineering (ASE)
Seoul, South Korea



Single Vulnerability Assessment Is Not Enough



To assess a disclosed vulnerability (CVE), we have:

- ❑ Common Vulnerability Scoring System (CVSS)
- ❑ Common Weakness Enumeration (CWE)
- ❑ Exploit Prediction Scoring System (EPSS)
- ❑ Other assessment metrics...

Sources:

- <https://www.cve.org/>
- <https://www.first.org/cvss>
- <https://cwe.mitre.org>
- <https://www.first.org/epss>

Listing 1. CVSS v3.x Assessment for CVE-2021-44228	
Base Score:	10.0 (Critical)
Vector String:	CVSS:3.1/AV:N/AC:L/PR:N/UI:N/S:C/C:H/I:H/A:H

EPSS (Probabilities)

CVE-2024-50603 93.6%	CVE-2024-13161 89.5%	CVE-2025-20029 75.6%
CVE-2024-57727 93.5%	CVE-2024-48248 87.8%	CVE-2025-23006 69.3%
CVE-2025-0108 93.2%	CVE-2025-24813 86.7%	CVE-2024-12986 68.1%
CVE-2024-13159 92.5%	CVE-2024-12849 83.6%	CVE-2025-30066 67.0%
CVE-2024-13160 91.9%	CVE-2024-12856 82.7%	CVE-2022-3365 58.5%
CVE-2024-53704 91.1%	CVE-2025-1094 81.9%	CVE-2024-46310 57.6%
CVE-2024-55591 90.3%	CVE-2025-24893 77.9%	CVE-2024-11613 56.8%
CVE-2025-0282 89.7%	CVE-2024-3393 75.6%	CVE-2025-0655 54.9%

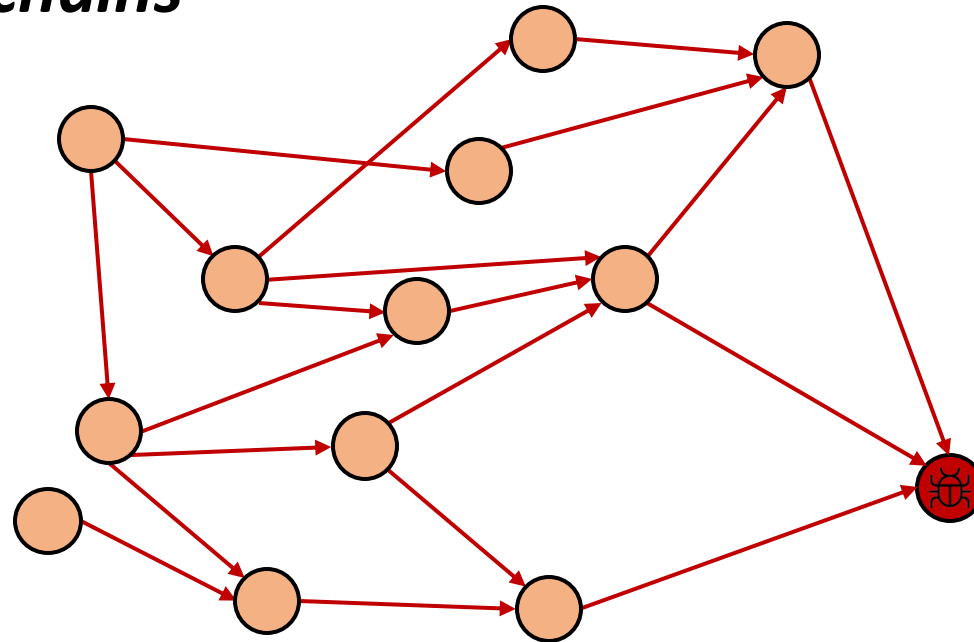
Table 2.1: CWEs associated with CVE-2021-44228



ID	Name	Source
CWE-917	Expression Language Injection	NIST
CWE-502	Deserialization of Untrusted Data	Apache Software Foundation
CWE-20	Improper Input Validation	Apache Software Foundation
CWE-400	Uncontrolled Resource Consumption	Apache Software Foundation

Vulnerability Impact Propagates



- ❑ Vulnerabilities are no longer confined to the packages they reside in
- ❑ Impact scope becomes much bigger due to ***dependency relations in software supply chains***



 Vulnerable project  Downstream projects affected by the vulnerable project

- ❑ How can we analyze the impact scope for a specific vulnerability
 - ❑ in software supply chains of the ecosystem where it is located?
 - ❑ **Lack of accurate and complete vulnerability propagation analysis**
- ❑ How can we quantify this impact scope?
 - ❑ **Lack of metrics for quantifying vulnerability impact across supply chains**
 - ❑ CVSS v4.0 FAQ* claims that there is no prescribed way to use CVSS Base and Environmental metrics to score a vulnerability along a long supply chain.



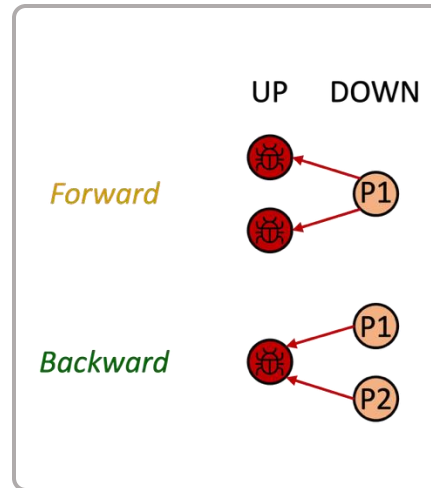
* <https://www.first.org/cvss/v4-0/faq>

Existing Studies

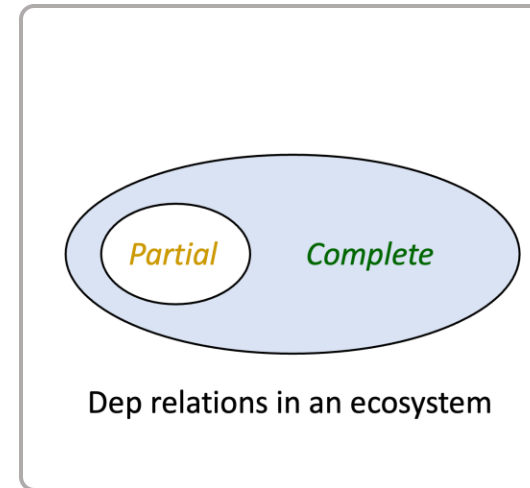


Vulnerability Propagation Analysis:
Identify the impact scope and scale of a vulnerability in software supply chains after it is disclosed.

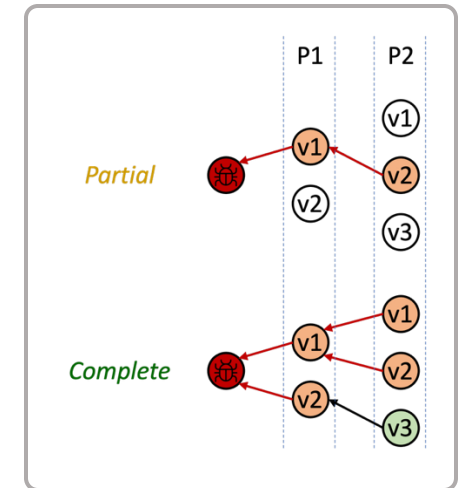
To clearly present and compare existing works, we profile them from six aspects in the figure on the right.



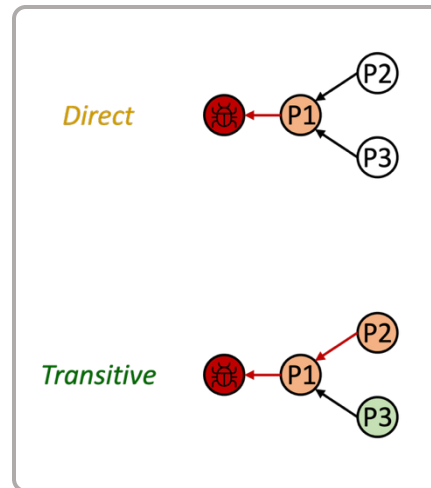
Direction



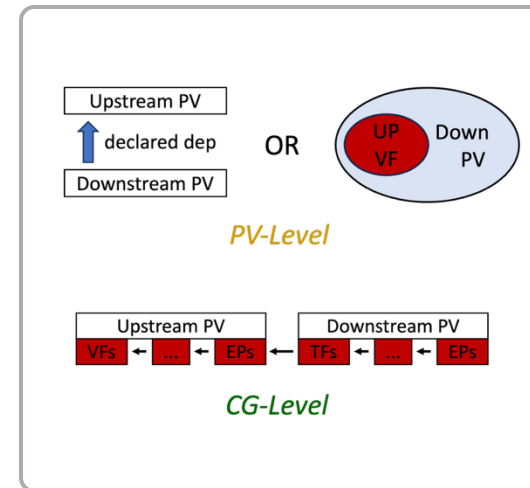
Scope



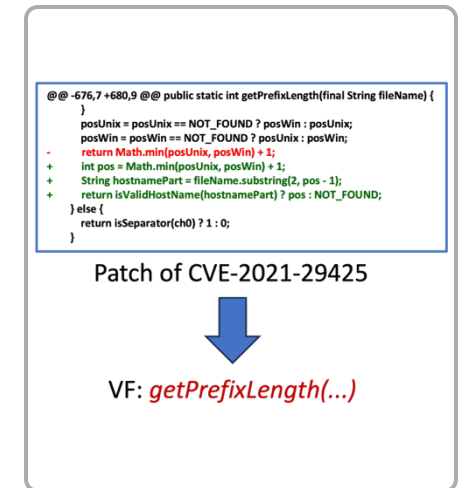
Coverage



Transitivity



Granularity



Vul Func Identification

Existing Studies



Vulnerability Propagation Analysis:

Identify the impact scope and scale of a vulnerability in software supply chains after it is disclosed.

To clearly present and compare existing works, we profile them from six aspects in the figure on the right.

Prior works are mainly empirical studies on *Java*, *JavaScript*, and *Python* ecosystems.

Year	LAN	Research	Direction	Dep Scope	Coverage	Transitivity	Granularity	VF Identification
2015	JA	Cadariu <i>et al.</i> [4]	Forward	Partial	Partial	Direct	PV	✗
2015	JA	Ponta <i>et al.</i> [5]	Forward	Partial	Partial	Direct	PV	Patch
2017	JS	Lauinger <i>et al.</i> [19]	Forward	Partial	Partial	Transitive	PV	✗
2018	JS	Decan <i>et al.</i> [20]	Forward	Partial	Partial	Direct	PV	✗
2018	JA	Kula <i>et al.</i> [6]	Forward	Partial	Partial	Direct	PV	✗
2018	JA	Du <i>et al.</i> [7]	Forward	Partial	Partial	Direct	PV	✗
2018	JA	Ponta <i>et al.</i> [8]	Forward	Partial	Partial	Direct	CG	Patch
2018	JA	Pashchenko <i>et al.</i> [9]	Forward	Partial	Partial	Direct	PV	Patch
2019	JA	Hu <i>et al.</i> [10]	Forward	Partial	Complete	Transitive	PV	✗
2019	JS	Zimmermann <i>et al.</i> [21]	Backward	Complete	Complete	Transitive	PV	✗
2020	JA	Wang <i>et al.</i> [11]	Forward	Partial	Partial	Direct	CG	Patch
2020	JA	Ponta <i>et al.</i> [12], [33]	Forward	Partial	Partial	Direct	CG	Patch
2020	PY	Ma <i>et al.</i> [24]	Backward	Partial	Partial	Transitive	CG	Manual
2022	JS	Liu <i>et al.</i> [23]	Backward	Complete	Complete	Transitive	PV	✗
2023	JS	Wang <i>et al.</i> [22]	Backward	Complete	Complete	Transitive	PV	✗
2023	JA	Zhang <i>et al.</i> [13]	Backward	Complete	Complete	Transitive	PV	✗
2023	JA	Wu <i>et al.</i> [14]	Forward	Complete	Partial	Direct	CG	Patch
2023	JA	Mir <i>et al.</i> [15]	Forward	Partial	Complete	Transitive	CG	Patch
2024	JA	Ma <i>et al.</i> [16]	Forward	Partial	Complete	Transitive	PV	✗
2024	JA	Zhang <i>et al.</i> [17]	Backward	Partial	Complete	Direct	CG	Patch (Optimized)
2025	JA	Shen <i>et al.</i> [18]	Backward	Partial	Partial	Transitive	CG	Patch



Existing works either stay at package-level and thus produce many false positives, or attempt call graph (CG)-level analysis but remain incomplete due to inefficient handling of large-scale ecosystem dependencies.

What We Need



Vulnerability Propagation Analysis:

Identify the impact scope and scale of a vulnerability in software supply chains after it is disclosed.



Year	LAN	Research	Direction	Dep Scope	Coverage	Transitivity	Granularity	VF Identification
2015	JA	Cadariu <i>et al.</i> [4]	Forward	Partial	Partial	Direct	PV	X
2015	JA	Ponta <i>et al.</i> [5]	Forward	Partial	Partial	Direct	PV	Patch
2017	JS	Lauinger <i>et al.</i> [19]	Forward	Partial	Partial	Transitive	PV	X
2018	JS	Decan <i>et al.</i> [20]	Forward	Partial	Partial	Direct	PV	X
2018	JA	Kula <i>et al.</i> [6]	Forward	Partial	Partial	Direct	PV	X
2018	JA	Du <i>et al.</i> [7]	Forward	Partial	Partial	Direct	PV	X
2018	JA	Ponta <i>et al.</i> [8]	Forward	Partial	Partial	Direct	CG	Patch
2018	JA	Pashchenko <i>et al.</i> [9]	Forward	Partial	Partial	Direct	PV	Patch
2019	JA	Hu <i>et al.</i> [10]	Forward	Partial	Complete	Transitive	PV	X
2019	JS	Zimmermann <i>et al.</i> [21]	Backward	Complete	Complete	Transitive	PV	X
2020	JA	Wang <i>et al.</i> [11]	Forward	Partial	Partial	Direct	CG	Patch
2020	JA	Ponta <i>et al.</i> [12], [33]	Forward	Partial	Partial	Direct	CG	Patch
2020	PY	Ma <i>et al.</i> [24]	Backward	Partial	Partial	Transitive	CG	Manual
2022	JS	Liu <i>et al.</i> [23]	Backward	Complete	Complete	Transitive	PV	X
2023	JS	Wang <i>et al.</i> [22]	Backward	Complete	Complete	Transitive	PV	X
2023	JA	Zhang <i>et al.</i> [13]	Backward	Complete	Complete	Transitive	PV	X
2023	JA	Wu <i>et al.</i> [14]	Forward	Complete	Partial	Direct	CG	Patch
2023	JA	Mir <i>et al.</i> [15]	Forward	Partial	Complete	Transitive	CG	Patch
2024	JA	Ma <i>et al.</i> [16]	Forward	Partial	Complete	Transitive	PV	X
2024	JA	Zhang <i>et al.</i> [17]	Backward	Partial	Complete	Direct	CG	Patch (Optimized)
2025	JA	Shen <i>et al.</i> [18]	Backward	Partial	Partial	Transitive	CG	Patch
		This Work	Backward	Complete	Complete	Transitive	CG	Patch (Optimized)

An accurate and comprehensive solution should be a **backward, transitive, CG-level** analysis that considers **complete dependency scope and project coverage** and is capable of **identifying vulnerable functions**.

- ❑ How can we analyze the impact scope for a specific vulnerability?
 - ❑ We draw inspiration from *data-flow analysis* and design a hierarchical worklist-based vulnerability propagation analysis algorithm to efficiently and accurately identify affected downstream dependencies across a whole software ecosystem.



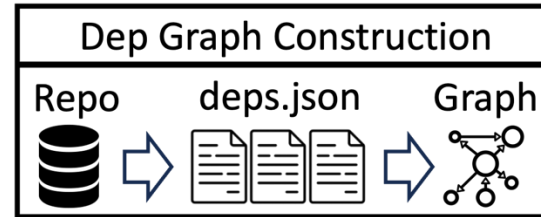
- ❑ How can we analyze the impact scope for a specific vulnerability?
 - ❑ We draw inspiration from *data-flow analysis* and design a hierarchical worklist-based vulnerability propagation analysis algorithm to efficiently and accurately identify affected downstream dependencies across a whole software ecosystem.
- ❑ How can we quantify this impact scope?
 - ❑ We propose the Vulnerability Propagation Scoring System (VPSS), a graph-theoretic dynamic indicator for quantifying vulnerability impact in software supply chains and reflecting the temporal evolution of impact.



Approach



The whole approach is designed to be ecosystem-agnostic and can be adapted to various programming languages.



Terminology

P: software Project

PV: Project Version release

- ❑ **Dep Graph Construction:** Build a *P*-level dep graph by analyzing all dep declaration files.

Approach

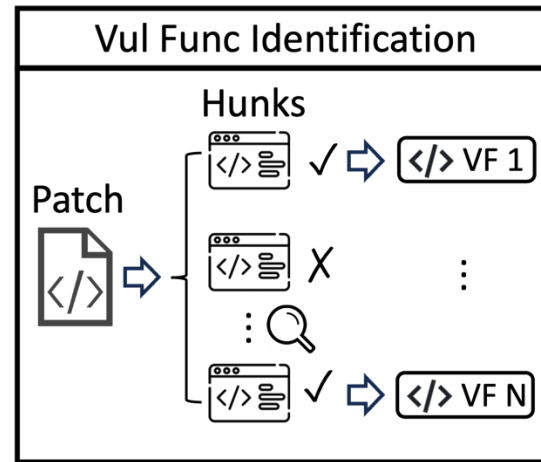
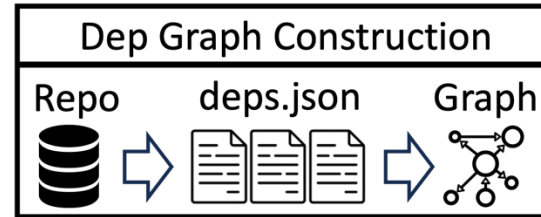


The whole approach is designed to be ecosystem-agnostic and can be adapted to various programming languages.

Terminology

P: software Project

PV: Project Version release



- ❑ **Dep Graph Construction:** Build a P -level dep graph by analyzing all dep declaration files.
- ❑ **Vulnerable Function Identification:** Generate VF candidates with patch-based method and filter with LLM-assisted strategy.

Approach

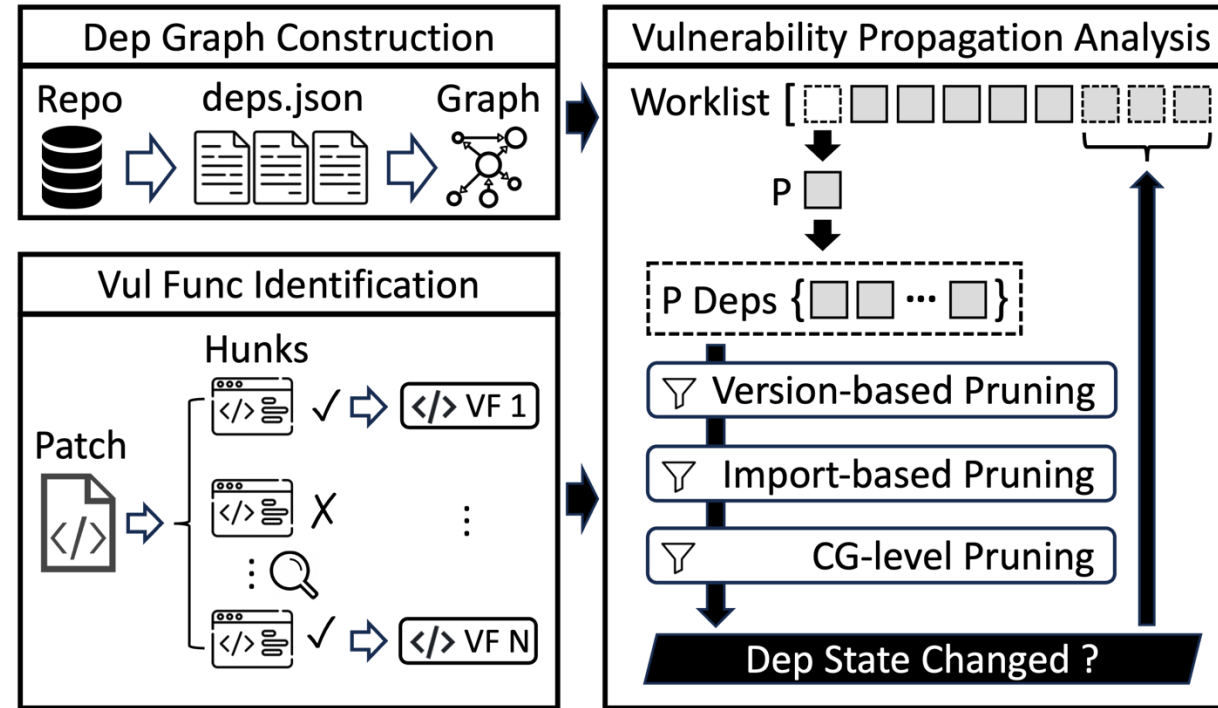


The whole approach is designed to be ecosystem-agnostic and can be adapted to various programming languages.

Terminology

P: software Project

PV: Project Version release



- ❑ **Dep Graph Construction:** Build a P -level dep graph by analyzing all dep declaration files.
- ❑ **Vulnerable Function Identification:** Generate VF candidates with patch-based method and filter with LLM-assisted strategy.

- ❑ **Vulnerability Propagation Analysis:** Identify all downstream PVs in the ecosystem that directly or transitively call VFs in vulnerable upstream PVs .

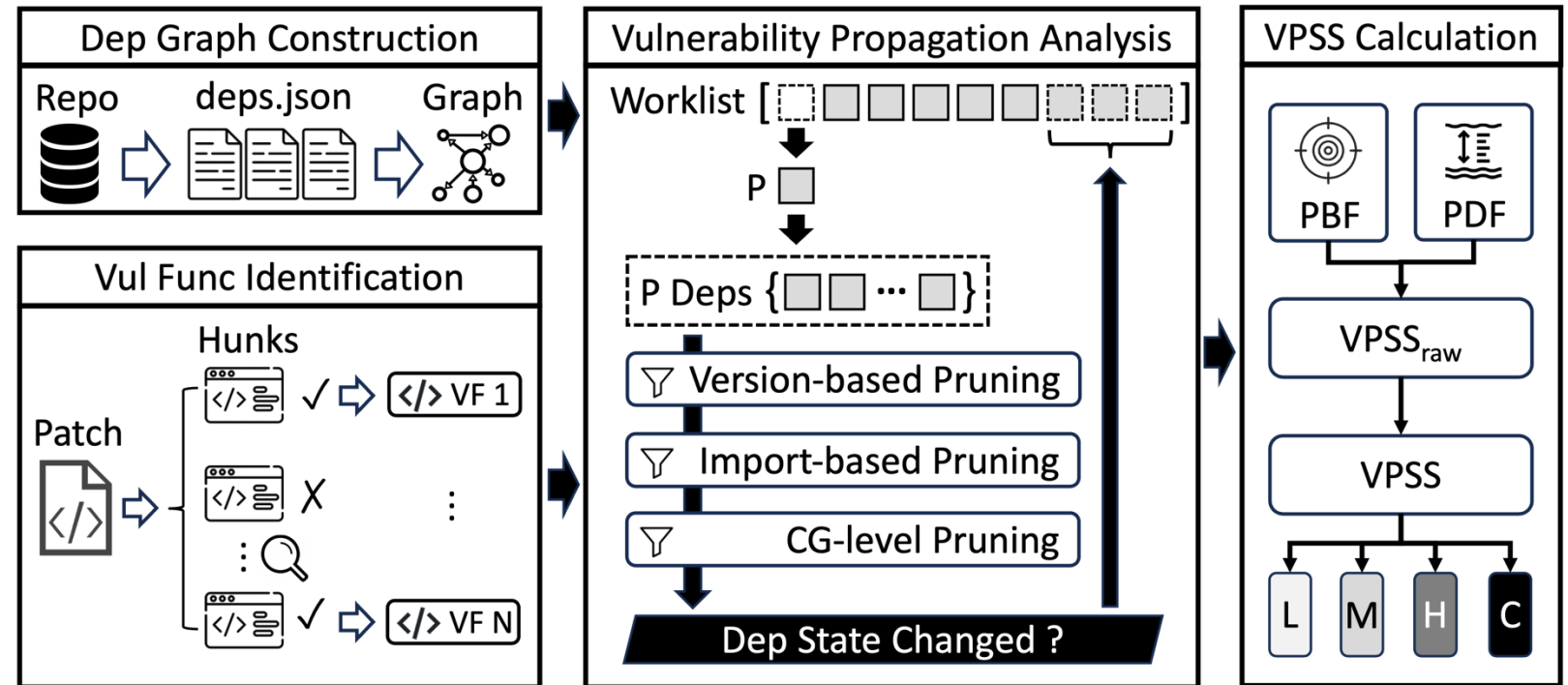
Approach



The whole approach is designed to be ecosystem-agnostic and can be adapted to various programming languages.

Terminology

P: software Project
PV: Project Version release



- ❑ **Dep Graph Construction:** Build a P -level dep graph by analyzing all dep declaration files.
- ❑ **Vulnerable Function Identification:** Generate VF candidates with patch-based method and filter with LLM-assisted strategy.
- ❑ **Vulnerability Propagation Analysis:** Identify all downstream PVs in the ecosystem that directly or transitively call VFs in vulnerable upstream PVs .
- ❑ **VPSS Calculation:** Compute the VPSS score based on the results of propagation analysis.

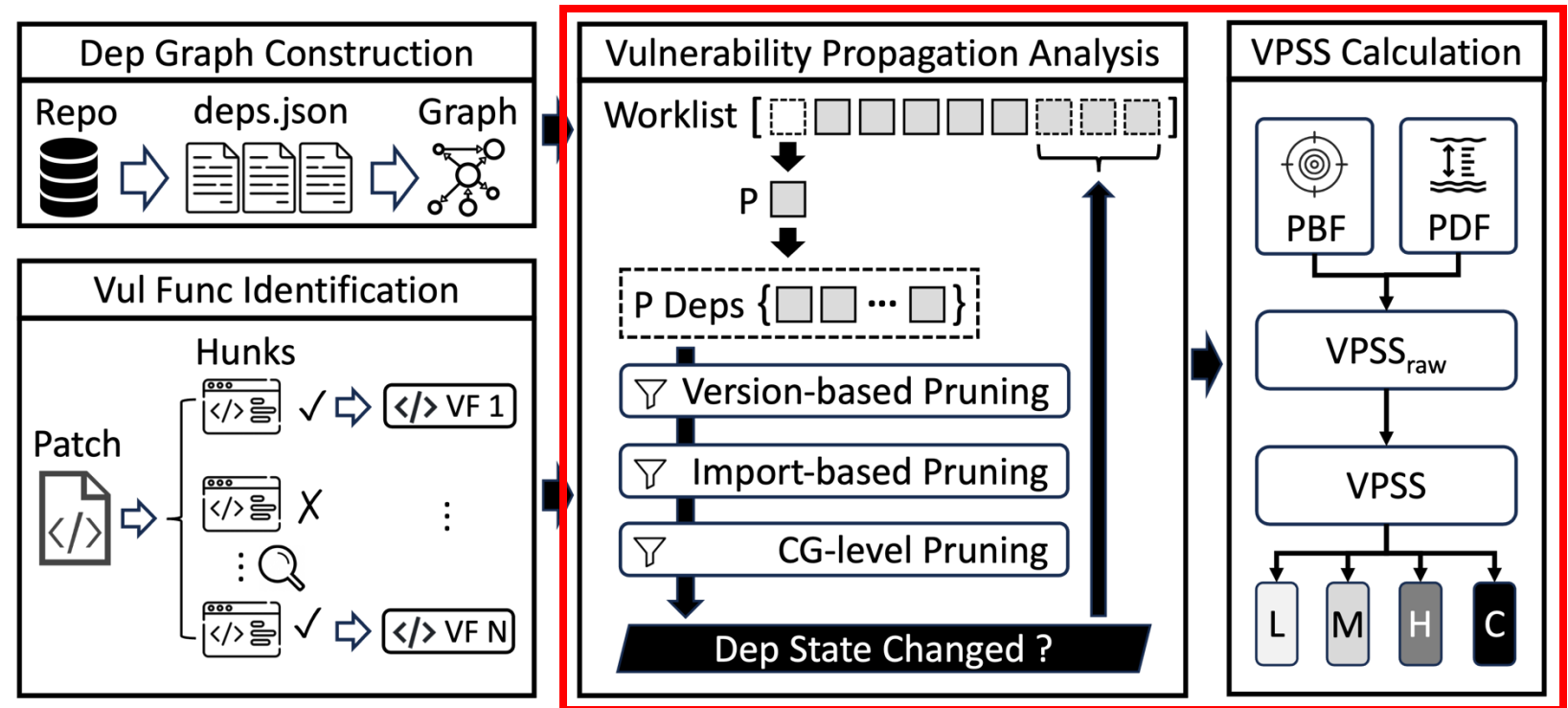
Approach



The whole approach is designed to be ecosystem-agnostic and can be adapted to various programming languages.

Terminology

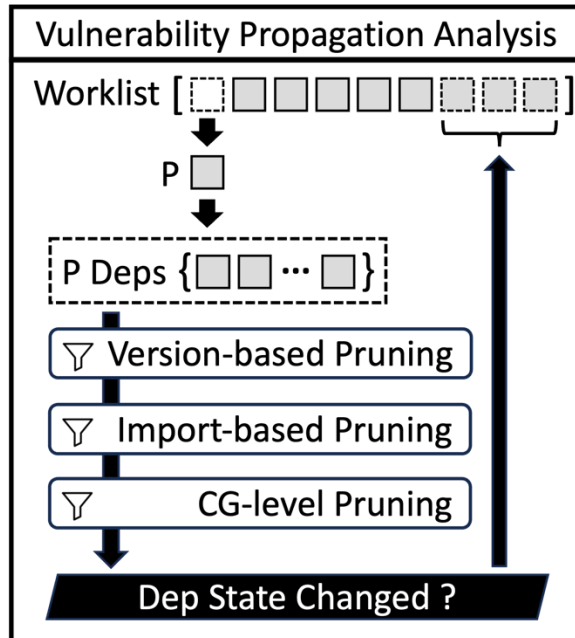
P: software Project
PV: Project Version release



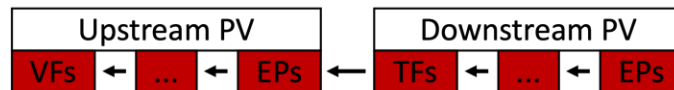
- ❑ **Dep Graph Construction:** Build a P -level dep graph by analyzing all dep declaration files.
- ❑ **Vulnerable Function Identification:** Generate VF candidates with patch-based method and filter with LLM-assisted strategy.

- ❑ **Vulnerability Propagation Analysis:** Identify all downstream PVs in the ecosystem that directly or transitively call VFs in vulnerable upstream PVs .
- ❑ **VPSS Calculation:** Compute the VPSS score based on the results of propagation analysis.

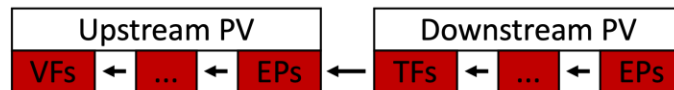
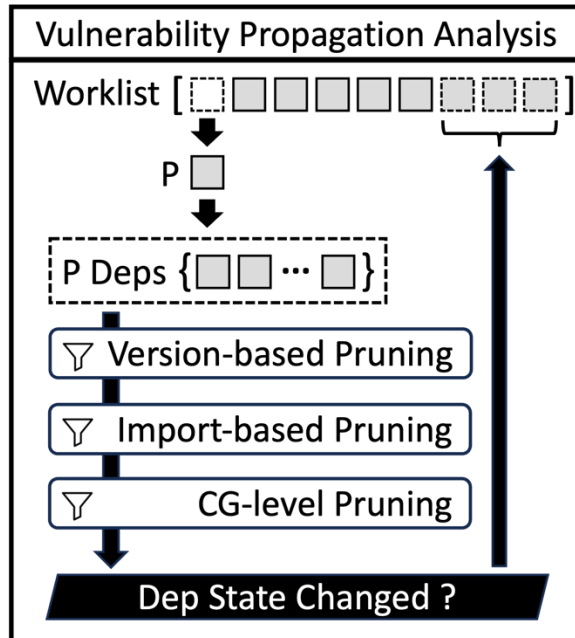
Vulnerability Propagation Analysis



- ❑ **Task:** Starting from a root P (comprising a series of vulnerable PVs), identify all downstream P s (with the corresponding PVs) affected by the vulnerability at CG level along the dependency graph.
- ❑ **Input:**
 - ❑ Root P of the vulnerability (VP); Worklist is initialized with VP
 - ❑ Vulnerable versions (VVs) of VP ; vulnerable functions (VFs) of VP
- ❑ **Output:** remaining dependency relations (P s and PVs), call paths

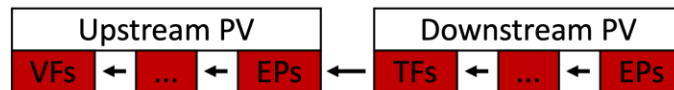
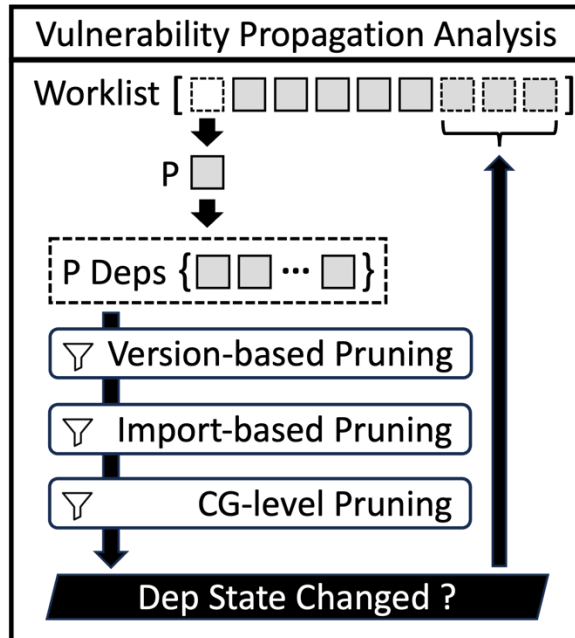


Vulnerability Propagation Analysis



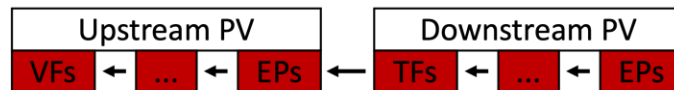
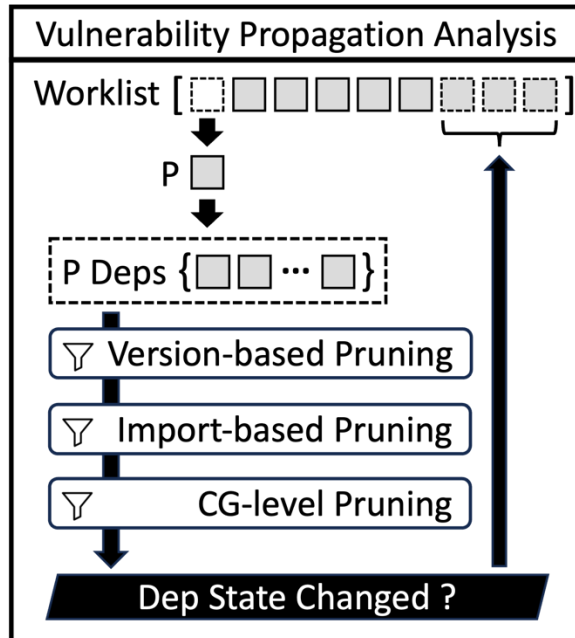
- ❑ **Task:** Starting from a root P (comprising a series of vulnerable PVs), identify all downstream P s (with the corresponding PVs) affected by the vulnerability at CG level along the dependency graph.
- ❑ **Input:**
 - ❑ Root P of the vulnerability (VP); Worklist is initialized with VP
 - ❑ Vulnerable versions (VVs) of VP ; vulnerable functions (VFs) of VP
- ❑ **Output:** remaining dependency relations (P s and PVs), call paths
- ❑ **Principles:**
 - ❑ *Perform as few analyses as possible*
 - ❑ Once the state of an upstream node changes, this change must propagate to downstream dependent nodes, even if they have been analyzed before.
 - ❑ Otherwise, the downstream nodes do not need to be analyzed again.

Vulnerability Propagation Analysis



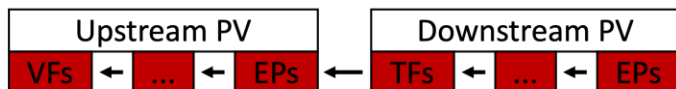
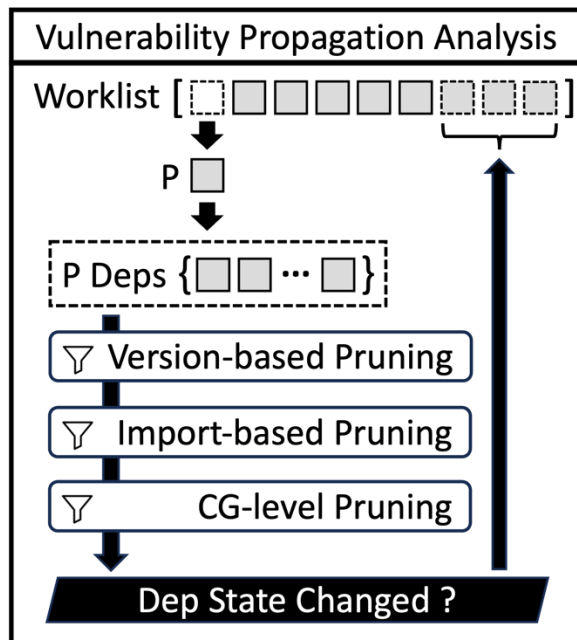
- ❑ **Task:** Starting from a root P (comprising a series of vulnerable PVs), identify all downstream P s (with the corresponding PVs) affected by the vulnerability at CG level along the dependency graph.
- ❑ **Input:**
 - ❑ Root P of the vulnerability (VP); Worklist is initialized with VP
 - ❑ Vulnerable versions (VVs) of VP ; vulnerable functions (VFs) of VP
- ❑ **Output:** remaining dependency relations (P s and PVs), call paths
- ❑ **Principles:**
 - ❑ *Perform as few analyses as possible*
 - ❑ Once the state of an upstream node changes, this change must propagate to downstream dependent nodes, even if they have been analyzed before.
 - ❑ Otherwise, the downstream nodes do not need to be analyzed again.
 - ❑ *Identify irrelevant nodes as early as possible*
 - ❑ Analyze one-hop dependency each time, filtering out false targets before proceeding to the next hop analysis.

Vulnerability Propagation Analysis



- ❑ **Task:** Starting from a root P (comprising a series of vulnerable PVs), identify all downstream P s (with the corresponding PVs) affected by the vulnerability at CG level along the dependency graph.
- ❑ **Input:**
 - ❑ Root P of the vulnerability (VP); Worklist is initialized with VP
 - ❑ Vulnerable versions (VVs) of VP ; vulnerable functions (VFs) of VP
- ❑ **Output:** remaining dependency relations (P s and PVs), call paths
- ❑ **Principles:**
 - ❑ *Perform as few analyses as possible*
 - ❑ Once the state of an upstream node changes, this change must propagate to downstream dependent nodes, even if they have been analyzed before.
 - ❑ Otherwise, the downstream nodes do not need to be analyzed again.
 - ❑ *Identify irrelevant nodes as early as possible*
 - ❑ Analyze one-hop dependency each time, filtering out false targets before proceeding to the next hop analysis.
 - ❑ *Perform rapid, coarse-grained analysis as early as possible*
 - ❑ *Perform heavy, fine-grained analysis as late as possible*

Vulnerability Propagation Analysis

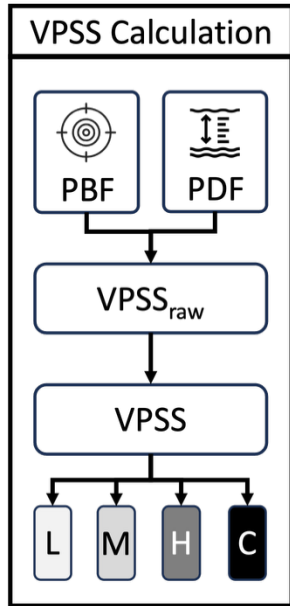


- ❑ **Input:**
 - ❑ Root P of the vulnerability (VP); Worklist is initialized with VP
 - ❑ Vulnerable versions (VVs) of VP ; vulnerable functions (VFs) of VP
- ❑ **Output:** remaining dependency relations (Ps and PVs), call paths
- ❑ **Workflow:**
 - ❑ At the beginning of each pass, one item (P) is popped from the worklist
 - ❑ Generate the target version (TV) list and target function (TF) list
 - ❑ For the first pass, they are initialized with VVs and VFs
 - ❑ Query the dep graph for downstream Ps
 - ❑ Hierarchical pruning:
 - ❑ **Version-based Pruning:** remove dependents on PVs not in TV
 - ❑ **Import-based Pruning:** remove dependents that declare but do not import upstream classes
 - ❑ **CG-level Pruning:** remove dependents that import but do not call vulnerable upstream entrypoints
 - ❑ If the dep state of current item changed, enqueue downstream Ps
 - ❑ **Dep State:** entry points (EPs) of the current item
 - ❑ *Continue to the next pass until the worklist becomes empty*

- ❑ **Motivation:** Propagation analysis provides statistics reflecting vulnerability impact in ecosystem
 - ❑ However, the raw data are not readily interpretable or actionable
 - ❑ To better profile this impact, we introduce **Vulnerability Propagation Scoring System (VPSS)**
- ❑ **Design Principles:**
 - ❑ **Graph Awareness:** Capture how widely and deeply a vulnerability propagates
 - ❑ Higher scores reflect wider and deeper propagation in the ecosystem
 - ❑ **Interpretability & Compatibility:** Be easy to understand & integrate
 - ❑ **Time Awareness:** Be dynamic, supporting longitudinal tracking and timely assessment
- ❑ **Definition:** A propagation-aware measure of vulnerability impact that combines two dimensions:
 - ❑ **Breadth** (affected share of downstream packages)
 - ❑ **Depth** (the length of dependency chains)



VPSS Calculation



Two multiplicative factors of VPSS:

- ❑ **Propagation Breadth Factor (PBF)** quantifies how widely a vuln spreads
 - ❑ via direct & transitive dependencies
- ❑ **Propagation Depth Factor (PDF)** measures how deeply a vuln penetrates the dep graph
 - ❑ based on propagation chain length

Terminology

P: software Project
PV: Project Version release

$$VPSS_{raw} = PBF \times PDF$$

$$PBF = \ln(1 + \gamma \cdot WX^T)$$

$$PDF = 1 + \frac{L_{max} + L_{avg}}{2L_{norm}}$$

$$VPSS = 10 \times \left(1 - \exp\left(-\frac{VPSS_{raw}}{k}\right)\right)$$

$$W = (W_{p_dir} \quad W_{p_trans} \quad W_{pv_dir} \quad W_{pv_trans})$$

$$X = (r_{p_dir} \quad r_{p_trans} \quad r_{pv_dir} \quad r_{pv_trans})$$

$$r_{p_dir} = \frac{P_{dir}}{Total_P} \quad r_{p_trans} = \frac{P_{trans}}{Total_P}$$

$$r_{pv_dir} = \frac{PV_{dir}}{Total_PV} \quad r_{pv_trans} = \frac{PV_{trans}}{Total_PV}$$

* $W_{p_dir}, W_{p_trans}, W_{pv_dir}, W_{pv_trans}, \gamma, L_{norm},$ and k are parameters



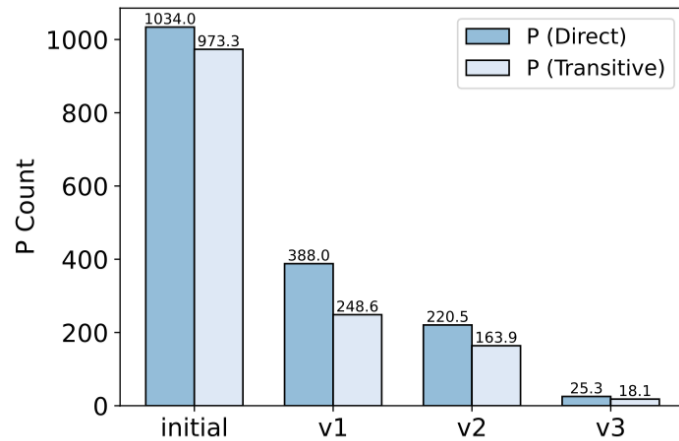
Low [0-4]	Medium [4-7]	High [7-9]	Critical [9-10]
-----------	--------------	------------	-----------------

- ❑ **Implementation:** We implement a prototype for Java Maven ecosystem
- ❑ **Evaluation Questions:**
 - ❑ **EQ1:** How effective and scalable is our ecosystem-scale vulnerability propagation analysis in identifying affected downstream projects?
 - ❑ **EQ2:** What insights can be drawn from the VPSS scores?
- ❑ **Dataset:**
 - ❑ **Base:** 100 vulnerabilities from the original dataset released by Wu et al. [1]
 - ❑ **Augmentation:**
 - ❑ Incorporate the complete list of vulnerable versions for each CVE
 - ❑ Identify and filter out inaccurate vulnerable functions

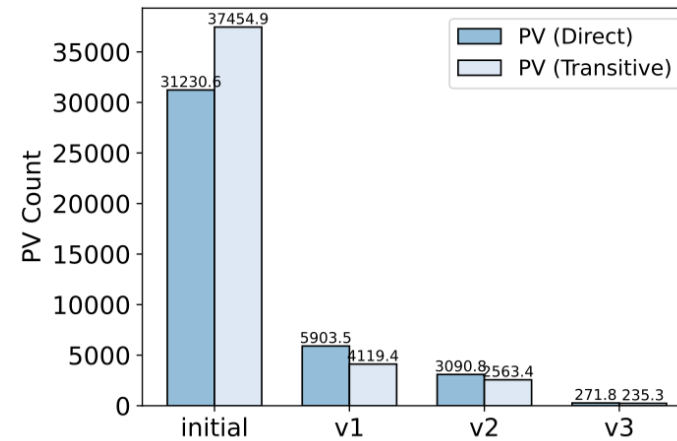


[1] Wu, Yulun, et al. "Understanding the threats of upstream vulnerabilities to downstream projects in the maven ecosystem." *2023 IEEE/ACM 45th International Conference on Software Engineering (ICSE)*. IEEE, 2023.

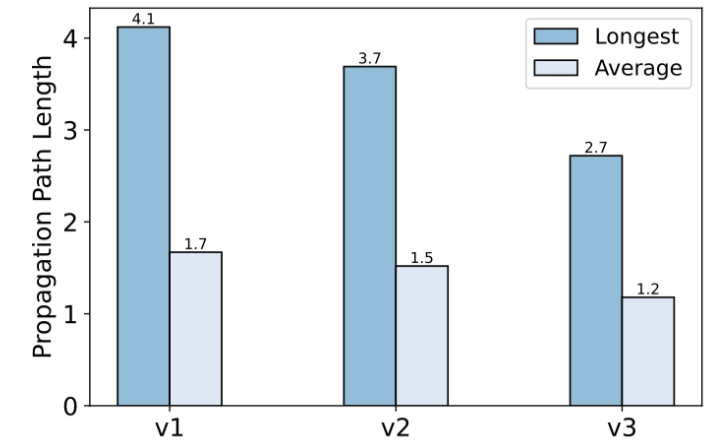
EQ1: Effectiveness



(a) Average P (Direct vs Transitive)



(b) Average PV (Direct vs Transitive)



(c) Longest and Average Path Length

❑ **Maven Snapshot:** December 26, 2024 (around 660K Ps, 15M PVs)

❑ **Per-Vulnerability Analysis Time:** From 1.2 seconds to 54 hours, with an average of 5.2 hours

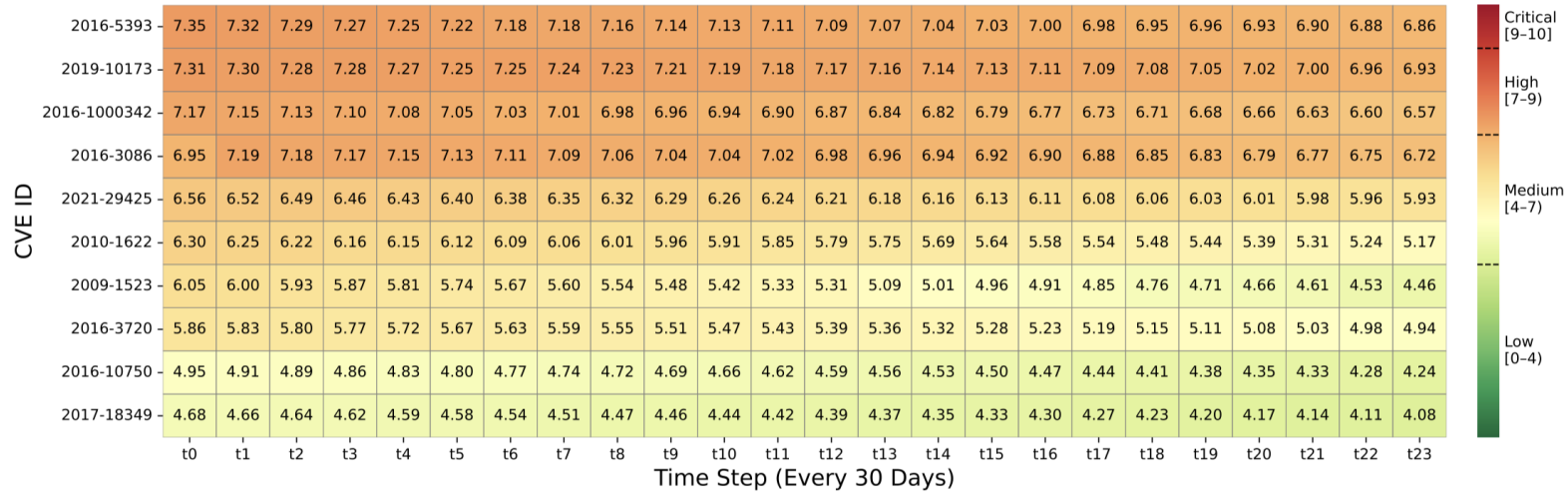
❑ **Findings:**

❑ Successfully & efficiently completes propagation analysis for all 100 vulnerabilities

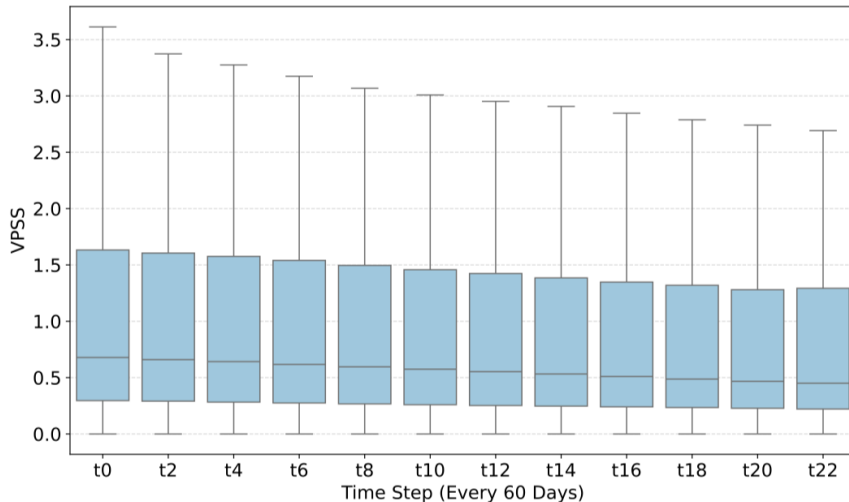
❑ On average, 97.8% of Ps & 99.2% of PVs are pruned, with the longest & average propagation path lengths reduced by at least 34.1% & 29.4%, respectively

❑ Significantly lowers the cost of CG construction by reducing the number of needed CGs

EQ2: Findings



VPSS Time Series for Top-10 CVEs (t0 to t23)



VPSS Distribution Across 100 CVEs

Parameter Setting:

$w_{p_dir} = 5, w_{p_trans} = 2.5, w_{pv_dir} = 3, w_{pv_trans} = 1.5$

$\gamma = 500, L_{norm} = 10, k = 0.5$

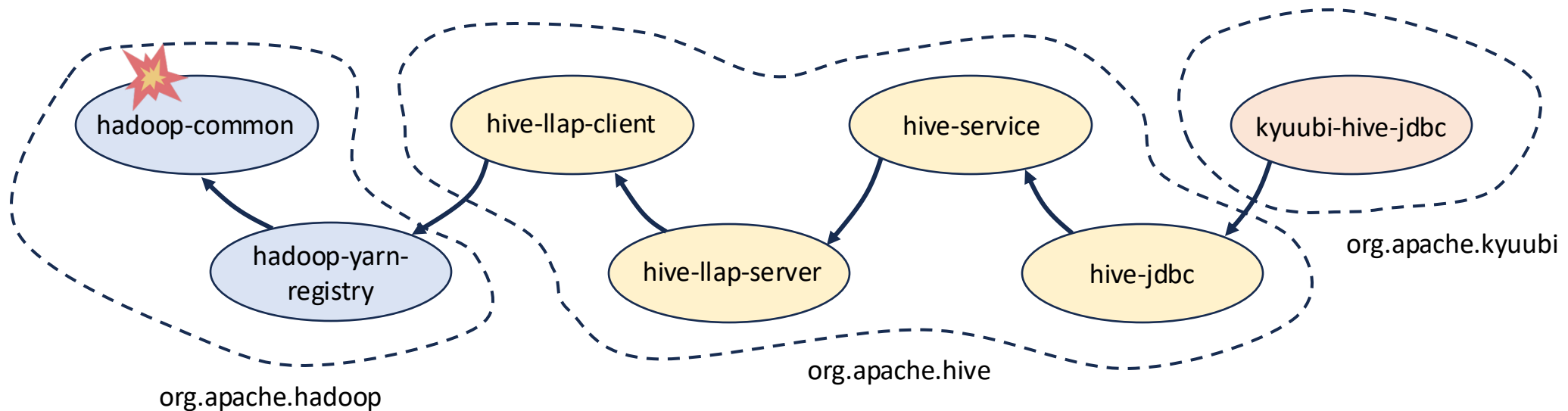
Findings:

- VPSS scores generally decline over time after disclosure
- Some CVEs (e.g., CVE-2016-3086) show temporary score increases
- Across the dataset, VPSS scores remain low and gradually stabilize

Case Study: CVE-2016-5393



- ❑ CVE-2016-5393, a high-severity vulnerability in *org.apache.hadoop:hadoop-common*
 - ❑ On exploitation, a remote attacker can execute commands with HDFS privileges
- ❑ At t0 (disclosure time), it directly affected 228 *Ps* and transitively propagated to 154 others, impacting a total of 618 direct and 321 transitive *PVs*
 - ❑ Its VPSS score reached 7.35 (high), the highest among all CVEs in our dataset
- ❑ Over 24 months, its VPSS score gradually declines to 6.86, indicating a slow mitigation pace
- ❑ The longest propagation chain consists of 7 dependency hops
 - ❑ spanning critical components of the Hadoop and Hive data processing stacks



Key points:

- ❑ Vulnerability propagation analysis algorithm
- ❑ Vulnerability Propagation Scoring System (VPSS)
- ❑ Prototype for the Java Maven ecosystem
- ❑ Augmented dataset for evaluation

Code & dataset: github.com/brant-ruan/vpss

Contact: bonan.ryan@u.nus.edu

