Computer & Communication Security Laboratory

V0Finder: Discovering the Correct Origin of Publicly Reported Software Vulnerabilities

Seunghoon Woo*, Dongwook Lee*, Sunghan Park*, Heejo Lee*, Sven Dietrich**

***Korea University**, **City University of New York

USENIX security 2021

Goal

Discovering the correct origin of publicly reported software vulnerabilities

- **Vulnerability Zero (VZ)**
  - The software and its version where a vulnerability **<u>originated</u>**

- **Motivation**
  - The incorrect VZ can cause several security problems

    🙁 To unintentionally overlook the propagated vulnerability

    🙁 To delay patch deployment

Motivating example: CVE-2017-0700

The VZ of CVE-2017-0700 is reported as Android

Current Description

A remote code execution vulnerability in the Android system ui. Product: Android. Versions: 7.1.1, 7.1.2. Android ID: A-35639138.

Known Affected Software Configurations

Configuration 1 (hide)

cpe:2.3:o:google:android:7.1.1:*:*:*:*:*:*:*

Show Matching CPE(s)

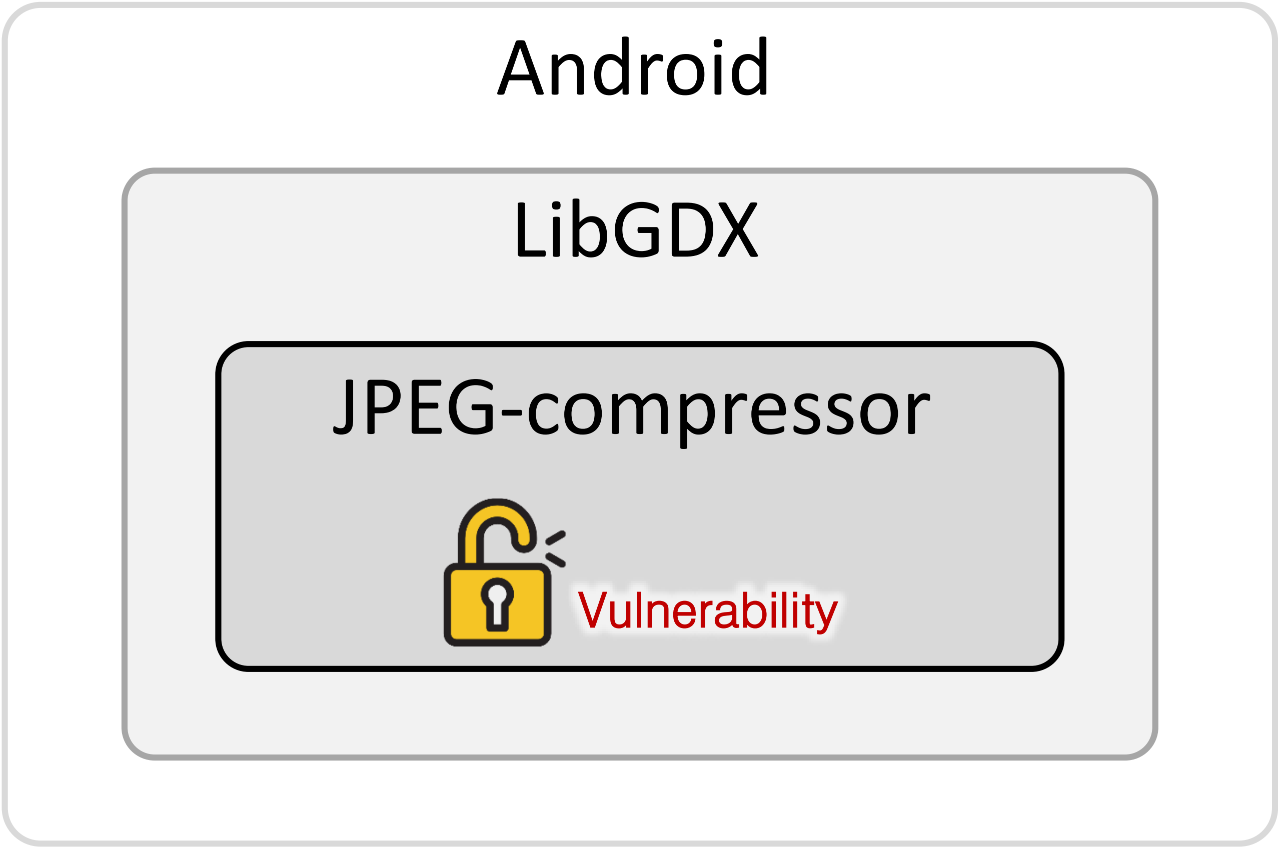
cpe:2.3:o:google:android:7.1.2:*:*:*:*:*:*:*

Show Matching CPE(s)

<https://nvd.nist.gov/vuln/detail/CVE-2017-0700>

Motivating example: CVE-2017-0700

The correct VZ of CVE-2017-0700 is JPEG-compressor



Motivating example: CVE-2017-0700

Developers reusing vulnerable Android

▶ Can **easily resolve** the vulnerability

Developers reusing vulnerable JPEG-compressor

▶ **Fail** to detect and patch the vulnerability in a timely manner

Motivating example: CVE-2017-0700

Developers reusing vulnerable Android

- Can **easily** resolve the vulnerability

Developers reusing vulnerable JPEG-compressor

- **Fail** to detect and patch the vulnerability in a timely manner

Motivating example: CVE-2017-0700

Developers reusing vulnerable Android

- Can easily resolve the vulnerability

Developers reusing vulnerable JPEG-compressor

Successfully reproduced CVE-2017-0700 in

- JPEG-compressor
- Godot (reported -> patched)
- LibGDX (reported -> patched)

(this CVE exists in the latest version of 12 software!)

Motivating example: CVE-2017-0700

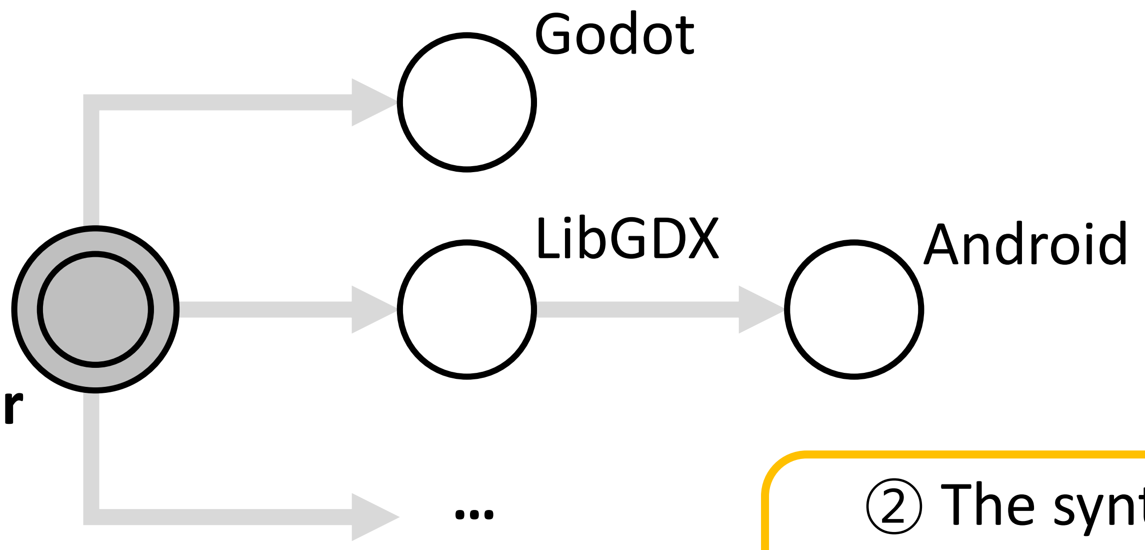
Discovering the correct VZ of a vulnerability in an automated way

- **Input**
  - CVE-2017-0700 vulnerability (i.e., vulnerable function)
- **Output**
  - JPEG-compressor (the correct VZ, not LibGDX and Android)

Challenge

1. **VZ ≠ Vulnerable software with the earliest birth date!**
2. **Addressing the syntax variety of vulnerable code**

① VZ may not be the vulnerable software with the earliest birth date



② The syntax of a vulnerable code frequently changes

V0Finder

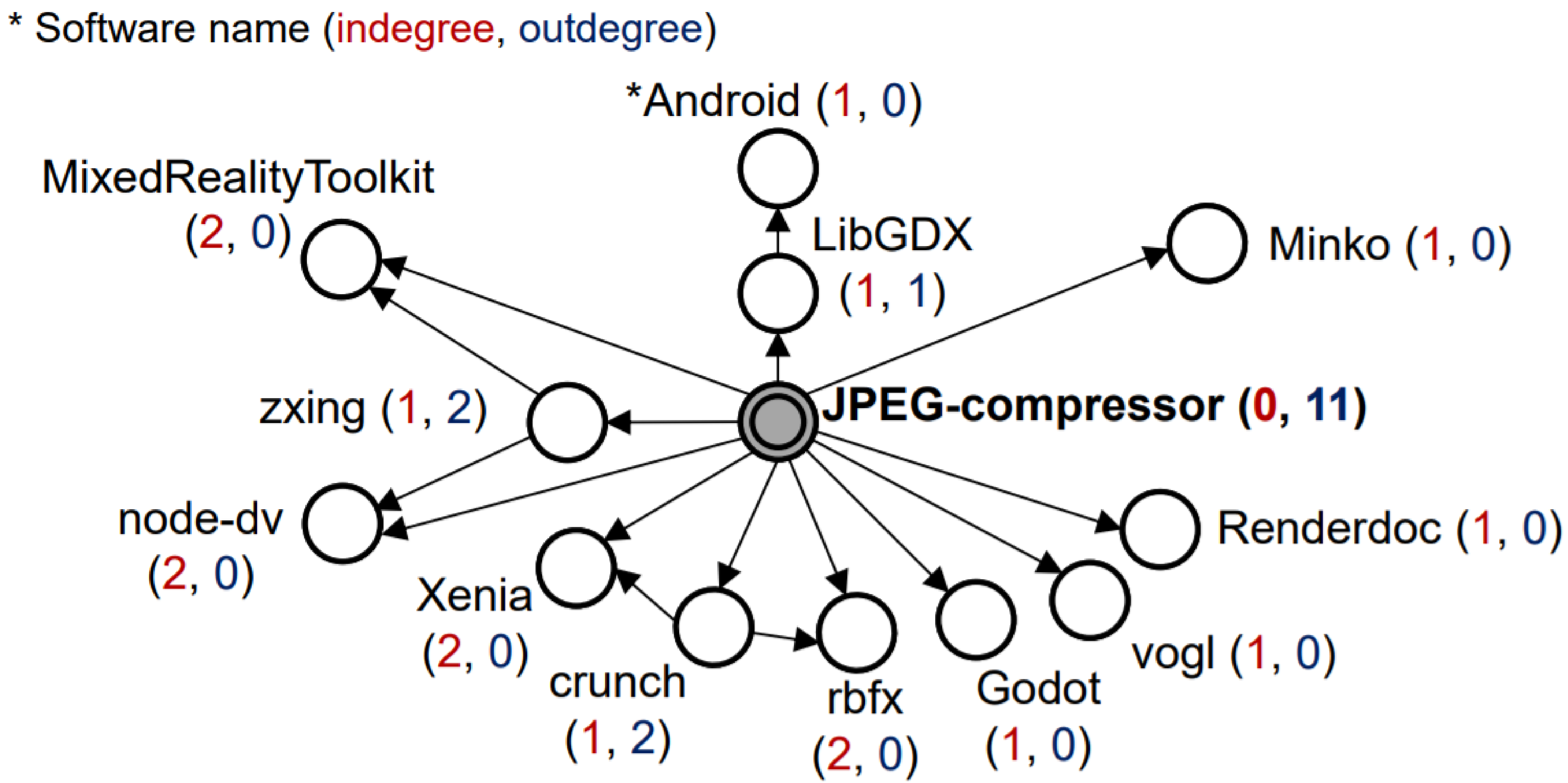
Vulnerability **Zero Finder**

- The first approach to discover **the correct origin (VZ)** of a vulnerability

- **Key idea**
  - Using a **graph-based approach** instead of using timestamp-based metadata
    - Generating a **vulnerability propagation graph** for each vulnerability
      - Nodes: vulnerable software
      - Edges: the propagation directions of the vulnerability
    - Discovering the VZ → <u>finding the root of the generated graph</u>

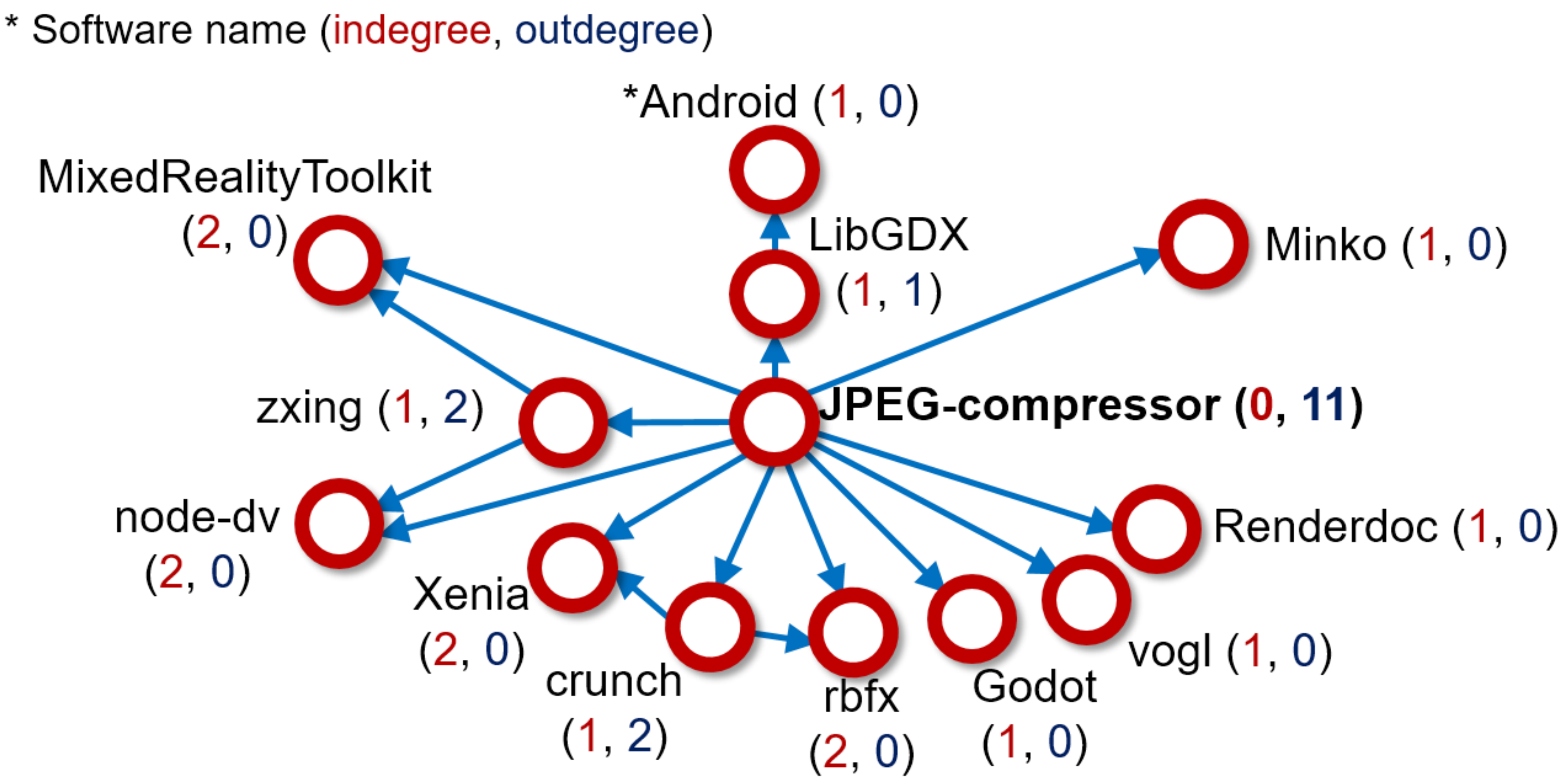
V0Finder: Example of the generated graph

- Example vulnerability propagation graph (CVE-2017-0700)



V0Finder: Example of the generated graph

- Example vulnerability propagation graph (CVE-2017-0700)

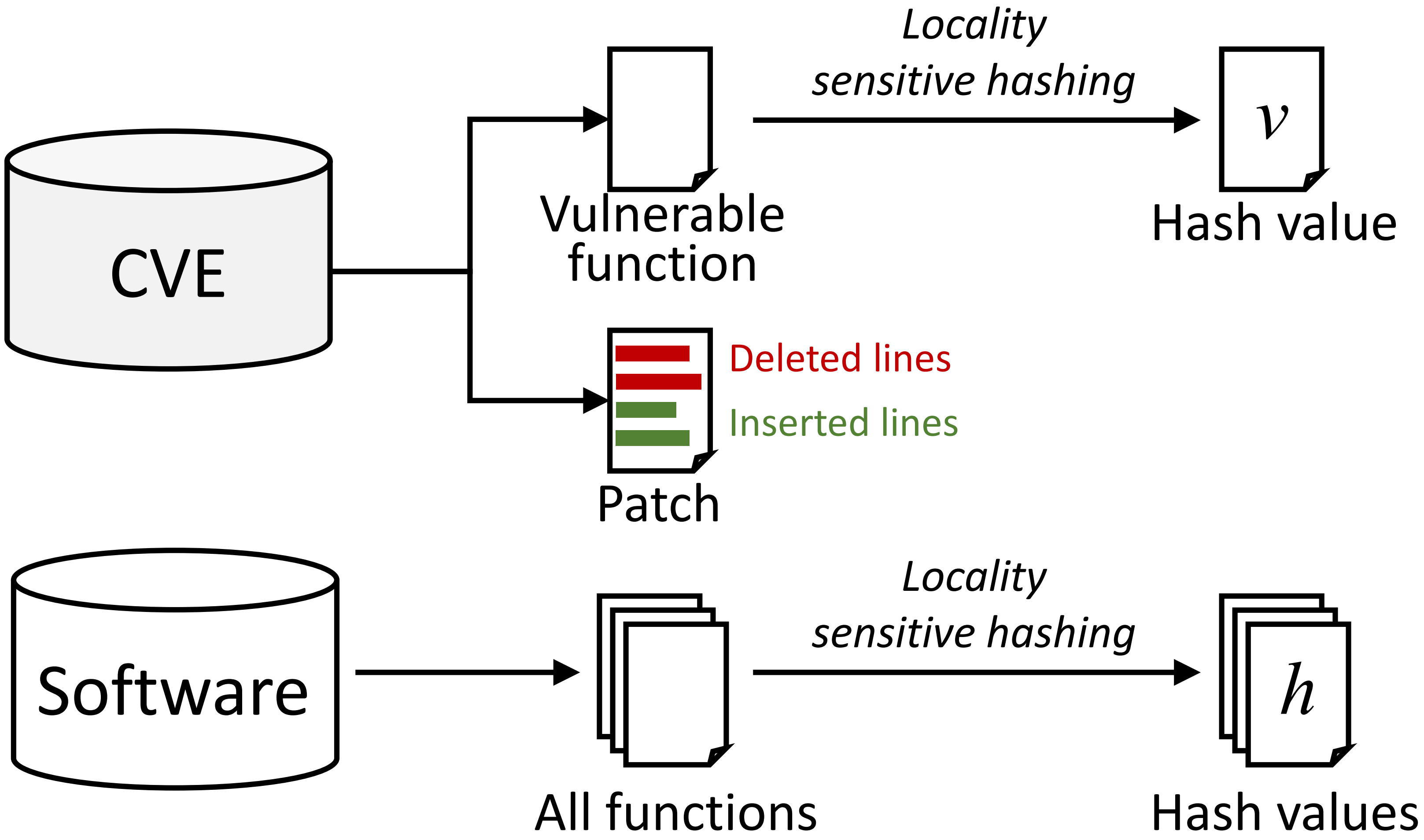


① Detects vulnerable software (nodes)

② Identifies propagation directions (edges)

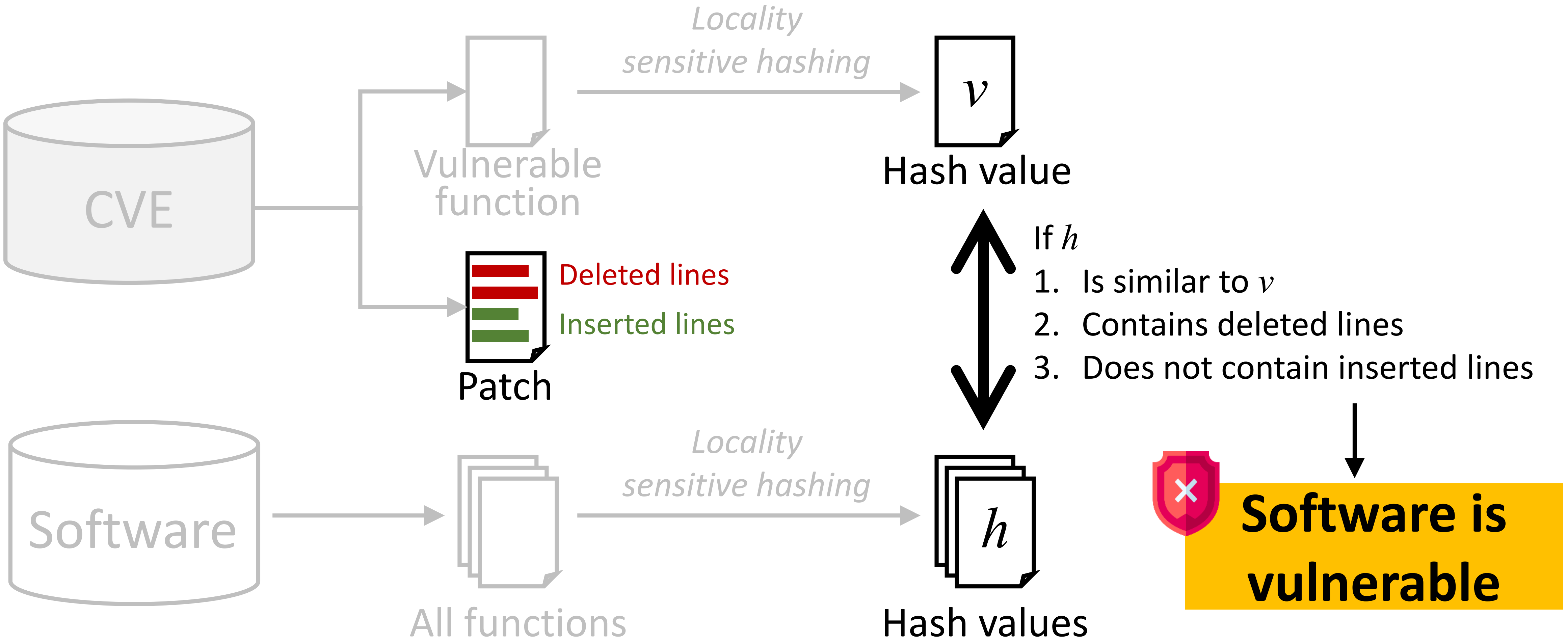
Detecting vulnerable software

Extracting functions and applying locality sensitive hashing



Detecting vulnerable software

Using vulnerable code clone detection technique

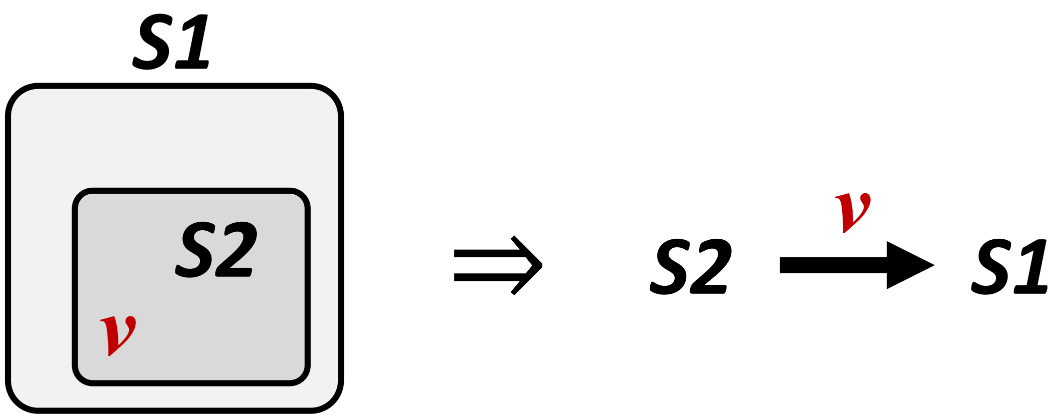


Identifying propagation directions

Focusing on a reuse relation

- **Reuse relation** between the vulnerable software pairs (S1, S2)
  - Let a vulnerability be *v*
- **If S1 reuses S2, and if S1 and S2 share the same vulnerability** *v*

  ⇒ *v* propagated from S2 to S1



Identifying propagation directions

Identifying reuse relations using three key factors

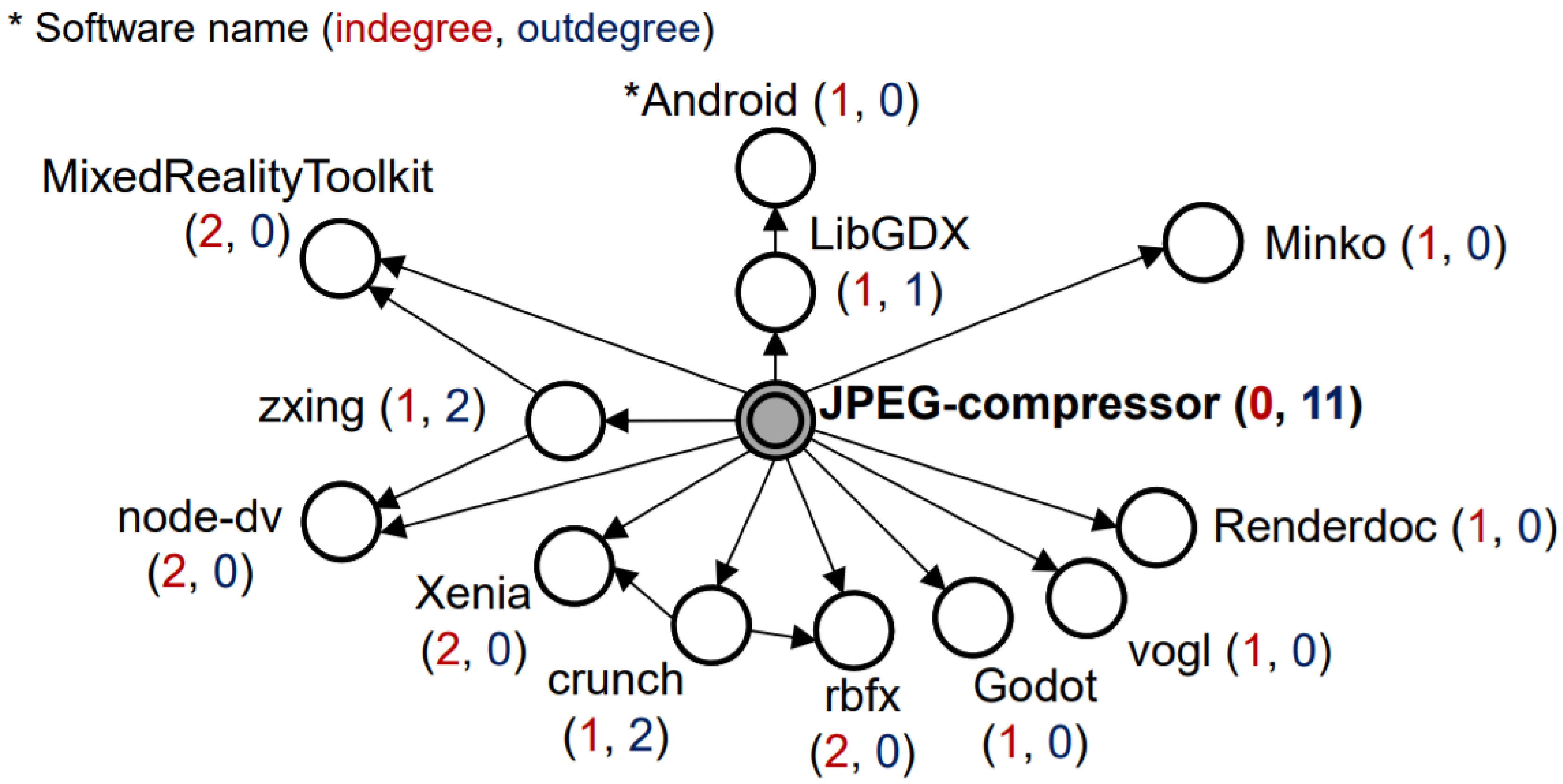
- V0Finder determines that **S1 reuses S2** in the following three cases
  1. **[Source code]** If S1 contains the entire codebase of S2
  2. **[Path information]** If path(*S1, a common file*) ⊃ path(*S2, a common file*)

     e.g.,
     ```
     JPEG-compressor: "./jpgd.cpp"
           Godot     : "./thirdparty/jpeg-compressor/jpgd.cpp"
     ```
  3. **[Metadata files]** If S1 contains a metadata file of S2
     - `README`, `LICENSE`, and `COPYING` files located in the root path of S2

* If S1 reuses S2, then the vulnerability propagated from S2 to S1 **(S2 → S1)**

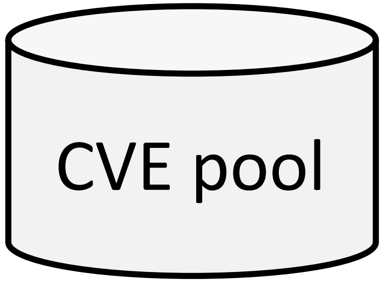
Finding the root of the generated graph

Discovering the VZ by finding the root of the graph

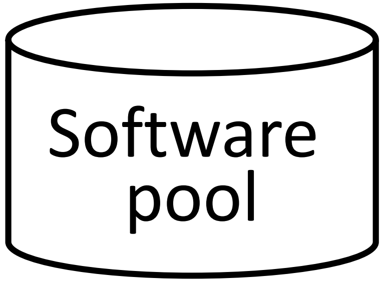


Evaluation

Dataset collection



- **5,671 CVEs**
  - 3,246 CVEs from NVD (all C/C++ CVEs that provide their patch information)
  - 2,425 CVEs from Issue trackers (Android, Chromium, Mozilla)



- **10,701 software programs**
  - Popular open-source software from GitHub (ranked by the number of stars)
  - A total of 229,326 versions and 80 billion lines of code

Evaluation

Evaluation methodology

1) **Discovering VZs for the collected 5,671 CVEs**

2) **Comparing the VZ discovery results of V0Finder using the CPEs**
   - Common Platform Enumeration (CPE)
   - Provides vulnerable software name & version

Known Affected Software Configurations

Configuration 1 (hide)

cpe:2.3:o:google:android:7.1.1:*:*:*:*:*:*:*

Show Matching CPE(s)

cpe:2.3:o:google:android:7.1.2:*:*:*:*:*:*:*

Show Matching CPE(s)

CPE of CVE-2017-0700

<https://nvd.nist.gov/vuln/detail/CVE-2017-0700>

Evaluation

VZ discovery results for the collected 5,671 CVEs

1. **V0Finder successfully discovered the correct VZs for 5,410 CVEs (95%)**
2. **V0Finder further found that 96 CVEs with the incorrect VZ**

Evaluation

VZ discovery results for the collected 5,671 CVEs

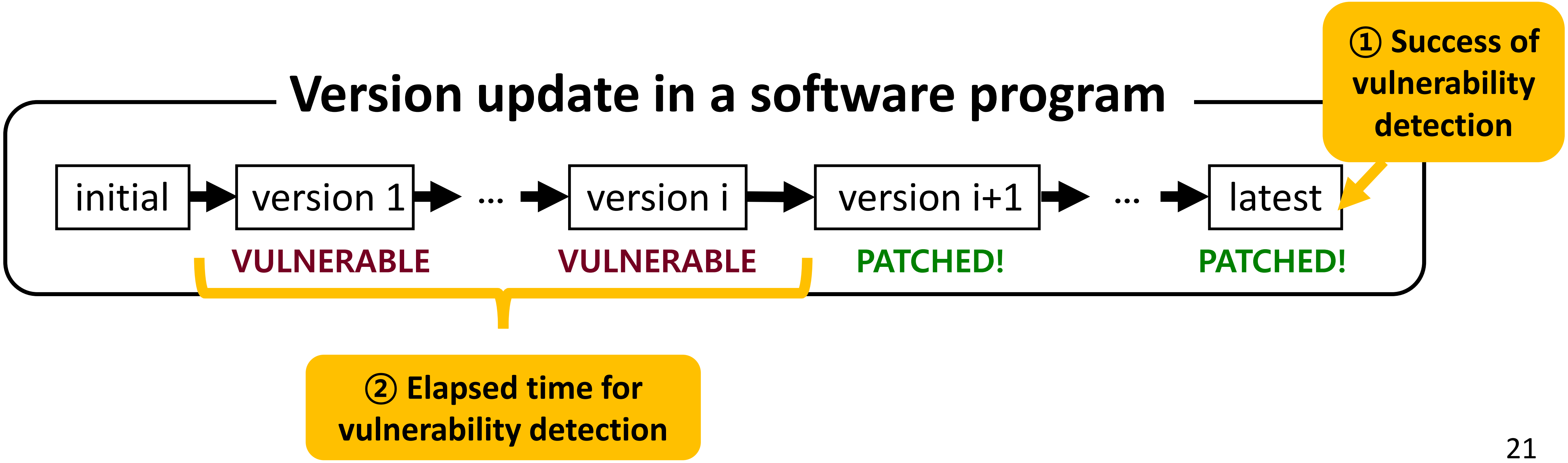
1. **V0Finder successfully discovered the correct VZs for 5,410 CVEs (95%)**
2. **V0Finder further found that 96 CVEs with the incorrect VZ**
3. **Graphs with multiple roots or with no root**
   1) VZ does not exist in our dataset
   2) V0Finder failed to identify reuse relations for some cases

#CVEs	#TP	#FP	#TN	#FN	Precision (%)	Recall (%)
Total results:						
5,671	5,410	52	70	139	**99**	**97**
Excluding CVEs with a single node in the graph:						
3,164	2,903	52	70	139	**98**	**95**

Findings

Analyzing the impact of VZ discovery

1) **Success rate** of vulnerability detection *VS.* the correctness of VZ
2) **Elapsed time** for vulnerability detection *VS.* the correctness of VZ



Success rate of vulnerability detection

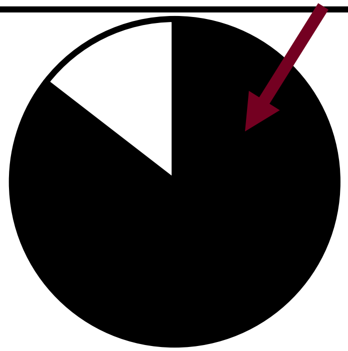
The incorrect VZ prevents appropriate vulnerability detection

CVEs with the correct VZ

3,068 CVEs

10,523 affected software

85% (8,994) affected software
can detect and patch the vulnerability

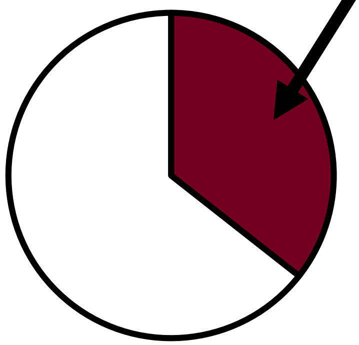


CVEs with the incorrect VZ

96 CVEs

1,000 affected software

36% (356) affected software
can detect and patch the vulnerability



※ We only consider the case where there is more than one node in the graph

Elapsed time for vulnerability detection

Elapsed time for vulnerability detection in the affected software

CVEs with the correct VZ

CVEs with the incorrect VZ

308 days (average) ≪ 521 days (average)

200 days

Implications

The implications of correct VZ discovery

1. **Some CVEs are reported with the incorrect VZ**
   - The incorrect VZ hinders detection and patching of propagated vulnerabilities
2. **The correct VZ of a vulnerability enables developers to detect and patch propagated vulnerabilities in a timely manner**
3. **The task of discovering the VZ should be automated and accurately performed with a system such as V0Finder**

Conclusion

- **Quality control of vulnerability reports is an important issue**
  - The correctness of VZ has a significant impact on the appropriate detecting and patching of propagated vulnerabilities
- **We present V0Finder, for the first time, an approach to precisely discover the correct VZ of software vulnerabilities**
  - Discovering VZs by generating vulnerability propagation graph
- **Equipped with VZ discovery results from V0Finder**
  - Developers can address software vulnerabilities potentially contained in their software due to vulnerable code reuse in a timely manner

Q&A

Thank you for your attention!

- V0Finder repository (https://github.com/wooseunghoon/V0Finder-public)

CONTACT

- Seunghoon Woo (seunghoonwoo@korea.ac.kr, https://wooseunghoon.github.io)
- Computer & Communication Security Lab (https://ccs.korea.ac.kr)
- Center for Software Security and Assurance (https://cssa.korea.ac.kr)