CLOVERY

Identifying affected versions in C/C++ public security vulnerability reports via function-lineage tracking

Duyeong Kim, Jimin Kang, Yeonhee Kim, Seunghoon Woo, Heejo Lee

Korea University | IFIP SEC 2026 | June 9, 2026

Computer & Communication Security Lab

MOTIVATION

Why Affected Versions Matter

Security response depends on one question:

Is my software using a vulnerable OSS version?

NVD CPE ranges are commonly treated as ground truth.

PROBLEM

When Version Metadata Is Wrong

Two error modes create very different risks.

False safe

A vulnerable version is reported as fixed or unaffected.

Impact: systems remain exposed.

False vulnerable

A safe version is reported as vulnerable.

Impact: unnecessary updates, noisy research results.

MOTIVATING EXAMPLE

A Concrete Failure: PoC Results Disagree With Metadata

CVE-2018-19210: CPE metadata implies 4.0.10 is safe, but reproduction shows otherwise.

LibTIFF 4.0.9

```
$ checkout v4.0.9
$ ./run_poc cve-2018-19210.tiff
Segmentation fault (core dumped)
=> vulnerable
```

LibTIFF 4.0.10

```
$ checkout v4.0.10
$ ./run_poc cve-2018-19210.tiff
Segmentation fault (core dumped)
=> still vulnerable
```

LibTIFF 4.1.0

```
$ checkout v4.1.0
$ ./run_poc cve-2018-19210.tiff
completed without crash
=> PoC not reproduced
```

false-safe version

Updating only to the metadata-safe version can leave the system exposed.

MOTIVATING EXAMPLE

The Missing Fix In Code

The metadata-safe version lacks the added check.

LibTIFF 4.0.10 (metadata-safe)

```
1 static int _TIFFVSetField(...) {
2     ...
3     td->td_smaxsamplevalue = NULL;
4 }
5
6 td->td_samplesperpixel = (uint16) v;
7 }
```

***transferfunction* guard absent**

Patch context / later fixed version

```
1 static int _TIFFVSetField(...) {
2     ...
3     td->td_smaxsamplevalue = NULL;
4 }
5 + if (td->td_transferfunction[0] != NULL &&
6 +     (v - td->td_extrasamples > 1) &&
7 +     !(samplesperpixel - extrasamples > 1)) { ... }
8 td->td_samplesperpixel = (uint16) v;
```

Affected-version validation needs code-level evidence, not only public metadata.

CHALLENGE

Why Existing Code-Level Ideas Fall Short

- Deleted-line assumptions fail when a patch only adds code.
- Syntax-only matching breaks under code drift and refactoring.
- Backports and parallel branches make version ranges discontinuous.

CLOVERY

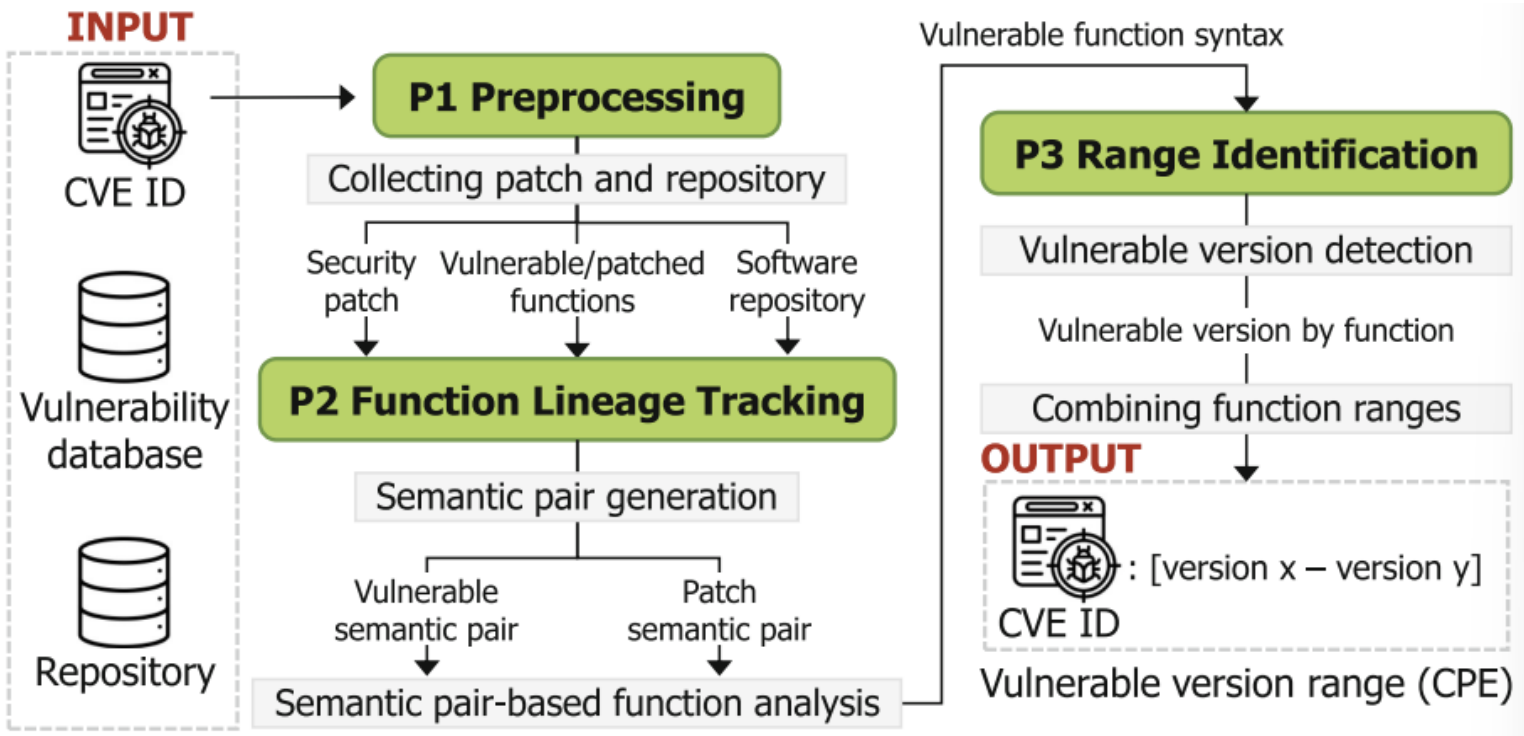
Code cLOne based vulnerability-affected VErsion discovery

: Identifying affected versions in C/C++ public security vulnerability reports via function-lineage tracking

METHOD

CLOVERY Overview

From a CVE and repository history, CLOVERY outputs a CVE-level affected-version range.



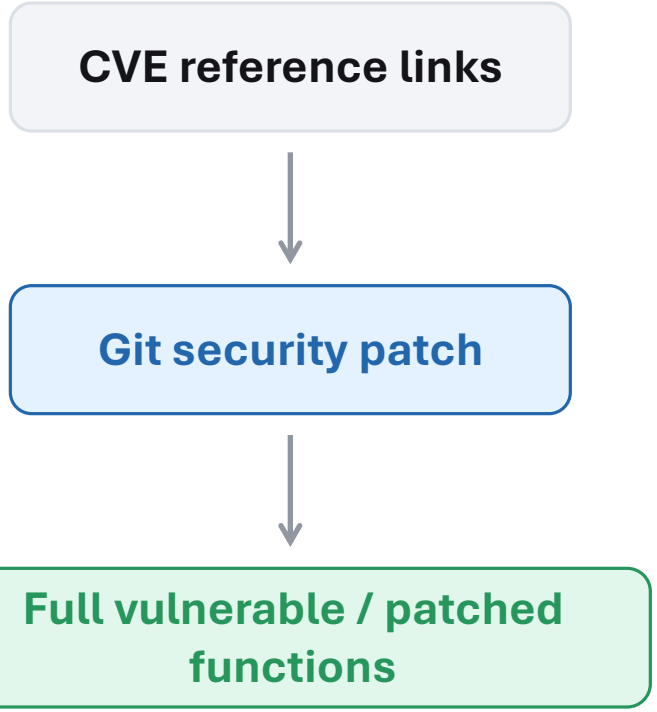
CLOVERY workflow: preprocess, track function lineage, identify ranges

P2 is the core: semantic pair-based function-lineage tracking

METHOD

P1: Recover Patch Context

URL	Source(s)	Tag(s)
http://www.libtiff.org/	VulDB	Product
https://gitlab.com/libtiff/libtiff/-/commit/e8c9d6c616b19438695fd829e58ae4fde5bfbc22	VulDB	Patch
https://gitlab.com/libtiff/libtiff/-/issues/715	CISA-ADP, VulDB	Exploit, Issue Tracking, Vendor Advisory
https://gitlab.com/libtiff/libtiff/-/merge_requests/737	VulDB	Product
https://vuldb.com/?ctiid.317591	VulDB	Permissions Required, VDB Entry
https://vuldb.com/?id.317591	VulDB	Third Party Advisory, VDB Entry
https://vuldb.com/?submit.621797	CISA-ADP, VulDB	Third Party Advisory, VDB Entry



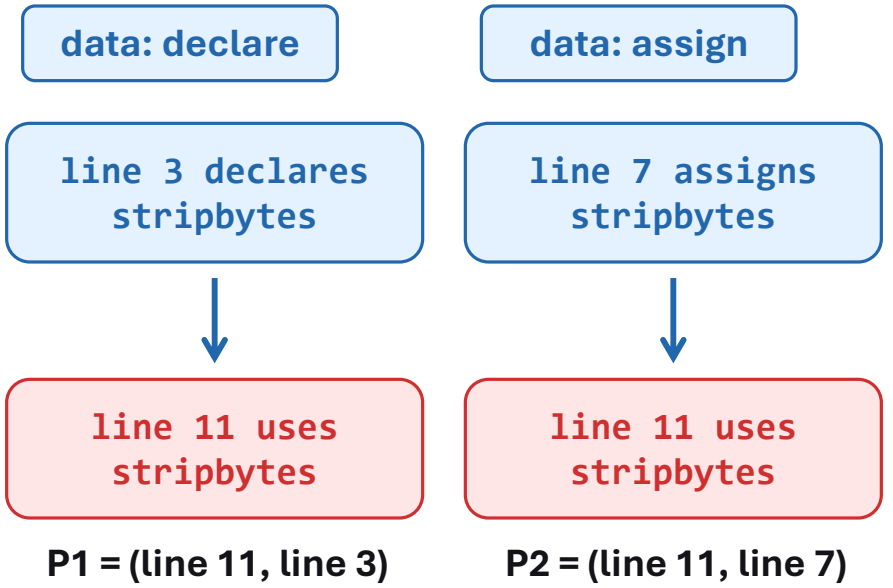
Why full functions? Dependent lines often sit outside the changed lines.

METHOD

P2: The Key Idea Is Semantic Pairs

Vulnerable function context

```
1  static void ChopUpSingleUncompressedStrip(TIFF* tif) {
2      ...
3      uint64 stripbytes;
4      uint64 rowblockbytes;
5      ...
6      if (rowblockbytes > STRIP_SIZE_DEFAULT) {
7          stripbytes = rowblockbytes;
8      }
9      if (rowsperstrip >= td->td_rowsperstrip) {
10         return;
11 -    nstrips64 = TIFFhowmany_64(bytecount, stripbytes);
12     ...
```

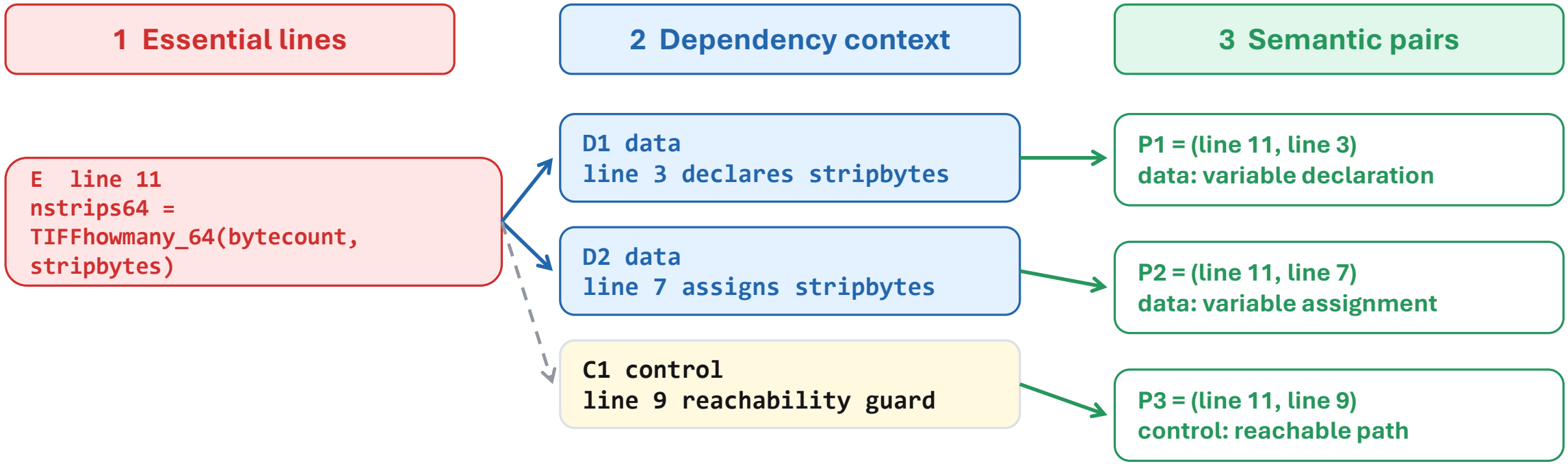


CLOVERY compares dependency-aware pairs, not isolated raw lines.

METHOD

Semantic Pair Generation: Step By Step

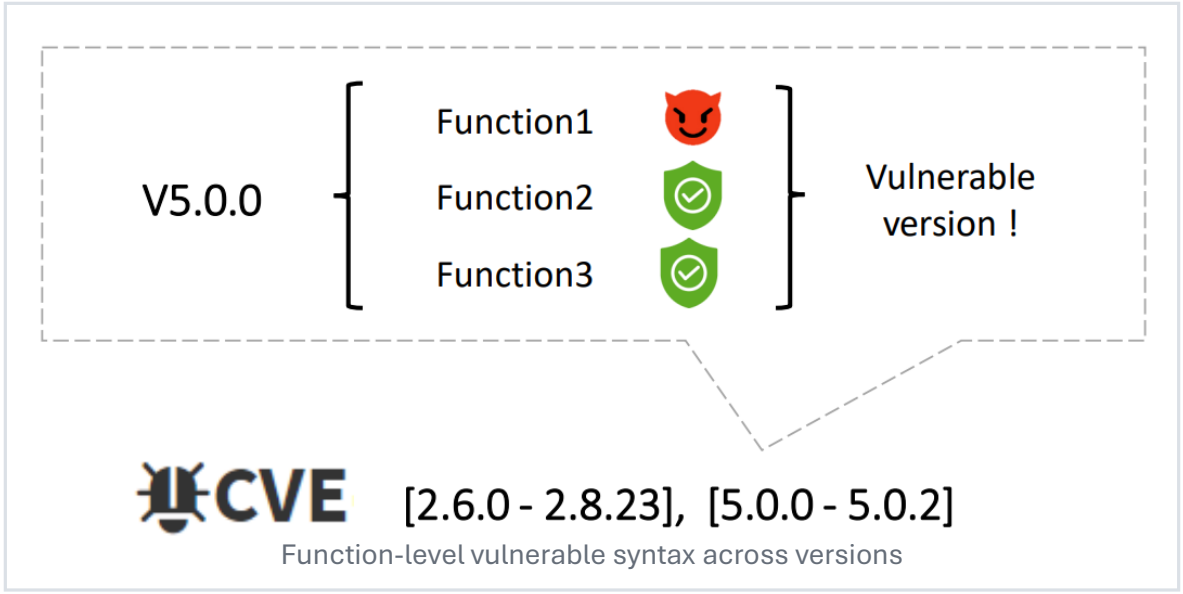
CLOVERY turns changed lines and dependency context into comparable function semantics.



This is the matching unit used later when classifying vulnerable function syntax.

METHOD

Tracking Function Lineage



1. File history

2. Target function history

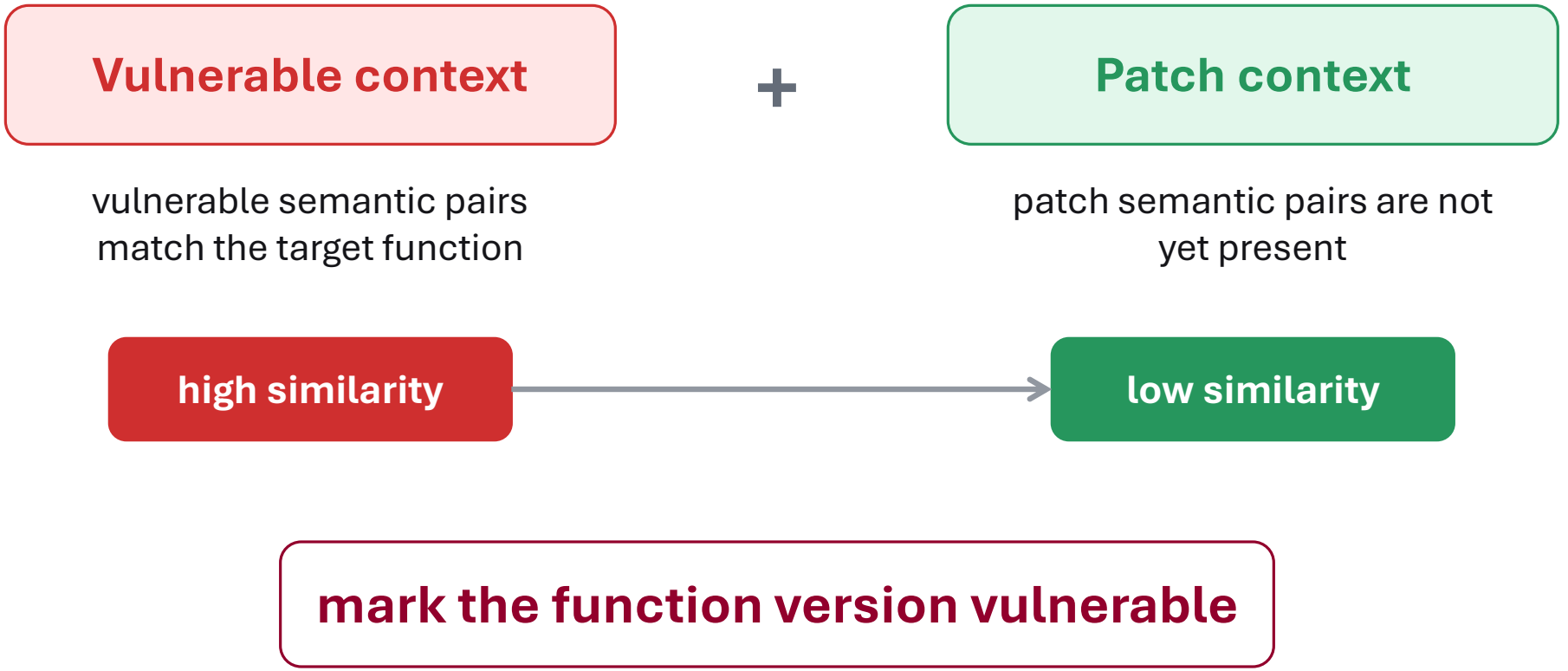
3. Function body per version

CLOVERY tracks vulnerable and patched function syntax through history.

METHOD

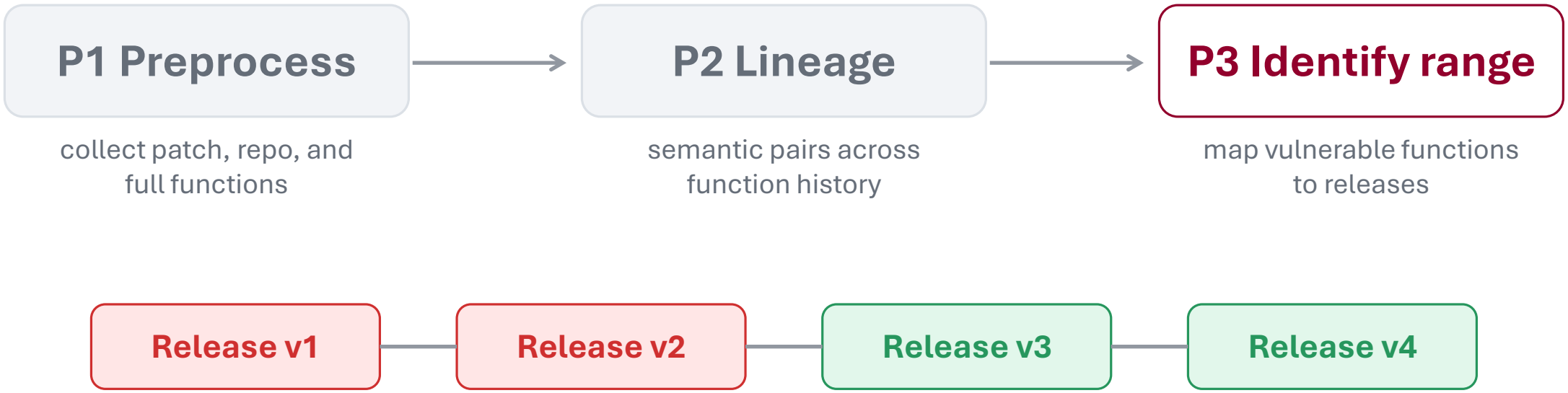
Classifying Vulnerable Function Syntax

Decision rule: vulnerable context present, patch context absent.



METHOD

P3: From Functions To Affected Versions



If any target function in a release is vulnerable, mark that software version affected.

EVALUATION

Dataset And Ground-Truth Validation

4,060

C/C++ security patches from NVD references

→

2,176

after filtering projects without version tags

→

1,502

analyzed CVEs after CPE-repository alignment

Validation methods

Reproduction
103 CVEs

Author sites
54 CVEs

Code review
1,345 CVEs

EVALUATION

Accuracy Across Validation Methods

CLOVERY has the best total **F1** and the strongest recall across 1,502 CVEs.

Validation method	#CVEs	NVD			VOFinder[24]		
		Precision	Recall	F1 score	Precision	Recall	F1 score
Reproduction	103	0.8627	0.8712	0.8669	**0.9973**	0.6645	0.7976
Author sites	54	0.8913	0.8612	**0.8759**	0.9129	0.5158	0.6592
Code review	1,345	0.8126	0.8566	0.8340	0.9450	0.5647	0.7070
Total	1,502	0.8189	0.8577	0.8379	0.9474	0.5698	0.7116

Validation method	#CVEs	SEM-SZZ[20]			Clovery		
		Precision	Recall	F1 score	Precision	Recall	F1 score
Reproduction	103	0.9421	0.8574	0.8978	0.8837	**0.9759**	**0.9275**
Author sites	54	**0.9570**	0.5108	0.6661	0.7302	**0.9245**	0.8160
Code review	1,345	0.9460	0.7644	0.8456	**0.9781**	**0.9894**	**0.9837**
Total	1,502	0.9484	0.7576	0.8423	**0.9627**	**0.9862**	**0.9743**

96.27% precision

98.62% recall

The recall gain matters because false-safe versions are the dangerous failures.

EVALUATION

Why CLOVERY Improves Accuracy

Semantic pairs preserve security-relevant relationships across code history.

Added-line patches

Fixes may add guards without deleting the vulnerable line.

Syntax drift

Older functions can express the same vulnerable logic differently.

Parallel branches

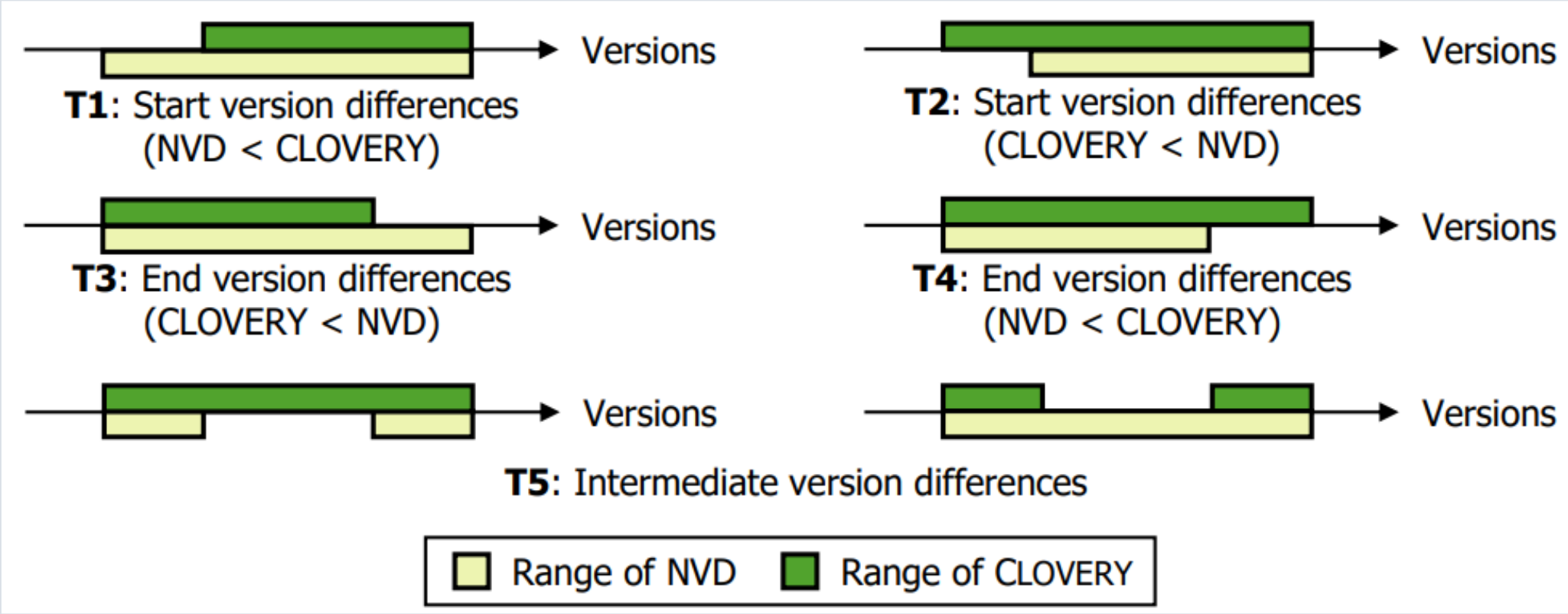
Backports and release lines create discontinuous affected ranges.

Runtime on 1,502 CVEs	**avg 295s**	**median 72s**	**60% <= 100s**

Practical for offline affected-range validation.

Five Ways Version Ranges Differ

NVD and code evidence can disagree at the start, end, or inside the affected range.



Range of NVD vs. range validated by CLOVERY

FINDINGS

65.65% Of NVD Ranges Differ

65.65%

986 / 1,502 CVEs

where CLOVERY is correct but NVD CPE metadata differs

Start differences

End differences

Intermediate gaps

986 cases become concrete NVD correction targets.

Distribution Of Range Differences

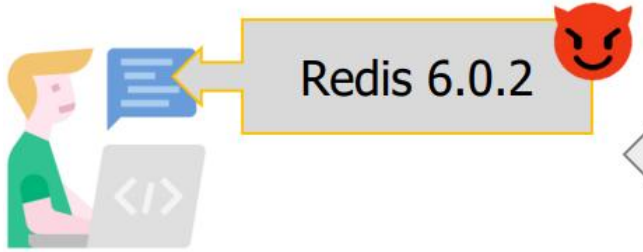
Table 2: categories for 986 corrected NVD discrepancies.

Type	Description	#CVEs	Ratio
T1	**NVD starts earlier**	**578**	**58.62%**
T2	CLOVERY starts earlier	325	32.96%
T3	**CLOVERY ends earlier**	**194**	**19.68%**
T4	**NVD ends earlier**	**357**	**36.21%**
T5	**Intermediate gap**	**184**	**18.66%**

Duplicates are allowed per CVE; T4 is the dangerous false-safe end case.

Case Study: Redis Shows Discontinuous Ranges

CVE-2020-14147, triggered by Lua reuse



compare

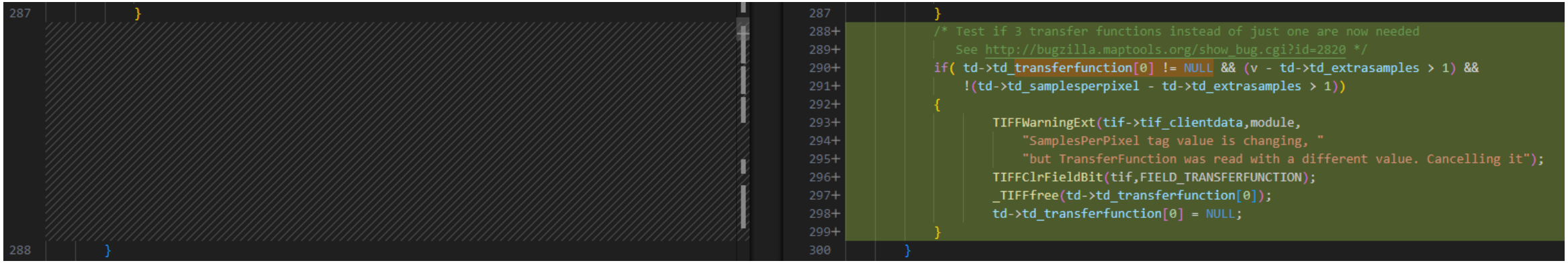
Validated / CLOVERY

- [2.6.0 - 2.8.23]
- [3.0.0 - 3.0.5]
- [3.2.12 - 3.2.13]
- [4.0.10 - 4.0.14]
- [5.0.0 - 5.0.7]

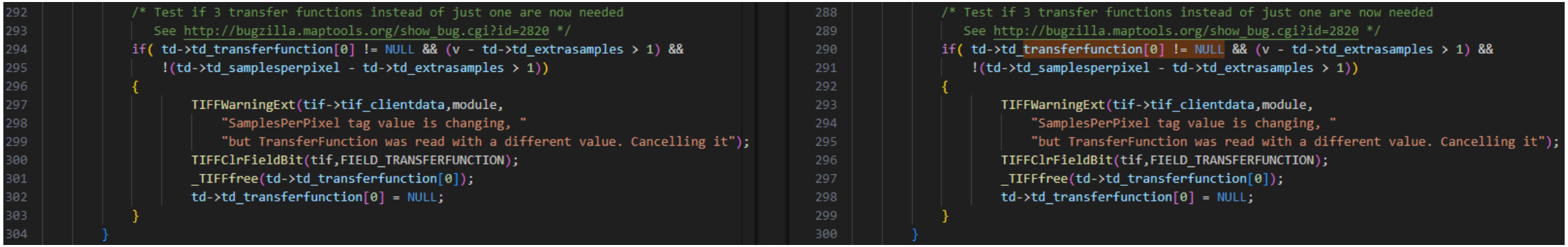
Discontinuous ranges come from reused components and backports.

Finding: Missed Newer Vulnerable Versions Are Actionable

T4 errors are false-safe cases: NVD ends the affected range too early.



4.0.10 vs Patch snippet

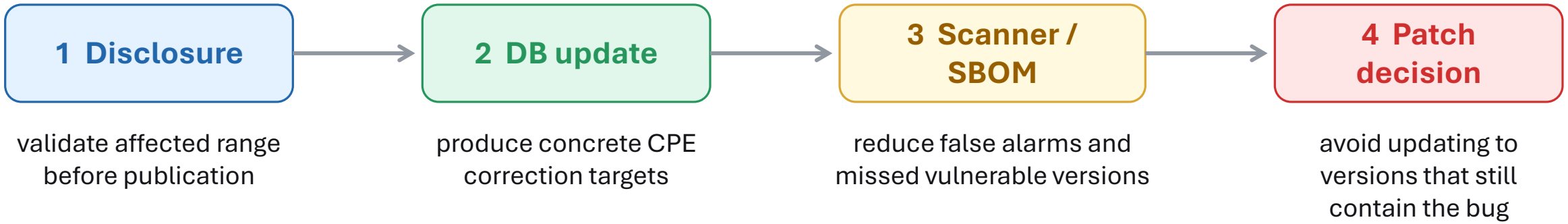


4.1.0 vs Patch snippet

CONCLUSION

How CLOVERY Can Be Used

CLOVERY turns code evidence into metadata that downstream security workflows can use.



Use case examples

NVD correction requests | VEX/SBOM validation | safer partial updates

Limitations

- Requires a source-level security patch and corresponding repository history.
- Does not solve CPE-to-repository name mismatches.
- Can miss cases where vulnerable code differs significantly from the patch context.
- Line-reordering-only patches are rare (<0.05%) but can cause false alarms.

Implemented for C/C++; the methodology is not language-specific.

Conclusion

1. Affected-version information in public reports is often inaccurate.
2. We found potential CPE inconsistencies in 65.65% of CVEs
3. Developers will be better equipped to respond to vulnerabilities, leading to improved supply chain security

Thank you for your attention.

Q&A

Duyeong Kim

Korea University
Computer & Communication Security Lab

duyeong@korea.ac.kr