CE 815 – Secure Software Systems

ML-Based Vulnerability Detection Methods (GraphSPD)

Mohammad Haddadian/Mehdi Kharrazi
Department of Computer Engineering
Sharif University of Technology

Acknowledgments: Some of the slides are fully or partially obtained from other sources. A reference is noted on the bottom of each slide, when the content is fully obtained from another source. Otherwise a full list of references is provided on the last slide. Thanks to Mohammad Haddadian for the help on the slides.

Introduction

- As announced by the 2021 report, 98% of codebases contain open source components
- Meanwhile, 84% of code-bases have at least one open-source vulnerability
- 60% of them contain high-risk vulnerabilities

- By exploiting the OSS vulnerabilities reported in the vulnerability databases (e.g., NVD), attackers can perform "N-day" attacks against unpatched software systems

Problem

- A large volume of OSS security patches (e.g., GitHub commits fixing vulnerabilities) are silently released.

```
From 7f9822a48213dd2feca845dbbb6bcb8beb9550de
Subject: [PATCH] Add blinding to a DSA signature

This is based on side channel attacks demonstrated by (NCC Group)
for ECDSA which are likely to be able to be applied to DSA.
```

Not report to NVD

```
From 41bdc78544b8a93a9c6814b8bbbfef966272abbe
Subject: [PATCH] x86/tls: Validate TLS entries to protect espfix

Installing a 16-bit RW data segment into the GDT defeats espfix.
AFAICT this will not affect glibc, Wine, or dosemu at all.
```

Not provide explicit description

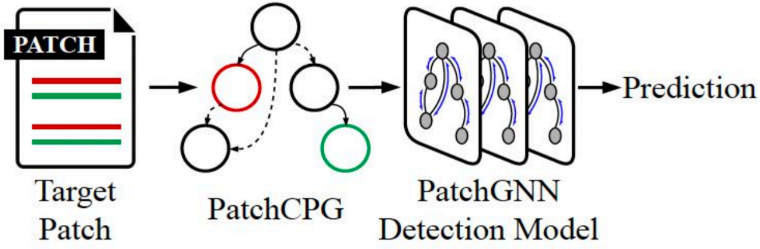
- Average users need to timely detect and apply security patches before being exploited by armored attackers.

Previous Solutions and Limitations

An OSS Patch

Natural Language Documentation (Commit Message/Changelog)	Source Code Changes
• **Mining security keywords** ❌ Requiring well-maintained doc.	• **Regarding code as sequential data** ❌ Losing important semantics. • **Our solution: representing code as graph** ✅ Retaining rich patch structural info.

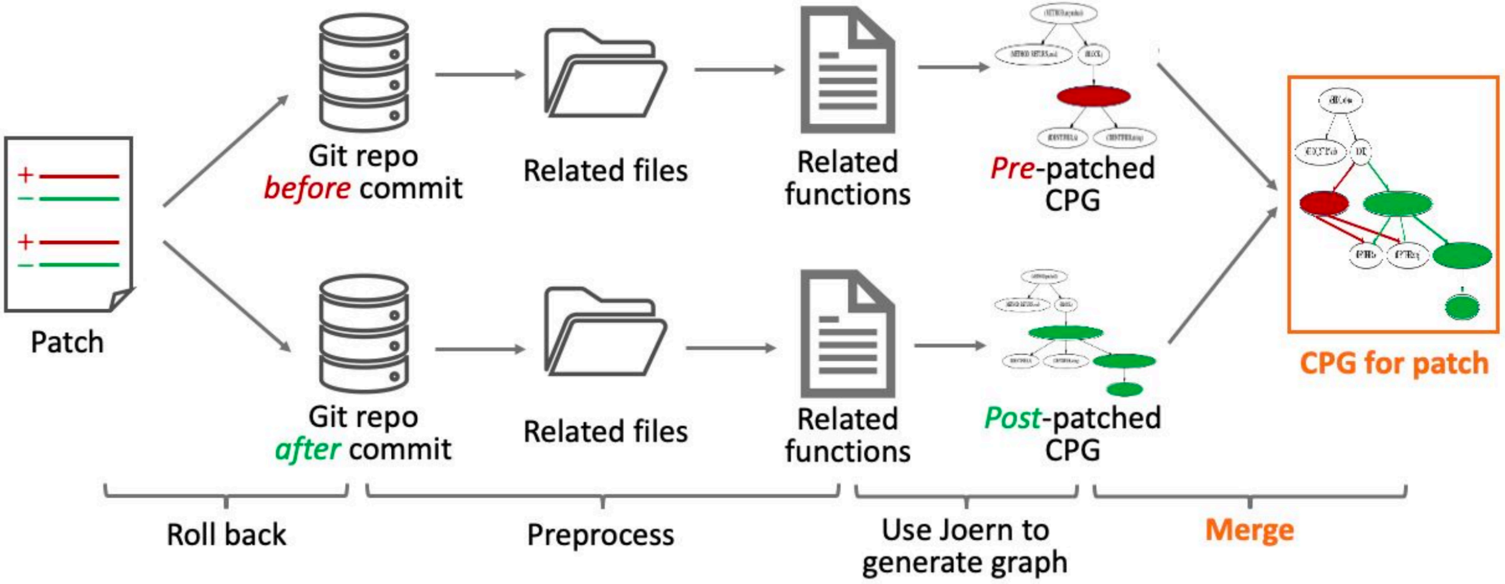
A Graph-Based Security Patch Detection System



- PatchCPG: a new graph representation of inherent code change structures.
  - Syntax and semantics: AST + control & data dependency graph.
  - Changes and relations with context: pre-patch + post-patch graph.
- PatchGNN: a tailored GNN model to capture diverse patch structural information

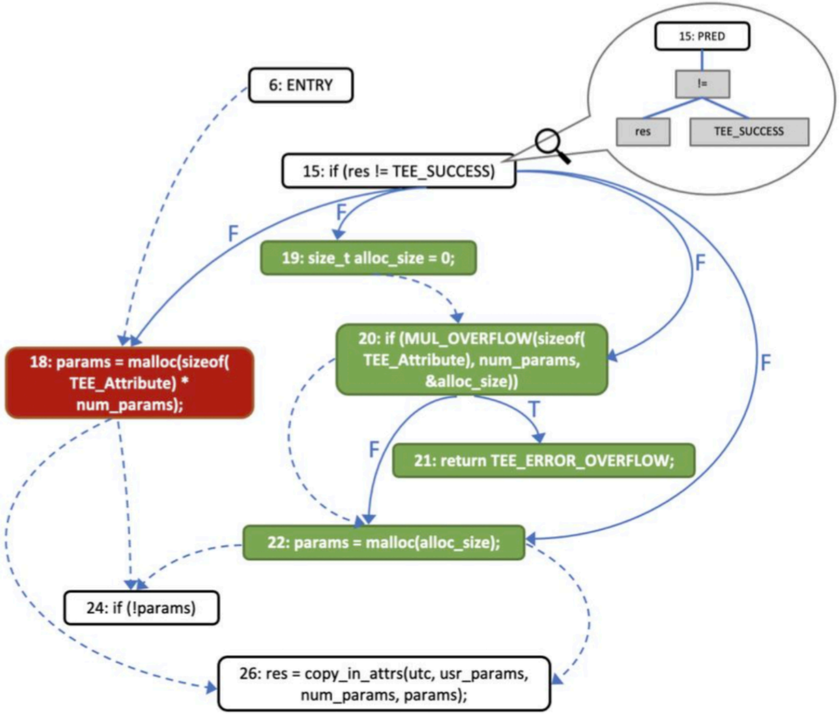
PatchCPG: From Patch to Graph

- Challenge: how to construct PatchCPG?



Patch Code Property Graph (PatchCPG)

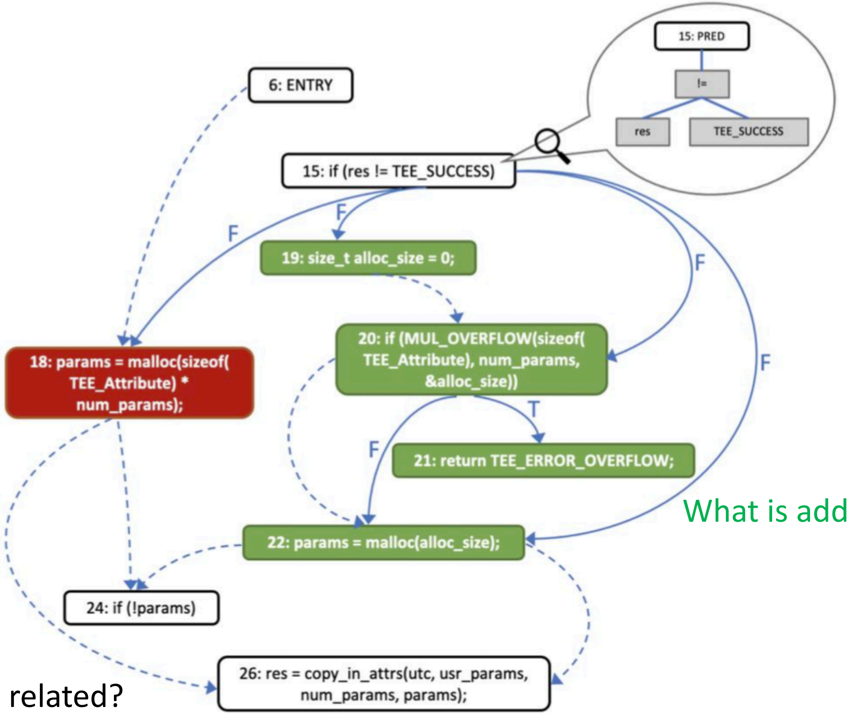
- A joint graph encodes rich patch structural information.



Patch Code Property Graph (PatchCPG)

- A joint graph encodes rich patch structural information.

What is deleted?

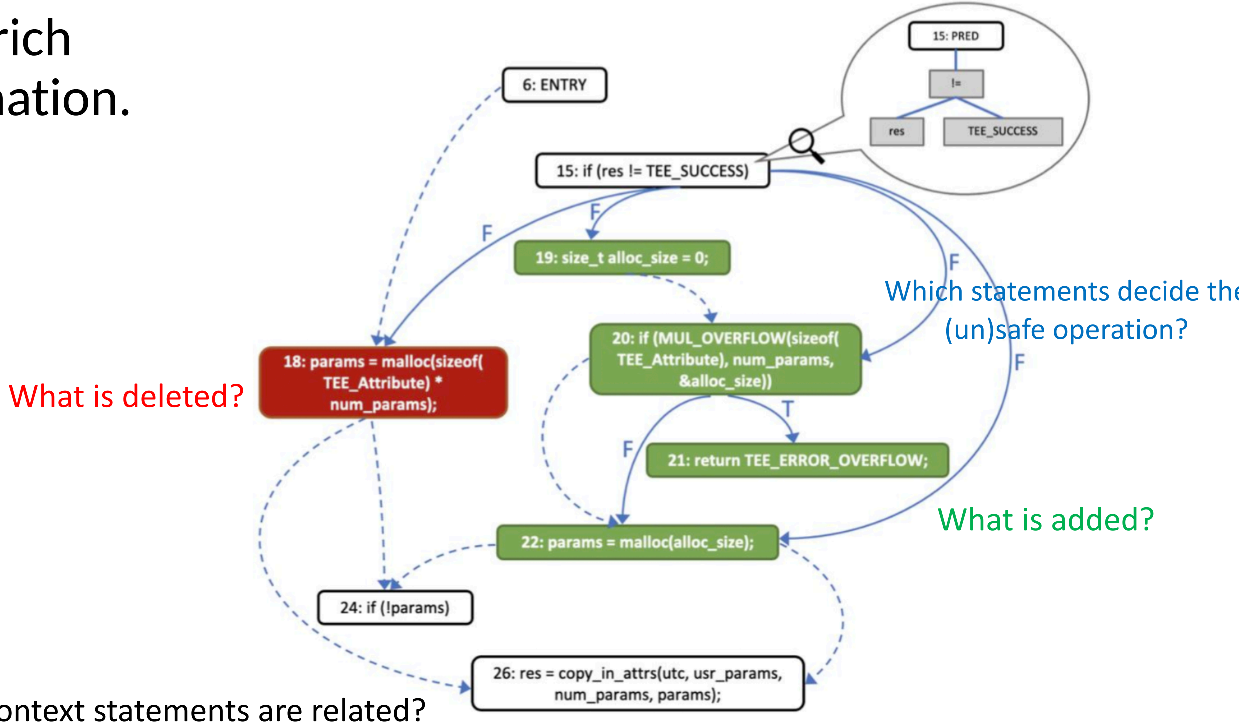


What is added?

What context statements are related?

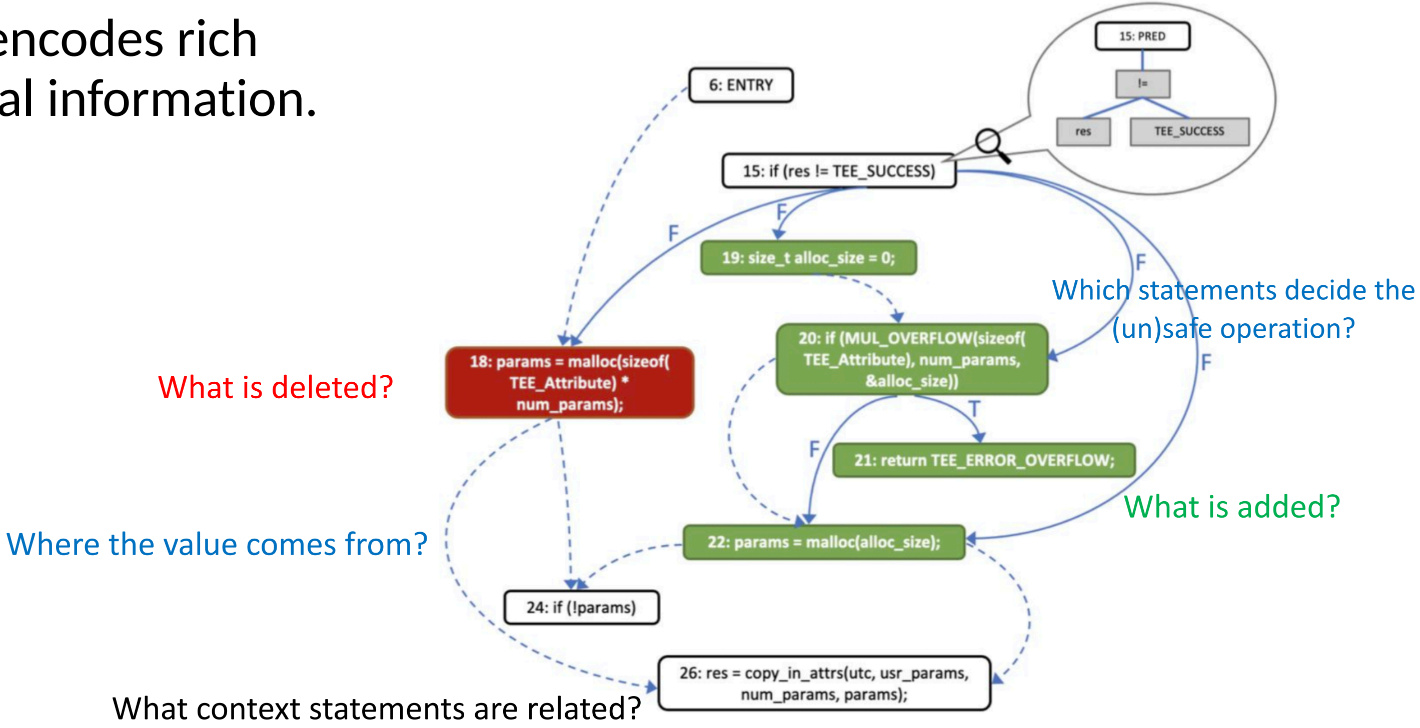
Patch Code Property Graph (PatchCPG)

- A joint graph encodes rich patch structural information.



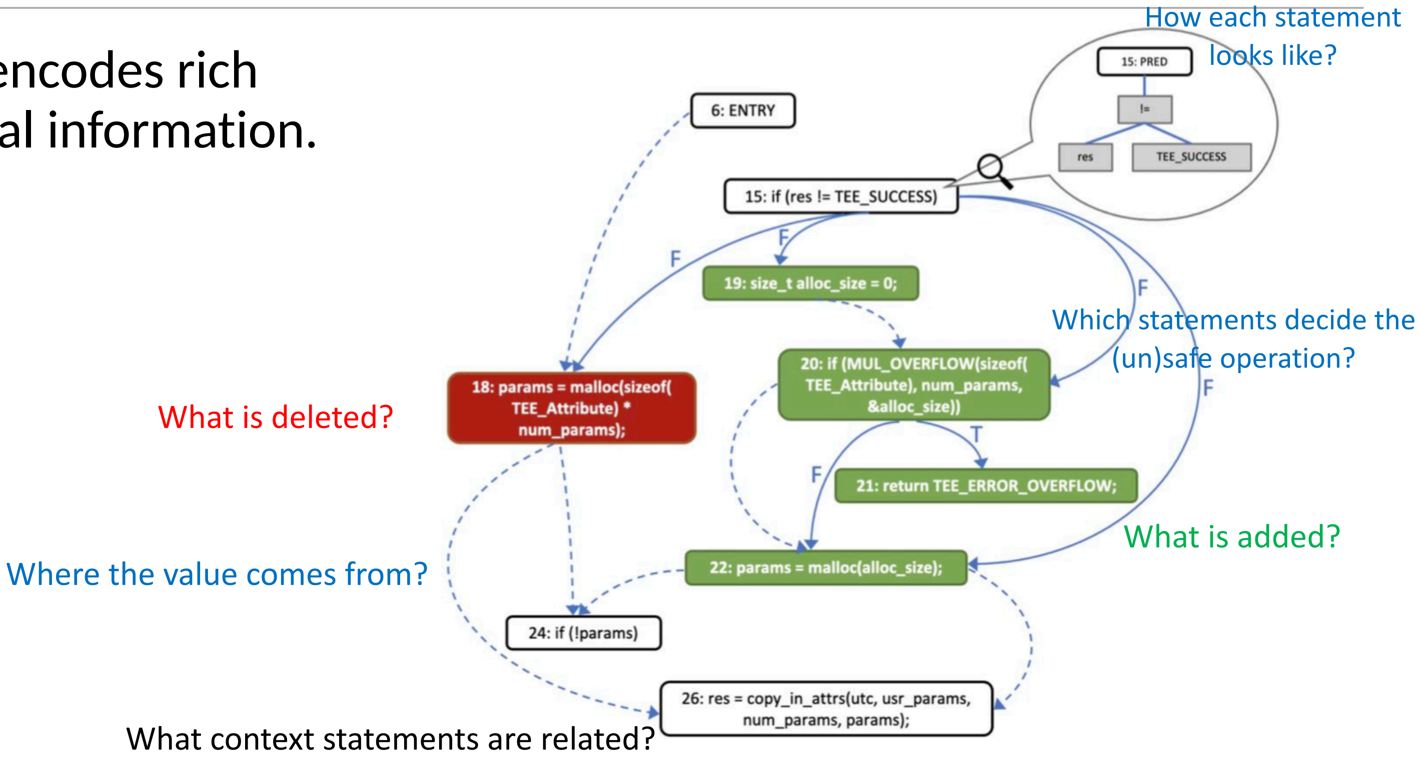
Patch Code Property Graph (PatchCPG)

- A joint graph encodes rich patch structural information.



Patch Code Property Graph (PatchCPG)

- A joint graph encodes rich patch structural information.



Reducing Noisy Information by Slicing

```
TEE_Result syscall_asymm_verify(unsigned long state,
    const struct utee_attribute *usr_params, size_t
    num_params, const void *data, size_t data_len,
    const void *sig, size_t sig_len)
{
    TEE_Result res;
    TEE_Attribute *params = NULL;
    struct user_ta_ctx *utc;

    ...

    res = tee_mmu_check_access_rights(utc,
     TEE_MEMORY_ACCESS_READ |
     TEE_MEMORY_ACCESS_ANY_OWNER, (uaddr_t)sig, sig_len)
     ;
    if (res != TEE_SUCCESS)
        return res;

-   params = malloc(sizeof(TEE_Attribute) * num_params);
+   size_t alloc_size = 0;
+   if (MUL_OVERFLOW(sizeof(TEE_Attribute), num_params,
     &alloc_size))
+       return TEE_ERROR_OVERFLOW;
+   params = malloc(alloc_size);

    if (!params)
        return TEE_ERROR_OUT_OF_MEMORY;
    res = copy_in_attrs(utc, usr_params, num_params,
     params);
    if (res != TEE_SUCCESS)
        goto out;

    ...

out:
    free(params);
    return res;
}
```

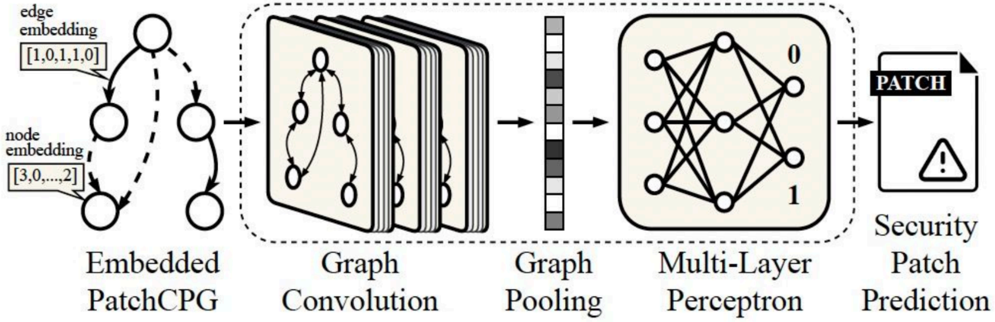
Too many statements

slicing


```
TEE_Result syscall_asymm_verify(unsigned long state,
    const struct utee_attribute *usr_params, size_t
    num_params, const void *data, size_t data_len,
    const void *sig, size_t sig_len)
{
    TEE_Result res;
    TEE_Attribute *params = NULL;
    struct user_ta_ctx *utc;

    ...

    res = tee_mmu_check_access_rights(utc,
     TEE_MEMORY_ACCESS_READ |
     TEE_MEMORY_ACCESS_ANY_OWNER, (uaddr_t)sig, sig_len)
     ;
    if (res != TEE_SUCCESS)
        return res;

-   params = malloc(sizeof(TEE_Attribute) * num_params);
+   size_t alloc_size = 0;
+   if (MUL_OVERFLOW(sizeof(TEE_Attribute), num_params,
     &alloc_size))
+       return TEE_ERROR_OVERFLOW;
+   params = malloc(alloc_size);

    if (!params)
        return TEE_ERROR_OUT_OF_MEMORY;
    res = copy_in_attrs(utc, usr_params, num_params,
     params);
    if (res != TEE_SUCCESS)
        goto out;

    ...

out:
    free(params);
    return res;
}
```


Only retain most relevant contexts

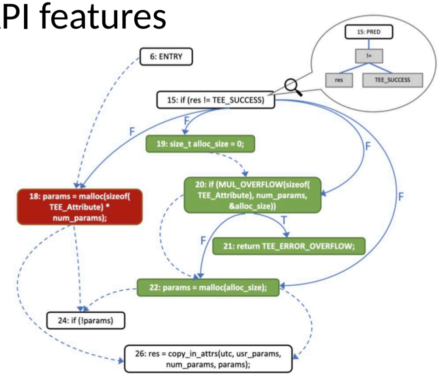
PatchGNN: Detect Security Patches from PatchCPGs



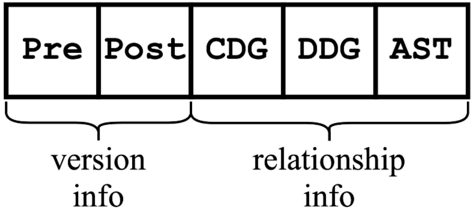
- Challenge 1: how to embed the PatchCPGs?
- Challenge 2: how to learn multiple attributes (CDG/DDG/AST/pre/post)?

PatchCPG Embeddings

- Node Embedding
  - 20-dimensional vulnerability features.
    - code snippet metadata
    - identifier and literal features
    - control flow features
    - operator features
    - API features



- Edge Embedding
  - 5-dimensional binary vector.



e.g., [1,1,0,1,0] means the edge is a context edge of data dependency.

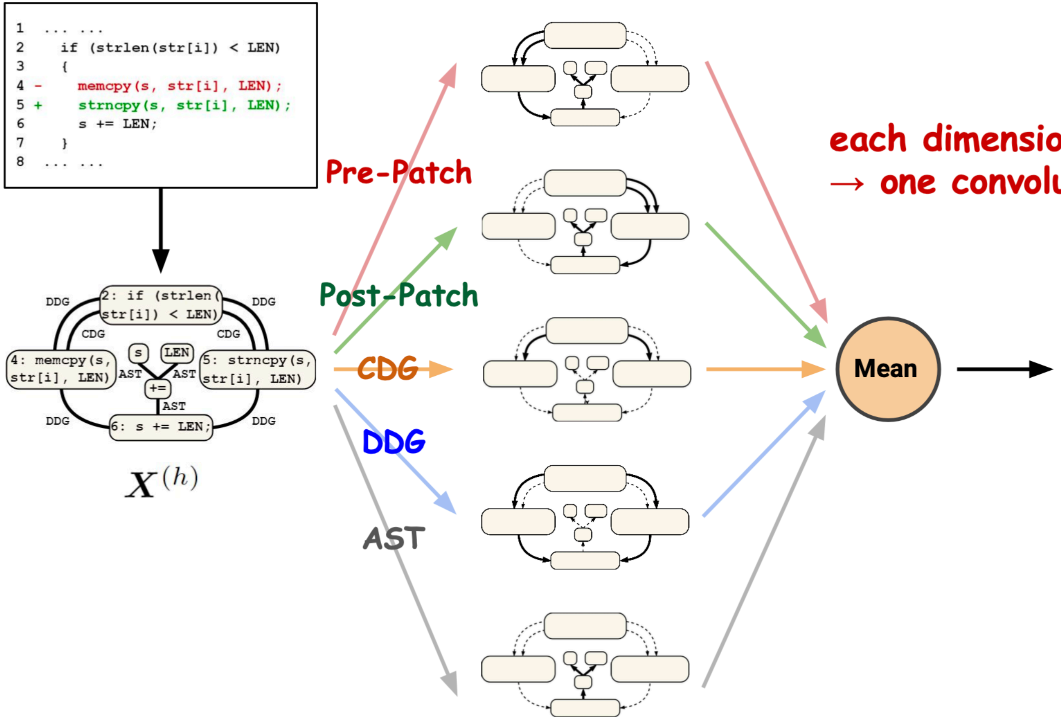
PatchCPG Embeddings

TABLE I: The involved tokens or sub-tokens of the control flow features, the operator features, and the API features.

Features	**Matched Tokens or Sub-tokens**
condition	if, switch
loop	for, while
jump	return, break, continue, goto, throw, assert
arithmetic†	++, --, +, -, *, /, %, =, +=, -=, *=, /=, %=
relational	==, !=, >=, <=, >, <
logical	&&, ||, !, and, or, not
bitwise⋆	&, |, <<, >>, ∼, ∧, bitand, bitor, oxr
memory API	alloc, free, mem, copy, new, open, close, delete, create, release, sizeof, remove, clear, dequene, enquene, detach, attach
string API	str, string
lock API	lock, mutex, spin
system API	init, register, disable, enable, put, get, up, down, inc, dec, add, sub, set, map, stop, start, prepare, suspend, resume, connect,

† Operator * is determined as dereference operator or arithmetic operator.
⋆ Operator & is determined as address-of operator or bitwise operator.

PatchGNN with Multi-Attribute Graph Convolution



each dimension of edge embeddings → one convolution channel

$$X^{(h+1)} = \frac{1}{5}\sum_{k=1}^{5} \sigma(\tilde{L}_k \cdot X^{(h)} \cdot W_k^{(h)})$$

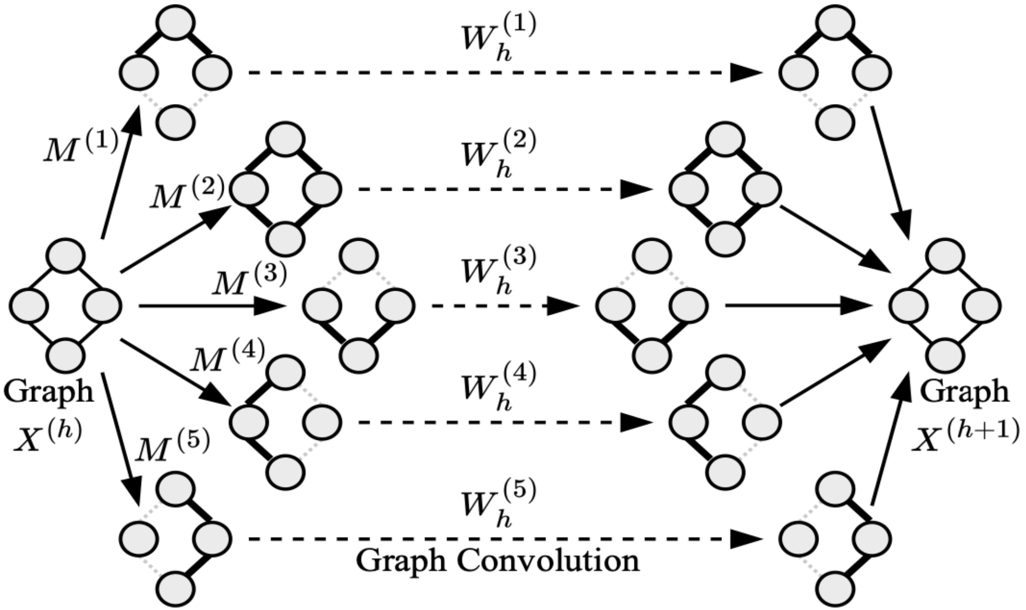
where

$$\tilde{L}_k = \tilde{D}_k^{-1/2} \cdot \tilde{A}_k \cdot \tilde{D}_k^{-1/2}$$

$$\tilde{A}_k = A \odot M^{(k)} + I$$

$$\tilde{D}_k = diag(\textstyle\sum_j A_{\cdot j})$$

PatchGNN with Multi-Attribute Graph Convolution



Implementation & Evaluation

Implementation

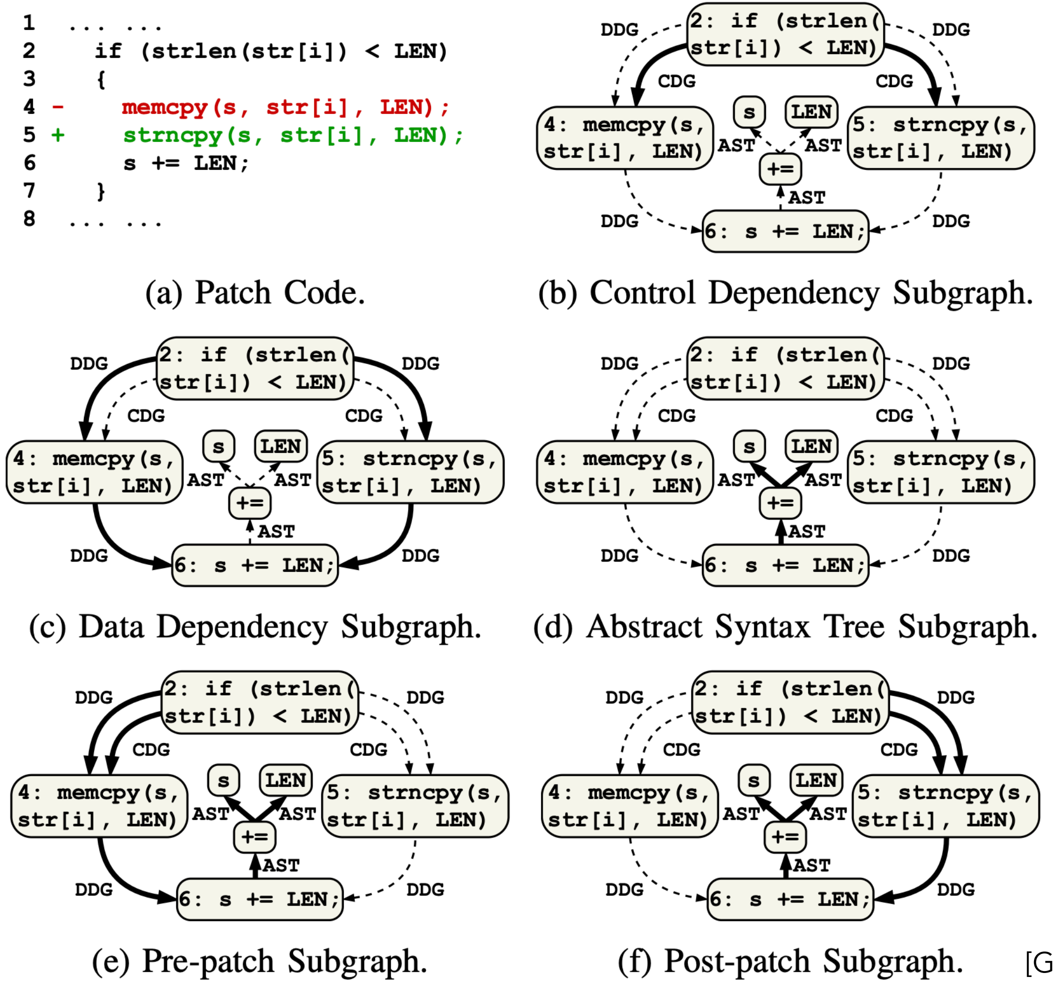
- **5K new LoC** in Scala and Python on top of *Joern* parser and *PyTorch* library.

Datasets:

- PatchDB: 12K security patches from 300+ GitHub repos.
- SPI-DB: 10K security patches from FFmpeg and QEMU.

Evaluation:

- Compared with sequential-based patch detector.
- Compared with vulnerability detection methods.
- Case study on four popular OSS repos.



(a) Patch Code.

(b) Control Dependency Subgraph.

(c) Data Dependency Subgraph.

(d) Abstract Syntax Tree Subgraph.

(e) Pre-patch Subgraph.

(f) Post-patch Subgraph.

[GraphSPD]

Implementation (Demo)

- https://github.com/liquidware/liquidware_beagleboard_linux/commit/176df2457ef6207156ca1a40991c54ca01fef567

```
arch/x86_64/kernel/ptrace.c                                    -4

@@ -232,10 +232,6 @@ static int putreg(struct task_struct *child,
 {
         unsigned long tmp;

-        /* Some code in the 64bit emulation may not be 64bit clean.
-           Don't take any chances. */
-        if (test_tsk_thread_flag(child, TIF_IA32))
-                value &= 0xffffffff;
         switch (regno) {
                 case offsetof(struct user_regs_struct,fs):
                         if (value && (value & 3) != 3)
```

Compared with Sequential-based Solution

- Accuracy 10.8%↑
- F-1 score: 0.096↑
- Precision: 28.82%↑
- False Positive Rate: 14.62%↓

Method	Dataset	General Metrics		Special Metrics	
		Accuracy	F1-score	Precision	FP Rate
TwinRNN [1][2]	PatchDB SPI-DB	69.60% 56.37%	0.461 0.512	48.45% 49.07%	19.67% 41.57%
GraphSPD	PatchDB SPI-DB	80.39% 63.04%	0.557 0.503	77.27% 63.96%	5.05% 19.16%

[1] PatchRNN: A Deep Learning-Based System for Security Patch Identification.
[2] SPI: Automated Identification of Security Patches via Commits.

Compared with Vulnerability Detection Solutions

- **2.5 - 50x** detection rate of vulnerability detectors.

Method	# $\text{Vul}_{\text{pre-patch}}$	# $\text{Vul}_{\text{post-patch}}$	# $\text{Patch}_{\text{security}}$	TP Rate
Cppcheck[3]	3	1	2	0.54%
flawfinder[4]	109	108	1	0.27%
ReDeBug[5]	29	29	0	0.00%
VUDDY[6]	22	16	21	5.71%
VulDeePecker[7]	3	0	3	0.82%
GraphSPD	-	-	53	14.40%

[3] Cppcheck. https://cppcheck.sourceforge.io.
[4] flawfinder. https://dwheeler.com/flawfinder/.
[5] Redebug: finding unpatched code clones in entire os distributions.
[6] VUDDY: A scalable approach for vulnerable code clone discovery.
[7] VulDeePecker: A deep learning- based system for vulnerability detection.

Case Study on False Negetives

```
commit 247d30a7dba6684ccce4508424f35fd58465e535
if (!s1->current_frame.data[0]
    ||s->width != s1->width
    ||s->height!= s1->height) {
    if (s != s1)
-       copy_fields(s, s1, golden_frame, current_frame);
+       copy_fields(s, s1, golden_frame, keyframe);
    return -1;
}
```

Listing 4: Security patch for a double free (CVE-2011-3934).

```
commit 360e95d45ac4123255a4c796db96337f332160ad
if (priv->cac_id_len) {
    serial->len=MIN(priv->cac_id_len, SC_MAX_SERIALNR);
-   memcpy(serial->val,priv->cac_id,priv->cac_id_len);
+   memcpy(serial->val,priv->cac_id,serial->len);
    SC_RETURN(card->ctx,SC_DEBUG_NORMAL,SC_SUCCESS);
}
```

Listing 5: A patch with similar patterns (CVE-2018-16393).

Case Study on OSS Repos

- **NGINX**: detect 21 security patches (Precision: 78%).

Changes w/	CVE	Total Commits	Valid Commits	Detected S.P.	Confirmed S.P.	Precision
1.19.x	3	180	127	7	6	86%
1.17.x	3	134	82	4	3	75%
1.15.x	1	203	120	7	4	57%
1.13.x	1	270	157	9	8	89%
Sum.	8	787	486	27	21	78%

- **Xen**: detect 29 security patches (Precision: 55%).
- **OpenSSL**: detect 45 security patches (Precision: 66%).
- **ImageMagick**: detect 6 security patches (Precision: 46.2%).

Slicing Depth

Slicing Iteration No. (N)	General Metrics		Special Metrics	
	Accuracy	F1-score	Precision	FP Rate
0	80.19%	0.555	76.11%	5.42%
1	80.39%	0.557	77.27%	5.05%
2	79.24%	0.531	73.61%	5.87%
∞	76.90%	0.501	64.42%	8.95%

Conclusion

- Silent security patches can be leveraged by attackers to launch N-day attacks.
- GraphSPD presents patches as graphs and identifies security patches with graph learning, achieving higher accuracy and fewer false alarms.
- GraphSPD can be extended to other programming languages.

Discuss

- PatchDB only collects security patches from NVD
- SPI-DB contains security patches only from two repositories
- GraphSPD mainly learns from the existing patterns and may not apply to the unseen ones
- Cannot distinguish specific security patch types:
  - Some patches are too rare to be included in training set
  - Data is imbalanced in security patch datasets. According to the analysis on NVD, 24.6% of vulnerabilities are related to code execution, whereas only 0.1% of vulnerabilities are HTTP response splitting
- Single function support!

Acknowledgments

- [GraphSPD] GraphSPD: Graph-Based Security Patch Detection with Enriched Code Semantics, S. Wang, X. Wang, K. Sun, S. Jajodia, H. Wang, Q. Li, IEEE S&P 2023.