Center for Cybersecurity Analytics and Automation
An NSF Industry/University Cooperative Research Center

Graph-based Security Patch Detection with Enriched Code Semantics

Faculty Members: Kun Sun, Sushil Jajodia

Students: Shu Wang, Xinda Wang

George Mason University

04/12/2022

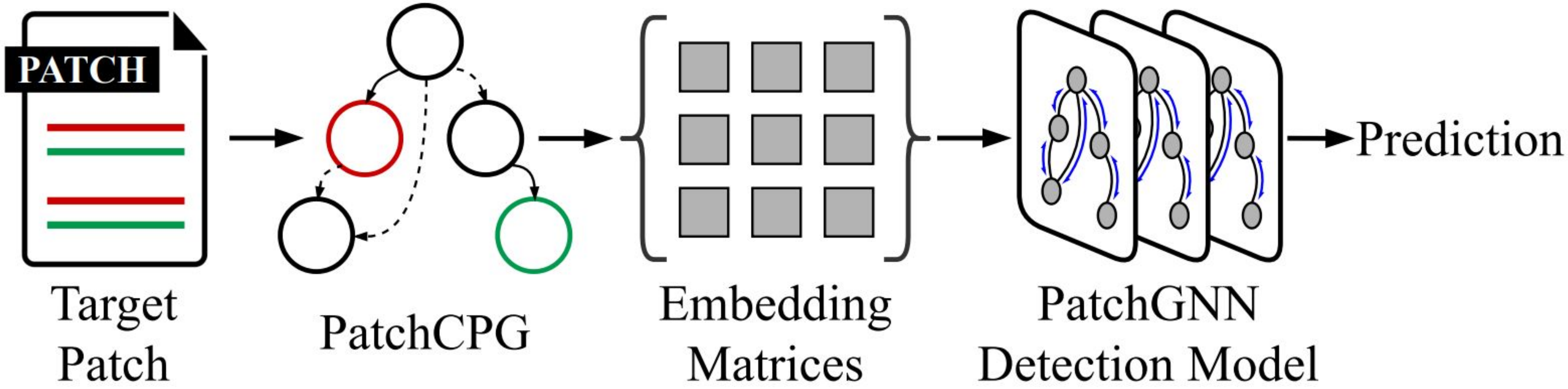
Motivation

- Problem: vendors may silently release security patches

- Limitations of existing solutions:
  - Lack of program semantics.
  - High false-positive rate.

- Our Work: help identify security patches with graphs.
  - Input: GitHub commits.
  - Output: tells if the given commit is a security patch.

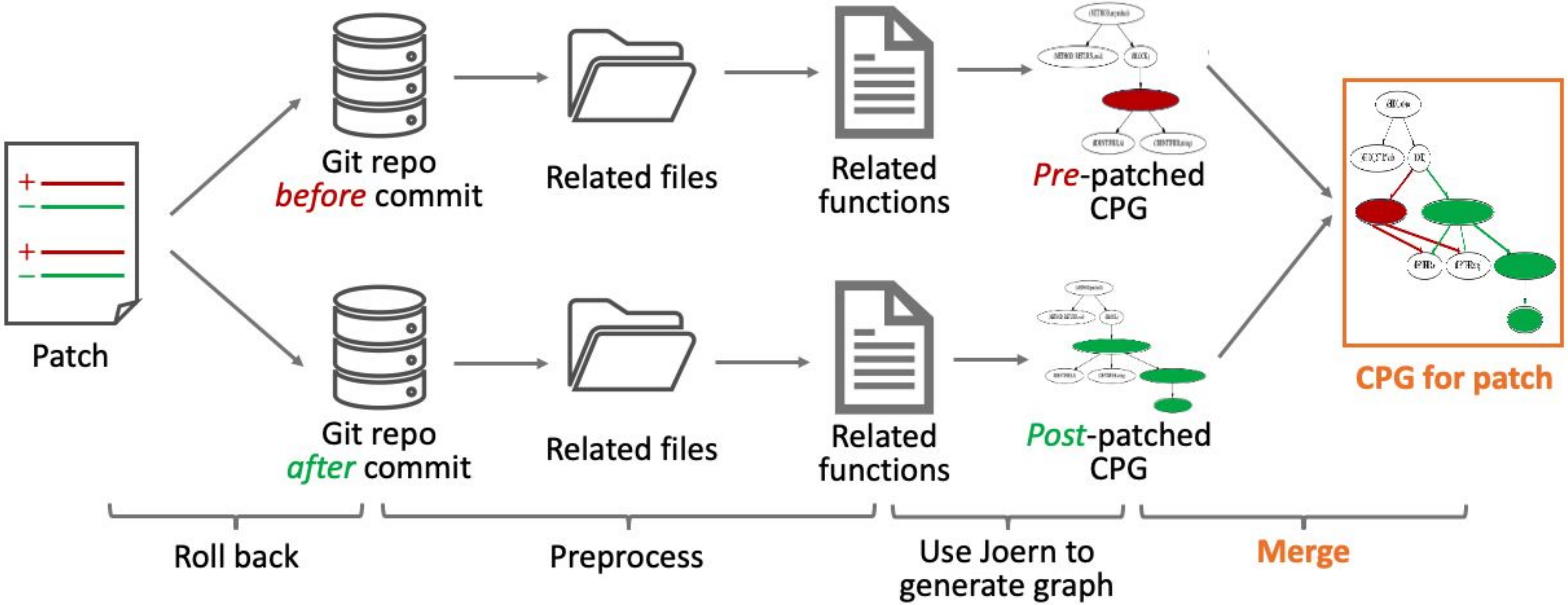
PatchSPD Overview



- Generate PatchCPG for a target patch;
- Embed PatchCPG into a numeric format;
- Detect security patches with Graph Neural Networks.

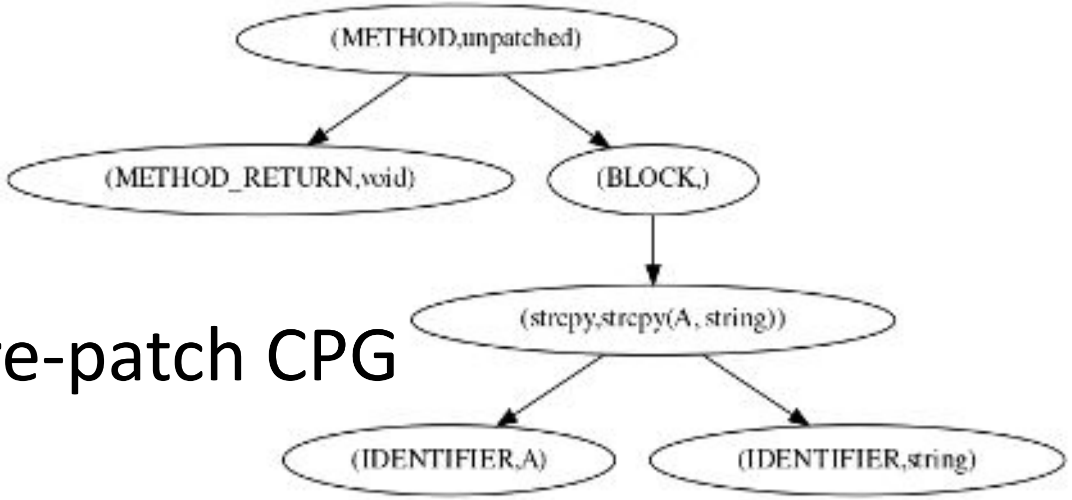
From Patch to Graph

- Challenge: how to construct PatchCPG?



```
void fuction(){
    char A[8] = ""
    unsigned short B = 1979;
    strcpy(A, string);
}
```

Pre-patch function



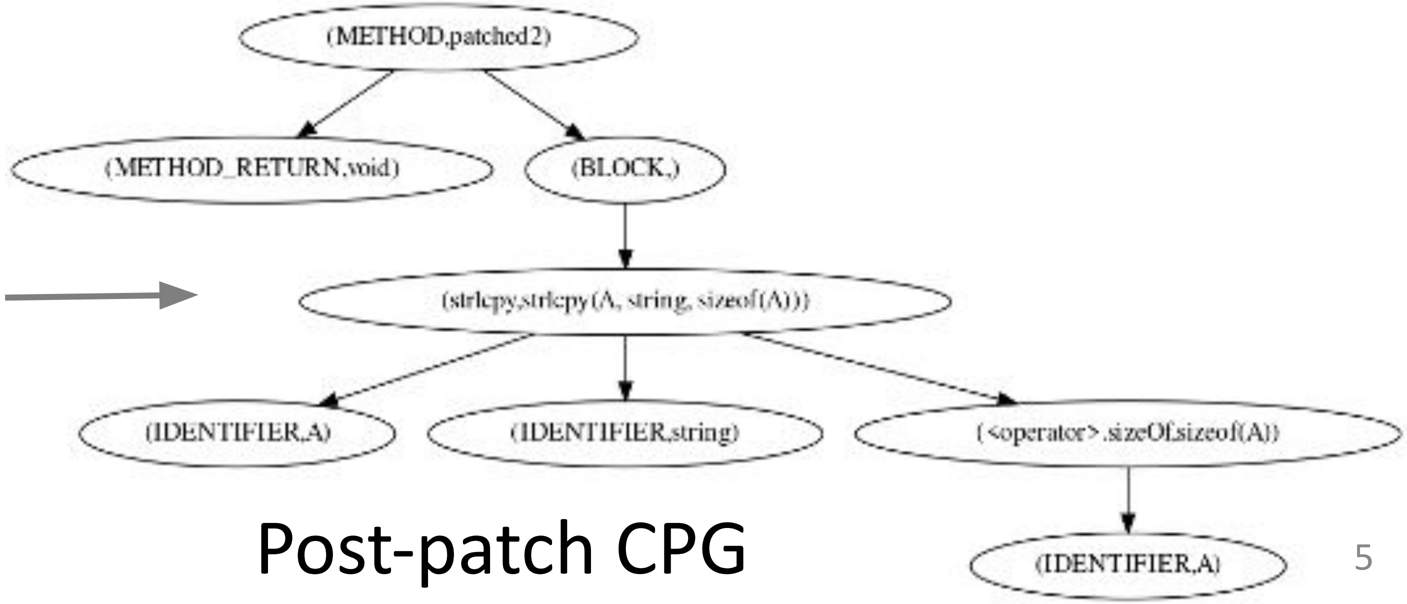
Pre-patch CPG

```
void fuction(){
    char A[8] = ""
    unsigned short B = 1979;
-   strcpy(A, string);
+   strlcpy(A, string, sizeof(A))
}
```

Patch

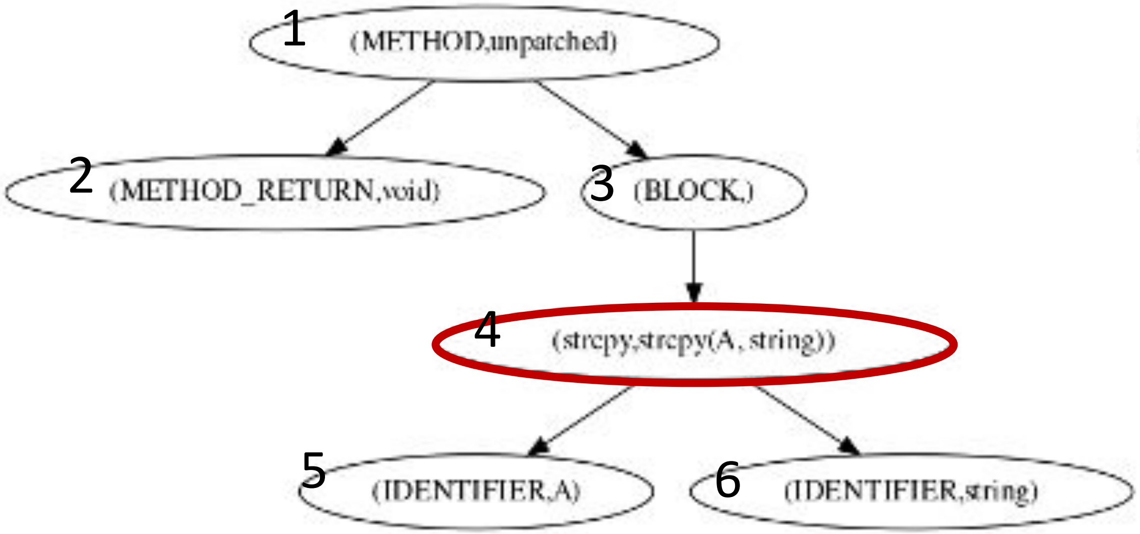
```
void fuction(){
    char A[8] = ""
    unsigned short B = 1979;
    strlcpy(A, string, sizeof(A))
}
```

Post-patch function



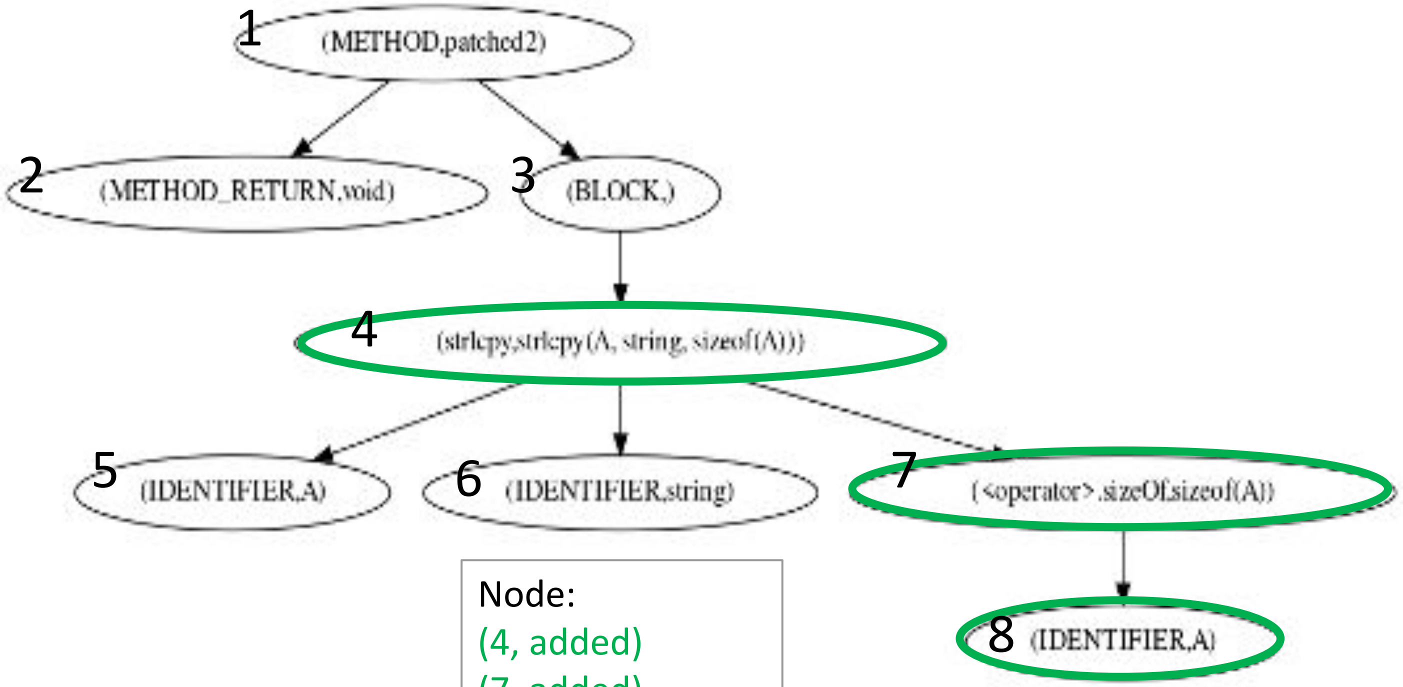
Post-patch CPG

Step1: identify the node types (delete, added, context)



Node:
(4, deleted)
(1, context)
(2, context)
(3, context)
(5, context)
(6, context)

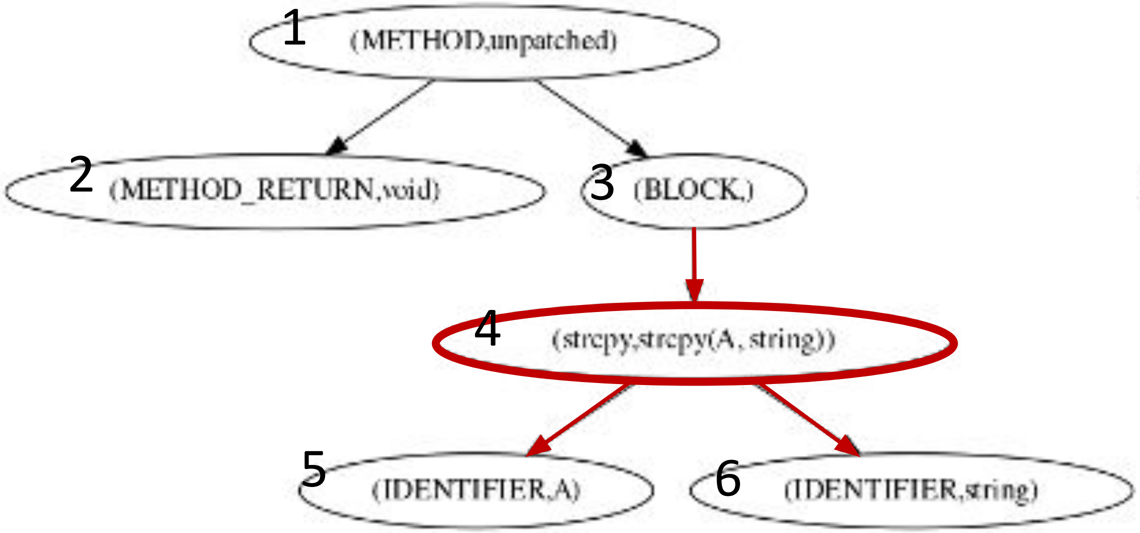
Pre-patched CPG



Node:
(4, added)
(7, added)
(8, added)
(1, context)
(2, context)
(3, context)
(5, context)
(6, context)

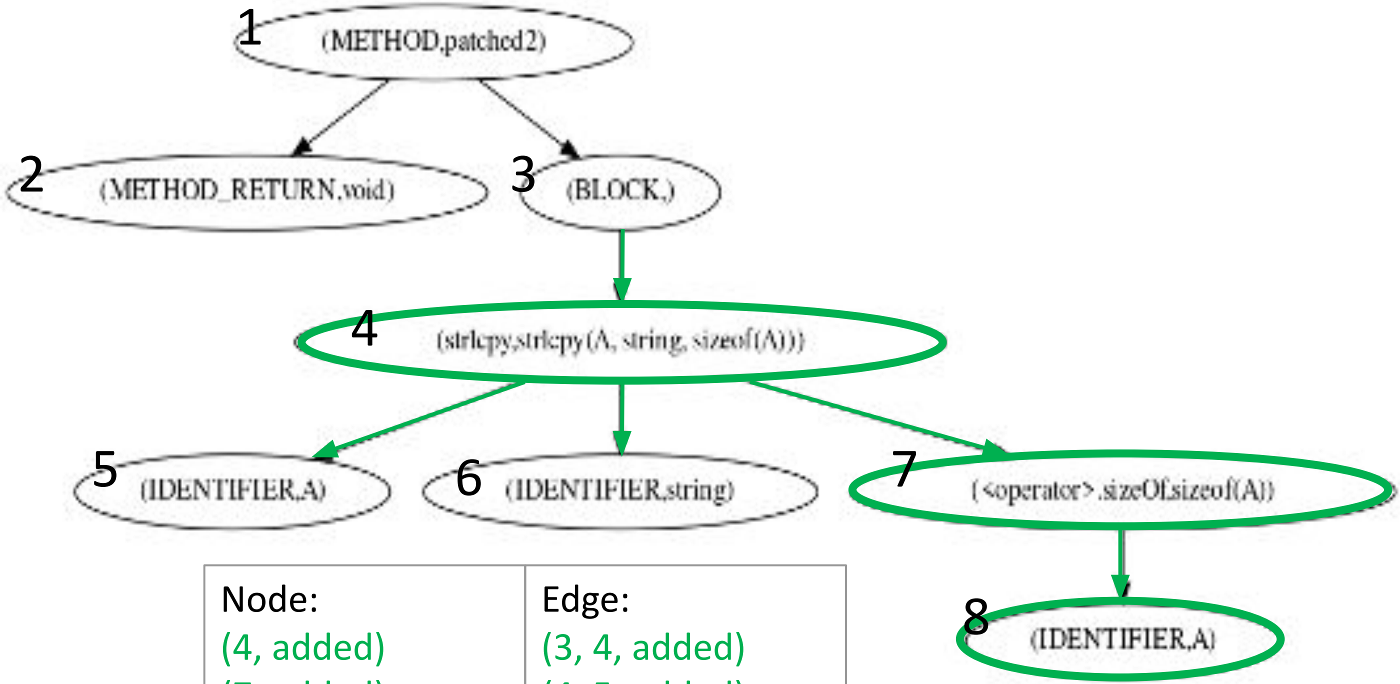
Post-patched CPG

Step 2: identify the edge types (delete, added, context)



Node:	Edge:
(4, deleted)	(3, 4, deleted)
(1, context)	(4, 5, deleted)
(2, context)	(4, 6, deleted)
(3, context)	(1, 2, context)
(5, context)	(1, 3, context)
(6, context)	

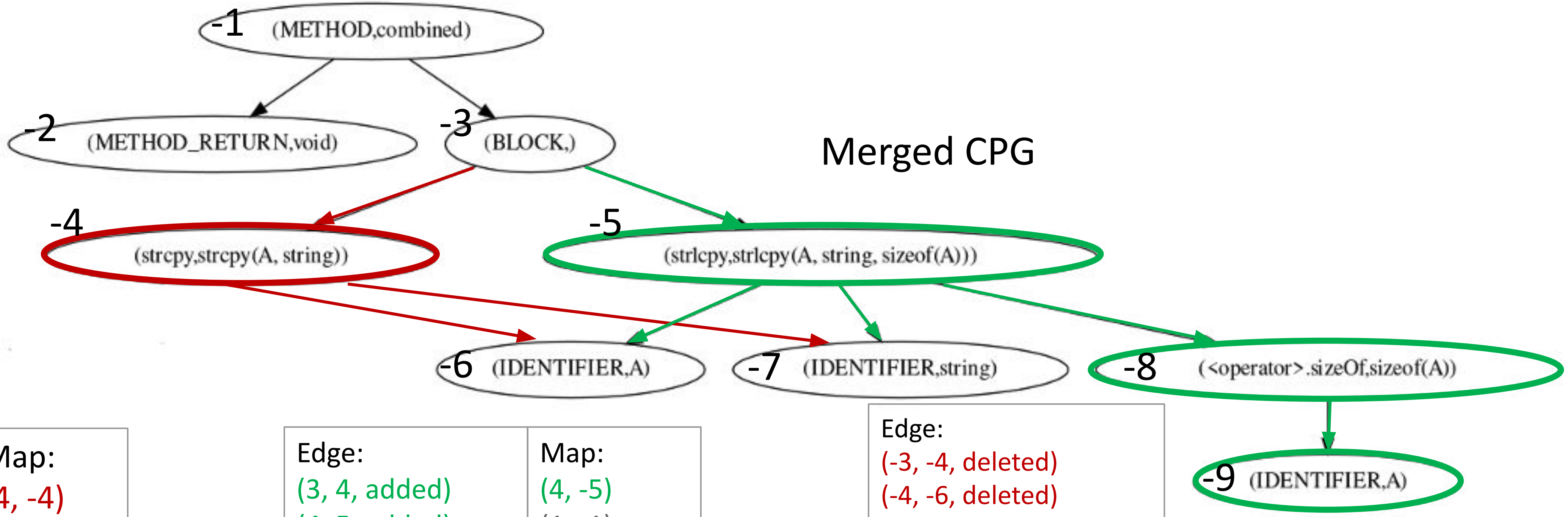
Pre-patched CPG



Node:	Edge:
(4, added)	(3, 4, added)
(7, added)	(4, 5, added)
(8, added)	(4, 6, added)
(1, context)	(4, 7, added)
(2, context)	(7, 8, added)
(3, context)	(1, 2, context)
(5, context)	(1, 3, context)
(6, context)	

Post-patched CPG

Step 3: re-map node IDs and merge edge sets of two CPGs



Merged CPG

Edge:	Map:
(3, 4, deleted)	(4, -4)
(4, 5, deleted)	(1, -1)
(4, 6, deleted)	(2, -2)
(1, 2, context)	(3, -3)
(1, 3, context)	(5, -6)
	(6, -7)

merge

Edge:	Map:
(3, 4, added)	(4, -5)
(4, 5, added)	(1, -1)
(4, 6, added)	(2, -2)
(4, 7, added)	(3, -3)
(7, 8, added)	(5, -6)
(1, 2, context)	(6, -7)
(1, 3, context)	(7, -8)
	(8, -9)

→

Edge:
(-3, -4, deleted)
(-4, -6, deleted)
(-4, -7, deleted)
(-3, -5, added)
(-5, -6, added)
(-5, -7, added)
(-5, -8, added)
(-8, -9, added)
(-1, -2, context)
(-1, -3, context)

PatchCPG Storage Format

- Edge: (start_ID, end_ID, type, version)

```
(-116, -8, AST, 0)
(-340, -175, CFG, 0)
(-249, -175, AST, 0)      //  0 denotes contexture
(-405, -399, CDG, -1)     // -1 denotes deleted
(-397, -422, CFG, 1)      //  1 denotes added
...
```

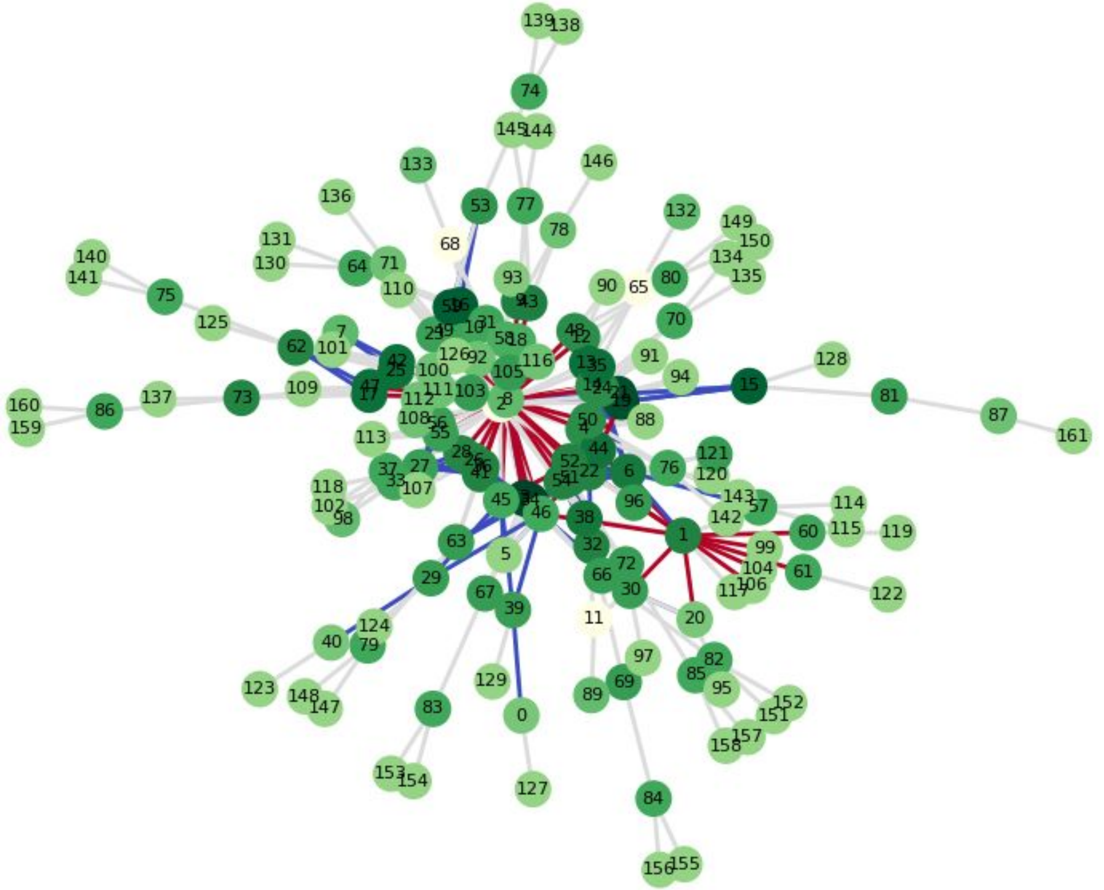
- Node: (node_ID, code, version)

```
(-391, (METHOD_FULL_NAME,posix_acl_update_mode), (SIGNATURE,TODO assignment
signature), (TYPE_FULL_NAME,ANY), (COLUMN_NUMBER,10), (ARGUMENT_INDEX,2), (ORDER,2), (
NAME,posix_acl_update_mode), (CODE,posix_acl_update_mode(
inode,\n\t\t\t\t&inode->i_mode, &acl)), (LINE_NUMBER,6), 1)
(-404, (DISPATCH_TYPE,STATIC_DISPATCH), (METHOD_FULL_NAME,<operator>.assignment), (
SIGNATURE,TODO assignment signature), (TYPE_FULL_NAME,ANY), (COLUMN_NUMBER,1), (
ARGUMENT_INDEX,2), (ORDER,2), (NAME,<operator>.assignment), (CODE,inode->i_ctime =
current_time(inode)), (LINE_NUMBER,13), -1)
(-8, (PARSER_TYPE_NAME,IfStatement), (COLUMN_NUMBER,1), (ARGUMENT_INDEX,1), (
ORDER,1), (CODE,if (type == ACL_TYPE_ACCESS)), (LINE_NUMBER,5), 0)
...
```

Code Slicing: Size Reduction of Patch-CPG

- The graph is too large.
- Not all the contexts are useful.

Solution: we prune the graph by code slicing (Only considering context nodes directly connected to deleted/added ones).



A mid-size PatchCPG sample (Ninf-AST) from the patch *torvalds.linux.fd6040ed57d8f200ab0cc2abf706c54995a48370*

Embeddings for Patch-CPGs

- Edge Embedding
  - 5-dimensional binary vector.
  - 2 bits: represent if the edge belongs to pre/post-patch version.
  - 3 bits: one-hot vector represent the edge type (CDG, DDG, AST).

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

Embeddings for Patch-CPGs

- Node Embedding
  - 20-dimensional numeric features.
  - extracted from the statement in the node.
  - vulnerability-relevant features.
    - code snippet metadata
    - Identifier and literal features
    - Control flow features
    - Operator features
    - API features

Embeddings for Patch-CPGs

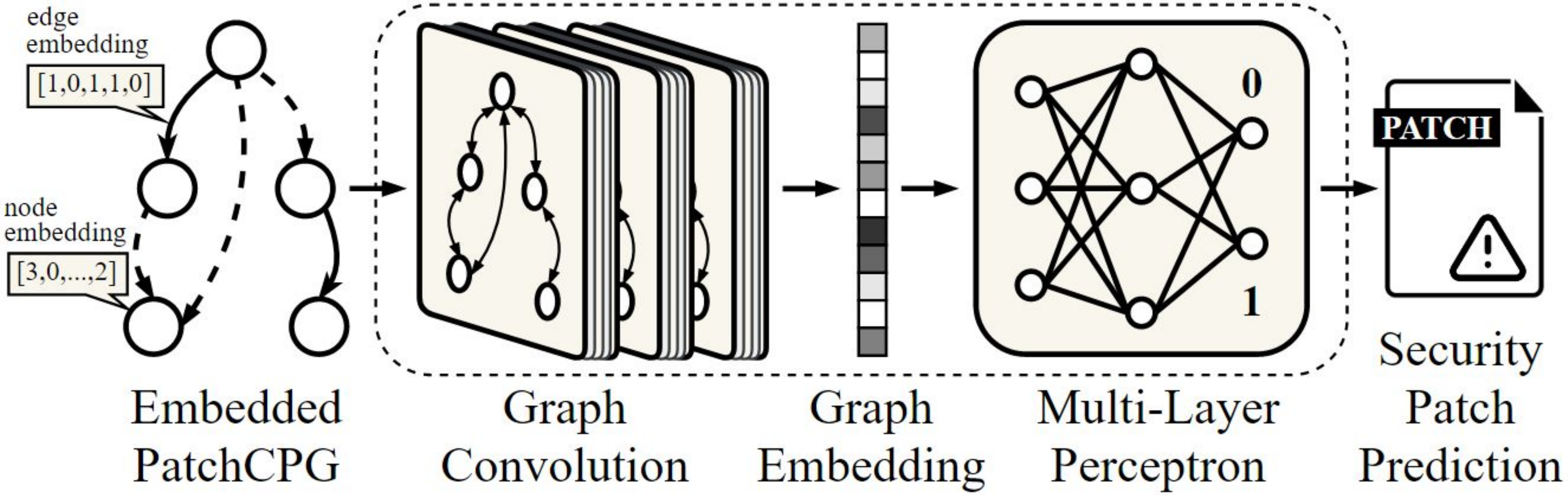
- pointer/array operations are related to OOB access, NULL pointer dereference;
- arithmetic expressions are related to integer overflow.

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

- **Graph Convolution & Pooling:** obtain graph embedding.
- **Multi-Layer Perceptron:** obtain the final prediction.



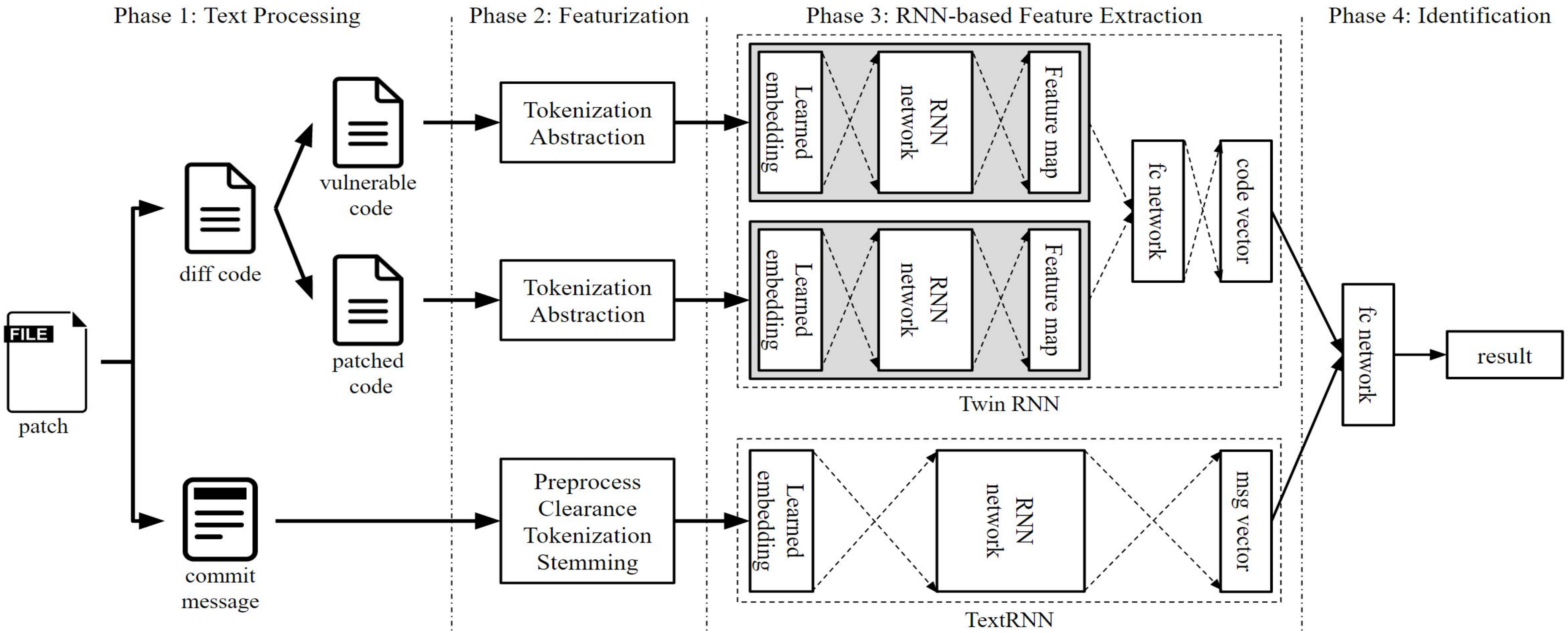
PatchGNN Training and Inference

- **Training phase**
  - Training dataset: PatchDB (38K)
    - https://sunlab-gmu.github.io/PatchDB/
  - Adam optimizer and cross-entropy loss function.
  - Yield a Graph Neural Network (GNN) model.

- **Inference phase**
  - Given a patch, our PatchGNN model tells if it is security-related.
  - Case study: NGINX, Xen, OpenSSL, and ImageMagick.

PatchRNN Architecture

- PatchRNN considers both source-code and text information.



Experimental Results (Comparing with TwinRNN)

Method	Dataset	General Metrics		Special Metrics	
		Accuracy	F1-score	Precision	F.P. Rate
TwinRNN	PatchSPD	69.60%	0.461	48.45%	19.67%
	SPI-DB	56.37%	0.512	48.07%	41.57%
GraphSPD	PatchSPD	80.39%	0.557	77.27%	5.05%
	SPI-DB	63.04%	0.503	63.96%	19.16%

Experimental Results (Comparing with Vulnerability Detection)

Method	#Vul_prepatch	#Vul_postpatch	#SecPatch	TP Rate
CppCheck	3	1	2	0.54%
flawfinder	109	108	1	0.27%
ReDeBug	29	29	0	0.00%
YUDDY	22	16	21	5.71%
VulDeePecker	3	0	3	0.82%
GraphSPD	-	-	53	14.40%

Case Study

- NGINX: detect 21 security patches.

Changes w/	CVE	Total Commits	Valid Commits	Detected SP	Confirmed SP	Precision
1.19.x	3	180	217	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%**

Case Study

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

Questions?