ASE 2024

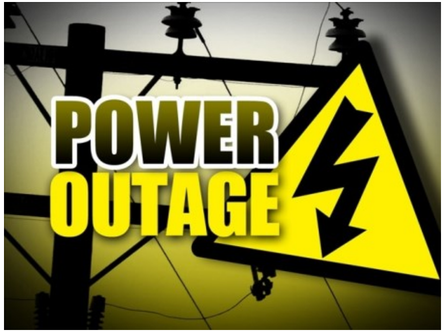
39th International Conference on Automated Software Engineering

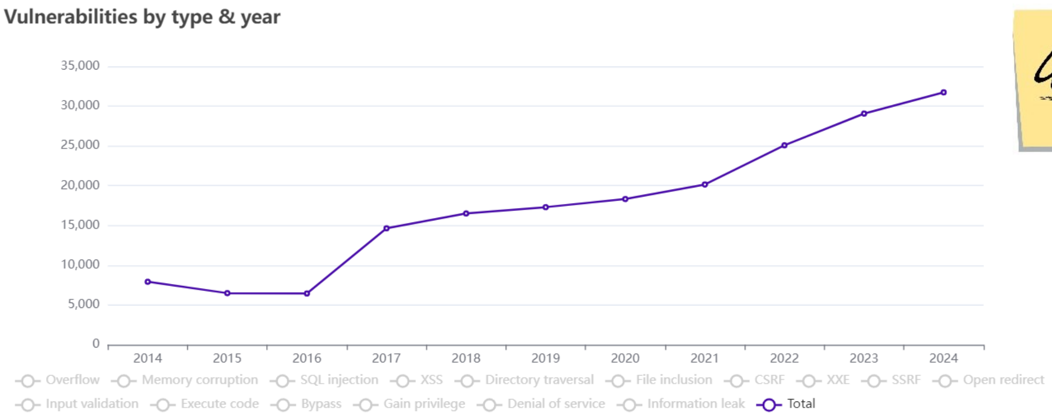
SNOPY: Bridging Sample Denoising with Causal Graph Learning for Effective Vulnerability Detection

Sicong Cao[1], Xiaobing Sun[1], Xiaoxue Wu[1], David Lo[2], Lili Bo[1],
Bin Li[1], Xiaolei Liu[3], Xingwei Lin[4], and Wei Liu[1]

[1] Yangzhou University
[2] Singapore Management University
[3] China Academy of Engineering Physics
[4] Zhejiang University

Software Vulnerability





Timely and Accurate Detection is Urgent!

Vulnerability Detection Advancement

Phase 1

Manual

Code Review, Reverse Engineering, Expertise

Phase 2

Rule

Static/Taint Analysis, Model Checking

Phase 3

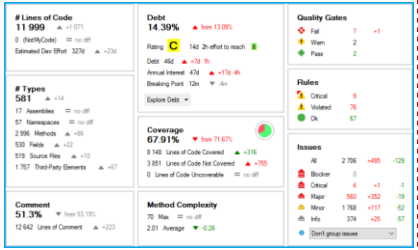
Dynamic

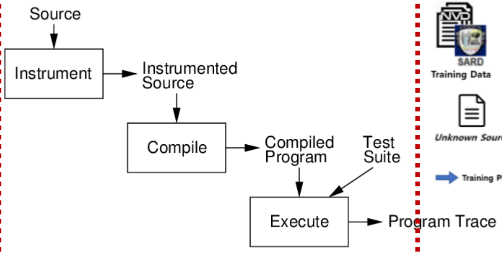
Fuzzing, Symbolic Execution

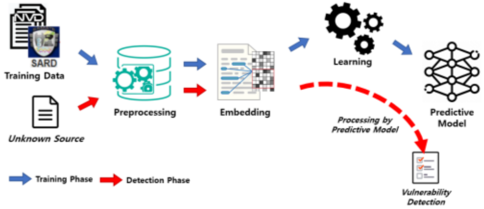
Phase 4

Intelligent

Machine/Deep Learning-Assisted







1960s 1970s 1990s 2013s

knowledge-Intensive, High FP, Poor scalability

High FN、 Low Coverage

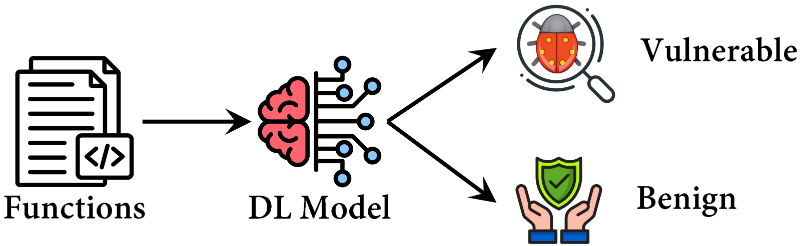
Challenge 1: Noisy Program Semantics

Vulnerability-Fixing Commit (fe9c8426) of CVE-2018-9518

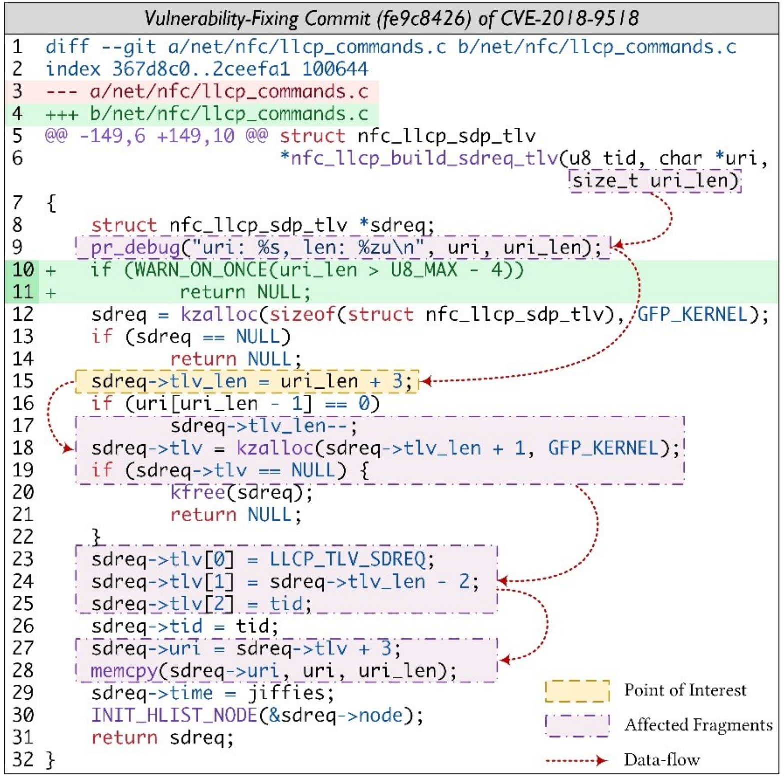
```
diff --git a/net/nfc/llcp_commands.c b/net/nfc/llcp_commands.c
index 367d8c0..2ceefa1 100644
--- a/net/nfc/llcp_commands.c
+++ b/net/nfc/llcp_commands.c
@@ -149,6 +149,10 @@ struct nfc_llcp_sdp_tlv
                     *nfc_llcp_build_sdreq_tlv(u8 tid, char *uri,
                                               size_t uri_len)
{
    struct nfc_llcp_sdp_tlv *sdreq;
    pr_debug("uri: %s, len: %zu\n", uri, uri_len);
+   if (WARN_ON_ONCE(uri_len > U8_MAX - 4))
+           return NULL;
    sdreq = kzalloc(sizeof(struct nfc_llcp_sdp_tlv), GFP_KERNEL);
    if (sdreq == NULL)
            return NULL;
    sdreq->tlv_len = uri_len + 3;
    if (uri[uri_len - 1] == 0)
            sdreq->tlv_len--;
    sdreq->tlv = kzalloc(sdreq->tlv_len + 1, GFP_KERNEL);
    if (sdreq->tlv == NULL) {
            kfree(sdreq);
            return NULL;
    }
    sdreq->tlv[0] = LLCP_TLV_SDREQ;
    sdreq->tlv[1] = sdreq->tlv_len - 2;
    sdreq->tlv[2] = tid;
    sdreq->tid = tid;
    sdreq->uri = sdreq->tlv + 3;
    memcpy(sdreq->uri, uri, uri_len);
    sdreq->time = jiffies;
    INIT_HLIST_NODE(&sdreq->node);
    return sdreq;
}
```

Point of Interest
Affected Fragments
Data-flow

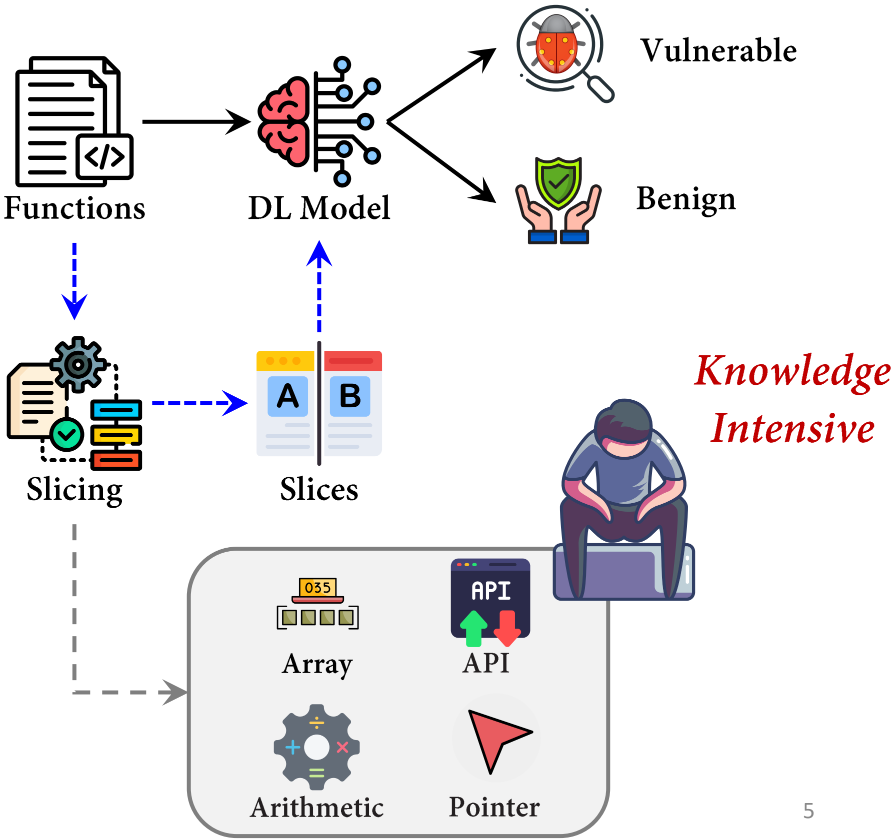
Motivating Example



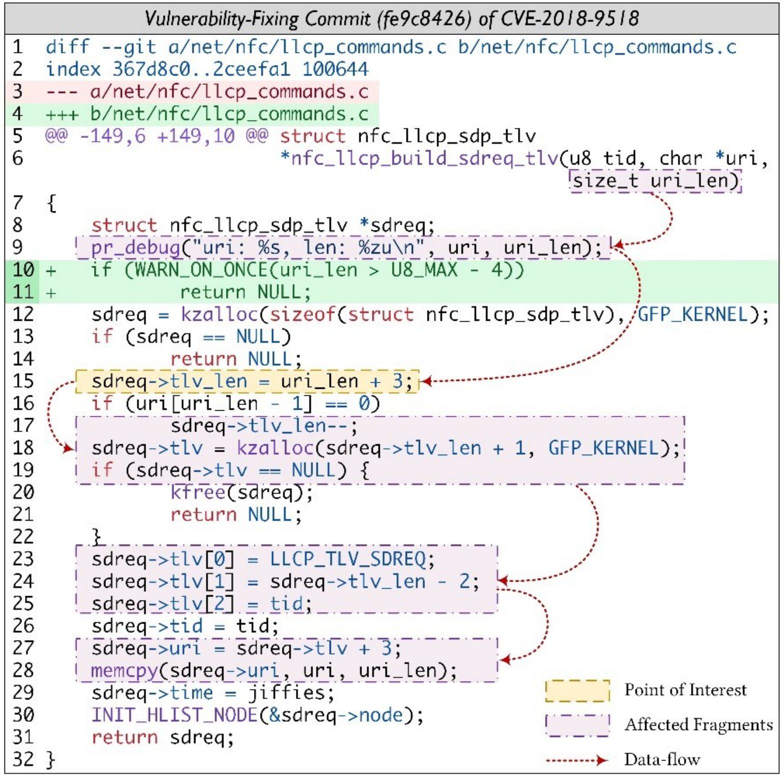
Challenge 1: Noisy Program Semantics



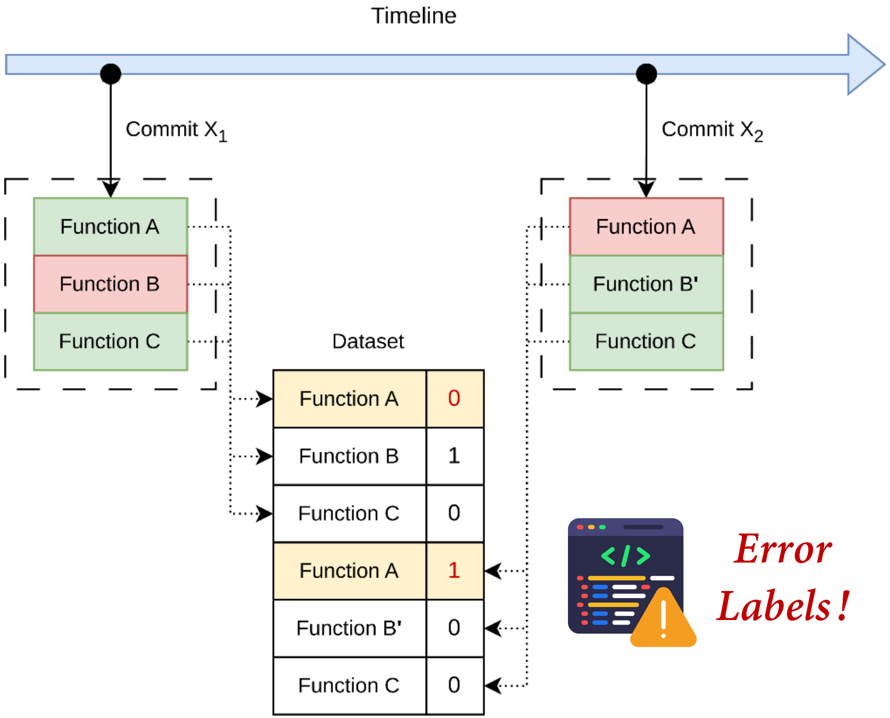
Motivating Example



Challenge 1: Noisy Program Semantics

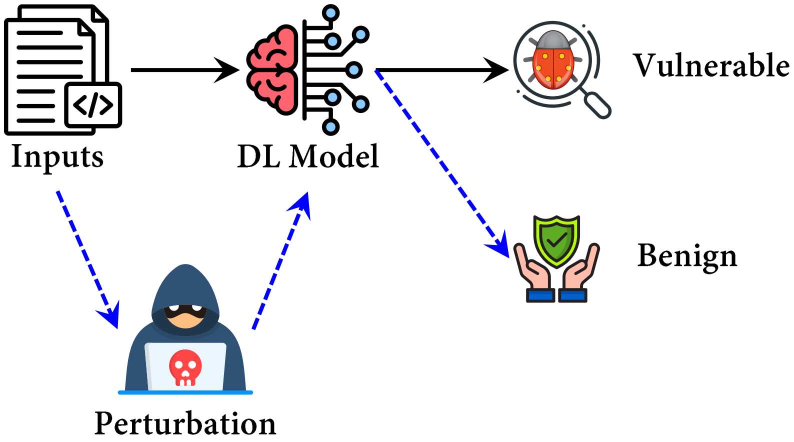


Motivating Example



Error Labels!

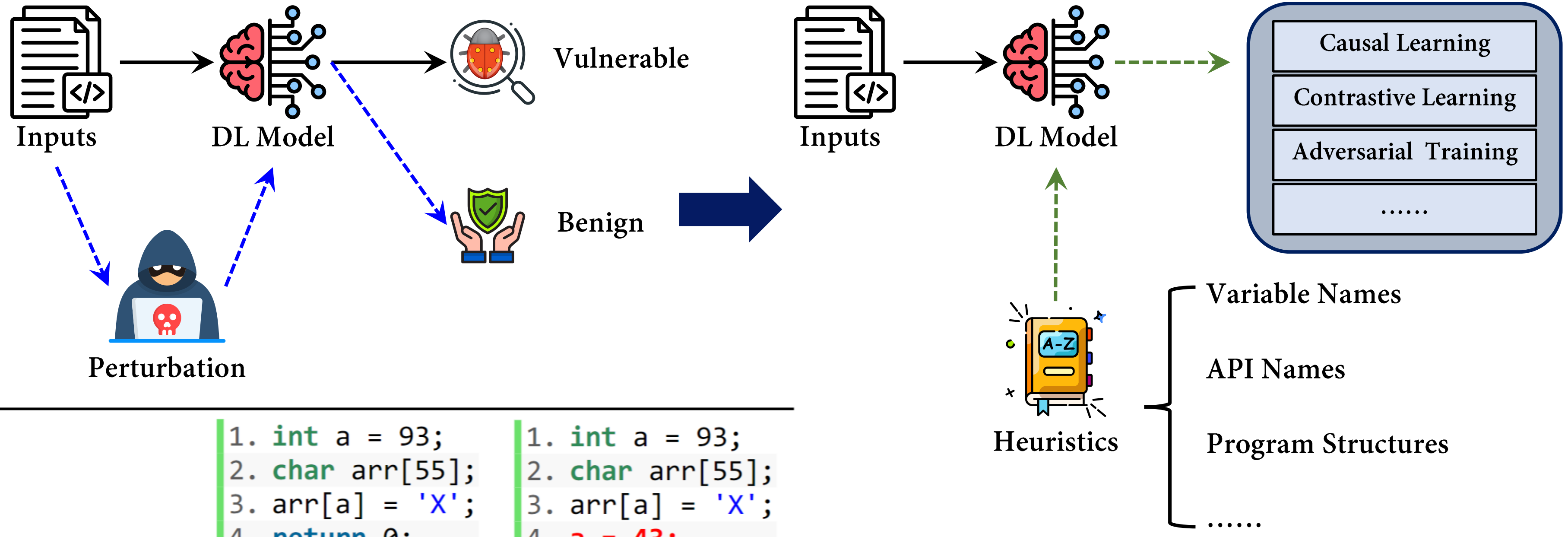
Challenge 2: Learning Spurious Correlations



	`1. int a = 93;`
	`2. char arr[55];`
	`3. arr[a] = 'X';`
	`4. return 0;`
		`1. int a = 93;`
		`2. char arr[55];`
		`3. arr[a] = 'X';`
		`4. a = 43;`
		`5. return 0;`

Ground Truth	Buggy	Buggy
Prediction	Buggy	Non-buggy

Challenge 2: Learning Spurious Correlations




```c
1. int a = 93;
2. char arr[55];
3. arr[a] = 'X';
4. return 0;
```

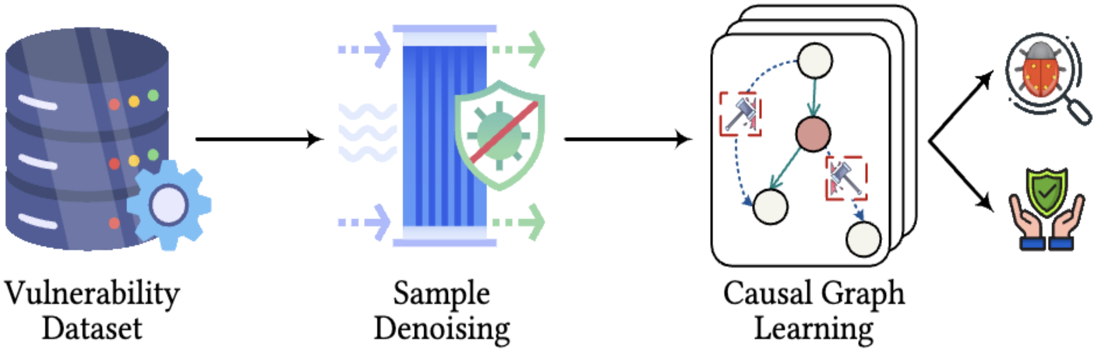
```c
1. int a = 93;
2. char arr[55];
3. arr[a] = 'X';
4. a = 43;
5. return 0;
```

Ground Truth	Buggy	Buggy
Prediction	Buggy	Non-buggy

Fail to capture real vulnerability patterns!

Our approach: SNOPY

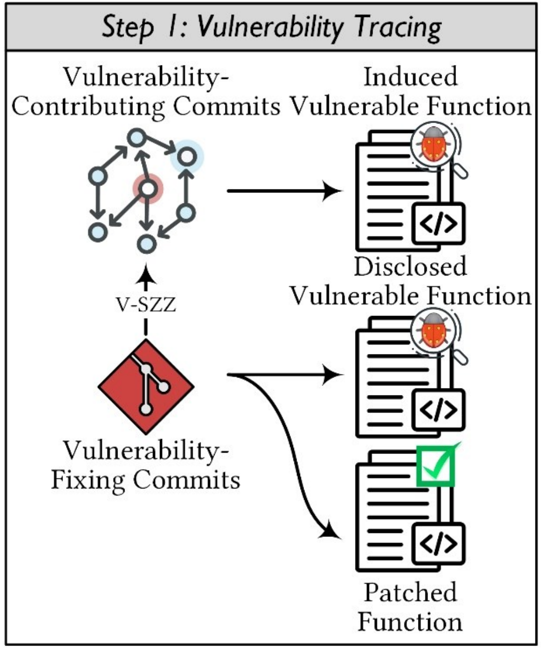
How to capture ***real vulnerability patterns*** *from vulnerable samples with* ***numerous noise*** *for effective detection?*



Workflow of SNOPY

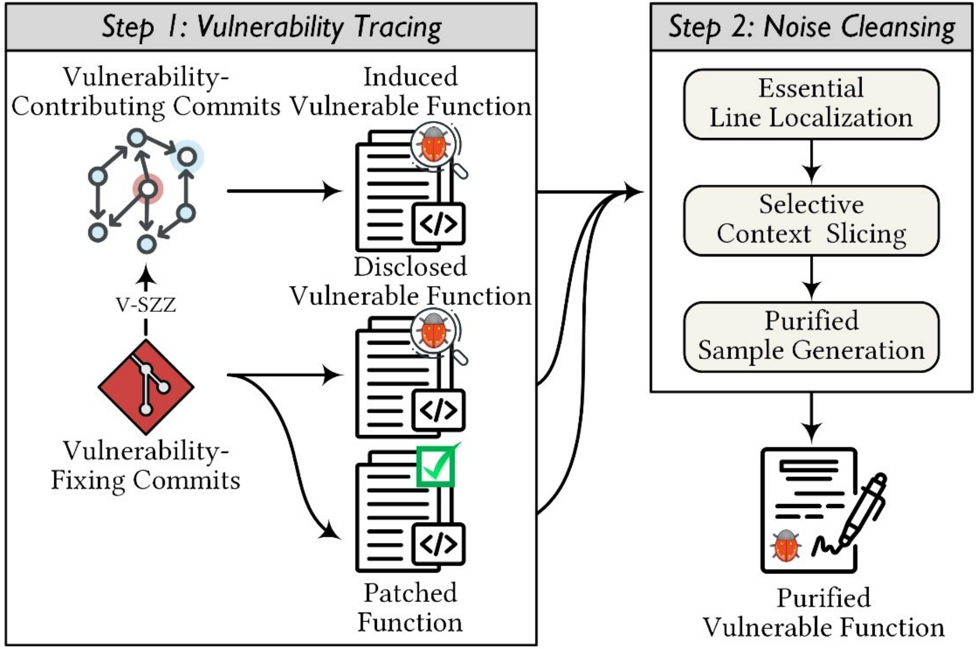
Sample Denoising

*How to localize **vulnerability-related code snippets**?*



Sample Denoising

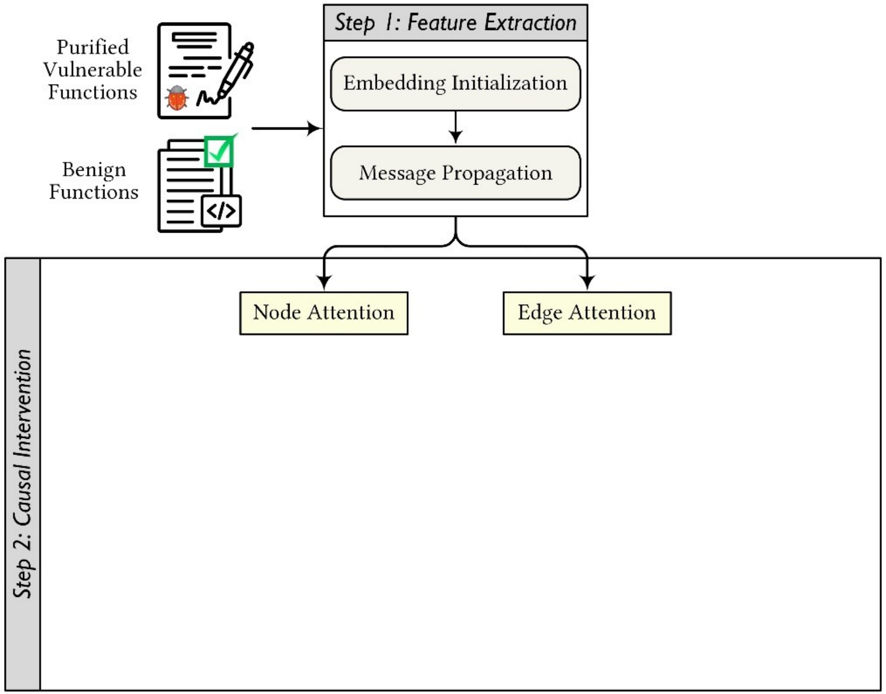
How to localize vulnerability-related code snippets?



- **Rule1:** Performing data-flow slicing as targets statements receive variables/parameters assigned or checked by vulnerable lines.
- **Rule2:** If the previous forward data-flow slicing result is empty, performing control-flow slicing.

Causal Graph Learning

How to learn ***vulnerability-related causal patterns****?*

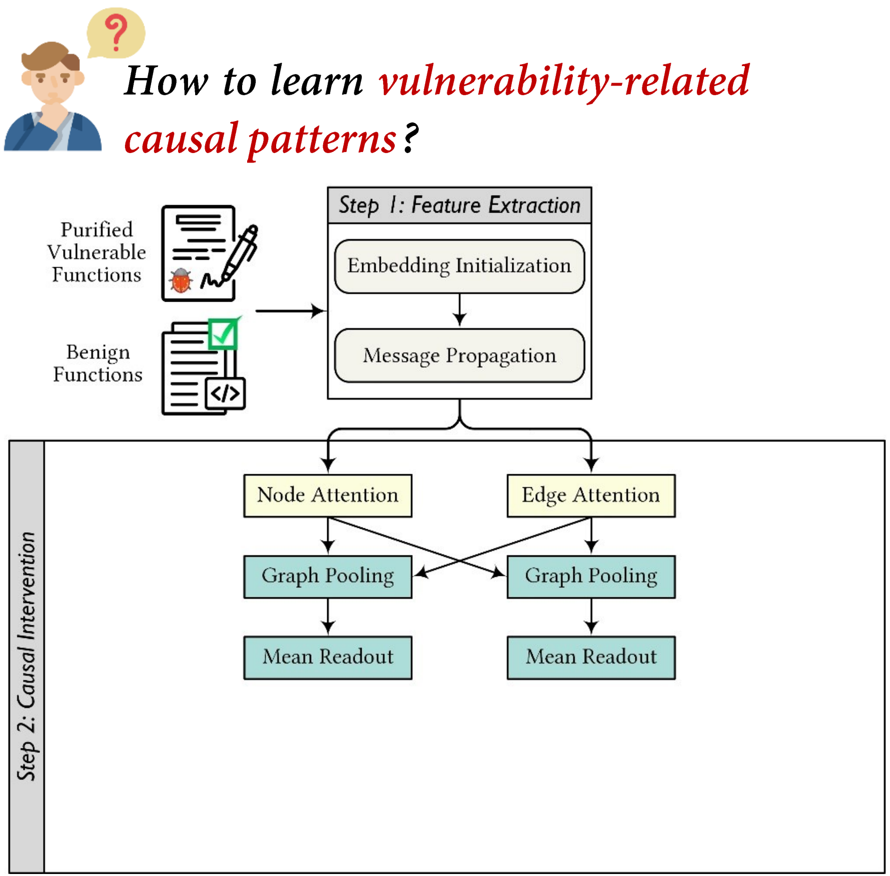


$$\alpha_{c_v}, \alpha_{s_v} = \mathrm{softmax}\left(\mathrm{MLP}_{\mathrm{node}}(h_v)\right)$$

$$\beta_{c_{vu}}, \beta_{s_{vu}} = \mathrm{softmax}\left(\mathrm{MLP}_{\mathrm{edge}}(h_v || h_u)\right)$$

Causal Graph Learning

*How to learn **vulnerability-related causal patterns**?*

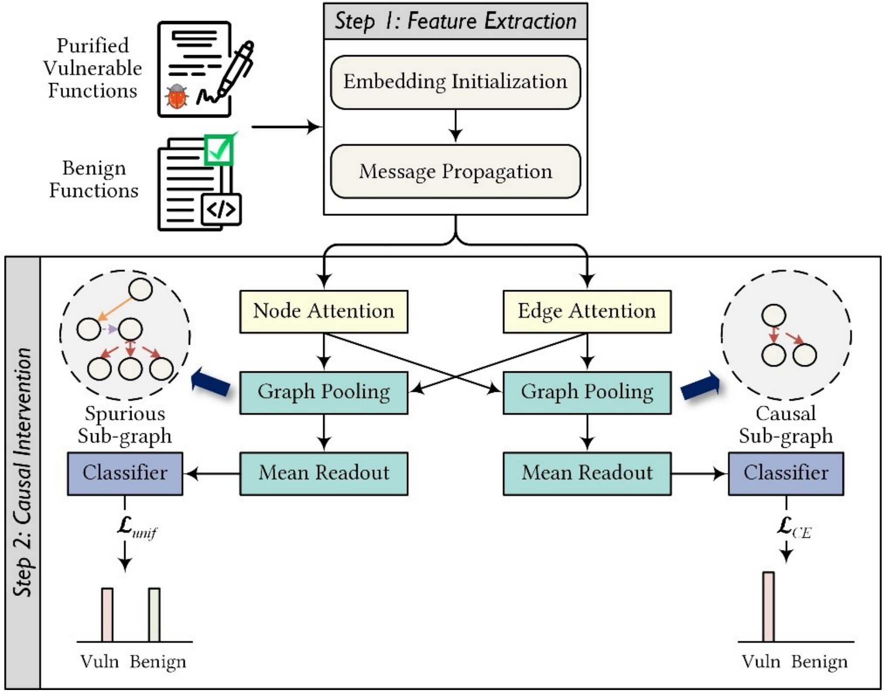


$$h_{\mathcal{G}_c} = \varphi\big(\mathrm{GPL}(A \odot M_c, X \odot F_c)\big)$$

$$h_{\mathcal{G}_s} = \varphi\big(\mathrm{GPL}(A \odot M_s, X \odot F_s)\big)$$

Causal Graph Learning

How to learn vulnerability-related causal patterns?

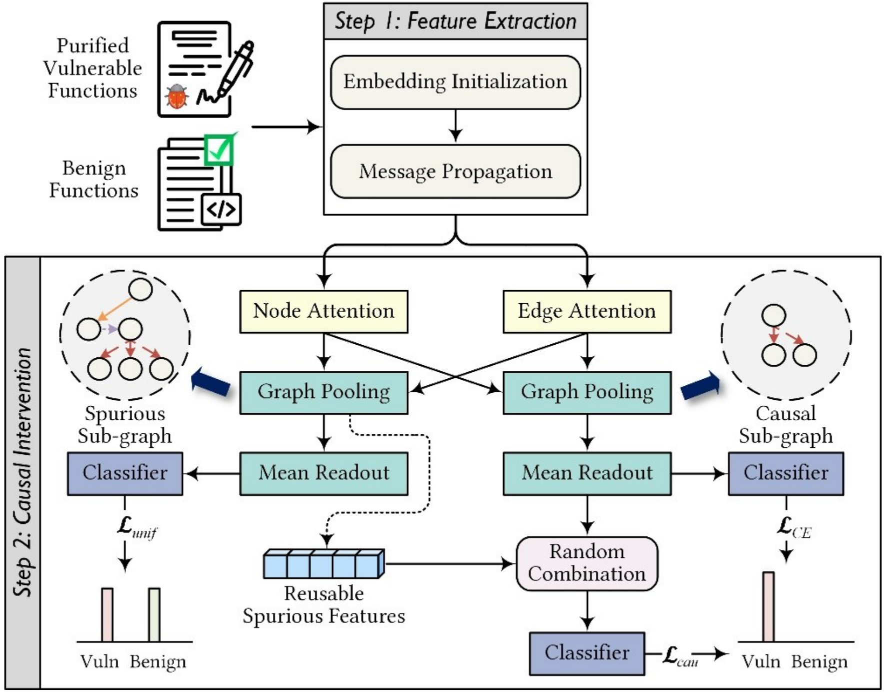


$$\mathcal{L}_{CE} = -\frac{1}{\mathcal{D}} \sum_{\mathcal{G} \in \mathcal{D}} y_{\mathcal{G}}^{\top} \log\left(\Phi\left(h_{\mathcal{G}_c}\right)\right)$$

$$\mathcal{L}_{unif} = -\frac{1}{\mathcal{D}} \sum_{\mathcal{G} \in \mathcal{D}} \mathrm{KL}\left(y_{unif}, \Phi\left(h_{\mathcal{G}_s}\right)\right)$$

Causal Graph Learning

How to learn ***vulnerability-related causal patterns****?*



$$\mathcal{L}_{cau} = -\frac{1}{|\mathcal{D}| \cdot |\mathcal{S}|} \sum_{\mathcal{G} \in \mathcal{D}} \sum_{\mathcal{S}' \in \mathcal{S}} y_{\mathcal{G}}^{\top} \log\left(\Phi\left(h_{\mathcal{G}_c} \oplus h_{\mathcal{G}_{\mathcal{S}'}}\right)\right)$$

$$\mathcal{L}_{total} = \mathcal{L}_{CE} + \lambda_1 \mathcal{L}_{unif} + \lambda_2 \mathcal{L}_{cau}$$

Performance of SNOPY

Research Questions

- RQ1: How does SNOPY perform compared to the state-of-the-art baselines on vulnerability detection?
- RQ2: How effective is SNOPY for detecting different types of vulnerabilities?
- RQ3: How do various components of SNOPY affect its overall performance?

Dataset

Dataset	Vul	Non-vul	Ratio	VFCs	VCCs
FFmpeg+QEMU	12,460	14,858	1:1.2	6,611	6,439
Big-Vul	10,900	177,736	1:16.3	3,754	3,385
DIVERSEVUL	18,945	311,547	1:16.4	7,514	7,022

Baselines

DNN-based

- VulDeePecker (NDSS'18)
- ReVeal (TSE'21)
- SySeVR (TDSC'21)
- IVDetect (ESEC/FSE'21)
- Devign (NeurIPS'19)
- ……

LLM-based

- LineVul (MSR'22)
- SVulD (ESEC/FSE'23)

RQ1: Detection Performance

Approach \ Dataset, Metrics (%)	FFmpeg+QEMU [5]				Big-Vul [17]				DiverseVul [7]			
Approach	**Accuracy**	**Precision**	**Recall**	**F1**	**Accuracy**	**Precision**	**Recall**	**F1**	**Accuracy**	**Precision**	**Recall**	**F1**
VulDeePecker	48.55	33.96	27.47	30.37	83.27	16.56	22.95	19.24	87.44	11.30	24.55	15.48
SySeVR	44.63	35.70	61.87	45.28	82.45	19.63	28.91	23.38	86.16	7.69	14.28	10.00
Devign	51.37	48.15	80.42	60.24	85.64	27.32	13.04	17.65	87.16	24.49	28.07	26.16
ReVeal	53.05	54.19	75.32	63.03	83.79	15.34	30.05	20.31	85.32	20.69	33.19	25.49
IVDetect	56.85	51.33	68.82	58.80	86.97	24.96	32.57	28.26	88.52	17.34	35.26	23.25
DeepWukong	54.61	52.70	71.96	60.84	79.64	13.08	32.55	18.66	82.39	21.64	29.30	24.89
AMPLE	62.88	55.06	77.34	64.33	85.95	28.40	36.11	31.79	88.79	26.35	34.01	29.69
LineVul	63.74	52.44	65.39	58.21	80.26	12.96	38.32	19.37	90.52	**36.18**	26.98	30.91
SVulD	60.51	54.99	**83.48**	66.30	**92.81**	33.24	41.65	36.97	**91.16**	31.44	40.17	35.27
Snopy	**67.33**	**59.64**	78.72	**67.86**	90.75	**38.12**	**46.39**	**41.85**	89.61	33.76	**42.53**	**37.64**

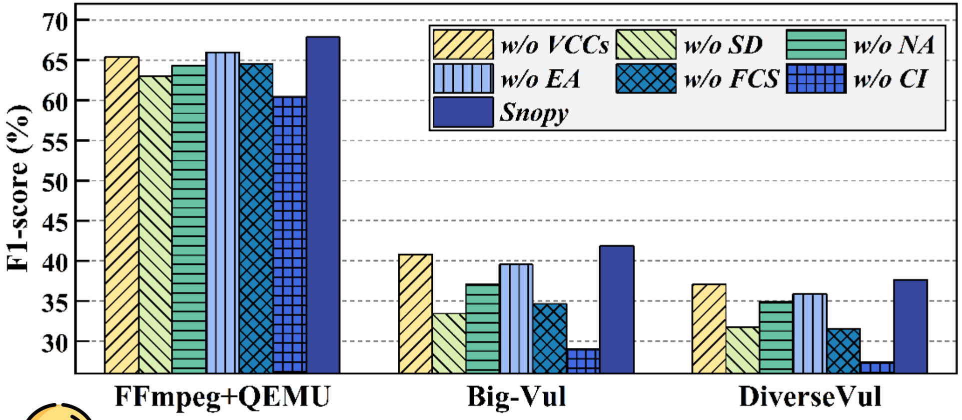
The performance improvements of Snopy over the state-of-the-art approaches are positive. Particularly, Snopy outperforms the best-performing baseline SVulD by 2.35%,13.20%, and 6.72% in F1-score on the three datasets, respectively.

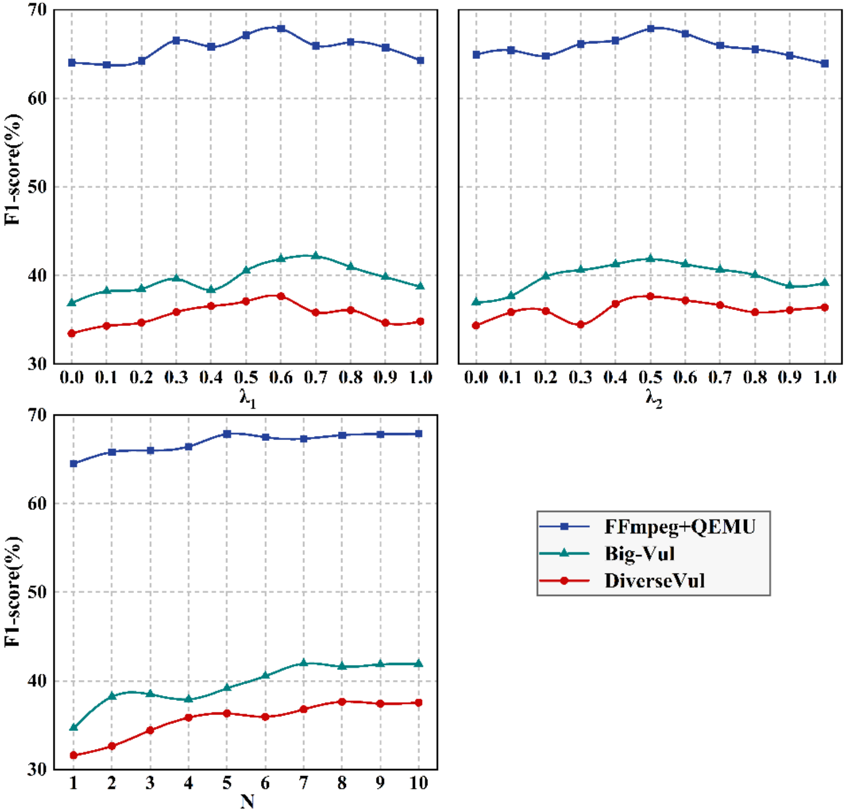
RQ2: Classification Performance

Dataset	Rank	Type	Ratio	SVulD	SNOPY
Big-Vul	1	CWE-787	2.25%	69.44	**73.89**
	4	CWE-416	3.76%	54.71	**68.32**
	6	CWE-20	13.62%	**62.99**	61.43
	7	CWE-125	7.12%	56.38	**69.47**
	12	CWE-476	2.45%	37.59	**75.23**
	14	CWE-190	3.50%	68.53	**72.06**
	17	CWE-119	24.22%	45.77	**69.28**
	21	CWE-362	3.17%	60.28	**65.33**
	Average			56.96	**69.38**
DIVERSEVUL	1	CWE-787	17.98%	**60.47**	56.99
	4	CWE-416	6.24%	53.58	**66.29**
	6	CWE-20	8.16%	44.39	55.83
	7	CWE-125	11.60%	49.51	62.30
	8	CWE-22	1.25%	**33.65**	26.74
	12	CWE-476	6.05%	22.01	**58.14**
	13	CWE-287	0.67%	8.24	13.59
	14	CWE-190	4.86%	**61.77**	48.25
	17	CWE-119	10.14%	56.74	**64.82**
	21	CWE-362	2.84%	38.13	**55.94**
	22	CWE-269	1.22%	6.99	13.34
	23	CWE-94	0.87%	11.37	17.55
	Average			37.24	**44.98**

On two large-scale vulnerability datasets with CWE information, SNOPY produces substantial improvements of up to 21.80% and 20.78% in terms of F1-score on average over the previous best-performing baseline in detecting different types of real-world vulnerabilities.

RQ3: Ablation Study & Sensitivity Analysis

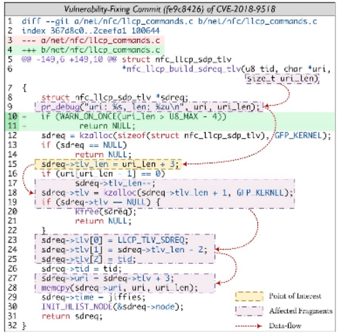




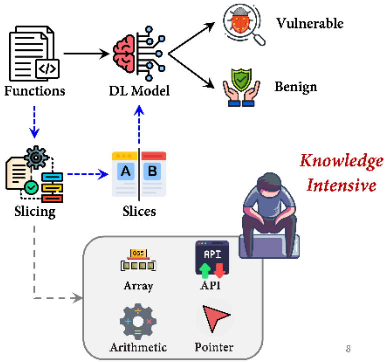
Both sample denoising and causal graph learning are essential for the performance of SNOPY. The most important component of SNOPY is the causal intervention module that results in at most 30.51% improvement in F1-score. Different weights of loss coefficients have varying impart on SNOPY's performance, and the larger capacity of FCS may not always guarantee better performance.

Conclusion

Challenge 1: Noisy Program Semantics

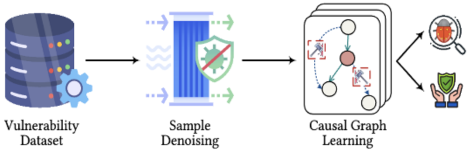


Motivating Example



8

Our approach: SNOPY



Workflow of SNOPY

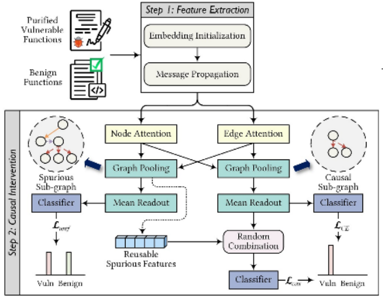
Code Change-based Sample Denoising

Causal Graph Learning

12

Causal Graph Learning

How to learn vulnerability-related causal patterns?

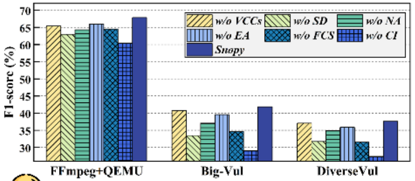


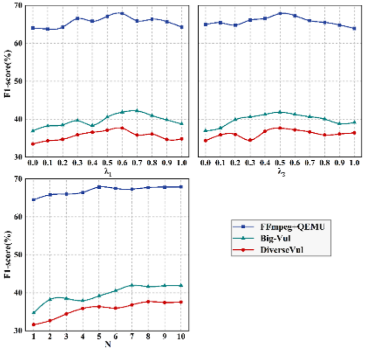
$$\mathcal{L}_{cau} = -\frac{1}{|\mathcal{D}| \cdot |\mathcal{S}|} \sum_{\mathcal{G} \in \mathcal{D}} \sum_{\mathcal{S}' \in \mathcal{S}} y_{\mathcal{G}}^{\mathsf{T}} \log\left(\Phi\left(h_{\mathcal{G}_c} \oplus h_{\mathcal{G}_{s'}}\right)\right)$$

$$\mathcal{L}_{total} = \mathcal{L}_{CE} + \lambda_1 \mathcal{L}_{unif} + \lambda_2 \mathcal{L}_{cau}$$

19

RQ3: Ablation Study & Sensitivity Analysis





Both sample denoising and causal graph learning are essential for the performance of SNOPY. The most important component of SNOPY is the causal intervention module that results in at most 30.51% improvement in F1-score. Different weights of loss coefficients have varying impart on SNOPY's performance, and the larger capacity of FCS may not always guarantee better performance.

23

ASE 2024

39th International Conference on Automated Software Engineering

Thanks for listening !

sicongcao1996@gmail.com

https://github.com/SnopyArtifact/Snopy

Paper

Artifact

揚州大學 YANGZHOU UNIVERSITY

SINGAPORE MANAGEMENT UNIVERSITY

中国工程物理研究院 CHINA ACADEMY OF ENGINEERING PHYSICS

浙江大學 ZHEJIANG UNIVERSITY