中国人民大学
RENMIN UNIVERSITY OF CHINA

HUAWEI

Raisin: Identifying Rare Sensitive Functions for Bug Detection

Jianjun Huang, Jianglei Nie, Yuanjun Gong, Wei You, Bin Liang, Pan Bian

2024.04.19

Background

- Detecting bugs, especially via static analysis techniques, requires **prior knowledge** about the bugs.
- E.g., detecting *memory leak* and *use-after-free* in the Linux kernel requires the knowledge about which functions (**sensitive functions**) allocate or free memory.

```
1169 victim_name = kmalloc(victim_name_len, GFP_NOFS);
1170 if (!victim_name)
1171     return -ENOMEM;
1172 read_extent_buffer(leaf, victim_name, ...
1173                    victim_name_len);
......
1180 ret = backref_in_log(log_root, &search_key,
1181      parent_objectid, victim_name,
1182      victim_name_len);
1183 if (ret < 0) {
1184 +   kfree(victim_name);
1185     return ret;
1186 }
```

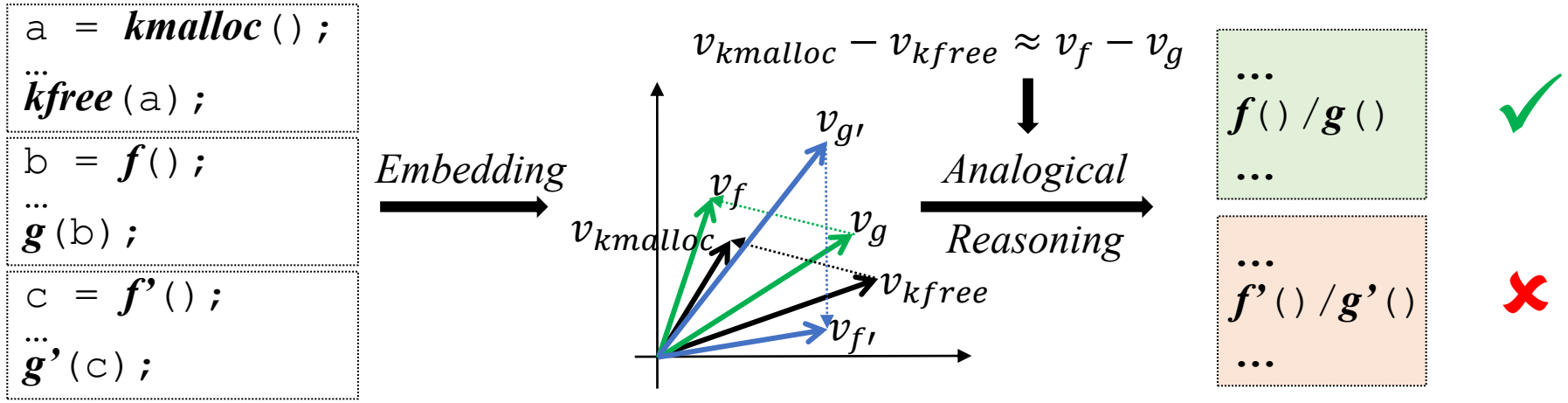
https://github.com/torvalds/linux/commit/f35838a6

```
260 - kfree_sensitive(ctx_p->user.key);
261   dev_dbg(dev, "...%p\n", ctx_p->user.key);
262 + kfree_sensitive(ctx_p->user.key);
```

https://github.com/torvalds/linux/commit/3d950c34

Reasoning about Sensitive Functions: Existing Approaches

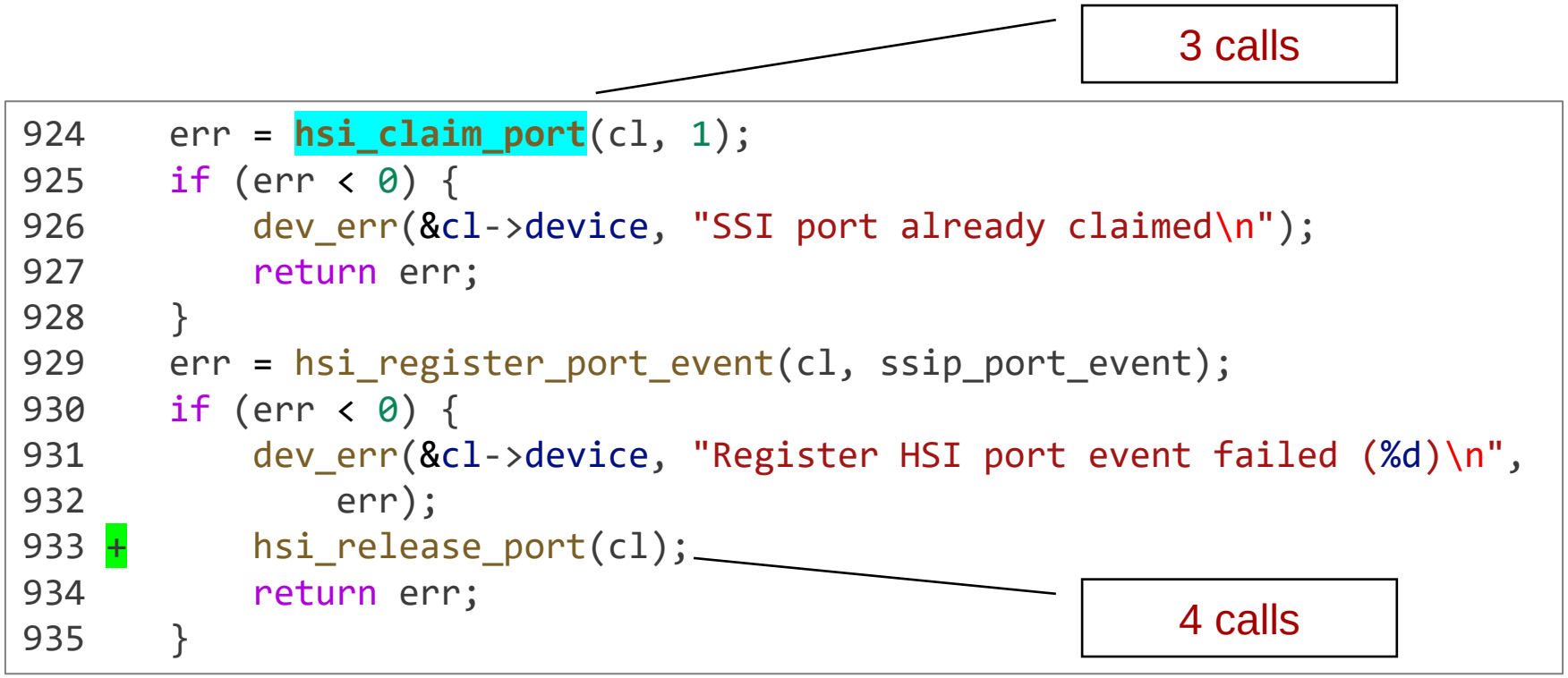
- *SuSi*[1]: SVM classifier
- *SinkFinder*[2]: **analogical reasoning**
  - "*If Germany is to Berlin, then Portugal is to ?*"
  - Pair-to-pair reasoning
  - **Frequently co-occurring** (10+ invocations)



[1] Siegfried Rasthofer, Steven Arzt, and Eric Bodden. *A Machine-learning Approach for Classifying and Categorizing Android Sources and Sinks*. In NDSS'14.
[2] Pan Bian, Bin Liang, Jianjun Huang, Wenchang Shi, Xidong Wang, and Jian Zhang. *SinkFinder: harvesting hundreds of unknown interesting function pairs with just one seed*. In ESEC/FSE'20.

Rare Sensitive Functions

- Rare functions are invoked only few times in a software system.
- In Linux v5.19, **230K rare** (< 10 invocations) *v.s.* 400K total
- **Rare sensitive functions** cause bugs as frequent sensitive functions.



https://github.com/torvalds/linux/commit/b28dbcb3

Research Problem & Solution

Research Problem

How to design a reasoning method specially for identifying rare sensitive functions?

Solution

Raisin: context-based analogical reasoning

- Infer rare sensitive functions using a single known one (e.g., kmalloc), instead of a know pair (e.g., kmalloc/kfree)
- Compose pairs consisting of functions and their **contexts** (i.e., *data correlated calls*)
- Perform analogical reasoning based on function-context pairs

Basic Idea

```
a = kmalloc();
memset(a, ...);
...
kvfree(a);
```

```
b = f();
g(b);
...
h(b);
```

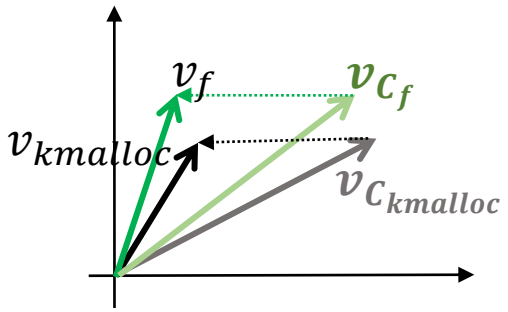
Function-Context Pair

kmalloc
$C_{kmalloc}$

f
C_f

Function-Context Pair

Embedding



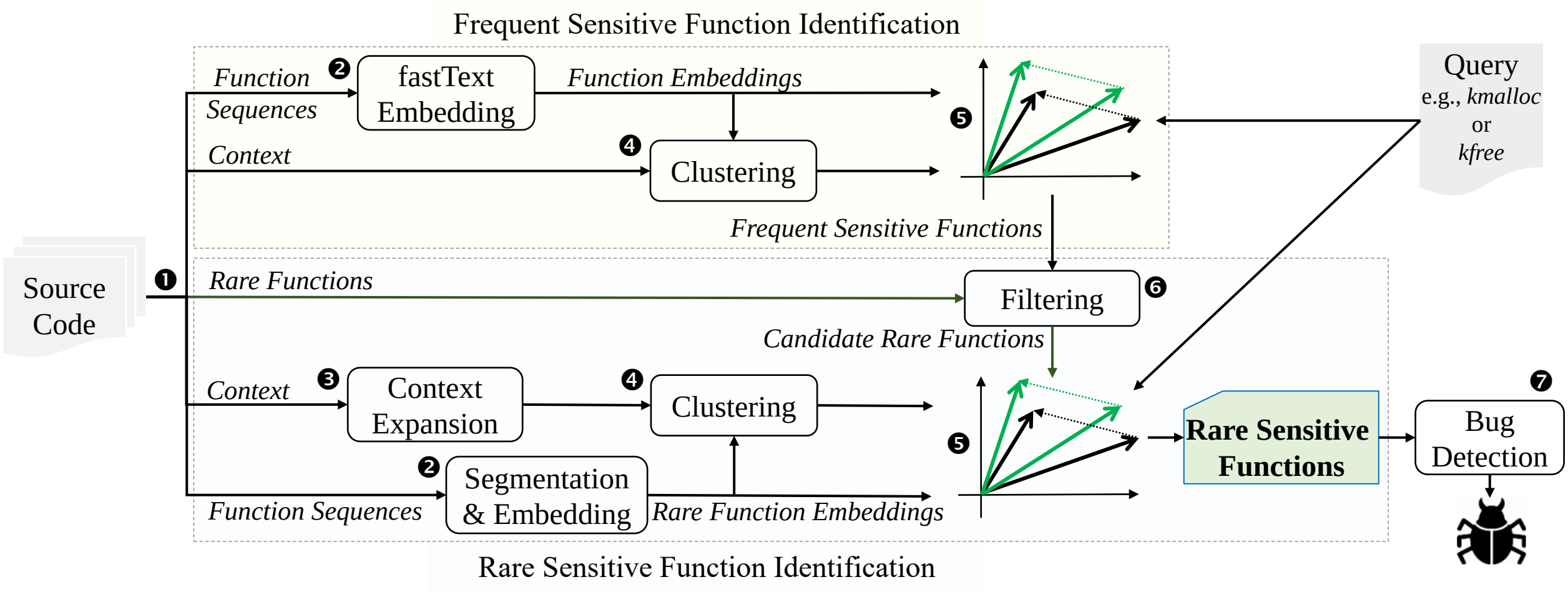
$$v_{kmalloc} - v_{C_{kmalloc}} \approx v_f - v_{C_f}$$

Analogical Reasoning

```
...
f()
...
```

Rare Sensitive Functions

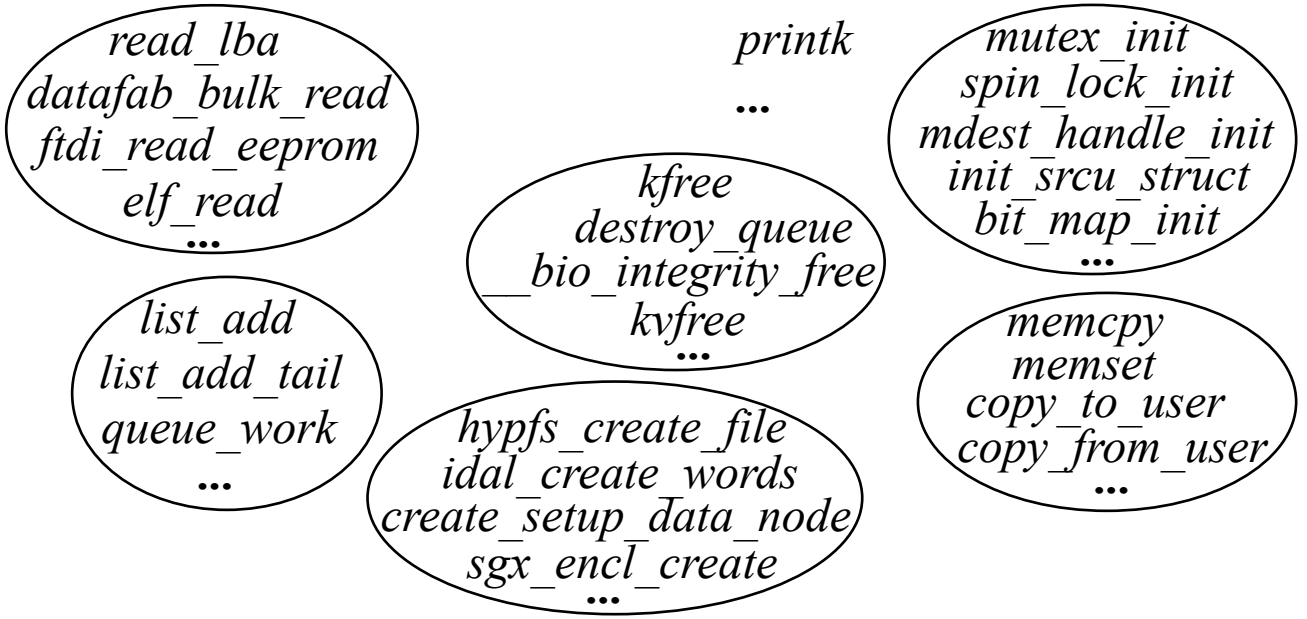
Raisin: Workflow



Context Clustering

- Using the context as a whole is not a good idea, because
  - Different kinds of operations can interleave each other
  - Context functions for one function may not appear in the context for another function

- **Cluster** the context
  - Embedding vectors of function names
  - DBSCAN algorithm
  - Sub-contexts



Example: context for kmalloc

Context Expansion

- Only for rare functions.
- Limited number of invocations → very few functions in the context;
- Some context functions are rare.
- Expand the rare functions in the context and gather correlated frequent functions, to augment the context.

Algorithm 2 Context collection and expansion for a rare function f at a specific *callsite*, given the call graph *cg*. The variables in $\mathcal{V}$ will be tracked if $\mathcal{V}$ is not empty.

```
1:  procedure CTX_EXPAND( f, callsite, cg, V )
2:      C = [];
3:      caller = (callsite ≠ null) ? get_caller(cg, callsite) : f;
4:      cfg = get_cfg(caller);
5:      T = get_related_func_in_caller(cfg, callsite, V);
6:      for cs ∈ T do
7:          callee = get_invoke_target(cs);
8:          if callee is a frequent function then
9:              C = concat(C, callee);
10:         else
11:             V' = extract_correlated_vars(callsite, cs);
12:             D = CTX_EXPAND(callee, null, cg, V');
13:             C = concat(C, D);
14:         end if
15:     end for
16:     if callsite == null then
17:         return C;
18:     end if
19:     for cs ∈ get_callsites(cg, caller) do
20:         if (V' = extract_correlated_vars(cs, callsite)) ≠ ∅ then
21:             D = CTX_EXPAND(caller, cs, cg, V');
22:             C = concat(C, D);
23:         end if
24:     end for
25:     return C;
26: end procedure
```

Weighted Subword Embedding for Rare Functions

- OOV in NLP ←→ Rare functions
- General word embedding techniques cannot generate as high-quality embeddings as for frequent functions.
- Rare sensitive functions usually contain an operational subword to indicate their sensitive behaviors.

- Split the function f into a set, apply frequency-based weight to each subword embedding, and aggregate the weighted subword embeddings:

$$V_f = \frac{\sum_{\{s \in f\}} \mathbf{E}[s] \times N(s, O)}{\sum_{\{s \in f\}} N(s, O)}$$

Key Subwords-based Filtering

- Audit the reported frequent sensitive functions and extract the project-specific operational key subwords.
  - (Linux) *tee_shm_**free***, *gss_**put**_ctx*, *daemon_**destroy**_ctx*; (FreeBSD) ***delete**_unrhdr*; (QEMU) *qcow2_cache_table_**release***
  - (Linux) *kz**alloc***, *usb_**get**_phy*, *debugfs_**create**_dir*; (OpenSSL) *EVP_CIPHER_CTX_**new***; (FFmpeg) *ff_**get**_video_buffer*
- Filter the rare functions to obtain the most likely candidates for rare sensitive function identification.

Evaluation

- **Implementation**:
  - Code parsing and data flow analysis: *fuzzyc2cpg*
  - Clustering, expansion, training and reasoning: Python code
  - Static bug detector: Clang Static Analyzer
- **Datasets**:

TOE	LoC	#Functions	#Rare Functions
Linux v5.19	31,317,255	456,086	241,366
FreeBSD v13.1	8,399,769	194,913	116,843
OpenSSL v3.1.1	1,048,122	14,206	9,836
FFmpeg v6.0	1,600,078	17,417	14,168
QEMU v8.1.0	22,479,680	86,980	75,443

Effectiveness

- Five Categories of sensitive functions.
- Randomly audit 100 candidates of rare sensitive functions.
- The average precision ranges from **83% to 98%**.
- **Comparison**: SinkFinder identifies only 15/4, 5/3, 0/0, 2/1 distinct rare alloc/free functions, respectively, in Linux, OpenSSL, FFmpeg and QEMU.

	Linux			FreeBSD		
	Query	**#RSF**	**P**	**Query**	**#RSF**	**P**
Alloc	*kmalloc*	3,223	83%	*malloc*	2,342	78%
Dealloc	*kfree*	4,990	92%	*free*	1,759	91%
Lock	*mutex_lock*	763	65%	*pthread_mutex_lock*	603	76%
Unlock	*mutex_unlock*	338	90%	*pthread_mutex_unlock*	239	93%
FormatStr	*sprintf*	46	96%	*sprintf*	257	78%
Overall		9,360	85%		5,200	83%

	OpenSSL			FFmpeg			QEMU		
	Query	**#RSF**	**P**	**Query**	**#RSF**	**P**	**Query**	**#RSF**	**P**
Alloc	*OPENSSL_malloc*	244	92%	*av_malloc*	127	94%	*g_malloc*	1,204	82%
Dealloc	*OPENSSL_free*	228	97%	*av_free*	73	96%	*g_free*	352	92%
Lock	*CRYPTO_THREAD_lock_new*	11	91%	*pthread_mutex_lock*	2	100%	*qemu_mutex_lock*	38	90%
Unlock	*CRYPTO_THREAD_lock_free*	6	100%	*pthread_mutex_unlock*	2	100%	*qemu_mutex_unlock*	23	100%
FormatStr	*sprintf*	10	100%	*snprintf*	3	100%	*sprintf*	14	100%
Overall		499	96%		207	98%		1,631	93%

Efficiency

- 16 GB memory, an Intel Core i5-10400F CPU @ 2.9GHz and Ubuntu 20.04.
- Most time-consuming step: preprocessing, including parsing source code, extracting context with a data-flow analysis.
- Embedding (including training) and analogical reasoning (including context clustering) are fast.

Target	Preprocessing	Frequent Functions		Rare Functions			
		Embedding	Analogy	Filtering	Embedding	Expansion	Analogy
Linux	8h43m	3m25s	5m36s	30m	5m11s	15m28s	1m21s
FreeBSD	3h42m	46s	16s	30m	2m13s	1m43s	12s
OpenSSL	46m23s	18s	5s	15m	40s	21s	2s
FFmpeg	1h24m	22s	7s	15m	1m17s	21s	5s
QEMU	5h19m	1m9s	38s	30m	3m19s	4m38s	14s

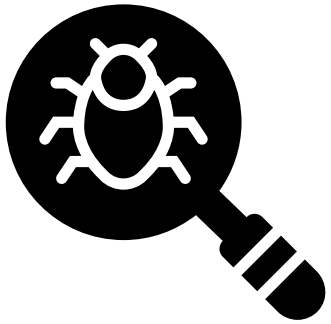
Ablation Study

- Evaluate the effectiveness of the three techniques in Raisin: *key subwords-based filtering*, *weighted subword embedding* and *context expansion*.
- Eight combinations of the techniques identify rare alloc/free functions in Linux: 50K+ functions are reported.
- Manually audit the result: 2,869 alloc and 4,788 free functions are confirmed (as the **estimated ground truth**).
- Filtering ↗Precision; Embedding ↗Recall; Expansion ↗Both.

ID	Key subwods-based Filtering	Weighted Subword Embedding	Context Expansion	Precision			Recall			F1
				Alloc	Free	Average	Alloc	Free	Average	
1	×	×	×	11.80%	13.40%	12.60%	9.90%	6.00%	8.00%	9.80%
2	×	×	✓	15.10%	13.00%	14.10%	25.60%	16.40%	21.00%	14.90%
3	×	✓	×	24.60%	13.50%	19.10%	71.60%	76.70%	74.20%	30.40%
4	×	✓	✓	32.90%	12.50%	22.70%	90.40%	94.50%	92.50%	36.60%
5	✓	×	×	71.00%	80.50%	75.80%	9.90%	6.00%	8.00%	11.10%
6	✓	×	✓	82.20%	92.50%	87.40%	25.60%	16.40%	21.00%	33.90%
7	✓	✓	×	75.40%	85.90%	80.70%	71.60%	76.70%	74.20%	77.30%
8	✓	✓	✓	80.50%	90.70%	85.60%	90.40%	94.50%	92.50%	**88.90%**

Bugs

- Use the discovered alloc/free functions to detect resource-related bugs in Linux and FreeBSD.
- Report 21 and 6 to the kernel maintainers; 19 and 2 are confirmed by the maintainers.
- **9 of the bugs result from sensitive functions that are invoked only once;** the other related sensitive functions have at most 3 invocations.



ID	Target Systems	Rare Sensitive Function	#Occurrences	Bug Type	Confirmation
1	Linux	*mhi_alloc_controller*	1	Memory Leak	[Github]: 43e7c350
2	Linux	*sfp_alloc*	1	Memory Leak	[Github]: 0a18d802
3	Linux	*xhci_alloc_stream_ctx*	1	Memory Leak	[Github]: 7e271f42
4	Linux	*audioreach_alloc_graph_pkt*	1	Memory Leak	[Github]: df5b4aca
5	Linux	*elfcorehdr_alloc*	1	Memory Leak	[Github]: 12b9d301
6	Linux	*aq_nic_init*	2	Memory Leak	[Github]: 65e5d27d
7	Linux	*nfp_cpp_area_alloc*	3	Memory Leak	[Github]: c56c9630
8	Linux	*auxiliary_device_uninit*	1	Use After Free	[Github]: 1c11289b
9	Linux	*nouveau_bo_del_ttm*	3	Use After Free	[Github]: 540dfd18
10	Linux	*amdgpu_vm_init*	2	Resource Leak	[Github]: c3c48339
11	Linux	*hfi1_alloc_ctxt_rcv_groups*	2	Memory Leak	[Github]: aa2a1df3
12	Linux	*init_mr_info*	2	Memory Leak	[Github]: b3236a64
13	Linux	*damon_new_ctx*	3	Memory Leak	[Github]: 188043c7
14	Linux	*init_rx_sa*	1	Resource Leak	[Github]: c7b205fb
15	Linux	*init_tx_sa*	1	Resource Leak	[Github]: c7b205fb
16	Linux	*hsi_claim_port*	3	Resource Leak	[Github]: b28dbcb3
17	Linux	*bnx2x_frag_alloc*	3	Memory Leak	[Github]: b43f9acb
18	Linux	*sec_queue_aw_alloc*	1	Inconsistent Argument	[Github]: 32c0f7d4
19	Linux	*mcba_usb_get_free_ctx*	2	Resource Leak	[Lore]: 20221124144532.6u3hnvb6b2ninlxy@pengutronix.de
20	FreeBSD	*ext4_ext_alloc_meta*	2	Memory Leak	[Bugzilla]: 265071
21	FreeBSD	*bhnd_alloc_pmu*	3	Resource Leak	[Bugzilla]: 265147

Conclusion

- Explore the automated identification of rare sensitive functions;
- Propose a context-based analogical reasoning method, with weighted subword embedding, context expansion and key subwords-based filtering to improve the detection performance;
- Develop a prototype Raisin, evaluate it on five large code bases and report tens of thousands of rare sensitive functions with 91% accuracy on average;
- Find real bugs with 21 confirmed by the Linux and FreeBSD kernel maintainers.

- Artifacts: *https://github.com/jlgithub66/rarefunctions*

Q & A