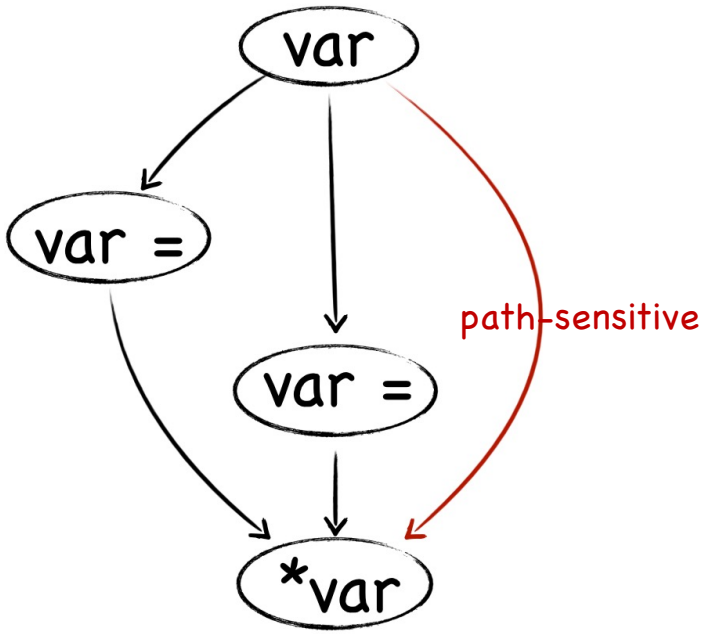
UBITect: A Precise and Scalable Method to Detect Use-Before-Initialization Bugs in Linux Kernel

Yizhuo Zhai, Yu Hao, Hang Zhang, Daimeng Wang, Chengyu Song,
Zhiyun Qian, Mohsen Lesani, Srikanth V. Krishnamurthy, Paul Yu

Use-Before-Initialization (UBI) Bugs

```
1   static int queue_manag(void *data)
2   {
3       /* backlog is declared without initialization */
4       struct crypto_async_request *backlog;
5       if (cpg->eng_st == ENGINE_IDLE) {
6           backlog = crypto_get_backlog(&cpg->queue);
7       }
8       /* Uninitialized backlog is used*/
9       if (backlog) {
10          /* uninitialized pointer dereferenced! */
11          backlog->complete(backlog, -EINPROGRESS);
12      }
13      return 0;
14  }
```

(1) Vulnerable Code



(2) UBI Scenario

Security Risks

```
malicious->func();
```

Arbitrary Code Execution

```
uninit->func();
```

Denial of Service

```
copy_to_user(dst, src, size);
```

Information Leakage

```
for (int i = 0; i < len; i++)
    a[i]
```

Out of Bound Memory Access

Previous Solutions and Limitations

Mitigation:

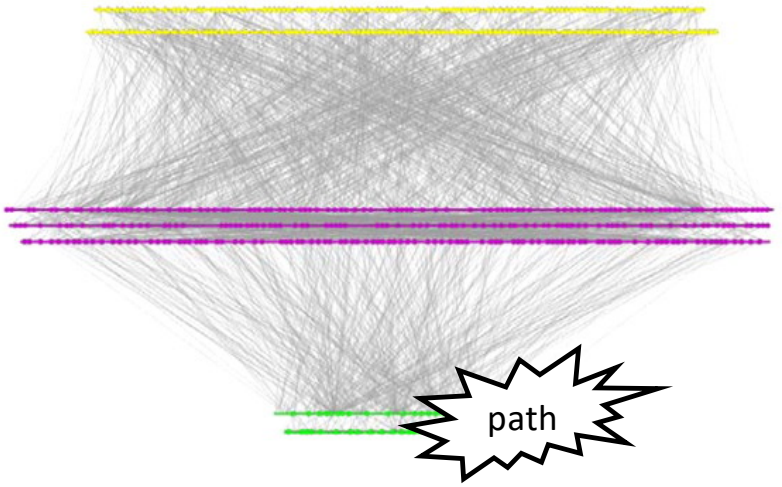
- Zeroing the allocated object:
  e.g. Unisan, SafeInit

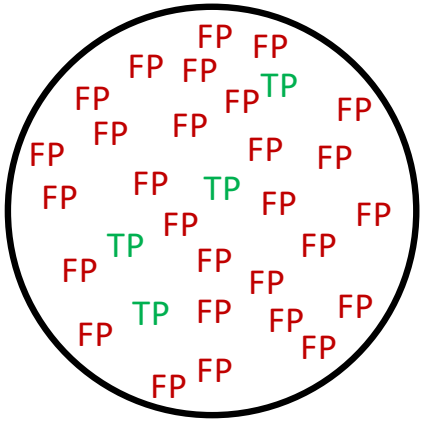
Detection:

- Intra-procedural static analysis:
  e.g. -Wuninitialized, cppcheck
- Symbolic execution:
  e.g. Clang Static Analyzer
- Dynamic Analysis:
  e.g. MemorySanitizer, kmemcheck

Challenges

Scalability  **Precise Analysis**





TP: True Positive
FP: False Positive

Reference: Yan, K.K., Fang, G., Bhardwaj, N., Alexander, R.P. and Gerstein, M., 2010. Comparing genomes to computer operating systems in terms of the topology and evolution of their regulatory control networks. *Proceedings of the National Academy of Sciences*, *107*(20), pp.9186-9191.

Approach: UBI bugs deTector

Flow-Sensitive Qualifier Inference

- Bottom-up, summary-based
- Inter-procedural
- flow-/filed-/context-sensitive
- Guidance

Scalable

Path-sensitive Symbolic Execution

- Path-sensitive

Precise

Approach: UBITect (Continue)

Type Qualifier

- **What?**

Type annotations

- **Why?**

Additional information
Ensure the correct use

- **Example:**

const int var;

Qualifier Analysis

Under-Constrained SE

Approach: UBITect (Continue)

Type Qualifier	Qualifier Analysis	Under-Constrained SE
init		
uninit		

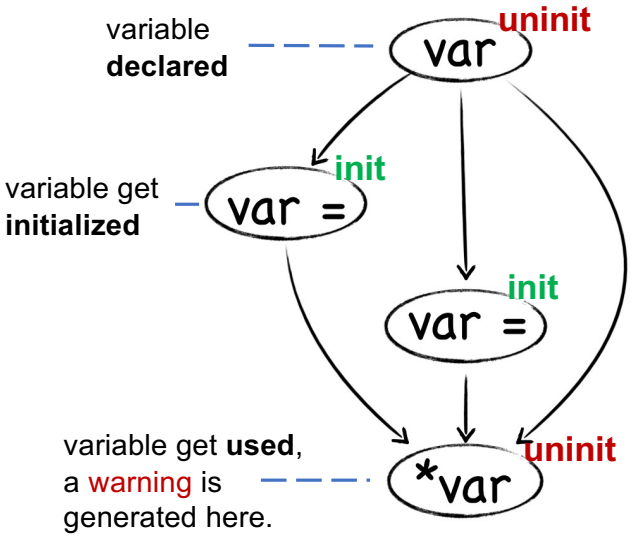
Approach: UBITect (Continue)

Type Qualifier

init

uninit

Qualifier Analysis



Under-Constrained SE

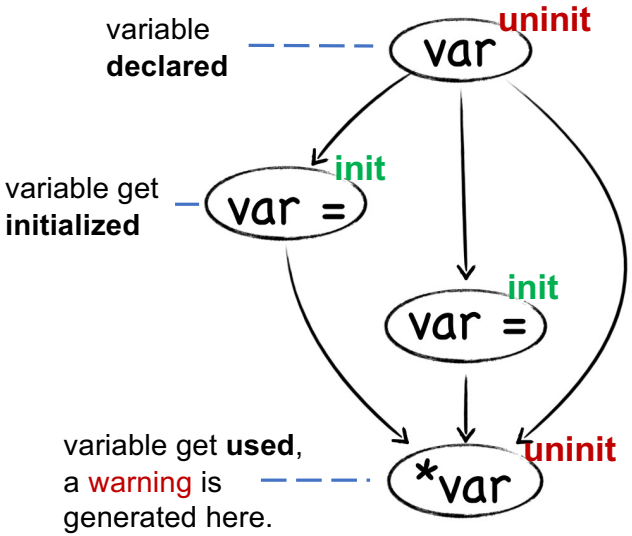
Approach: UBITect (Continue)

Type Qualifier

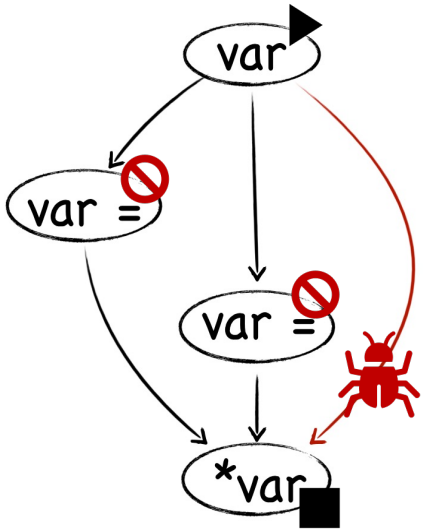
init

uninit

Qualifier Analysis

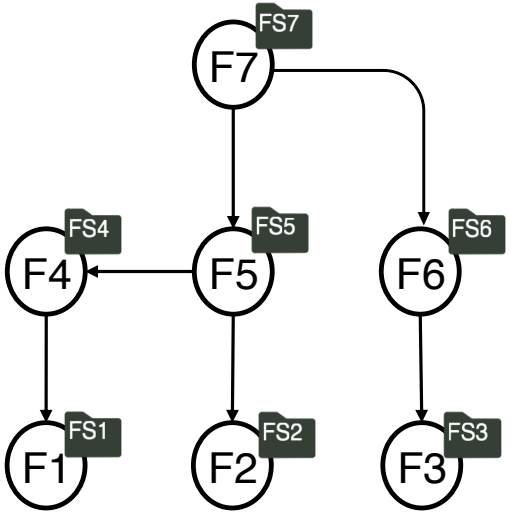


Under-Constrained SE



Approach: UBITect (Continue)

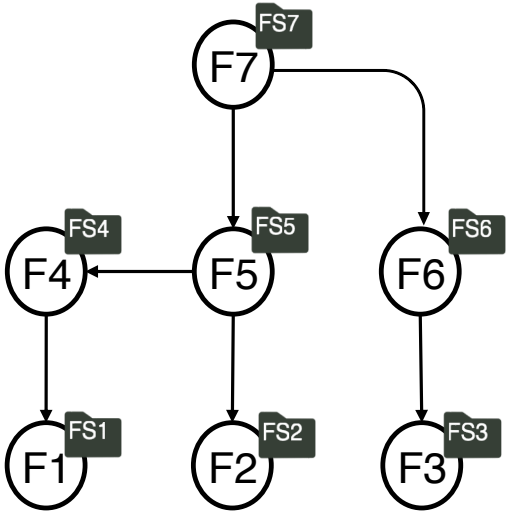
Bottom-up, summary-based



* FS: Function Summary

Approach: UBITect (Continue)

Bottom-up, summary based

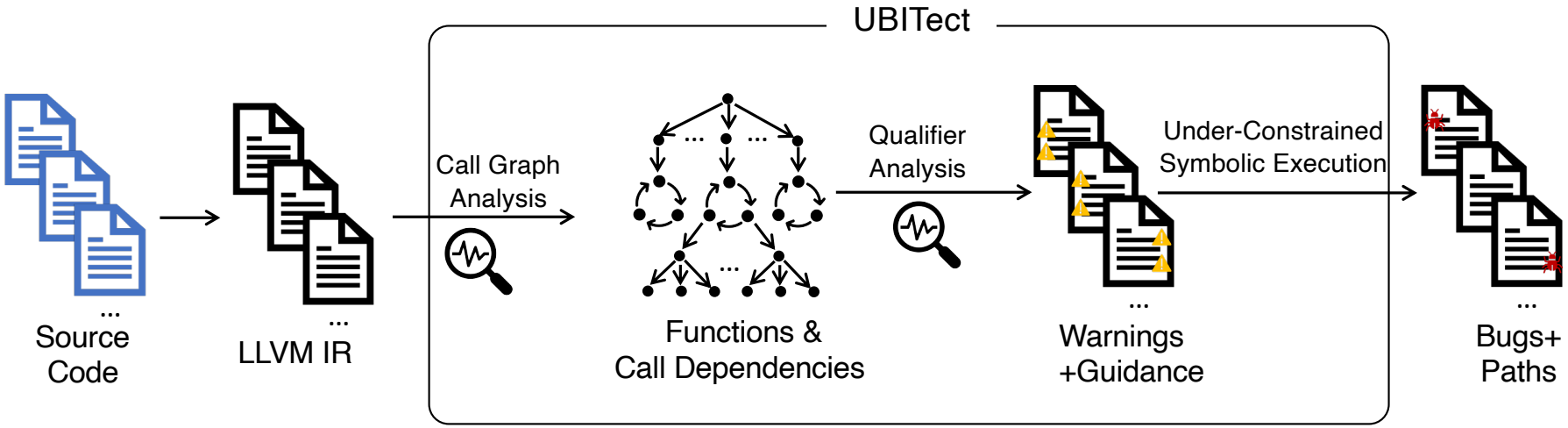


* FS: Function Summary

```
int F3(int a, int *pa)
{
        *pa = 4;
        if (a) {
                //do sth here
                return 0;
        } else
                return -1;
}
```

FS3	requirement	update
a	init	N/A
pa	init	N/A
pa_obj	N/A	init
ret	N/A	init

Putting them together



- Implementation:
  - LLVM 7.0.0
  - 13K+ LoC
  - SE Engine: KLEE

Evaluations

- Detecting Known UBI bugs
- Detecting New UBI bugs
- Comparison with cppcheck and Clang Static Analyzer

Evaluation I: Detecting Known UBI bugs

Table 2: Evaluation I: UBI bugs patched since 2013. All of the uninitialized variables are located on stack. UbiTect can successfully detect all of them.

Commit or CVE No	Type	UbiTect
bde6f9d	intra-procedural	Yes
1a92b2b	intra-procedural	Yes
8134233	inter-procedural	Yes
c94a3d4	inter-procedural	Yes
da5efff	inter-procedural	Yes
CVE 2010-2963	inter-procedural	Yes
7814657	inter-procedural	Yes
6fd4b15	inter-procedural	Yes

Evaluation II: Detecting New UBI bugs

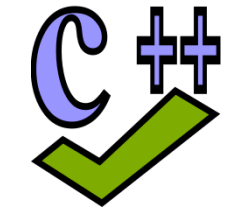
- Linux 4.14, allyesconfig
- 16163 files, 616893 functions
- 1 week analysis
- SE timeout to 120s
- SE memory out as 2GB

- 138 human verified bugs
- 118 unpatched
- 52 bugs confirmed

False Positive Reasons

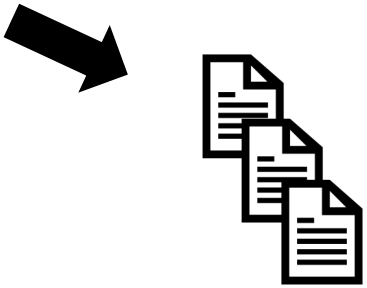
- Incomplete guidance
- Imprecise indirect call resolution
- LLVM optimizations
- Limitations of SE

Evaluation II: Detecting New UBI bugs



cppcheck

- Intra-procedural analysis



...

78 Files

164 bugs
2 TPs

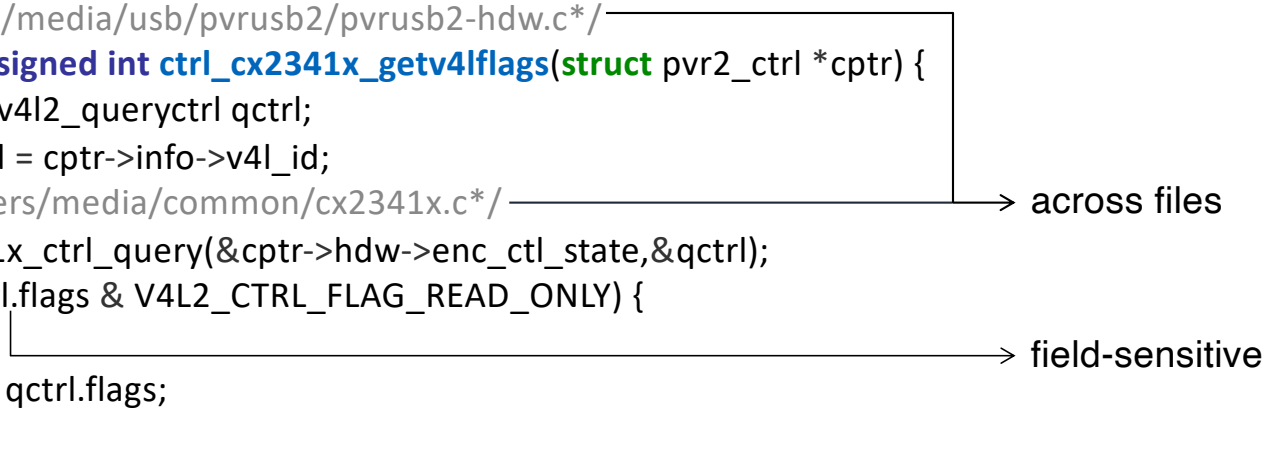
Clang Static Analyzer (CSA)

- Symbolic Execution in a single file

17 bugs

Case Study

```
1   /*drivers/media/usb/pvrusb2/pvrusb2-hdw.c*/
2   static unsigned int ctrl_cx2341x_getv4lflags(struct pvr2_ctrl *cptr) {
3       struct v4l2_queryctrl qctrl;
4       qctrl.id = cptr->info->v4l_id;
5       /*drivers/media/common/cx2341x.c*/
6       cx2341x_ctrl_query(&cptr->hdw->enc_ctl_state,&qctrl);
7       if (qctrl.flags & V4L2_CTRL_FLAG_READ_ONLY) {
8       }
9       return qctrl.flags;
10  }
```



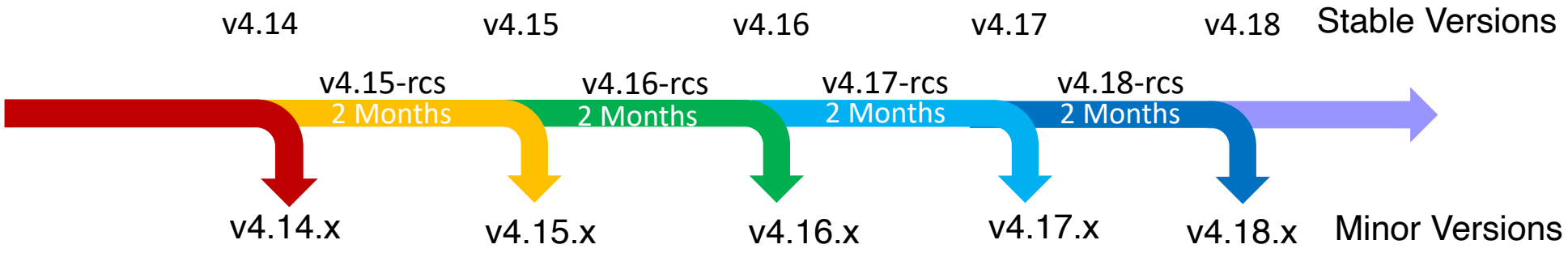
Conclusion

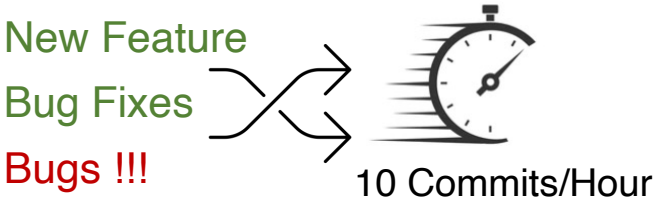
- UBI bugs cause critical security issues and zeroing the variable cannot fully mitigate them.
- UBITect: A precise and scalable tool to detect UBI bugs in Linux kernel
- 52 new bugs have been confirmed in Linux

Progressive Scrutiny: Incremental Detection of UBI bugs in the Linux Kernel

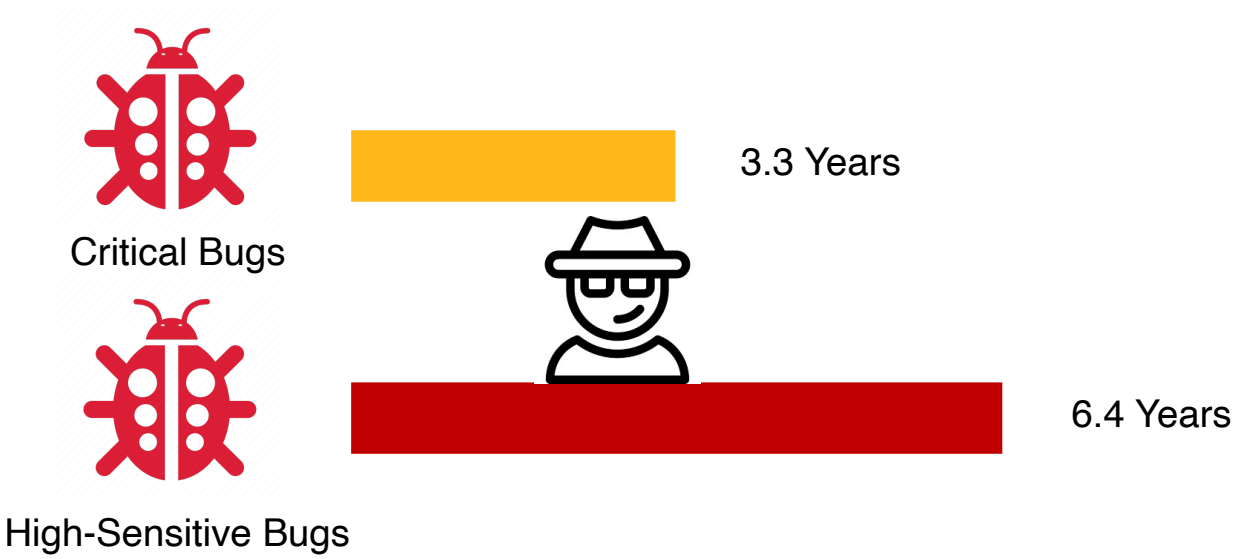
Yizhuo Zhai, Yu Hao, Zheng Zhang, Weiteng Chen, Guoren Li, Zhiyun Qian, Chengyu Song, Manu Sridharan, Srikanth V. Krishnamurthy, Trent Jaeger, Paul Yu

Background: Rapid Linux Kernel Development Cycle



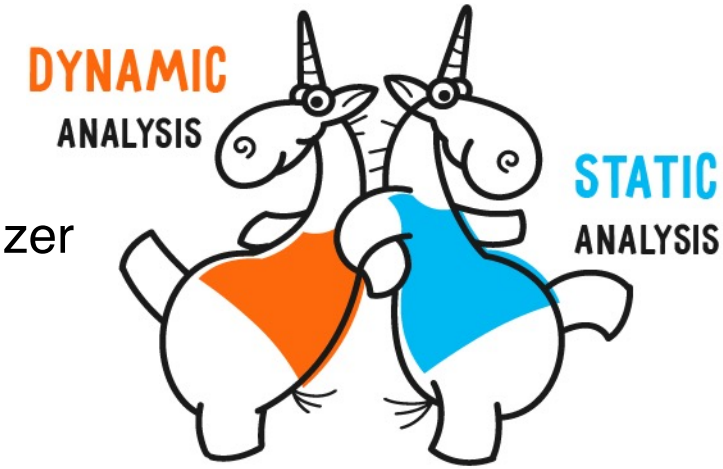


Background: Security Issues in Linux Kernel



Existing Effort

Fuzzing + Sanitizer



Specified bugs

Framework for specified modules.

Existing Effort - Limitations

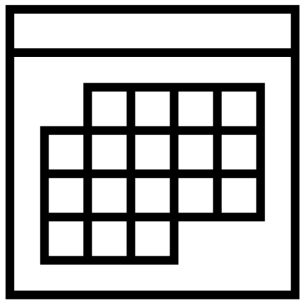
DYNAMIC ANALYSIS

Key
Not instrumented
Conditionalized out
Executed
Not executed

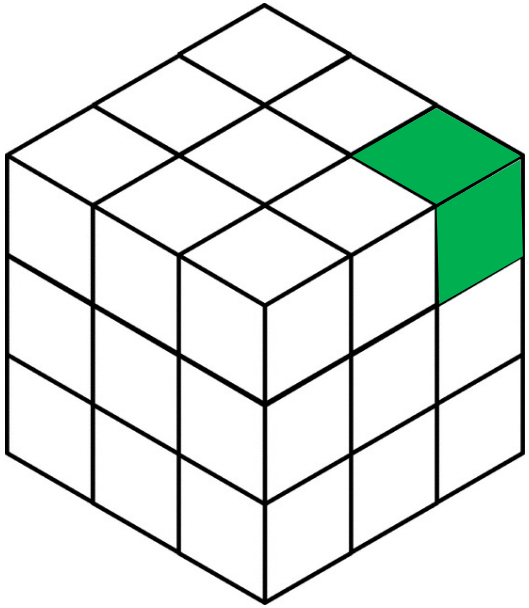
Both branches taken
One branch taken
Neither branch taken

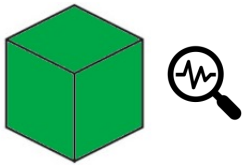
```
 1  (declaim (optimize sb-cover:store-cover
 2
 3  (defun test (n)
 4    (when (zerop n)
 5      (if (eql n 0)
 6          (print 'zero)
 7          (if (eql n 0.0)
 8              (print 'single-fp-zero)
 9              (print 'double-fp-zero))))
10    (when (minusp n)
11      (print 'negative))
12    (when (plusp n)
13      (tagbody
14         (print 'positive)
15         (go end)
16         (print 'dummy)
17         end)))
```

STATIC ANALYSIS



Observation

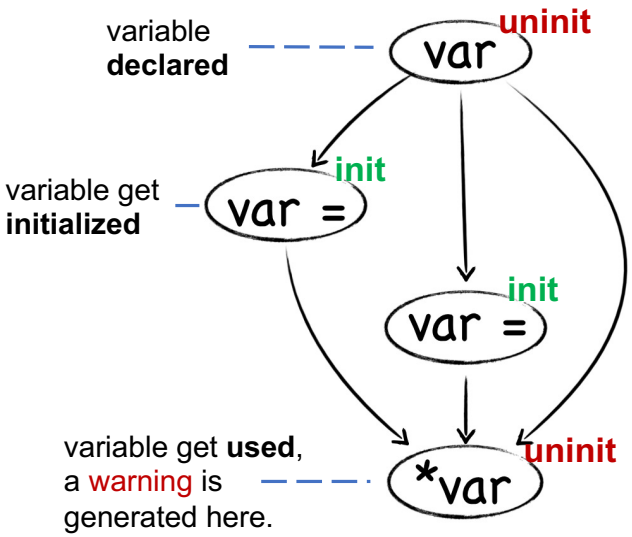




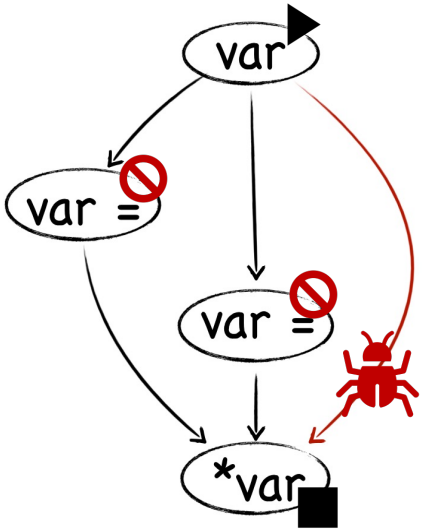
- Quicker Turnaround time
- Test proposed patches
- Exhaustive coverage

Background: UBITect (FSE'20)

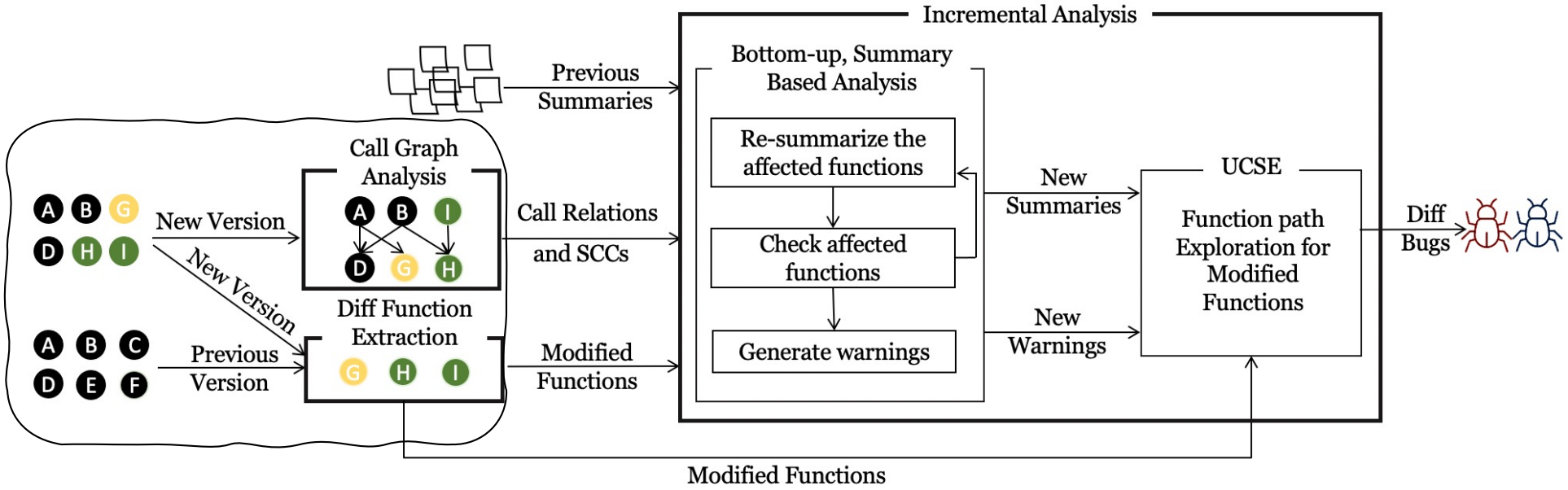
Qualifier Analysis



Under-Constraint SE



IncreLux Workflow



* SCC: Strongly Connected Component

Evaluations

- V4.14 as the baseline
- Stable versions: v4.15-rcs-v4.19, v5.4, v5.9
- Minor versions: v4.14.20 – v4.14.220
  v4.15.1-v4.15.18
- Per patch analysis
- Equivalence Analysis

- Speed Improvement
- Time Breakdown
- Bug Finding Results
- Patch Identification Results.

Evaluation – Time Speedup

TABLE II: Incremental analysis results for mainline versions v4.14 to v4.16. Please refer to the Appendix for the full table from v4.14 to v5.9. **T(h)** : Total analysis time in hours. **SU** : Speedup compared to exhaustive analysis of v4.14. **FM** : Number of functions modified compared to the immediate. predecessor. **FR** : Number of functions (re-)analyzed in this version. **Warn** : Number of Warnings reported in the current version. **Disappearing** : Number of warnings that disappear in the current version (compared with the last analyzed version). **Equal** : Number of warnings that remain in the current version compared to the immediate previous analyzed version. **New** : Number of warnings newly introduced in the current version compared with the last analyzed version. **SE-New** : New bugs confirmed by SE that are introduced in the new version.

versions	T(h)	SU	FM	FR	Warn	Disappearing	Remaining	New	SE-New
v4.14	106.75	N/A	N/A	629862	103616	N/A	N/A	N/A	622
v4.15-rc1	28.26	3.78	23941	42548	21190	4325	99291	4979	69
v4.15-rc2	2.91	36.74	719	2617	2190	422	103848	211	0
v4.15-rc3	2.27	47.04	656	3084	1937	301	103758	238	0
v4.15-rc4	1.13	94.52	332	1268	718	116	103880	70	0
v4.15-rc5	1.28	83.31	329	1793	1339	207	103743	331	0
v4.15-rc6	1.15	92.6	273	1761	1282	96	103978	96	0
v4.15-rc7	0.43	248.74	101	403	263	21	104053	27	0
v4.15-rc8	1.33	80.15	243	1707	1305	114	103966	151	0
v4.15-rc9	0.63	169.82	217	1031	696	49	104068	48	0
v4.15	0.13	800.63	122	262	215	36	104080	48	2
v4.16-rc1	26.77	3.99	21251	52151	26922	4240	99888	4742	55
v4.16-rc2	0.24	453.72	220	521	342	10	104620	25	0
v4.16-rc3	0.7	151.6	434	1012	713	136	104509	77	1
v4.16-rc4	3.39	31.48	278	7398	3446	502	104084	509	4
v4.16-rc5	0.76	141.23	422	979	598	80	104513	76	0
v4.16-rc6	0.26	415.01	144	401	279	15	104574	27	0
v4.16-rc7	0.93	115.2	498	1543	819	107	104494	190	2
v4.16	0.33	318.92	183	535	278	18	104666	15	0
v4.17rc1-v4.19	...	...	...	...	...	...	...	...	...
v5.4	97.9	1.09	99370	205327	158018	43762	69652	88366	N/A
v5.9	99.65	1.07	91741	197413	152746	40720	85425	67321	N/A

- Stable: 3.78x - 800x
- V4.14.20x: 11.04 - 32.29
- V4.15.z: 32x - 800x

Evaluations - New Bugs

- 44 bug report sampled
- 22 TP (FP: 50%)
- 17 can be triggered

TABLE VI: Bugs introduced in the new code, in the column of the **Patch**, **E** means that the patch is not easy to draft; here, we e-mail the bug to the maintainer. **A** means that the patch that we submitted was applied; **C** means that our bug was confirmed by the maintainers. **F** means that the bug has been fixed in the latest version of the kernel by others. **IL** : Information Leakage. **MC** :Memory Corruption. **B** :Benign. **HWCC** :Hardware configuration corruptions.

Sub-System	Module	Variable	Line No.	Intro.	Patch	Impact
Input/hideep	hideep.c	unmask_code	380	v4.15-rc1	A	IL
atomisp	atomisp-mt9m114.c	retvalue	1552	v4.15-rc1	A	MC
drm/nouveau	ioctl.c	type	269	v4.15-rc1	S	B
media/imx274	imx274.c	err	659	v4.15-rc1	F	B
net/mlx25	en_common.c	min_inline_mode	180	v4.15-rc1	F	B
net/mlx5e	en_dcbnl.c	params→ tx_min_inline_mode	989	v4.15-rc1	F	B
xfs	xfs_bmap.c	got.br_startoff	4868	v4.15-rc1	E	MC
xfs	xfs_bmap.c	s	1521	v4.15-rc1	E	MC
iio/adc	stm32-dfsdm-adc.c	status	866	v4.16-rc1	C	MC
iio/adc	stm32-dfsdm-adc.c	int_en	873	v4.16-rc1	C	MC
iio/adc	qcom-pm8xxx-xoadc.c	ch	599	v4.16-rc1	F	MC
net: mscc	ocelot.c	val	365	4.18-rc1	F	MC
media: davinci_vpfe	dm365_isif.bc	format.pixelformat	234	4.18-rc1	F	HWCC
display	dc_link.c	old_downspread.raw	1259	4.18-rc1	A	HWCC
display	dc_link_dp.c	training_rd_interval	61	4.18-rc1	F	HWCC
net:mscc	ocelot.c	val	34	4.18-rc1	E	MC
scsi: sd	sd.c	sshdr.asc	2390	v4.19-rc1	E	B

Bug Lifetime – Case Study

```
1   /* drivers/media/i2c/imx274.c
2   * uninteresting code lines are omitted */
3   static int imx274_regmap_util_write_table_8 ()
4   {
5       int err;
6       if (range_count == 1)
7                   err = regmap_write(regmap,
8                           range_start, range_vals[0]);
9       else if (range_count > 1)
10                  err = regmap_bulk_write(regmap, range_start,
11                                  &range_vals[0],
12                                  range_count);
13  +    else
14  +        err = 0;
15      if (err) {
16          return err;
17      }
18  }
```

Fig. 6: The patch that fixed the previous bug; this bug was introduced in v4.15-rc1 and the patch was applied in v4.16-rc1. By continuously tracking the bug, INCRELUX could find both the bug upon introduction, and the time of the bug disapperance. If we use this patch as the input for the incremental analysis, the disapperance of the bug indicates that this commit was related to a bug fix.

Conclusion

- IncreLux : A framework for principled incremental analysis of the Linux kernel.
- Dramatic speed ups compared to today's expensive whole-kernel analysis.
- Fit into the kernel development cycle.
- Effectively identify bugs and bug fixes.
- https://github.com/seclab- ucr/IncreLux.git

Q & A