ARTIFACT EVALUATED usenix ASSOCIATION AVAILABLE

ARTIFACT EVALUATED usenix ASSOCIATION FUNCTIONAL

ARTIFACT EVALUATED usenix ASSOCIATION REPRODUCED

Mitigating Security Risks in Linux with KLAUS

A Method for Evaluating Patch Correctness

Yuhang Wu, Zhenpeng Lin, Yueqi Chen, Dang K Le,
Dongliang Mu, Xinyu Xing

Linux Patching in the Fuzzing Era: Navigating a Bug Surge

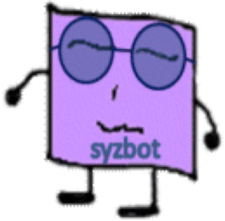
Syzbot has verified ~ **5000** valid bug reports in 5 years

Bug Fix

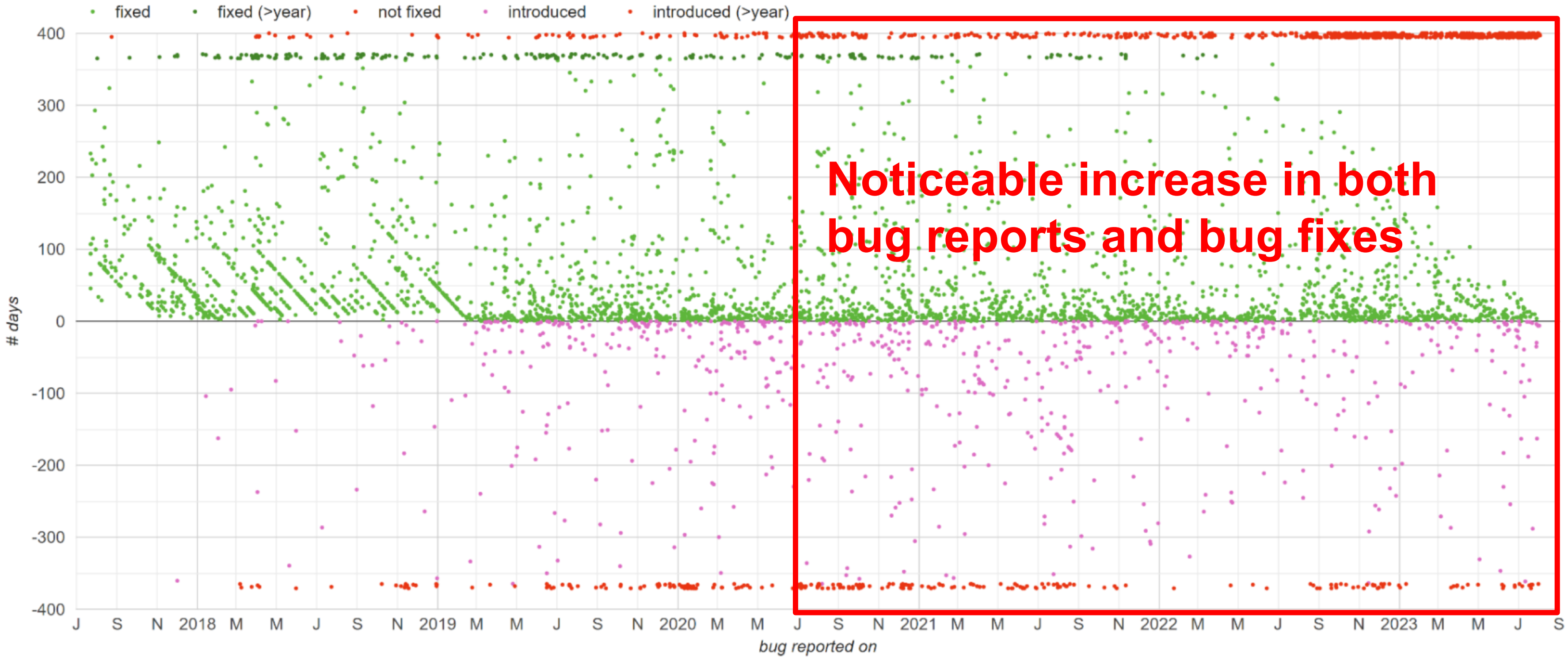
~ **4600** patches were developed to fix these bugs

Code Review

User Report



Linux Patching in the Fuzzing Era: Navigating a Bug Surge



Noticeable increase in both bug reports and bug fixes

Bug lifetimes of Linux kernel (https://syzkaller.appspot.com/upstream/graph/lifetimes)

Linux Patching in the Fuzzing Era: Navigating a Bug Surge

More bugs

More experts

More patches

… but

~ **6%** of the Linux kernel patches are **incorrect**

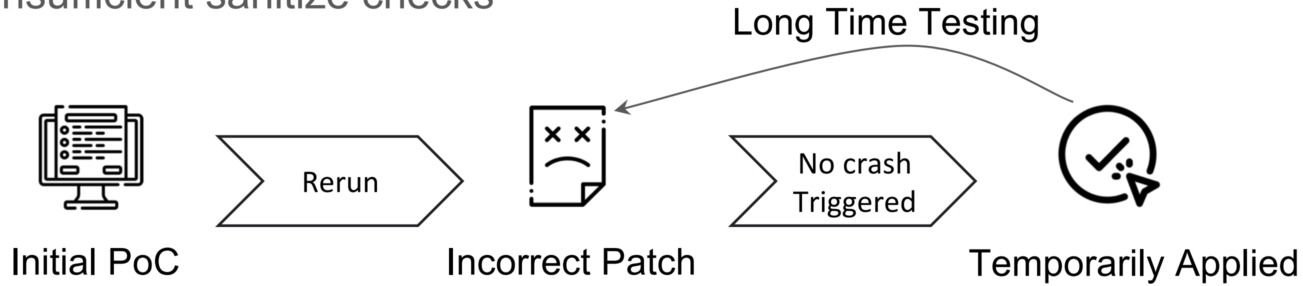
The Pitfalls of Incorrect Patching

Root Causes of Incorrect Patches

- Lack of understanding of the code
- Misdiagnosis of the root cause of the bug

Common Patching Mistakes

- Not considering all potential branches or pathways that lead to the patched site
- Adding insufficient sanitize checks



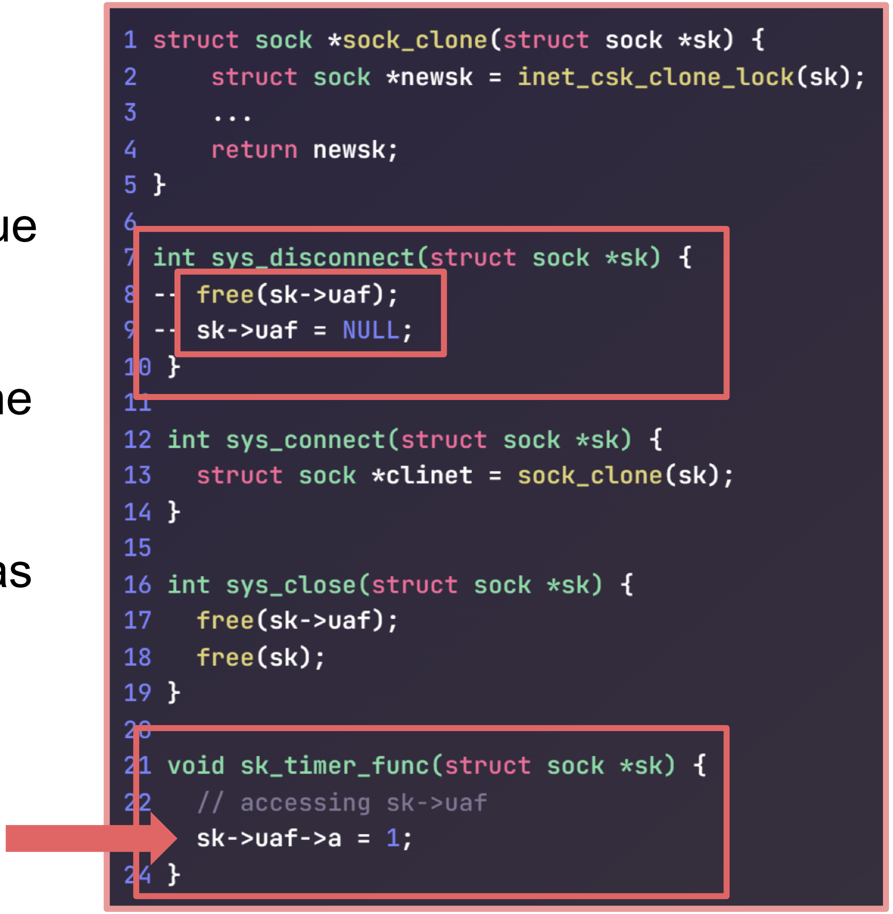
A Real-World Example

Initial UAF: Dangling pointer in timer queue after sys_disconnect

Incorrect patch: line 8, 9 are deleted in the patch

New UAF: Dangling pointer left after sk has been cloned and sys_close

Reason: sk->uaf is not set to NULL

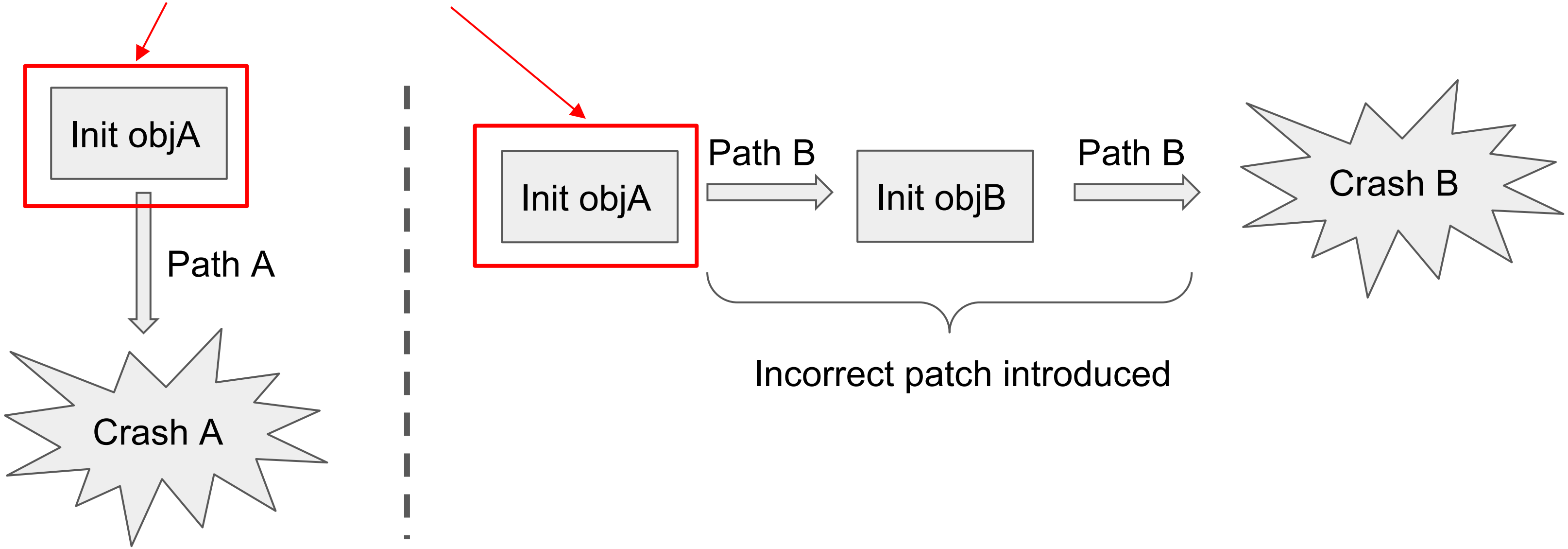


```
1 struct sock *sock_clone(struct sock *sk) {
2     struct sock *newsk = inet_csk_clone_lock(sk);
3     ...
4     return newsk;
5 }
6
7 int sys_disconnect(struct sock *sk) {
8 --  free(sk->uaf);
9 --  sk->uaf = NULL;
10 }
11
12 int sys_connect(struct sock *sk) {
13   struct sock *clinet = sock_clone(sk);
14 }
15
16 int sys_close(struct sock *sk) {
17   free(sk->uaf);
18   free(sk);
19 }
20
21 void sk_timer_func(struct sock *sk) {
22   // accessing sk->uaf
23   sk->uaf->a = 1;
24 }
```

The Birth of AWRP (Altered Write-Read Pairs)

- Manual analysis of 182 incorrect patches in Linux kernel

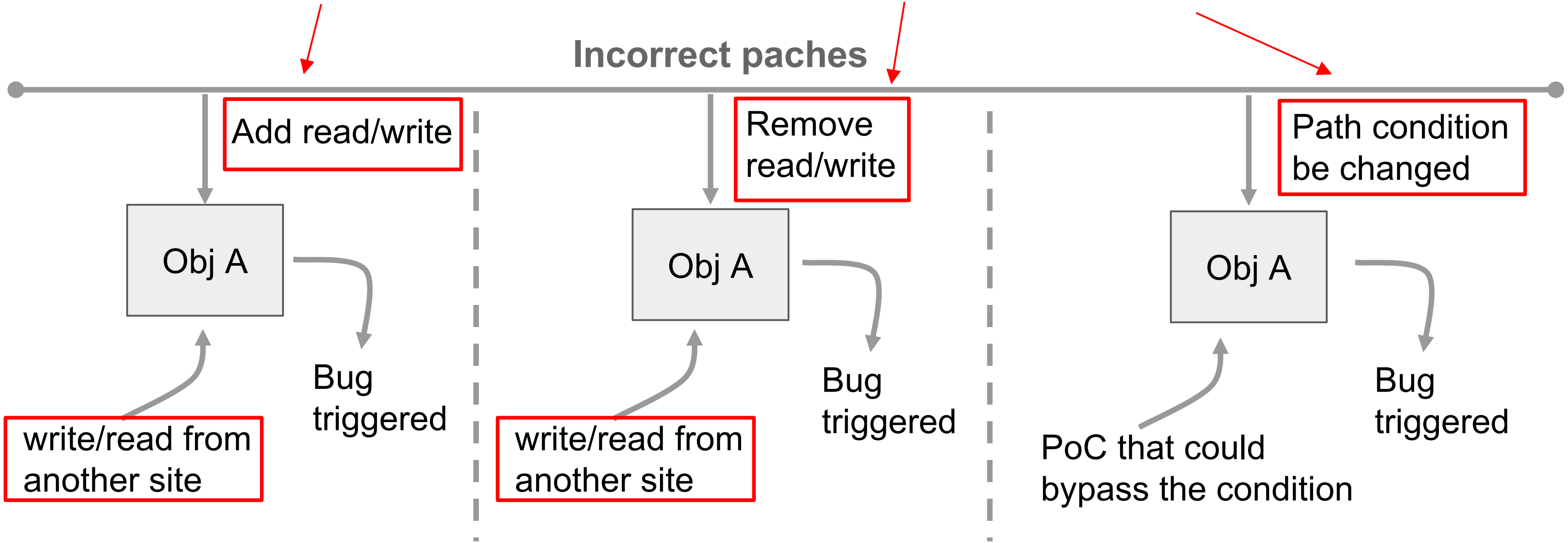
Observation 1: Old and new vulnerabilities share similar contexts



The Birth of AWRP (Altered Write-Read Pairs)

- Manual analysis of 182 incorrect patches in Linux kernel

Observation 2: New vulnerability results from Altered Write-Read Pairs (AWRP)



Incorrect paches

Add read/write

Obj A

write/read from another site

Bug triggered

Remove read/write

Obj A

write/read from another site

Bug triggered

Path condition be changed

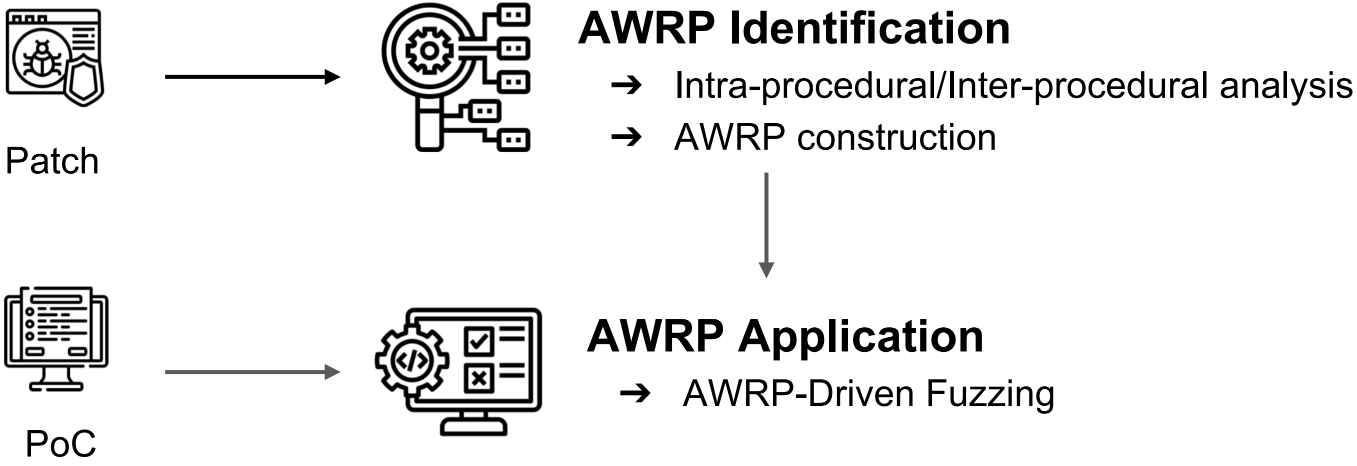
Obj A

PoC that could bypass the condition

Bug triggered

KLAUS: A Framework to Identify and Utilize AWRP

- ❖ The AWRP mechanism can provide a method for analyzing patches



Patch

AWRP Identification

- ➔ Intra-procedural/Inter-procedural analysis
- ➔ AWRP construction

PoC

AWRP Application

- ➔ AWRP-Driven Fuzzing

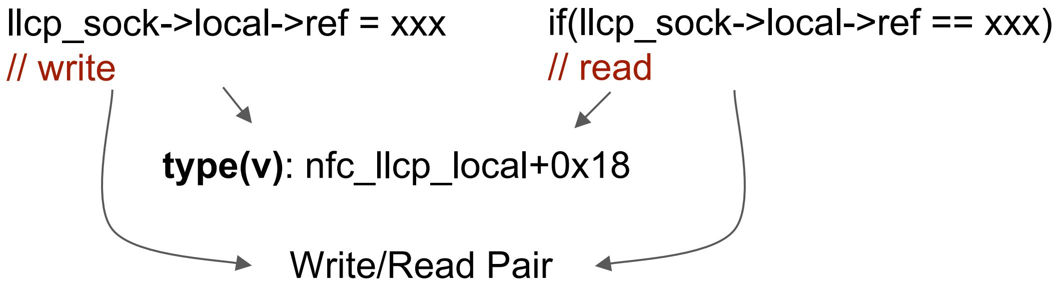
AWRP Identification: The Abstract State

Variables in Kernel: $V = \{v_1, \cdots , v_n\}$

AWRP Identity: type(v)

- Local Variables: type(v) = function_name + stack_offset
- Global or Static Variables: type(v) = module_name + variable_name
- Heap Objects:
  - Individual Object: type(v) = object_type_name
  - Field of an Object: type(v) = object_type_name + field_offset

```
struct nfc_llcp_local {
    struct list_head list;
    struct nfc_dev *dev;
    struct kref ref;
    …
}
```



llcp_sock->local->ref = xxx
// write

if(llcp_sock->local->ref == xxx)
// read

type(v): nfc_llcp_local+0x18

Write/Read Pair

AWRP Identification: The Abstract State

Variables in Kernel: $V = \{v_1, \cdots, v_n\}$

AWRP Info: value(v)

- value(v) = {⟨cond, content⟩} : under the condition ***cond***, the value of v is equal to ***content***

```
if (llcp_sock->ssap == LLCP_SAP_MAX) {
    llcp_sock->sock = NULL;
}
```

value(v): $\{\langle \underbrace{\text{'llcp\_sock ->ssap == LLCP\_SAP\_MAX', 'NULL'}}_{\text{Symbolic Strings}} \rangle\}$

The Abstract State: S = {cond,⟨type(v1), value(v1)⟩,⋯ ,⟨type(vn), value(vn)⟩}

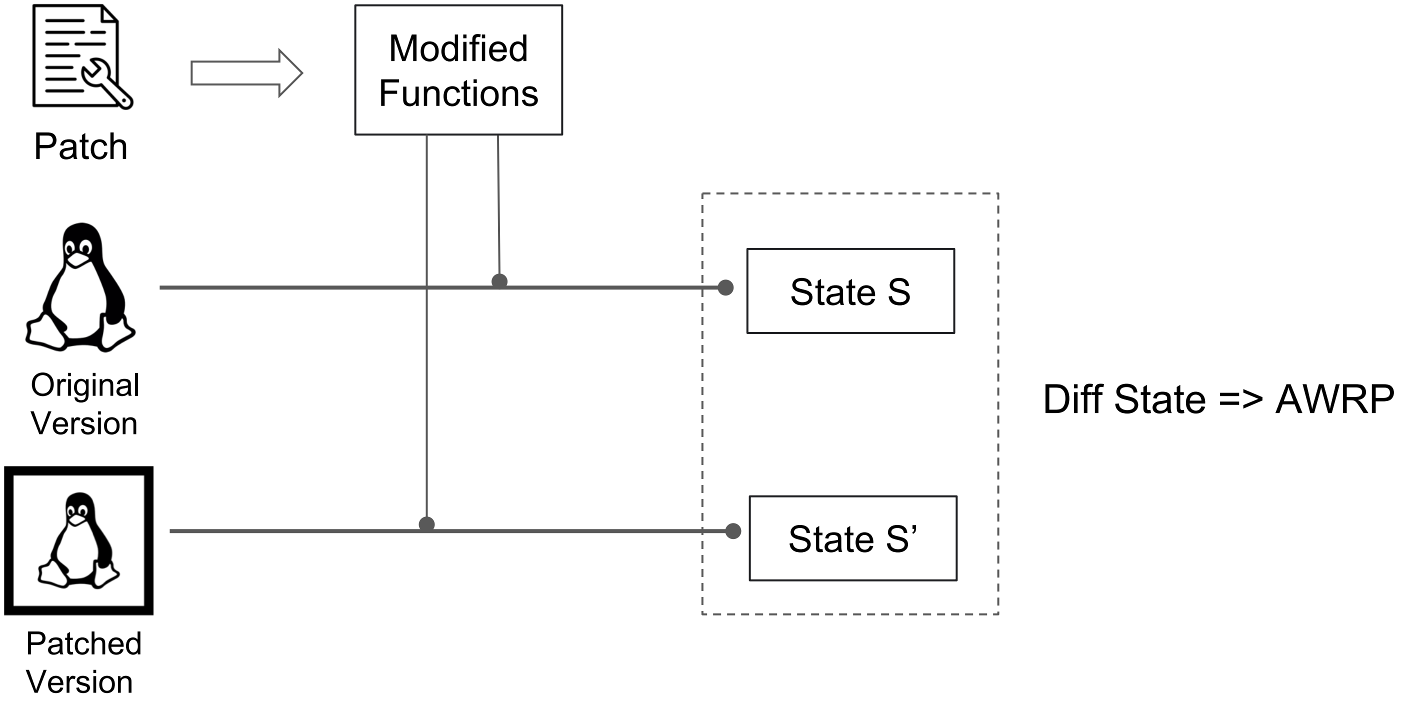
AWRP Identification: The Transfer Function

The Abstract State: S = {cond,⟨type(v1), value(v1)⟩,··· ,⟨type(vn), value(vn)⟩}

Transfer(S,inst): The impact of executing inst in the state S

- ❖ The inst writes to a variable v
  - ➢ replace value(v) by a new <cond, content>
- ❖ The inst casts variable v from one type to another type
  - ➢ update type(v) to a new one
- ❖ The inst is a conditional jump
  - ➢ cond in S is conjuncted with the jump condition

AWRP Identification: Intra/Inter-procedural Analysis



The Application of AWRP: AWRP-driven Fuzzer

- Developed based on Syzkaller.
- Prefer to cover more locations where AWRP is used
- Instrument the basic blocks on the essential route leading to AWRP

Evaluation

- ❖ Used 23 ground-truth cases from syzkaller community
- ❖ Same initial seed & time (3 days) & rounds (5) & environment
- ❖ Compared with Syzkaller

KLAUS found **23**/23 incorrect patches

Syzkaller found **13**/23 incorrect patches

KLAUS triggers crashes caused by incorrect patches faster than **Syzkaller** in **12**/13 cases

KLAUS found **30** new incorrect patches in the wild! The community has confirmed and fixed **25** of these patches

Takeaways

- The AWRP method provides a framework for patch analysis
- KLAUS, utilizing the AWRP, can better detect incorrect patches
- We look forward to more research on AWRP

Source Code: https://github.com/wupco/KLAUS

yuhang.wu@northwestern.edu

@wupco1996