UC RIVERSIDE

What Do They Fix? LLM-Aided Categorization of Security Patches for Critical Memory Bugs

Motivation

- Memory Bugs Are Dangerous and Prevalent
  - Memory corruption (e.g., Use-After-Free, Out-of-Bounds) enables privilege escalation
  - 90% of recent public kernel exploits leverage such vulenrabilities
- Delayed Patch Adoption = Real-World Risk
  - Weeks or months gap between upstream Linux kernel and downstream (e.g., Ubuntu)
  - Attackers analyze patches and develop exploits before downstream kernels patch them
- CVE Assignment is Delayed, Incomplete, or Missing
- Goal: Accurately identify UAF/OOB/non-UAF-OOB patches, even without CVEs

Limitations of prior works

- Rule-Based (e.g., SID):
  - Pattern-based, high false positive rate.
- ML-on-Diff (e.g., TreeVul):
  - Ignores commit messages, lacks code context awareness.
- ML-on-Slice (e.g., ColeFunda):
  - Slicing algorithm inducing too much noise.
- All prior tools miss significant portions of real-world patches.

UC RIVERSIDE

Key observation 1:

Untapped potential exists in commit descriptions

ksmbd: fix use-after-free bug in smb2_tree_disconect
smb2_tree_disconnect() freed the struct ksmbd_tree_connect,but it left the dangling pointer. It can be accessed again under compound requests.

Explicit indication of bug type

ipv6: raw: Deduct extension header length in rawv6_push_-pending_frames
The total cork length created by ip6_append_data includes extension headers, so we must exclude them when comparing them against the IPV6_CHECKSUM offset which does not include extension headers.

Implicit indication of bug type

Key observation 2:

Code diff spatterns cannot effectively identify bug types in prior research

- Some patches do not follow common patch patterns.

```
@@ -316,6 +316,11 @@ int
st21nfca_connectivity_event_received(struct nfc_hci_dev *
hdev, u8 host,
          return -ENOMEM;

       transaction->aid_len = skb->data[1];

+
+      /* Checking if the length of the AID is valid */
+      if (transaction->aid_len > sizeof(transaction->aid))
+         return -EINVAL;
+

       memcpy(transaction->aid, &skb->data[2],
              transaction->aid_len);
```

Add bounds checking
(Supported by SID)

```
@@ -567,12 +567,11 @@ static void nfsd_init_dirlist_pages(
struct svc_rqst *rqstp,
    struct xdr_stream *xdr = &resp->xdr;

-   count = clamp(count, (u32)(XDR_UNIT * 2),
svc_max_payload(rqstp));
    memset(buf, 0, sizeof(*buf));

  /* Reserve room for the NULL ptr & eof flag (-2 words) */
-   buf->buflen = count - XDR_UNIT * 2;
+   buf->buflen = clamp(count, (u32)(XDR_UNIT * 2), (u32)
PAGE_SIZE);
+   buf->buflen -= XDR_UNIT * 2;

     buf->pages = rqstp->rq_next_page;
```

Recalculating size
(Not supported by SID)

UC RIVERSIDE

Key observation 2:

Code diff patterns cannot effectively identify bug types in prior research

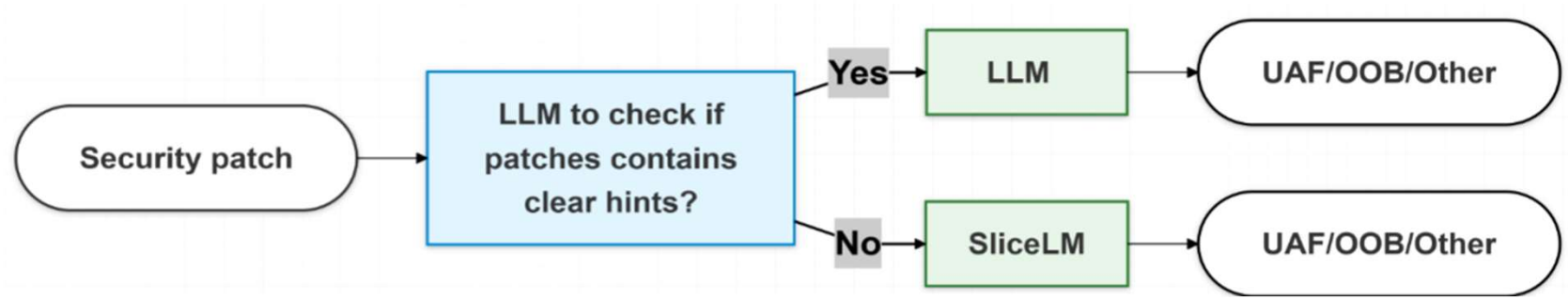
- Code diff lacks enough context information
- Traditional slicing algorithm induce bloated code blocks

```
--- a/arch/x86/kvm/i8254.c
+++ b/arch/x86/kvm/i8254.c
@@ -465,6 +465,9 @@ static int pit_ioport_read(struct kvm_io
                return -EOPNOTSUPP;

        addr &= KVM_PIT_CHANNEL_MASK;
+       if (addr == 3)
+               return 0;
+
        s = &pit_state->channels[addr];

        mutex_lock(&pit_state->lock);
```

Lacks contextual information in the diff

Overall design

- Our pipeline:
  - Classify patches into with/without clear hints.
  - With hints: leverage LLM to classify
  - Without hints: leverage our SliceLM to perform code slicing and classification

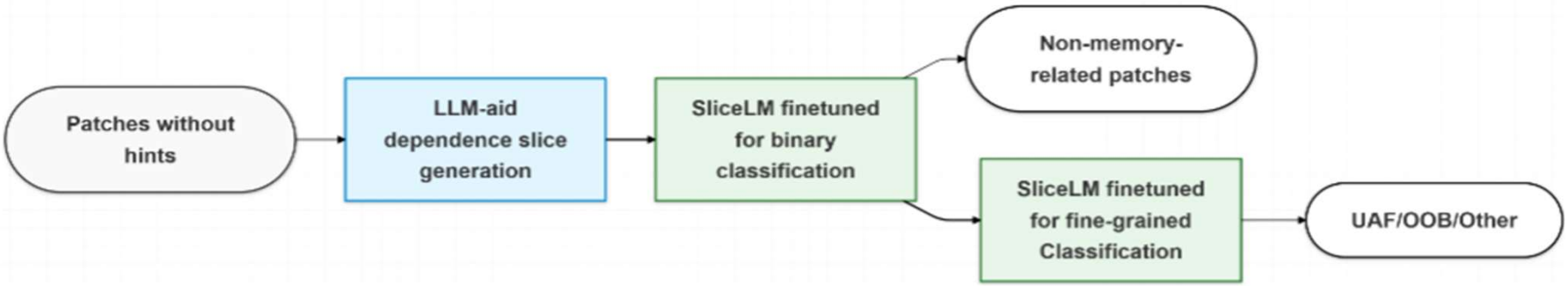


Patch with hints: classifying patches with an LLM

- LLMs are particularly well-suited for
  - Analyze textual information in patch messages
  - Combine textual information with code patterns
  - Unedrstand new patterns through few-shot learning
- We present both UAF/OOB examples and non-UAF-OOB examples as samples to perform few-shot learning

Patch without hints: SLICELM pipeline

- Pipeline
  - Selective slicing aided by LLM
  - Custom SliceLM model to perform classification



Selective slicing

- Criterion variables: The variables involved in the change sites
- Focus data dependency on criterion variables
- Traditional slicing induce 40+ lines for this case

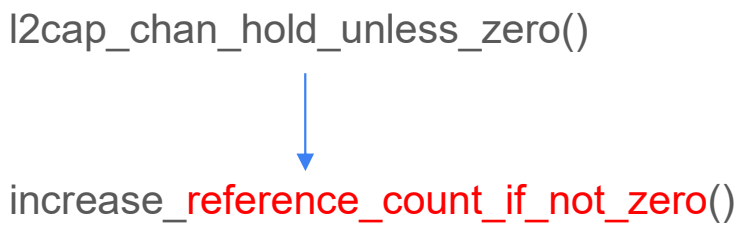
```
--- a/arch/x86/kvm/i8254.c
+++ b/arch/x86/kvm/i8254.c
@@ -465,6 +465,9 @@ static int pit_ioport_read(struct kvm_io
                return -EOPNOTSUPP;

        addr &= KVM_PIT_CHANNEL_MASK;
+       if (addr == 3)
+               return 0;
+
        s = &pit_state->channels[addr];

        mutex_lock(&pit_state->lock);
```



```
KVM: PIT: control word is write-only

PIT control word (address 0x43) is write-only, reads are undefined.

@@ -467,6 +467,9 @@ static int pit_ioport_read(struct kvm_io_device *
this,
static int pit_ioport_read(struct kvm_io_device *this, gpa_t addr,
int len, void *data)
    ...
    addr &= KVM_PIT_CHANNEL_MASK;
+   if (addr == 3)
+       return 0;
    ...(omitted 24 lines)
    } else {
        switch (s->read_state) {
        ...(omitted 14 lines)
        case RW_STATE_WORD1:
            count = pit_get_count(pit, addr);
            ret = (count >> 8) & 0xff;
            s->read_state = RW_STATE_WORD0;
            break;
        }
    }
    if (len > sizeof(ret))
        len = sizeof(ret);
    memcpy(data, (char *)&ret, len);
```

LLM-aided slicing: function renaming

- Inter-procedural slicing introduces noise
- Intra-procedural slicing lacks contextual information
- Solution: function renaming for intra-procedural slicing
- Leverage LLM to to so:



LLM-aided slicing: redundant code pruning

- Not all changes in a patch is fixing the vulnerability
- LLM to identify two patterns:
  - Redundant changes: e.g., adjusting function call due to parameter changs
  - Refactoring: e.g., move if-check into a function without changing the actual behavior

```
@@ btrfs_ioctl_resize:
-device = btrfs_find_device(fs_info->fs_devices, devid, NULL
, NULL);
+device = btrfs_find_device(fs_info->fs_devices, devid, NULL
, NULL, true);

@@ btrfs_scrub_dev:
-dev = btrfs_find_device(fs_info->fs_devices, devid, NULL,
NULL);
+dev = btrfs_find_device(fs_info->fs_devices, devid, NULL,
NULL, true);

(11 other instances of changes similar to the above)

@@ device_list_add:
-device = find_device(fs_devices, devid,disk_super->dev_item
.uuid);
+device = btrfs_find_device(fs_devices, devid,disk_super->
dev_item.uuid, NULL, false);

-find_device() // function is totally removed

+btrfs_find_device() // add a bool argument and
corresponding operation for it

 ...
```

Sample Redundant Changes

UC RIVERSIDE

SliceLM: patch classification on sliced code

- Training
  - BERT-based model
  - Pretrained from scratch on Linux kernel history (~1.1M commits), learning dependence-based relationships in slices/diffs
- Multi-class fine-tuning:
  - 10,540 samples of UAF / OOB / other memory-corruption cases
  - Ground truth from keywords in commit message
- Binary fine-tuning
  - Positive sample: dataset for multi-class fine-tuning
  - Negative sample: non-security commits from PatchDB

Evaluation

- 946 CVE-labeled Linux patches (2015–2024)

Task	Accuracy	Precision	F1-score	FP Rate	FN Rate
Overall pipeline	**87.4%**	87.7%	0.875	5.4%	10.0%
— **Classification on patches with hints**	**90.3%**	90.7%	0.904	4.0%	6.6%
— **Classification on patches without hints**	**77.9%**	78.4%	0.776	13.4%	24.1%
| **Binary classification**	84.8%	85.8%	0.847	13.9%	16.5%
| **Fine-grained classification**	81.8%	83.5%	0.821	8.8%	18.8%

- 5,140 recent patches from Linux 6.6 LTS (2024)
  - 90/111 verified UAF/OOB cases
  - 2% false negative rate
  - Two PoCs and one new control-flow hijack exploit

Comparative study

- DualLM outperforming SID and TreeVul by 23.6% and 21.2% in accuracy, and 0.329 and 0.216 in F1-score.

Method	Accuracy	Precision	F1-score	FP rate	FN rate
SID	63.8%	77.7%	0.546	21.2%	67.8%
DualLM	87.4%	87.7%	0.875	5.4%	10.0%

Method	Accuracy	Precision	F1-score	FP rate	FN rate
TreeVul	65.6%	72.5%	0.653	21.6%	23.0%
DualLM	86.8%	87.5%	0.869	5.1%	9.3%

UC RIVERSIDE

Generalization to other bug types and beyond Linux

- DUALLM achieves an 80.35% accuracy and an F1-score of 0.801
- DUALLM achieving F1-scores of 0.832 and 0.834 on FFmpeg and OpenSSL respectively

Case study – UAF patch (no explicit hints)

```
io_uring: fix xa_alloc_cycle() error return value check

We currently check for ret != 0 to indicate error, but '1'
is a valid return and just indicates that the allocation
succeeded with a wrap. Correct the check to be for < 0, like
 it was before the xarray conversion.

@@ -9843,10 +9843,11 @@ static int io_register_personality(
struct io_ring_ctx *ctx)
    ret = xa_alloc_cyclic(&ctx->personalities, &id, (void *)
creds,
          XA_LIMIT(0, USHRT_MAX), &ctx->pers_next, GFP_KERNEL
);
-   if (!ret)
-       return id;
-   put_cred(creds);
-   return ret;
+   if (ret < 0) {
+       put_cred(creds);
+       return ret;
+   }
+   return id;
  }
```

- put_cred(creds) is wrongly called when ret is ‘1’, causing unexpected reference count decrement resulting for UAF
- Our function renaming approach rename put_cred() into decrease_credential_reference_count() to emphase its conection to the decrement
- Without this renaming, we found that our model would misclassify the case as OOB instead
- Other methods(like TreeVul) will wrongly classify it as OOB

Case study – UAF Patch (race condition)

- **netfilter: ipset: Missing gc cancellations fixed**
  The patch fdb8e12cc2cc ("netfilter: ipset: fix performance regression in swap operation") missed to add the calls to gc cancellations at the error path of create operations and at module unload. Also, because the half of the destroy operations now executed by a function registered by call_rcu(), neither NFNL_SUBSYS_IPSET mutex or rcu read lock is held and therefore the checking of them results false warnings.

```
1  static int ip_set_create(struct sk_buff *skb, const struct
   nfnl_info *info, ...) {
2   ...
3   cleanup:
4  +    set->variant->cancel_gc(set);
5       set->variant->destroy(set);
6   ...
7  }
8
9  // free `h`
10 static void  mtype_destroy(struct ip_set *set) {
11      struct htype *h = set->data;
12      ...
13      kfree(h);
14      set->data = NULL;
15 }
16
17 static void mtype_cancel_gc(struct ip_set *set) {
18      struct htype *h = set->data;
19      if (SET_WITH_TIMEOUT(set))
20          cancel_delayed_work_sync(&h->gc.dwork);
21 }
22
23 static void expire_timers(struct timer_base *base, struct
   hlist_head *head) {
24      ...
25      struct timer_list *timer;
26      void (*fn)(struct timer_list *);
27      // timer is a dangling pointer
28      timer = hlist_entry(head->first, struct timer_list,
   entry);
29      fn = timer->function;
30      ...
31      // Control flow hijacking
32      call_timer_fn(timer, fn, baseclk);
33 }
```

- Ip_set_create() frees an object, which will be used in a timer thread. The patch **aborts** the timer thread.
  - The function call to mtype_destroy() on line 5 ultimately frees the htype object in line 13.
  - cancel_gc() invoked on line 4 calls mtype_cancel_gc() to terminate the use of a dangling pointer in expire_timers().

Exploit - from UAF to control flow hijack

UC RIVERSIDE

- Timer in line 28 is a dangling pointer pointing to freed memory
- By overwriting timer->function, i.e., with a sprayed object, we can control the value of the function pointer.
- We developed a working exploit that sprayed user_key_payload objects (which are elastic) in kmalloc-2k slabs, to achieve the overwrite and successfully achieved the control flow hijack

```
1  static int ip_set_create(struct sk_buff *skb, const struct
   nfnl_info *info, ...) {
2   ...
3   cleanup:
4  +    set->variant->cancel_gc(set);
5       set->variant->destroy(set);
6   ...
7  }
8
9  // free `h`
10 static void  mtype_destroy(struct ip_set *set) {
11      struct htype *h = set->data;
12      ...
13      kfree(h);
14      set->data = NULL;
15 }
16
17 static void mtype_cancel_gc(struct ip_set *set) {
18      struct htype *h = set->data;
19      if (SET_WITH_TIMEOUT(set))
20          cancel_delayed_work_sync(&h->gc.dwork);
21 }
22
23 static void expire_timers(struct timer_base *base, struct
   hlist_head *head) {
24      ...
25      struct timer_list *timer;
26      void (*fn)(struct timer_list *);
27      // timer is a dangling pointer
28      timer = hlist_entry(head->first, struct timer_list,
   entry);
29      fn = timer->function;
30      ...
31      // Control flow hijacking
32      call_timer_fn(timer, fn, baseclk);
33 }
```