IUBIK: Isolating User Bytes in Commodity OS Kernels via Memory Tagging Extensions

Marius Momeu[1,2] Alexander J. Gaidis[2] Jasper v.d. Heidt[1] Vasileios P. Kemerlis[2]

{marius.momeu, jasper.von-der-heidt}@tum.de
{agaidis, vpk}@cs.brown.edu

Technical University of Munich[1]
Brown University[2]

***IEEE Security & Privacy (S&P)*, 2025**

Why *IUBIK*?

Several temporal (UAF) and spatial (OOB) exploits in Linux reveal:

- ***User-controlled data*** $\rightarrow$ crucial for corrupting security-critical data (code pointers).
- ***Elastic objects*** $\rightarrow$ essential for achieving *type confusion*.

Why *IUBIK*?

Several temporal (UAF) and spatial (OOB) exploits in Linux reveal:

- ***User-controlled data*** $\rightarrow$ crucial for corrupting security-critical data (code pointers).
- ***Elastic objects*** $\rightarrow$ essential for achieving *type confusion*.

IUBIK mitigates the above by:

- Leveraging *ARM MTE* to isolate user data in the kernel.

Why *IUBIK*?

Several temporal (UAF) and spatial (OOB) exploits in Linux reveal:

- ***User-controlled data*** $\rightarrow$ crucial for corrupting security-critical data (code pointers).
- ***Elastic objects*** $\rightarrow$ essential for achieving *type confusion*.

IUBIK mitigates the above by:

- Leveraging *ARM MTE* to isolate user data in the kernel.
- Rewriting *elastic objects* to prevent type confusion.

Why *IUBIK*?

Several temporal (UAF) and spatial (OOB) exploits in Linux reveal:

- ***User-controlled data*** $\rightarrow$ crucial for corrupting security-critical data (code pointers).
- ***Elastic objects*** $\rightarrow$ essential for achieving *type confusion*.

IUBIK mitigates the above by:

- Leveraging *ARM MTE* to isolate user data in the kernel.
- Rewriting *elastic objects* to prevent type confusion.
- Employing *dynamic profiling* to reveal user-controlled objects.

Background – Processing User Data in Linux

Data is transferred *from* and *to* user programs via a well-defined interface:

- `copy_from_user` and `copy_to_user`.

Background – Processing User Data in Linux

Data is transferred *from* and *to* user programs via a well-defined interface:

– `copy_from_user` and `copy_to_user`.

User-controlled *elastic objects*

```
1 // simplified
2 int sys_add_key(void __user *pload, size_t plen) {
3   struct user_key_payload *upload;
4   upload = kmalloc(sizeof(*upload) + plen, GFP_KERNEL);
5   copy_from_user(upload->data, pload, plen);
6   memcpy(..., upload->data, plen);
7 }
```

Background – Processing User Data in Linux

Data is transferred *from* and *to* user programs via a well-defined interface:

– `copy_from_user` and `copy_to_user`.

User-controlled *elastic objects*

```
1 // simplified
2 int sys_add_key(void __user *pload, size_t plen) {
3   struct user_key_payload *upload;
4   upload = kmalloc(sizeof(*upload) + plen, GFP_KERNEL);
5   copy_from_user(upload->data, pload, plen);
6   memcpy(..., upload->data, plen);
7 }
```

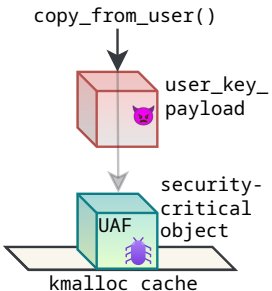
Mixed *(meta)data objects*

```
1 struct user_key_payload {
2     struct rcu_head rcu; // object destructor
3     unsigned short  datalen; // data length
4     char        data[];  // flexible array, user data
5 };
```

Background – User Data in Kernel Exploitation

Crucial for causing *type confusion* scenarios:

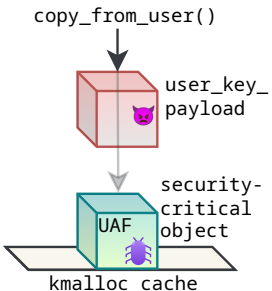
Same-cache attacks



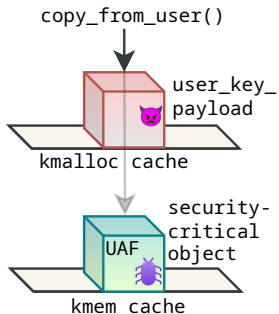
Background – User Data in Kernel Exploitation

Crucial for causing *type confusion* scenarios:

Same-cache attacks



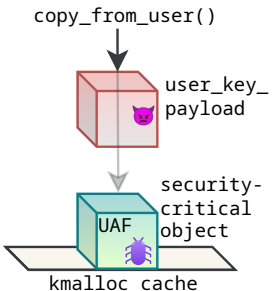
Cross-cache attacks



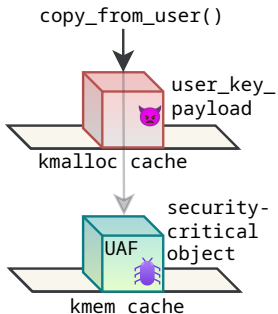
Background – User Data in Kernel Exploitation

Crucial for causing *type confusion* scenarios:

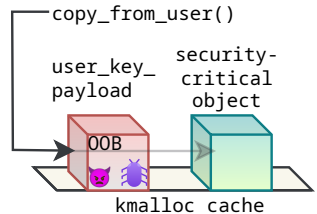
Same-cache attacks



Cross-cache attacks



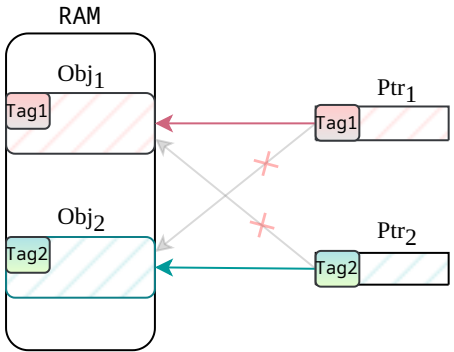
Heap OOB attacks



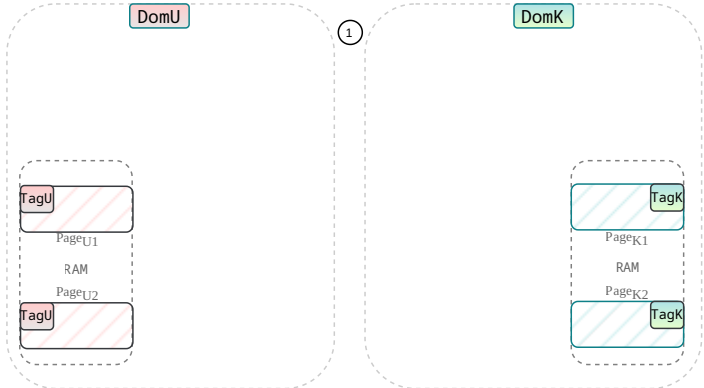
Background – ARM Memory Tagging Extension

Each 16-byte memory granule is tagged (colored) with one of 16 tags (colors).

Memory access is permitted only via pointer tagged with the same tag.



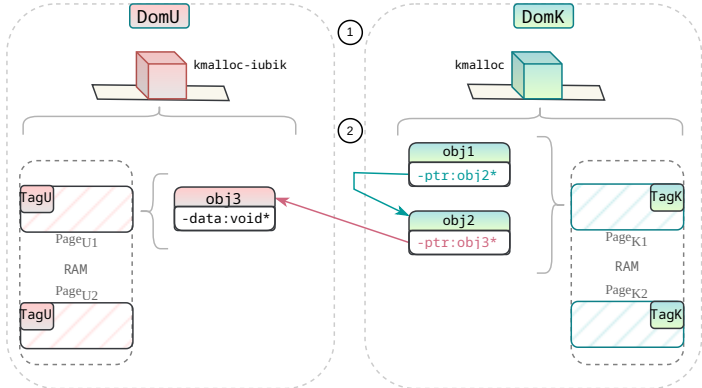
IUBIK – Isolating User Data with MTE



① IUBIK decomposes kernel memory in two MTE domains:

– *DomU* hosts user data, *DomK* hosts everything else.

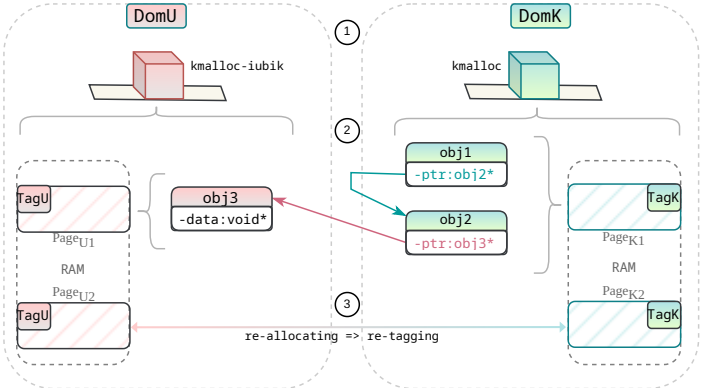
IUBIK – Isolating User Data with MTE



② User data allocated in new `kmalloc` caches in DomU ⇒ mitigates same-cache attacks.

- via new memory allocation flag: *kmalloc(..., GFP_IUBIK)*.

IUBIK – Isolating User Data with MTE



③ DomU pages re-tagged when allocated in DomK (and vice-versa) ⇒ mitigates cross-cache and OOB attacks.

IUBIK – Rewriting Mixed (Meta)Data Objects

```
1 struct user_key_payload {
2     struct rcu_head rcu; // object destructor
3     unsigned short  datalen; // data length
4     -char      data[]; // flexible array, user data
5     +char      *data; // pointer, user data
6 };
```

IUBIK – Rewriting Mixed (Meta)Data Objects

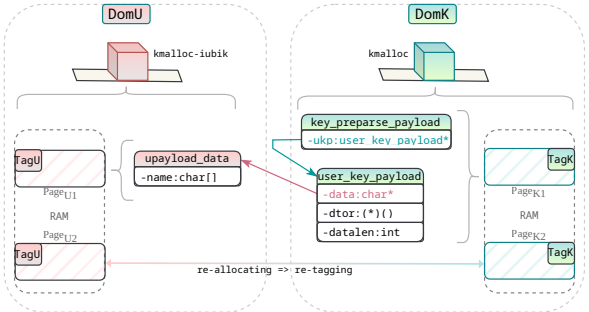
```
1 struct user_key_payload {
2     struct rcu_head rcu; // object destructor
3     unsigned short  datalen; // data length
4     -char       data[]; // flexible array, user data
5     +char       *data; // pointer, user data
6 };
```

```
1 // simplified
2 int sys_add_key(void __user *pload, size_t plen) {
3   struct user_key_payload *upload;
4   -upload = kmalloc(sizeof(*upload) + plen, GFP_KERNEL);
5   +upload = kmalloc(sizeof(*upload), GFP_KERNEL);
6   +upload->data = kmalloc(plen, GFP_IUBIK);
7   copy_from_user(upload->data, pload, plen);
8   memcpy(..., upload->data, plen);
9 }
10
11 void user_destroy(struct user_key_payload *upload) {
12   +kfree(upload->data);
13   kfree(upload);
14 }
```

IUBIK – Rewriting Mixed (Meta)Data Objects

```
1  struct user_key_payload {
2      struct rcu_head rcu; // object destructor
3      unsigned short  datalen; // data length
4      -char       data[];  // flexible array, user data
5      +char       *data;   // pointer, user data
6  };

1  // simplified
2  int sys_add_key(void __user *pload, size_t plen) {
3    struct user_key_payload *upload;
4    -upload = kmalloc(sizeof(*upload) + plen, GFP_KERNEL);
5    +upload = kmalloc(sizeof(*upload), GFP_KERNEL);
6    +upload->data = kmalloc(plen, GFP_IUBIK);
7    copy_from_user(upload->data, pload, plen);
8    memcpy(..., upload->data, plen);
9  }
10
11 void user_destroy(struct user_key_payload *upload) {
12   +kfree(upload->data);
13   kfree(upload);
14 }
```



IUBIK – Usercopy Profiler

Dynamic profiling to reveal user-controlled objects and alloc sites.

Extends the *memory allocation profiling* feature in Linux.

Records alloc, usercopy, memcpy, and free sites involving user data.

```
1  // simplified
2  int sys_add_key(void __user *pload, size_t plen) {
3    struct user_key_payload *upload;
4    upload = kmalloc(sizeof(*upload) + plen, GFP_KERNEL);
5    copy_from_user(upload->data, pload, plen);
6    memcpy(..., upload->data, plen);
7  }
8
9  void user_destroy(struct user_key_payload *upayload) {
10   kfree(upayload);
11 }
```

Evaluation – Usercopy Profiler

Deployed workloads:

- Functional tests: *Linux Test Project (LTP)*, *Kselftests*, *nftables*.
- Performance benchmarks: *LMbench*, *Phoronix Test Suite (PTS)*.

Setup ⇒ 128-core AMD CPU, 128GB RAM, Linux kernel v6.10.

	Total alloc sites	`copy_from_user`	`copy_to_user`
Privileged	292	183 (63%)	163 (56%)
Unprivileged	212	131 (62%)	167 (79%)

Evaluation – Performance Results

IUBIK prototype ⇒ rewrote 79 alloc sites and 23 objects:

- Recorded during boot, LMbench, and Phoronix.

Setup ⇒ Pixel 8, 9-core CPU, 8GB RAM, Linux kernel v5.15 (Android 14).

Overhead	LMbench benchmark
≈ 0%	syscall, read, write, select (500 fds), select (10 fds), select (500 tcp fds), select (10 tcp fds), sigaction, sig deliver, pipe, unix socket, tcp socket, udp socket
2%	fork+"/bin/sh"
3%	prot fault, fork+execve
4%	stat, fork+exit
7%	open/close
8%	fstat
	Phoronix benchmark
≈ 0%	unpack-linux, compile-linux, ffmpeg, openssl (sign), openssl (verify), nginx, sqlite, redis
2%	hackbench

Evaluation – Memory Overhead

Measured *maximum resident size set (RSS)* during boot on the Pixel 8:

- IUBIK requires 2.17% ($\approx$ 7MB) more memory.

Evaluation – Security

Analyzed 31 recent exploits that target UAF and OOB CVEs in Linux:

- Most of them rely on *user-controlled objects* allocated by *SLUB*.
- IUBIK mitigates 28 (90.3%) of them.

Conclusion

IUBIK:

- Introduces a novel memory isolation approach for hardening OS kernels.
- Leverages ARM MTE to isolate user data in shadow kernel memory:
  - With negligible performance and memory overhead.
  - Mitigating $28/31$ (90.3%) recent kernel exploits.
- Provides the *usercopy profiler* to identify call sites that involve user data.

Check out more details and findings in our paper!