



HITBSecConf
2022 Singapore

Settlers of Netlink

Exploiting a limited kernel UAF on Ubuntu 22.04

#HITB2022SIN



Introduction



About

- NCC Group - Exploit Development Group
- Recently working on Pwn2Own competitions
 - Pwn2Own Austin 2021: Western Digital NAS and Lexmark printer
 - Blogs [here](#), [here](#), and [here](#)
- Aaron Adams
 - @fidgetingbits, aaron.adams@nccgroup.com

Pwn2Own Desktop 2022



HITBSecConf
2022 Singapore

- Originally found and exploited one bug



Pwn2Own Desktop 2022

- Originally found and exploited one bug
 - Publicly patched before competition ([CVE-2022-0185](#))



Pwn2Own Desktop 2022

- Originally found and exploited one bug
 - Publicly patched before competition ([CVE-2022-0185](#))
- Started exploiting a second bug we found



Pwn2Own Desktop 2022

- Originally found and exploited one bug
 - Publicly patched before competition ([CVE-2022-0185](#))
- Started exploiting a second bug we found
 - Publicly patched before we were finished ([CVE-2022-0995](#))



Pwn2Own Desktop 2022

- Originally found and exploited one bug
 - Publicly patched before competition ([CVE-2022-0185](#))
- Started exploiting a second bug we found
 - Publicly patched before we were finished ([CVE-2022-0995](#))
- Started exploiting third bug...



Pwn2Own Desktop 2022

- Originally found and exploited one bug
 - Publicly patched before competition ([CVE-2022-0185](#))
- Started exploiting a second bug we found
 - Publicly patched before we were finished ([CVE-2022-0995](#))
- Started exploiting third bug...
 - Fell short by about a week :(



Pwn2Own Desktop 2022

- Originally found and exploited one bug
 - Publicly patched before competition ([CVE-2022-0185](#))
- Started exploiting a second bug we found
 - Publicly patched before we were finished ([CVE-2022-0995](#))
- Started exploiting third bug...
 - Fell short by about a week :(
- We decided to disclose the bug anyway
- This talk is about the third bug ([CVE-2022-32250](#))
 - We targeted Ubuntu 22.04 Kernel 5.15



Tooling: Basic

- `gdb` and `pwndbg`
 - `vmlinux-gdb.py`
- `qemu` and `vmware`
- `pahole`
- CodeQL
- `rp` rop gadget hunter



Tooling: SLUB Allocation Analysis

- We found ftrace left something to be desired
- Found slabdbg, but ARM only
- Pull request for x64 support, but broken on newer kernels
 - Freelist encoding, etc
- We wrote our own new library **libslub**
 - Inspired by **slabdbg**
 - But lots more analysis functionality
- Will be made publicly available at some point
- Functionally similar to our other public heap analysis plugins:
 - libptmalloc
 - libdlmalloc
 - libtalloc



Talk Overview

- Introduction
- Linux netlink/netfilter Recap
- Bug Analysis
- Exploitation approach
- Patch Analysis
- Conclusions



netlink / netfilter / nf_tables



nf_tables Userland Usage

- `nft` command-line interface for interacting with firewall
- Drop input to a TCP port: `nft add rule ip filter input tcp dport 80 drop`
- Well documented tool
- We are interested in what's underneath...



nf_tables Kernel Overview

- netlink is a socket-based communication mechanism
 - Allows userland to control various network functionality in the kernel
 - libmnl helper library



nf_tables Kernel Overview

- netlink is a socket-based communication mechanism
 - Allows userland to control various network functionality in the kernel
 - libmnl helper library
- netfilter is a network filtering mechanism in the kernel
 - Functionality exposed via netlink
 - Hooks into tons of the linux network subsystem
 - Responsible for connection tracking, NAT, nf_tables, etc



nf_tables Kernel Overview

- netlink is a socket-based communication mechanism
 - Allows userland to control various network functionality in the kernel
 - libmnl helper library
- netfilter is a network filtering mechanism in the kernel
 - Functionality exposed via netlink
 - Hooks into tons of the linux network subsystem
 - Responsible for connection tracking, NAT, nf_tables, etc
- nf_tables is the next generation firewall
 - Filtering subsystem that replaced iptables
 - libnftnl helper library



nf_tables Kernel Overview

- netlink is a socket-based communication mechanism
 - Allows userland to control various network functionality in the kernel
 - libmnl helper library
- netfilter is a network filtering mechanism in the kernel
 - Functionality exposed via netlink
 - Hooks into tons of the linux network subsystem
 - Responsible for connection tracking, NAT, nf_tables, etc
- nf_tables is the next generation firewall
 - Filtering subsystem that replaced iptables
 - libnftnl helper library
- All exposed via **CAP_NET_ADMIN**
 - Accessible from unprivileged user or network namespace



Recent netfilter/nf_tables vulnerabilities

- March 2022: Nick Gregory
- April 2022: David Bouman
 - Documented nf_tables in great detail
 - Highly recommended reading as background for our research
- May 2022: @bienpnn Team Orca of Sea Security (Pwn2Own Desktop 2022)
- June 2022: @ezrak1e Ant Group Light-Year Security Lab
- June 2022: Arthur Mongodin RANDORISEC
- July 2022: Arthur Mongodin RANDORISEC



Important nf_tables Terms and Structures

- Tables (`struct nft_table`)
 - Holds groups of chains associated with a specific network protocol (ie: ip, ip6)



Important nf_tables Terms and Structures

- Tables (`struct nft_table`)
 - Holds groups of chains associated with a specific network protocol (ie: ip, ip6)
- Chains (`struct nft_chain`)
 - Holds groups of rules for processing specific protocol traffic according to a policy (ie: accept, drop)



Important nf_tables Terms and Structures

- Tables (`struct nft_table`)
 - Holds groups of chains associated with a specific network protocol (ie: ip, ip6)
- Chains (`struct nft_chain`)
 - Holds groups of rules for processing specific protocol traffic according to a policy (ie: accept, drop)
- Rules (`struct nft_rule`)
 - Holds groups of expressions for processing packets



Important nf_tables Terms and Structures

- Tables (`struct nft_table`)
 - Holds groups of chains associated with a specific network protocol (ie: ip, ip6)
- Chains (`struct nft_chain`)
 - Holds groups of rules for processing specific protocol traffic according to a policy (ie: accept, drop)
- Rules (`struct nft_rule`)
 - Holds groups of expressions for processing packets
- Expressions (`struct nft_expr`)
 - We are interested in `struct nft_dynset`, `struct nft_lookup`, `struct nft_connlimit`



Important nf_tables Terms and Structures

- Tables (`struct nft_table`)
 - Holds groups of chains associated with a specific network protocol (ie: ip, ip6)
- Chains (`struct nft_chain`)
 - Holds groups of rules for processing specific protocol traffic according to a policy (ie: accept, drop)
- Rules (`struct nft_rule`)
 - Holds groups of expressions for processing packets
- Expressions (`struct nft_expr`)
 - We are interested in `struct nft_dynset`, `struct nft_lookup`, `struct nft_connlimit`
- Sets (`struct nft_set`)
 - Tracks a set of data elements associated with a rule or table (ex: list of ports, ips, etc)



Important nf_tables Terms and Structures

- Tables (`struct nft_table`)
 - Holds groups of chains associated with a specific network protocol (ie: ip, ip6)
- Chains (`struct nft_chain`)
 - Holds groups of rules for processing specific protocol traffic according to a policy (ie: accept, drop)
- Rules (`struct nft_rule`)
 - Holds groups of expressions for processing packets
- Expressions (`struct nft_expr`)
 - We are interested in `struct nft_dynset`, `struct nft_lookup`, `struct nft_connlimit`
- Sets (`struct nft_set`)
 - Tracks a set of data elements associated with a rule or table (ex: list of ports, ips, etc)
- Elements
 - Data tracked by a set in special high-performance data structures



Set: struct nft_set

```
1 struct nft_set {
2     struct list_head list;
3     struct list_head bindings;
4     [...]
5     char *name;
6     [...]
7     u8 field_count;
8     u32 use;
9     atomic_t nelems;
10    u32 ndeact;
11    [...]
12    u16 udlen;
13    unsigned char *udata;
14    struct nft_set_ops *ops;
15    [...]
16    u8 num_exprs;
17    struct nft_expr *exprs[NFT_SET_EXPR_MAX];
18    struct list_head catchall_list;
19    unsigned char data[]
20 };
21
```



struct nft_set Members of Interest

- During exploitation we are especially interested in the following `nft_set` members:
 - `list`: Doubly linked list of `nft_set` structures associated with the same table



struct nft_set Members of Interest

- During exploitation we are especially interested in the following `nft_set` members:
 - `list`: Doubly linked list of `nft_set` structures associated with the same table
 - `bindings`: Doubly linked list of expressions that are bound to this set



struct nft_set Members of Interest

- During exploitation we are especially interested in the following `nft_set` members:
 - `list`: Doubly linked list of `nft_set` structures associated with the same table
 - `bindings`: Doubly linked list of expressions that are bound to this set
 - `name`: Name of the set used for lookups in API



struct nft_set Members of Interest

- During exploitation we are especially interested in the following `nft_set` members:
 - `list`: Doubly linked list of `nft_set` structures associated with the same table
 - `bindings`: Doubly linked list of expressions that are bound to this set
 - `name`: Name of the set used for lookups in API
 - `use`: Counter indicating the number of external references



struct nft_set Members of Interest

- During exploitation we are especially interested in the following `nft_set` members:
 - `list`: Doubly linked list of `nft_set` structures associated with the same table
 - `bindings`: Doubly linked list of expressions that are bound to this set
 - `name`: Name of the set used for lookups in API
 - `use`: Counter indicating the number of external references
 - `udata`: A pointer into the set's inline `data[]` array



struct nft_set Members of Interest

- During exploitation we are especially interested in the following `nft_set` members:
 - `list`: Doubly linked list of `nft_set` structures associated with the same table
 - `bindings`: Doubly linked list of expressions that are bound to this set
 - `name`: Name of the set used for lookups in API
 - `use`: Counter indicating the number of external references
 - `udata`: A pointer into the set's inline `data[]` array
 - `udlen`: The length of user data stored in the set's data array



struct nft_set Members of Interest

- During exploitation we are especially interested in the following `nft_set` members:
 - `list`: Doubly linked list of `nft_set` structures associated with the same table
 - `bindings`: Doubly linked list of expressions that are bound to this set
 - `name`: Name of the set used for lookups in API
 - `use`: Counter indicating the number of external references
 - `udata`: A pointer into the set's inline `data[]` array
 - `udlen`: The length of user data stored in the set's data array
 - `ops`: A function table pointer for operating on the set



struct nft_set Members of Interest

- During exploitation we are especially interested in the following `nft_set` members:
 - `list`: Doubly linked list of `nft_set` structures associated with the same table
 - `bindings`: Doubly linked list of expressions that are bound to this set
 - `name`: Name of the set used for lookups in API
 - `use`: Counter indicating the number of external references
 - `udata`: A pointer into the set's inline `data[]` array
 - `udlen`: The length of user data stored in the set's data array
 - `ops`: A function table pointer for operating on the set
- Allocated `kmalloc-512` by default
- Variable length user data can bump it to be placed on `kmalloc-1k`



A closer look at nft_set->bindings

- Expressions bound to a set end up on `set->bindings` doubly-linked list
- Expressions will contain a `struct nft_set_binding` member

```
1 struct nft_set_binding {  
2     struct list_head    list;  
3     const struct nft_chain *chain;  
4     u32                  flags;  
5 };
```

- So `set->bindings` entries will point into `list` member above



Expression: struct nft_expr

- All expression types extend `struct nft_expr`, and are stored in `data` member

```
1 struct nft_expr {
2     const struct nft_expr_ops  *ops;
3     unsigned char               data[]
4         __attribute__((aligned(__alignof__(u64)))));
5 };
6
7 static inline void *nft_expr_priv(const struct nft_expr *expr)
8 {
9     return (void *)expr->data;
10 }
11
```

- Typical use:

```
1 const struct nft_lookup *priv = nft_expr_priv(expr);
```

- Noteworthy because size overhead influences slab cache selection



Lookup Expression: struct nft_lookup

- Fetches of value from a key in the specified set
- Allocated on `kmalloc-48` slab cache
- We are interested in `binding` being at offset 0x10

```
1 struct nft_lookup {  
2     struct nft_set    *set;  
3     u8                sreg;  
4     u8                dreg;  
5     bool              invert;  
6     struct nft_set_binding binding;  
7 };  
8
```



Dynamic Set Expression: struct nft_dynset

- Allows expressions to be associated with set elements
- Allocated on `kmalloc-96` slab cache
- We are interested in `binding` being at offset 0x38

```
1 struct nft_dynset {
2     struct nft_set      *set;
3     struct nft_set_ext_tmpl tmpl;
4     enum nft_dynset_ops  op:8;
5     u8                   sreg_key;
6     u8                   sreg_data;
7     bool                 invert;
8     bool                 expr;
9     u8                   num_exprs;
10    u64                   timeout;
11    struct nft_expr       *expr_array[NFT_SET_EXPR_MAX];
12    struct nft_set_binding binding;
13 };
14
```

Normal Set Expression Binding Relationship

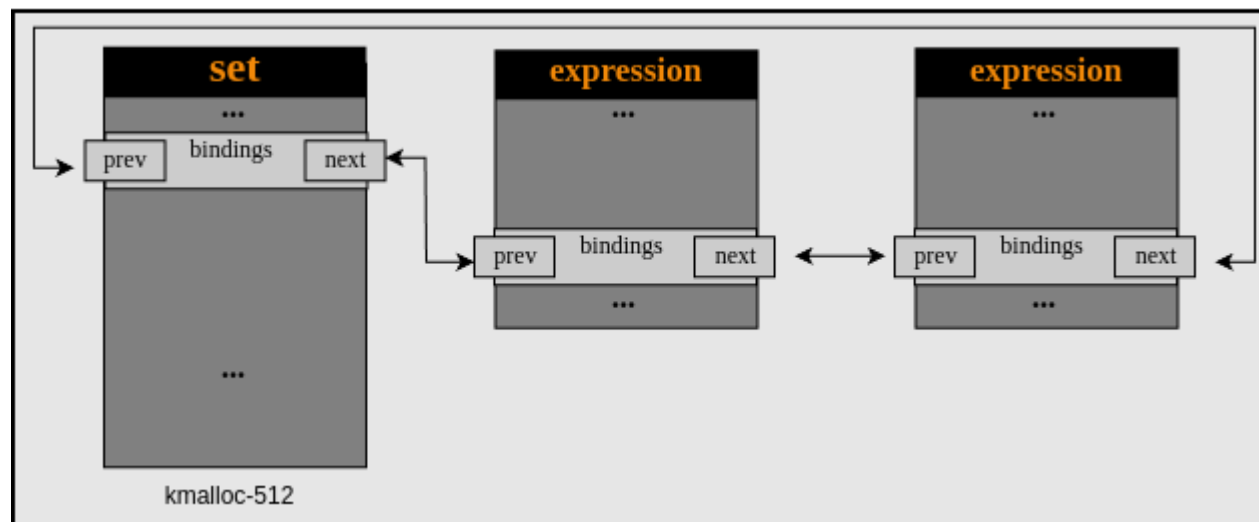
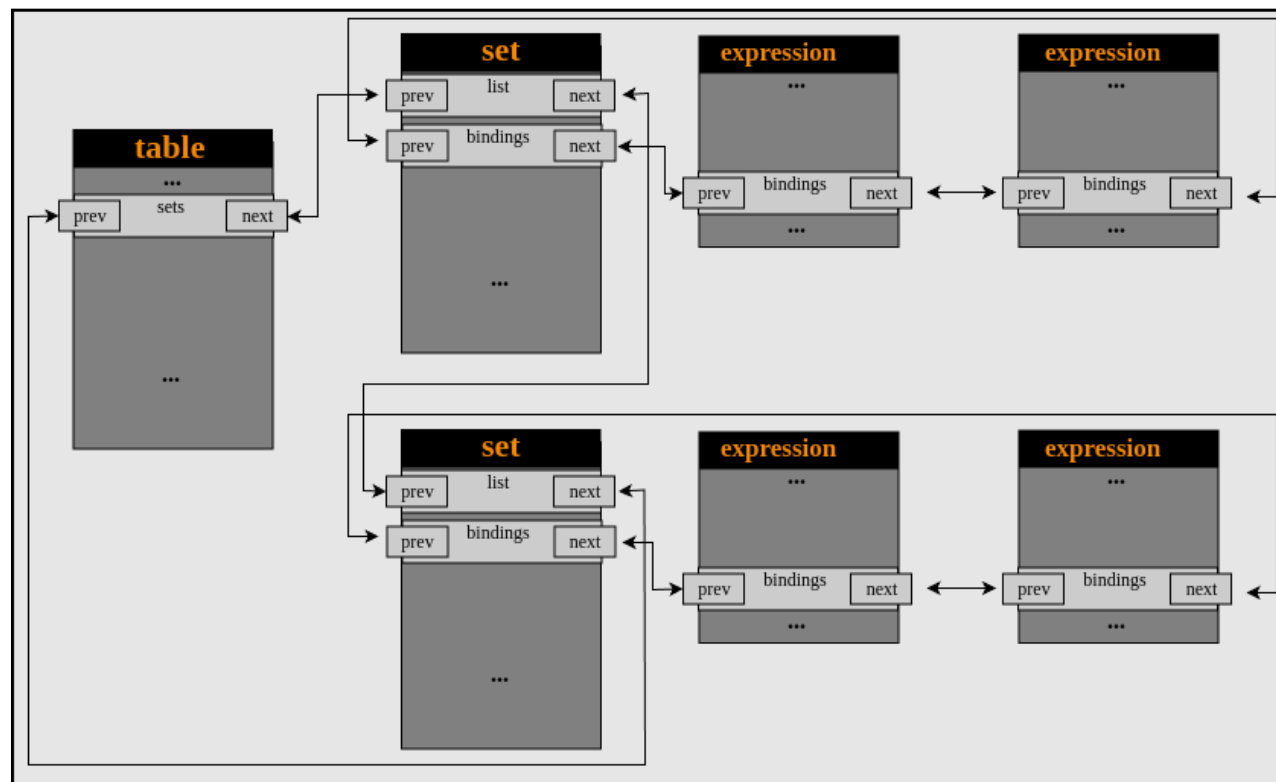


Table With Linked Sets





Embedding Expressions in Sets

- Set's support embedding expressions during creation
- Similar to a "dynset" expression
- Expressions will be run when elements in the set are updated
- Only specific types of expressions can be embedded in a set
 - Expression must be "stateful" (ie: a counter)



CVE-2022-32250



Bug Overview

- Original disclosure [here](#)
- Found with syzkaller
 - No repro could be generated
 - Triaged manually
- UAF while handling expressions on `set->bindings` list
- Writes one uncontrolled pointer to an uncontrolled offset



Bug Overview

- Original disclosure [here](#)
- Found with syzkaller
 - No repro could be generated
 - Triaged manually
- UAF while handling expressions on `set->bindings` list
- Writes one uncontrolled pointer to an uncontrolled offset
- [@dvyukov](#) noticed after our disclosure that syzbot found it in [November 2021](#)
 - Automatically closed as invalid



Initialize Expression First, Check Validity After

```
1 struct nft_expr *nft_set_elem_expr_alloc(const struct nft_ctx *ctx,
2                                         const struct nft_set *set,
3                                         const struct nlattr *attr)
4 {
5     struct nft_expr *expr;
6     int err;
7
8     expr = nft_expr_init(ctx, attr);
9     if (IS_ERR(expr))
10         return expr;
11     err = -EOPNOTSUPP;
12     if (!(expr->ops->type->flags & NFT_EXPR_STATEFUL))
13         goto err_set_elem_expr;
14
15     [...]
16     return expr;
17
18 err_set_elem_expr:
19     nft_expr_destroy(ctx, expr);
20     return ERR_PTR(err);
21 }
22
```

Initializes expression
first

Checks if expression
is valid type second

Destroys immediately
if type is wrong

Indirect Expression Destruction

- `nft_expr_destroy()` calls into expression-specific `destroy` function

```
1 void nft_expr_destroy(const struct nft_ctx *ctx, struct nft_expr *expr)
2 {
3     nf_tables_expr_destroy(ctx, expr);
4     kfree(expr);
5 }
6
7 static void nf_tables_expr_destroy(const struct nft_ctx *ctx,
8     struct nft_expr *expr)
9 {
10     const struct nft_expr_type *type = expr->ops->type;
11
12     if (expr->ops->destroy)
13         expr->ops->destroy(ctx, expr);
14     module_put(type->owner);
15 }
16
```



Lookup and Dynset Expressions

- Both of these expressions look up a set when initialized
- Added to the `set->bindings` on initialization via `nf_tables_bind_set()`
- But, their destroy method called by `nft_expr_destroy()` won't remove them from `set->bindings` list



Lookup and Dynset Expressions

- Both of these expressions look up a set when initialized
- Added to the `set->bindings` on initialization via `nf_tables_bind_set()`
- But, their destroy method called by `nft_expr_destroy()` won't remove them from `set->bindings` list
- UAF on subsequent `set->bindings` use
 - List updates add or remove `struct nft_set_binding` linkage
 - Ability to write address of set, or another expressions, to freed memory

Dynset Expression: Initialization

```

1 static int nft_dynset_init(const struct nft_ctx *ctx,
2                             const struct nft_expr *expr,
3                             const struct nlattr * const tb[])
4 {
5     struct nftables_pernet *nft_net = nft_pernet(ctx->net);
6     struct nft_dynset *priv = nft_expr_priv(expr);
7     [...]
8     err = nf_tables_bind_set(ctx, set, &priv->binding);
9     if (err < 0)
10         goto err_expr_free;
11
12     if (set->size == 0)
13         set->size = 0xffff;
14
15     priv->set = set;
16     return 0;
17     [...]
18 }
19

```

Expression added
to set->bindings

Dynset Expression: Destruction

- "dynset" expression is not unbound from this set when destroyed
- Normally would be done by `nf_tables_unbind_set()`

```

1 static void nft_dynset_destroy(const struct nft_ctx *ctx,
2                               const struct nft_expr *expr)
3 {
4     struct nft_dynset *priv = nft_expr_priv(expr);
5     int i;
6
7     for (i = 0; i < priv->num_exprs; i++)
8         nft_expr_destroy(ctx, priv->expr_array[i]);
9
10    nf_tables_destroy_set(ctx, priv->set);
11 }
12

```

- Set destruction doesn't happen since `set->bindings` is not empty

```

1 void nf_tables_destroy_set(const struct nft_ctx *ctx, struct nft_set *set)
2 {
3     if (list_empty(&set->bindings) && nft_set_is_anonymous(set))
4         nft_set_destroy(ctx, set);
5 }
6

```



Example: How to Write Set Address to a Free Chunk

- Create a valid set that expressions we initialize can reference



Example: How to Write Set Address to a Free Chunk

- Create a valid set that expressions we initialize can reference
- Bind a expression to the valid set, to populate `set->bindings` with one entry



Example: How to Write Set Address to a Free Chunk

- Create a valid set that expressions we initialize can reference
- Bind a expression to the valid set, to populate `set->bindings` with one entry
- Create a new invalid set



Example: How to Write Set Address to a Free Chunk

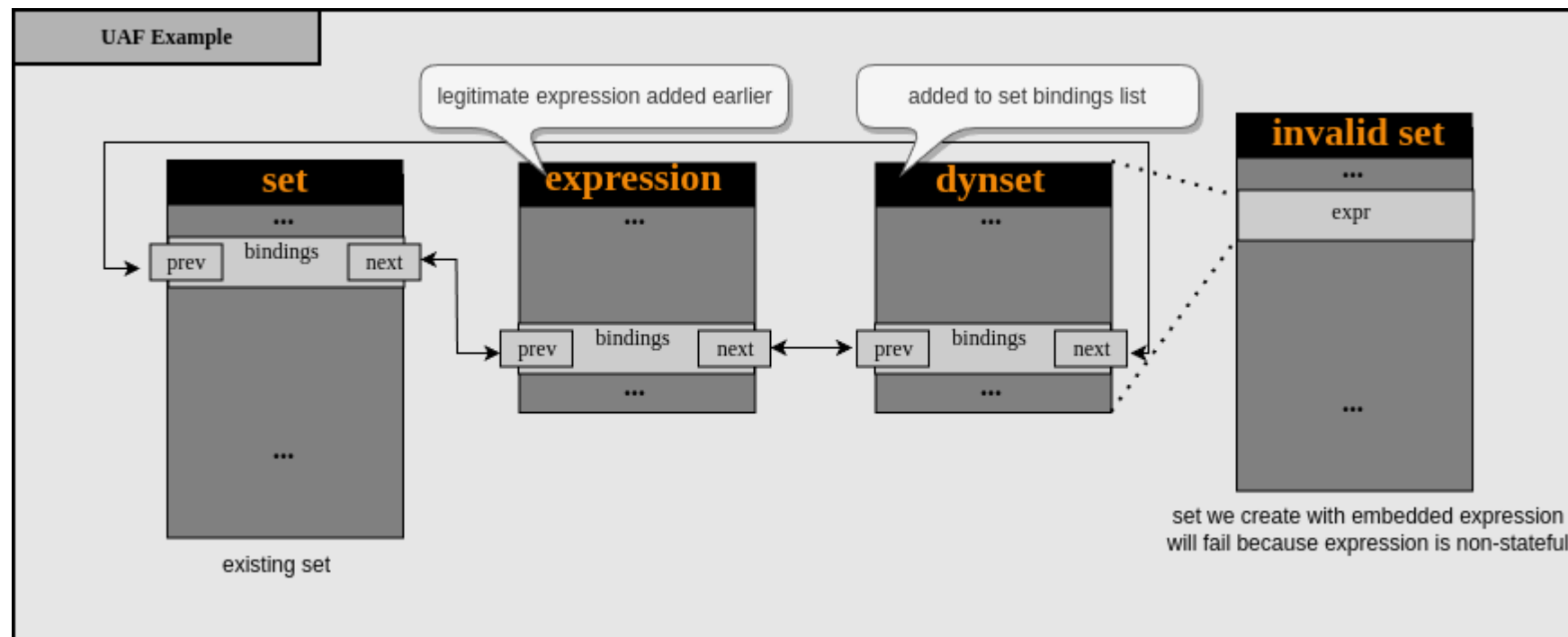
- Create a valid set that expressions we initialize can reference
- Bind a expression to the valid set, to populate `set->bindings` with one entry
- Create a new invalid set
- Embed "lookup" or "dynset" expression in the invalid set
 - Embedded expression references valid set
 - Added to the `set->bindings` list of referenced set on initialization
 - Immediately destroyed after initialization, but not removed from `set->bindings`



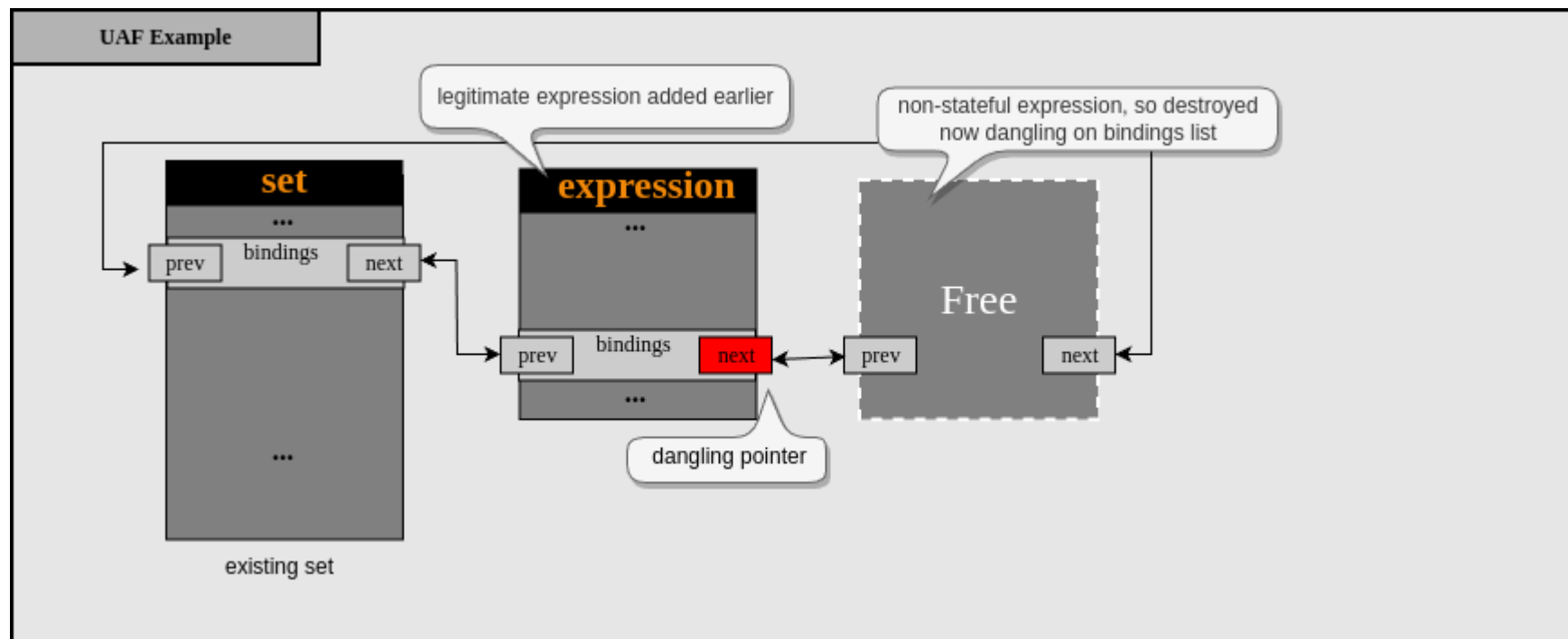
Example: How to Write Set Address to a Free Chunk

- Create a valid set that expressions we initialize can reference
- Bind a expression to the valid set, to populate `set->bindings` with one entry
- Create a new invalid set
- Embed "lookup" or "dynset" expression in the invalid set
 - Embedded expression references valid set
 - Added to the `set->bindings` list of referenced set on initialization
 - Immediately destroyed after initialization, but not removed from `set->bindings`
- Destroy first expression on `set->bindings`
 - UAF when updating dangling expression with new `prev` pointer

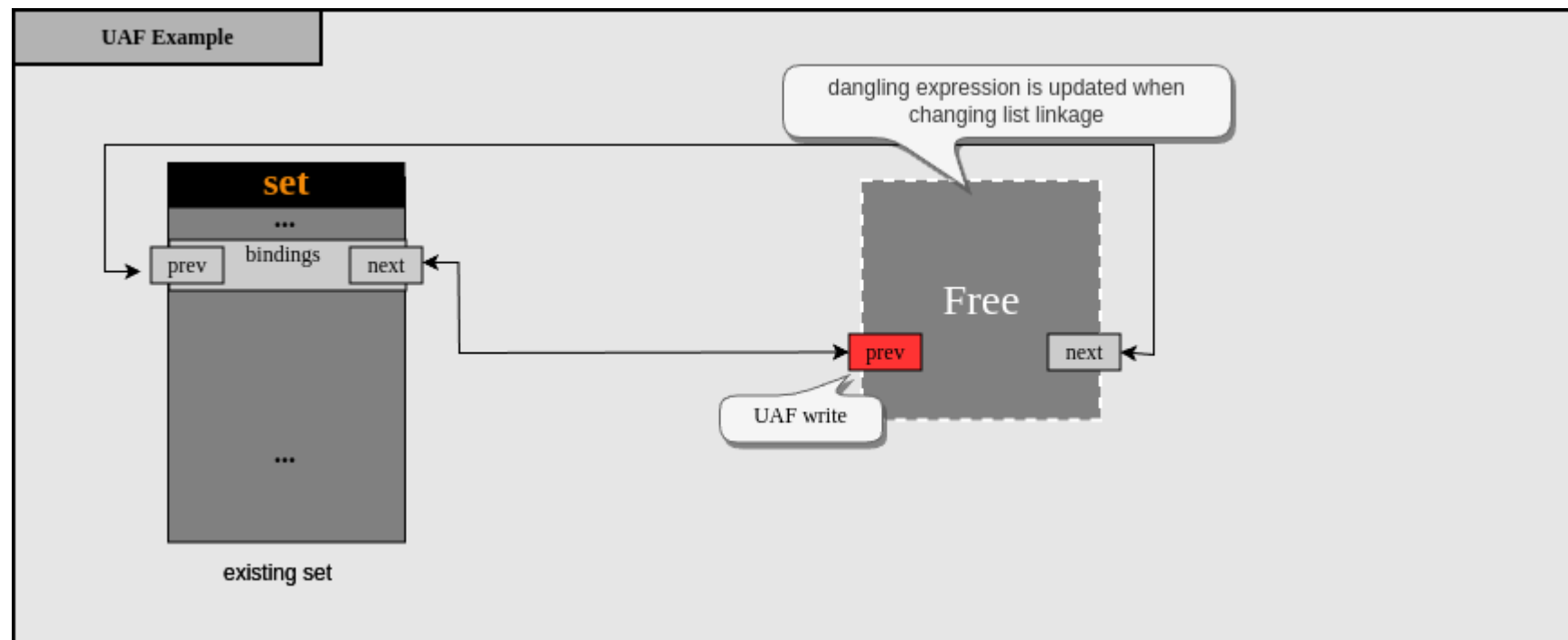
Non-Stateful Expression Added to Bindings List



Non-Stateful Expression Freed, Dangling On Bindings



UAF Write of New Expression Added to List





Exploiting CVE-2022-32250





Initial Exploitation Ideas

- How to exploit this?
- Ideas:
 - Overwrite some length parameter with the pointer?
 - Overwrite some pointer with new pointer, and create better UAF?
 - Write pointer to buffer, and leak back to userland?
- Constraints of where the pointer is written is quite limiting



Easy Win: Leak Some Address

- Confirm mental model
- Leak a set or expression address
 - Offset of **bindings** member
- How to leak the data?



Easy Win: Leak Some Address

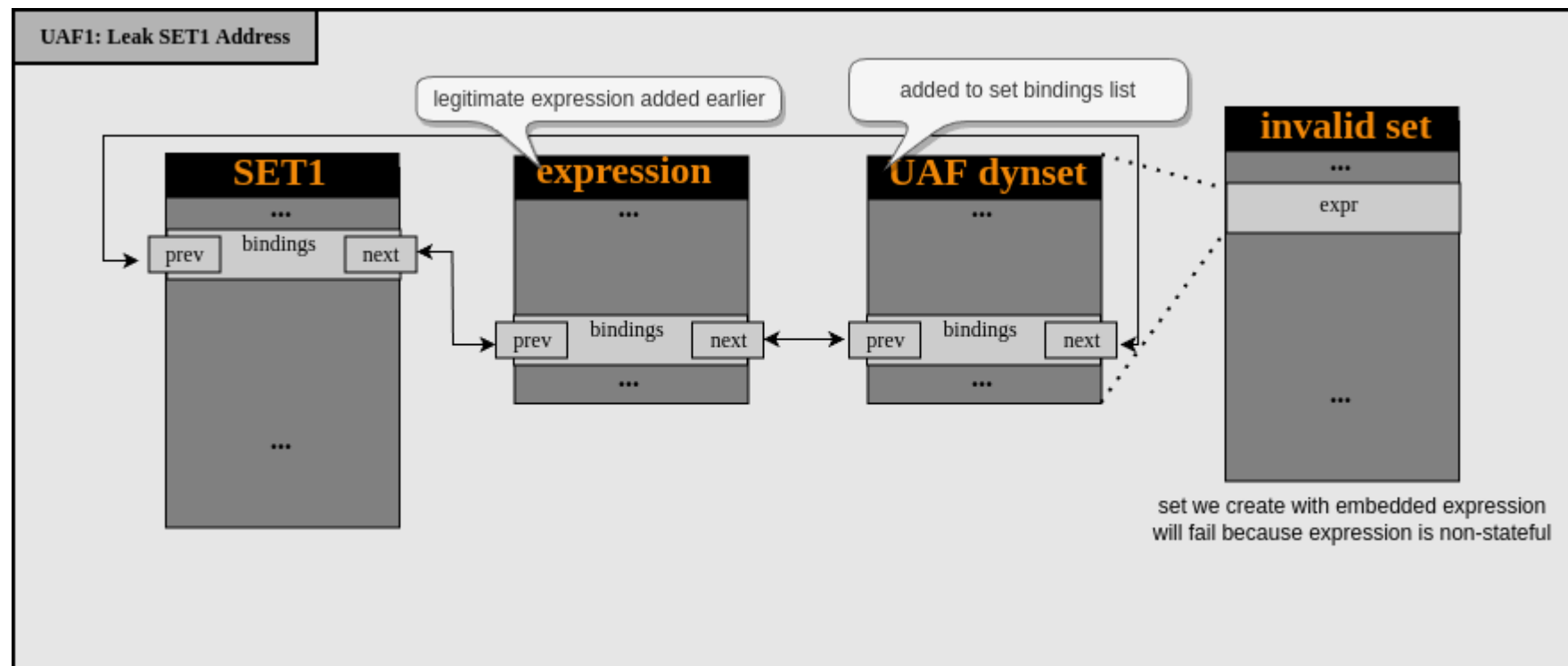
- Confirm mental model
- Leak a set or expression address
 - Offset of `bindings` member
- How to leak the data?
- Use popular `struct user_key_payload` technique
 - `add_key()` syscall: Controlled size to get allocated on different slab caches
 - `keyctl(KEYCTL_READ)`: Can read payload contents at any time



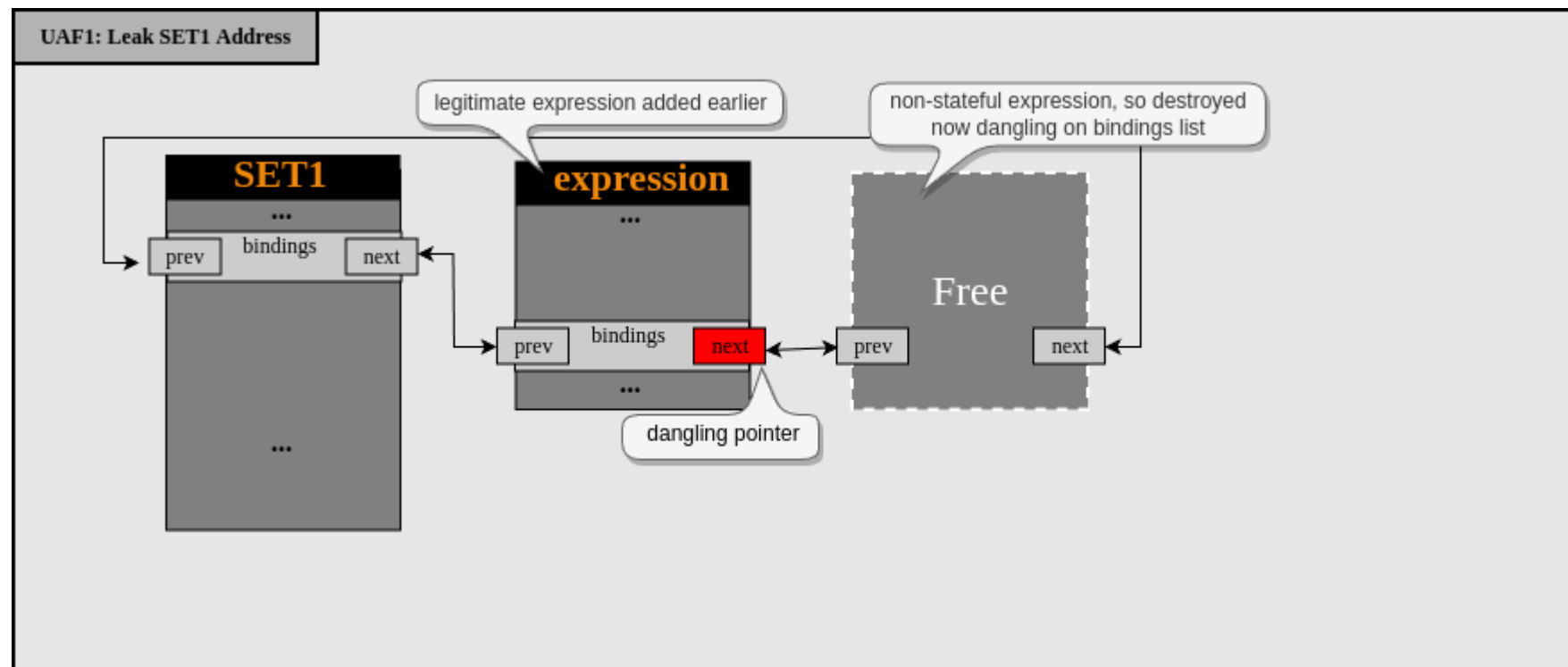
Easy Win: Leak Some Address

- Confirm mental model
- Leak a set or expression address
 - Offset of `bindings` member
- How to leak the data?
- Use popular `struct user_key_payload` technique
 - `add_key()` syscall: Controlled size to get allocated on different slab caches
 - `keyctl(KEYCTL_READ)`: Can read payload contents at any time
- Terminology:
 - This stage will be `UAF1`
 - The set we leak will be referred to as `SET1`

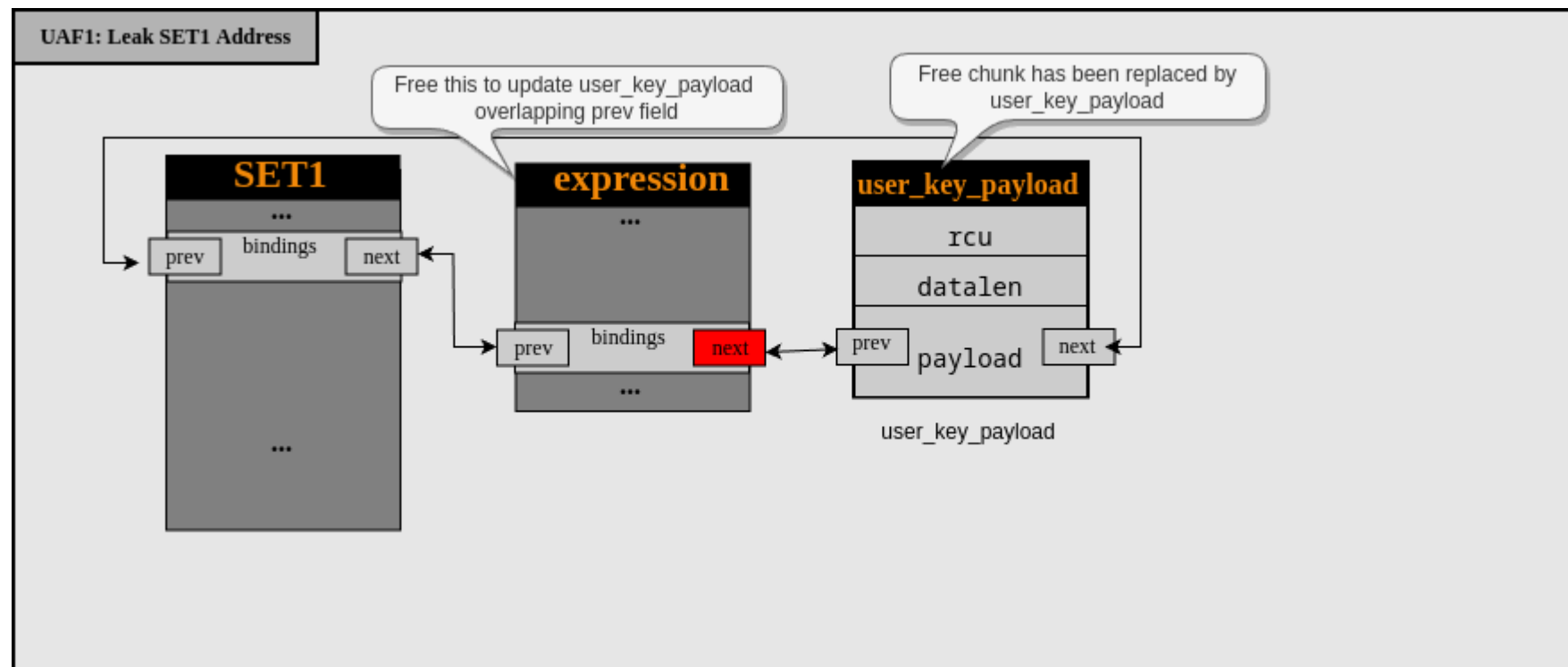
UAF1: SET1 Address Leak



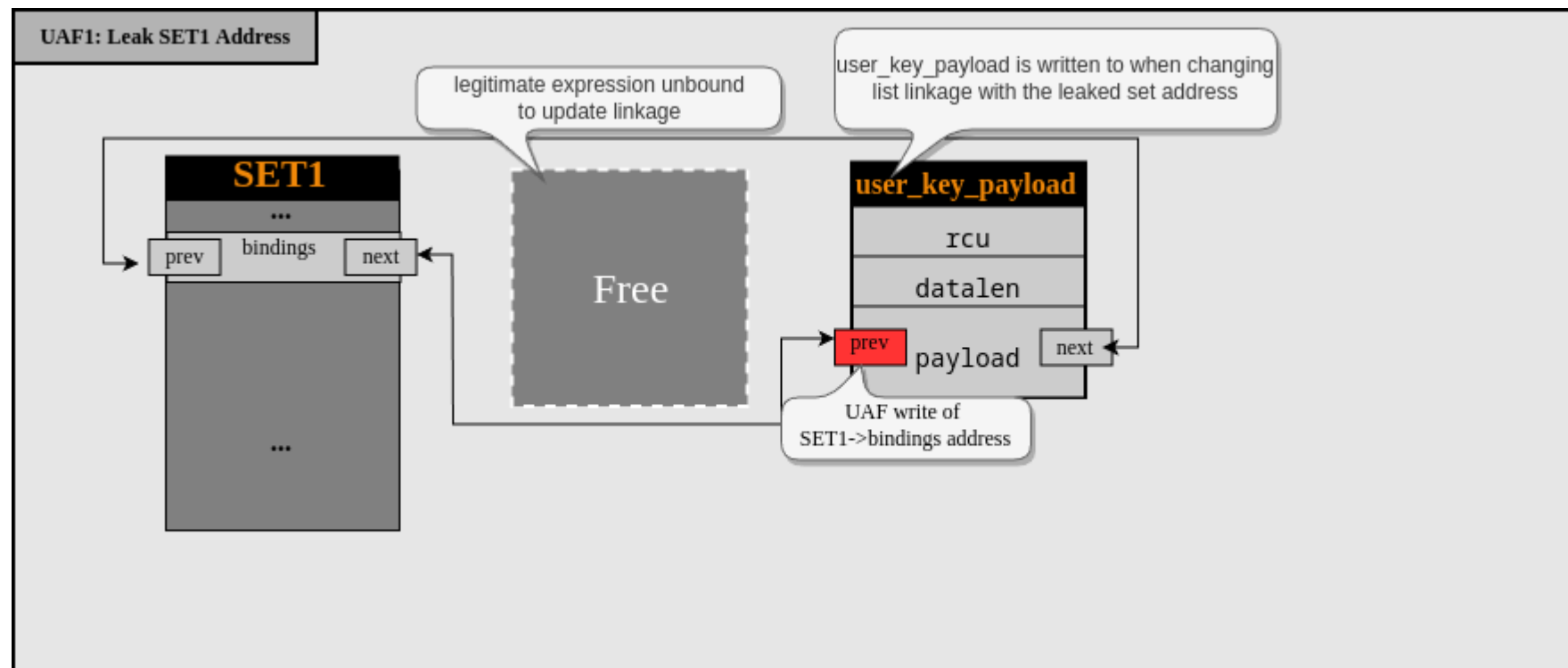
UAF1: SET1 Address Leak



UAF1: SET1 Address Leak



UAF1: SET1 Address Leak



- Possible to read the written address from userland



Success, But What Next?

- This **SET1** address isn't useful for now...
 - But confirms stuff works as expected
- Let's try to free some other object



Success, But What Next?

- This **SET1** address isn't useful for now...
 - But confirms stuff works as expected
- Let's try to free some other object
- Goal: Find an object on **kmalloc-48** or **kmalloc-96** with overlapping pointer offsets
 - Constraint: overlapping pointer must be freeable on demand
 - Outcome: gives a new free primitive



Success, But What Next?

- This SET1 address isn't useful for now...
 - But confirms stuff works as expected
- Let's try to free some other object
- Goal: Find an object on `kmalloc-48` or `kmalloc-96` with overlapping pointer offsets
 - Constraint: overlapping pointer must be freeable on demand
 - Outcome: gives a new free primitive
- Two options of what to free using such a primitive:
 - Free `sizeof(expression)` bytes @ `&expression->bindings` address (quirky)
 - Free `sizeof(set)` bytes @ `&set->bindings` address (better)
- We chose to use a set. See our blog for more details



Success, But What Next?

- This **SET1** address isn't useful for now...
 - But confirms stuff works as expected
- Let's try to free some other object
- Goal: Find an object on **kmalloc-48** or **kmalloc-96** with overlapping pointer offsets
 - Constraint: overlapping pointer must be freeable on demand
 - Outcome: gives a new free primitive
- Two options of what to free using such a primitive:
 - Free sizeof(expression) bytes @ **&expression->bindings** address (quirky)
 - Free sizeof(set) bytes @ **&set->bindings** address (better)
- We chose to use a set. See our blog for more details
- Now to need to find a replacement object that gives us a free primitive
 - CodeQL to the rescue



Finding a Suitable Object Using CodeQL

- Find 96-byte structures allocated on slab cache
 - Specific member offsets must be pointers

```
1 import cpp
2
3 from FunctionCall fc, Type t, Variable v, Field f, Type t2
4 where (fc.getTarget().hasName("kmalloc") or
5        fc.getTarget().hasName("kzalloc") or
6        fc.getTarget().hasName("kcalloc"))
7        and
8        exists(Assignment assign | assign.getRValue() = fc and
9                assign.getLValue() = v.getAnAccess() and
10               v.getType().(PointerType).refersToDirectly(t)) and
11        t.getSize() <= 96 and t.getSize() > 64 and t.fromSource() and
12        f.getDeclaringType() = t and
13        (f.getType().(PointerType).refersTo(t2) and t2.getSize() <= 8) and
14        (f.getByteOffset() = 72)
15 select fc, t, fc.getLocation()
16
```



Candidate: cgroup_fs_context

- Allocated when creating a new `cgroup`
- Lives on `kmalloc-96`, same as `nft_dynset`
- `cgroup_fs_context->release_agent` overlaps with `nft_dynset->bindings->prev`
- Exposed via `fd = syscall(__NR_fsopen, "cgroup2", 0);`
- Free on demand by destroying the cgroup: `close(fd);`



struct cgroup_fs_context

```
1 struct cgroup_fs_context {
2     struct kernfs_fs_context kfc;
3     struct cgroup_root      *root;
4     struct cgroup_namespace *ns;
5     unsigned int    flags;          /* CGRP_ROOT_* flags */
6
7     /* cgroup1 bits */
8     bool    cpuset_clone_children;
9     bool    none;                  /* User explicitly requested empty subsystem */
10    bool    all_ss;                 /* Seen 'all' option */
11    u16     subsys_mask;            /* Selected subsystems */
12    char     *name;                 /* Hierarchy name */
13    char     *release_agent;        /* Path for release notifications */
14 };
15
```



Freeing release_agent

```
1 static void cgroup_fs_context_free(struct fs_context *fc)
2 {
3     struct cgroup_fs_context *ctx = cgroup_fc2context(fc);
4
5     kfree(ctx->name);
6     kfree(ctx->release_agent);
7     put_cgroup_ns(ctx->ns);
8     kernfs_free_fs_context(fc);
9     kfree(ctx);
10 }
11
```

Free primitives

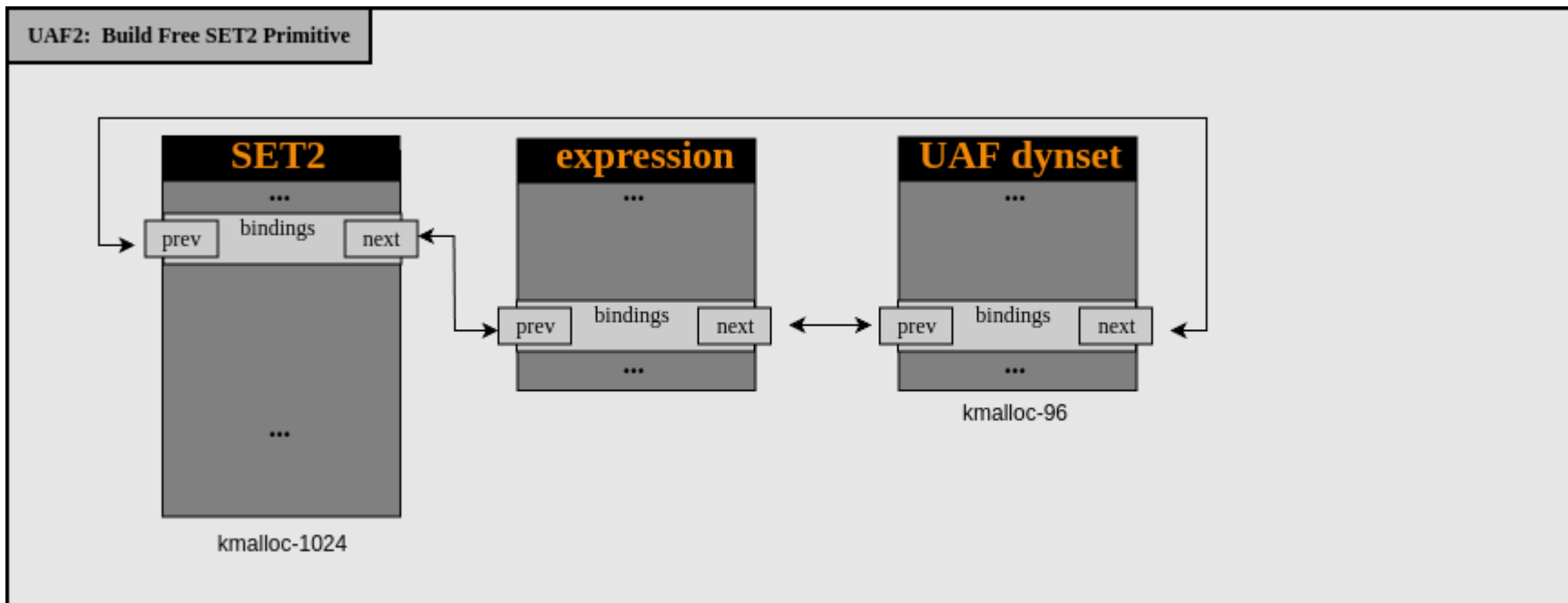


Preparing a Set Freeing Primitive

- We will refer to this phase as **UAF2**
- We will refer to this freed set as **SET2**

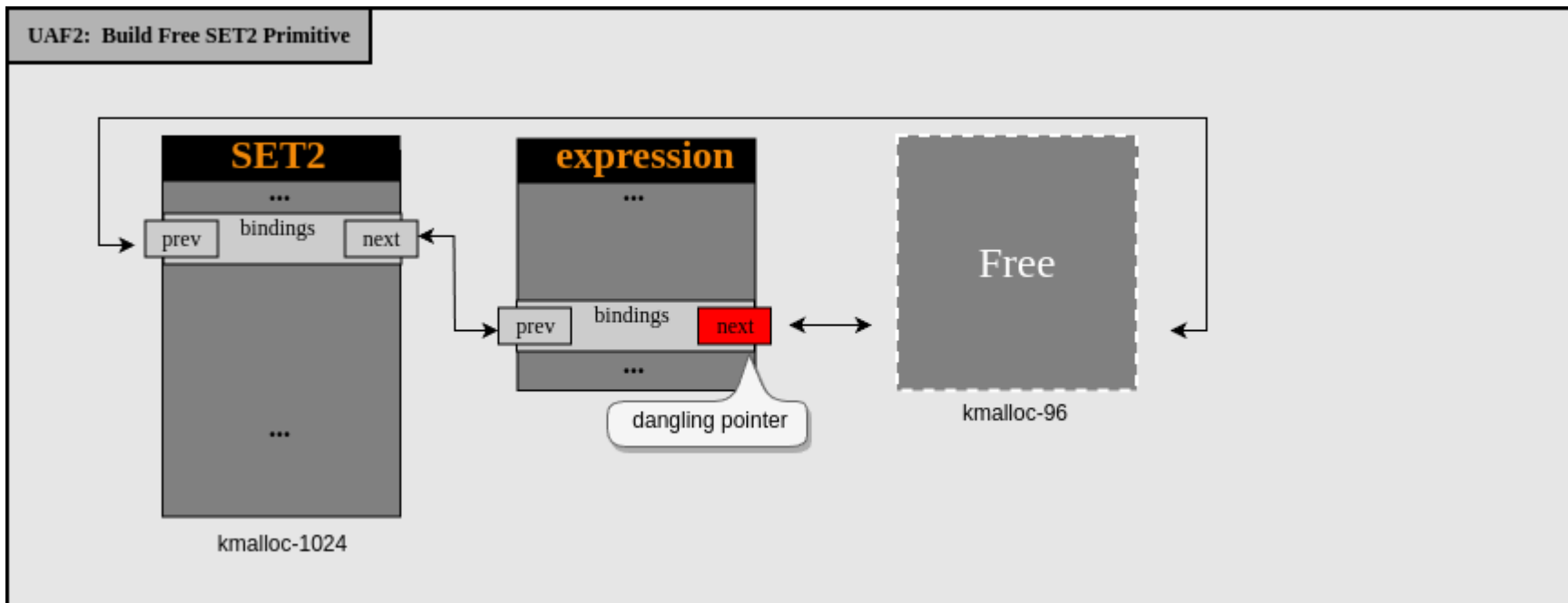
UAF2: release_agent Overwrite

- Trigger `set->bindings` UAF with a `nft_dynset` expression



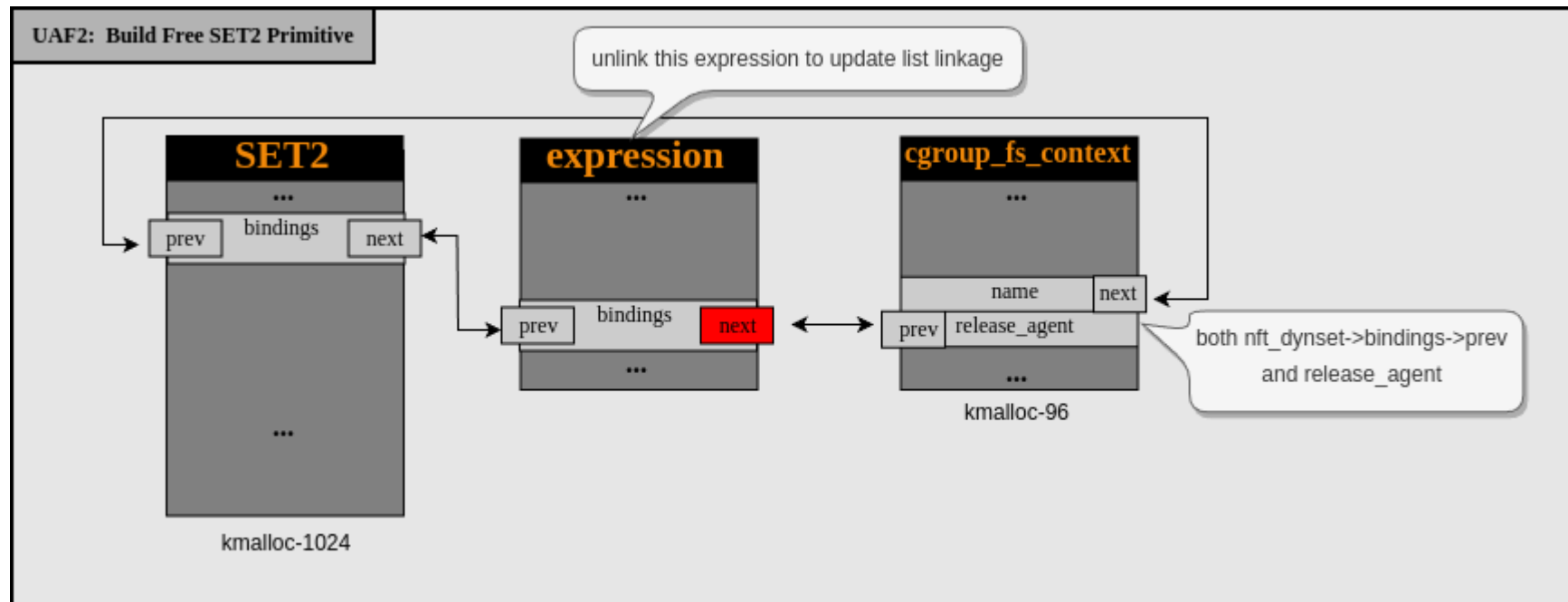
UAF2: release_agent Overwrite

- Replace `nft_dynset` with a `cgroup_fs_context`



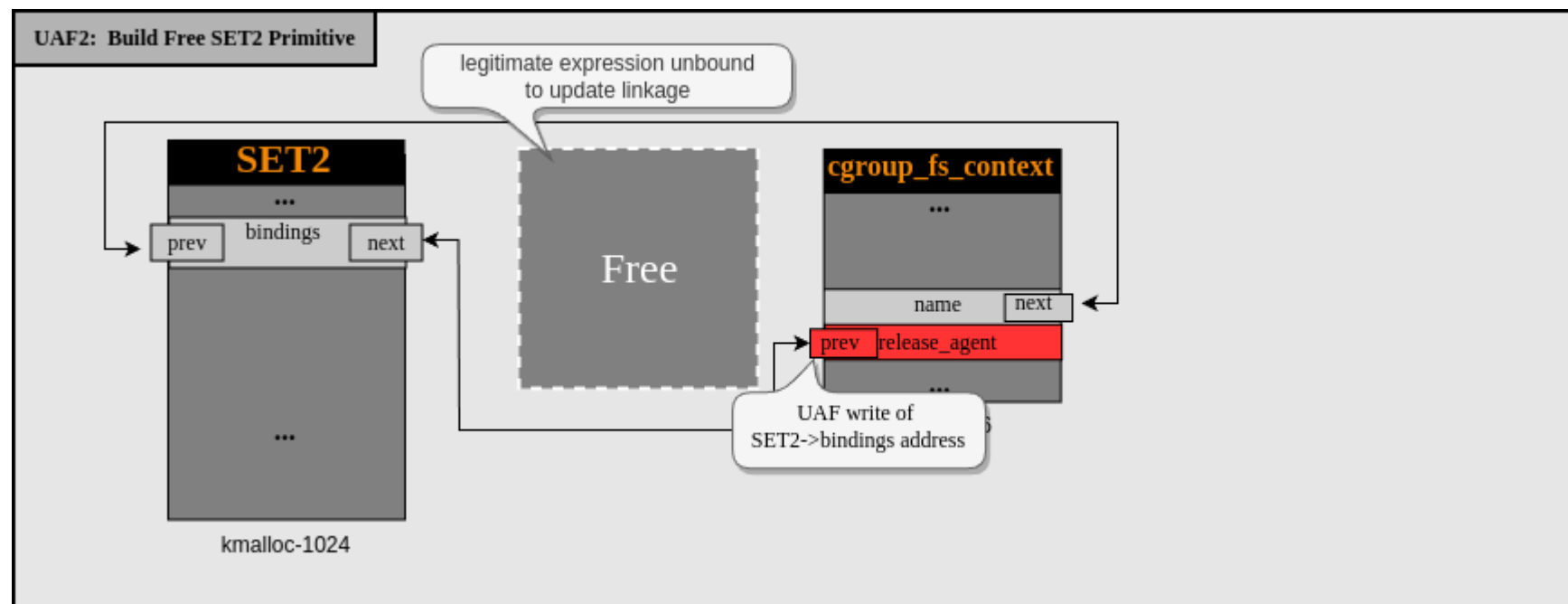
UAF2: release_agent Overwrite

- Remove an entry from the `set->bindings`



UAF2: release_agent Overwrite

- Overwrite `cgroup_fs_context->release_agent` with `&set->bindings->next`



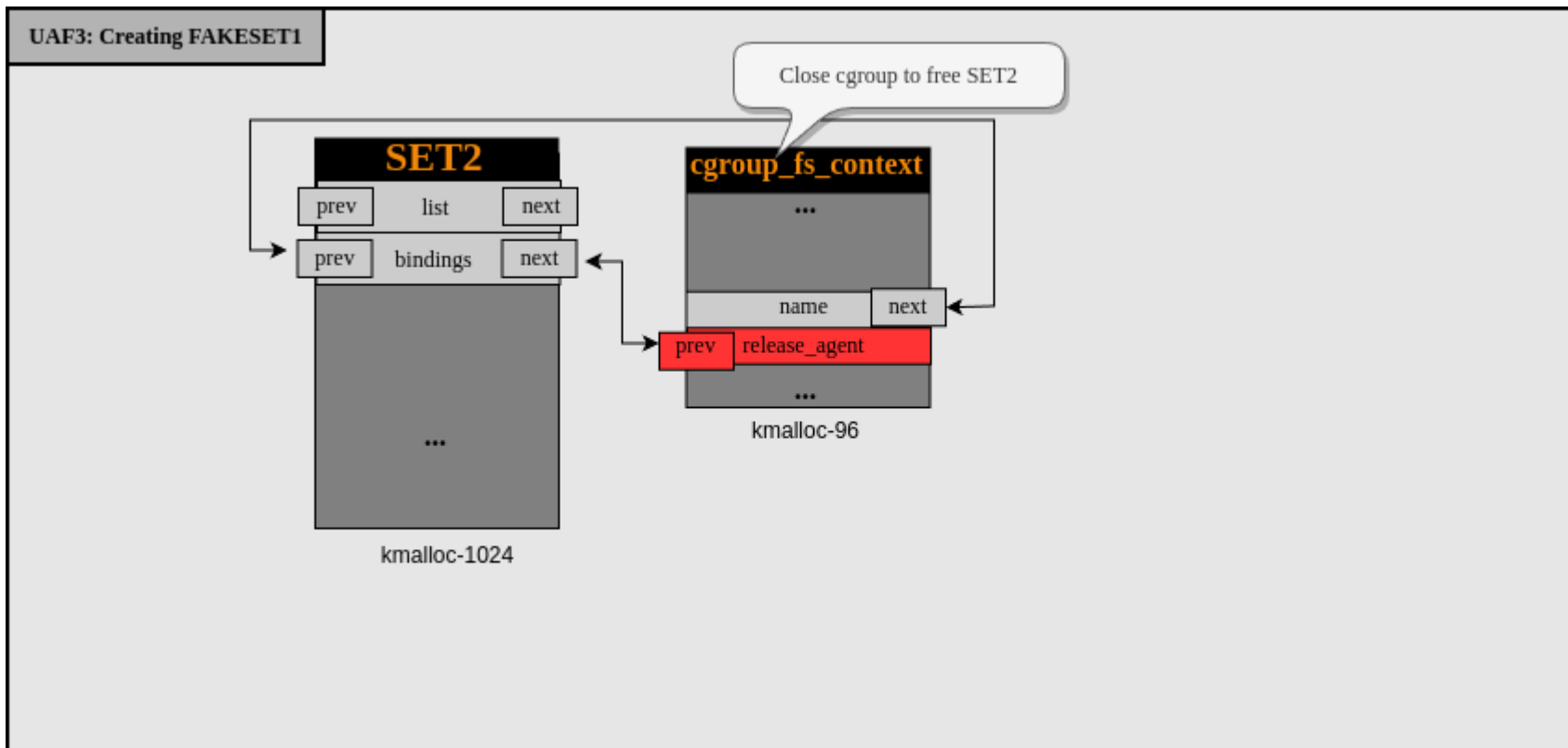


Freeing and Replacing a Set

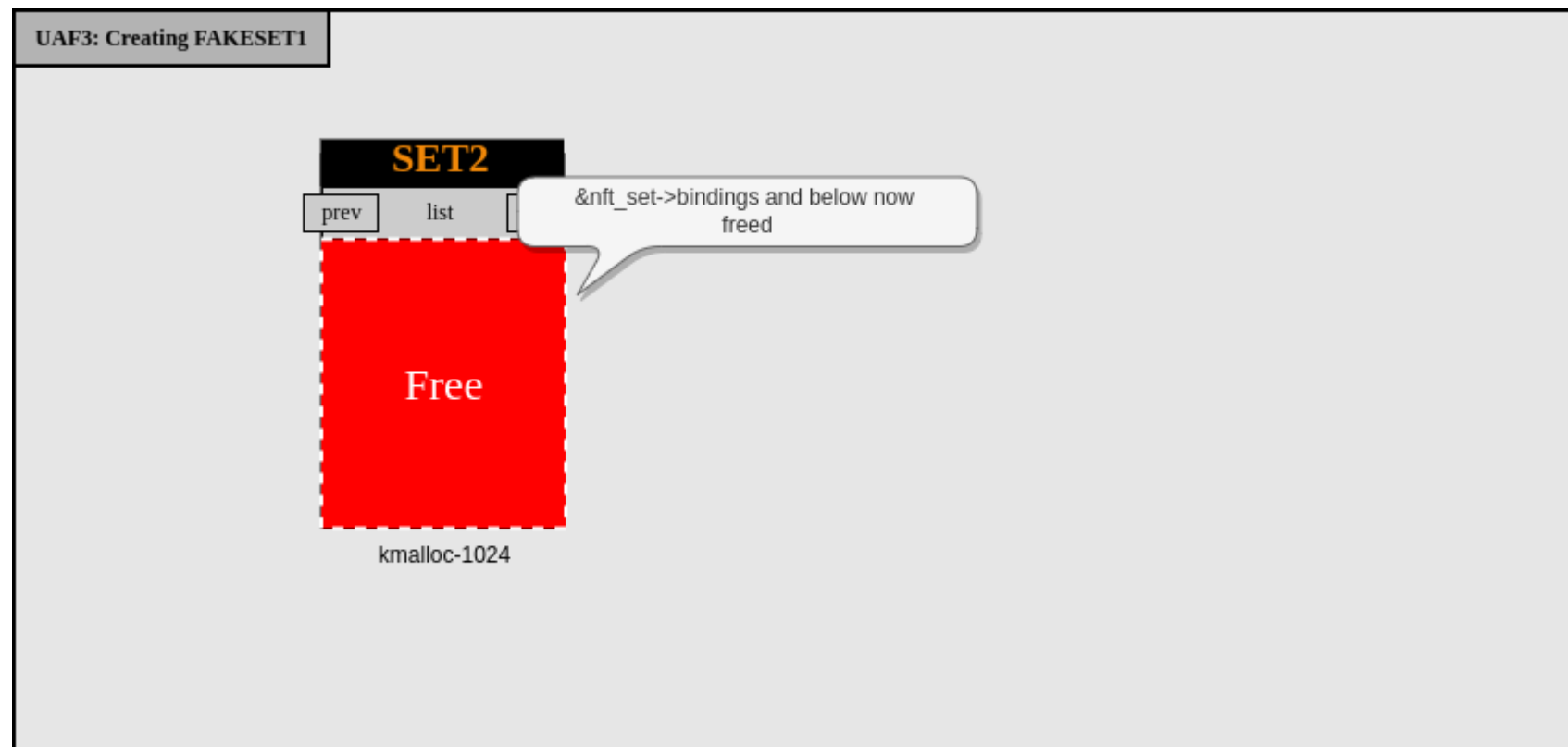
- We will refer to this phase as **UAF3**
- We will refer to the replaced **SET2** as **FAKESET1**

UAF3: FAKESET1 to Bypass KASLR

- Destroying the cgroup will free SET2

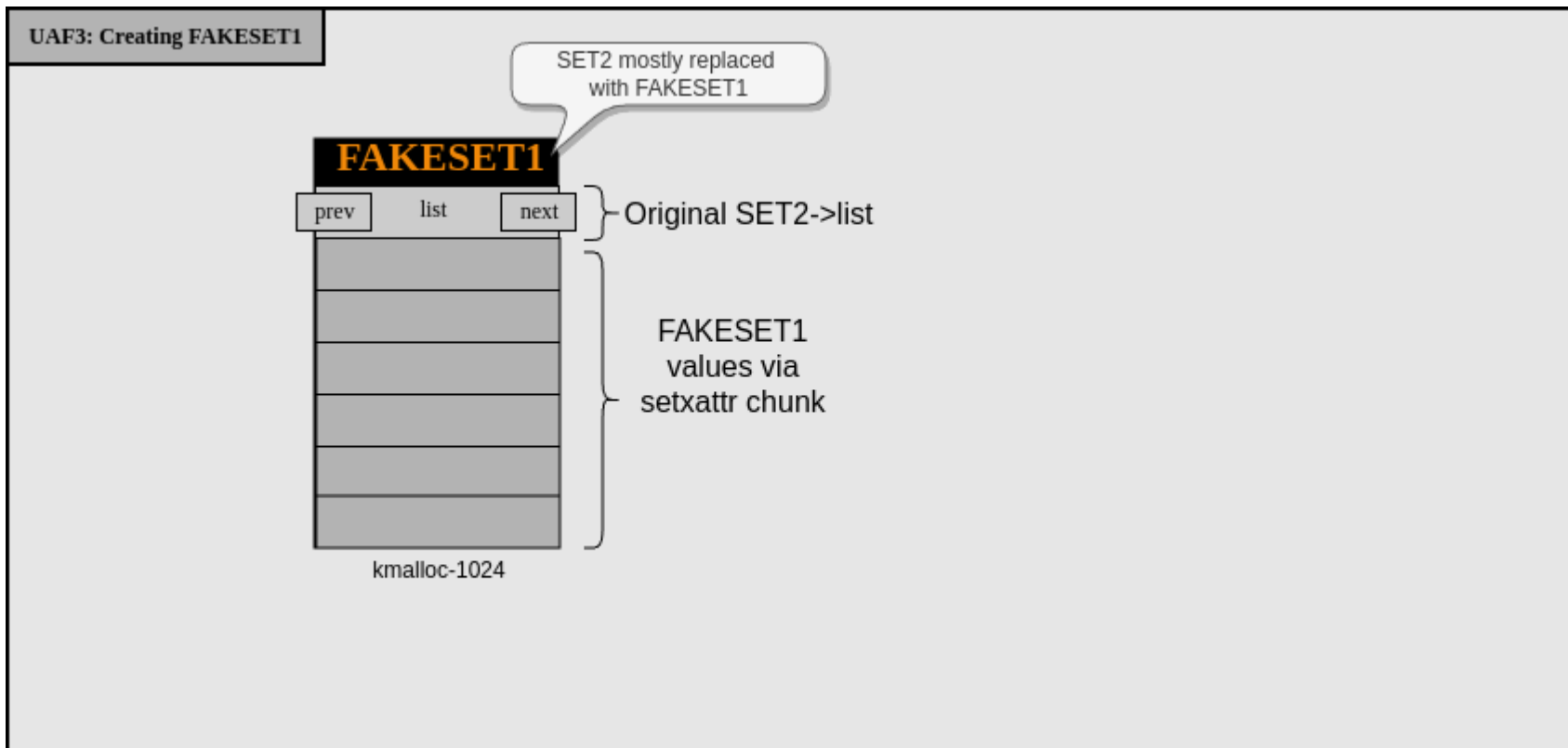


UAF3: FAKESET1 to Bypass KASLR



UAF3: FAKESSET1 to Bypass KASLR

- We can replace freed **SET2+0x10** chunk via FUSE and setxattr().





SET1 Memory Revelation

- We already know address of SET1, thanks to UAF1
 - The address we leaked with `keyctl(KEYCTL_READ)`



SET1 Memory Revelation

- We already know address of SET1, thanks to UAF1
 - The address we leaked with `keyctl(KEYCTL_READ)`
- Replace SET2 with FAKESET1
 - Use `setxattr()` call that blocks the kernel waiting on a controlled FUSE server



SET1 Memory Revelation

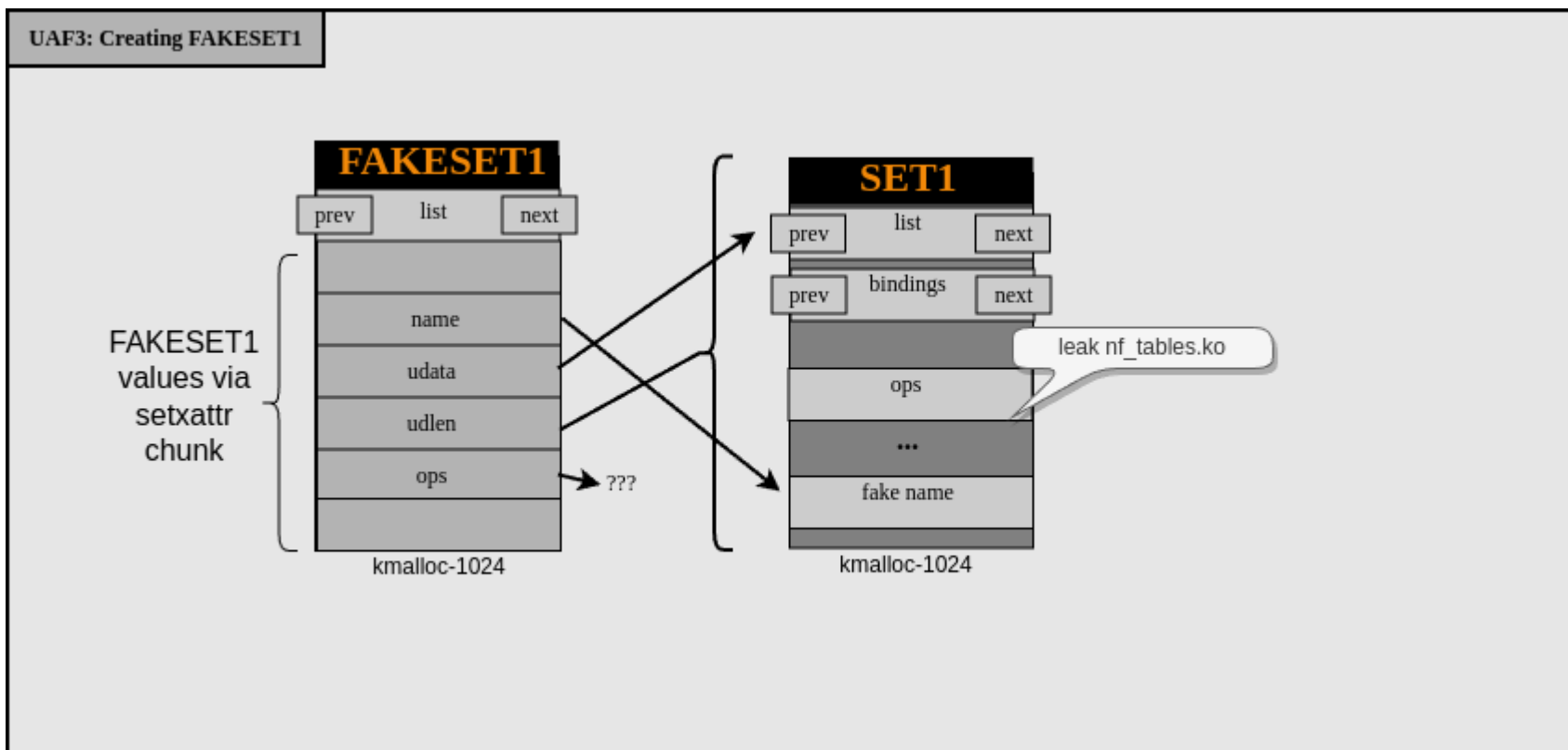
- We already know address of SET1, thanks to UAF1
 - The address we leaked with `keyctl(KEYCTL_READ)`
- Replace SET2 with FAKESET1
 - Use `setxattr()` call that blocks the kernel waiting on a controlled FUSE server
- FAKESET1->udata points to SET1
- FAKESET1->udlen at least `sizeof(SET1)`
- FAKESET1->name points to somewhere in SET1->data[] contents
 - This lets us continue lookup FAKESET1 via netlink



SET1 Memory Revelation

- We already know address of SET1, thanks to UAF1
 - The address we leaked with keyctl(KEYCTL_READ)
- Replace SET2 with FAKESET1
 - Use setxattr() call that blocks the kernel waiting on a controlled FUSE server
- FAKESET1->udata points to SET1
- FAKESET1->udlen at least sizeof(SET1)
- FAKESET1->name points to somewhere in SET1->data[] contents
 - This lets us continue lookup FAKESET1 via netlink
- Leak full SET1 contents
- Leaks nf_tables.ko's .data pointer via SET1->ops
 - Fairly limited for ROP gadgets

UAF3: FAKESSET1 to Bypass KASLR





Even Better Memory Revelation

- We can do better...



Even Better Memory Revelation

- We can do better...
- `nft_set->list`, linked list of sets on a table
- Create SET1 and SET2 on same table
- Leaking SET1->list->next gives us address of SET2 (aka FAKESSET1)
 - Allows us to craft future fake ops at known memory address



Even Better Memory Revelation

- We can do better...
- `nft_set->list`, linked list of sets on a table
- Create `SET1` and `SET2` on same table
- Leaking `SET1->list->next` gives us address of `SET2` (aka `FAKESET1`)
 - Allows us to craft future fake `ops` at known memory address
- `FAKESET1->udlen` is not limited to `sizeof(SET1)`
- We can also leak objects adjacent to `SET1`



Even Better Memory Revelation

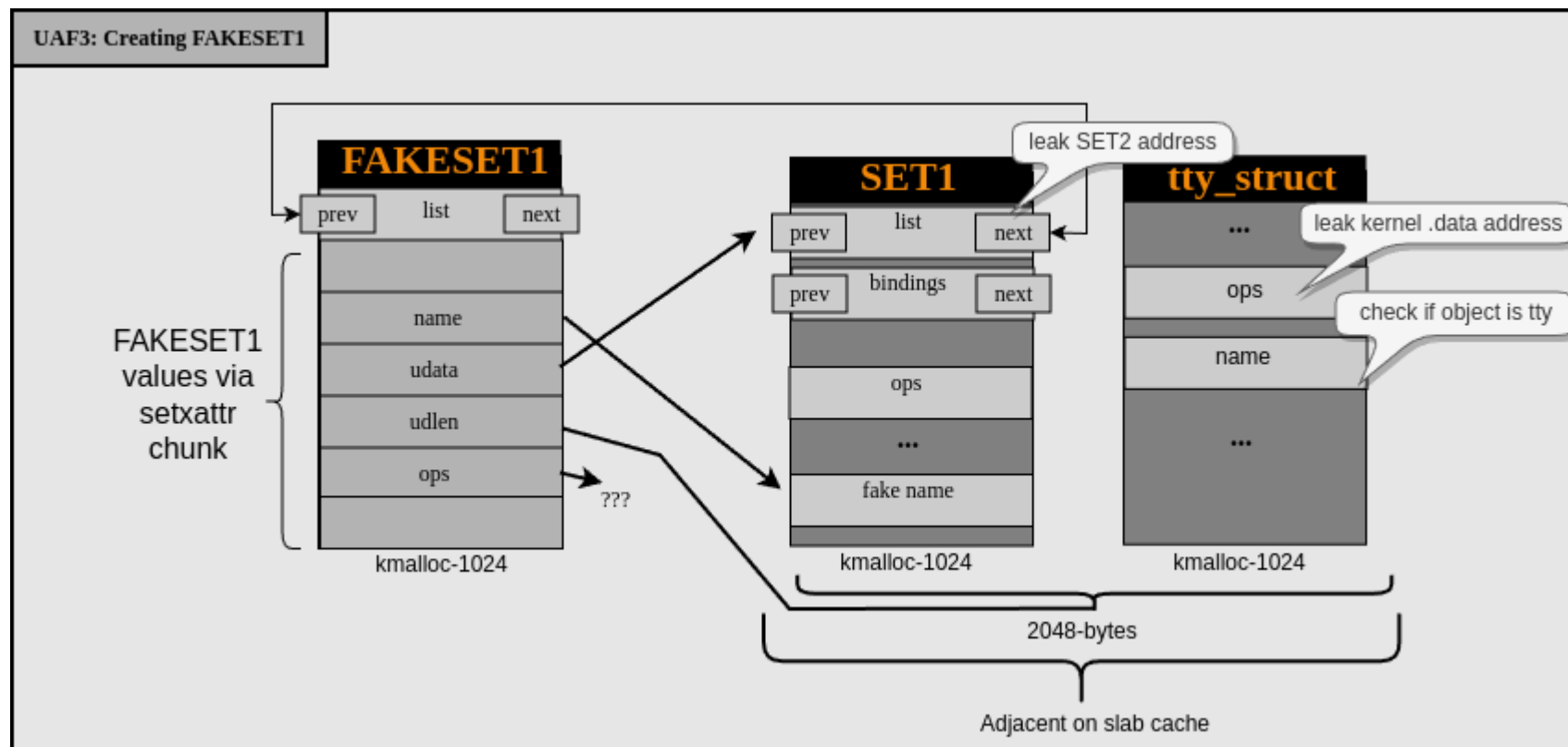
- We can do better...
- `nft_set->list`, linked list of sets on a table
- Create `SET1` and `SET2` on same table
- Leaking `SET1->list->next` gives us address of `SET2` (aka `FAKESET1`)
 - Allows us to craft future fake `ops` at known memory address
- `FAKESET1->udlen` is not limited to `sizeof(SET1)`
- We can also leak objects adjacent to `SET1`
- Spray `tty` objects prior to `SET1` creation
 - `open("/dev/ptmx", O_RDWR|O_NOCTTY);`
 - Places `tty_struct` on `kmalloc-1k`



Even Better Memory Revelation

- We can do better...
- `nft_set->list`, linked list of sets on a table
- Create `SET1` and `SET2` on same table
- Leaking `SET1->list->next` gives us address of `SET2` (aka `FAKESET1`)
 - Allows us to craft future fake `ops` at known memory address
- `FAKESET1->udlen` is not limited to `sizeof(SET1)`
- We can also leak objects adjacent to `SET1`
- Spray `tty` objects prior to `SET1` creation
 - `open("/dev/ptmx", O_RDWR|O_NOCTTY);`
 - Places `tty_struct` on `kmalloc-1k`
- Allows us to leak address from `vmlinux` (Better ROP gadgets)

UAF3: FAKESET1 to Bypass KASLR





UAF4: Getting Code Execution

- Now to put new KASLR-adjusted pointers in controlled memory



UAF4: Getting Code Execution

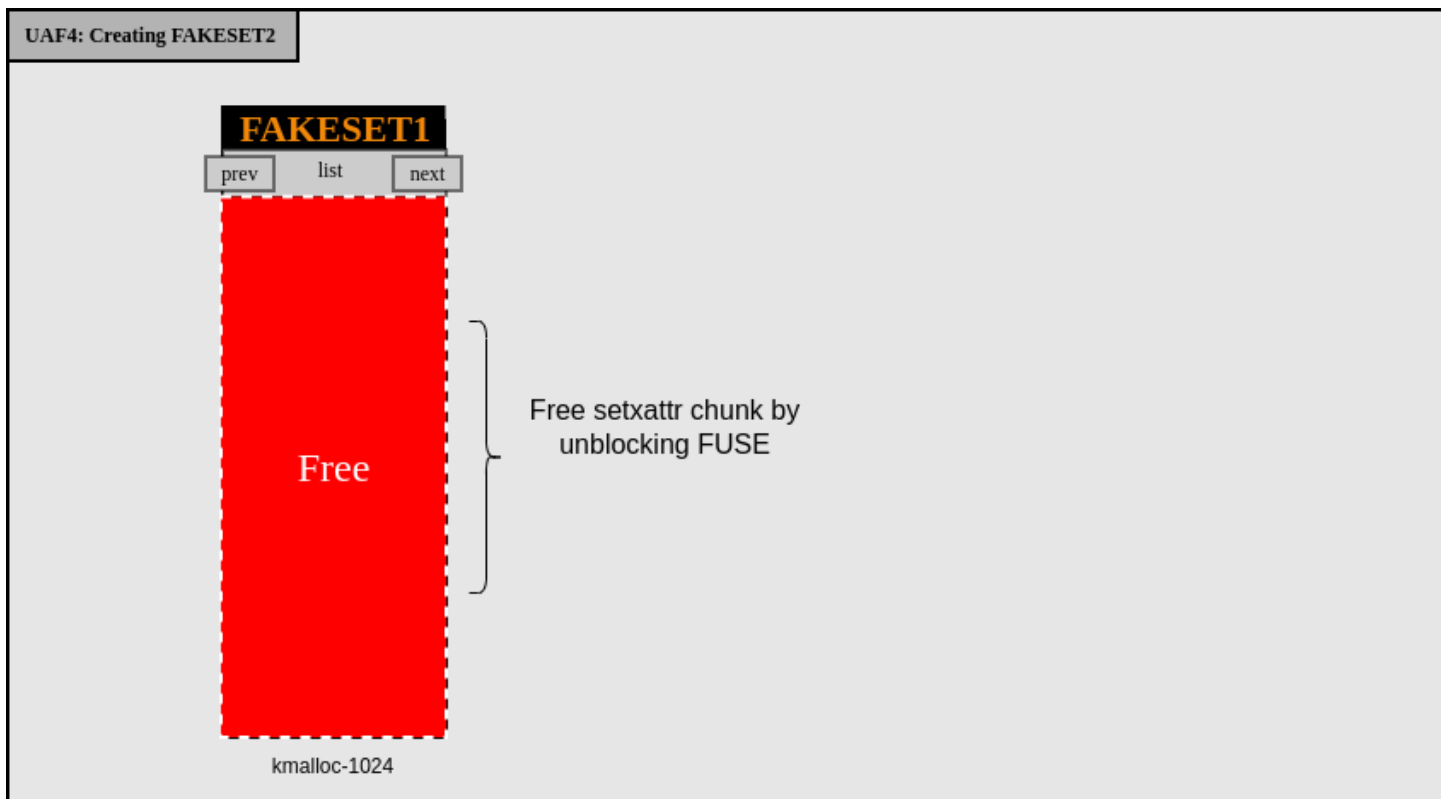
- Now to put new KASLR-adjusted pointers in controlled memory
- We just leaked the address of **FAKESET1**
- We control when **FAKESET1** is freed
 - Thanks to FUSE and **setxattr()**



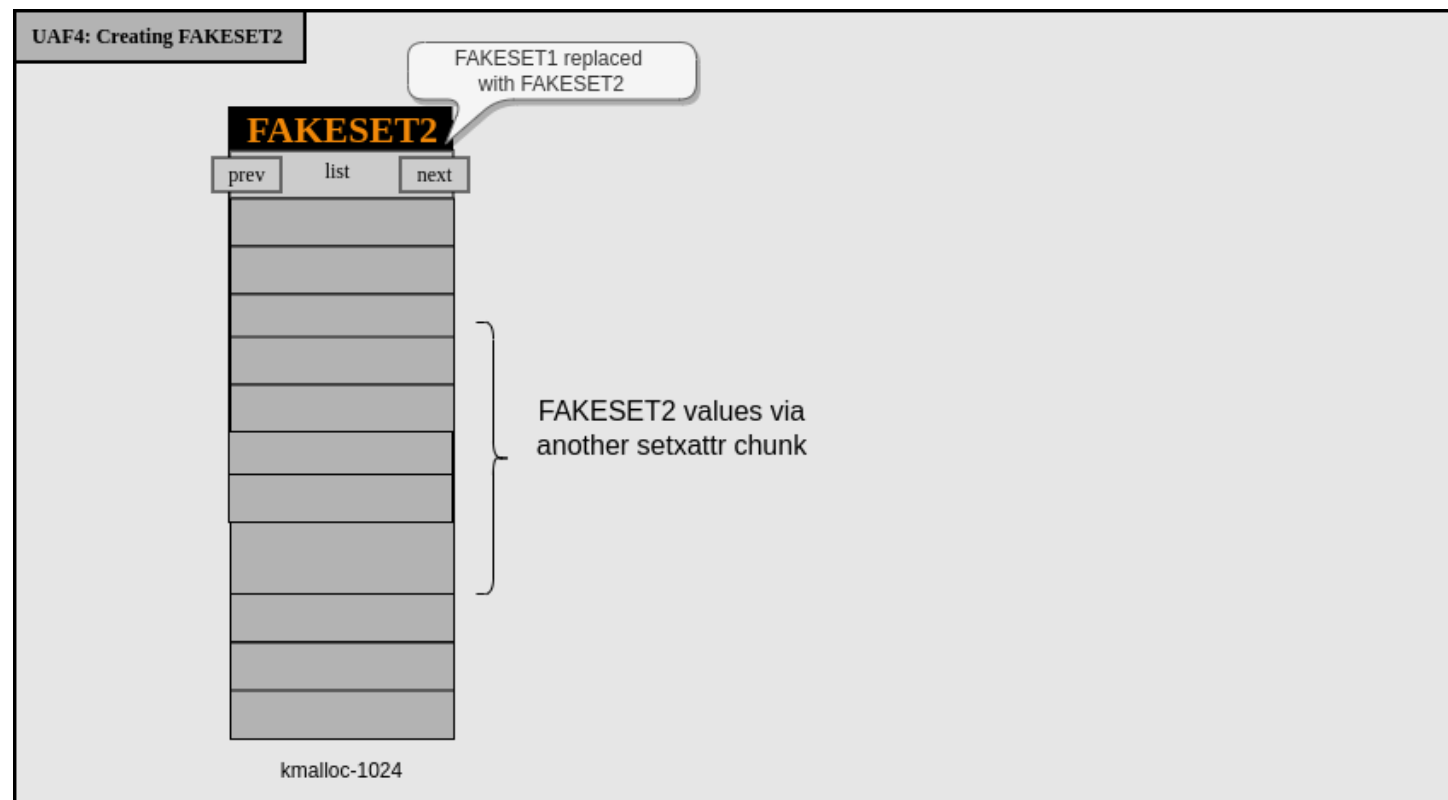
UAF4: Getting Code Execution

- Now to put new KASLR-adjusted pointers in controlled memory
- We just leaked the address of **FAKESET1**
- We control when **FAKESET1** is freed
 - Thanks to FUSE and **setxattr()**
- Can replace **FAKESET1** again with new data
 - We refer to this as **UAF4**
 - We will refer to the replaced **FAKESET1** as **FAKESET2**
- **FAKESET2->ops** points to a fake table in **FAKESET2->data**

UAF4: FAKESSET1 Replacement With FAKESSET2



UAF4: FAKESSET1 Replacement With FAKESSET2





ROP Gadget Hunting

- `nft_set->ops` function call register constraints are mostly:
 - Some functions: `rdi`, `r14` points to `FAKESET2`
 - Other functions: `rsi`, `r12` points to `FAKESET2`
- `FAKESET2` completely controlled
 - So most offsets into the object could be useful
- Find a gadget that does something interesting with this data
- Preferably fetch controlled pointer and then write there controlled data
- We did manual hunting using public tools `rp`



__hlist_del gadget

- Function offsets happen to perfectly overlap with controlled set values

```
1 pwndbg> x/10i __hlist_del
2 <perf_swevent_del>:      mov     rax,QWORD PTR [rdi+0x60] // this overlaps with set->field_count and set->use
3 <perf_swevent_del+4>:    mov     rdx,QWORD PTR [rdi+0x68] // this overlaps with set->nelems
4 <perf_swevent_del+8>:    mov     QWORD PTR [rdx],rax // this lets us write 8-bytes to controlled address
5 <perf_swevent_del+11>:   test    rax,rax
6 <perf_swevent_del+14>:   je      0xffffffff812795e4 <perf_swevent_del+20>
7 <perf_swevent_del+16>:   mov     QWORD PTR [rax+0x8],rdx // this will OOPS if rax is an invalid address
8 <perf_swevent_del+20>:   movabs  rax,0xdead000000000122
9 <perf_swevent_del+30>:   mov     QWORD PTR [rdi+0x68],rax
10 <perf_swevent_del+34>:  ret
11
```



Unsafe Double Unlink

- Double unlink will OOPS after our controlled write!
- Problem? Nope...
 - Ubuntu uses `panic_on_oops=0` `sysctl` so we don't actually care
- Quite similar to recent STAR Labs io_uring `__list_del` technique
 - But we don't leak or need physmap

```
1 panic_on_oops:
2
3 Controls the kernel's behaviour when an oops or BUG is encountered.
4
5 0: try to continue operation
6
7 1: panic immediately. If the `panic' sysctl is also non-zero then the
8    machine will be rebooted.
```



Invoking Gadget

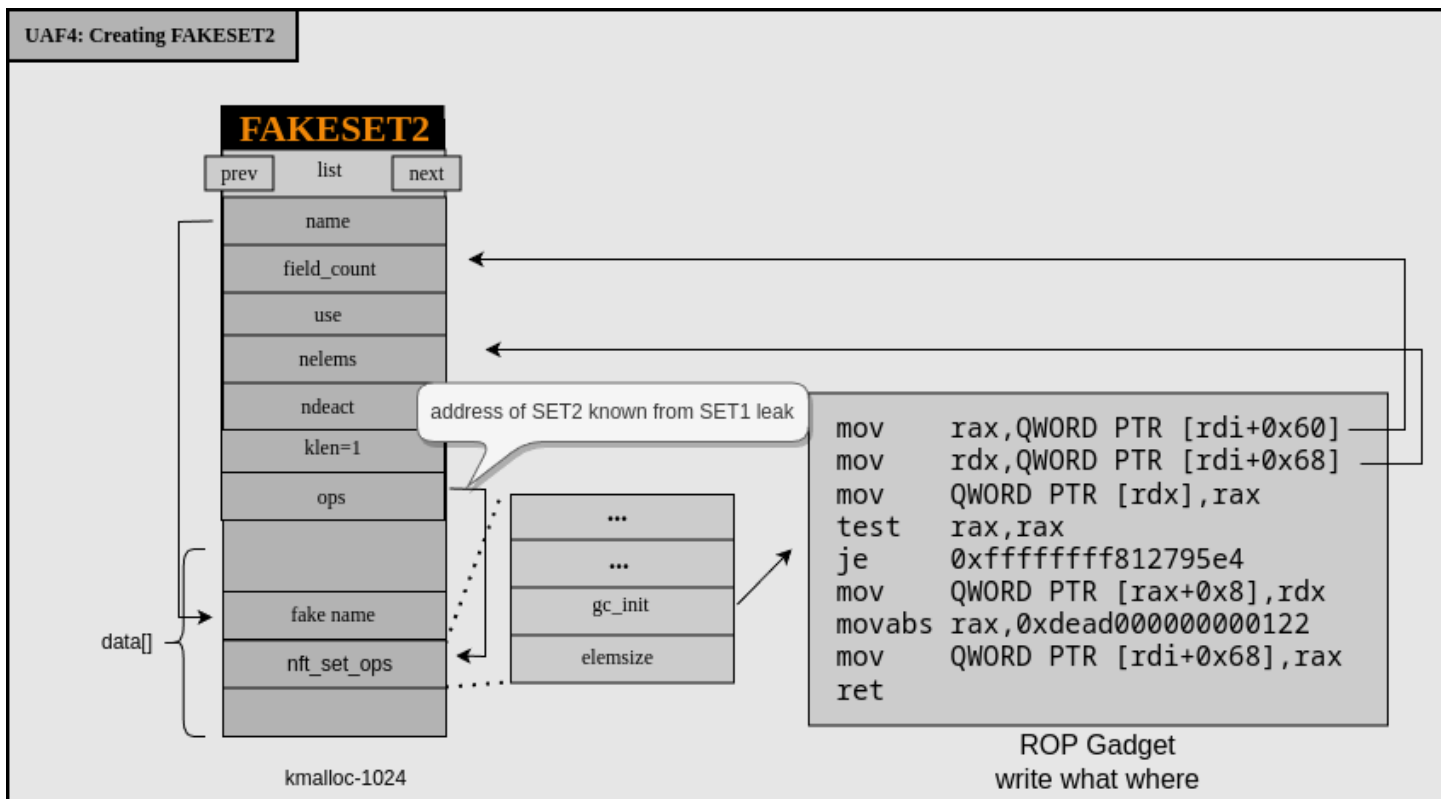
- We chose to use `nft_set->ops->gc_init()` to trigger ROP gadget
- Require some setup and explicit expression type to trigger
- Requires an expression with `NFT_EXPR_GC` flag
- `nft_connlimit` is only one with this flag
- If flag set, `gc_init()` invoked during expression initialization



Targeting modprobe_path

- We chose to write to `modprobe_path` for quick win
- Well documented and widely used technique by now
 - Overwrite kernel string holding binary path, execute new path as root
- We write a 8-byte address that we can also use as a string
 - Ex: `/tmp/x\0`
- Obviously some real-world limitations
 - `/tmp/` mounted as non-executable, etc
 - Per-container temporary folder different from executing context

UAF4: FAKESET2 For Code Execution





Putting It All Together

- Trigger 4 UAF scenarios
- **UAF1**: Replace `nft_dynset` with `user_key_payload` and leak `SET1` address



Putting It All Together

- Trigger 4 UAF scenarios
- **UAF1**: Replace `nft_dynset` with `user_key_payload` and leak `SET1` address
- **UAF2**: Replace `nft_dynset` with `cgroup_fs_context` and overwrite `cgroup_fs_context->release_agent`



Putting It All Together

- Trigger 4 UAF scenarios
- **UAF1**: Replace `nft_dynset` with `user_key_payload` and leak **SET1** address
- **UAF2**: Replace `nft_dynset` with `cgroup_fs_context` and overwrite `cgroup_fs_context->release_agent`
- **UAF3**: Destroy cgroup to free **SET2** and replace with **FAKESET1**
- Bypass KASLR and leak address of **SET2** and by ""reading **SET1** and adjacent slab memory



Putting It All Together

- Trigger 4 UAF scenarios
- **UAF1**: Replace `nft_dynset` with `user_key_payload` and leak **SET1** address
- **UAF2**: Replace `nft_dynset` with `cgroup_fs_context` and overwrite `cgroup_fs_context->release_agent`
- **UAF3**: Destroy cgroup to free **SET2** and replace with **FAKESET1**
- Bypass KASLR and leak address of **SET2** and by ""reading **SET1** and adjacent slab memory
- **UAF4**: Replace **FAKESET1** with **FAKESET2** and `ops` now pointing to valid gadget



Putting It All Together

- Trigger 4 UAF scenarios
- **UAF1**: Replace `nft_dynset` with `user_key_payload` and leak **SET1** address
- **UAF2**: Replace `nft_dynset` with `cgroup_fs_context` and overwrite `cgroup_fs_context->release_agent`
- **UAF3**: Destroy cgroup to free **SET2** and replace with **FAKESET1**
- Bypass KASLR and leak address of **SET2** and by ""reading **SET1** and adjacent slab memory
- **UAF4**: Replace **FAKESET1** with **FAKESET2** and `ops` now pointing to valid gadget
- Trigger `gc_init()` to overwrite `modprobe_path`
- Trigger module load from userland and get root



Aftermath



Patch Analysis

- Prevented the initialization of any non-stateful expression during set creation
- This should actually kill a lot of underlying bugs
- BONUS: Fix also stops a separate reference counting bug we had found
- Fixed [here](#)

Patch

- **NFT_EXPR_STATEFUL** flag is now checked prior to allocation

```

1 static struct nft_expr *nft_expr_init(const struct nft_ctx *ctx,
2                                     const struct nlattr *nla)
3 {
4     struct nft_expr_info expr_info;
5     struct nft_expr *expr;
6     struct module *owner;
7     int err;
8
9     err = nf_tables_expr_parse(ctx, nla, &expr_info);
10    if (err < 0)
11        goto err_expr_parse;
12
13    err = -EOPNOTSUPP;
14    if (!(expr_info.ops->type->flags & NFT_EXPR_STATEFUL))
15        goto err_expr_stateful;
16
17    err = -ENOMEM;
18    expr = kzalloc(expr_info.ops->size, GFP_KERNEL_ACCOUNT);
19    [...]
20 }
21

```

Parse expression info without initializing

Check flag before initialization



Conclusion

- netlink and `nf_tables` is a fairly rich attacks surface
 - Lots of new bugs/writeups/exploits in 2022
- Same old tune:
 - Unprivileged namespaces still seems very risky to have enabled
 - `panic_on_oops=0` is dangerous
 - Userland FUSE server + `setxattr()` is very powerful
 - Writable `modprobe_path` remains a big weakness
- `msg_msg` is popular for many exploits, but not explicitly required
- Constructing bug-specific primitives is still very feasible!



Mitigations / Prevention

- How to avoid exploitation of these types of bugs?
- Prevent ability to free misaligned slab cache addresses
- More object-specific slab caches to reduce UAF replacement possibilities
 - grsecurity's autoslab
 - Google's experimental mitigations
- CFI to avoid ROP gadget execution
 - No idea when it's available for x64?
- `panic_on_oops=1` to prevent unlink trick
 - Fairly inconvenient in the real world
- Read-only `modprobe_path` via `CONFIG_STATIC_USERMODEHELPER`
- Disable unprivileged namespaces
- Disable userland FUSE server support



Contact

- Accompanying blog will be released shortly with a lot more details
- EDG team group effort
 - Aaron Adams: @fidgetingbits
 - Cedric Halbronn: @saidelike
 - Alex Plaskett: @alexjplaskett
- We are hiring!

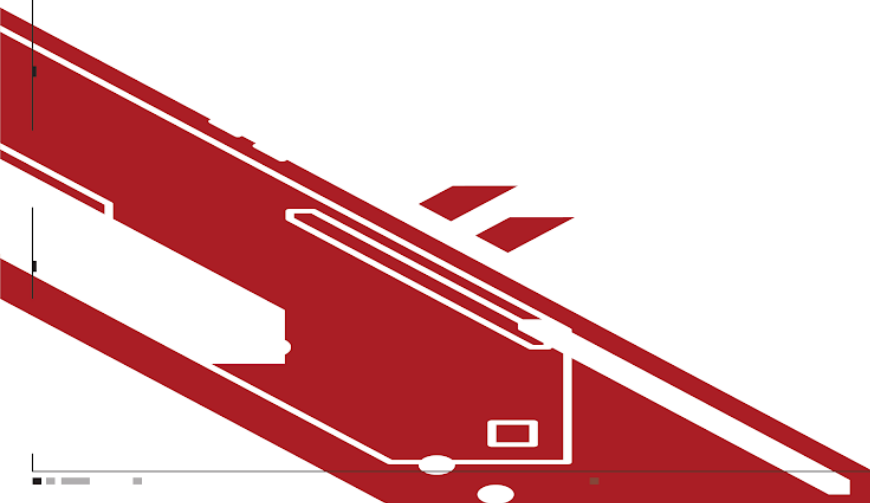


Talon Voice Coding

- I have bad RSI for a really long time
- For the last ~2 years I've used voice coding and eye tracking for my 99% of work/research
- Shout out to @linuxbochs's voice coding framework Talon
- Take care of your hands/body everyone!



HITBSecConf
2022 Singapore



#HITB2022SIN