HITBSecConf 2022 Singapore

Settlers of Netlink

Exploiting a limited kernel UAF on Ubuntu 22.04

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

- 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`

```
struct nft_set {
    struct list_head   list;
    struct list_head   bindings;
    [...]
    char               *name;
    [...]
    u8                 field_count;
    u32                use;
    atomic_t           nelems;
    u32                ndeact;
    [...]
    u16                udlen;
    unsigned char      *udata;
    struct nft_set_ops *ops;
    [...]
    u8                 num_exprs;
    struct nft_expr    *exprs[NFT_SET_EXPR_MAX];
    struct list_head   catchall_list;
    unsigned char      data[]
};

```

`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

```
struct nft_set_binding {
    struct list_head        list;
    const struct nft_chain  *chain;
    u32                     flags;
};
```

- 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

```
struct nft_expr {
    const struct nft_expr_ops    *ops;
    unsigned char                data[]
        __attribute__((aligned(__alignof__(u64))));
};

static inline void *nft_expr_priv(const struct nft_expr *expr)
{
    return (void *)expr->data;
}

```

- Typical use:

```
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

```
struct nft_lookup {
    struct nft_set      *set;
    u8                  sreg;
    u8                  dreg;
    bool                invert;
    struct nft_set_binding binding;
};

```

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

```c
struct nft_dynset {
    struct nft_set          *set;
    struct nft_set_ext_tmpl tmpl;
    enum nft_dynset_ops     op:8;
    u8                      sreg_key;
    u8                      sreg_data;
    bool                    invert;
    bool                    expr;
    u8                      num_exprs;
    u64                     timeout;
    struct nft_expr         *expr_array[NFT_SET_EXPR_MAX];
    struct nft_set_binding  binding;
};

```

Normal Set Expression Binding Relationship

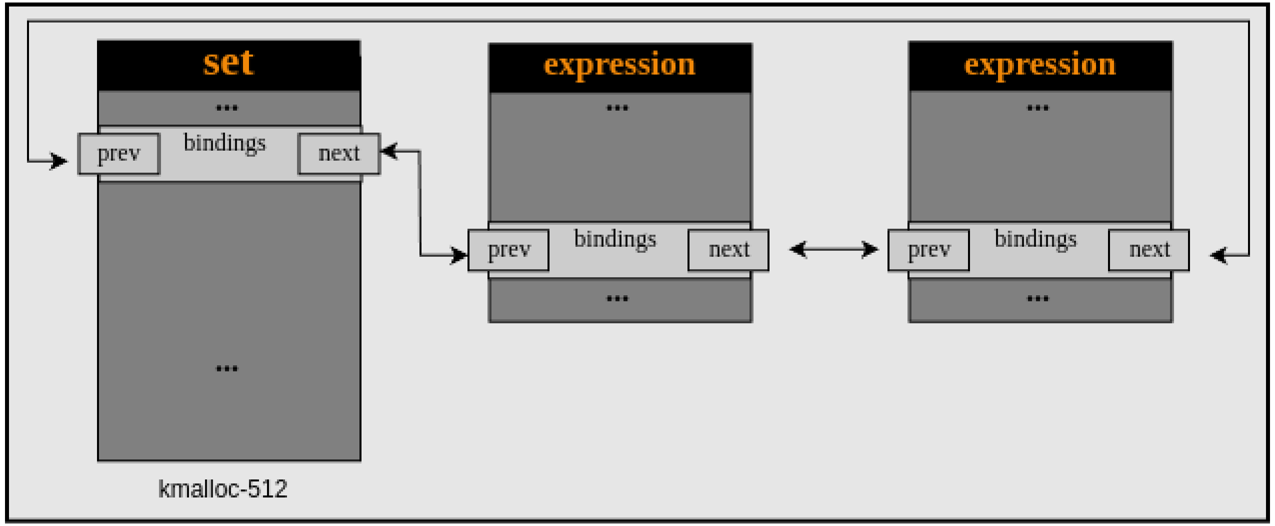
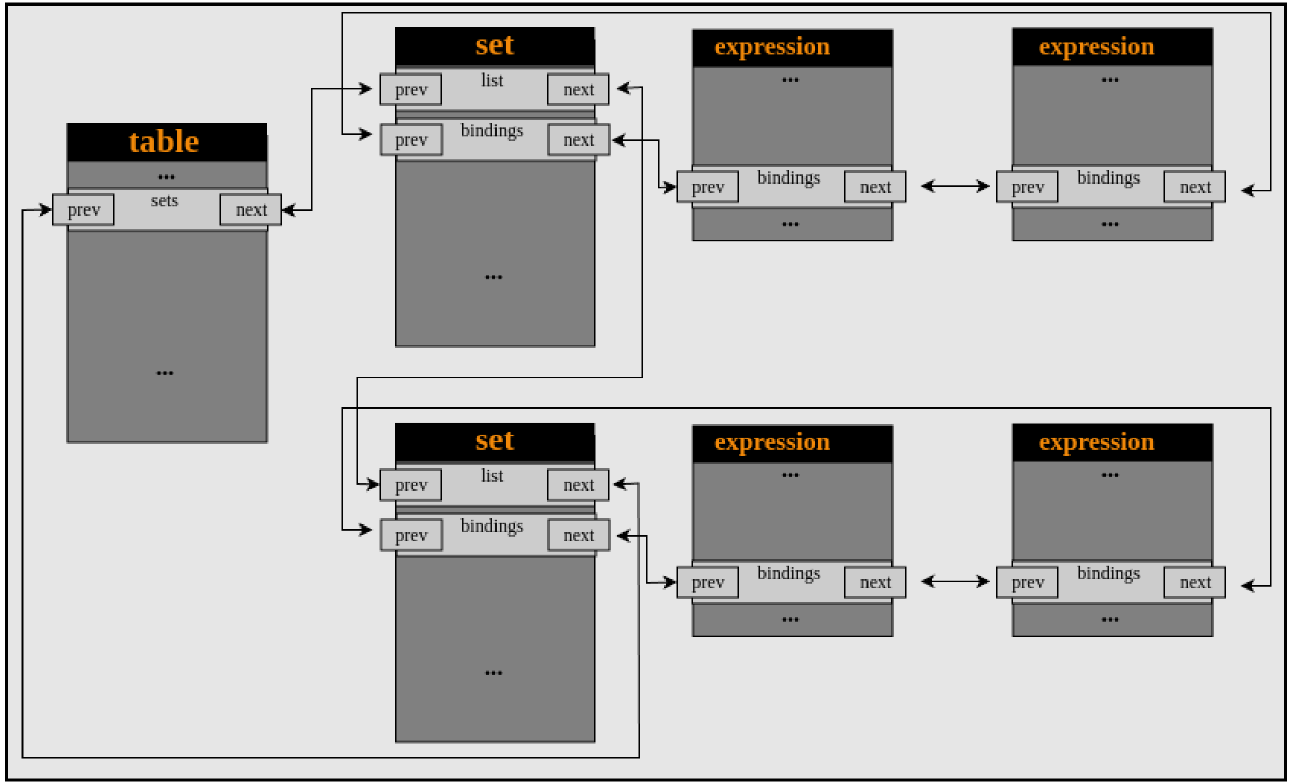


Table With Linked Sets



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

```c
struct nft_expr *nft_set_elem_expr_alloc(const struct nft_ctx *ctx,
                     const struct nft_set *set,
                     const struct nlattr *attr)
{
    struct nft_expr *expr;
    int err;

    expr = nft_expr_init(ctx, attr);
    if (IS_ERR(expr))
        return expr;
    err = -EOPNOTSUPP;
    if (!(expr->ops->type->flags & NFT_EXPR_STATEFUL))
        goto err_set_elem_expr;

    [...]
    return expr;

err_set_elem_expr:
    nft_expr_destroy(ctx, expr);
    return ERR_PTR(err);
}
```

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

```
void nft_expr_destroy(const struct nft_ctx *ctx, struct nft_expr *expr)
{
    nf_tables_expr_destroy(ctx, expr);
    kfree(expr);
}

static void nf_tables_expr_destroy(const struct nft_ctx *ctx,
                   struct nft_expr *expr)
{
    const struct nft_expr_type *type = expr->ops->type;

    if (expr->ops->destroy)
        expr->ops->destroy(ctx, expr);
    module_put(type->owner);
}

```

Lookup and Dynset Expressions

HITBSecConf 2022 Singapore

- 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

```
static int nft_dynset_init(const struct nft_ctx *ctx,
                           const struct nft_expr *expr,
                           const struct nlattr * const tb[])
{
    struct nftables_pernet *nft_net = nft_pernet(ctx->net);
    struct nft_dynset *priv = nft_expr_priv(expr);
    [...]
    err = nf_tables_bind_set(ctx, set, &priv->binding);
    if (err < 0)
        goto err_expr_free;

    if (set->size == 0)
        set->size = 0xffff;

    priv->set = set;
    return 0;
    [...]
}

```

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()

```
static void nft_dynset_destroy(const struct nft_ctx *ctx,
                               const struct nft_expr *expr)
{
    struct nft_dynset *priv = nft_expr_priv(expr);
    int i;

    for (i = 0; i < priv->num_exprs; i++)
        nft_expr_destroy(ctx, priv->expr_array[i]);

    nf_tables_destroy_set(ctx, priv->set);
}

```

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

```
void nf_tables_destroy_set(const struct nft_ctx *ctx, struct nft_set *set)
{
    if (list_empty(&set->bindings) && nft_set_is_anonymous(set))
        nft_set_destroy(ctx, set);
}

```

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

HITBSecConf 2022 Singapore

- 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

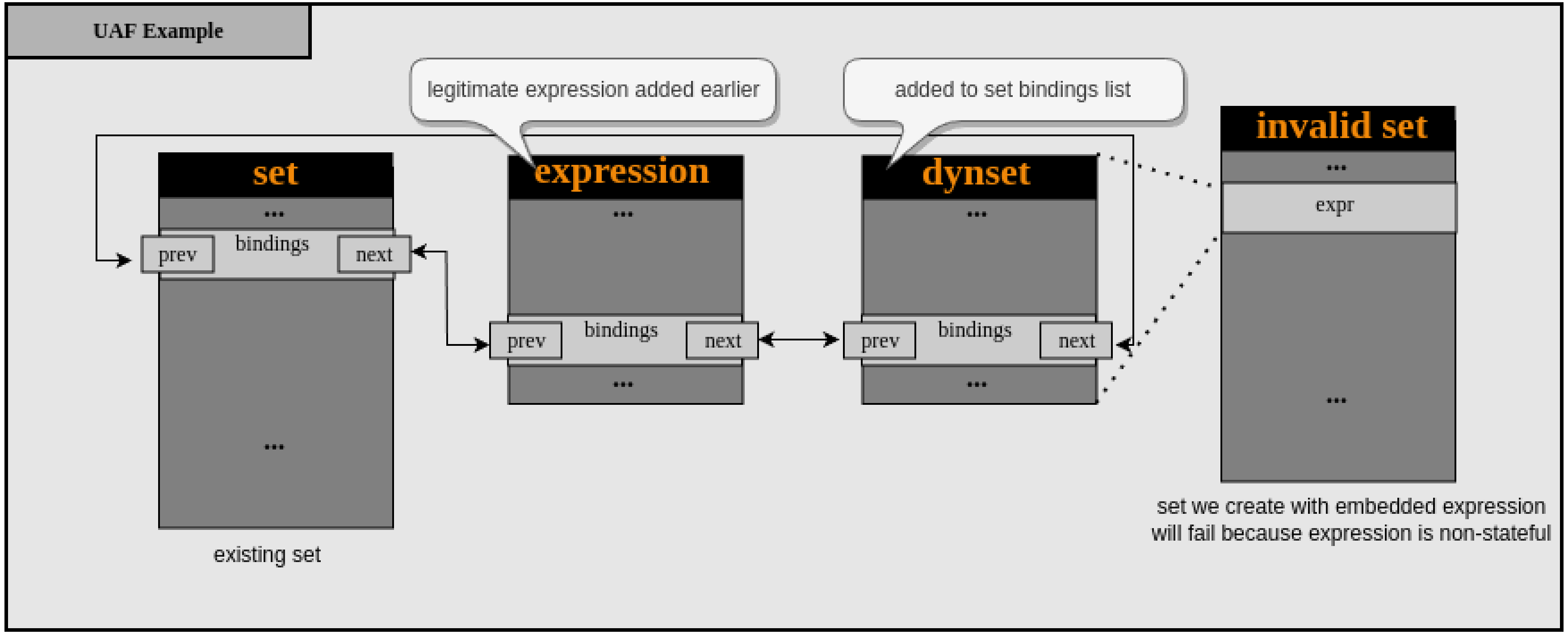
- Create a valid set that expressions we initialize can reference
- Bind a expression to the valid set, to populate set->bindings with one entry
- Create a new invalid set
- Embed "lookup" or "dynset" expression in the invalid set
    - Embedded expression references valid set
    - Added to the set->bindings 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

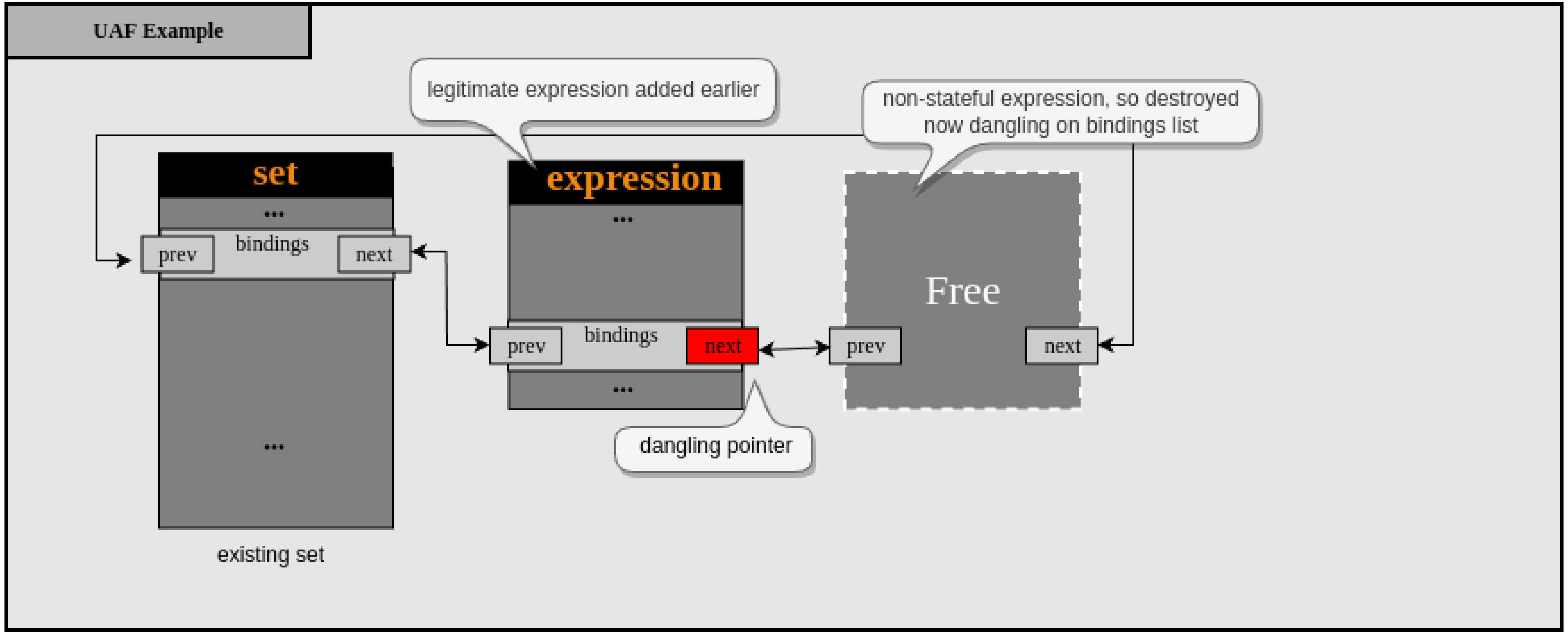
HITBSecConf 2022 Singapore

- 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

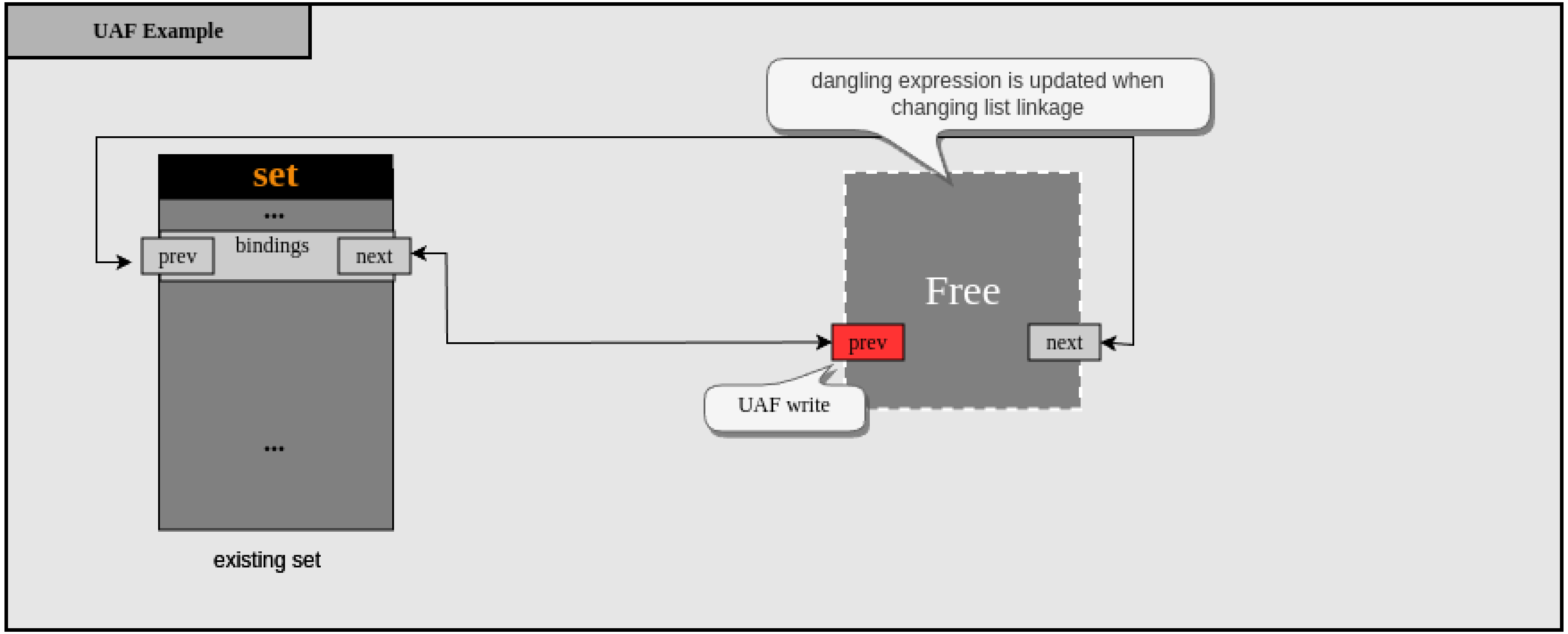
Non-Stateful Expression Added to Bindings List



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?

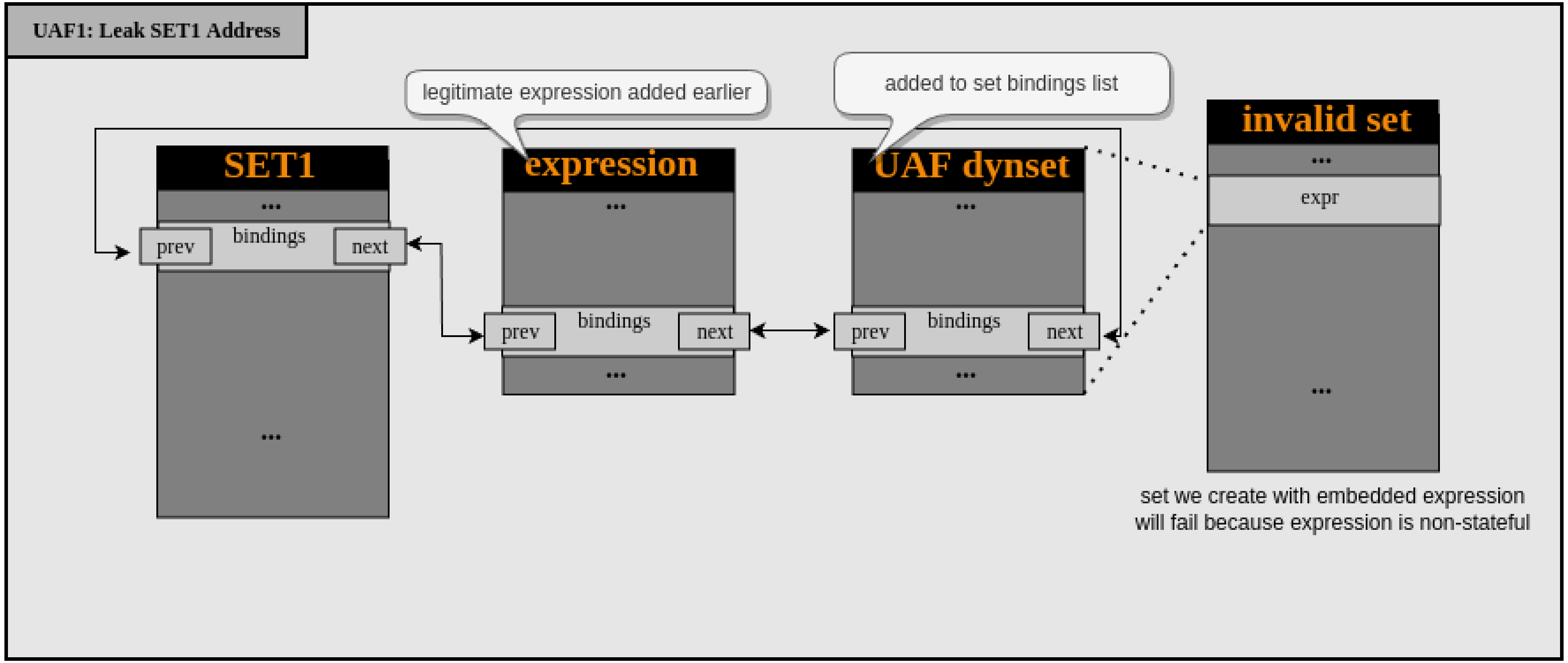
Easy Win: Leak Some Address

- 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
  - key_ctl(KEYCTL_READ): Can read payload contents at any time

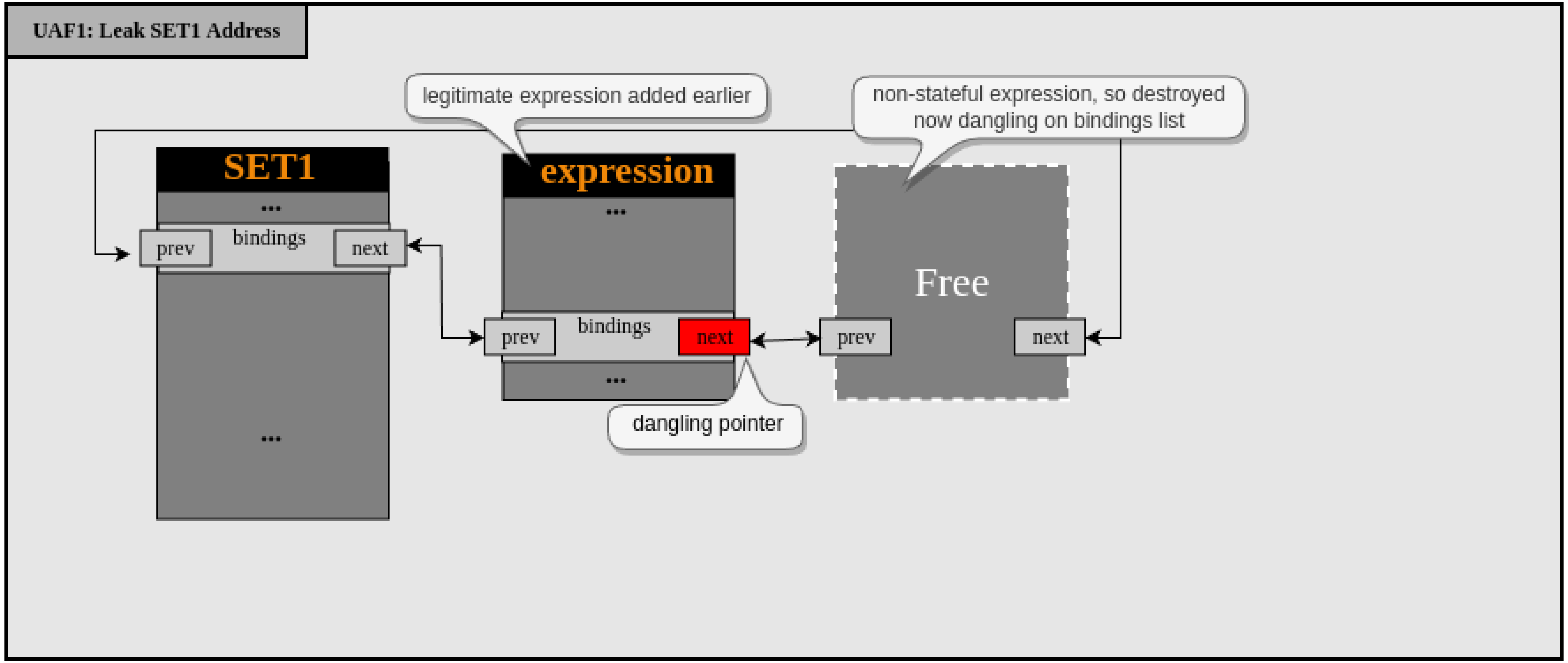
Easy Win: Leak Some Address

- 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
  - key_ctl(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

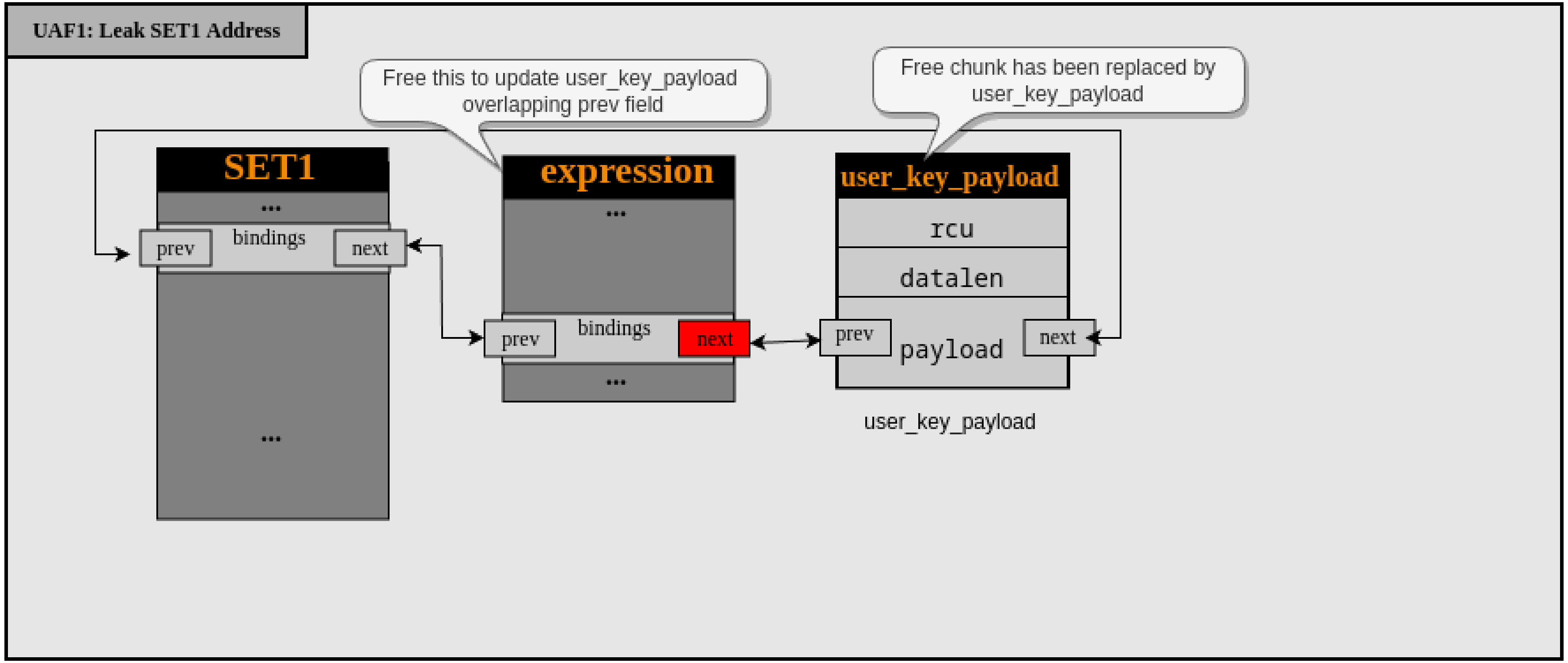
UAF1: SET1 Address Leak



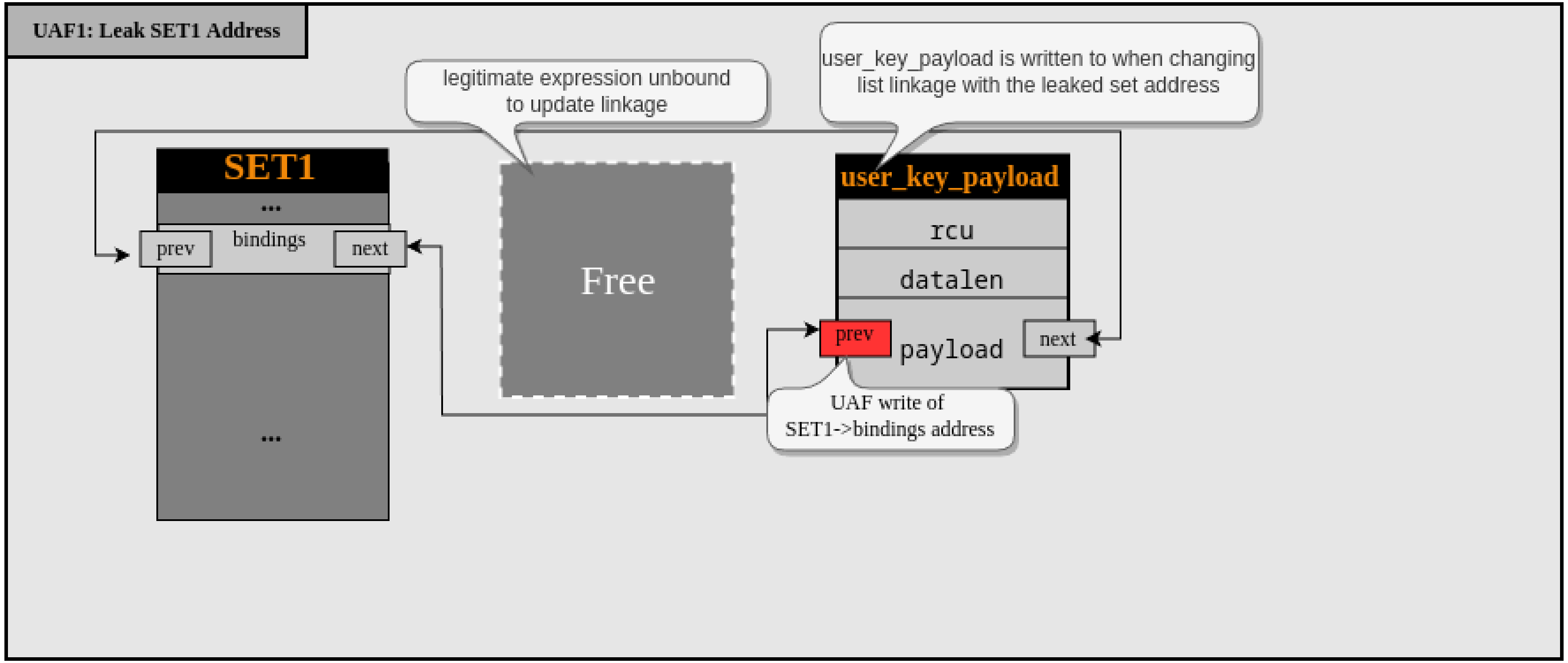
UAF1: SET1 Address Leak



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

```
import cpp

from FunctionCall fc, Type t, Variable v, Field f, Type t2
where (fc.getTarget().hasName("kmalloc") or
       fc.getTarget().hasName("kzalloc") or
       fc.getTarget().hasName("kcalloc"))
      and
      exists(Assignment assign | assign.getRValue() = fc and
             assign.getLValue() = v.getAnAccess() and
             v.getType().(PointerType).refersToDirectly(t)) and
      t.getSize() <= 96 and t.getSize() > 64 and t.fromSource() and
      f.getDeclaringType() = t and
      (f.getType().(PointerType).refersTo(t2) and t2.getSize() <= 8) and
      (f.getByteOffset() = 72)
select fc, t, fc.getLocation()
```

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

```c
struct cgroup_fs_context {
    struct kernfs_fs_context kfc;
    struct cgroup_root    *root;
    struct cgroup_namespace    *ns;
    unsigned int    flags;            /* CGRP_ROOT_* flags */

    /* cgroup1 bits */
    bool        cpuset_clone_children;
    bool        none;            /* User explicitly requested empty subsystem */
    bool        all_ss;            /* Seen 'all' option */
    u16        subsys_mask;        /* Selected subsystems */
    char        *name;            /* Hierarchy name */
    char        *release_agent;        /* Path for release notifications */
};
```

Freeing `release_agent`

```
static void cgroup_fs_context_free(struct fs_context *fc)
{
    struct cgroup_fs_context *ctx = cgroup_fc2context(fc);

    kfree(ctx->name);
    kfree(ctx->release_agent);
    put_cgroup_ns(ctx->ns);
    kernfs_free_fs_context(fc);
    kfree(ctx);
}

```

Free primitives

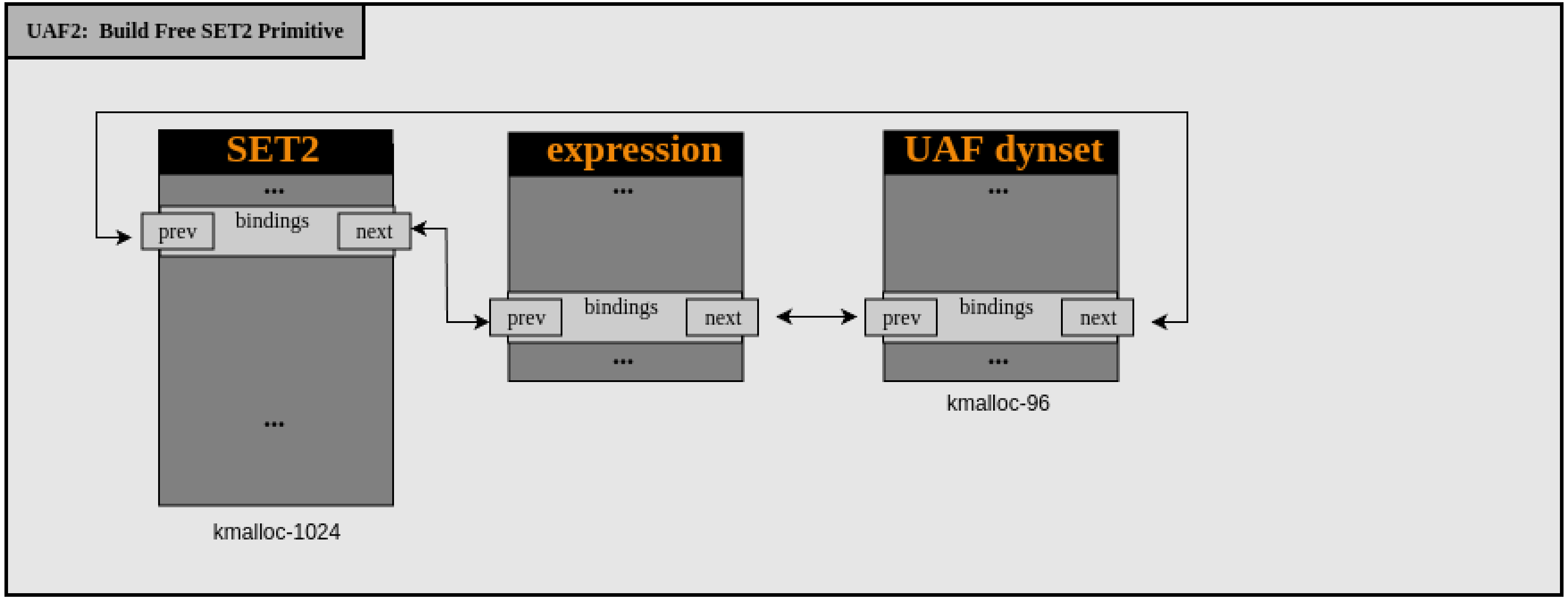
Preparing a Set Freeing Primitive

HITBSecConf 2022 Singapore

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

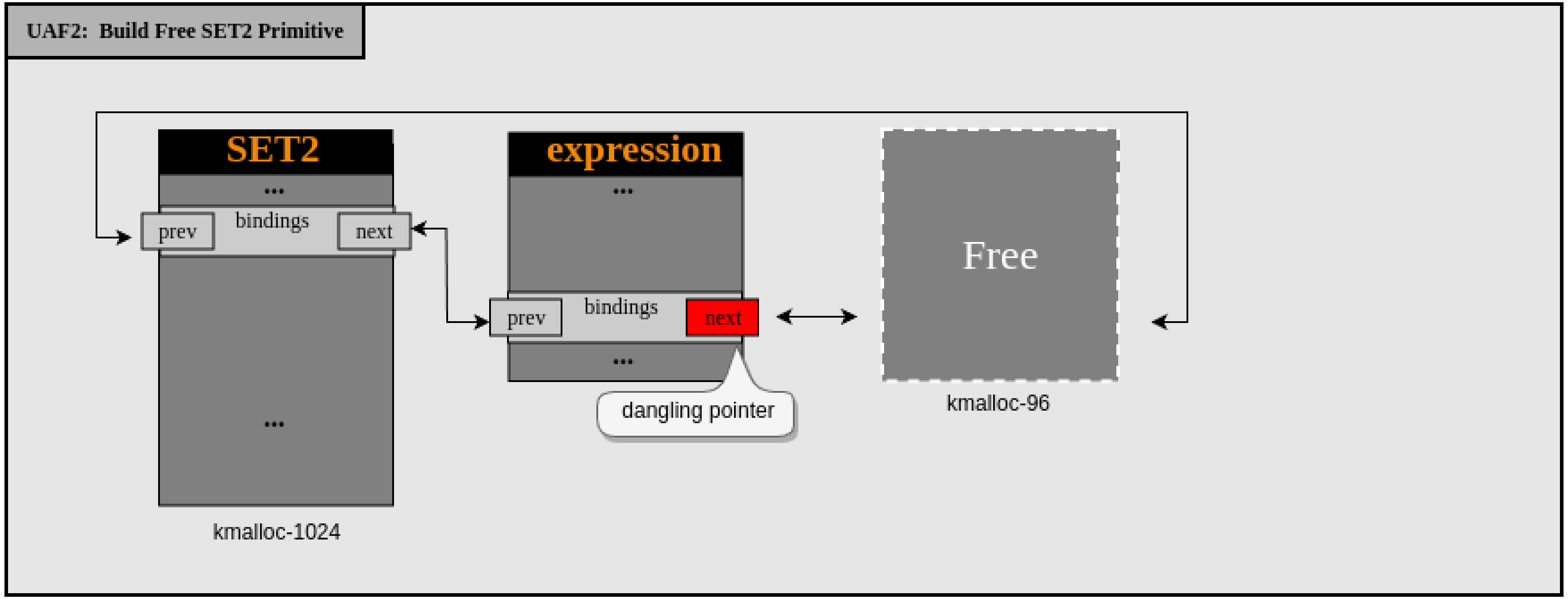
UAF2: `release_agent` Overwrite

- Trigger set->bindings UAF with a nft_dynset expression



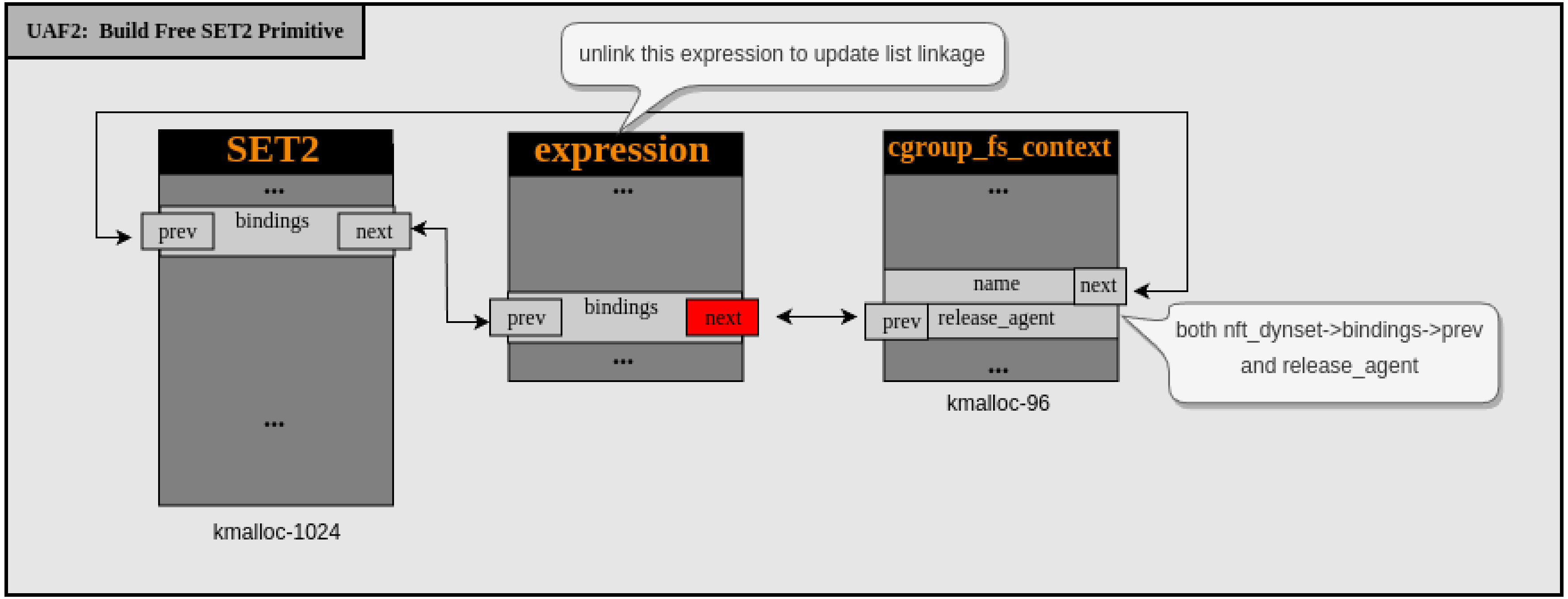
UAF2: `release_agent` Overwrite

- Replace nft_dynset with a cgroup_fs_context



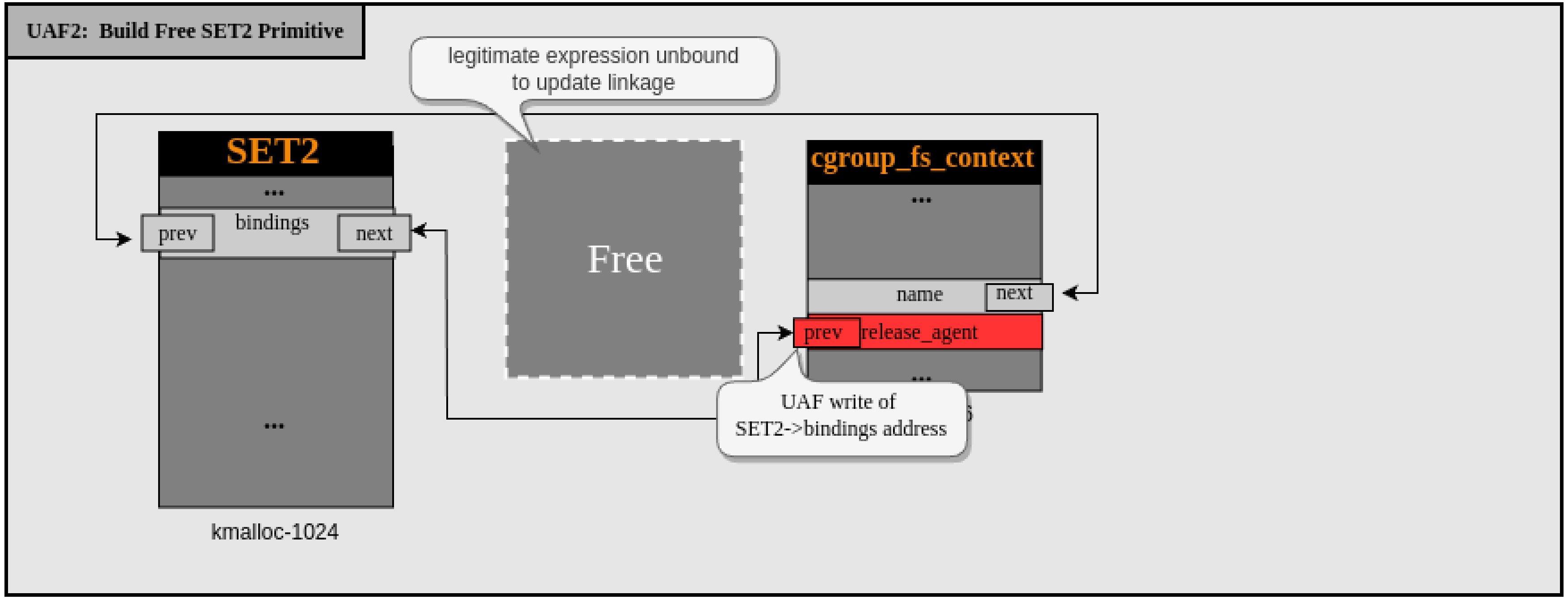
UAF2: `release_agent` Overwrite

- Remove an entry from the set->bindings



UAF2: `release_agent` Overwrite

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

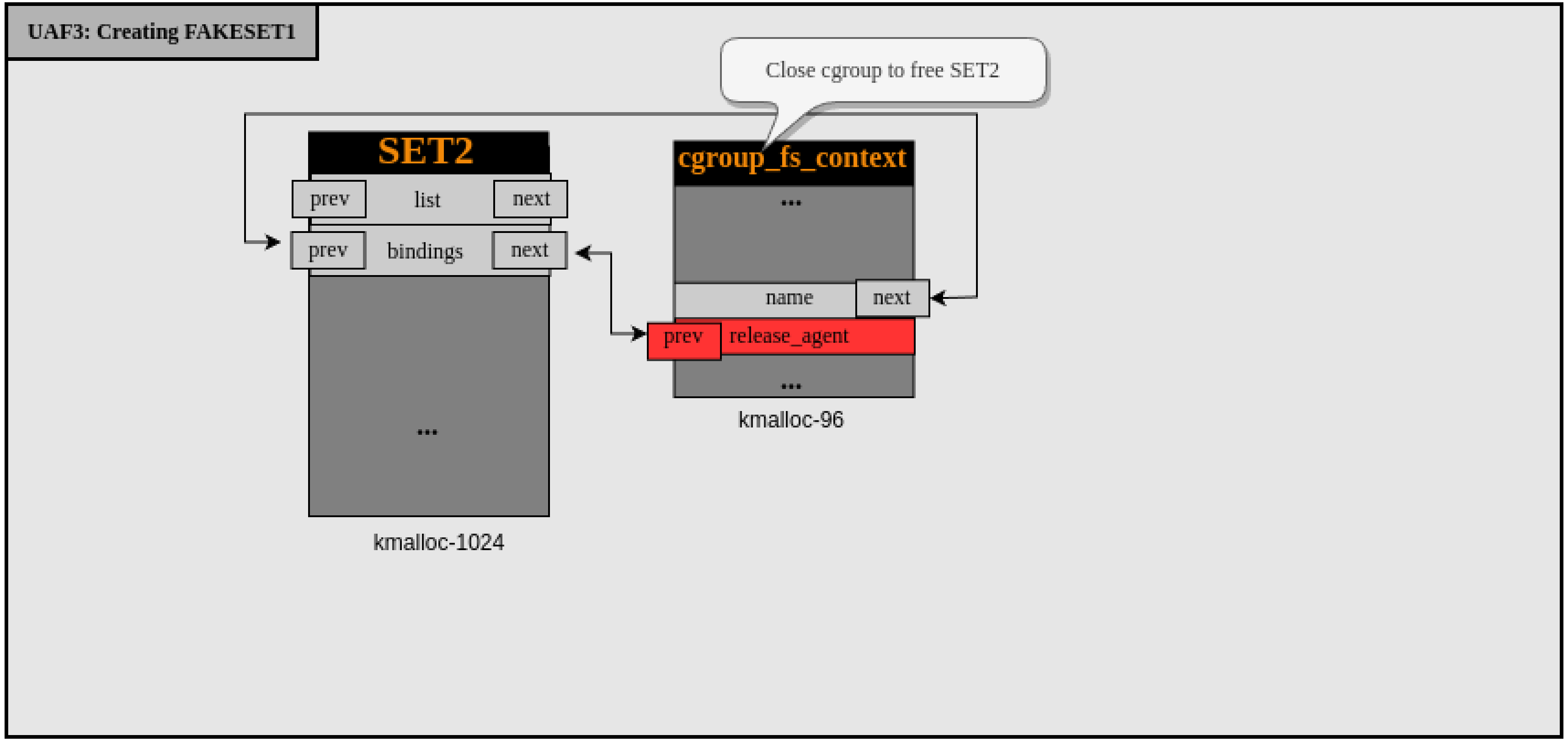


Freeing and Replacing a Set

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

UAF3: FAKESET1 to Bypass KASLR

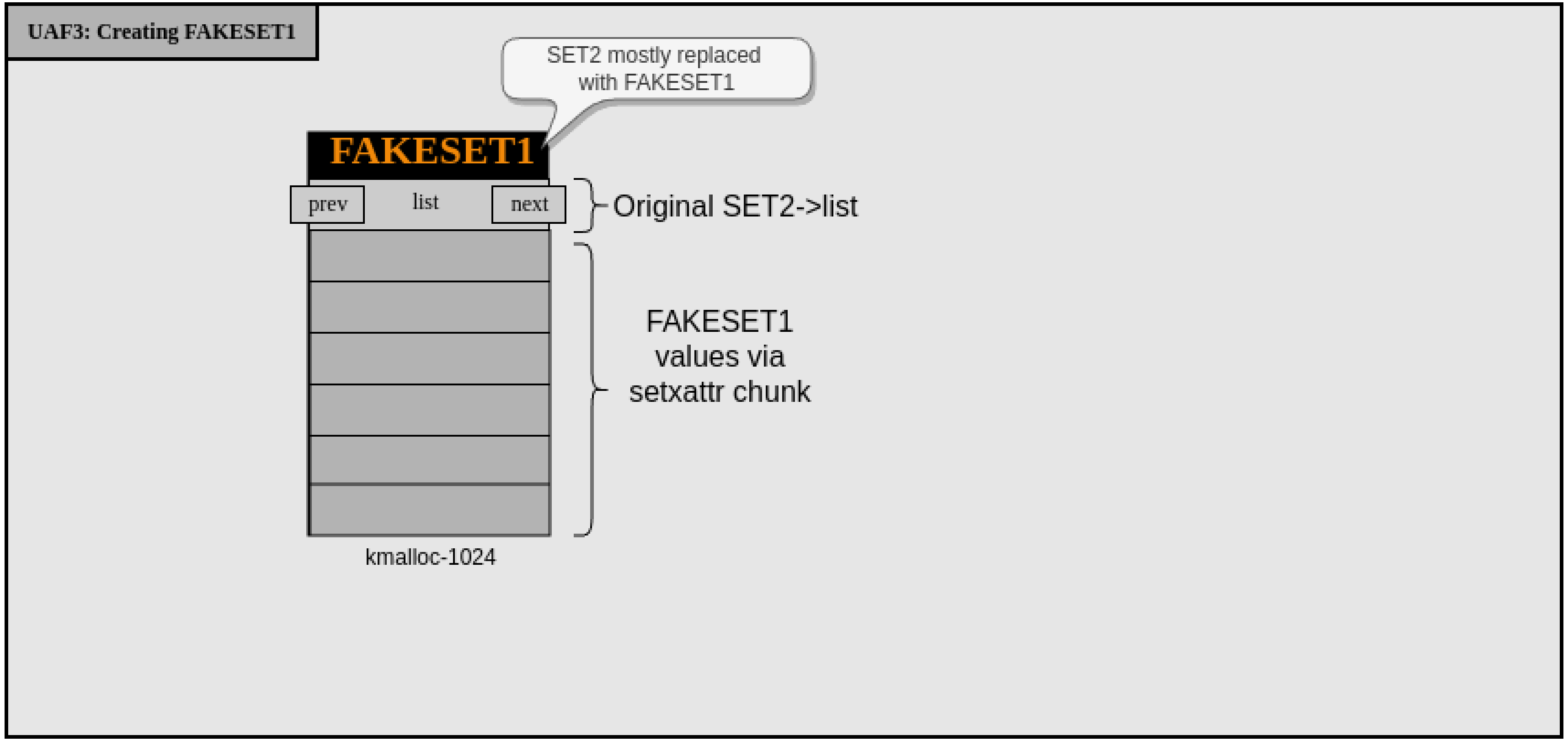
- Destroying the cgroup will free SET2



UAF3: FAKESET1 to Bypass KASLR

UAF3: FAKESET1 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

HITBSecConf
2022 Singapore

- 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

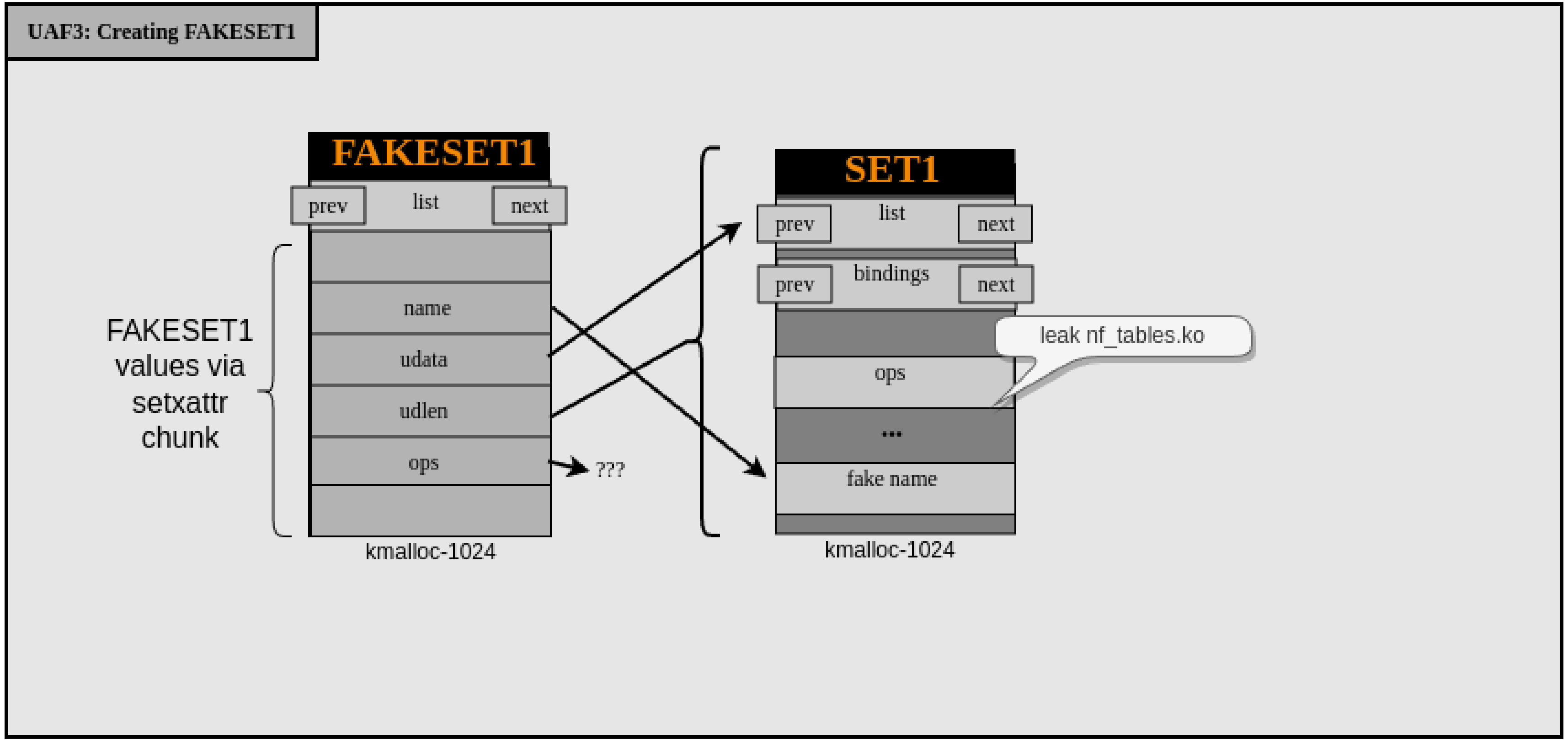
SET1 Memory Revelation

- 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: FAKESET1 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 FAKESET1)
    - 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

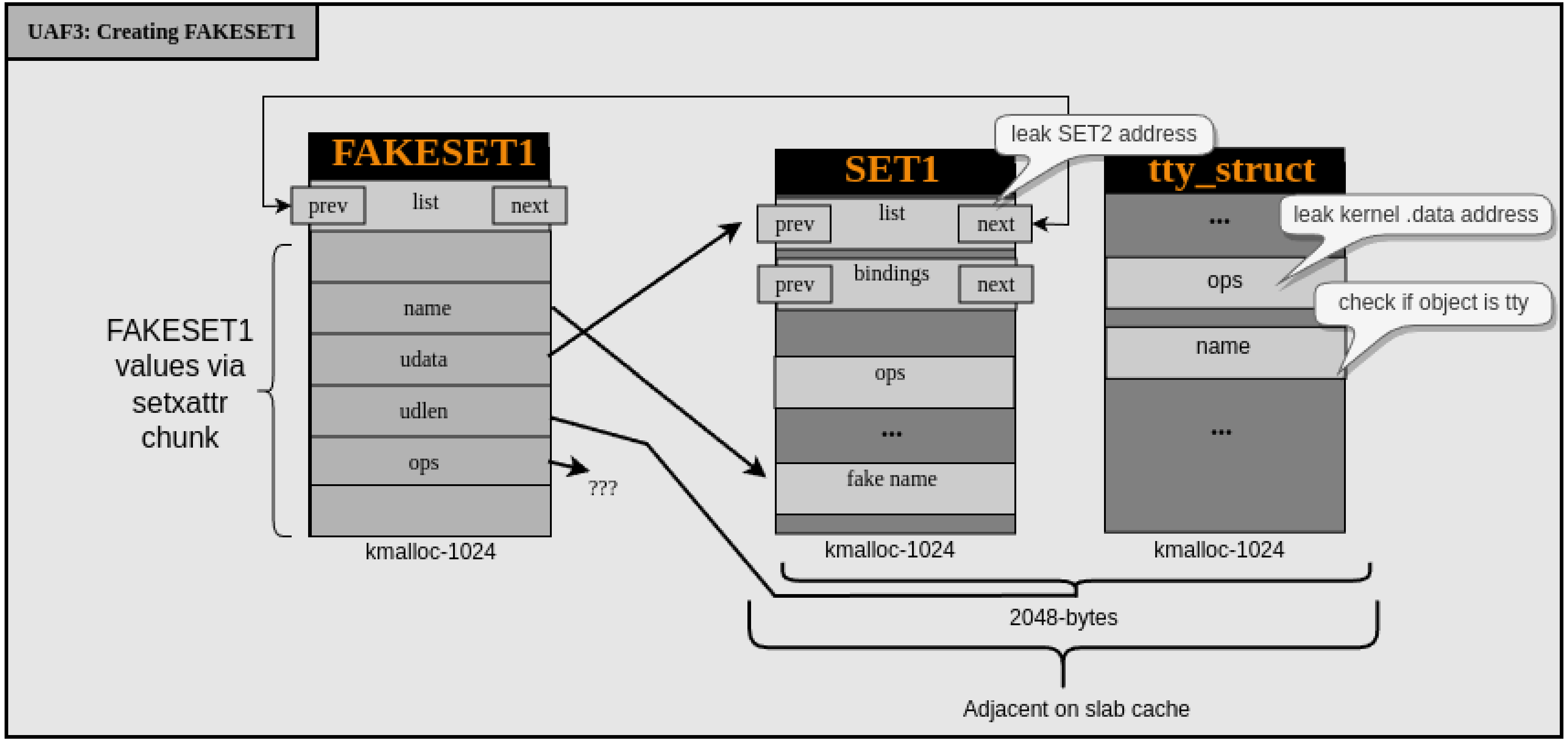
Even Better Memory Revelation

- 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

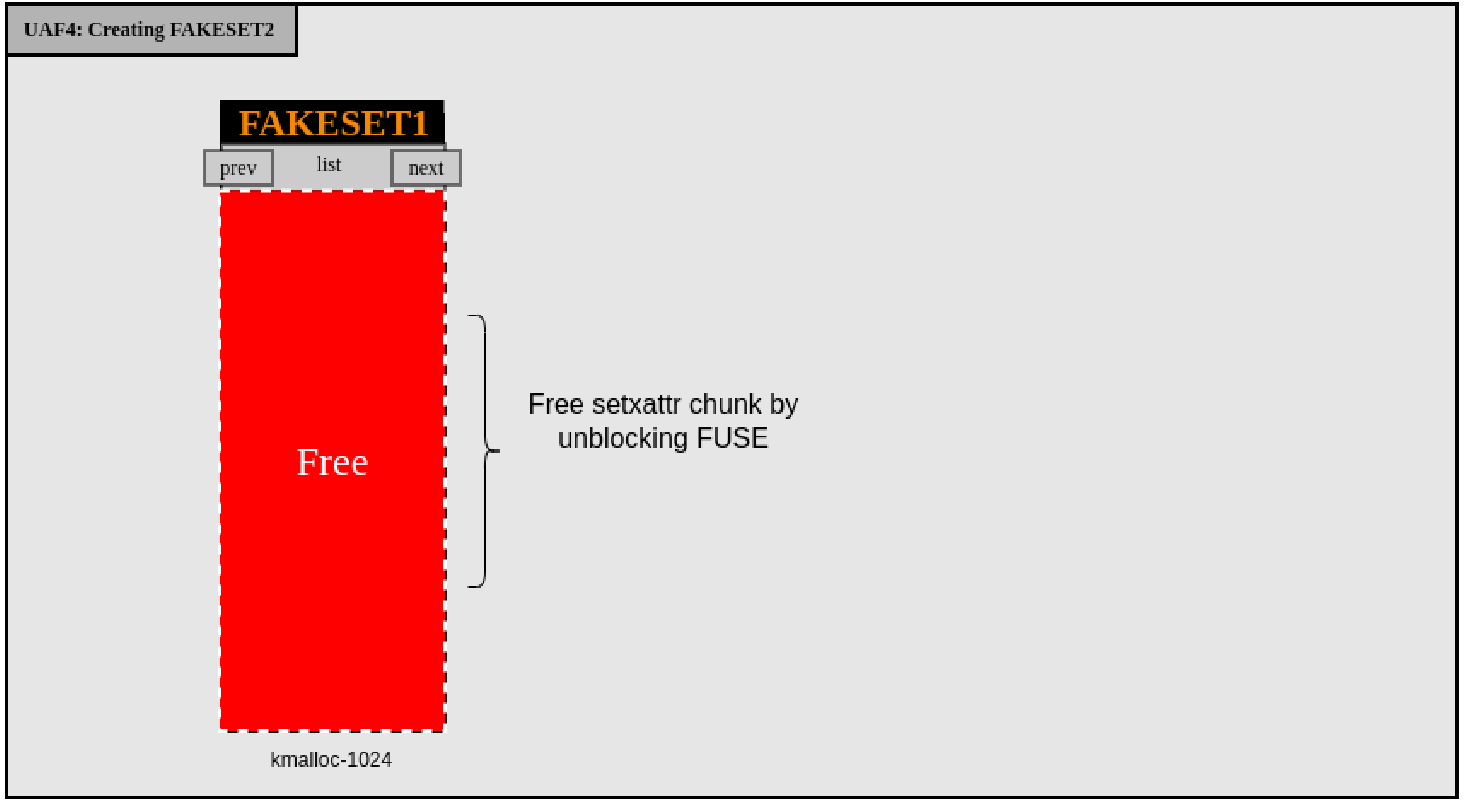
UAF4: Getting Code Execution

- 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()

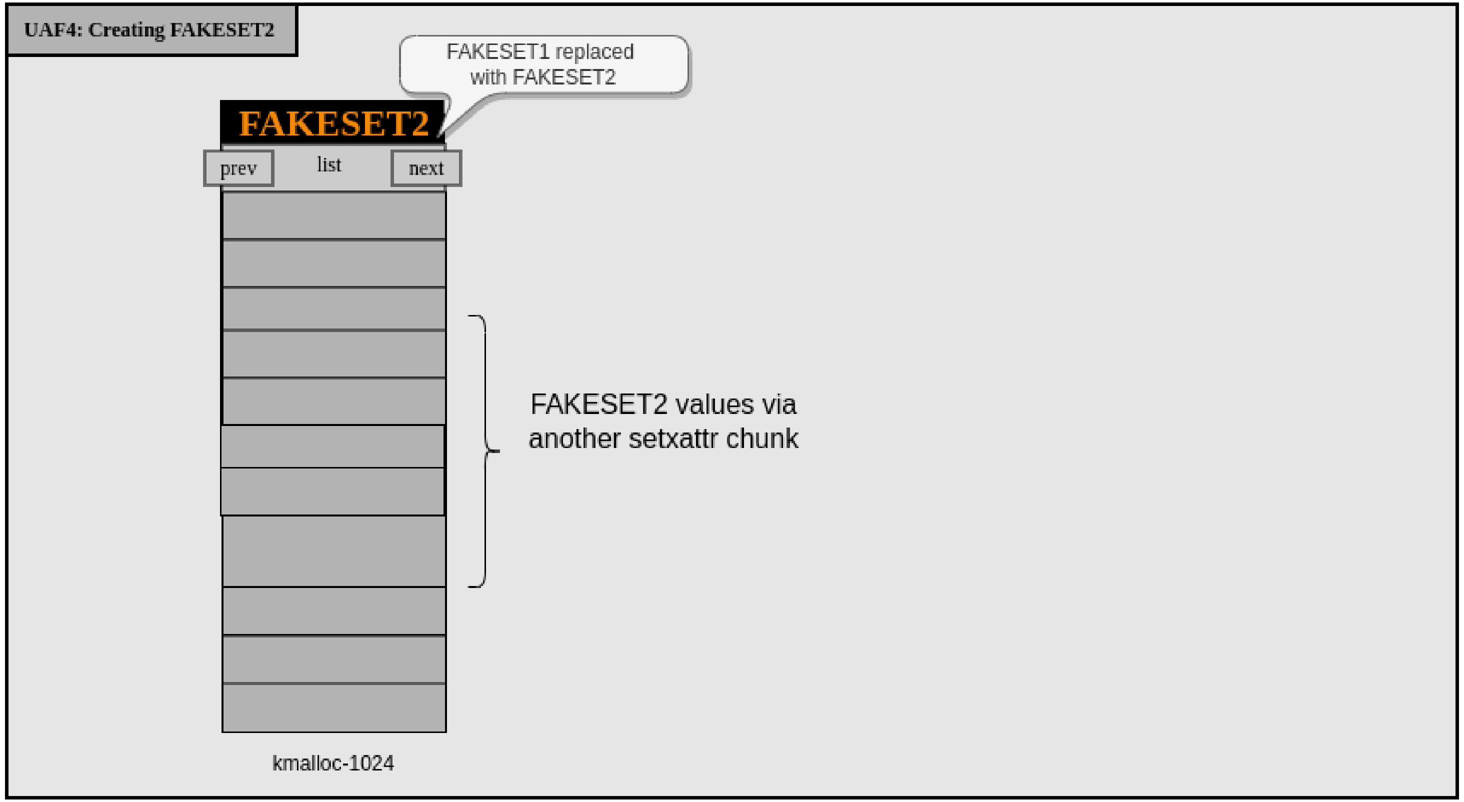
UAF4: Getting Code Execution

- 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: FAKESET1 Replacement With FAKESET2



UAF4: FAKESET1 Replacement With FAKESET2



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

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

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

```
panic_on_oops:

Controls the kernel's behaviour when an oops or BUG is encountered.

0: try to continue operation

1: panic immediately.  If the `panic' sysctl is also non-zero then the
   machine will be rebooted.
```

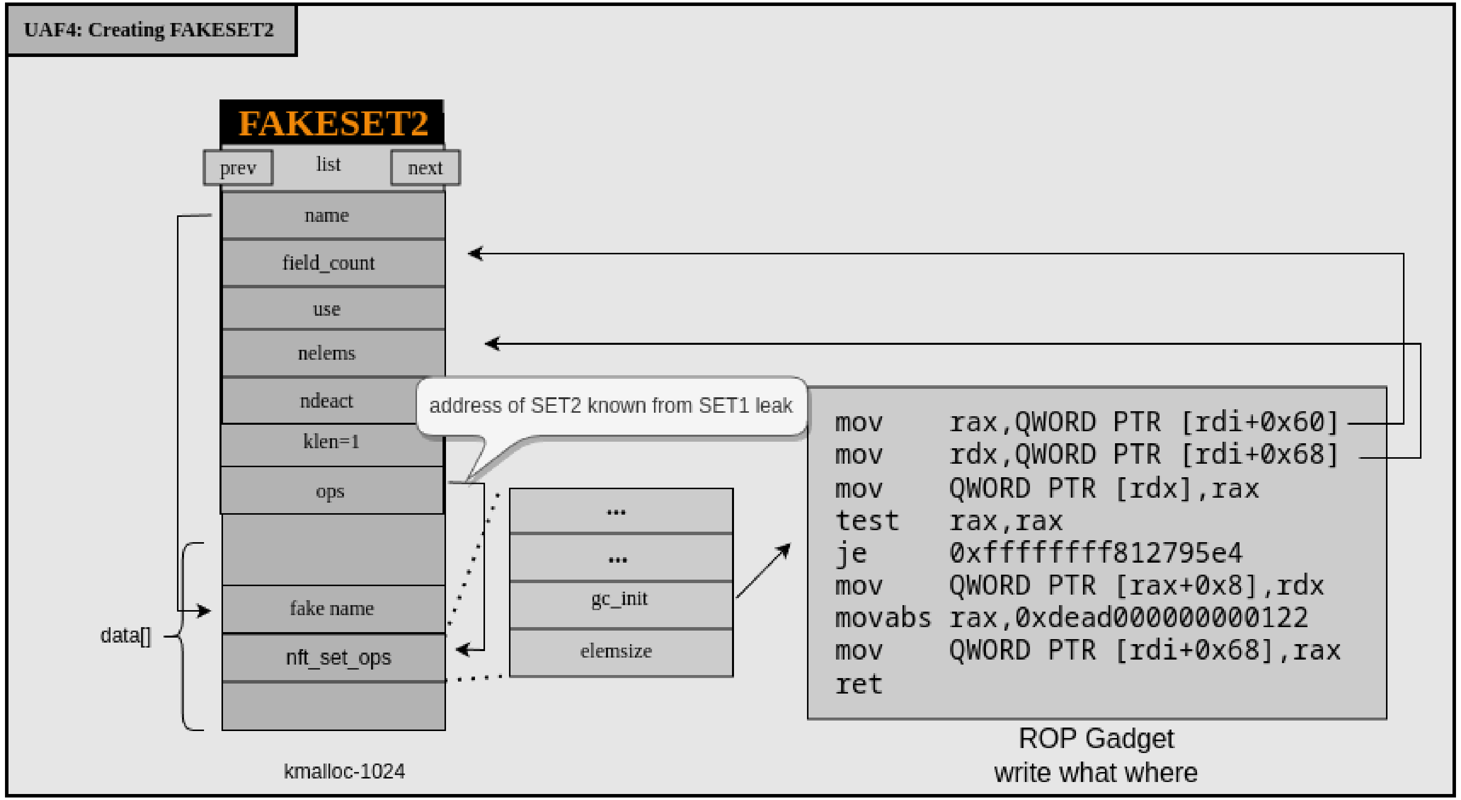
Invoking Gadget

- 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

```c
static struct nft_expr *nft_expr_init(const struct nft_ctx *ctx,
                                      const struct nlattr *nla)
{
    struct nft_expr_info expr_info;
    struct nft_expr *expr;
    struct module *owner;
    int err;

    err = nf_tables_expr_parse(ctx, nla, &expr_info);
    if (err < 0)
        goto err_expr_parse;

    err = -EOPNOTSUPP;
    if (!(expr_info.ops->type->flags & NFT_EXPR_STATEFUL))
        goto err_expr_stateful;

    err = -ENOMEM;
    expr = kzalloc(expr_info.ops->size, GFP_KERNEL_ACCOUNT);
    [...]
}
```

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 @lunixbochs's voice coding framework Talon
- Take care of your hands/body everyone!