The Power of Variant Analysis in Software Vulnerability Discovery

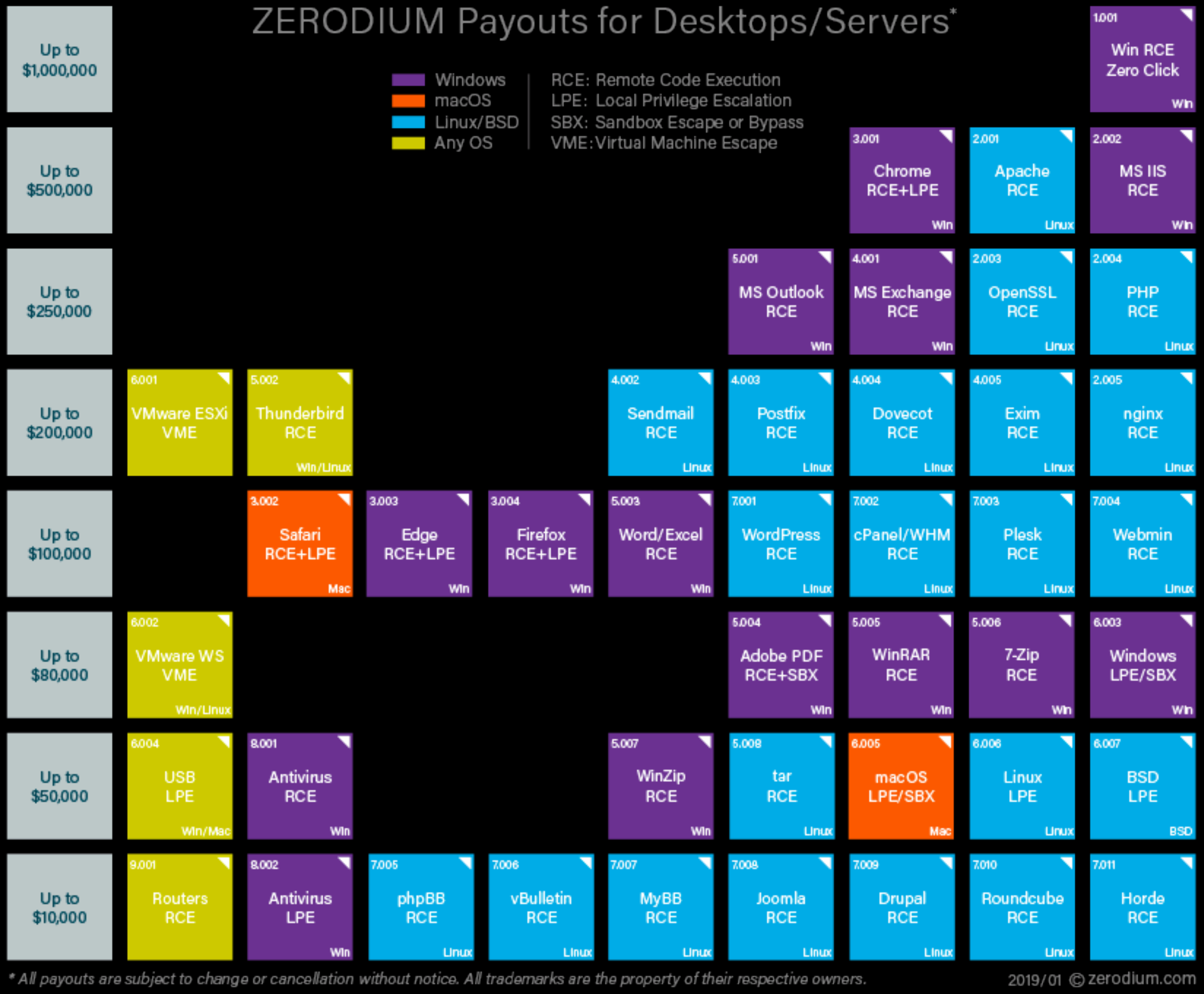
Tielei Wang

Software vulnerability & exploit

- Vulnerability: a flaw or weakness in a system's design, implementation, or operation and management that could be exploited to violate the system's security policy

- Exploit: a piece of software, a chunk of data, or a sequence of commands that takes advantage of a vulnerability to cause unintended or unanticipated behavior within the target systems

A glance of vulnerability&exploit market



ZERODIUM Payouts for Desktops/Servers*

🟪 Windows	RCE: Remote Code Execution
🟧 macOS	LPE: Local Privilege Escalation
🟦 Linux/BSD	SBX: Sandbox Escape or Bypass
🟨 Any OS	VME: Virtual Machine Escape

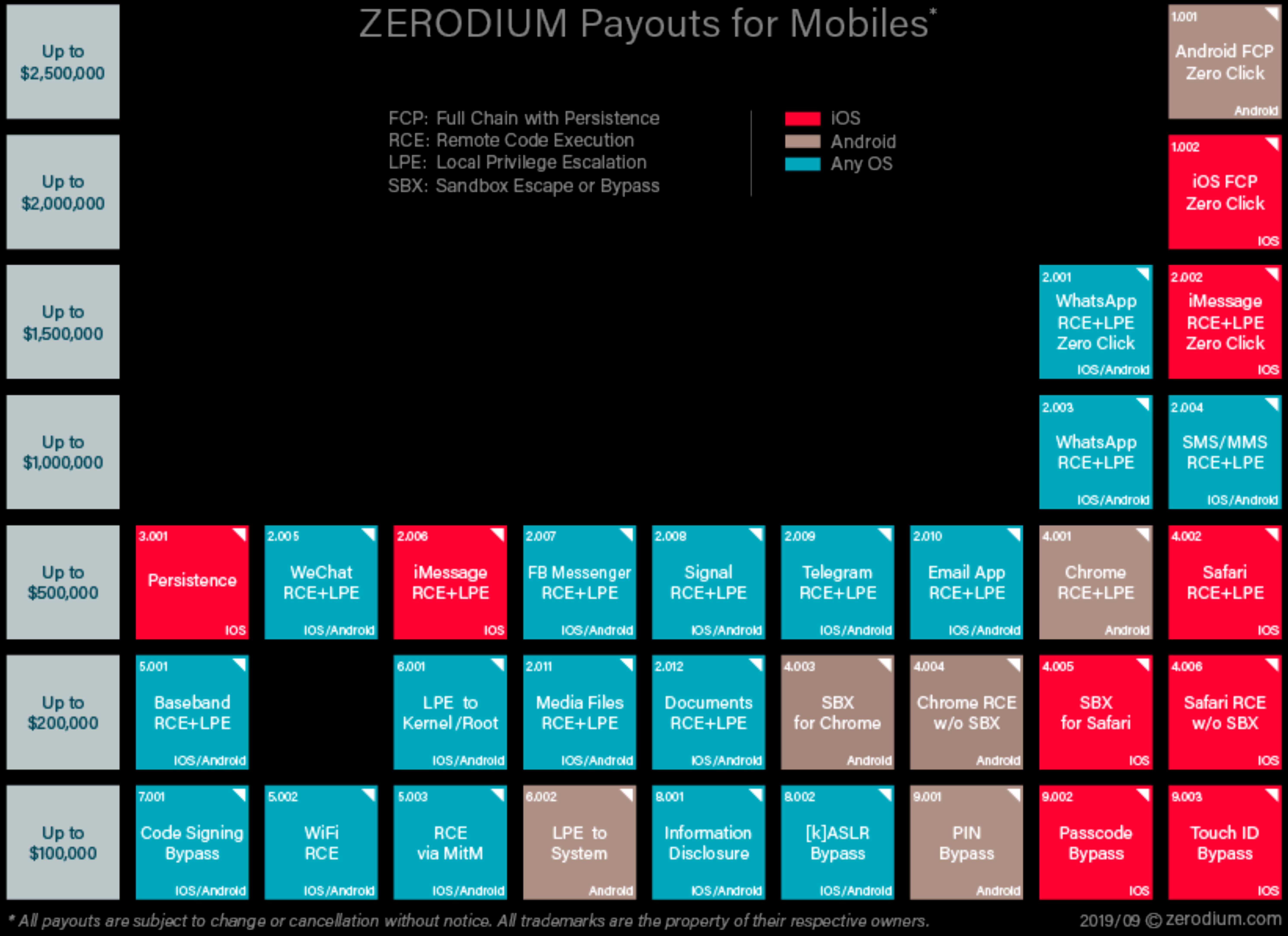
Up to $1,000,000									🟪 1.001 Win RCE Zero Click Win
Up to $500,000							🟪 3.001 Chrome RCE+LPE Win	🟦 2.001 Apache RCE Linux	🟪 2.002 MS IIS RCE Win
Up to $250,000						🟪 5.001 MS Outlook RCE Win	🟪 4.001 MS Exchange RCE Win	🟦 2.003 OpenSSL RCE Linux	🟦 2.004 PHP RCE Linux
Up to $200,000	🟨 6.001 VMware ESXi VME	🟨 5.002 Thunderbird RCE Win/Linux			🟦 4.002 Sendmail RCE Linux	🟦 4.003 Postfix RCE Linux	🟦 4.004 Dovecot RCE Linux	🟦 4.005 Exim RCE Linux	🟦 2.005 nginx RCE Linux
Up to $100,000		🟧 3.002 Safari RCE+LPE Mac	🟪 3.003 Edge RCE+LPE Win	🟪 3.004 Firefox RCE+LPE Win	🟪 5.003 Word/Excel RCE Win	🟦 7.001 WordPress RCE Linux	🟦 7.002 cPanel/WHM RCE Linux	🟦 7.003 Plesk RCE Linux	🟦 7.004 Webmin RCE Linux
Up to $80,000	🟨 6.002 VMware WS VME Win/Linux					🟪 5.004 Adobe PDF RCE+SBX Win	🟪 5.005 WinRAR RCE Win	🟪 5.006 7-Zip RCE Win	🟪 6.003 Windows LPE/SBX Win
Up to $50,000	🟨 6.004 USB LPE Win/Mac	🟪 8.001 Antivirus RCE Win			🟪 5.007 WinZip RCE Win	🟦 5.008 tar RCE Linux	🟧 6.005 macOS LPE/SBX Mac	🟦 6.006 Linux LPE Linux	🟦 6.007 BSD LPE BSD
Up to $10,000	🟨 9.001 Routers RCE	🟪 8.002 Antivirus LPE Win	🟦 7.005 phpBB RCE Linux	🟦 7.006 vBulletin RCE Linux	🟦 7.007 MyBB RCE Linux	🟦 7.008 Joomla RCE Linux	🟦 7.009 Drupal RCE Linux	🟦 7.010 Roundcube RCE Linux	🟦 7.011 Horde RCE Linux

* *All payouts are subject to change or cancellation without notice. All trademarks are the property of their respective owners.*

2019/01 © zerodium.com

Zerodium

A glance of vulnerability&exploit market



ZERODIUM Payouts for Mobiles*

FCP: Full Chain with Persistence
RCE: Remote Code Execution
LPE: Local Privilege Escalation
SBX: Sandbox Escape or Bypass

🟥 iOS
🟫 Android
🟦 Any OS

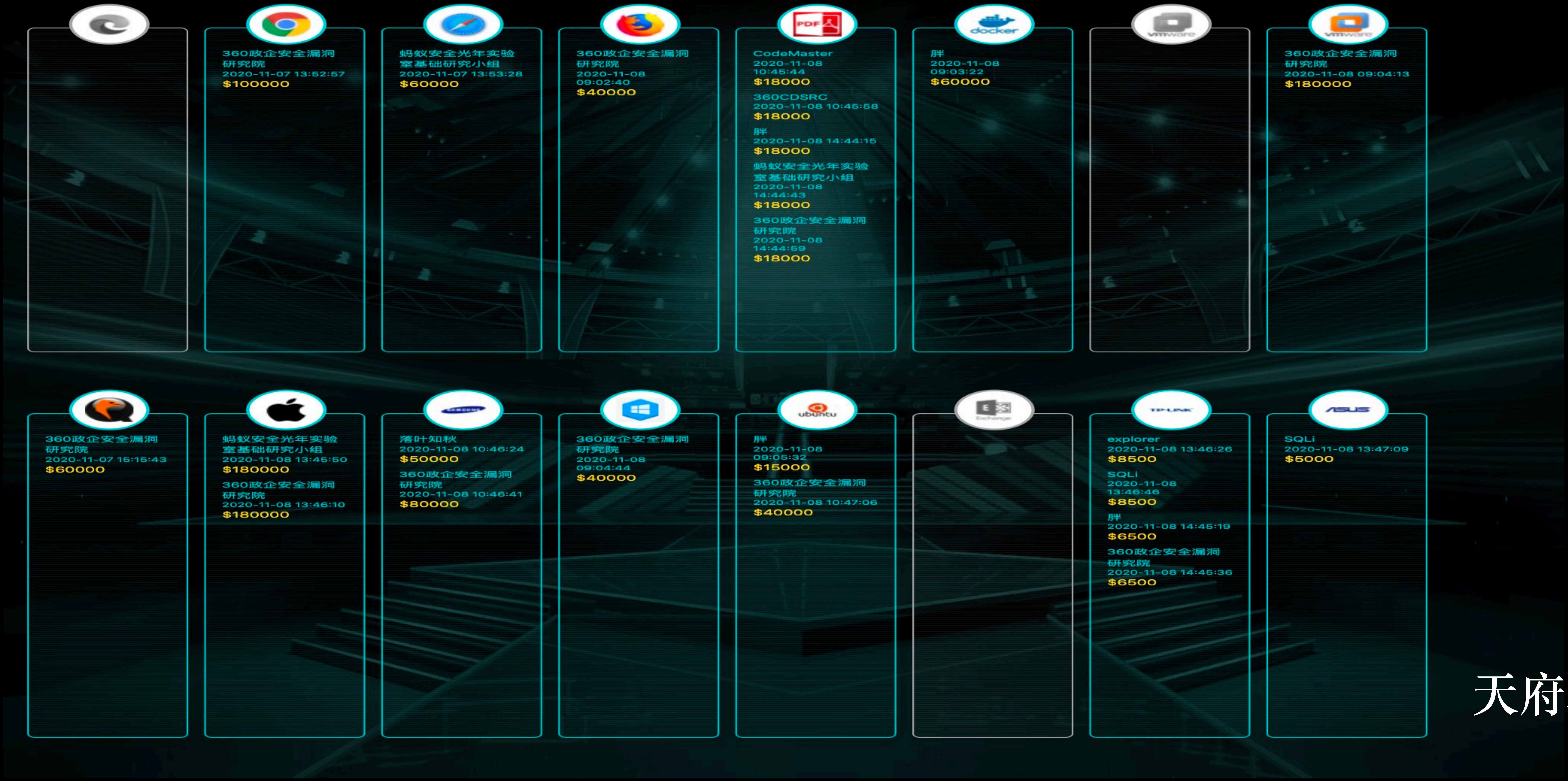
Up to $2,500,000									1.001 Android FCP Zero Click Android
Up to $2,000,000									1.002 iOS FCP Zero Click iOS
Up to $1,500,000								2.001 WhatsApp RCE+LPE Zero Click iOS/Android	2.002 iMessage RCE+LPE Zero Click iOS
Up to $1,000,000								2.003 WhatsApp RCE+LPE iOS/Android	2.004 SMS/MMS RCE+LPE iOS/Android
Up to $500,000	3.001 Persistence iOS	2.005 WeChat RCE+LPE iOS/Android	2.006 iMessage RCE+LPE iOS	2.007 FB Messenger RCE+LPE iOS/Android	2.008 Signal RCE+LPE iOS/Android	2.009 Telegram RCE+LPE iOS/Android	2.010 Email App RCE+LPE iOS/Android	4.001 Chrome RCE+LPE Android	4.002 Safari RCE+LPE iOS
Up to $200,000	5.001 Baseband RCE+LPE iOS/Android		6.001 LPE to Kernel/Root iOS/Android	2.011 Media Files RCE+LPE iOS/Android	2.012 Documents RCE+LPE iOS/Android	4.003 SBX for Chrome Android	4.004 Chrome RCE w/o SBX Android	4.005 SBX for Safari iOS	4.006 Safari RCE w/o SBX iOS
Up to $100,000	7.001 Code Signing Bypass iOS/Android	5.002 WiFi RCE iOS/Android	5.003 RCE via MitM iOS/Android	6.002 LPE to System Android	8.001 Information Disclosure iOS/Android	8.002 [k]ASLR Bypass iOS/Android	9.001 PIN Bypass Android	9.002 Passcode Bypass iOS	9.003 Touch ID Bypass iOS

* All payouts are subject to change or cancellation without notice. All trademarks are the property of their respective owners.

2019/09 © zerodium.com

Zerodium

A glance of vulnerability&exploit market



360政企安全漏洞
研究院
2020-11-07 13:52:57
$100000

蚂蚁安全光年实验
室基础研究小组
2020-11-07 13:53:28
$60000

360政企安全漏洞
研究院
2020-11-08
09:02:40
$40000

CodeMaster
2020-11-08
10:45:44
$18000

360CDSRC
2020-11-08 10:45:58
$18000

胖
2020-11-08 14:44:15
$18000

蚂蚁安全光年实验
室基础研究小组
2020-11-08
14:44:43
$18000

360政企安全漏洞
研究院
2020-11-08
14:44:59
$18000

胖
2020-11-08
09:03:22
$60000

360政企安全漏洞
研究院
2020-11-08 09:04:13
$180000

360政企安全漏洞
研究院
2020-11-07 15:15:43
$60000

蚂蚁安全光年实验
室基础研究小组
2020-11-08 13:45:50
$180000

360政企安全漏洞
研究院
2020-11-08 13:46:10
$180000

落叶知秋
2020-11-08 10:46:24
$50000

360政企安全漏洞
研究院
2020-11-08 10:46:41
$80000

360政企安全漏洞
研究院
2020-11-08
09:04:44
$40000

胖
2020-11-08
09:05:32
$15000

360政企安全漏洞
研究院
2020-11-08 10:47:06
$40000

explorer
2020-11-08 13:46:26
$8500

SQLi
2020-11-08
13:46:46
$8500

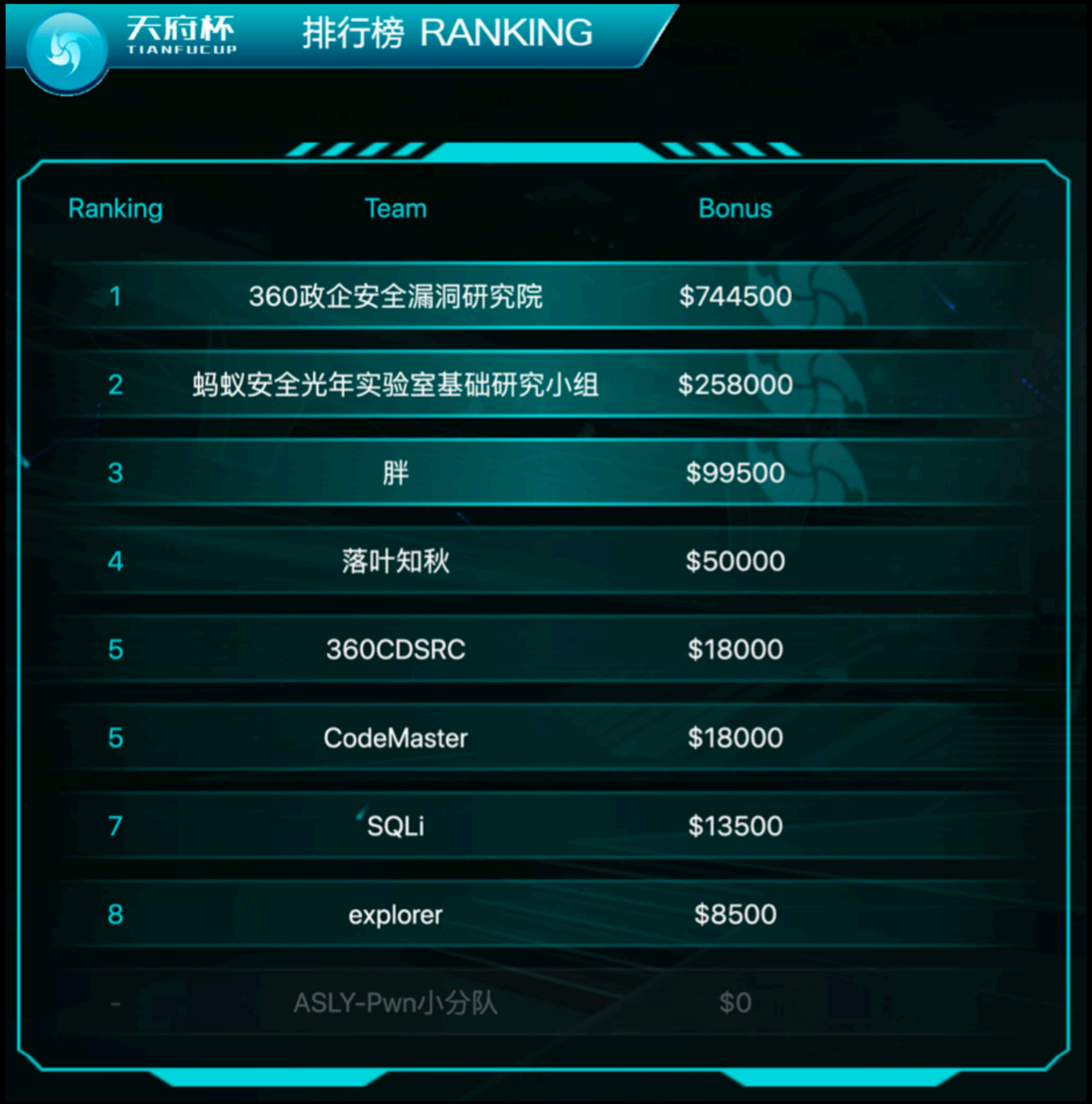
胖
2020-11-08 14:45:19
$6500

360政企安全漏洞
研究院
2020-11-08 14:45:36
$6500

SQLi
2020-11-08 13:47:09
$5000

天府杯

A glance of vulnerability&exploit market


天府杯 TIANFUCUP 排行榜 RANKING

Ranking	Team	Bonus
1	360政企安全漏洞研究院	$744500
2	蚂蚁安全光年实验室基础研究小组	$258000
3	胖	$99500
4	落叶知秋	$50000
5	360CDSRC	$18000
5	CodeMaster	$18000
7	SQLi	$13500
8	explorer	$8500
-	ASLY-Pwn小分队	$0

天府杯

A glance of vulnerability&exploit market

		Maximum Payout
Unauthorized access to iCloud account data on Apple servers		$100,000
Attack via physical access	Lock screen bypass	$100,000
	User data extraction	$250,000
Attack via user-installed app	Unauthorized access to high-value user data	$100,000
	Kernel code execution	$150,000
	CPU side channel attack on high-value user data	$250,000
Network attack requiring user interaction	One-click unauthorized access to high-value user data	$150,000
	One-click kernel code execution	$250,000
Network attack with no user interaction	Zero-click radio to kernel with physical proximity	$250,000
	Zero-click access to high-value user data	$500,000

Apple Bug Bounty Program

So how to find vulnerabilities?

- Static Analysis
- Dynamic Analysis
- Fuzzing
- Manuel Auditing Source Code or Reverse engineering
- …

Our focus today

- ~~Static Analysis~~
- ~~Dynamic Analysis~~
- ~~Fuzzing~~
- ~~Manuel Auditing Source Code or Reverse engineering~~
- Variant analysis

Variant analysis

- Refers to the process of studying a known security bug and then looking for code which is vulnerable in a similar way
- A concept that was widely accepted by industry researchers
- Sounds easy?

Variant analysis

- Requires:
  - Deep understanding to the known vulnerabilities
  - Deep understanding to the target systems
  - Open and curious mind

Outline

- ~~Introduction~~
- UNIX Socket Bind Race Vulnerability in XNU
- How to Apply Variant Analysis
- Conclusion

Background

- XNU is the OS kernel developed by Apple and used in iOS and macOS products
- A UNIX socket is an inter-process communication mechanism that allows bidirectional data exchange between processes running on the same machine.
- We already discussed this vulnerability at Blackhat USA 2019.

Take a deep breath

A lot of C code is coming

```c
int sock;
struct sockaddr_un name;
char buf[1024];
/* Create socket from which to read. */
sock = socket(AF_UNIX, SOCK_DGRAM, 0);

/* Create name. */
name.sun_family = AF_UNIX;
strcpy(name.sun_path, "1.txt");
name.sun_len = strlen(name.sun_path);

/* Bind socket to the path. */
bind(sock, (struct sockaddr *)&name,
     SUN_LEN(&name));

/* Read from the socket. */
read(sock, buf, 1024);

close(sock);
```

A simple server

```c
int sock;
struct sockaddr_un name;
char buf[1024];
/* Create socket from which to write. */
sock = socket(AF_UNIX, SOCK_DGRAM, 0);

/* Create name. */
name.sun_family = AF_UNIX;
strcpy(name.sun_path, "1.txt");
name.sun_len = strlen(name.sun_path);

/* Connect the socket to the path. */
connect(sock, (struct sockaddr *)&name,
        SUN_LEN(&name));

/* Write to the socket. */
write(sock, buf, 1024);

close(sock);
```

A simple client

```
int sock;
struct sockaddr_un name;
char buf[1024];
```

A simple server

From the kernel point of view

please refer to xnu source code for more details

```c
int sock;
struct sockaddr_un name;
char buf[1024];
/* Create socket from which to read. */
sock = socket(AF_UNIX, SOCK_DGRAM, 0);
```

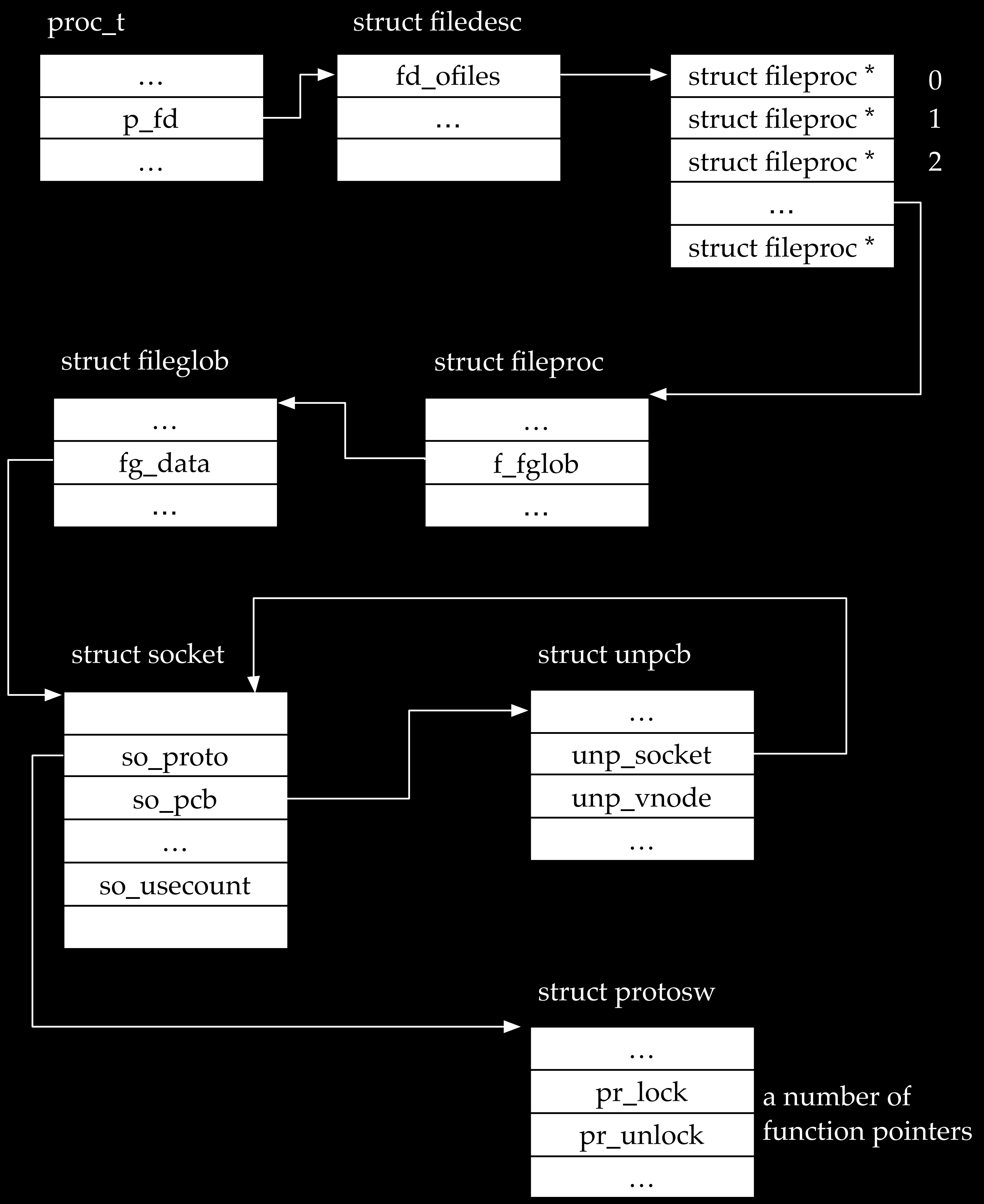
- socket
  - socket_common
    - socreate_internal
      - soalloc
      - unp_attach

A simple server

From the kernel point of view

```c
int sock;
struct sockaddr_un name;
char buf[1024];
/* Create socket from which to read. */
sock = socket(AF_UNIX, SOCK_DGRAM, 0);
```

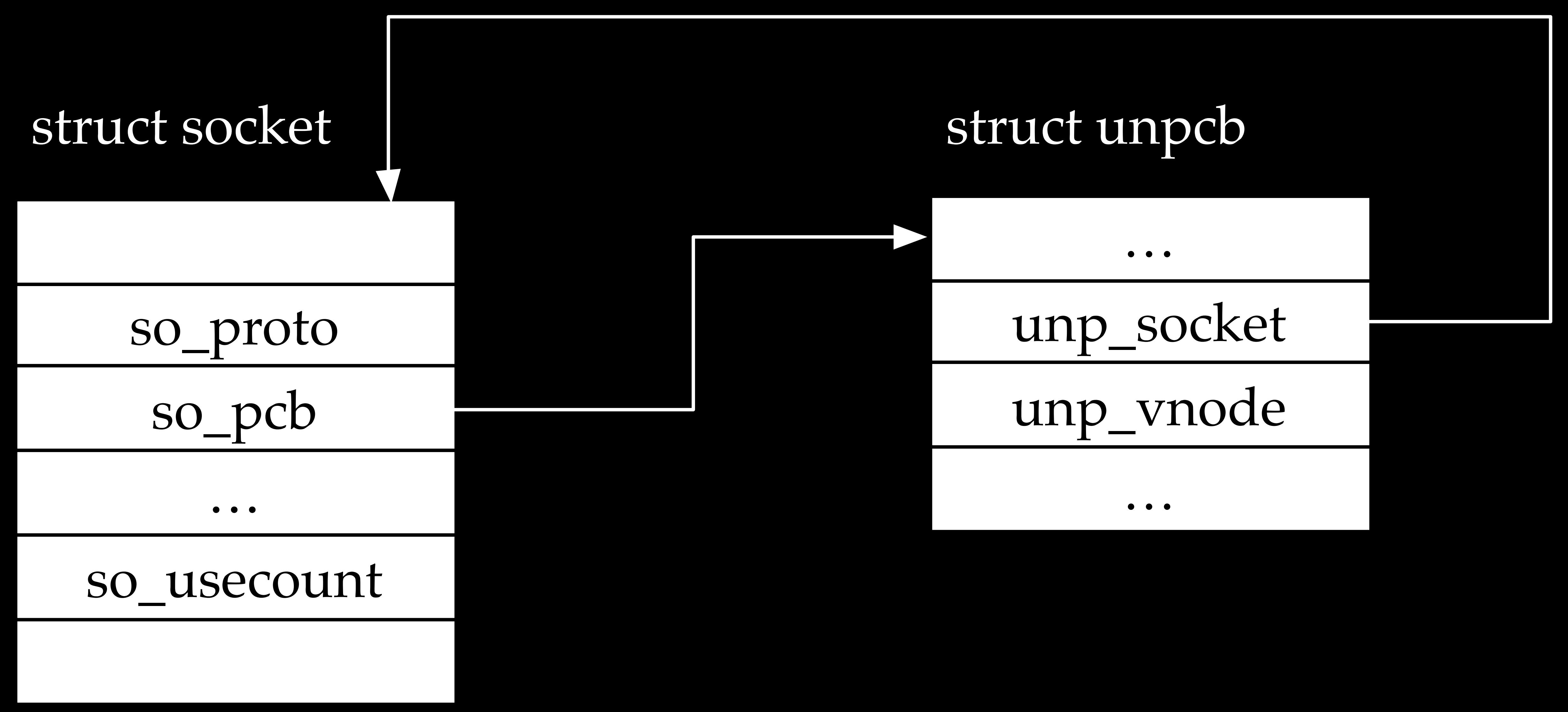
A simple server



From the kernel point of view

```
int sock;
struct sockaddr_un name;
char buf[1024];
/* Create socket from which to read. */
sock = socket(AF_UNIX, SOCK_DGRAM, 0);
```

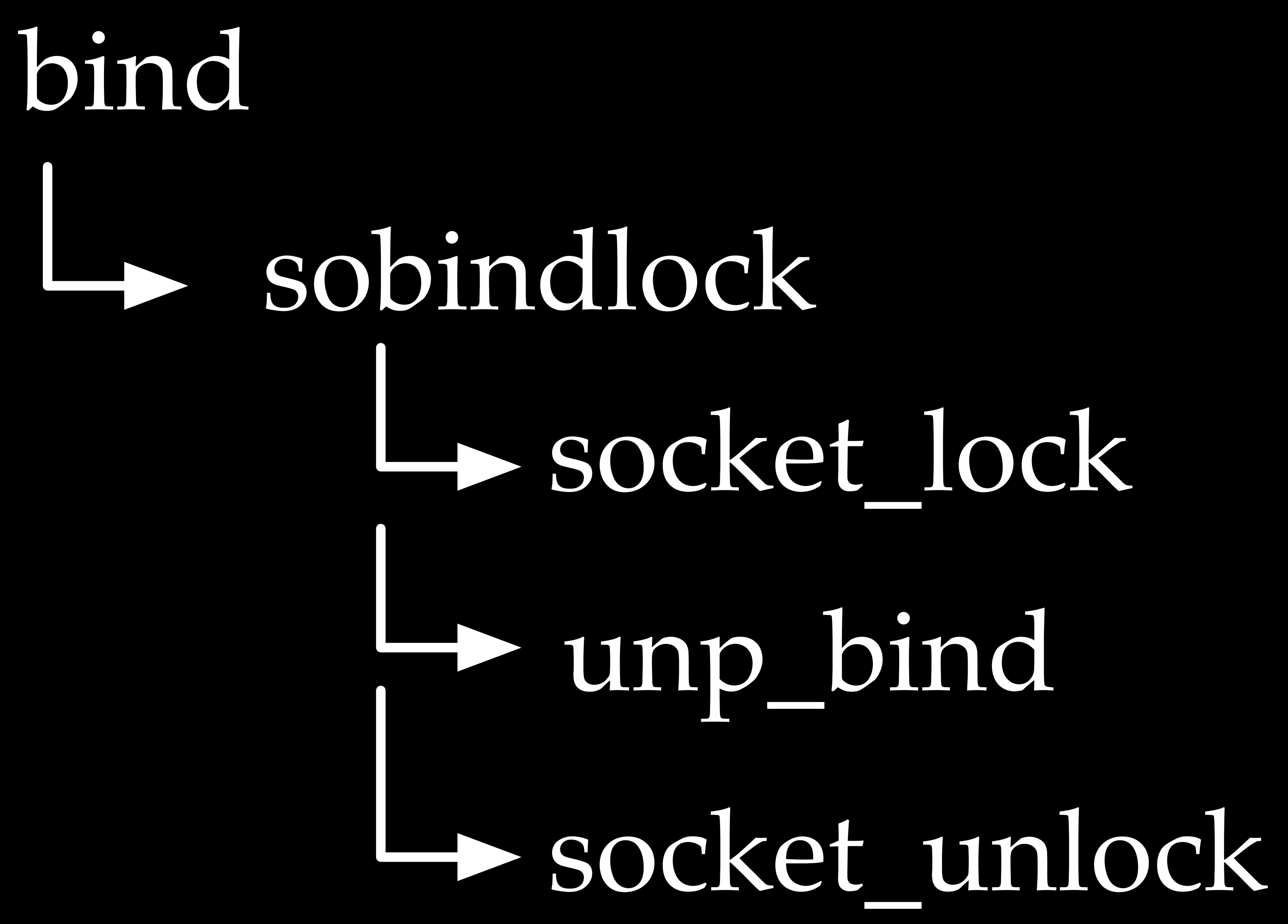
A simple server



From the kernel point of view

```c
int sock;
struct sockaddr_un name;
char buf[1024];
/* Create socket from which to read. */
sock = socket(AF_UNIX, SOCK_DGRAM, 0);

/* Create name. */
name.sun_family = AF_UNIX;
strcpy(name.sun_path, "1.txt");
name.sun_len = strlen(name.sun_path);

/* Bind socket to the path. */
bind(sock, (struct sockaddr *)&name,
     SUN_LEN(&name));
```

A simple server



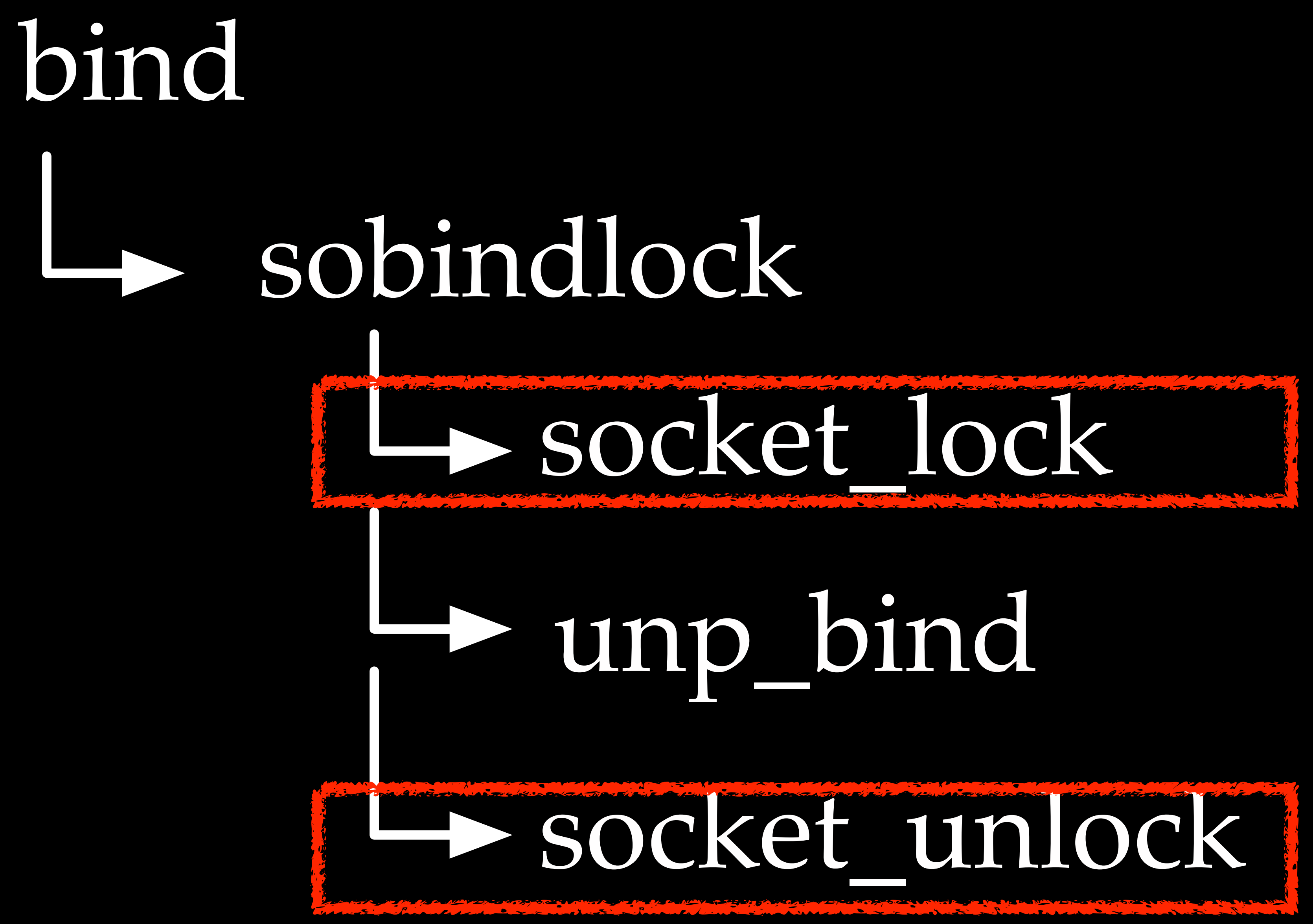
From the kernel point of view

```c
int sock;
struct sockaddr_un name;
char buf[1024];
/* Create socket from which to read. */
sock = socket(AF_UNIX, SOCK_DGRAM, 0);

/* Create name. */
name.sun_family = AF_UNIX;
strcpy(name.sun_path, "1.txt");
name.sun_len = strlen(name.sun_path);

/* Bind socket to the path. */
bind(sock, (struct sockaddr *)&name,
     SUN_LEN(&name));
```

A simple server

Note that unp_bind is surrounded by socket_(un)lock

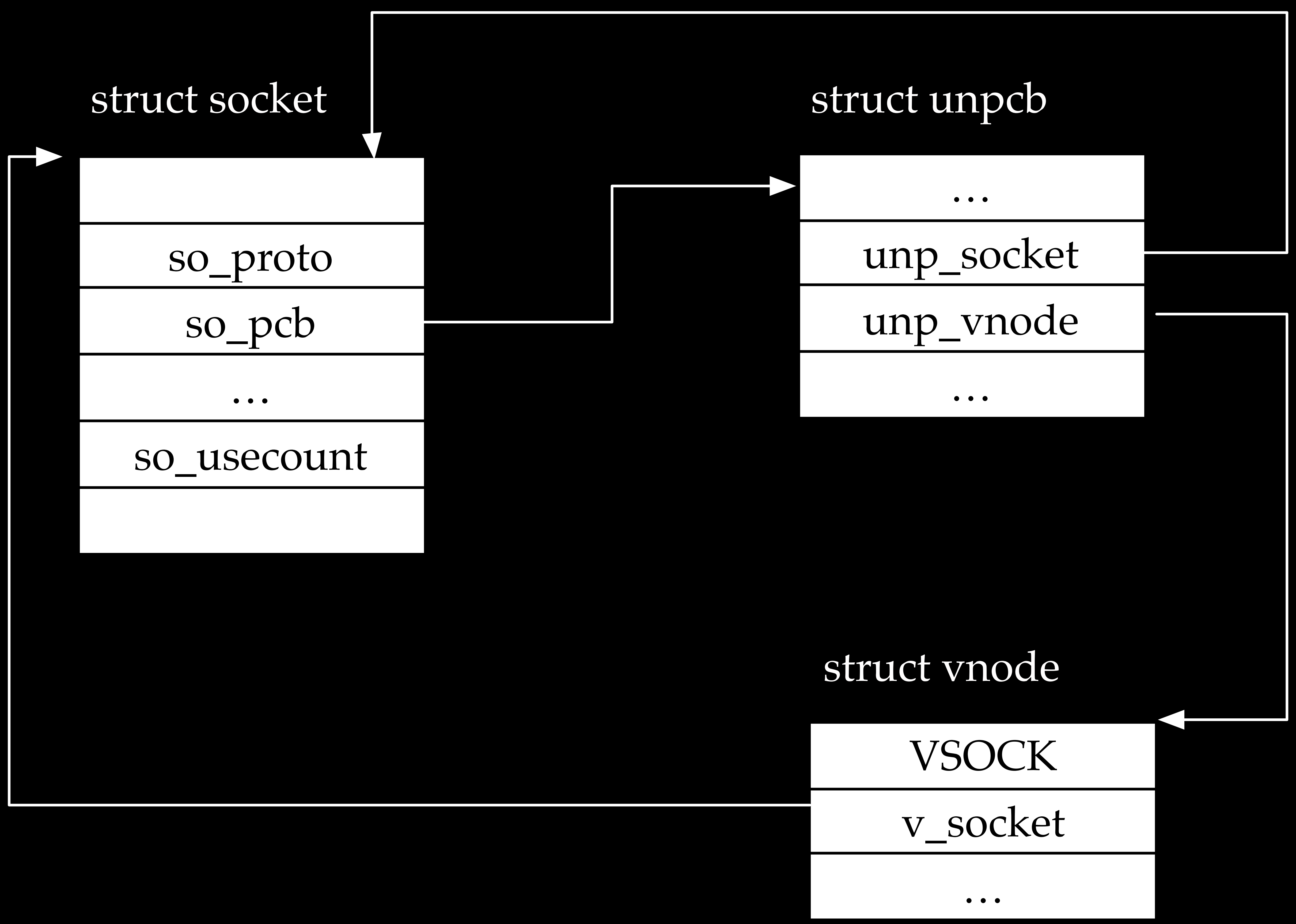
so it is unraceable?



From the kernel point of view

```
int sock;
struct sockaddr_un name;
char buf[1024];
/* Create socket from which to read. */
sock = socket(AF_UNIX, SOCK_DGRAM, 0);

/* Create name. */
name.sun_family = AF_UNIX;
strcpy(name.sun_path, "1.txt");
name.sun_len = strlen(name.sun_path);

/* Bind socket to the path. */
bind(sock, (struct sockaddr *)&name,
     SUN_LEN(&name));
```

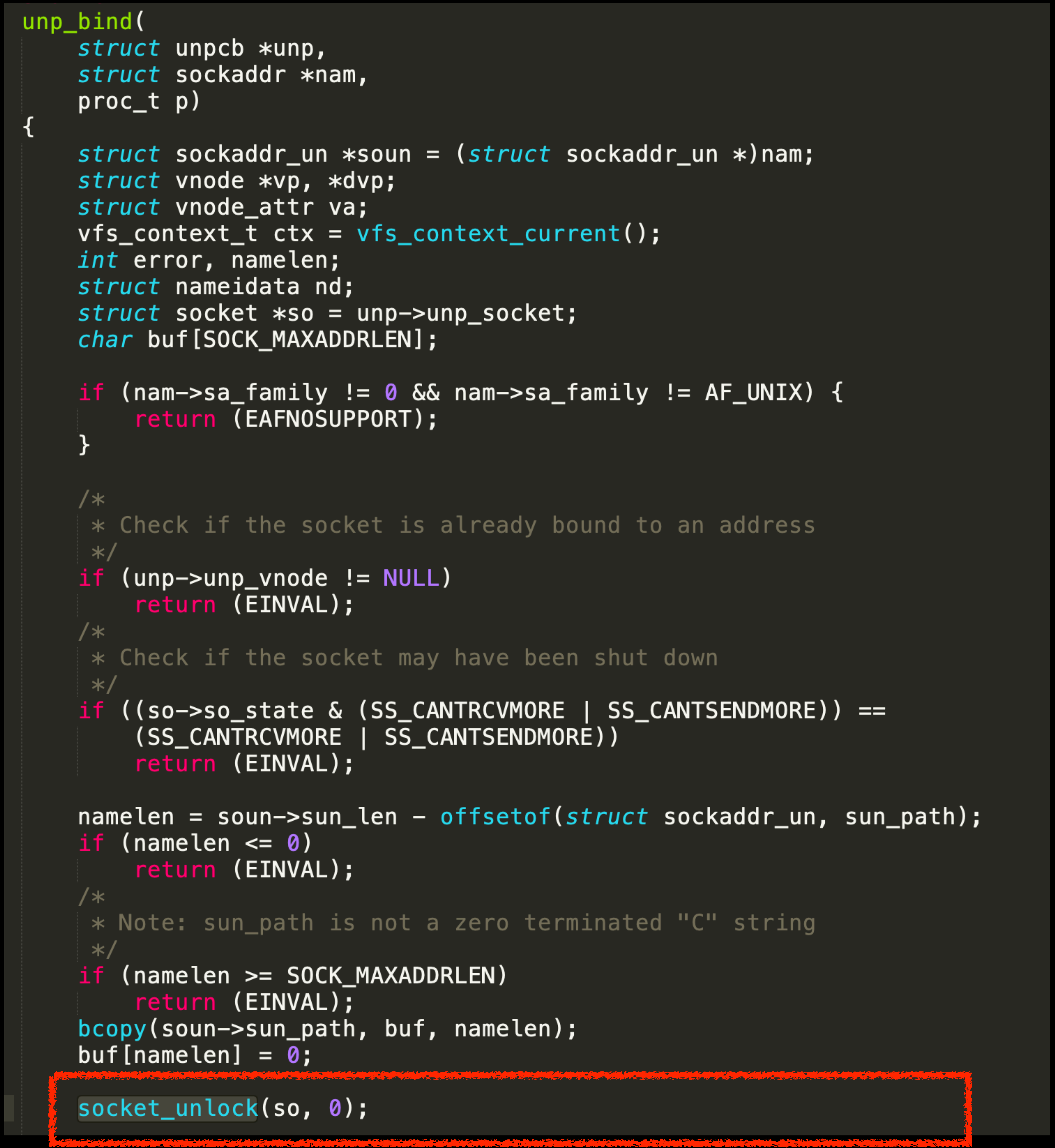
A simple server



From the kernel point of view

Race Condition

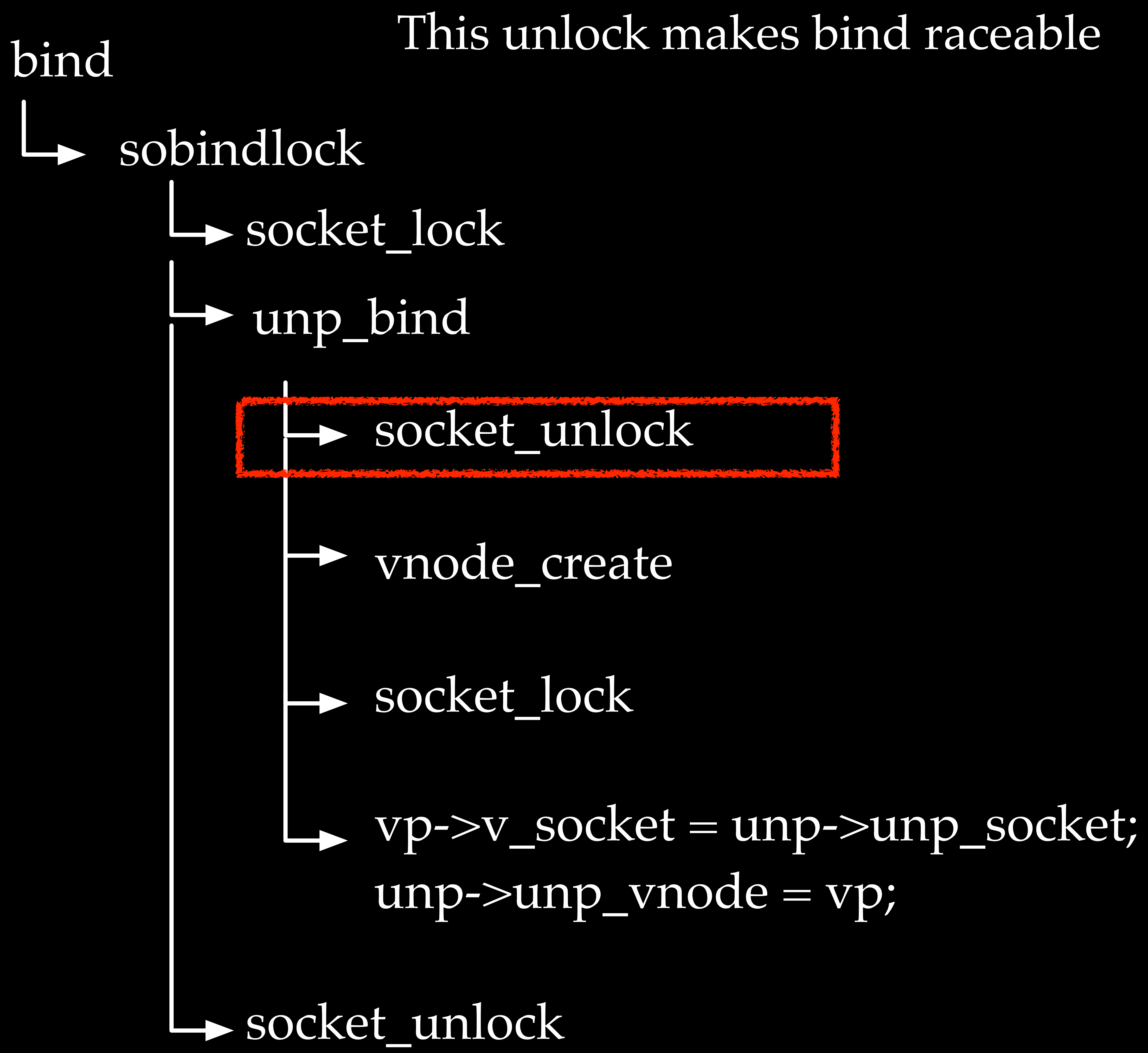
- The creation of a vnode is time consuming
- unp_bind has a temporary unlock



```c
unp_bind(
    struct unpcb *unp,
    struct sockaddr *nam,
    proc_t p)
{
    struct sockaddr_un *soun = (struct sockaddr_un *)nam;
    struct vnode *vp, *dvp;
    struct vnode_attr va;
    vfs_context_t ctx = vfs_context_current();
    int error, namelen;
    struct nameidata nd;
    struct socket *so = unp->unp_socket;
    char buf[SOCK_MAXADDRLEN];

    if (nam->sa_family != 0 && nam->sa_family != AF_UNIX) {
        return (EAFNOSUPPORT);
    }

    /*
     * Check if the socket is already bound to an address
     */
    if (unp->unp_vnode != NULL)
        return (EINVAL);
    /*
     * Check if the socket may have been shut down
     */
    if ((so->so_state & (SS_CANTRCVMORE | SS_CANTSENDMORE)) ==
        (SS_CANTRCVMORE | SS_CANTSENDMORE))
        return (EINVAL);

    namelen = soun->sun_len - offsetof(struct sockaddr_un, sun_path);
    if (namelen <= 0)
        return (EINVAL);
    /*
     * Note: sun_path is not a zero terminated "C" string
     */
    if (namelen >= SOCK_MAXADDRLEN)
        return (EINVAL);
    bcopy(soun->sun_path, buf, namelen);
    buf[namelen] = 0;

    socket_unlock(so, 0);
```

```c
int sock;
struct sockaddr_un name;
char buf[1024];
/* Create socket from which to read. */
sock = socket(AF_UNIX, SOCK_DGRAM, 0);

/* Create name. */
name.sun_family = AF_UNIX;
strcpy(name.sun_path, "1.txt");
name.sun_len = strlen(name.sun_path);

/* Bind socket to the path. */
bind(sock, (struct sockaddr *)&name,
      SUN_LEN(&name));
```

A simple server



From the kernel point of view

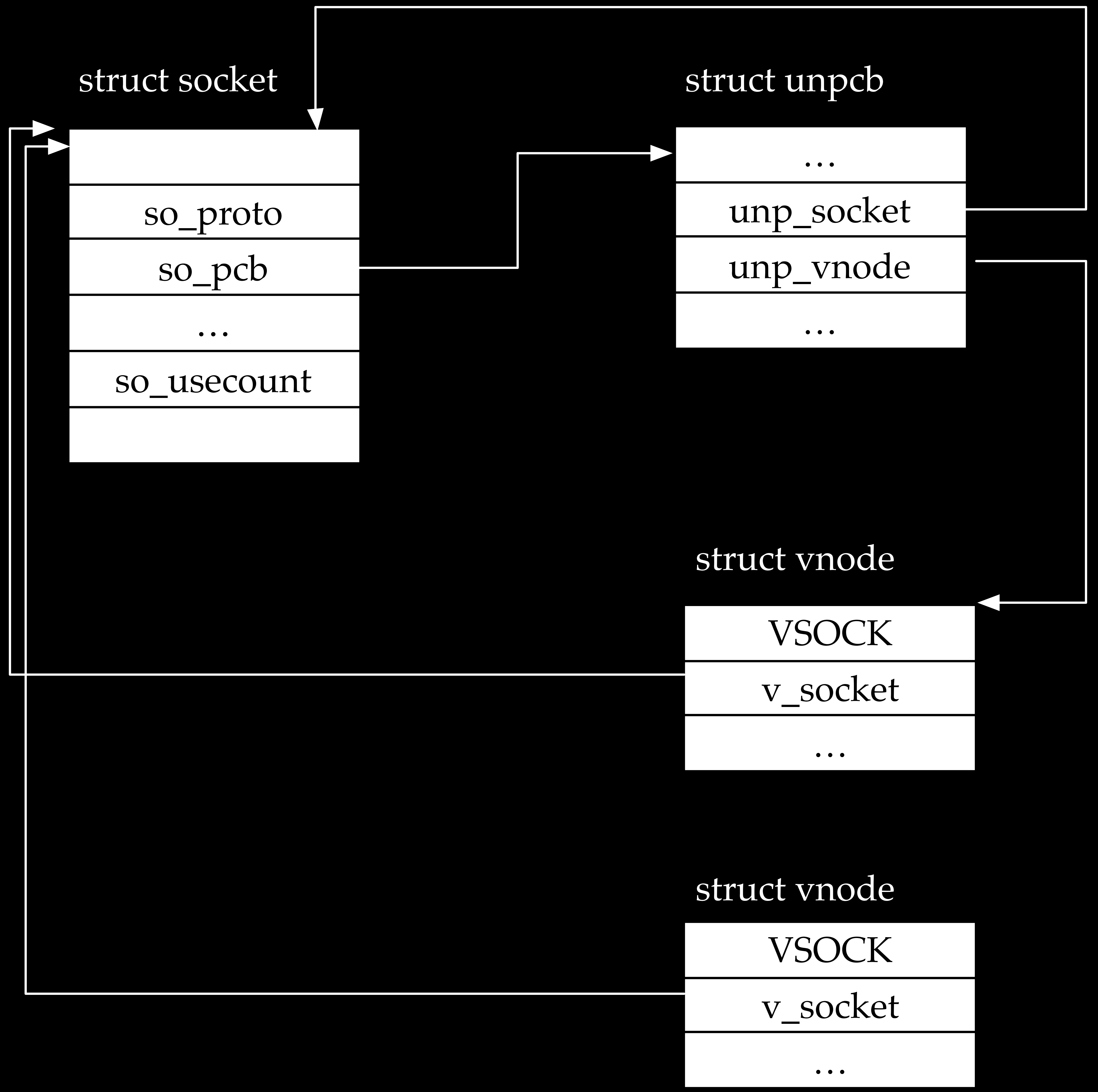
```c
int sock;
struct sockaddr_un name;
char buf[1024];
/* Create socket from which to read. */
sock = socket(AF_UNIX, SOCK_DGRAM, 0);

/* Create name. */
name.sun_family = AF_UNIX;
strcpy(name.sun_path, "1.txt");
name.sun_len = strlen(name.sun_path);

/* Bind socket to the path. */
bind(sock, (struct sockaddr *)&name,
     SUN_LEN(&name));
```

Thread 1

```c
/* Create name. */
name.sun_family = AF_UNIX;
strcpy(name.sun_path, “2.txt");
name.sun_len = strlen(name.sun_path);

/* Bind socket to the path. */
bind(sock, (struct sockaddr *)&name,
     SUN_LEN(&name));
```

Thread 2

bind the socket to two file paths in parallel



we can make a socket binding to two vnodes (two references)

bind the socket to two file paths in parallel

```c
int sock;
struct sockaddr_un name;
char buf[1024];
/* Create socket from which to read. */
sock = socket(AF_UNIX, SOCK_DGRAM, 0);

/* Create name. */
name.sun_family = AF_UNIX;
strcpy(name.sun_path, "1.txt");
name.sun_len = strlen(name.sun_path);

/* Bind socket to the path. */
bind(sock, (struct sockaddr *)&name,
     SUN_LEN(&name));

/* Read from the socket. */
read(sock, buf, 1024);
close(sock);
```

A simple server

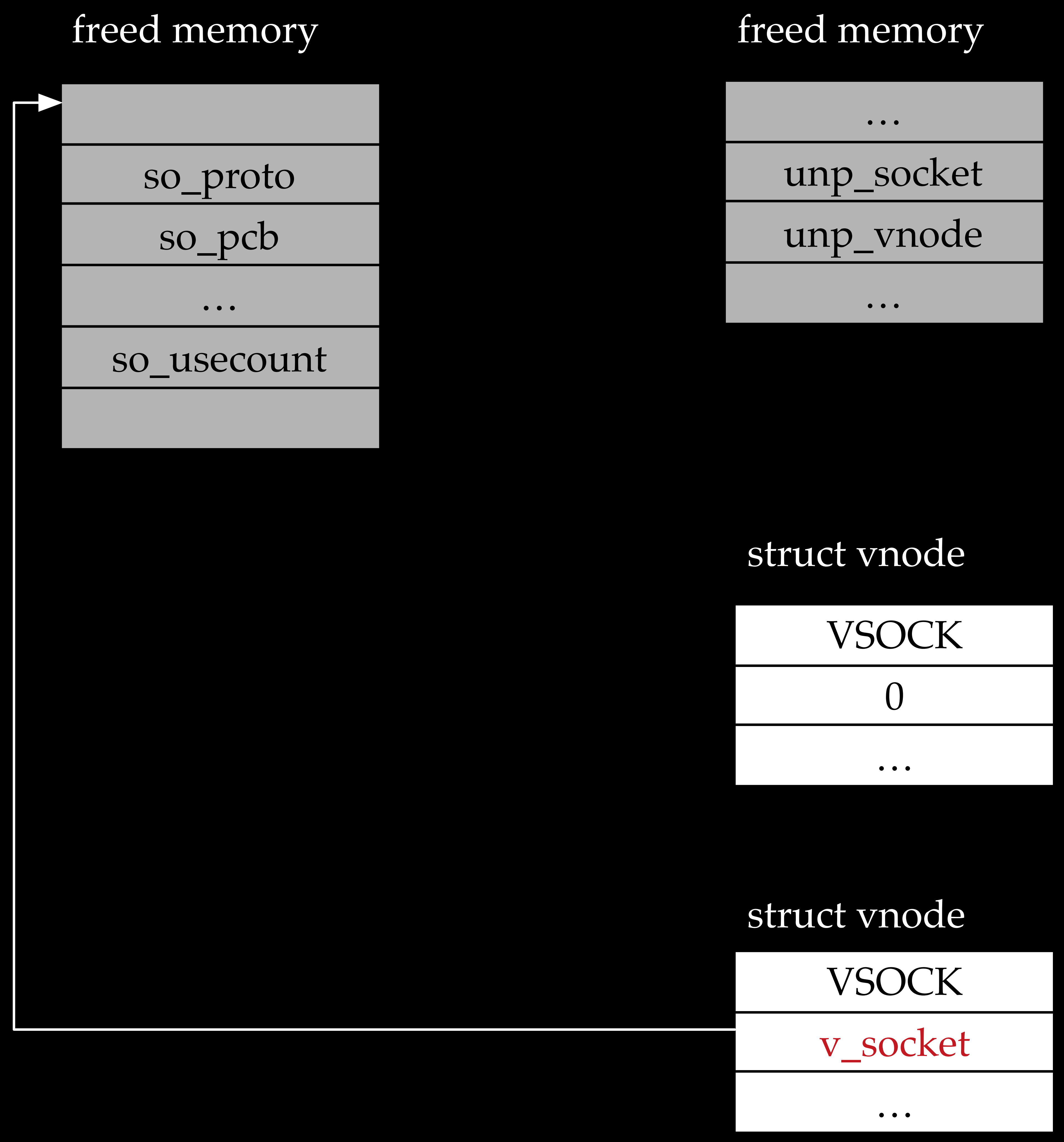
$$\begin{tikzcd}
\text{close} \arrow[dr] & & \\
& \text{soo\_close} \arrow[dr] & \\
& & \text{soclose}
\end{tikzcd}$$

From the kernel point of view

```
int sock;
struct sockaddr_un name;
char buf[1024];
/* Create socket from which to read. */
sock = socket(AF_UNIX, SOCK_DGRAM, 0);

/* Create name. */
name.sun_family = AF_UNIX;
strcpy(name.sun_path, "1.txt");
name.sun_len = strlen(name.sun_path);

/* Bind socket to the path. */
bind(sock, (struct sockaddr *)&name,
     SUN_LEN(&name));

/* Read from the socket. */
read(sock, buf, 1024);

close(sock);
```

A simple server

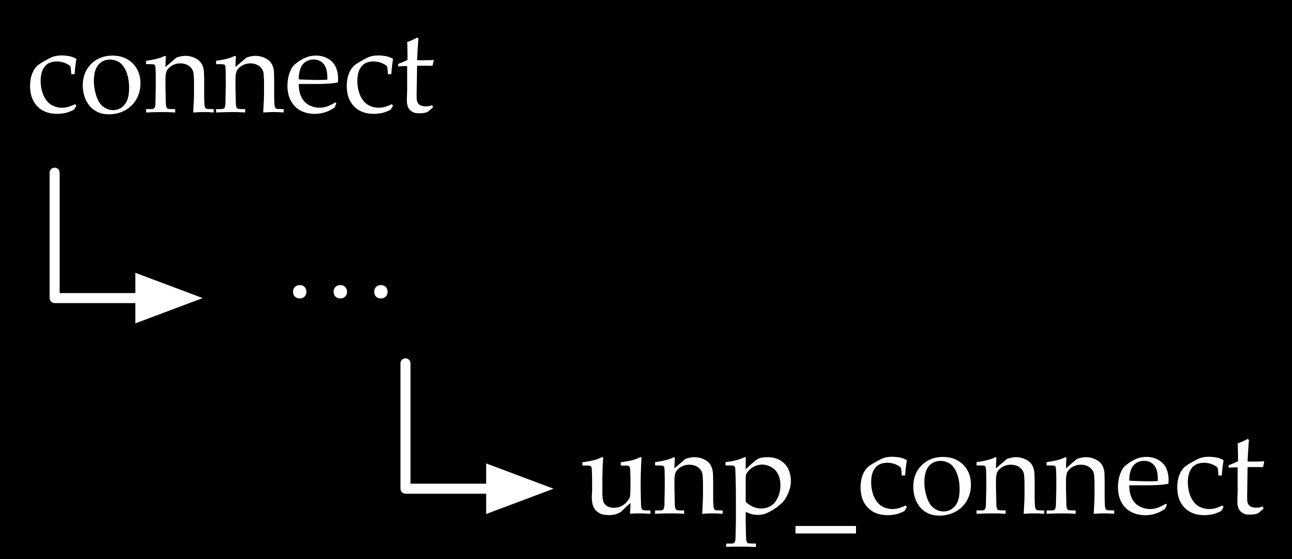
One of the vnodes will hold a dangling pointer



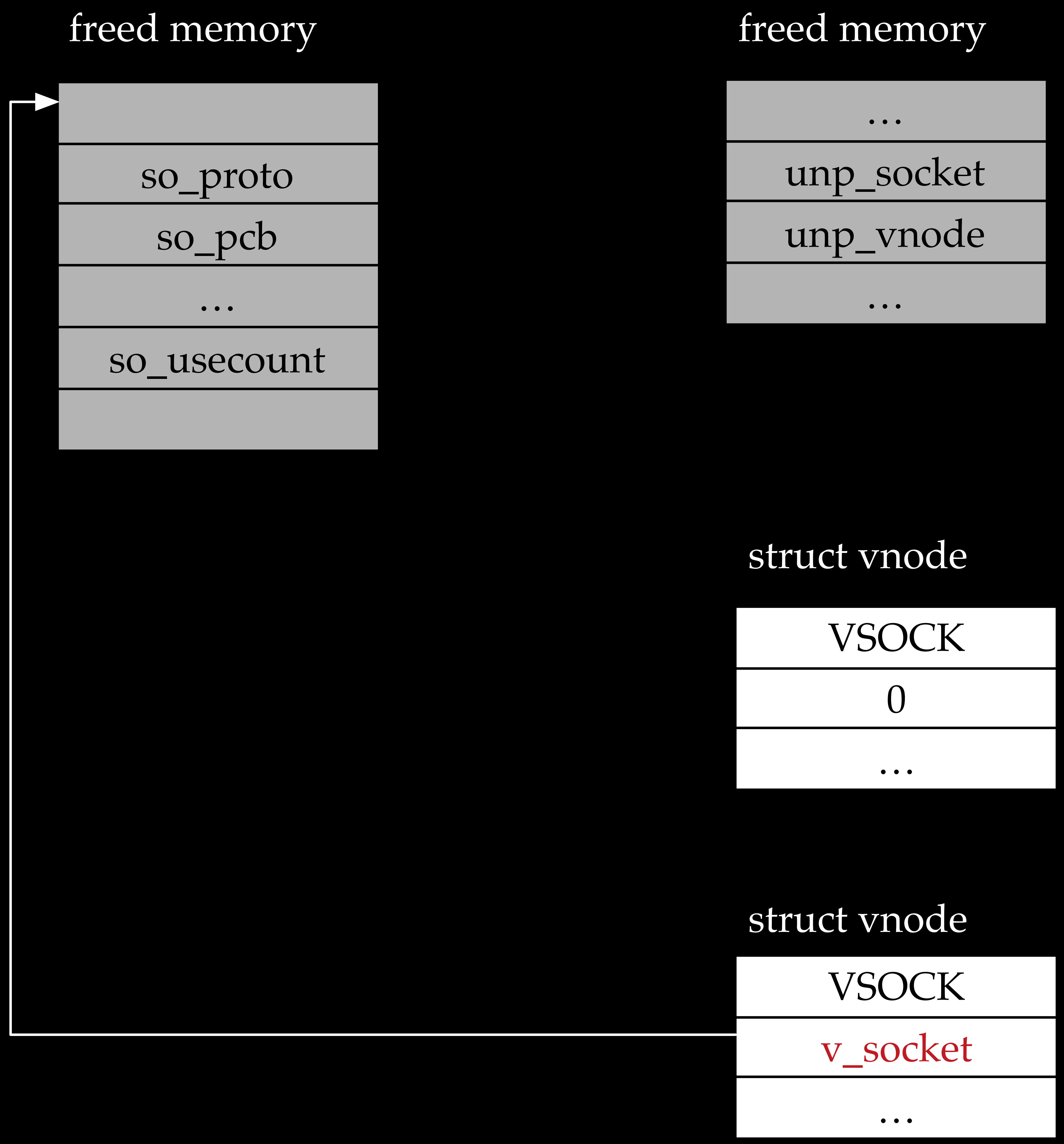
From the kernel point of view

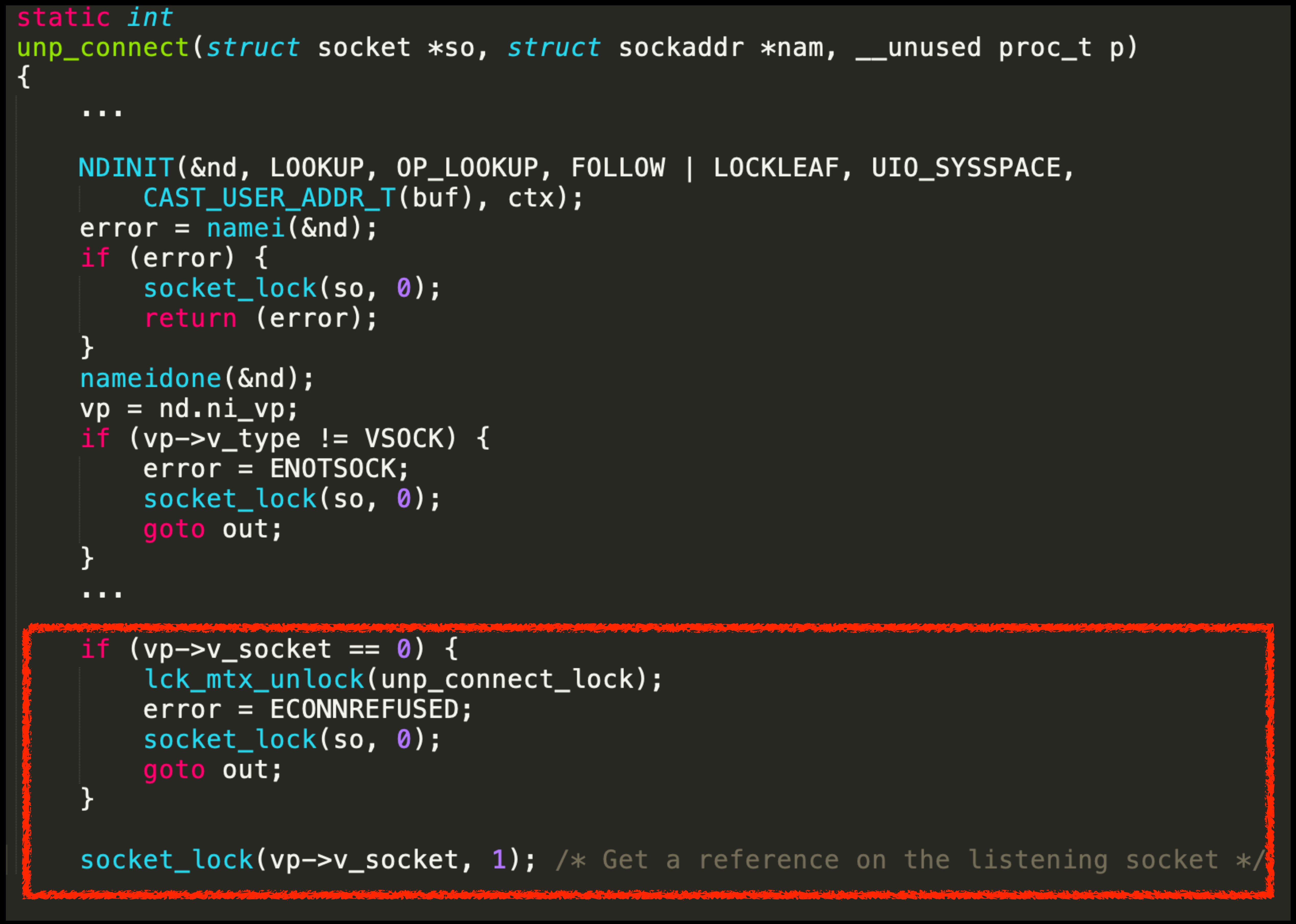
```c
int sock;
sock = socket(AF_UNIX, SOCK_DGRAM, 0);

/* Connect the socket to the path1. */
connect(sock, (struct sockaddr *)&name1,
        SUN_LEN(&name));
/* Connect the socket to the path2. */
connect(sock, (struct sockaddr *)&name2,
        SUN_LEN(&name));
```

Trigger UAF by connecting two names



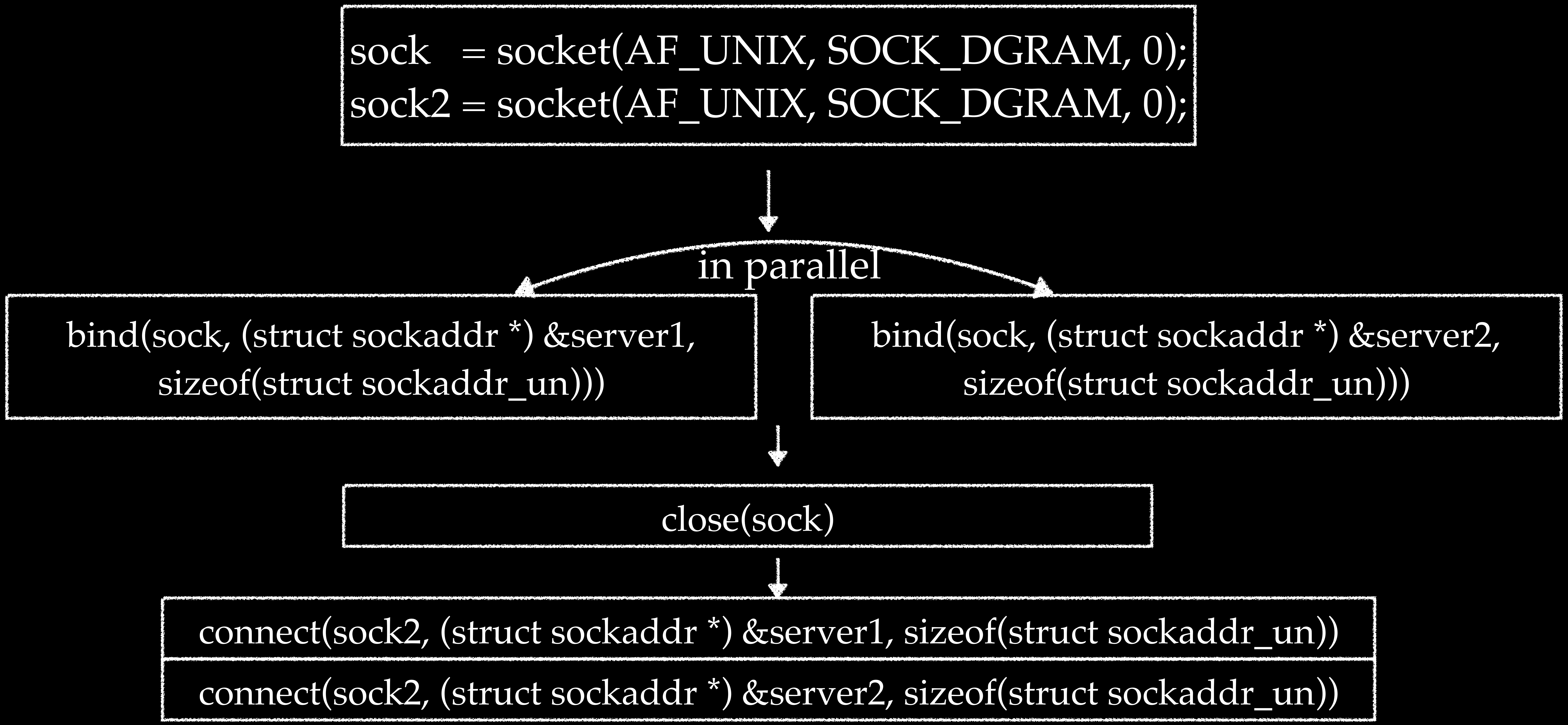
From the kernel point of view





```
static int
unp_connect(struct socket *so, struct sockaddr *nam, __unused proc_t p)
{
    ...

    NDINIT(&nd, LOOKUP, OP_LOOKUP, FOLLOW | LOCKLEAF, UIO_SYSSPACE,
        CAST_USER_ADDR_T(buf), ctx);
    error = namei(&nd);
    if (error) {
        socket_lock(so, 0);
        return (error);
    }
    nameidone(&nd);
    vp = nd.ni_vp;
    if (vp->v_type != VSOCK) {
        error = ENOTSOCK;
        socket_lock(so, 0);
        goto out;
    }
    ...

    if (vp->v_socket == 0) {
        lck_mtx_unlock(unp_connect_lock);
        error = ECONNREFUSED;
        socket_lock(so, 0);
        goto out;
    }

    socket_lock(vp->v_socket, 1); /* Get a reference on the listening socket */
```

The dangling pointer in one of the vnodes will pass into socket_lock()



```
sock   = socket(AF_UNIX, SOCK_DGRAM, 0);
sock2 = socket(AF_UNIX, SOCK_DGRAM, 0);
```

```
bind(sock, (struct sockaddr *) &server1,
sizeof(struct sockaddr_un)))
```

```
bind(sock, (struct sockaddr *) &server2,
sizeof(struct sockaddr_un)))
```

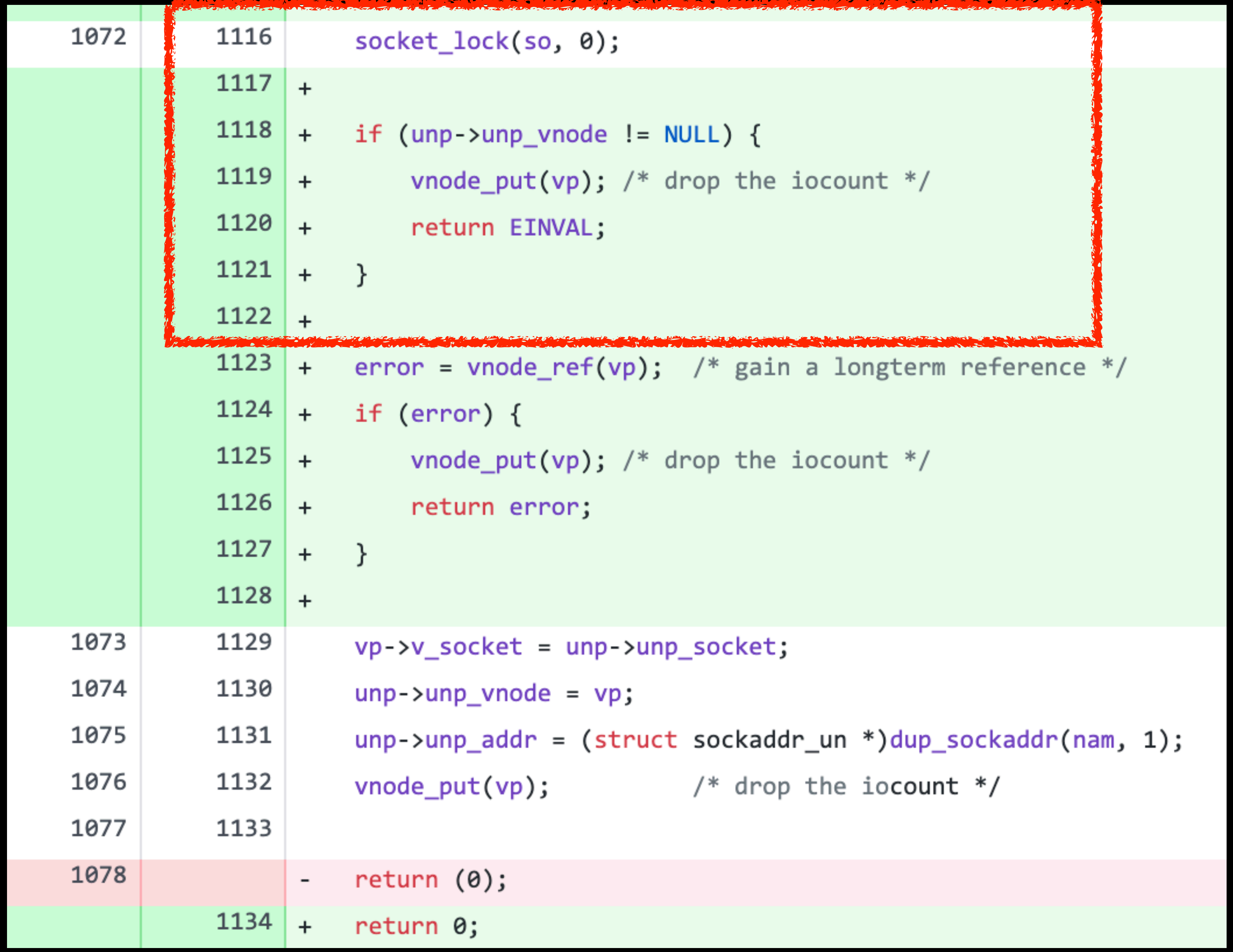
```
close(sock)
```

```
connect(sock2, (struct sockaddr *) &server1, sizeof(struct sockaddr_un))
connect(sock2, (struct sockaddr *) &server2, sizeof(struct sockaddr_un))
```

The race condition bug results in a UAF

The fix

- Fixed in iOS 12.2
  - Still raceable, but adding extra checks to make sure two vnodes will only keep one reference to the socket



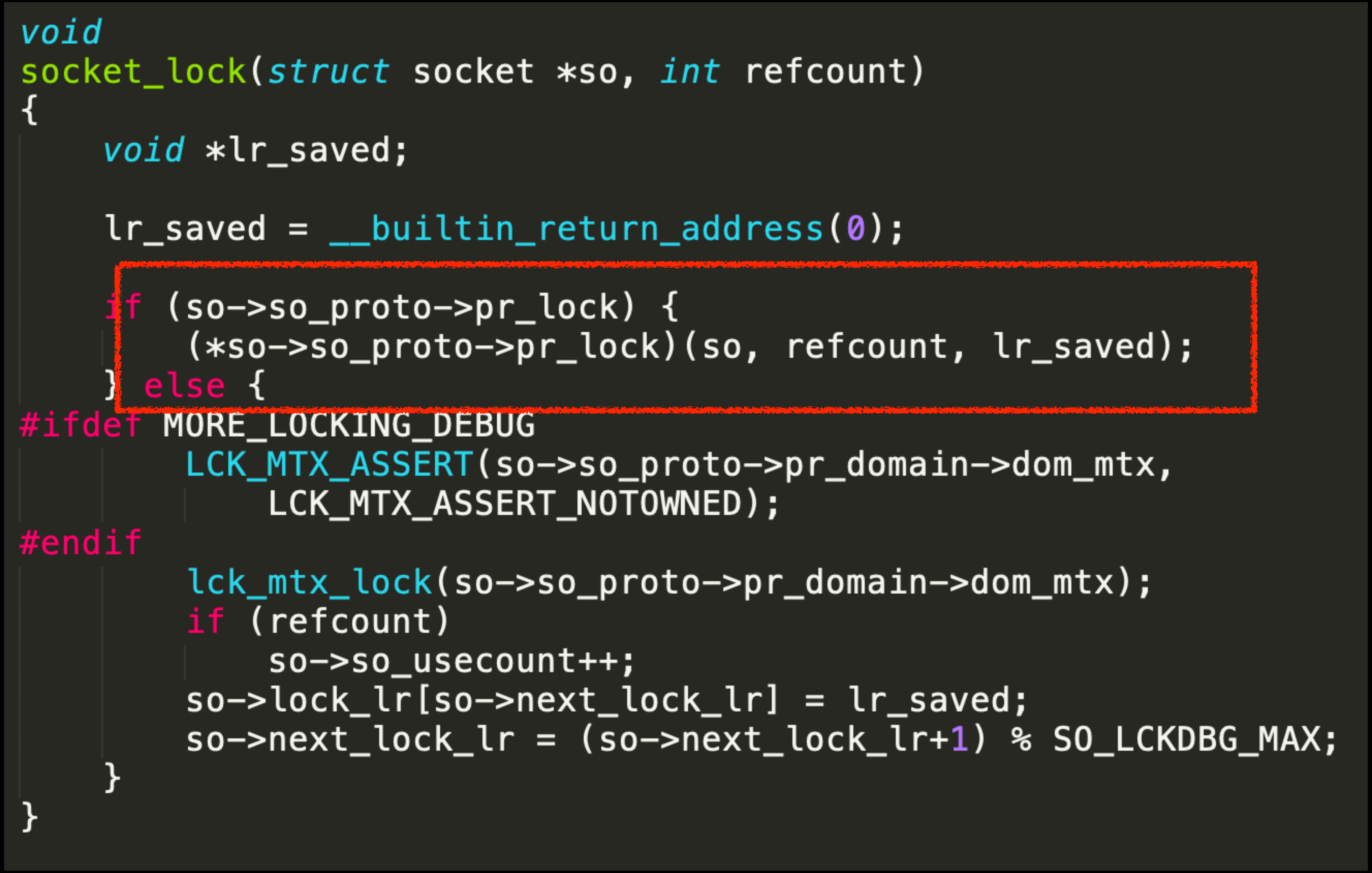
```
     socket_lock(so, 0);
+
+    if (unp->unp_vnode != NULL) {
+        vnode_put(vp); /* drop the iocount */
+        return EINVAL;
+    }
+
+    error = vnode_ref(vp);  /* gain a longterm reference */
+    if (error) {
+        vnode_put(vp); /* drop the iocount */
+        return error;
+    }
+
     vp->v_socket = unp->unp_socket;
     unp->unp_vnode = vp;
     unp->unp_addr = (struct sockaddr_un *)dup_sockaddr(nam, 1);
     vnode_put(vp);           /* drop the iocount */

-    return (0);
+    return 0;
```

Exploitation

```c
void
socket_lock(struct socket *so, int refcount)
{
    void *lr_saved;

    lr_saved = __builtin_return_address(0);

    if (so->so_proto->pr_lock) {
        (*so->so_proto->pr_lock)(so, refcount, lr_saved);
    } else {
#ifdef MORE_LOCKING_DEBUG
        LCK_MTX_ASSERT(so->so_proto->pr_domain->dom_mtx,
            LCK_MTX_ASSERT_NOTOWNED);
#endif
        lck_mtx_lock(so->so_proto->pr_domain->dom_mtx);
        if (refcount)
            so->so_usecount++;
        so->lock_lr[so->next_lock_lr] = lr_saved;
        so->next_lock_lr = (so->next_lock_lr+1) % SO_LCKDBG_MAX;
    }
}
```

Exploitation

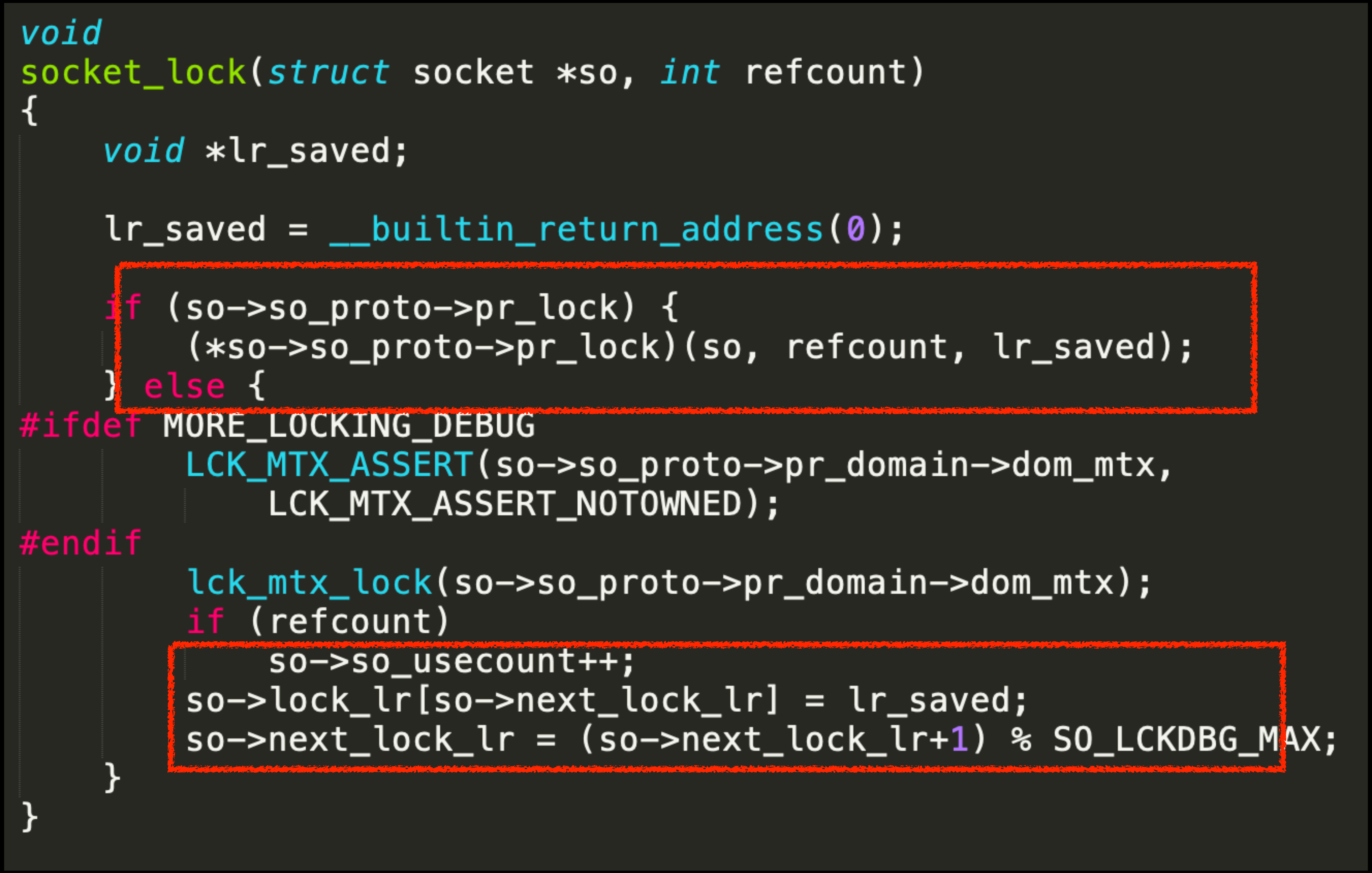
fetch and call a function pointer through two deferences to a freed socket



```
void
socket_lock(struct socket *so, int refcount)
{
    void *lr_saved;

    lr_saved = __builtin_return_address(0);

    if (so->so_proto->pr_lock) {
        (*so->so_proto->pr_lock)(so, refcount, lr_saved);
    } else {
#ifdef MORE_LOCKING_DEBUG
        LCK_MTX_ASSERT(so->so_proto->pr_domain->dom_mtx,
            LCK_MTX_ASSERT_NOTOWNED);
#endif
        lck_mtx_lock(so->so_proto->pr_domain->dom_mtx);
        if (refcount)
            so->so_usecount++;
        so->lock_lr[so->next_lock_lr] = lr_saved;
        so->next_lock_lr = (so->next_lock_lr+1) % SO_LCKDBG_MAX;
    }
}
```

Exploitation

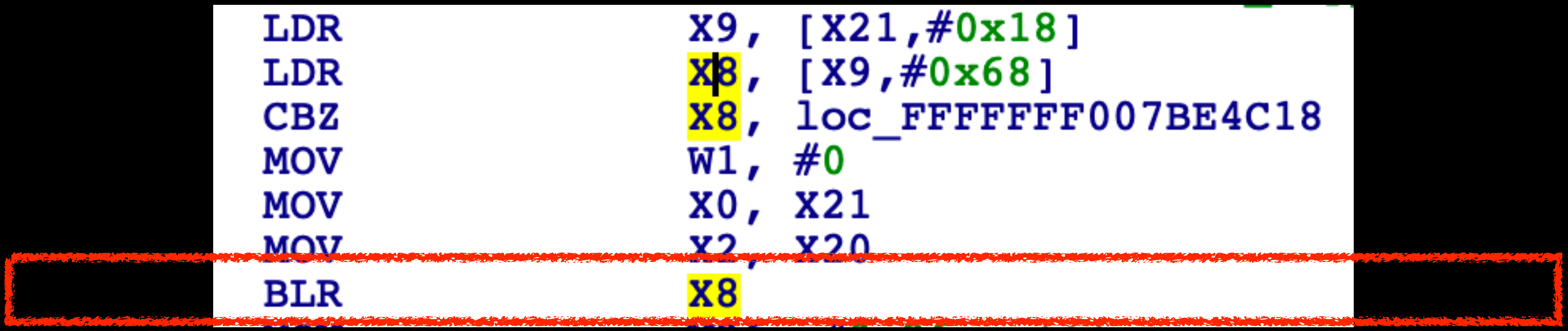
fetch and call a function pointer through two deferences to a freed socket



```
void
socket_lock(struct socket *so, int refcount)
{
    void *lr_saved;

    lr_saved = __builtin_return_address(0);

    if (so->so_proto->pr_lock) {
        (*so->so_proto->pr_lock)(so, refcount, lr_saved);
    } else {
#ifdef MORE_LOCKING_DEBUG
        LCK_MTX_ASSERT(so->so_proto->pr_domain->dom_mtx,
            LCK_MTX_ASSERT_NOTOWNED);
#endif
        lck_mtx_lock(so->so_proto->pr_domain->dom_mtx);
        if (refcount)
            so->so_usecount++;
        so->lock_lr[so->next_lock_lr] = lr_saved;
        so->next_lock_lr = (so->next_lock_lr+1) % SO_LCKDBG_MAX;
    }
}
```

save a return address to the freed socket

Binary version may be better



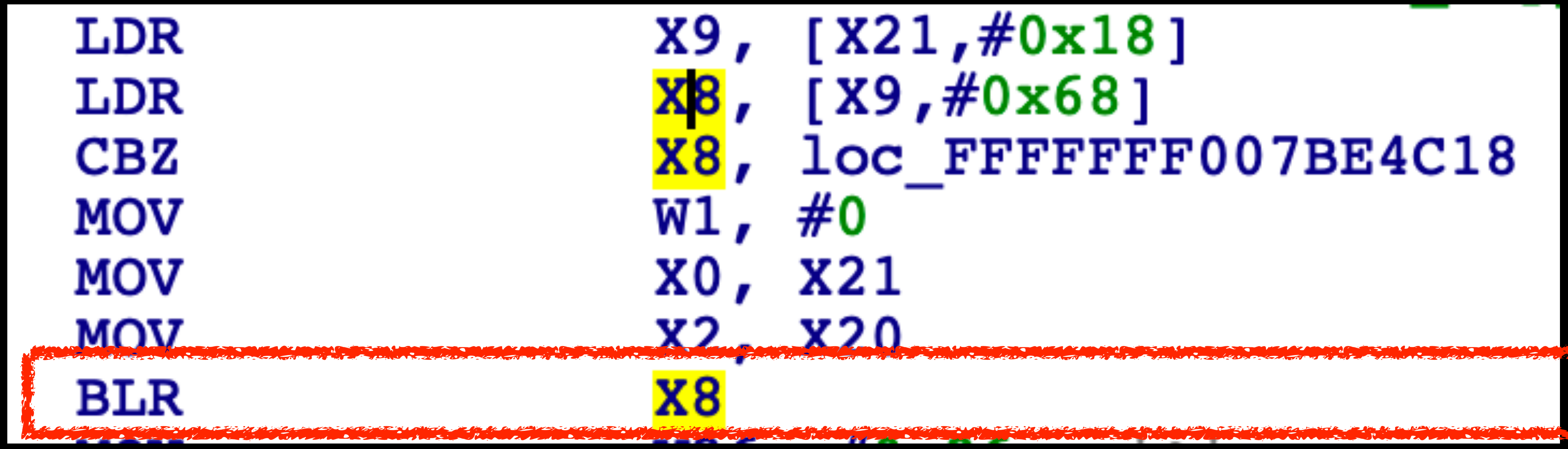
```
LDR                X9, [X21,#0x18]
LDR                X8, [X9,#0x68]
CBZ                X8, loc_FFFFFFF007BE4C18
MOV                W1, #0
MOV                X0, X21
MOV                X2, X20
BLR                X8
```

By controlling X8, we can easily chain ROP/JOP gadgets

JOP/ROP does NOT work on A12
due to the PAC mitigation

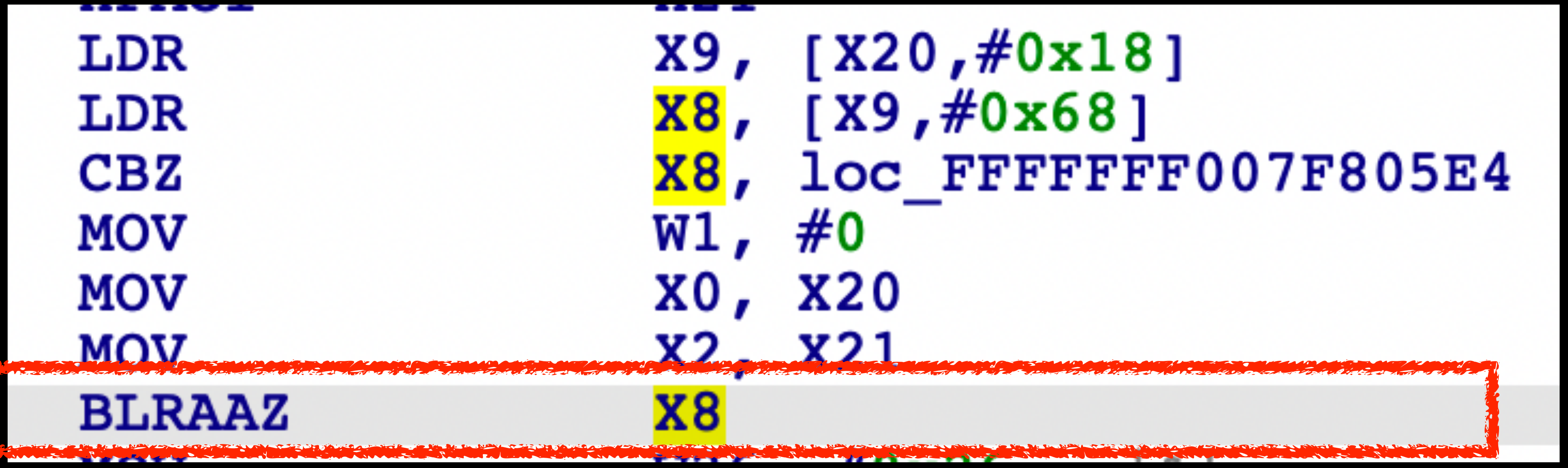
(*so->so_proto->pr_lock)(so, refcount, lr_saved);

Instructions on old devices



```
LDR             X9, [X21,#0x18]
LDR             X8, [X9,#0x68]
CBZ             X8, loc_FFFFFFF007BE4C18
MOV             W1, #0
MOV             X0, X21
MOV             X2, X20
BLR             X8
```

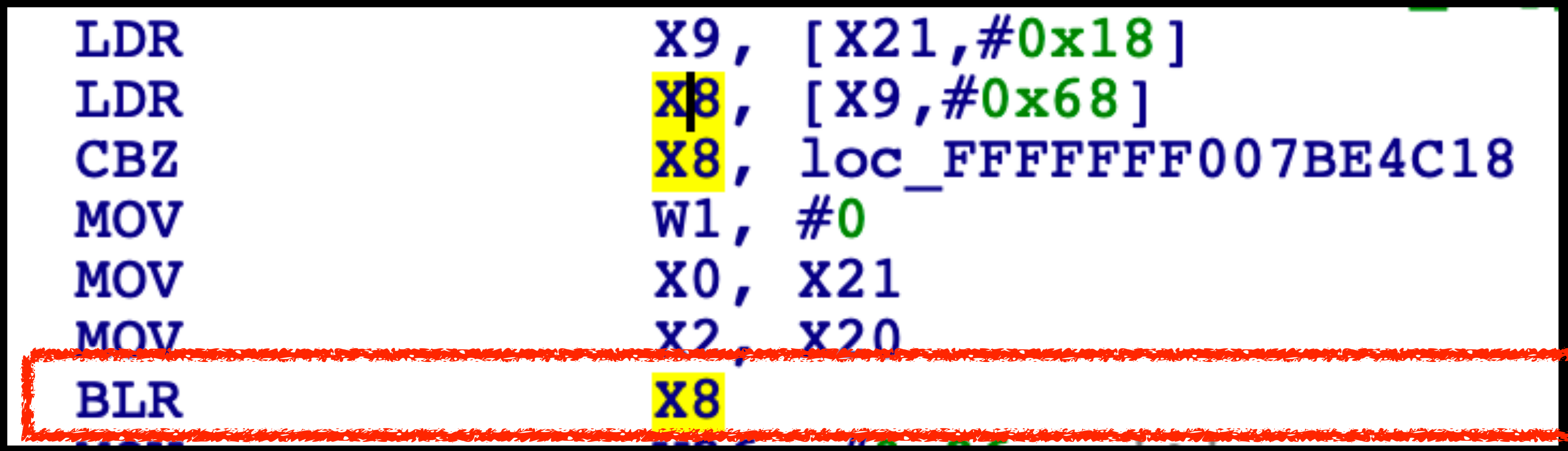
Instructions on A12 devices



```
LDR             X9, [X20,#0x18]
LDR             X8, [X9,#0x68]
CBZ             X8, loc_FFFFFFF007F805E4
MOV             W1, #0
MOV             X0, X20
MOV             X2, X21
BLRAAZ          X8
```

(*so->so_proto->pr_lock)(so, refcount, lr_saved);

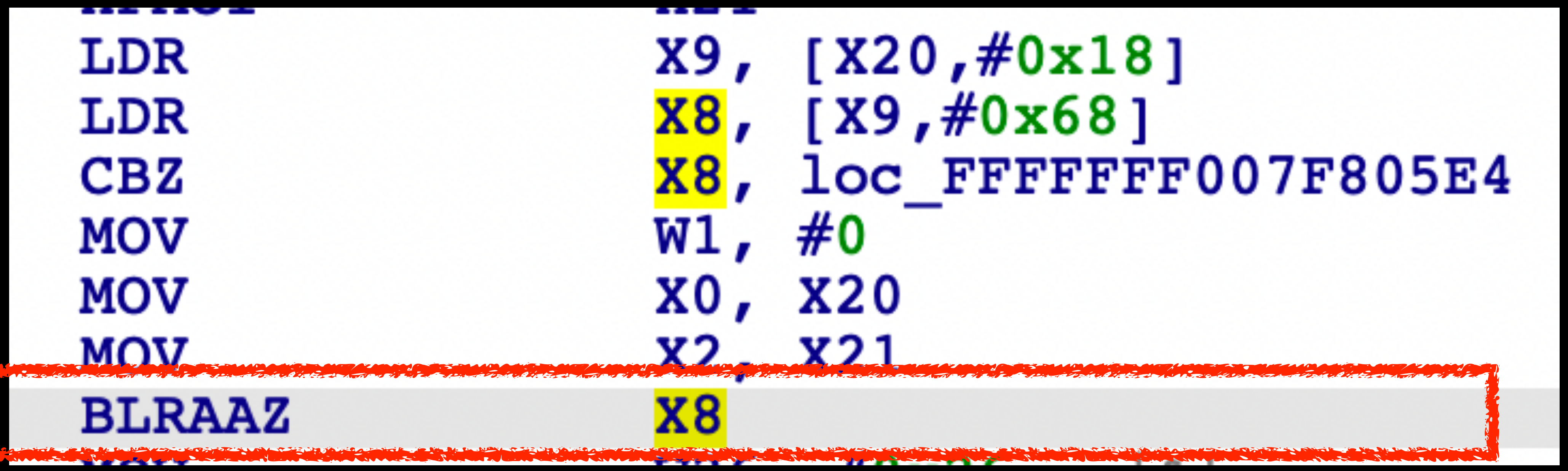
Instructions on old devices



```
LDR             X9, [X21,#0x18]
LDR             X8, [X9,#0x68]
CBZ             X8, loc_FFFFFFF007BE4C18
MOV             W1, #0
MOV             X0, X21
MOV             X2, X20
BLR             X8
```

Hijack control flow by controlling X8

Instructions on A12 devices



```
LDR             X9, [X20,#0x18]
LDR             X8, [X9,#0x68]
CBZ             X8, loc_FFFFFFF007F805E4
MOV             W1, #0
MOV             X0, X20
MOV             X2, X21
BLRAAZ          X8
```

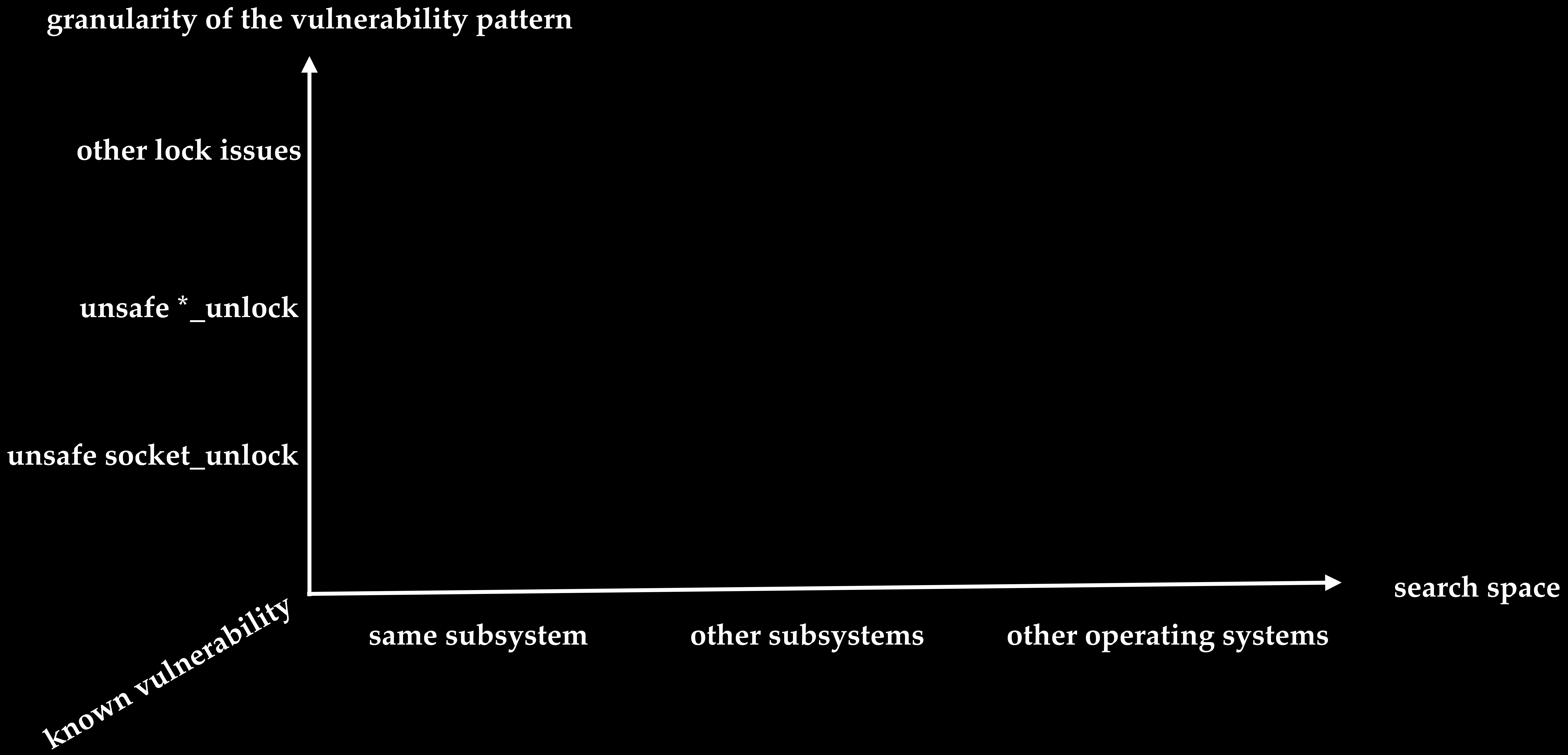
Cannot hijack control flow by controlling X8

- Please refer to our talk at Black Hat USA 2019 for more details regarding how to exploit this vulnerability and bypass PAC

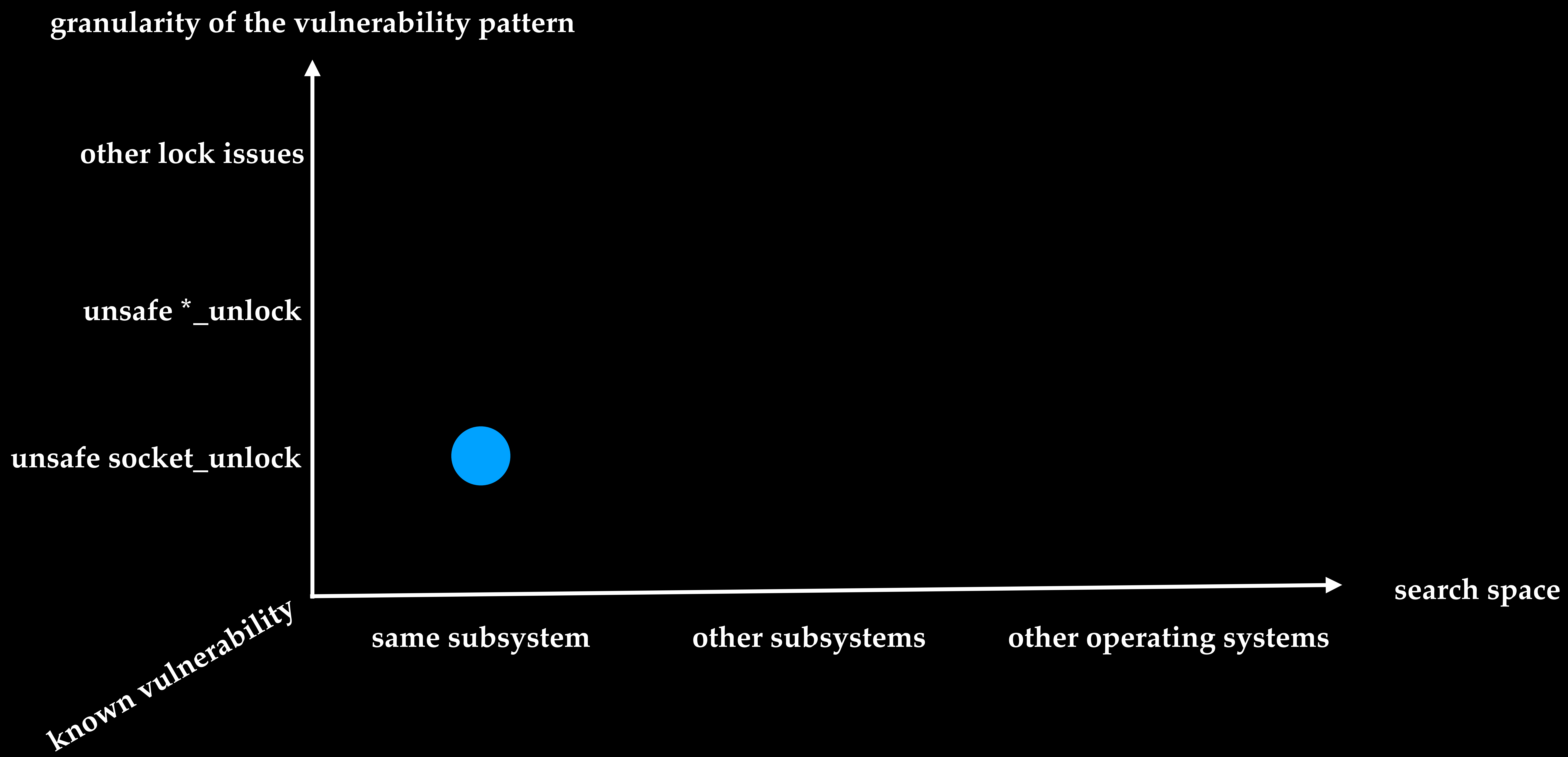
Outline

- ~~Introduction~~
- ~~UNIX Socket Bind Race Vulnerability in XNU~~
- How to Apply Variant Analysis
- Conclusion

Dimensions of variant analysis



Case 1: check the same patten in the same subsystem



check temporary unlocks in unp_connect

```c
int sock;
struct sockaddr_un name;
char buf[1024];
/* Create socket from which to write. */
sock = socket(AF_UNIX, SOCK_DGRAM, 0);

/* Create name. */
name.sun_family = AF_UNIX;
strcpy(name.sun_path, "1.txt");
name.sun_len = strlen(name.sun_path);

/* Connect the socket to the path. */
connect(sock, (struct sockaddr *)&name,
        SUN_LEN(&name));

/* Write to the socket. */
write(sock, buf, 1024);

close(sock);
```

$$\begin{tikzcd}
\text{connect} \arrow[dr] & & & \\
 & \text{connectit} \arrow[dr] \arrow[ddr] \arrow[ddddr] & & \\
 & & \text{socket\_lock} & \\
 & & \text{soconnectlock} \arrow[dr] & \\
 & & & \text{unp\_connect} \\
 & & \text{socket\_unlock} &
\end{tikzcd}$$

check temporary unlocks in unp_connect

```
unp_connect
    └→ socket_unlock
    └→ namei
    └→ socket_lock
    └→ …
    └→ socket_unlock
    └→ sonewconn
    └→ socket_lock
    └→ …
```

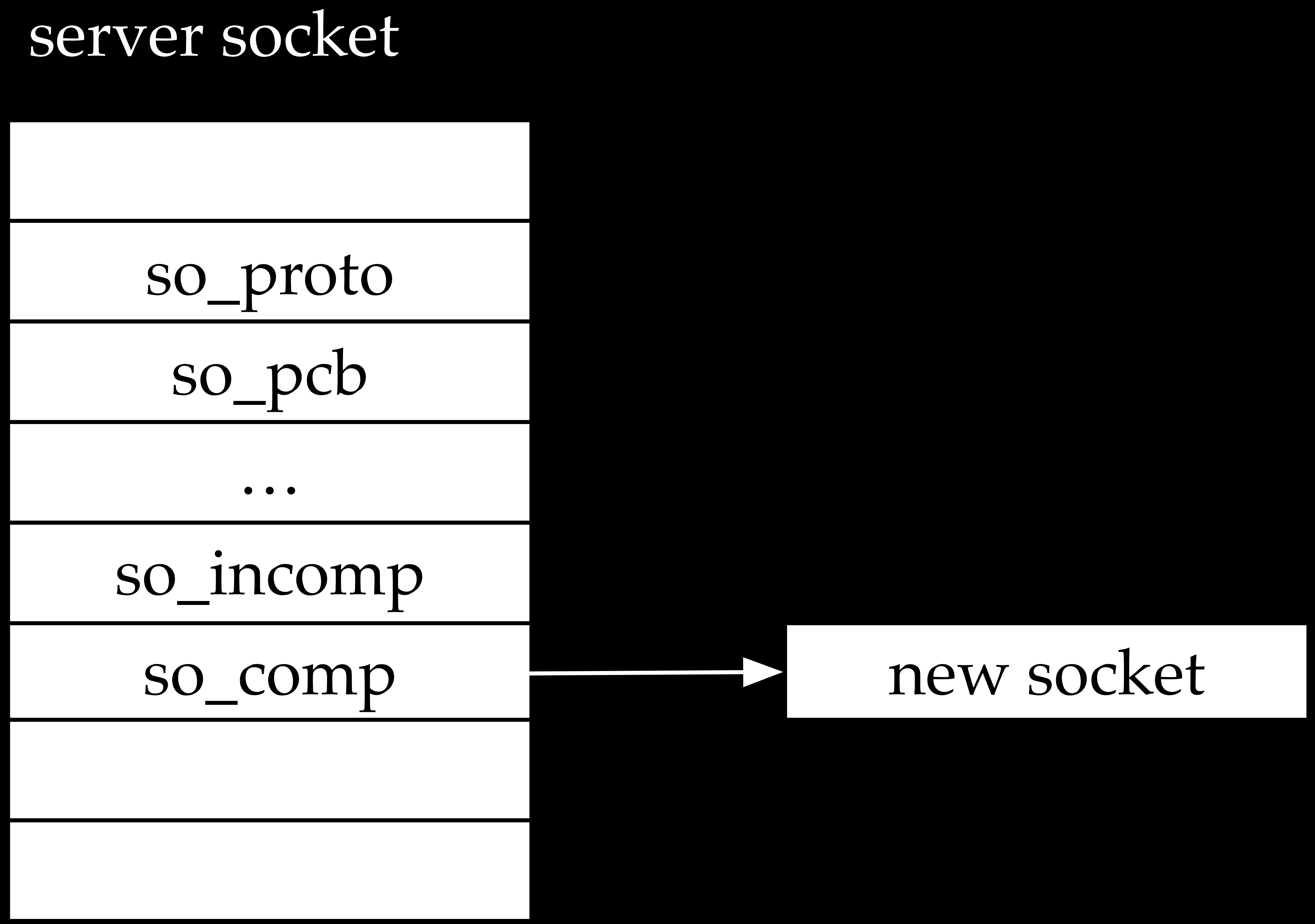
- socket_lock and socket_unlock are called many times
- But the developers are very caution. Every time the socket is re-locked, unp_connect performs checks on any change of the socket state.

```
/*
 * Check if socket was connected while we were trying to
 * get the socket locks in order.
 * XXX - probably shouldn't return an error for SOCK_DGRAM
 */
if ((so->so_state & SS_ISCONNECTED) != 0) {
```

```
/* Check again if the socket state changed when its lock was released */
if ((so->so_state & SS_ISCONNECTED) != 0) {
    error = EISCONN;
```

Normal execution

- A new socket object is created and inserted into the server socket's so_comp queue
- so_incomp: q of partially unaccepted conns
- so_comp: q of complete unaccepted conns



The vulnerability

- The error handling code for race condition leads to a mistake

```
...
socket_unlock(so, 0);
...
so3 = sonewconn(so2, 0, nam);
...
socket_lock(so, 0);
...
/* Check again if the socket state changed when its lock was released */
if ((so->so_state & SS_ISCONNECTED) != 0) {
    ...
    socket_lock(so3, 0);
    sofreelastref(so3, 1);
```

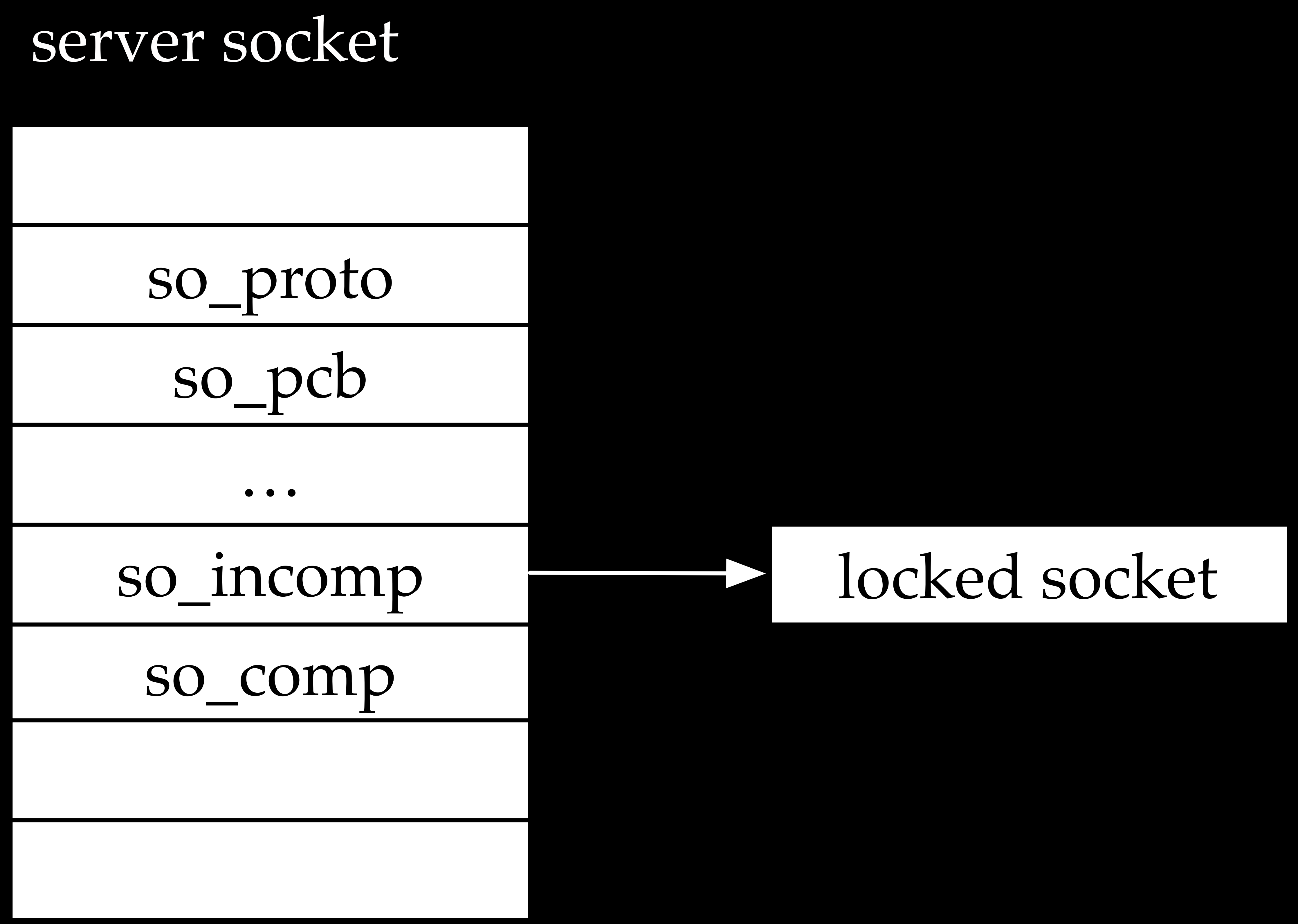
- sofreelastref is supposed to free the newly-created socket object so3, but unfortunately it fails to deallocate the object due to incomplete flag setting

Abnormal execution with race condition detected

- A new socket object is created and inserted into the server socket's so_incomp queue

- The locked socket records the thread_t pointer

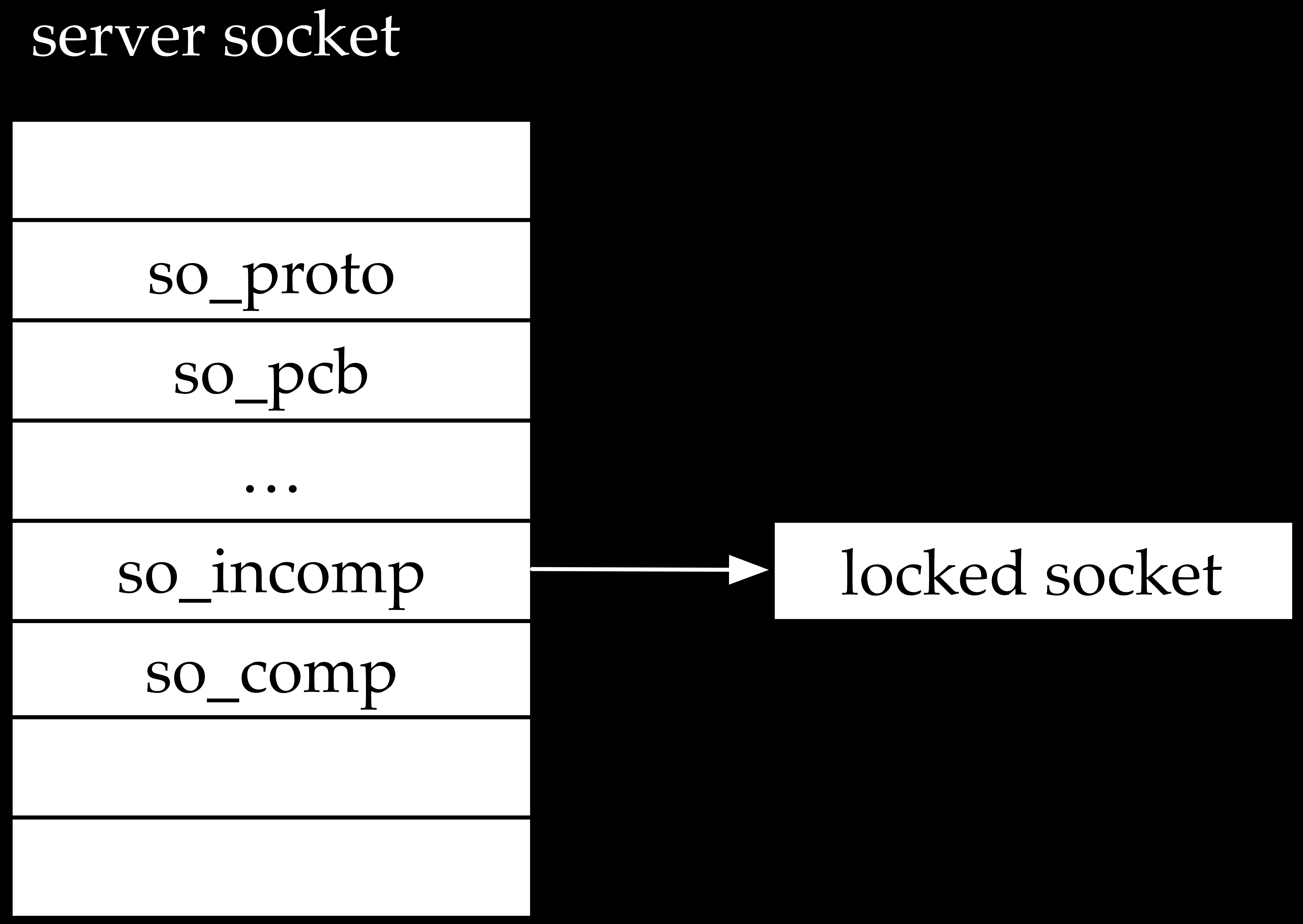
  - After the thread is terminated, the thread_t pointer is invalid



```
socket_lock(so3, 0);
sofreelastref(so3, 1);
```

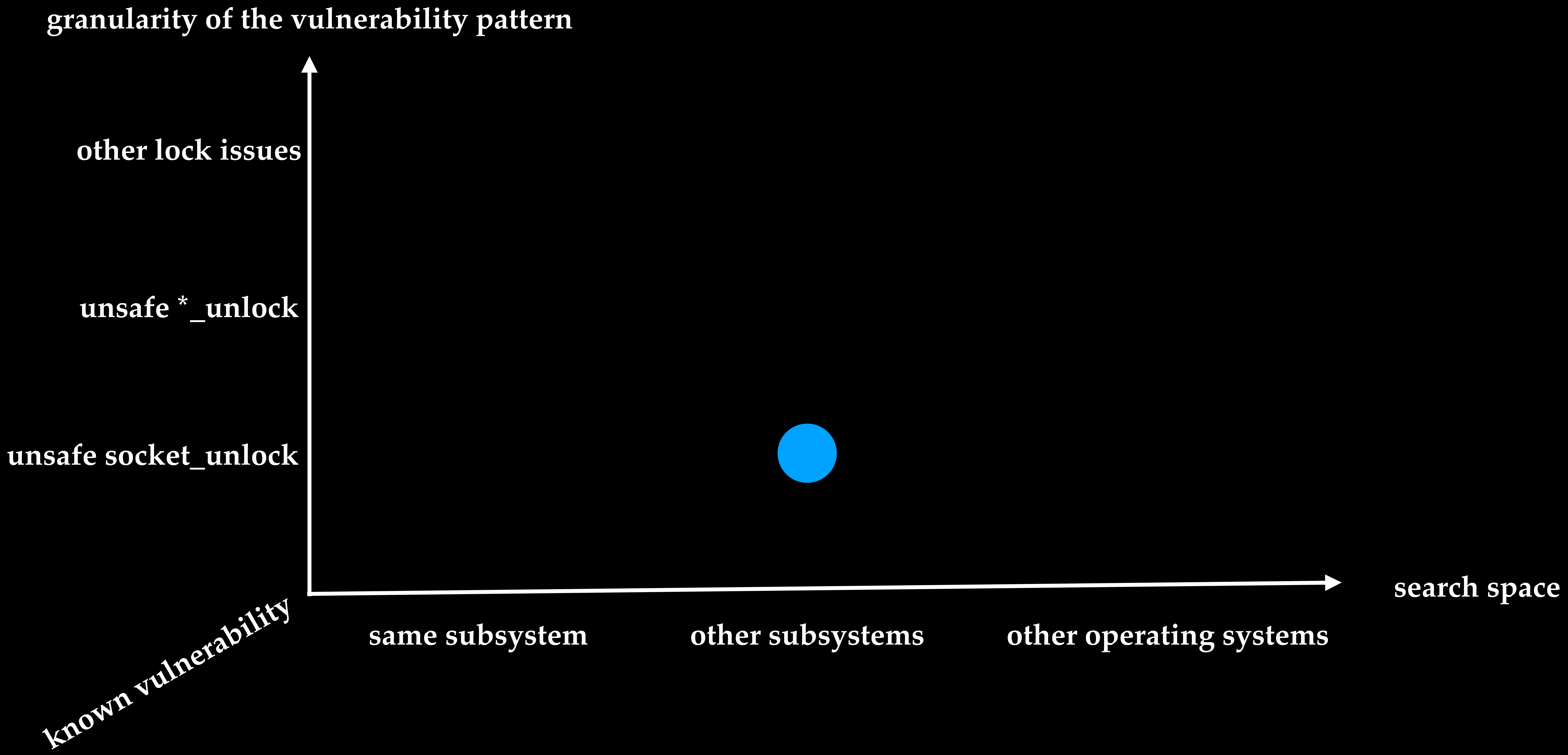
Abnormal execution with race condition detected

- Closing the server socket will lead to cleaning the so_incomp queue
- Cleaning the so_incomp queue will try to relock the socket object
- The relock operation will trigger the thread_t UAF (use-after-free) issue.
- Please refer to https://blog.pangu.io/?p=230 for more details. Apple fixed this issue in iOS 13.7 after we reported it.



```
socket_lock(so3, 0);
sofreelastref(so3, 1);
```

Case 2: check the same patten in other subsystems



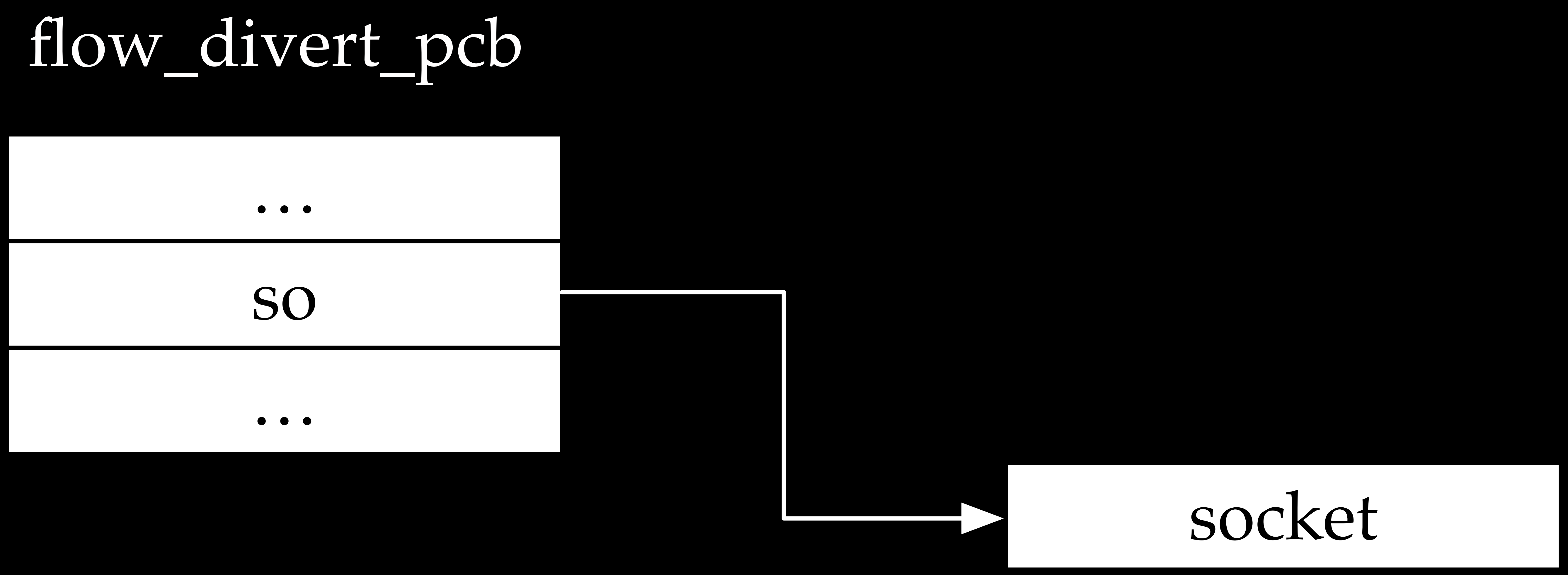
flow-divert socket UAF

- flow-divert is a subsystem in the XNU kernel for flow diversion and network traffic management.
- the temporary unlock of the socket in function flow_divert_pcb_insert leads to a socket UAF vulnerability

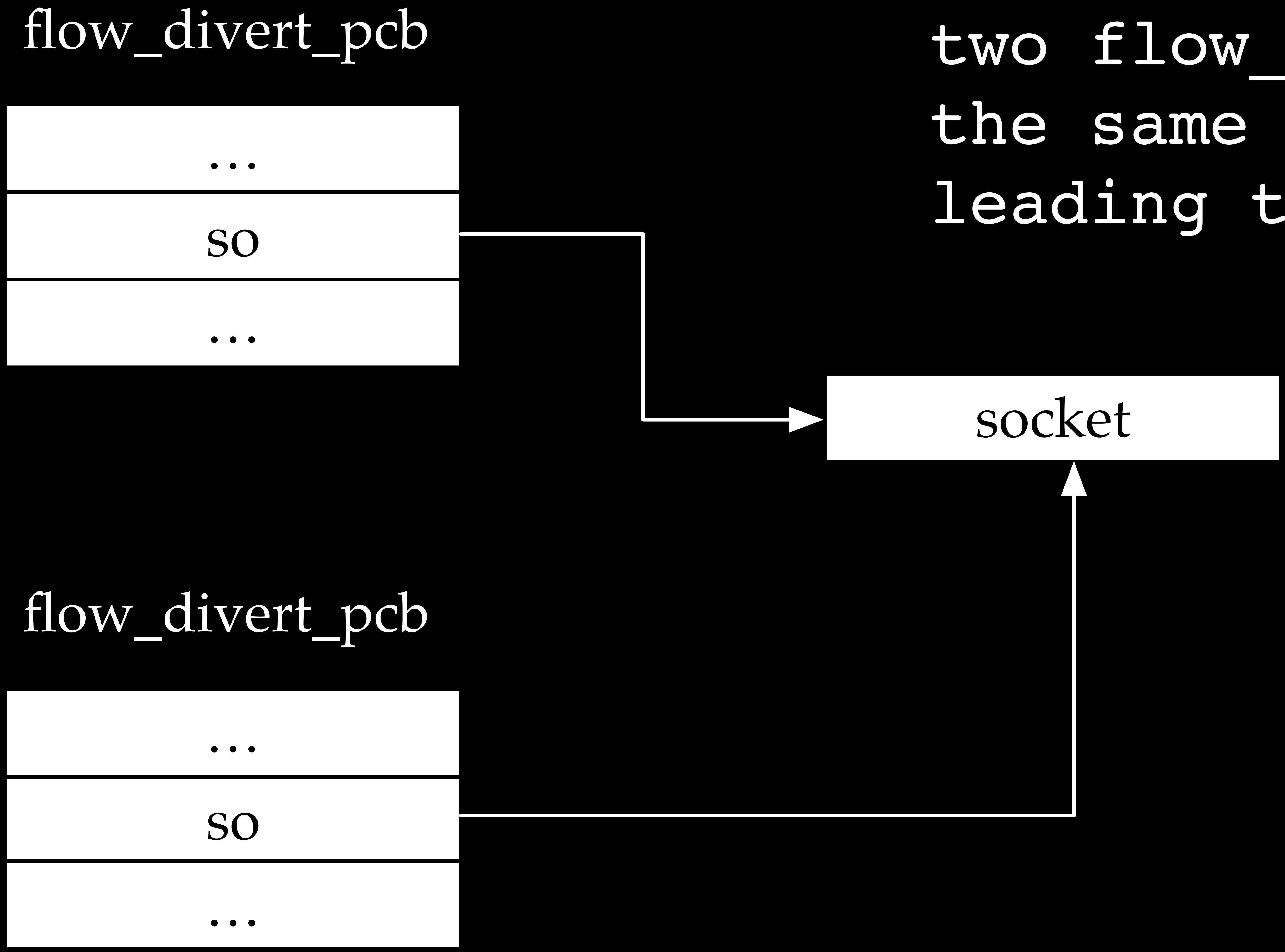
workflow

- socket_lock
- flow_divert_pcb_init
  - MALLOC_ZONE(new_pcb
  - new_pcb->so = so
  - socket_unlock
  - …
  - socket_lock

Normal Execution



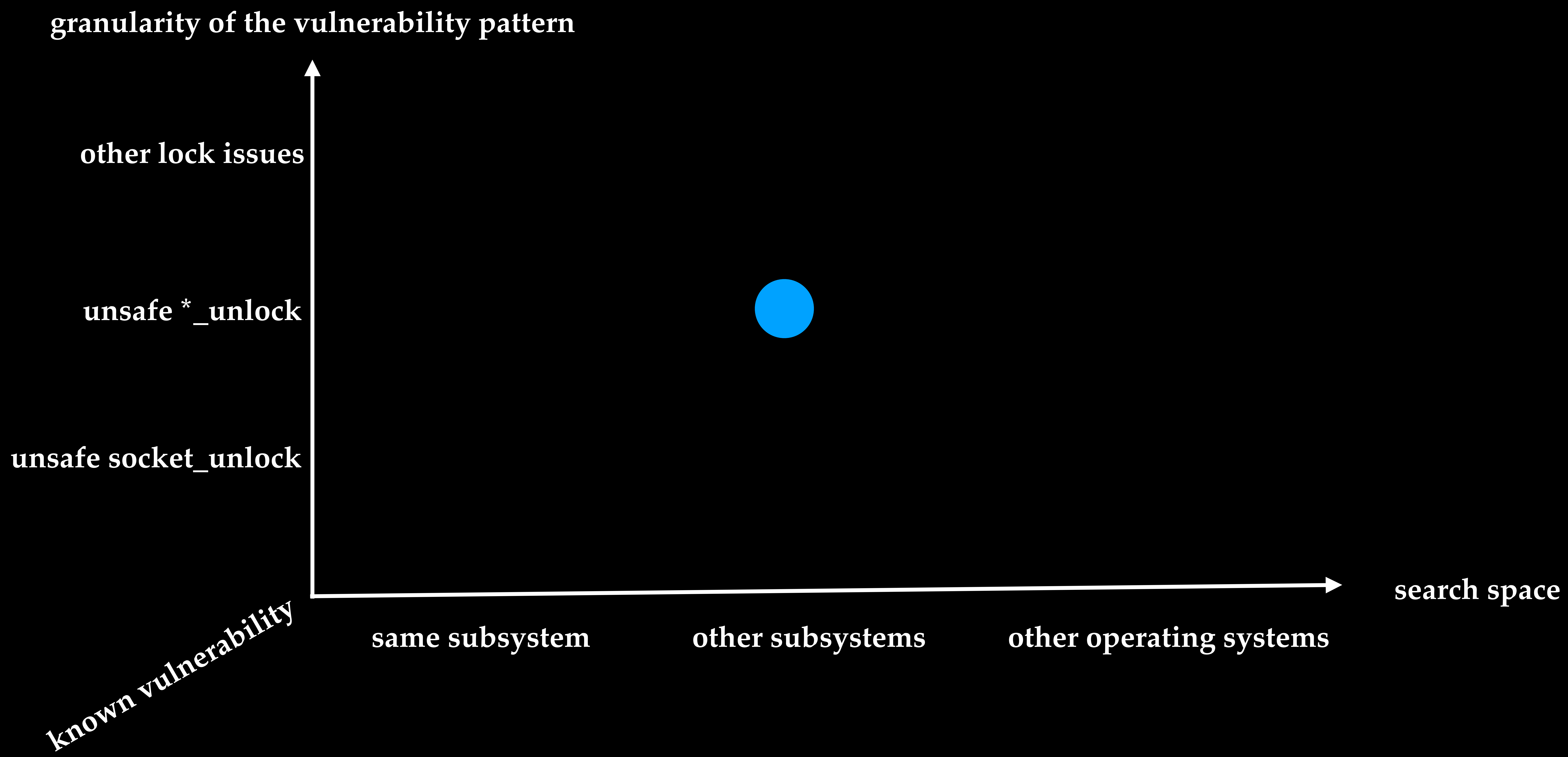
Abnormal Execution under race condition



two flow_divert_pcb pointing to the same socket, eventually leading to socket UAF

Apple fixed the issue in iOS 14

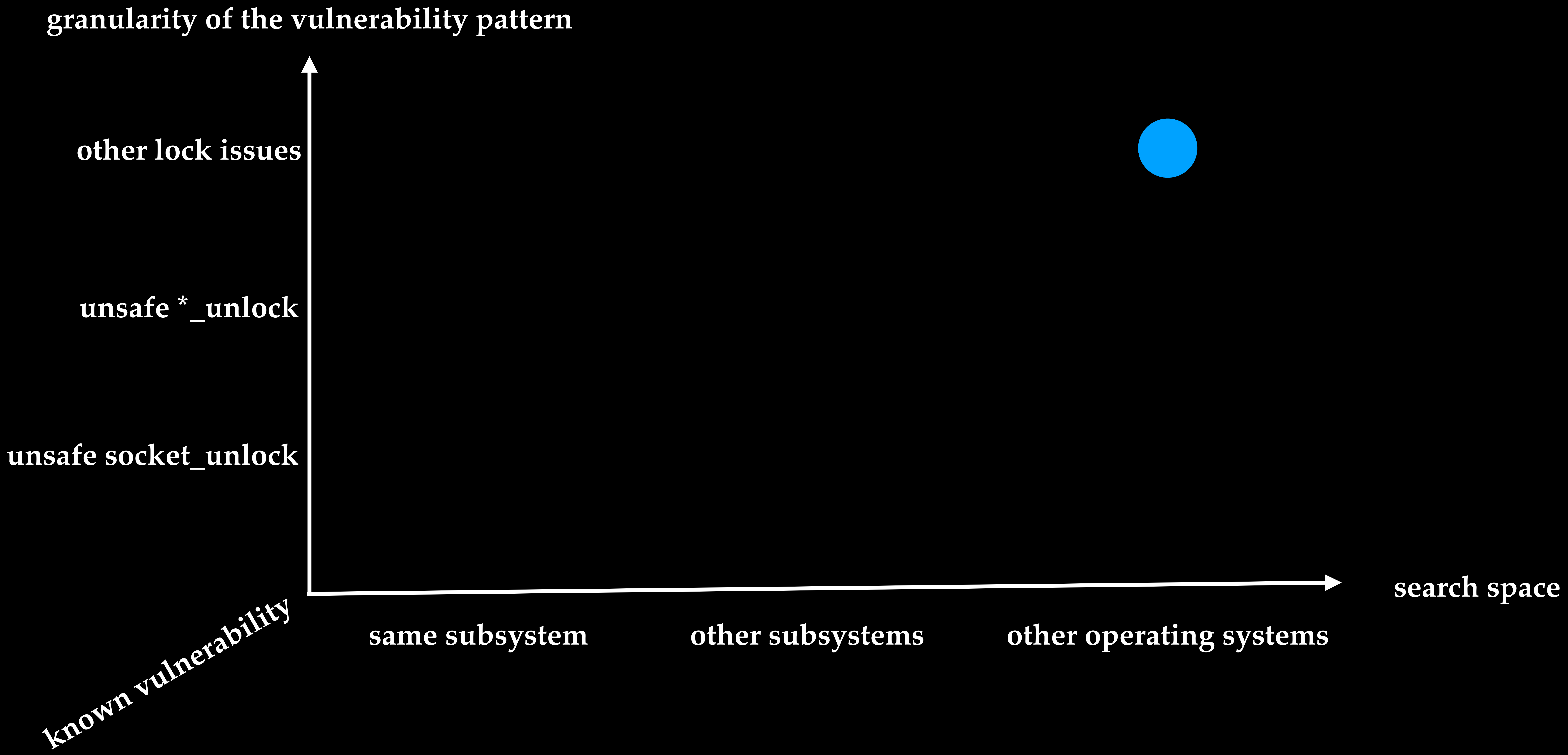
Case 3: check similar pattens in other subsystems



temporary unlocks in other subsystems

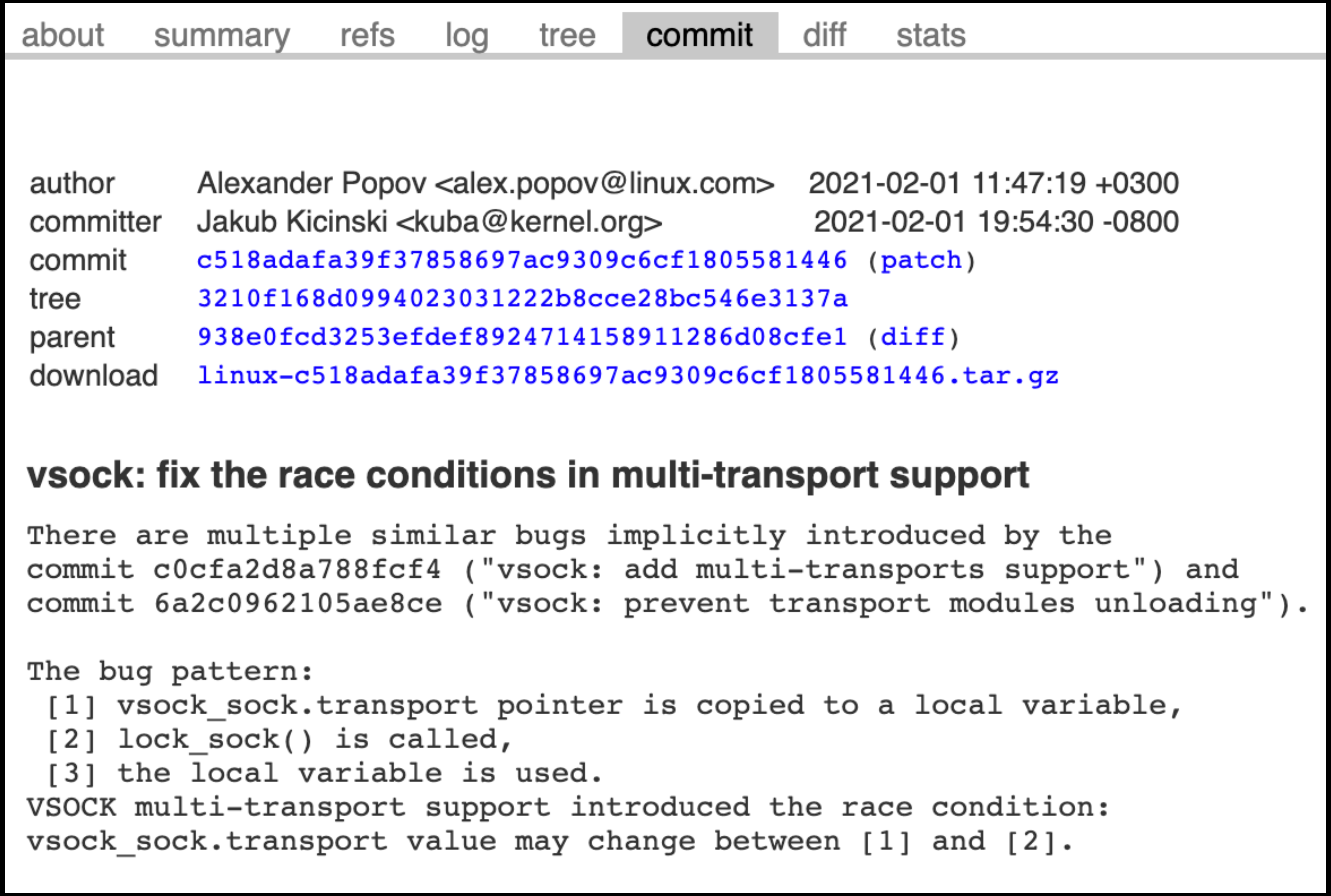
- More and more bugs caused by temporary unlocks were discovered, implying an important bug pattern
- Race condition in VM subsystem
  - CVE-2019-6205, Ian Beer
  - https://googleprojectzero.blogspot.com/2019/04/splitting-atoms-in-xnu.html
- Race condition in IOSurface kernel extension
  - CVE-2017-6979, Adam Donenfeld
  - https://blog.zimperium.com/ziva-video-audio-ios-kernel-exploit/

Case 4: check relative pattens in other OS



vsock race condition in the Linux kernel

- CVE-2021-26708
    - by Alexander Popov



author Alexander Popov <alex.popov@linux.com> 2021-02-01 11:47:19 +0300
committer Jakub Kicinski <kuba@kernel.org> 2021-02-01 19:54:30 -0800
commit c518adafa39f37858697ac9309c6cf1805581446 (patch)
tree 3210f168d0994023031222b8cce28bc546e3137a
parent 938e0fcd3253efdef8924714158911286d08cfe1 (diff)
download linux-c518adafa39f37858697ac9309c6cf1805581446.tar.gz

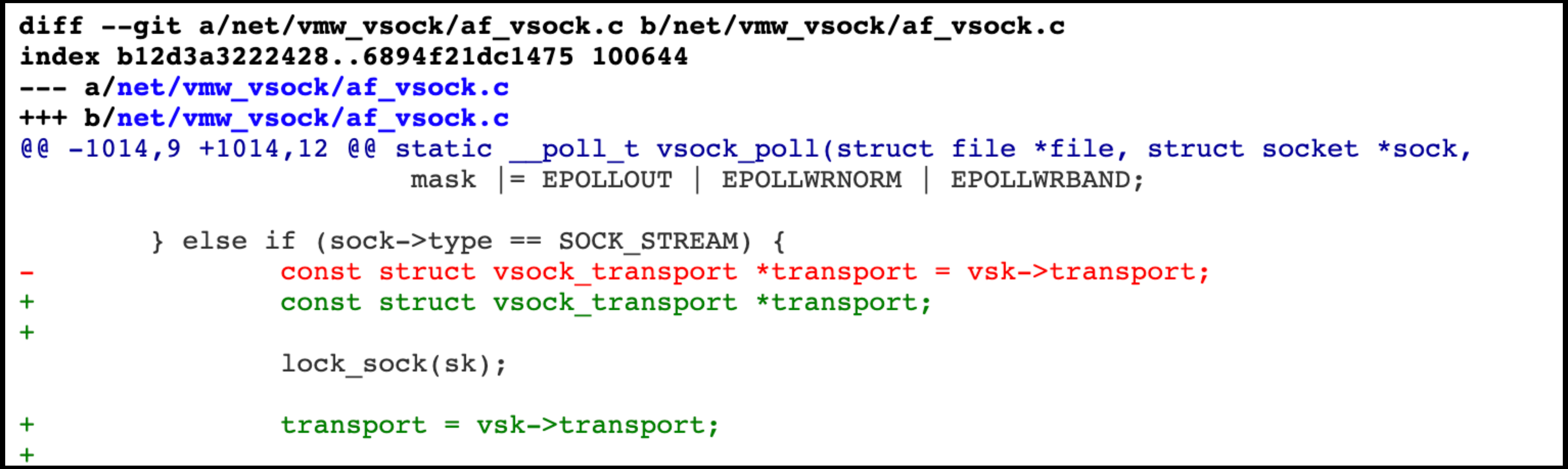
vsock: fix the race conditions in multi-transport support

```
There are multiple similar bugs implicitly introduced by the
commit c0cfa2d8a788fcf4 ("vsock: add multi-transports support") and
commit 6a2c0962105ae8ce ("vsock: prevent transport modules unloading").

The bug pattern:
 [1] vsock_sock.transport pointer is copied to a local variable,
 [2] lock_sock() is called,
 [3] the local variable is used.
VSOCK multi-transport support introduced the race condition:
vsock_sock.transport value may change between [1] and [2].
```

vsock race condition in the Linux kernel

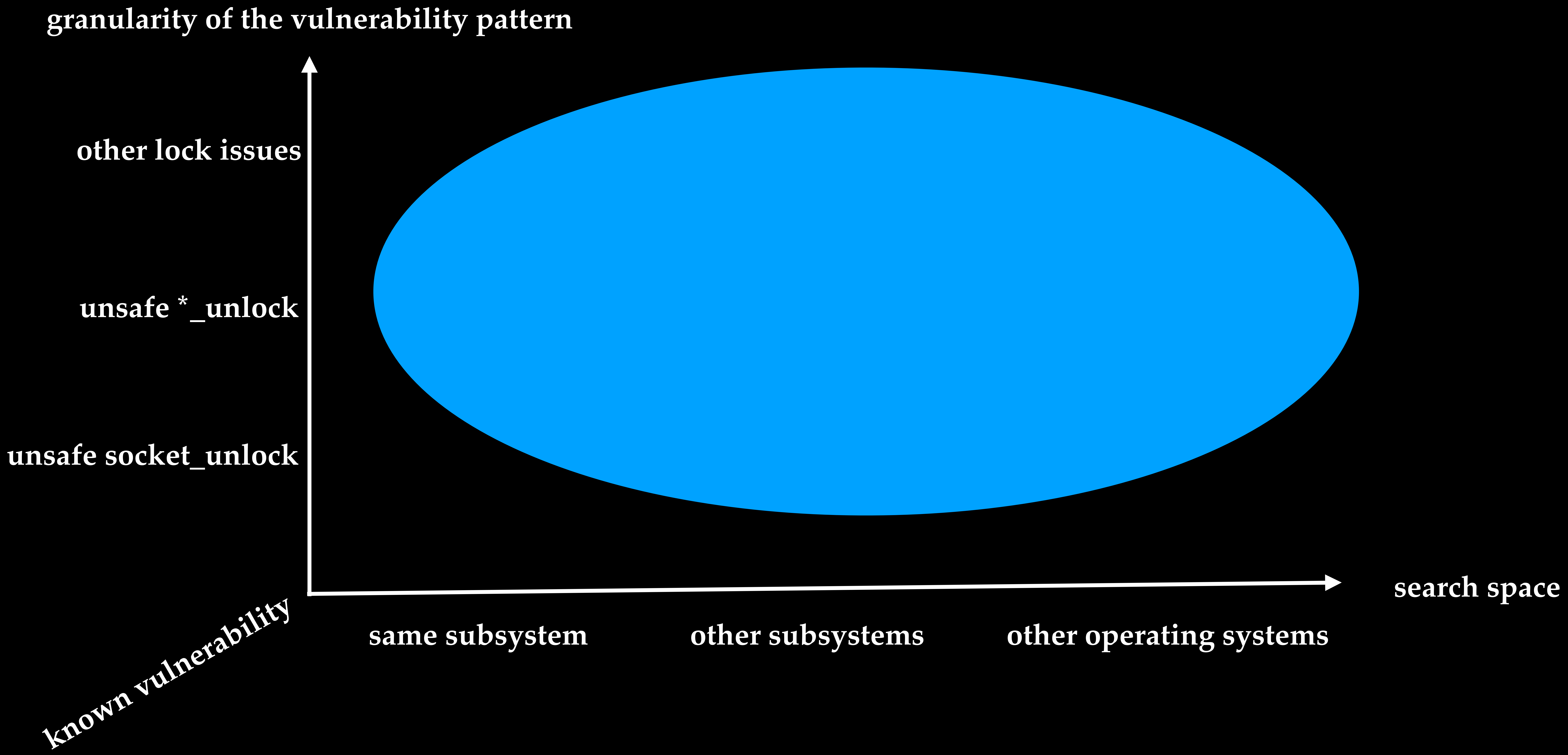
- vsk->transport pointer, is copied into a local variable, which is not protected by the lock_sock

- vsk->transport may be changed / freed by another thread while being used by current thread



```
diff --git a/net/vmw_vsock/af_vsock.c b/net/vmw_vsock/af_vsock.c
index b12d3a3222428..6894f21dc1475 100644
--- a/net/vmw_vsock/af_vsock.c
+++ b/net/vmw_vsock/af_vsock.c
@@ -1014,9 +1014,12 @@ static __poll_t vsock_poll(struct file *file, struct socket *sock,
                                mask |= EPOLLOUT | EPOLLWRNORM | EPOLLWRBAND;

        } else if (sock->type == SOCK_STREAM) {
-               const struct vsock_transport *transport = vsk->transport;
+               const struct vsock_transport *transport;
+
                lock_sock(sk);

+               transport = vsk->transport;
+
```

Don’t limit your imagination



Conclusion

- People usually make similar mistakes
- Programmers usually make similar bugs
- How to automate variant analysis?

Thank you!