E’rybody Gettin’ TIPC
Demystifying Remote Linux Kernel Exploitation

Sam Page, HITB2022SIN
$ whoami

- sam (@sam4k1)
- i do vr and xdev
- linux, security and gaming enthusiast
$ ls talk/

1. shock
2. denial
3. anger
4. bargaining
5. depression
6. testing
7. acceptance
shock
aka discovery
shock
aka discovery

• at the time was looking for a cve, play it safe right?
  • look for low-hanging fruit: simple primitive, familiar module, existing poc?
• queue cve-2021-43267, a remote linux kernel heap overflow (@maxpl0it)
  • spoiler alert: none of the above, but … RCE???
• enter panic gameplan
shock
the gameplan

• these are inherently complex, open-ended problems
• no clear route to “winning”, sometime’s no route at all
• let’s break what can seem a daunting task into simpler steps:
  1. develop understanding of exploitation primitive and attack surface
  2. use this to put together plan of attack(s)
  3. begin enumerating surface for primitives
  4. win ???
shock
developing understanding

• unauthd, remote heap overflow of attacker controlled data
  • affected kernel vers? trigger? constraints? target caches?
• remote attack surface is still a lot of code…
  • let’s start with the net/tipc module
  • intra-cluster comms managed via nodes and their links, used by telcos
  • interested in remote surface; TIPC messages types and handling?
  • queue far too much time trawling through docs, pcaps and src
shock
interrupt vs process context
shock
plan of attack

• we understand what we have: arb heap overflow
• we understand where we are: 5.10 - 5.15 kernel in net/tipc
• coming up with a plan of attack:
  1. remote heap feng shui primitives
  2. leveraging mem corruption to gain CFHP
  3. using CFHP to pivot from interrupt context to process context
  4. pivot into full RCE via final payload (e.g. reverse shell or smth right?)
sh

Oh and mitigations
shock

enumerating primitives

• primitives? building blocks that help us progress our attack plan

• many techniques and approaches, here’s mine:
  • developing deep understanding is fundamental
  • documentation and a methodical, targeted approach
  • static analysis to locate candidates
  • deeper analysis via kernel debugging

snippet from my many, many markdown notes
The shock

the shock

pls don’t @ me regex wizards

CVE-2022-0435

```c
/* tipc_mon_rcc - process monitor domain event message
  * @param struct *net, void *data, u16 dlen, u32 addr,
  * struct tipc_mon_state *state, int bearer_id)
  *
  * struct tipc_mon_domain *arrv_dom = data;
  * struct tipc_mon_domain *mbr;
  * struct tipc_mon_domain *dom;
  * struct tipc_mon_domain *arrv_peer;
  * u16 new_member_cnt = mon_le16_to_cpu(arrv_dom->member_cnt);
  * int new_dlen = dom_rec_len(arrv_dom, new_member_cnt);
  * u16 new_gen = mon_le16_to_cpu(arrv_dom->gen);
  * u16 acked_gen = mon_le16_to_cpu(arrv_dom->ack_gen);
  * u16 arrv_gen = mon_le16_to_cpu(arrv_dom->gen);
  * ...
  *
  * /* Sanity check received domain record */
  * if (dlen < dom_rec_len(arrv_dom, 0))
  *     return;
  * if (dlen != dom_rec_len(arrv_dom, new_member_cnt))
  *     return;
  * if (dlen < new_dlen || arrv_dlen != new_dlen)
  *     return;
  * ...
  *
  * /* Cache current domain record for later use */
  * dom->member_cnt = 0;
  * dom = peer->domain;
  * if (dom)
  *     memcpy(&dom->bf, dom, dom->len);
  *
  * /* Transform and store received domain record */
  * if (!dom || (dom->len < new_dlen)) {
  *     kfree(dom);
  *     dom = kmalloc(new_dlen, GFP_ATOMIC);
  *     peer->domain = dom;
  *     if (!dom)
  *         goto exit;
  * }
  * ...
  */
```
denial
aka verification & disclosure
denial
aka verification & disclosure

- double quadruple checking this is legit
- time to move onto the disclosure process... *cries in whitespace*
  - embargoed disclosure, patch submission, public disclosure

On Thu, Jan 27, 2022 at 02:38:06PM +0000, Samuel Page wrote:

> The information contained in this email is confidential

I don't see content in this email apart from the partially quoted disclaimer. Probably something went wrong (or my MUA is screwing with me)
anger

aka trying to achieve RCE on a modern kernel
anger
aka trying to achieve RCE on a modern kernel

• let’s recall our gameplan:
  1. develop understanding of exploitation primitive and attack surface
  2. use this to put together plan of attack(s)
  3. begin enumerating surface for primitives
  4. win ???
anger
developing understanding

• our understanding on net/tipc still relevant
• however, exploit primitive has changed
  • diff requirements to reach RCE now
  • looking at a remote stack overflow now
• ~1400 byte payload, 272 byte stack buffer
• execution flow is in the interrupt context
• kernels 4.8 through 5.16
anger
plan of attack

- an updated plan of attack:
  1. leverage stack overflow CFHP to more flexible arb code execution
  2. use code exec to pivot from interrupt ctx to process ctx
  3. pivot into full RCE via final payload (e.g. reverse shell or smth right?)
Oh and mitigations
bargaining

aka okay what if we just got rid of KASLR and canaries?
bargaining
aka okay what if we just got rid of KASLR and canaries?

• given a nice leak, what does our plan of attack really look like?

• a high level overview:

1. pivot RIP control to shellcode exec
2. hooking syscalls to pivot to process context
3. using our hook to deliver a user mode payload
4. win ???
bargaining
getting our bearings

• the situation so far:

kernel stack context for our buffer overflow, in `tipc_mon_rcv()`

disassembly snippet showing `tipc_mon_rcv()` epilogue

dump of assembler code for function `tipc_mon_rcv`:
bargaining

getting shell code execution

- rop + set_memory_x()

- jmp to shellcode

- cleanup!!!
bargaining
1337 shellcode to escape process ctx

• now have arb kernel code exec!
• but we’re in the interrupt ctx :(  
• solution? syscall hooking
bargaining

the hook

• now in process context, need to make final pivot to usermode

• no need to reinvent wheel, plenty of tools provided by kernel :)

```c
;; Minimal syscall hook.
;;
;; Essentially does nothing, hands execution onto the correct syscall handler, as defined
;; in the DATA section.

_start:
    jmp DATA
PAYLOAD:
    pop rax
    mov rax, [rax]
    jmp rax
DATA:
    call PAYLOAD
    dq (kbase + sys_reboot)
```

```c
if (not_root() || is_exploited)
    goto cleanup; // via ptregs
save_user_state(payload);
payload_dst = mmap(NULL, payload_size, RWX, MAP_ANON|MAP_PRIVATE, -1, @);
copy(payload_dst, payload, payload);
update_rip(payload+code_offset); // via ptregs

cleanup:
    fix_regs()
    jmp syscall()
```

pseudocode for a full functioning hook
bargaining

win ????

• now have arb code exec in priv process, gg
• still need to cleanup though! don’t know where we are

```c
fork()
if (parent)
    repair_registers() // from the state our hook saved
    jmp_old_ip() // hand back execution to original value
else
    callback_payload() // establish connection with attacker
```
depression
aka let’s actually get round to looking at some mitigations
depression
aka let’s actually get round to looking at some mitigations

• kernel version, arch, config, bug type & techniques are all factors
• cat and mouse game between mitigations and bypass techniques
• want to be aware and factor in relevant mitigations throughout process
  • soft vs hard mitigations
• apply understanding to our specific context, e.g. LPE vs RCE?
depression
contemporary mitigations

• And plenty more out there! (CFI, heap hardening, FG-KASLR etc. etc.)
testing
aka how i do mine, workflow and why emacs is the best ide
testing
aka how i do mine, workflow and why emacs is the best ide

• emacs, yeah i’m being fr
• QEMU + gdb (+ gef)
• structured .md notes, try document much as pos
• generalise solutions when possible, for next time!
• don’t be afraid to share hacky scripts/setups

doom emacs (probably should have put some kernel grokking here, but here’s magit)
acceptance
aka this talk
acceptance
aka this talk

• kernel exploitation is cool
• not so scary once you break it down, draws from lots of skill
• success/winning isn’t binary
• sharing is caring and will make your life + other’s easier
• remote kernel exploitation is both familiar yet wildly different
resources
and misc links

- https://twitter.com/sam4k1
- https://sam4k.com
- https://github.com/sam4k/linux-kernel-resources
- https://github.com/a13xp0p0v/linux-kernel-defence-map
- https://github.com/doomemacs/doomemacs
- https://hugsy.github.io/gef/
- https://elixir.bootlin.com/linux/latest/source
exit(0);