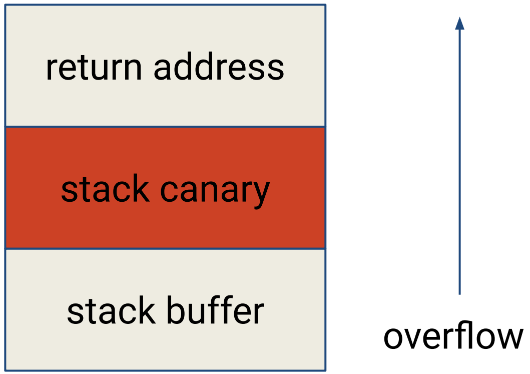
LINUX SECURITY SUMMIT EUROPE

Bypassing Many Kernel Protections Using Elastic Objects

Yueqi Chen & Zhenpeng Lin
The Pennsylvania State University

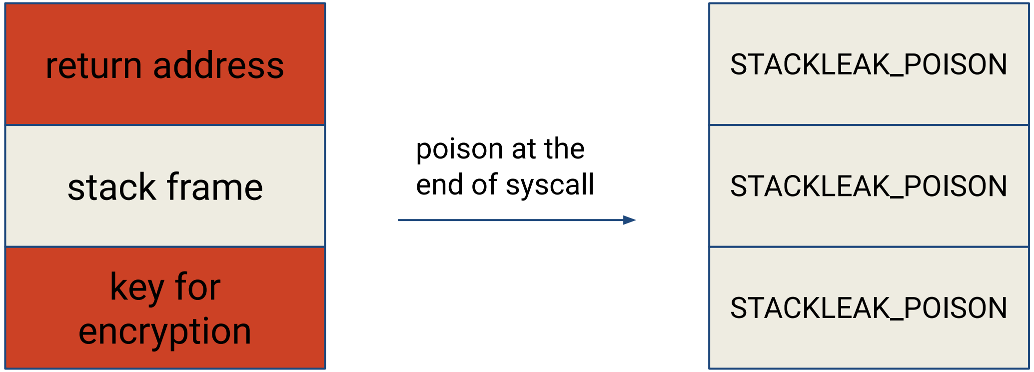
October 30, 2020

Linux Kernel Protections 101 - Stack



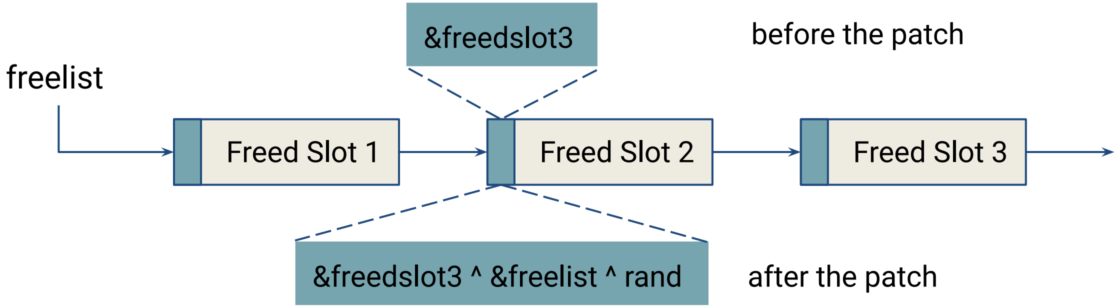
CONFIG_STACKPROTECTOR (stack canary)

Linux Kernel Protections 101 - Stack



CONFIG_INIT_STACK_ALL & CONFIG_GCC_PLUGIN_STACKLEAK

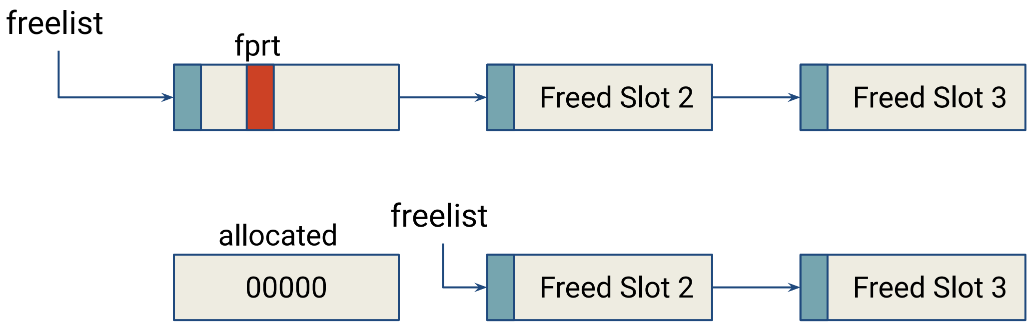
Linux Kernel Protections 101 - SLAB/SLUB



CONFIG_SLAB_FREELIST_HARDENED

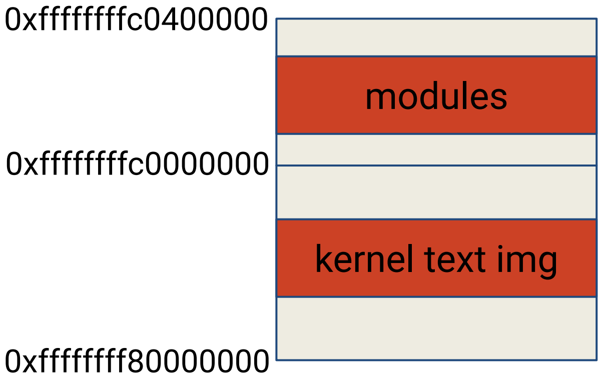
[1] “Weakness in Linux Kernel Heap Hardening” - Dr. Silvio Cesare

Linux Kernel Protections 101 - SLAB/SLUB



CONFIG_INIT_ON_ALLOC_DEFAULT_ON (init_on_alloc)
CONFIG_INIT_ON_FREE_DEFAULT_ON (init_on_free)

Linux Kernel Protections 101 - Memory Layout

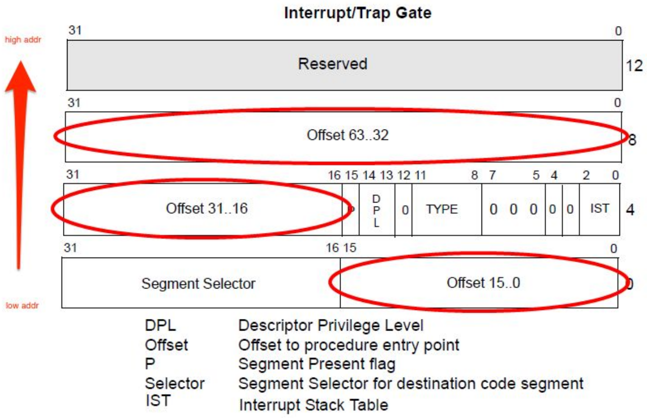


CONFIG_RANDOMIZE_BASE (KASLR)
fgkaslr is out of scope

Other Sensitive Kernel Data

```
root:!:yyy:0:99999:7:::
yueqi:xxx:yyy:0:99999:7:::
```

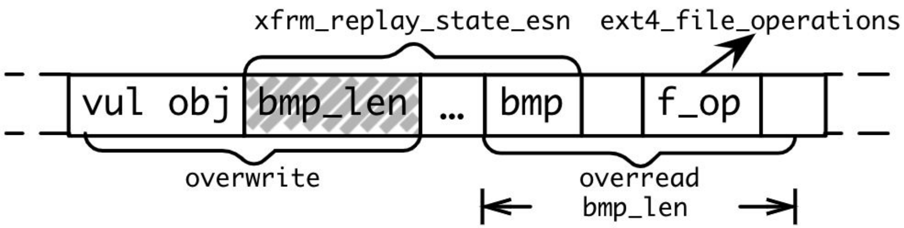
"/etc/shadow" content in gnome-keyring-daemon



Interrupt descriptor table

[2] https://github.com/xairy/kernel-exploits/tree/master/CVE-2017-18344

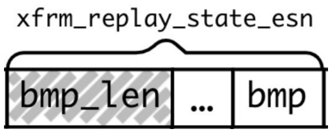
Elastic Object is Not A New Attack



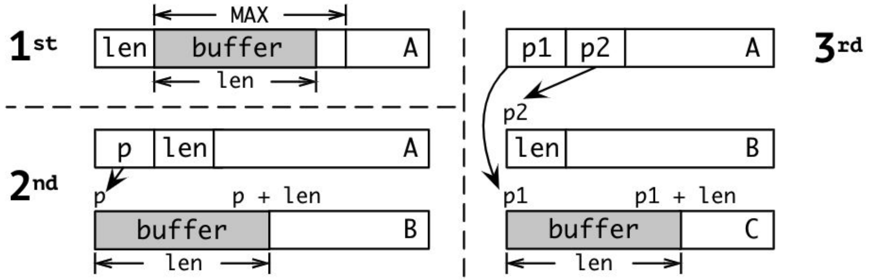
leaked through nla_put() invoked from recvmsg syscall

Flexible Structure: **xfrm_replay_state_esn**
used in Pwn2Own 2017 for CVE-2017-7184

Elastic Object is Extended From Flexible Object

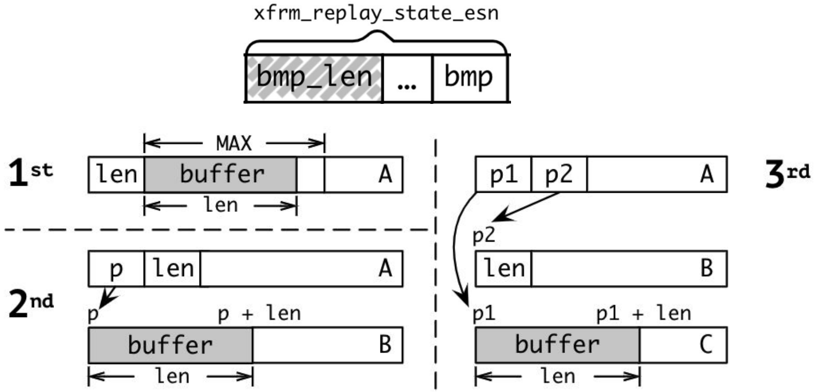


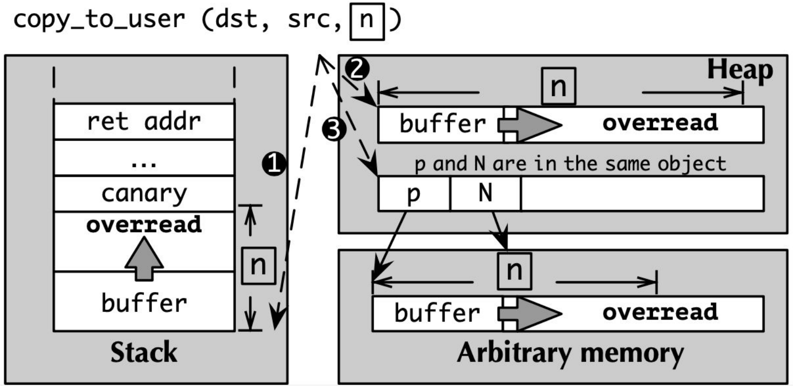
Standard Flexible Object



Extension of Flexible Object
(i.e., Elastic Object)

Using Elastic Objects in Attack





- Assuming that write primitive on kernel heap tampers “p” or “len” in elastic objects
- Later, “len” is propagated to “n” in copy_to_user() and probably also “p” to “src”

Severity of Elastic Object Attack

- Obtain leak primitive from write primitive
- The leak primitive can expose
  - Stack canary
  - Return address on stack for KASLR bypassing
  - Encrypted heap cookies
  - Function pointer value on heap for KASLR bypassing
  - “/etc/shadow” in gnome-keyring-daemon
  - Interrupt descriptor table for KASLR bypassing
  - And more

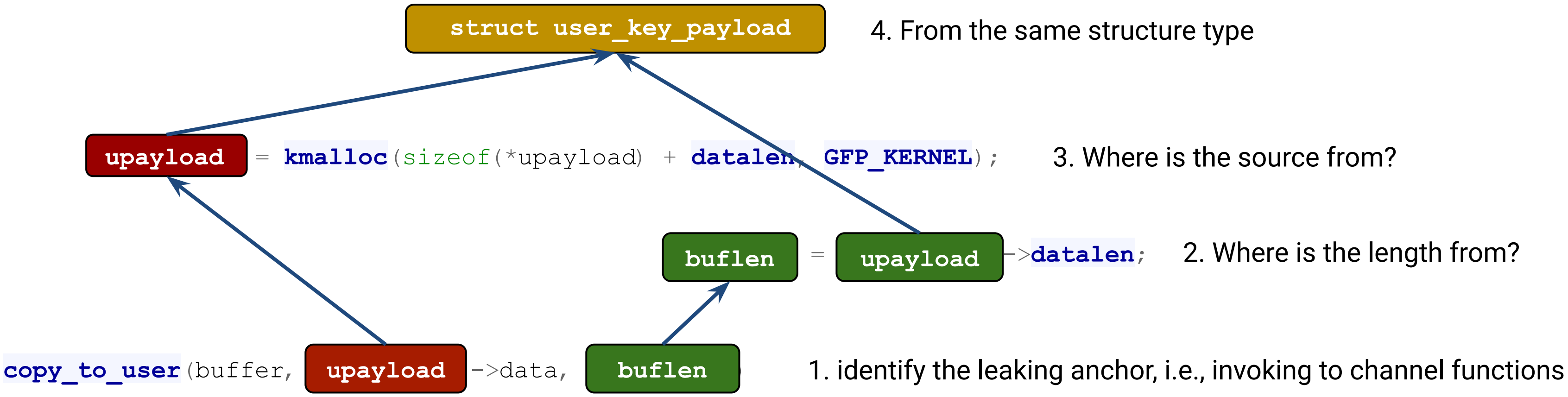
Generality of Elastic Object Attack

- Unknown
  - Functions as kernel-user space communication channel, e.g., like copy_to_user()
  - # of elastic objects in the kernel code base
  - # of elastic objects whose “p” and “n”can be propagated to those channel functions
  - Given a vulnerability, # of elastic objects can be used for leaking
- Important
  - Do we need to pay attention to this attack?
  - Do we need a mitigation?

Our Study - Kernel/User Channels Functions

Types of Channel	Function Prototypes
Memory Access APIs	unsigned long copy_to_user(void __user* to, **const void* from**, unsigned long n);
Netlink	int nla_put(struct sk_buff* skb, int attrtype, int attrlen, **const void* data**);
	int nla_put_nohdr(struct sk_buff* skb, int attrlen, **const void* data**);
	int nla_put_64bit(struct skb_buff* skb, int attrtype, int attrlen, **const void* data**, int padattr);
	void* nlmsg_data(const struct nlmsghdr* nlh); void* memcpy(void* dest, **const void* src**, size_t count);
	void* nla_data(const struct nlattr* nla); void* memcpy(void* dest, **const void* src**, size_t count);
General Networking	void* skb_put_data(struct sk_buff* skb, **const void* data**, unsigned int len);
	void* skb_put(struct sk_buff* skb, unsigned int len); void* memcpy(void* dst, **const void* src**, size_t count);

Our Study - Static Analysis to Pinpoint Elastic Objects



Our Study - Static Analysis to Pinpoint Elastic Objects

```
struct user_key_payload

upayload = kmalloc(sizeof(*upayload) + datalen, GFP_KERNEL);

buflen = upayload->datalen;

if (buflen > upayload->datalen)

copy_to_user(buffer, upayload->data, buflen)
```

5. Collect the constraints along the path from the anchor retrieving length to the anchor leaking data

Our Study - Static Analysis to Pinpoint Elastic Objects

```
[+]                  ip_options                Sample record
(1)[cache]           kmalloc_16*
(2)[len offset]      [8, 9)
(3)[ptr offset]      NA
(4)[alloc site]      net/ipv4/ip_output.c:1251
(5)[leak  anchor]    net/ipv4/ip_sockglue.c:1356
(6)[capability]      stack canary, KASLR
```

- 49 elastic structures are tracked down in defconfig
- 38 elastic structures are confirmed using kernel fuzzing and manual analysis

Our Study - Results in Linux

Linux					
Cache	**Struct**	**Offset (len/ptr)**	**Potential**	**Privilege**	**Constraints**
kmalloc-8	ipv6_opt_hdr	[1, 2)/NA	H	∅	[1, 2) < Arg , p ≠ null
kmalloc-16	sock_fprog_kern	[0, 2)/[2, 10)	H & A	NET_RAW	[0, 2) ≤ Arg
	policy_load_memory	[0, 4)/[4, 12)	H & A	∅	[0, 4) < Arg
	ldt_struct	[8, 12)/[0, 8)	H & A	∅	[8, 12) < 65536
	ip_options⋆	[8, 9)/NA	anchor1: H anchor2: S	∅	[8, 9) < Arg, anchor1 in put_cmsg() [8, 9) ≠ 0, anchor2 in do_ip_getsockopt()
	iovec	[8, 16)/[0, 8)	H & A	∅	∅
kmalloc-32	cfg80211_pkt_pattern	[16, 20)/[8, 16)	H & A	NET_ADMIN	∅
	user_key_payload⋆	[16, 18)/NA	H	∅	[16, 18) < Arg
	xfrm_replay_state_esn⋆	[0, 4)/NA	H	NET_ADMIN	∅
	ip_sf_socklist⋆	[4, 8)/NA	H	∅	[4, 8) < Arg, [4, 8) ≠ 0
	cache_reader †	[24, 28)/NA	H	∅	[0, 8) ≠ cache_detail.20
	tc_cookie	[8, 12)/[0, 8)	H & A	NET_ADMIN	[8, 12) ≠ 0
	cfg80211_bss_ies⋆	[24, 28)/NA	H	NET_ADMIN	[24, 28) ≠ 0, p ≠ null
kmalloc-64	sg_header	[4, 8)/NA	H	∅	∅
	inotify_event_info	[36, 40)/NA	H	∅	∅
	fb_cmap_user	[4, 8)/[8, 16), [16, 24), [24, 32)	S	∅	[4, 8) ≠ 0
	cache_request	[40, 44)/[32, 40)	H & A	∅	[20, 24) ≠ 0, [40, 44) ≠ 0
	msg_msg	[24, 32)/[32, 40)	H & A	∅	[24, 32) < Arg, [24, 32) ≤ 4048
	fname⋆ †	[44, 45)/NA	H	∅	[44, 45) ≤ compat_getdents_callback.3, p ≠ null, [32, 40) == null, [40, 44) < Arg
	ieee80211_mgd_auth_data⋆	[48, 52)/NA	H	∅	∅
	tcp_fastopen_context	[32, 36)/NA	S	∅	[32, 36) < Arg
kmalloc-96	request_key_auth	[48, 52)/[40, 48)	H & A	∅	[48, 52) < Arg, p ≠ null
	xfrm_algo_auth⋆	[64, 68)/NA	H	NET_ADMIN	∅
	cfg80211_wowlan_tcp	[28, 32)/[32, 40), [56, 62)/NA, [80, 84)/[84, 88)	H & A	NET_ADMIN	∅
	xfrm_algo⋆	[64, 68)/NA	H	NET_ADMIN	∅
	xfrm_algo_aead⋆	[64, 68)/NA	H	NET_ADMIN	∅
kmalloc-192	cfg80211_scan_request	[32, 36)/[24, 32)	H & A	NET_ADMIN	p ≠ null, [24, 32) ≠ null
	mon_reader_bin	[16, 20)/[24, 32)	H & A	∅	[16, 20) < 4096, [16, 20) ≠ 0, [16, 20) < Arg,
	cfg80211_sched_scan_request	[40, 44)/[32, 40)	H & A	NET_ADMIN	[8, 16) == kaddr, [48, 56) == kaddr, p ≠ null
kmalloc-256	mon_reader_text	[112, 116)/[116, 124)	H & A	∅	[112, 116) < Arg
	station_info	[120, 124)/[112, 120)	H & A	NET_ADMIN	∅
kmalloc-512	ext4_dir_entry_2⋆†	[6, 7)/NA	H	∅	[6, 7) ≤ compat_getdents_callback.3
kmalloc-1024	xfrm_policy	[372, 373)/NA	S	NET_ADMIN	∅
	fb_info	[816, 824)/[808, 816)	H & A	∅	[832, 836) == 0, [768, 776) == kaddr
kmalloc-2048	audit_rule_data⋆	[1036, 1040)/NA	S	AUDIT_CONTROL AUDIT_READ	∅
kmalloc-16384	n_tty_data	[8800, 8804)/NA	H	∅	[8800, 8804) < 4096
proc_dir_entry_cache △	proc_dir_entry †	[177, 178)/NA	H	∅	p ≠ null, [177, 178) ≤ compat_getdents_callback.3
seq_file_cache △	seq_file †	[24, 28)/NA	H	∅	[24, 28) ≠ 0, [24, 28) < Arg, [24, 28) < seq_file.1, [96, 104) == kaddr

- 36/38 structures are in general cache
- Cover most general caches

Our Study - Results in Linux

CVE-ID or Syzkaller-ID	Type	Capability	Suitable objects #	Security Impact
1379... [71]	OOB	kmalloc-512:[0, 512)=*	10 + (1)	SC, HC, BA
3d67... [68]	OOB	NA	0	NA
422a... [69]	OOB	kmalloc-64:[0, 4)=0x8	0	NA
5bb0... [76]	UAF	kmalloc-192:[16, 24)=0, kmalloc-192:[48, 56)=kaddr	1	HC, BA
6a6f... [73]	UAF	kmalloc-1024:[0, 8)=kaddr	3	HC, BA
a84d... [67]	OOB	kmalloc-32:[0, 4)=*	1	HC, BA
bf96... [74]	UAF	ip_dst_cache:[64, 68)=*	0	NA
e4be... [70]	OOB	kmalloc-64:[0, 16)=*, [16, 24)=192, [24, 64)=0	6	SC, HC, BA
e928... [72]	UAF	kmalloc-256:[120, 128)=kaddr	1	HC, BA, AR
ebeb... [75]	UAF	kmalloc-1024:[15, 24)=kaddr	1	HC, BA
2018-6555	UAF	kmalloc-96:[0, 8)=kaddr, kmalloc-96:[8, 16)=kaddr	3	SC, HC, BA
2018-5703	OOB	NA	0	NA
2018-18559	UAF	kmalloc-2048:[1328, 1336)=*	0	NA
2018-12233	OOB	NA	0	NA
2017-8890	DF	kmalloc-64:[0, 8)=kaddr:[8, 16)=kaddr:[16, 18)<46:[18, 64)=*	12 + (1)	SC, HC, BA, AR
2017-7533	OOB	kmalloc-96:[0, 11)=*:[11, 12)='\0'	2	HC, BA
2017-7308	OOB	kmalloc-1024:[0, 1024)=*, kmalloc-2048:[0, 2048)=*	12 + (1)	SC, HC, BA
2017-7184	OOB	kmalloc-32:[0, 32)=*, kmalloc-64:[0, 64)=*, kmalloc-96:[0, 96)=*, kmalloc-128:[0, 128)=* kmalloc-196:[0, 192)=*, kmalloc-256:[0, 256)=*, kmalloc-512:[0, 512)=*	22 + (2)	SC, HC, BA, AR
2017-6074	DF	kmalloc-256:[0, 8)=kaddr:[8, 16)=kaddr:[16, 18)<238:[18, 256)=*	11 + (1)	SC, HC, BA
2017-2636	DF	kmalloc-8192:[0, 8)=kaddr:[8, 16)=kaddr:[16, 18)<8174:[18, 8192)=*	10 + (1)	HC, BA
2017-17053	DF	kmalloc-16:[0, 8)=*	4	SC, HC, BA
2017-17052	UAF	kmalloc-256:[0, 8)=kaddr, kmalloc-256:[8, 16)=kaddr	3	SC, HC, BA
2017-15649	UAF	kmalloc-4096:[2160, 2168)=*	0	NA
2017-10661	UAF	kmalloc-256:[192, 200)=kaddr, kmalloc-256:[200, 208)=kaddr	0	NA
2017-1000112	OOB	NA	0	NA
2016-6187	OOB	kmalloc-8:[0, 8)=*, kmalloc-16:[0, 16)=*, kmalloc-32:[0, 32)=* kmalloc-64:[0, 64)=*, kmalloc-128:[0, 128)=*	24 + (2)	SC, HC, BA, AR
2016-4557	UAF	kmalloc-256:[56, 64)=*, kmalloc-256:[64, 72)=*	3	HC, BA
2016-10150	UAF	kmalloc-64:[24, 32)=*, kmalloc-64:[32, 40)=*	3	SC, HC, BA, AR
2016-0728	UAF	kmalloc-256:[0, 8)=*	2	HC, BA
2014-2851	UAF	kmalloc-192:[0, 8)=*	2	HC, BA
2010-2959	OOB	kmalloc-256:[0, 256)=*	11 + (1)	SC, HC, BA

- 21 CVEs, 10 from syzbot
- 23/31 bypass KASLR and leak heap cookies
- 12/31 leak stack canary
- 5/31 performs arbitrary read

Our Study - Results in FreeBSD & XNU

XNU					
Cache	Struct	Offset (len/ptr)	Potential	Privilege	Constraints
kalloc.16	user_ldt	[4, 8)/NA	H	∅	[4, 8) ≤ 8192, [4, 8) ≤ Arg
	sockaddr⋆	[0, 1)/NA	H	∅	[0, 1) ≤ 255
	accessx_descriptor⋆	[0, 4)/NA	H	∅	[0, 4) ≠ 0
kalloc.32	msg †	[16, 18)/NA	H	∅	[16, 18) < msgrcv_nocancel_args.7, [16, 18) > 0
	audit_sdev_entry	[8, 16)/[0, 8)	H & A	∅	∅

FreeBSD					
Cache	Struct	Offset (len/ptr)	Potential	Privilege	Constraints
kmem.16	TWE_Param⋆†	[3, 4)/NA	H	∅	p ≠ null, [3,4) ≤ twe_paramcommand.3
	iovec	[8, 16)/[0, 8)	H & A	∅	∅
	sockaddr	[0, 1)/NA	H	∅	∅
kmem.32	i40e_nvm_access	[12, 16)/NA	H	∅	[12, 16) > 0, [12, 16) < 4097
	vpd_readonly	[16, 20)/[8,16)	H & A	∅	∅
	vpd_writeonly	[20, 24)/[8,16)	H & A	∅	∅
kmem.64	uio	[24, 32)/NA	H	∅	[32, 36) ≠ 2
	gctl_req_arg	[28, 32)/[40, 48)	H & A	∅	[24, 28) == 32
	ips_ioctl	[20, 24)/[8, 16)	H & A	∅	∅
kmem.128	usb_symlink	[80, 82)/NA	H	∅	p ≠ null, [80, 81)+[81, 82) ≤ 252
kmem.256	ucred	[52, 56)/[176, 184)	H & A	∅	∅
	shmfd	[0, 8)/NA	H	∅	∅
	iso_node†	[56, 64)/NA	H	∅	[56, 64) ≤ uio.2
	iso_mnt†	[48, 52)/NA	H	∅	[48, 52) ≤ uio.3
kmem.512	acc_fib†	[8, 10)/NA	H	∅	[8, 10)≤ aac_softc.61
	dirent	[20, 22)/NA	H	∅	[20, 22) ≤ Arg
kmem.1024	buf	[48, 52)/[24, 32)	H & A	∅	∅
kmem.2048	fw_device	[16, 20)/NA	H	∅	p ≠ null,[16, 20) ≥ 1024
mbuf	mbuf	[24, 28)/[8, 16)	H & A	∅	∅
TMPFS node	tmpfs_node	[40, 48)/NA	H	∅	∅

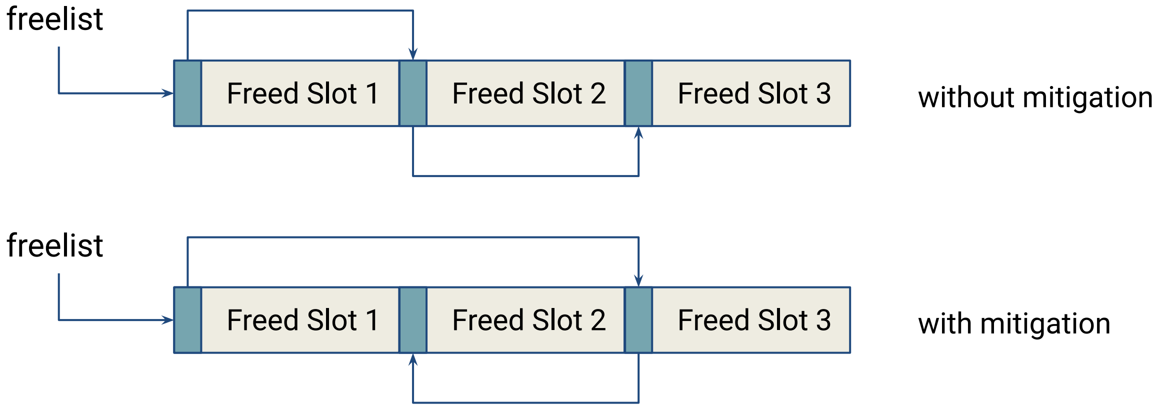
- 20 structures in FreeBSD
- 16 structures in XNU

Our Study - Results in FreeBSD & XNU

CVE-ID or Syzkaller-ID	Type	Capability	Suitable objects #	Security Impact
		FreeBSD		
2019-5603	UAF	file_zone:[40, 44)=*	0	NA
2019-5596	UAF	file_zone:[40, 44)=*	0	NA
2016-1887	OOB	zone_mbuf:[0, 256)=*	1	BA & AR [7]
		XNU		
2019-8605	UAF	kalloc.192:[0, 192)=*	4 + (1)	HC, BA, AR
2019-6225	UAF	kalloc.96:[8, 16)=*	0	NA
2018-4243	OOB	kalloc.16:[0, 8)=0	0	NA
2018-4241	OOB	kalloc.2048:[0, 2048)=*	5	HC, BA
2017-2370	OOB	kalloc.256:[0, 256)=*	3	HC, BA
2017-13861	DF	kalloc.192:[0, 192)=*	4 + (1)	HC, BA, AR

- 9 CVEs
- 5/9 bypasses KASLR
- 4/9 leaks heap cookie (FreeBSD doesn’t have heap cookie)
- 3/9 performs arbitrary read

Potential Mitigations Against Elastic Object Attack



CONFIG_SLAB_FREELIST_RANDOM (freelist randomization)

- No effects on UAF/double free exploitation
- Many bypassing techniques
  - Heap Groom
  - Freelist Reversal

Potential Mitigations Against Elastic Object Attack

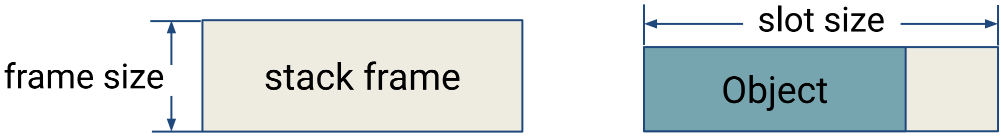
```
struct subprocess_info {
        struct work_struct work;
        struct completion *complete;
        const char *path;
        char **argv;
        char **envp;
        int wait;
        int retval;
        int (*init)(struct subprocess_info *info, struct cred *new);
        void (*cleanup)(struct subprocess_info *info);
        void *data;
} __randomize_layout;
```

CONFIG_GCC_PLUGIN_RANDSTRUCT (structure layout randomization)

- Random seed has to be exposed for building third-party kernel modules

Potential Mitigations Against Elastic Object Attack

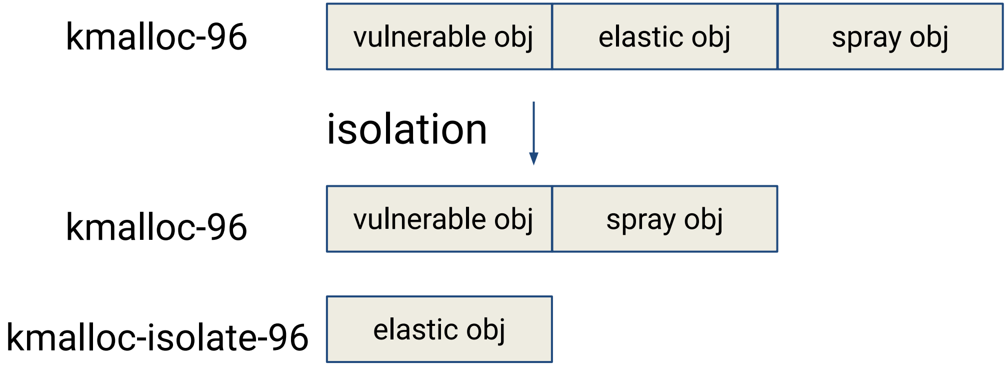
copy_to_user(dst, src, n)



CONFIG_HARDENED_USERCOPY

- n <= frame size; n <= slot size
- Miss other channel functions
- Not restrict enough, sensitive data can be in the slot and stack frame

Our Proposed Mitigation Against Elastic Object Attack



- Create kmalloc-isolate-xxx during boot up
- Add one more flag to specify which cache for allocation
- More advanced isolation is in Grsecurity’s AUTOSLAB (which I know later)

Performance Evaluation of Our Proposed Mitigation

Benchmark	w/o defense	w/ defense	Overhead
LMbench - latency (ms)			
syscall()	0.3813	0.3796	-0.46%
open()/close()	1.5282	1.5290	0.05%
read()	0.4596	0.4529	-0.94%
write()	0.4125	0.4127	0.05%
select() (10 fds)	0.5114	0.5043	-1.39%
select() (100 fds)	1.1805	1.1774	-0.26%
stat()	0.7590	0.7600	0.14%
fstat()	0.4576	0.4584	0.19%
fork() + exit()	90.37	91.71	1.46%
fork() + execve()	255.18	257.85	1.05%
fork() + /bin/sh	858.86	863.77	0.57%
sigaction()	0.4182	0.4192	0.25%
Signal delivery	0.9337	0.9309	-0.30%
Protection fault	0.6914	0.7093	2.58%
Pipe I/O	3.7497	3.7951	1.87%
UNIX socket I/O	5.9786	5.882	-1.62%
TCP socket I/O	9.7846	9.6776	-1.09%
UDP socket I/O	6.5358	6.2251	-4.75%
LMbench - throughput (MB/s)			
Pipe I/O	4755.49	4753.89	0.03%
UNIX socket I/O	10385.07	10307.40	0.75%
TCP socket I/O	6327.32	6725.17	-6.29%
mmap() I/O	13559.20	13511.95	0.35%
File I/O	7707.81	7702.82	0.06%

Benchmark	w/o defense	w/ defense	Overhead
Phoronix - latency (s)			
FFmpeg	14.01	14.46	3.22%
GnuPG	17.39	17.35	-0.22%
Phoronix - throughput			
Apache (request/s)	16700.23	16088.00	3.67%
OpenSSL (signs/s)	272.00	272.00	0
7-Zip (MIPS)	9970.00	9374.00	5.98%
Customized bench - latency (ms)			
sock_fprog_kern	28.54	28.30	0.09%
ldt_struct	33.81	31.48	-2.52%
ip_options	29.29	30.67	2.40%
user_key_payload	34.04	35.33	-2.87%
xfrm_replay_state_esn	29.69	30.06	1.67%
ip_sf_socklist	29.13	28.05	-3.78%
sg_header	31.84	30.75	-2.99%
inotify_event_info	32.68	31.77	0.42%
msg_msg	27.75	26.83	0.66%
tcp_fastopen_context	28.79	28.65	-1.04%
request_key_auth	81.23	79.88	2.98%
xfrm_algo_auth	30.32	29.50	-0.28%
xfrm_algo	28.64	28.43	-0.11%
xfrm_algo_aead	31.36	31.39	0.13%
xfrm_policy	31.07	30.53	-1.43%
Average			0.19%

Security Evaluation of Our Proposed Mitigation

- Out of 31 vulnerabilities used to study the generality of elastic object attack
- Only two vulnerabilities can potentially be exploited after the mitigation enforced
  - CVE-2017-7184: vulnerable object is xfrm_replay_state_esn which is also the elastic object
  - CVE-2017-17053: vulnerable object is ldt_struct which is also the elastic object
- Still raise the bar because
  - kernel objects enclosing a function pointer are almost in general cache

Other Mitigation Designs

- Shadow memory for each elastic object
  - Record the actual size of the corresponding object
  - Heavy memory and performance overhead
- Introduce a checksum field for integrity check
  - Encrypt the length value and store it in the checksum field
  - Usability is an issue when elastic object is designed for protocols having specific formats

Contributions

- Extension of flexible cobject to elastic object
- A systematic method demonstrating the generality of elastic object attack
- A defense design that could mitigate the attack

Thank You!

Contact

Twitter: @Lewis_Chen_
Email: ychen@ist.psu.edu
Personal Page: http://www.personal.psu.edu/yxc431/


LINUX
SECURITY
SUMMIT
EUROPE