A Systematic Study of Elastic Objects in Kernel Exploitation

Yueqi Chen, Zhenpeng Lin, Xinyu Xing
The Pennsylvania State University

ACM CCS 2020
Nov 9th
Kernel Wars

- A lot of exploit mitigations (e.g. KASLR, stack canary, heap cookies...)
- A lot of exploitation methods to circumvent kernel mitigations
- One of commonly known methods is to utilize elastic kernel objects to bypass mitigations
Elastic Objects

- Contain a length field
- The length field indicates the size of an elastic kernel buffer
CVE-2017-7184 & Exploit

kmalloc-256

Vul Obj bmp_len f_op

bmp

xfrm_replay_state_esn
f_op
ext4_file_operations

bmp_len
CVE-2017-7184 & Exploit

kmalloc-256

Vul Obj  bmp_len  bmp  f_op

kmalloc-256

Vul Obj  bmp_len  bmp  f_op

overwrite

xfrm_replay_state_esn  f_op  ext4_file_operations
CVE-2017-7184 & Exploit

kmalloc-256

Vul Obj   bmp_len   bmp   f_op

overwrite

kmalloc-256

Vul Obj   bmp_len   bmp   f_op

overread
Conditions of Elastic Object Attack

- The same cache

- The length field can be enlarged by the vulnerability

- Existing a channel to leak the elastic buffer to the userland
Severity and Generality of Elastic Object Attack

- Severity is obvious
  - Leaking kernel information from an overwrite primitive

- Generality is unknown
  - Pervasive object
  - Exploiting different vulnerabilities

Do we have the need to build defense?
Struct Analysis

```c
struct user_key_payload
{
    buflen = upayload->datalen;
}
```

1. identify the leaking site
2. backward taint the length
3. find the elastic object
Static Analysis

1. identify the leaking site
2. backward taint the length
3. find the elastic object
4. identify the allocation site
5. find the elastic object

\[ \text{upayload} = \text{kmalloc} (\text{sizeof}(*\text{upayload}) + \text{datalen}, \text{GFP_KERNEL}); \]

\[ \text{bufflen} = \text{upayload} -> \text{datalen}; \]

\[ \text{copy_to_user} (\text{buffer, upayload} -> \text{data, bufflen}); \]
Static Analysis

1. Identify the leaking site
2. Backward taint the length
3. Find the elastic object
4. Identify the allocation site
5. Find the elastic object

```c
struct user_key_payload

buflen = upayload->datalen;

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

copy_to_user(buffer, upayload->data, buflen)
```
Experiment Setup and Results

- Select 3 commonly used open-sourced OSes
- Identify 38 structures in Linux, 16 structures in XNU, and 20 structures in FreeBSD
Experiment Setup and Results

- Select 3 commonly used open-sourced OSes
- Identify 38 structures in Linux, 16 structures in XNU, and 20 structures in FreeBSD
- Cover most of general caches/zones
- 18/74 structures are general cache/zone-flexible kernel structures
Effectiveness in Bypassing Mitigation

- 27/40 vulnerabilities are able to bypass not only KASLR but also heap cookies
- 12/40 vulnerabilities are able to uncover stack canary
- 8/40 vulnerabilities are able to exhibit the capability of performing arbitrary kernel read.

Elastic objects could nearly always facilitate a kernel vulnerability to bypass exploitation mitigation
Defense

- Key idea: Isolating elastic objects into individual shadow caches/zones
Defense

- Key idea: Isolating elastic objects into individual shadow caches/zones
Defense Evaluation

- **Performance overhead**
  - The average performance drop is 0.19% on LMBench, Phoronix and our customized benchmark

- **Security improvement**
  - 29/31 vulnerabilities find no suitable elastic object
  - CVE-2017-7184, CVE-2017-17053: vulnerable objects are also elastic objects
Summary

- A systematic approach to finding out the elastic kernel objects
- An evaluation of the effectiveness of utilizing elastic kernel objects on 40 kernel vulnerabilities across three OSes
- A new defense mechanism to mitigate the threat of elastic kernel objects
- An evaluation of the defense mechanism in terms of performance overhead and security improvement
Thank You!

Code & Data
https://github.com/chenyueqi/w2l

Contact
Twitter: @Markakkkk
Email: zplin@psu.edu
Personal Page: https://zplin.me
Before isolating

After isolating