How Double-Fetch Situations turn into Double-Fetch Vulnerabilities:

A Study of Double Fetches in the Linux Kernel

Pengfei Wang, Jens Krinke, Kai Lu, Gen Li, Steve Dodier-Lazaro

College of Computer
National University of Defense Technology, China
Centre for Research on Evolution, Search and Testing
University College London
OUTLINE

• What is a double fetch?

• A static pattern-based double fetch analysis.

• Results and Findings.
Microsoft Security Bulletin MS08-061 – Important

Vulnerabilities in Windows Kernel Could Allow Elevation of Privilege (954211)
Published: October 14, 2008
Version: 1.0

General Information
Executive Summary
This security update resolves one publicly disclosed and two privately reported vulnerabilities in the Windows kernel. A local attacker who successfully exploited these vulnerabilities could take complete control of an affected system. The vulnerabilities could not be exploited remotely or by anonymous users.

This security update is rated Important for all supported editions of Microsoft Windows 2000, Windows XP, Windows Server 2003, Windows Vista, and Windows Server 2008. For more information, see the subsection, Affected and Non-Affected Software, in this section.

The security update addresses the vulnerabilities by correcting window property validation passed during the new window creation process, correcting the manner in which system calls from multiple threads are handled, and correcting validation of parameters passed to the Windows Kernel from user mode. For more information about the vulnerabilities, see the Frequently Asked Questions (FAQ) subsection for the specific vulnerability entry under the next section, Vulnerability Information.

Recommendation. Microsoft recommends that customers apply the update at the earliest opportunity.

Known Issues. Microsoft Knowledge Base Article 954211 documents the currently known issues that customers may experience when installing this security update. The article also documents recommended solutions for these issues.

Affected and Non-Affected Software
The following software have been tested to determine which versions or editions are affected. Other versions or editions are either past their support life cycle or are not affected. To determine the support life cycle for your software version or edition, visit Microsoft Support Lifecycle.
Double Fetch
First Study: Jurczyk & Coldwind - 2013

Stats: bochspwn vs Windows

- **89 potential** new issues discovered
  - + part of the initial 27 bugs were also rediscovered
  - All were reported to Microsoft (Nov 2012 - Jan 2013)
- **36 EoPs (+3 variants)** addressed by: MS13-016, MS13-017, MS13-031, MS13-036
- **13 issues** have been classified as **Local DoS only**
- **7 more** are being analyzed / are scheduled to be fixed
- The rest were unexploitable / non-issues / etc

Double Fetch
Exploit Instructions on GitHub - 2016

Double-Fetch race condition vulnerability demonstration

- The Windows kernel-mode driver (rcdriver) is vulnerable.
- The user-mode exploit (exploit) takes advantage of the vulnerability to elevate its process’s privileges.
- Link to article: http://resources.infosecinstitute.com/exploiting-windows-drivers-double-fetch-race-condition-vulnerability/
- Link to blog: http://rce4fun.blogspot.com/
Double Fetch Vulnerabilities Today: Where are they?

<table>
<thead>
<tr>
<th>Bochspwn</th>
<th>Is dynamic, slow and limited code coverage.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Weakness</td>
<td>Did not show why double fetches happen.</td>
</tr>
<tr>
<td></td>
<td>Only workable for Windows.</td>
</tr>
<tr>
<td></td>
<td>Cannot analyze driver code without hardware.</td>
</tr>
</tbody>
</table>

- Linux had double fetch vulnerabilities, but no dedicated audit has been done.

- We need a static analysis to cover the complete kernel including all drivers (44%).
Operating Systems: Separate Address Spaces

- Each user process has its own virtual memory space
- User spaces are isolated.
- Only the kernel can access all user spaces.
Operating Systems: System Call Interface

Kernel

Syscall

User Space

• Fundamental Interface between application and kernel

• Arguments are copied
  • either directly or
  • as pointers to data structures

• The kernel cannot trust any data coming from the application!
Anatomy of a Double Fetch
Transfer Functions in Linux

• Linux uses dedicated functions to copy data between user and kernel space:

  get_user(src)
  copy_from_user(dst, src, size)
  put_user(dst)
  copy_to_user(dst, src, size)

• Data in user space is not accessed directly: Ensures that the access is valid.
Double-fetch bug in Linux (CVE-2016-5728)

```c
522 static int mic_copy_dp_entry(...) {
    ...
533     if (copy_from_user(&dd, argp, sizeof(dd))) {
536         return -EFAULT;
537     } else {
546         dd_config = kmalloc(mic_desc_size(&dd), GFP_KERNEL);
547         if (dd_config == NULL) {
552             return -ENOMEM;
553             ...
557         }
560         for (i = sizeof(struct mic_bootparam);
561             i < MIC_DP_SIZE - mic_total_desc_size(dd_config);
562             i += mic_total_desc_size(devp)) {
565             devp = mdev->dp + i;
568             ...
571         } else {
575             ret = -EFAULT;
578             ...
581         }
586     }
591     memcpyp(devp, dd_config, mic_desc_size(dd_config));
597 }
```

Allocate buffer use ‘size’ from first fetch

Use ‘size’ from second fetch
Static Pattern-Based Approach
Pattern-based Double Fetch Analysis

Based on Coccinelle (Julia Lawall, LIP6 – France)
  Program matching and transformation engine used for Linux checking

Developed two analyses:
1. A simple analysis to identify double-fetch situations
2. A refined analysis to discover double-fetch bugs
Pattern-based Double Fetch Analysis

### Phase 1: Basic Pattern

```c
void function_name(*src) {
    copy_from_user(dst1, src, len1)
    ...
    copy_from_user(dst2, src, len2)
}
```

### Phase 2: Refined Pattern

- **Rule 0**: Basic pattern
- **Rule 1**: No pointer change
- **Rule 2**: Pointer aliasing
- **Rule 3**: Explicit type conversion
- **Rule 4**: Combination of element fetch and pointer fetch
- **Rule 5**: Loop involvement

### Candidate Files

### Manual Analysis

<table>
<thead>
<tr>
<th>Trigger &amp; Consequence</th>
<th>Context Information</th>
<th>Bug Details</th>
<th>Categorization</th>
</tr>
</thead>
</table>
Manual Analysis

**Characteristics**
- How user data is transferred and used in the kernel
- Trigger and consequence

**Details at C code level**
- Context information
- Implementation details
- Add rules to refine the pattern

**Categorization**
- Size Checking
- Type Selection
- Shallow Copy
Categorization – Size Checking, Type Selection

```c
struct header(*ptr)
{
    unsigned int size;
    unsigned type;
    ...
} hdr;

User Msg

Header content

*ptr

copy_from_user(hdr, ptr, sizeof(header));
...
buf = kalloc(hdr.size)
...
copy_from_user(buf, ptr, hdr.size);
...

copy_from_user(hdr, ptr, sizeof(header));

switch(hdr.type){
    case 1:
        copy_from_user()
        ...
    case 2:
        copy_from_user()
        ...
    default:
        ...
}
```

Size Checking

Type Selection
Categorization – Shallow Copy

User Msg

*ptr

int | unsigned int len | char * m

First Buffer

Copy from user again to get element

msg

Second Buffer

msg'

Msg is shallow copied to kernel
Refined Double Fetch Bug Detection

### Basic rule
- `trans_func(dst1, src)`
- `...`
- `trans_func(dst2, src)`

### Pointer aliasing
- `trans_func(dst1, src)`
- `p = src`
- `...`
- `trans_func(dst2, p)`

### Pointer & element
- `trans_func(dst1, ptr->len)`
- `...`
- `trans_func(dst2, ptr)`

### Loop involvement
- `trans_func(dst1, msg.len)`
- `...`
- `trans_func(dst2, &msg)`

### No pointer change
- `trans_func(dst1, src)`
- `when != src = src + offset`
- `...`
- `trans_func(dst2, src)`

### Explicit type conversion
- `trans_func(dst1, (T1)src)`
- `...`
- `trans_func(dst2, (T2)src)`

### Source Code Files

---

**Refined Coccinelle-Based Double-Fetch Bug Detection**

A double-fetch bug has been eliminated in Linux 4.6. Attempting to trigger the bug will simply end in termination.

### Scenario 1: Alias
- When two reads fetch data from the same source location, we added the following five additional rules for more refined double-fetch bug detection.

### Scenario 2: Element fetch
- In the second fetch, the message length is fetched by accessing the data structure.

### Scenario 3: Type conversion
- The right side of the transfer function arguments, but they cover the same scenario.

### Scenario 4: Loop
- The second fetch of the last iteration and the first fetch of the next iteration will be matched.

### Scenario 5: Pointer type conversion
- Any of the two source pointers could involve type conversion.

### False Positives
- A double-fetch bug rule should be removed as a false positive because the user access the address of the data structure.

### False Negatives
- A double-fetch bug rule could cause false negatives. Besides, as can be seen from Rule 3 in Figure 9, any of the two source pointers could involve type conversion. Missing aliasing situation could cause false negatives.
Results and Findings
Evaluation - Basic Double Fetch Analysis

<table>
<thead>
<tr>
<th>Category</th>
<th>Occurrences</th>
<th>In Drivers</th>
</tr>
</thead>
<tbody>
<tr>
<td>Size Checking</td>
<td>30</td>
<td>22</td>
</tr>
<tr>
<td></td>
<td>33%</td>
<td>73%</td>
</tr>
<tr>
<td>Type Selection</td>
<td>11</td>
<td>9</td>
</tr>
<tr>
<td></td>
<td>12%</td>
<td>82%</td>
</tr>
<tr>
<td>Shallow Copy</td>
<td>31</td>
<td>19</td>
</tr>
<tr>
<td></td>
<td>34%</td>
<td>61%</td>
</tr>
<tr>
<td>Other</td>
<td>18</td>
<td>7</td>
</tr>
<tr>
<td></td>
<td>20%</td>
<td>39%</td>
</tr>
<tr>
<td>Total</td>
<td>90</td>
<td>57</td>
</tr>
<tr>
<td></td>
<td>100%</td>
<td>63%</td>
</tr>
<tr>
<td>True Bugs</td>
<td>5</td>
<td>4</td>
</tr>
<tr>
<td></td>
<td>6%</td>
<td>80%</td>
</tr>
</tbody>
</table>

- Most **double fetches** don’t cause **double-fetch bugs**.
- Double fetches are more likely to **occur in drivers**.
  - About 63% (57 out of 90) of the cases were driver related.
  - About 80% (4 out of 5) of the true double-fetch bugs inside drivers.
Evaluation – Refined Detection

<table>
<thead>
<tr>
<th>Kernel</th>
<th>Total Files</th>
<th>Reported Files</th>
<th>True Bugs</th>
<th>Size Check.</th>
<th>Type Sel.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Linux 4.5</td>
<td>39,906</td>
<td>53</td>
<td>5</td>
<td>23</td>
<td>6</td>
</tr>
<tr>
<td>Android 6.0.1</td>
<td>35,313</td>
<td>48</td>
<td>3</td>
<td>18</td>
<td>6</td>
</tr>
<tr>
<td>FreeBSD</td>
<td>32,830</td>
<td>16</td>
<td>0</td>
<td>8</td>
<td>3</td>
</tr>
</tbody>
</table>

- **Totally 6 bugs found:**
  - 5 new bugs in newest Linux kernel 4.5.
  - 2 shared between Android and Linux.
  - 1 bug only showed in Android.
  - No bug found in FreeBSD.
# The Confirmed Bugs

| CVE-2016-5728 | MIC VOP (Virtio Over PCIe) driver  
|                | Linux-4.5/drivers/misc/mic/host/mic_virtio.c |
| CVE-2016-6130 | IBM (z-Series) s390 platform driver  
|                | Linux-4.5/drivers/s390/char/sclp_ctl.c |
| CVE-2016-6136 | Auditing subsystem  
|                | Linux-4.5/kernel/auditsc.c |
| CVE-2016-6156 | Expose the Chrome OS Embedded Controller to user-space  
|                | Linux-4.5/drivers/platform/chrome/cros_ec_dev.c |
| CVE-2016-6480 | The aacraid driver (adds support for AdaptecRAID controllers)  
|                | Linux-4.5/drivers/scsi/aacraid/commctrl.c |
| CVE-2015-1420 | File system  
|                | Android-6.0.1/fs/fhandle.c |
Findings

Double fetches have a long history
- Windows, Linux, Android, FreeBSD
- Some double-fetch bugs existed over 10 years (CVE-2016-6480).

Some double fetches are inevitable
- Size checking, type selection, shallow copy
- Size checking is more likely to cause true bugs (5/6)

Benign double fetches are not all safe
- Can turn into harmful ones by code update (CVE-2016-5728).
- Can cause performance issue.
Conclusion

• Double fetches occur in operating systems and can cause bugs and vulnerabilities.

• With a static pattern-matching analysis, we analyzed the complete kernel (all drivers) and categorized bug prone scenarios.

• We found 6 true bugs (vulnerabilities), all have been confirmed by the maintainers and patched already.

Pengfei Wang E-mail: pfwang@nudt.edu.cn
National University of Defense Technology, China

Jens Krinke E-mail: j.krinke@ucl.ac.uk
University College London, UK