Finding User/Kernel Pointer Bugs with Type Inference

13th USENIX Security Symposium, 2004
Rob Johnson, David Wagner

Presented by
Kashyap Thimmaraju
ECS 289C, Winter 2015
Agenda

● Introduction
● Motivation
● Using CQUAL to find user/kernel pointer bugs
● Experimental results
● Conclusion
Introduction

- Type System, Type Checker and Type Inference
- 2 Principles of Type Theory
  - Progress
  - Preservation
- Type Inference is a misnomer, should be Type Reconstruction
User/kernel pointer bugs

- OS kernel cannot trust user arguments in a syscall
- Many syscalls take pointers to user buffers
  - read, write, ioctl
  - 126 more in Linux 2.4.20
- Kernel needs to check user pointer before dereferencing else
  - Attacker can corrupt kernel data structures
  - Attacker can read kernel data structures
  - Kernel OOPS
Motivation

- User/kernel pointer bugs are very dangerous!
- Good practices not enough
- Automatic program verification?
- Monolithic kernels and microkernels
- Who cares about soundness? We do!
Toy example

```c
int x;

void sys_setint(int *p)
{
    memcpy(&x, p, sizeof(x)); // BAD!
}

void sys_getint(int *p)
{
    memcpy(p, &x, sizeof(x)); // BAD!
}

...

sys_getint(buf);
sys_setint(buf);
```
Safe Toy Example

```c
int x;

void sys_setint(int *p)
{
    copy_from_user(&x, p, sizeof(x)); // GOOD!
}

void sys_getint(int *p)
{
    copy_to_user(p, &x, sizeof(x)); // GOOD!
}

...

sys_getint(buf);
sys_setint(buf);
```
Add qualifiers!

```c
int copy_from_user(void * kernel to, void * user from, int len);
int memcpy(void * kernel to, void * kernel from, int len);
int x;
void sys_setint(int * user p)
{
    copy_from_user(&x, p, sizeof(x));
}
void sys_getint(int * user p)
{
    memcpy(p, &x, sizeof(x));
}
```
Modify the C type system
Type checker
Nominal type checker, not structural!
Creates a constraint graph
Valid path from user to kernel
Type Qualifiers

- Qualifiers make basic types granular
- Qualifiers must match assignments
- “$kernel int” is a qualified type

```c
$kernel int k1, k2;
$user int u1;

k1 = k2;  //OK
k2 = u1    //ERROR
```
Type Qualifier Inference

- Reduce annotation burden
- Derive missing annotations from context

```c
$kernel    int k;
           int x, y;
$user      int u;

y = k;
u = x;
```
Type Qualifier Inference

\[
\begin{align*}
\text{$kernel$} & \quad \text{int } k; \quad Q_k = \text{$kernel$} \\
\text{int } x, y; \quad Q_{x,y} = \text{?} \\
\text{$user$} & \quad \text{int } u; \quad Q_u = \text{$user$} \\
y = k; \\
u = x;
\end{align*}
\]
Type Qualifier Inference

\[ \text{kernel} \quad \text{int} \quad k; \quad Q_k = \text{kernel} \]
\[ \text{int} \quad x, \quad y; \quad Q_{x,y} = ? \]
\[ \text{user} \quad \text{int} \quad u; \quad Q_u = \text{user} \]

\[ y = k; \quad Q_k = Q_y \]
\[ u = x; \quad Q_x = Q_u \]

Exactly one solution

\[ \text{kernel} = Q_k = Q_y \quad Q_x = Q_u = \text{user} \]
What about a violation?

$kernel
int k;
int y;

$user
int u;

y = u;
k = y;
What about a violation?

\[$\text{kernel}\quad \text{int } k; \quad Q_k = \text{\$kernel}\]
\[\text{int } y; \quad Q_y = ?\]
\[$\text{user}\quad \text{int } u; \quad Q_u = \text{\$user}\]

\[y = u;\]
\[k = y;\]

\[\text{user} = Q_u = Q_y = Q_k = \text{\$kernel}\]

But, by definition \text{\$user} \neq \text{\$kernel}, hence \text{NO SOLUTION}!!!
Experimental Setup

- Annotated Linux kernel
  - Syscalls
  - User-pointer access functions
  - Common inline assembly functions
  - Dereference operator
  - ~300 annotations

- Two kernel configurations
  - Full: all drivers and features enabled
  - Default: core kernel, only few drivers

- Used CQUAL in 2 modes
  - Bug-finding mode: unsound, but interactive
  - Verification mode: sound but batch processing
Assumptions and Refinements

- No buffer overflows
- Union-safety
- Annotations are complete
- Ignores inline assembly

- Context-sensitivity
- Field-sensitivity
- Well-formedness constraints
- Sound and precise Pointer/Integer casts
### Experimental Results

- Found 17 different security vulnerabilities
- Found bugs missed by other auditing tools
- Found bugs missed by manual audits
- All but one bug confirmed exploitable
- Verification-90 minutes and 10GB RAM on an 800 MHz Itanium.

<table>
<thead>
<tr>
<th>Version</th>
<th>Configuration</th>
<th>Mode</th>
<th>Raw Warnings</th>
<th>Unique Warnings</th>
<th>Exploitable Bugs</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.4.20</td>
<td>Full</td>
<td>Bug finding</td>
<td>512</td>
<td>275</td>
<td>7</td>
</tr>
<tr>
<td>2.4.23</td>
<td>Full</td>
<td>Bug finding</td>
<td>571</td>
<td>264</td>
<td>6</td>
</tr>
<tr>
<td></td>
<td>Default</td>
<td>Bug finding</td>
<td>171</td>
<td>76</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>Default</td>
<td>Verification</td>
<td>227</td>
<td>53</td>
<td>4</td>
</tr>
</tbody>
</table>
# False Positives

<table>
<thead>
<tr>
<th>Source</th>
<th>Frequency</th>
<th>Useful</th>
<th>Fix</th>
</tr>
</thead>
<tbody>
<tr>
<td>User flag</td>
<td>50</td>
<td>Maybe</td>
<td>Pass two pointers instead of from user flag</td>
</tr>
<tr>
<td>Address of array</td>
<td>24</td>
<td>Yes</td>
<td>Don’t take address of arrays</td>
</tr>
<tr>
<td>Non-subtyping</td>
<td>20</td>
<td>No</td>
<td>Enable subtyping</td>
</tr>
<tr>
<td>C type misuse</td>
<td>19</td>
<td>Yes</td>
<td>Declare explicit, detailed types</td>
</tr>
<tr>
<td>Field unification</td>
<td>18</td>
<td>No</td>
<td>None</td>
</tr>
<tr>
<td>Field update</td>
<td>15</td>
<td>No</td>
<td>None</td>
</tr>
<tr>
<td>Open structure</td>
<td>5</td>
<td>Yes</td>
<td>Use C99 open structure support</td>
</tr>
<tr>
<td>Temporary variable</td>
<td>4</td>
<td>Yes</td>
<td>Don’t re-use temporary variables</td>
</tr>
<tr>
<td>User-kernel assignment</td>
<td>3</td>
<td>Yes</td>
<td>Set user pointers to NULL instead</td>
</tr>
<tr>
<td>Device buffer access</td>
<td>2</td>
<td>Maybe</td>
<td>None</td>
</tr>
<tr>
<td>FS Tricks</td>
<td>2</td>
<td>Maybe</td>
<td>None</td>
</tr>
</tbody>
</table>
Related Work

- CQUAL [Foster]
- Percent-S [Shankar]
- MECA [Engler]
- sparse [Torvalds]
- Model checkers
  - MOPS [Chen]
  - SLAM [Ball]
  - BLAST [Henzinger]
- Lexical tools: LCLint, ITS-4, RATS, etc.
Conclusion

● Type qualifier inference
  ○ Id serious security bugs
  ○ Scalable solution
  ○ Closer to automatic program verification
Discussion

References


Uncovering the unknown - [https://www.youtube.com/watch?v=fDTt_uo0F-g](https://www.youtube.com/watch?v=fDTt_uo0F-g)

Rob and David presentation slides - [http://courses.cs.washington.edu/courses/cse503/10wi/lectures/user_kernel_slides.pdf](http://courses.cs.washington.edu/courses/cse503/10wi/lectures/user_kernel_slides.pdf)