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

```
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

```
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!

```
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));
 }
```

CQUAL

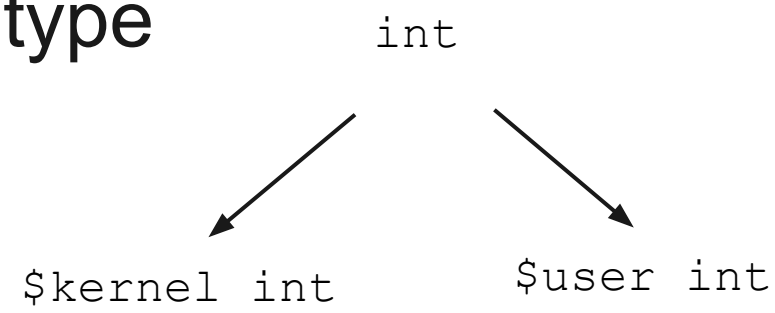
- Modify the C type system
- Type checker
- Nominal type checker, not structural!
- Creates a constraint graph
- Valid path from user to kernel

Type Qualifiers

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

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

k1 = k2;  //OK
k2 = u1   //ERROR
```



Type Qualifier Inference

- Reduce annotation burden
- Derive missing annotations from context

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

y = k;
u = x;
```

Type Qualifier Inference

```
$kernel   int k;        Q_k   = $kernel
          int x, y;     Q_x,y = ?
$user     int u;        Q_u   = $user

y = k;
u = x;
```

Type Qualifier Inference

```
$kernel   int k;        Q_k   = $kernel
          int x, y;     Q_x,y = ?
$user     int u;        Q_u   = $user

y = k;                  Q_k   = Q_y
u = x;                  Q_x   = Q_u
```

Exactly one solution

$\$kernel = Q_k = Q_y \quad Q_x = Q_u = \$user$

What about a violation?

```
$kernel   int k;
          int y;
$user     int u;

y = u;
k = y;
```

What about a violation?

```
$kernel    int k;
           int y;
$user      int u;

y = u;
k = y;
```

Q_k = $kernel
Q_y = ?
Q_u = $user

Q_u = Q_y
Q_y = Q_k

$user = Q_u = Q_y = Q_k = $kernel

But, by definition $user != $kernel, hence **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

Version	Configuration	Mode	Raw Warnings	Unique Warnings	Exploitable Bugs
2.4.20	Full	Bug finding	512	275	7
2.4.23	Full	Bug finding	571	264	6
2.4.23	Default	Bug finding	171	76	1
2.4.23	Default	Verification	227	53	4

- 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.

False Positives

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

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

CQUAL-http://www.cs.umd.edu/~jfoster/cqual/

Uncovering the unknown - 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