Academic tools and real-life bug finding in Win32

22nd Chaos Communication Congress
Rafal Wojtczuk
nergal@openwall.com

About the speaker

- 8 years of computer security research
- discoveries of privilege escalation bugs in Linux, OpenBSD, Windows2000
- an article in phrack58 about advanced return-into-libc exploits
- currently PhD student at Warsaw University

Lecture plan

- theorem provers
- ... and what practical we can do with them
- two powerful academic tools: UQBT and CBMC
- the code I am working on
- preliminary results – example of automatic finding of exploitable integer overflow in NWWKS.DLL
  - vulnerability addressed by MS05-046

Theorem provers

- computer programs capable of proving certain range of mathematical theorems
  - what theorems ?
- Examples: PVS, Coq, CVC lite, Simplify...
- automatic/interactive
- commercial use: mostly integrated circuit design and verification
- fact: computer programs behavior can be described by mathematical constructions

Model checking

- Algorithmical verification of a formal system
- works on a model constructed so that it mirrors the important properties of a system
- builds the state space
  - and usually tries to explore it all
- state explosion

Theoretical limits

- Halting problem: for a program X and its input, find out whether this program loops infinitely
- if X is a Turing machine, this problem is undecidable (unsolvable)
- first-order theorem prover may loop in proof calculations for an invalid statement
- even for undecidable problems, reasonable heuristics should work most of the time

the problem with loops

- loops are the cause of undecidability
  - sequential programs are easy to prove
- proving loops properties usually requires constructing a “loop invariant”
- example of simple loop proving with Caduceus

“verification” vs “checking”

- verification: proving all properties of a program as a whole (full program correctness)
  - usually requires program specification
  - expensive
  - example: specification of JavaCard API in JML and JavaCard applets verification
  - idea: proof-carrying code
- checking: proving particular, usually local properties of a program
  - similar to a typical security audit

"checking" approach example

- work by Dawson Engler and others on Linux 2.5.x kernel usage of user-mode pointers

```
/* linux-2.5.63/fs/quota.c */
asmlinkage long sys quotactl(unsigned int cmd,
const char *special, qid t id, caddr t addr) {
bdev = lookup bdev(special);
}
/* linux-2.5.63/fs/block dev.c */
struct block device *lookup bdev(const char *path) {
if (!path || !*path)
return ERRPTR(EINVAL);}
```

Focus: integer overflows

- dangerous when happen in buffer size calculation
- quite proliferated today
  - in MS05-53 patch for GDI32.DLL, over 50 integer overflow checks were added
- example
- well suited to be searched for automatically
- C Bounded Model Checker example

CBMC features

- arithmetics with bounded integers
- pointer arithmetics
- bitwise operations
- arbitrary “struct” and “union” usage
- function pointers
- bounded loop unwinding
- nondeterminism
- arrays (with bounds checking)

CBMC limitations

- no source code :(
- not possible to specify loop invariants
  - ... but we don't want to
- insists that all pointers must point to something allocated
- inefficient arrays implementation
- sometimes requires large amount of memory

Focus: MS Windows

- very popular
- known to be buggy – lowest TC0 (not TCO)
- often the weakest point in the network
- debugging symbols are available

Assembly semantics

- single instruction decoding is not a problem
  - ... although convenient representation is nontrivial
- need to recover higher level constructions (C/C++)
  - decompilation
- no good decompiler available publicly ?
- IDA has numerous disadvantages
  - lack of documentation (besides the include files)
  - not open source
  - probably too close to the assembly, insufficient abstraction

UQBT

- University of Queensland Binary Translator
- goal: translation of a binary for platform A to a binary for a different platform B
  - requires decompilation
- architectures: i386, sparc, hppa, mc68k
- binary file formats: ELF, DOS .exe, PE (almost)
- convenient instruction representation
- capable of emitting C code from assembly
- code with BSD license; excellent documentation

UQBT problems (with win32 files)

- supports only “cdecl” calling convention
- does not recognize library functions as such
- assumes fixed function prologue/epilogue patterns
- does not recognize pseudofunctions used in SEH implementation
- eax/ax relation handling incompatible with CBMC
- incorrect generation of “for” and “while” loops
- does not recognize VC6 inlined memcpy(), strlen() etc

UQBTng

- I fixed or worked around the above problems
- added checks for integer overflow in buffer size allocation
  - lea eax, [edi+2*ecx+0x10]; LocalAlloc(heap,eax)
  - checks for overflow in addition, multiplication, 32bit to 16bit conversions
- removed all loops
  - required for CBMC
  - should not introduce much false positives/negatives

UQBTng run with NWWKS.DLL

- size of the binary: 60688
- 215 C functions generated in 20 seconds
- three function calling convention manually entered
- 60 LocalAlloc calls, 132 assertions
- need to specify a simplified semantics of two functions: wcslen() and strlen()
- CBMC run took ca 6 minutes
- 7 failed assertions, 3 real bugs

UQBTng status and todo

- highly preliminary code
- to be more useful and reliable:
  - make it much more configurable
  - enable usage of pointers converted from integers
  - implement type recovery
  - better support for computed execution redirection, e.g. C++ virtual functions and optimized “switch” statements
  - detect integer underflow in memcpy() length
- apply to many binaries, find real 0-days

Questions ?

Links

- PVS prover: http://pvs.csl.sri.com
- Coq prover: http://coq.inria.fr
- Links to Dawson Engler's publications index: http://www.stanford.edu/~engler
- CBMC: http://www.cs.cmu.edu/~modelcheck/cbmc
- UQBT: http://www.itee.uq.edu.au/~cristina/uqbt.html