CSci 5271
Introduction to Computer Security
Day 11: OS security: higher assurance

Stephen McCamant
University of Minnesota, Computer Science & Engineering

Outline

Capability-based access control

OS trust and assurance

Assignment debrief and announcements

More Unix access control

ACLs: no fine-grained subjects

- Subjects are a list of usernames maintained by a sysadmin
- Unusual to have a separate subject for an application
- Cannot easily subset access (sandbox)

ACLs: ambient authority

- All authority exists by virtue of identity
- Kernel automatically applies all available authority
- Authority applied incorrectly leads to attacks

Confused deputy problem

- Compiler writes to billing database
- Compiler can produce debug output to user-specified file
- Specify debug output to billing file, disrupt billing

(Object) capabilities

- A *capability* both designates a resource and provides authority to access it
- Similar to an object reference
  - Unforgeable, but can copy and distribute
- Typically still managed by the kernel

Capability slogans (Miller et al.)

- No designation with authority
- Dynamic subject creation
- Subject-aggregated authority mgmt.
- No ambient authority
- Composability of authorities
- Access-controlled delegation
- Dynamic resource creation

Partial example: Unix FDs

- Authority to access a specific file
- Managed by kernel on behalf of process
- Can be passed between processes
  - Though rare other than parent to child
- Unix not designed to use pervasively

Distinguish: password capabilities

- Bit pattern itself is the capability
  - No centralized management
- Modern example: authorization using cryptographic certificates

Revocation with capabilities

- Use indirection: give real capability via a pair of middlemen
- $A \rightarrow B$ via $A \rightarrow F \rightarrow R \rightarrow B$
- Retain capability to tell R to drop capability to B
- Depends on composability

Confinement with capabilities

- A cannot pass a capability to B if it cannot communicate with A at all
- Disconnected parts of the capability graph cannot be reconnected
- Depends on controlled delegation and data/capability distinction

OKL4 and seL4

- Commercial and research microkernels
- Recent versions of OKL4 use capability design from seL4
- Used as a hypervisor, e.g. underneath paravirtualized Linux
- Shipped on over 1 billion cell phones

Joe-E and Caja

- Dialects of Java and JavaScript (resp.) using capabilities for confined execution
- E.g., of JavaScript in an advertisement
- Note reliance on Java and JavaScript type safety

Outline

Capability-based access control

OS trust and assurance

Assignment debrief and announcements

More Unix access control

Trusted and trustworthy

- Part of your system is trusted if its failure can break your security
- Thus, OS is almost always trusted
- Real question: is it trustworthy?
- Distinction not universally observed: trusted boot, Trusted Solaris, etc.

Trusted (I/O) path

- How do you know you're talking to the right software?
- And no one is sniffing the data?
- Example: Trojan login screen
  - Or worse: unlock screensaver with root password
  - Origin of "Press Ctrl-Alt-Del to log in"

Minimizing trust

- Kernel $\rightarrow$ microkernel $\rightarrow$ nanokernel
- Reference monitor concept
- TCB size: measured relative to a policy goal
- Reference monitor $\subseteq$ TCB
  - But hard to build monitor for all goals

How to gain assurance

- Use for a long time
- Testing
- Code / design review
- Third-party certification
- Formal methods / proof

Evaluation / certification

- Testing and review performed by an independent party
- Goal: separate incentives, separate accountability
- Compare with financial auditing
- Watch out for: form over substance, misplaced incentives

Orange book OS evaluation

- Trusted Computer System Evaluation Criteria

D. Minimal protection

C. Discretionary protection
  - C2 adds, e.g., secure audit over C1

B. Mandatory protection
  - B1<B2<B3: stricter classic MLS

A. Verified protection

Common Criteria

- International standard and agreement for IT security certification
- Certification against a *protection profile*, and *evaluation assurance level* EAL 1-7
- Evaluation performed by non-government labs
- Up to EAL 4 automatically cross-recognized

Common Criteria, Anderson's view

- Many profiles don't specify the right things
- OSes evaluated only in unrealistic environments
  - E.g., unpatched Windows XP with no network attacks
- "Corruption, Manipulation, and Inertia"
  - Pernicious innovation: evaluation paid for by vendor
  - Labs beholden to national security apparatus

Formal methods and proof

- Can math come to the rescue?
- Checking design vs. implementation
- Automation possible only with other tradeoffs
  - E.g., bounded size model
- Starting to become possible: machine-checked proof

Proof and complexity

- Formal proof is only feasible for programs that are small and elegant
- If you honestly care about assurance, you want your TCB small and elegant anyway
- Should provability further guide design?

Some hopeful proof results

- seL4 microkernel (SOSP'09 and ongoing)
  - 7.5 kL C, 200 kL proof, 160 bugs fixed, 25 person years
- CompCert C-subset compiler (PLDI'06 and ongoing)
- RockSalt SFI verifier (PLDI'12)

Outline

Capability-based access control

OS trust and assurance

Assignment debrief and announcements

More Unix access control

Exercise set 1 comments

- Net risk reduction: this is a formula, know how to compute it
- ‘`grep $username`‘: I should have said two *good* ways to change the code
- Solving $13 \cdot x \equiv 10 \pmod{2^{32}}$

Silly function

```
stat(pathname, &f);
stat("/what/ever", &we);
if (f.st_dev == we.st_dev && f.st_ino == we.st_ino) {
  return;
}
rfd = open(pathname, O_RDONLY);
buf = malloc(f.st_size - 1);
read(rfd, buf, f.st_size - 1);
close(rfd);
stat(pathname, &f);
if (f.st_dev == we.st_dev && f.st_ino == we.st_ino) {
  return;
}
wfd = open(pathname, O_WRONLY | O_TRUNC);
write(wfd, buf, f.st_size-1);
close(wfd);
```

Reversing the stack

```
void func(char *str) {
  char buf[128];
  strcpy(buf, str);
  do_something();
  return;
}
```

Payment app

```
void payment(char *name, int amount_gbp,
             char *purpose) {
    int amount_usd = (amount_gbp*156)/100;
    char memo[32];
    strcpy(memo, "Payment for: ");
    strcat(memo, purpose);
    write_check(name, amount_usd, memo);
}
```

Big- and little-endian

- Overwriting `0x12345678` with `"...AAAAA\0"`:
  - `0x00345678`
  - `0x41005678`
  - `0x41410078`
  - `0x41414100`
  - `0x41414141`

Big- and little-endian

- Overwriting `0x12345678` with `"...AAAAA\0"`:
  - `0x12345600`
  - `0x12340041`
  - `0x12004141`
  - `0x00414141`
  - `0x41414141`

Zip function

```
char *zip(char *a, char *b) {
  char *result;
  int len, i;
  len = strlen(a);
  result = malloc(2*len);
  for(i = 0; i <= len; i++) {
    result[2*i] = a[i];
    result[2*i+1] = b[i];
  }
  return result;
}
```

BCVS vulnerabilities

- Type 1: Buffer overflows and similar
  - Some easy to spot, but hard to exploit
- Type 2: Logic errors in running programs, file accesses, etc.
  - Usually easier to exploit once found

BCVS exploiting overflows

- Make sure control flow reaches the return
- Compensate for collateral damage
- Find your shellcode
- Writing shellcode

BCVS design changes

- Avoid unnecessary changes to benign functionality
  - Restricting length or character sets of arguments
  - Though, what is the benign functionality?
- Not a great candidate for privilege separation

Lattice model notation

- Element in lattice is a pair of:
  - Clearance level, totally ordered by $\leq$
  - Set of compartments, partially ordered by $\subseteq$
- Different notations:

  TA $\rightarrow$ (TA, $\varnothing$)

  Faculty//5271//8271 $\rightarrow$ (Faculty, $\{5271, 8271\}$)

Midterm exam Monday

- Usual class time and location
- Covers up through today's lecture
- Mix of short-answer and exercise-like questions
- Open books/notes/printouts, no computers or other electronics

Outline

Capability-based access control

OS trust and assurance

Assignment debrief and announcements

More Unix access control

Special case: `/tmp`

- We'd like to allow anyone to make files in `/tmp`
- So, everyone should have write permission
- But don't want Alice deleting Bob's files
- Solution: "sticky bit" 01000

Special case: group inheritance

- When using group to manage permissions, want a whole tree to have a single group
- When 02000 bit set, newly created entries with have the parent's group
  - (Historic BSD behavior)
- Also, directories will themselves inherit 02000

"POSIX" ACLs

- Based on a withdrawn standardization
- More flexible permissions, still fairly Unix-like
- Multiple user and group entries
  - Decision still based on one entry
- Default ACLs: generalize group inheritance
- Command line: `getfacl`, `setfacl`

ACL legacy interactions

- Hard problem: don't break security of legacy code
  - Suggests: "fail closed"
- Contrary pressure: don't want to break functionality
  - Suggests: "fail open"
- POSIX ACL design: old group permission bits are a mask on all novel permissions

"POSIX" "capabilities"

- Divide root privilege into smaller (~35) pieces
- Note: not real capabilities
- First runtime only, then added to FS similar to setuid
- Motivating example: `ping`
- Also allows permanent disabling

Privilege escalation dangers

- Many pieces of the root privilege are enough to regain the whole thing
  - Access to files as UID 0
  - `CAP_DAC_OVERRIDE`
  - `CAP_FOWNER`
  - `CAP_SYS_MODULE`
  - `CAP_MKNOD`
  - `CAP_PTRACE`
  - `CAP_SYS_ADMIN` (`mount`)

Legacy interaction dangers

- Former bug: take away capability to drop privileges
- Use of temporary files by no-longer setuid programs
- For more details: "Exploiting capabilities", Emeric Nasi

Next time

- Good luck on the midterm