Lecture 2: Exploiting Vulnerabilities: Buffer Overflow, Format string, Timing Attacks, and Logic Attacks

ECE1776
David Lie

Vulnerabilities

- We define a vulnerability as:
  - "A program flaw (or bug) that when exercised has a security implication"
  - Notice that two things need to be true. There has to be a flaw or a bug, that an attacker can **exploit** to weaken the security of a system (security implication)
- Examples of flaws are:
  - buffer overflow
  - unchecked inputs
  - race conditions
- A security implication could be:
  - Attacker can execute arbitrary code
  - Attacker can gain access to a system
  - Attacker can put the system in an illegal state

Some Commonly Exploitable Flaws

- 4 General Categories of flaws:
  - Memory corruption: redirection of control flow, code injection, argument overwrite
    - Buffer overflows, format string errors, heap overflows
  - Incomplete mediation. An attacker can access a resource because a certain path through the code does not check for authorization.
    - Forgetting to check for valid inputs, undocumented features
  - Difference between time of check and time of use. In concurrent systems, the attacker may exploit races.
    - Attacker gains access to a low security object, but switches it with a high security one.
  - Information Leakage:
    - System behavior unintentionally reveals some information

Code Injection - Buffer overflows

- Extremely common bug.
  - First major exploit in 1988: Morris Worm exploited fingerd.
- 10 years later: over 50% of all CERT advisories:
  - 1997: 16 out of 28 CERT advisories.
  - 1998: 9 out of 13 -"-
  - 1999: 6 out of 12 -"-
- Often leads to total compromise of host because of arbitrary code execution:
  - Fortunately: exploit requires expertise and patience.
  - Two steps:
    - Locate buffer overflow within an application.
    - Design an exploit.

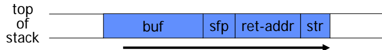
What are buffer overflows?

- Suppose a web server contains a function:

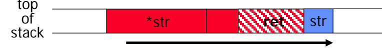
```
void func(char *str) {
  char buf[128];

 strcpy(buf, str);
}
```

Stack grows downwards, but buffers are written upwards in memory

- When the function is invoked the stack looks like:



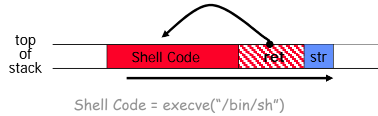
- What if `*str` is 136 bytes long? After `strcpy`:



5

Basic stack smashing exploit

- Now the attacker wants to execute injected code, so attacker puts the code in the buffer, and returns back into it



Shell Code = execve("/bin/sh")

- When func() exits, the user will be given a shell !!
- The difficult part:
  - To determine ret, the attacker has to correctly guess position of stack when func() is called.
- Main problem: no range checking in strcpy().
- Technique throughly explained in Aleph's article

6

Some unsafe libc functions

strcpy (char *dest, const char *src)
strcat (char *dest, const char *src)
gets (char *s)
scanf (const char *format, …)
printf (conts char *format, …)

⋮

7

Other types of Redirection/Injection Attacks

- Function pointers:

buf[128] | FuncPtr — Heap or stack

  - Overflowing buf will override function pointer.
- Other injection attacks:
  - Malloc errors can cause free to overwrite arbitrary addresses in memory
  - Attacker can overwrite entries in Global Offset Table (GOT)
    - Table of function pointers used for dynamic linking in ELF
- Not all attacks require/allow attacker to inject code:
  - Attacker can setup a false stack frame and redirect execution into existing code (i.e. system() in libc)

8

Incomplete Mediation

- These are cases where access should have been controlled/checked but was not, as a result the access is “unmediated” e.g.:
  - The “debug” mode for sendmail
  - Recent do_brk vulnerability in Linux 2.4.22 kernels
    - Incorrect checking of arguments allows a user to map pages from kernel into process address space
  - Undocumented system calls (in windows) allow direct access to protected structures
- Somewhat less common than code-injection, but still common enough
- These missed checks are not always obvious

Example of a Missing Check

```
/* 2.4.5/drivers/char/drm/i810_dma.c */
if(copy_from_user(&d, arg, sizeof(arg)))
  return –EFAULT;
if(d.idx > dma->buf_count)
  return –EINVAL;
buf = dma->buflist[d.idx];
Copy_from_user(buf->virtual, d.address, d.used);
```

- Missing lower bounds check for d.idx, what if d.idx < 0?
- This allows attacker to put arbitrary kernel address into buf, and then copy arbitrary data (in d.used) into that address

Concurrency Vulnerabilities

- Involve the time between time-of-check to time-of-use where the attacker is able to manipulate system state in a way to gain access or privileges.
- In a recent Usenix Security 2005 paper some authors demonstrated a similar file system race:
  - Programs like the print daemon, run as root. To determine whether you are allowed to print a file or not, they check the ownership of the file
  - To cause the race to happen, try to print a file in a VERY deep directory structure with a sym-link at the end, so it takes a long time to open the file.
  - Initially point the sym-link to a file you own. After the print daemon has checked the permissions, the OS spends a long time opening the file. At this time, switch the sym-link to point at a file that doesn’t belong to you and print it out.

Other Races

- More recent (in Linux 2.2.x and 2.4.x) ptrace-kmod exploit:
  - Ptrace is a facility used by debuggers to observe and modify another process, can stop the process, examine memory, and inject code
  - Normally, a user process cannot ptrace a root process.
  - A user process to can attach to a root setuid process before it becomes root. However, a race condition allows the tracing process to remain attached even if the setuid process takes on it’s root privileges, thus the tracing process inherets control over a root process.
- Exploits for these are rarer, since the flaws are very difficult to find.

Information Leakage

- Systems leak information that can be used by an attacker
- While computer systems are vulnerable, many other types of devices are often more vulnerable since they are simpler and usually only have one application:
  - Smartcards.
  - Cell phones.
  - PCI cards.
- Some examples are:
  - Leakage in Timing Channels
  - Power analysis

13

Timing attacks: example

- Consider the following pwd checking code:

```
int password-check( char *inp, char *pwd)
    if  (strlen(inp)  !=  strlen(pwd)) return 0;
    for( i=0; i < strlen(pwd); ++i)
        if  ( *inp[i]  !=  *pwd[i]  )
            return 0;
    return 1;
```

- A simple timing attack will expose the password one character at a time.

14

Timing attacks: example

- Correct code:

```
int password-check( char *inp, char *pwd)  {
    oklen = ( strlen(inp)  ==  strlen(pwd) ) ;
    for( ok=1, i=0;  i < strlen(pwd) ; ++i)
        ok = ok  &  ( inp[i]  ==  pwd[i] ) ;
    return ok & oklen;
}
```

- Make the function take the same amount of time no matter what

15

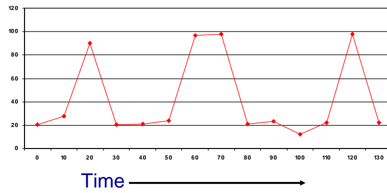
Timing Attacks

- Brumly and Boneh
  - Timing attack on OpenSSL
  - With detailed knowledge on cryptographic routines used, can correlate chose cipher texts with time to decrypt
  - Effective even with network and OS by trying many times to reduce random noise
- Other things may affect the timing of functions:
  - Page Faults
  - System calls
  - Interrupts

16

Power Analysis

- Power Analysis (Kocher)
  - To attack chips that perform cryptographic encryption/decryption
  - Can watch the power consumption of a device to ascertain the value of the key it is using



Time ⟶

17

Finding buffer overflows

- Hackers find buffer overflows as follows:
  - Run web server on local machine.
  - Issue requests with long tags.
    All long tags end with "$$$$$".
  - If web server crashes,
    search core dump for "$$$$$" to find overflow location.
- Some automated tools exist in industry

18

Preventing buf overflow attacks

- Main problem:
  - strcpy(), strcat(), sprintf() have no range checking.
  - "Safe" versions strncpy(), strncat() are misleading
    - strncpy() may leave buffer unterminated.
    - strncpy(), strncat() encourage off by 1 bugs.
- Defenses:
  - Type safe languages (Java, ML). Legacy code?
  - Mark stack as non-execute or Random stack location.
  - Static source code analysis.
  - Run time checking: StackGuard, Libsafe, SafeC, (Purify).

19

Marking stack as non-execute

- Basic stack exploit can be prevented by marking stack segment as non-executable or randomizing stack location.
  - Openwall code patches exist for Linux to allow this (Solar Designer)
  - Made obsolete by hardware support in Intel (XD bit) and AMD (NX bit) processors
- Problems:
  - Does not block more general overflow exploits:
    - Overflow on heap: overflow buffer next to func pointer.
    - Return into libc attack
  - Some apps need executable stack
    - Interpreted Languages
    - OS Trampolines for signals

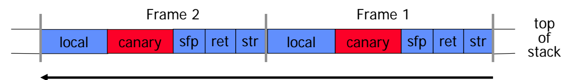
20

Static source code analysis

- Statically check source to detect buffer overflows.
  - Several consulting companies.
- Can we automate the review process?
- Several tools exist:
  - @stake.com (l0pht.com): SLINT (designed for UNIX)
  - rstcorp: its4. Scans function calls.
  - Berkeley: Wagner, et al. Test constraint violations.
  - Stanford: Engler, et al. Test trust inconsistency.
- Find lots of bugs, but not all.

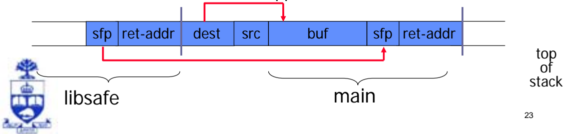
Run time checking: StackGuard

- Many many run-time checking techniques …
- StackGuard (WireX)
  - Run time tests for stack integrity.
  - Embed "canaries" in stack frames and verify their integrity prior to function return.



Run time checking: Libsafe

- Libsafe (Avaya Labs)
  - Dynamically loaded library
    - Overrids libc.so (with /etc/ld.so.preload)
    - Done at runtime so doesn't need recompile
  - Intercepts calls to strcpy (dest, src)
    - Validates sufficient space in current stack frame:
      |frame-pointer – dest| > strlen(src)
    - If so, does strcpy.
      Otherwise, terminates application.



Libsafe Drawbacks

- Makes assumptions about the stack
  - Only works on x86 machines
  - Statically linked programs won't be protected
  - Some optimized programs (no frame pointer) won't be protected
  - Only catches overflows on library functions:

```
While (*c != '\n') {
  *p++ =  *c++
}
```

More methods …

- PAX/grsecurity:
  - Randomize location of functions in libc. Attacker cannot jump directly to exec function.
  - Emulate execute bits on pages so only pages originating on disk can have code
- Exec Shield
  - Make code pages executable only
  - Randomize starting stack location

Extra Slides

Exploiting buffer overflows

- Suppose web server calls func() with given URL.
- Attacker can create a 200 byte URL to obtain shell on web server.
- Some complications:
  - Program P should not contain the '\0' character.
  - Overflow should not crash program before func() exists.
- Sample buffer overflows of this type:
  - Overflow in MIME type field in MS Outlook.
  - Overflow in ISAPI in IIS.

Format string problem

```
int func(char *user) {
   fprintf( stdout, user);
}
```

Problem: what if user = "%s%s%s%s%s%s%s" ??

- Most likely program will crash: DoS.
- If not, program will print memory contents. Privacy?
- Full exploit using user = "%n"

Correct form:

```
int func(char *user) {
   fprintf( stdout, "%s", user);
}
```

History

- Danger discovered in June 2000.
- Examples:
  - wu-ftpd 2.* : remote root.
  - Linux rpc.statd: remote root
  - IRIX telnetd: remote root
  - BSD chpass: local root

29

Vulnerable functions

Any function using a format string.

Printing:
  printf, fprintf, sprintf, …
  vprintf, vfprintf, vsprintf, …

Logging:
  syslog, err, warn

30

Exploit

- Dumping arbitrary memory:
  - Walk up stack until desired pointer is found.
  - printf(“%08x.%08x.%08x.%08x|%s|”)
- Writing to arbitrary memory:
  - printf(“hello %n”, &temp) -- writes ‘6’ into temp.
  - printf(“%08x.%08x.%08x.%08x.%n”)

31

Overflow using format string

```
char errmsg[512], outbuf[512];
sprintf (errmsg, “Illegal command: %400s”, user);
…
sprintf( outbuf, errmsg );
```

- What if user = “%500d <nops> <shellcode>”
  - Bypass “%400s” limitation.
  - Will ovreflow outbuf.

32