Security Attacks and Lessons

COMP 321

Dave Johnson

1

Types of Security Services

Authentication

- Knowing that data read or received is the same as written or sent
- And that the writer/sender is who they claim they are

Integrity

- Ensuring that data is transmitted without undetected alteration

Confidentiality

- Communicating so intended recipients know what is being said
- But unintended parties cannot determine what was said
- Traffic flow confidentiality also protects privacy of who is talking to who

2

Types of Security Services (continued)

Non-repudiation

- Reader/receiver of data is able to prove that the writer/sender of it did in fact write/send it
- Even if writer/sender later denies having done so

Replay protection

- Any message already received cannot be replayed to the receiver without detection

Denial of service protection

- Avoiding attacks that can disable or otherwise swamp a machine

Computer Viruses and Worms

Virus

- Code ***embedded*** in another program
- So that, when the program is run, the virus attempts to replicate by embedding copies of its code in other programs

Worm

- A ***standalone*** program that can run on its own
- Attempts to replicate itself by spreading itself such as over networks

Example: The "Morris Internet Worm"

Launched on evening of November 2, 1988, spread to about 10% of Internet

- Launched from computer at ***MIT*** sometime after 5:00 PM EST
- Goals of the worm
  - Spread as quickly as possible to other computers
  - Be hard to detect and stop
- Targeted ***VAX*** and ***Sun-3*** computers running ***BSD Unix***
- After 2 days, mostly cleaned up
  - Estimated cost of dealing with it at each installation was $200 to > $53,000
  - US General Accounting Office (GAO) estimated total loss between $100,000 and $10,000,000

Trial, Conviction, and Appeal

On investigation, discovered the worm was launched by Robert Tappan Morris

- Had been an undergrad at Harvard, was then first year grad student at ***Cornell***
- ***First person*** convicted of violating the ***Computer Fraud and Abuse Act of 1986***
  - 18 U.S.C. Section 1030 (a) (5) (A) covers anyone who
  - "(5) ***intentionally accesses*** a Federal interest computer without authorization, ***and*** by means of one or more instances of such conduct alters, damages, ***or*** destroys information in any such Federal interest computer, or ***prevents authorized use*** of any such computer or information, ***and thereby***"
  - "(A) ***causes loss*** to one or more others of a value aggregating $1,000 or more during any one year period"

Trial, Conviction, and Appeal

- Sentenced on May 16, 1990
  - 3 years on probation
  - 400 hours of community service
  - $10,050 fine ("a fine of 10,000 and a special assessment of $50")
  - plus probation costs ("at a rate of $91.00 per month")
- Appealed to Circuit Court of Appeals for the Second Circuit – verdict upheld
- Appealed to U.S. Supreme Court – declined to hear the case
- Suspended for 1 year from Cornell
- Later became a grad student at Harvard, completed his PhD there in 1999
- Now a ***tenured full Professor of Computer Science at MIT***, and an ***ACM Fellow***

THURSDAY
SYRACUSE
HERALD-JOURNAL
SUBURBAN EDITION
JANUARY 18, 1990
30 cents

Worm went 'out of control'

Morris says he didn't mean to hurt computers

By Dan Kane
Staff Writer

Robert Morris, speaking publicly for the first time, testified today about the computer worm he unleashed in November 1988.

"I had never heard of any such thing before," he said of his plan to infiltrate a national computer network. "My purpose was to see if I could write a program that would spread as widely as possible in the Internet."

But, Morris said, he made a mistake.

"I was scared. It seemed like the worm was getting out of control," Morris testified in federal court this morning.

He said he had been experimenting with the computer program for two weeks before he activated it on Nov. 2, 1988.

https://media.syracuse.com/vintage/other/2016/01/20/Testifies%20merged.pdf

Syracuse "Post Standard", January 21, 2016

"Throwback Thursday: Cornell student Robert Morris guilty of unleashing first Internet worm"

- "Ironically, the eight men and four women that would determine Morris' fate were chosen specifically for their lack of computer knowledge. The three prospective jurors who had home computers were immediately dismissed."
- "In a Jan. 9, 1990 Post-Standard story, FBI agent Joseph O'Brien joked that he could prove Morris committed a crime, but it would be tough to explain how Morris did it."
- "Morris greeted Judge Howard Munson's ruling with a smile. His father gave him a handshake and Anne Morris gave her son a hug."

Methods of Spreading: (1) rlogin and rsh

rlogin and rsh do remote login/execution, similar to today's ssh

- Can configure destination machine to not require a password
  - ~/.rhosts (for specific user) or /etc/hosts.equiv (for all users)
- Worm also tried brute force ***password*** guessing, trying
  - Based on usernames and full names on source machine, for example
    "dbj", "dbjdbj", "Dave", "Johnson", "dave", "johnson", "jbd"
  - An internal dictionary of 432 words
  - The standard system dictionary, then at "/usr/dict/words"
- On success, the worm then spread to the destination machine
- ***Lessons***: use ***strong*** passwords, and be careful if not requiring a password

Methods of Spreading: (2) the sendmail server

The standard Unix email server for Simple Mail Transfer Protocol (SMTP)

- Server listens for connections over network for sending email to this machine
- At the time, Internet email wasn't the only email system
  - Many different email address formats, different gateways between systems
- To simplify debugging, sendmail server supported a DEBUG command
  - Server then takes email destination address as a local command to fork and exec to run (worm used "/bin/sh"), with email body as standard input
  - Sendmail server always ran as the "root" user
  - The default DEBUG password empty
- ***Lessons***: Use ***strong*** passwords (even as defaults), and ***disable*** "unknown" interfaces like this when not needed

Methods of Spreading: (3) the finger server

Command and server for looking up information about users

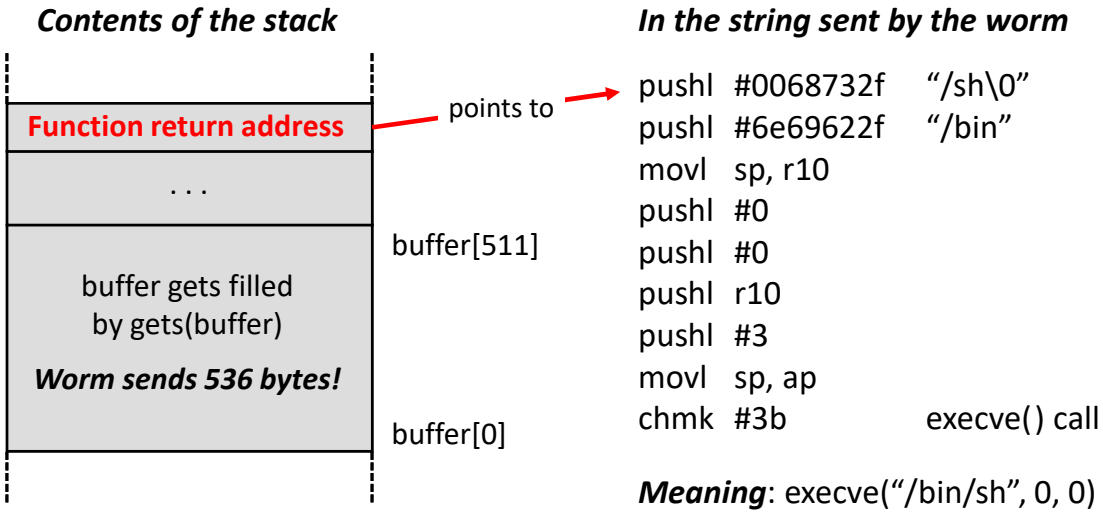
- "finger dbj@machine" connects to ***fingerd*** on "machine" over the network
  - Sends "dbj\n" to server (server's standard input), and server returns result
- ***This part of the worm targeted only VAX computers, not Sun-3 computers***
- The code on the server side looks like

```
function_name() {
        char buffer[512];
        . . .
        gets(buffer)
        . . .
}
```

Remember the COMP 222 "Attack Lab"?

- The finger server always ran as the "root" user

Methods of Spreading: (3) the finger server



Methods of Spreading: (3) the finger server

This is an example of what now is known as a "buffer overflow" attack

- The result was a "root" shell, with its standard input connected to network
- Other buffer overflow attacks may be less dramatic but are still dangerous
- ***Lessons***
  - Be very careful about whether your code can overflow a buffer
  - Never use interfaces that can't check for buffer overflows, for example

Don't use	***Use instead***
gets	fgets
strcpy	strncpy
strcat	strncat
sprint	snprintf

Doesn't Execute Protection on Pages Solve This?

Hardware checking of page execute protection was not common then

- Neither the VAX nor Sun-3 hardware had any execute protection bit for pages
- The x86 hardware architecture didn't support it until
  - 2003 (AMD NX "no execute" bit) or
  - 2004 (Intel XD "execution disable" bit)
- And then Microsoft had to add support for it
  - Windows XP Service Pack 2, 2004
    (DEP "Data Execution Prevention" feature)
- Then the system administrator still had to ***turn on*** that feature on ***each individual*** machine

A New Kind of Overflow Attack: Return-to-libc

In the attack, use executable code that's already there!

- Can't attack by buffer overflow putting new code into memory (on the stack)
  - The stack is readable/writable only, and now execute protection is checked
- But you can still overflow the stack (or any other buffer)
- And a lot of code is already in memory, since most systems use shared libraries, such as for standard C library (libc)
- libc contains many "interesting" functions useful for an attack, such as
  - execve (and execl, execlp, execle, execv, execvp, and execvpe)
  - popen (fork and exec, with standard input or output connected by pipe)
  - system (run a command by the shell)

```
int  system(const char  *command);
```

A New Kind of Overflow Attack: Return-to-libc

A simple example of a return-to-libc attack

- Overflow a buffer on the stack to overwrite the return address with address of an existing function (e.g., address of system( ) function in libc)
- When current function returns, will branch to and execute that new function
- But what about the arguments to that new function?
  - Many architectures/systems pass the function arguments on the stack (example: 32-bit x86)
  - "Easy", just need to be careful to put the arguments in the right place on the stack as part of the buffer overflow
  - ***But what about architectures/systems that pass function arguments in registers (example: 64-bit x86-64)?***

A New Kind of Overflow Attack: Return-to-libc

Example: x86-64 function calling (arguments in registers)

```
; Want to call a function "myFunc" that takes three
; integer parameters. First parameter is in rax.
; Second parameter is the constant 123. Third
; parameter is in memory location "var"

push rdi          ; rdi will be a param, so saving it

                  ; long retVal = myFunc(x, 123, z);
mov rdi, rax      ; put first param in rdi
mov rsi, 123      ; put second param in rsi
mov rdx, [var]    ; put third param in rdx

call myFunc       ; call the function

pop rdi           ; restore saved rdi value

; return value of myFunc is now available in rax
; (if there is any return value)
```

(note saving of rdi)

first argument goes in rdi register

(note restoring of rdi)

https://aaronbloomfield.github.io/pdr/book/x86-64bit-ccc-chapter.pdf

A New Kind of Overflow Attack: Return-to-libc

For systems that pass function arguments in registers

- Can put intended argument value on the stack as part of the buffer overflow
- Find (e.g., in libc) a small snippet of code that does (for the first argument)

```
pop   rdi
ret
```

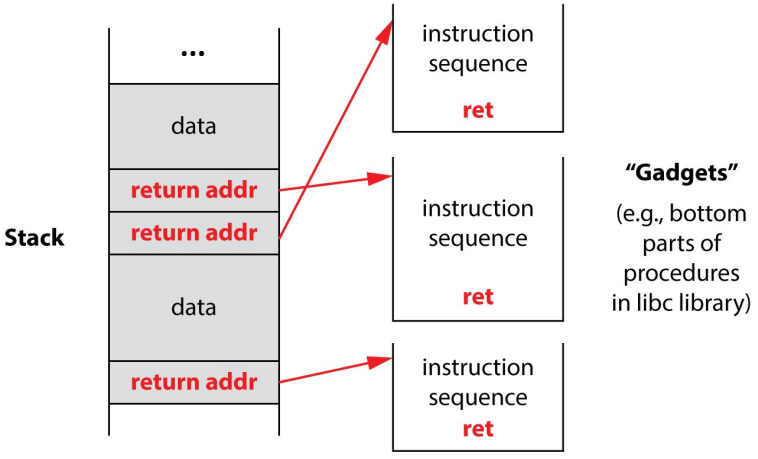
or

```
pop   rdi
. . . don't change the value of rdi . . .
ret
```

- Also put ***two*** new return address on the stack as part of the buffer overflow
  - ***First*** return address is the address of this snippet of code
  - (intended argument value)
  - ***Second*** return address (when ***that*** function returns) is the address of the system( ) function

A Generalization: Return-Oriented-Programming

Chain together perhaps many code snippets, each of which ends with a return



A Generalization: Return-Oriented-Programming

A "Turing-complete" catalog of gadgets in libc

- This means can be used to compute ***anything*** (equivalent to a Turing machine)
- See, e.g., "Return-Oriented Programming: Systems, Languages, and Applications", Ryan Romer, et al., ACM Transactions on Information and System Security, March 2012
- They examined the standard libc on two architectures/systems
  - Linux/x86 (CISC, arguments on the stack)
  - Solaris/SPARC (RISC, arguments in registers)
- Found and described a Turing-complete catalog of gadgets found in libc on both architectures/systems
  - Each of length typically just 2-5 instructions
  - Combining them can compute/do literally ***anything***