Chapter 6

My Cup Runneth Over
(Note: most O/S concepts are not covered in these slides, use textbook)

Buffer Overflows

• Attack?
  – Buffer overflow with dialer.exe
  – How long is a phone number?

• Harm?
  – Destruction of code and data
Memory allocation

- OS allocate memory wherever it finds a “hole”/space
- Program Counter
  - Points to next instruction to execute
- Instruction set
  - If, branch (while, for), goto, call
  - Data “execution”? Arithmetic on instructions?
    - It can either be a valid instruction or not

Data vs Code

- Consider program:
  - 0x5:1A 14
  - 0x6:1B 3C
  - 0x7:1E 50
Data vs Code

– Consider program:
  • 0x5:1A 14 (1A = "read from", 14 = "memory position 0x14 (20)"
  • 0x6:1B 3C (1B = "add to", 3C = "memory position 0x3C (60)"
  • 0x7:1E 50 (1E = "store at", 50 = "memory position 0x50 (80)"
  • 0x8:1C 60 (1C = "jump to", 60 = "memory position 0x60 (96)"
  • …
  • 0x14:190
  • …
  • 0x3C:1F4
  • …
  • 0x50:0

PC: 0x4 Register: 0
Data vs Code

– Consider program:
  • 0x5:1A 14 (1A = “read from”, 14 = “memory position 0x14 (20)”)
  • 0x6:1B 3C (1B = “add to”, 3C = “memory position 0x3C (60)”)
  • 0x7:1E 50 (1E = “store at”, 50 = “memory position 0x50 (80)”)
  • 0x8:1C 60 (1C = “jump to”, 60 = “memory position 0x60 (96)”)
  • …
  • 0x14:190 (400)
  • …
  • 0x3C:1F4 (500)
  • …
  • 0x50:0

PC: 0x5 Register: 0x190 (400)
Data vs Code

– Consider program:
  • 0x5:1A 14 (1A = "read from", 14 = "memory position 0x14 (20)"
  • 0x6:1B 3C (1B = "add to", 3C = "memory position 0x3C (60)"
  • 0x7:1E 50 (1E = "store at", 50 = "memory position 0x50 (80)"
  • 0x8:1C 60 (1C = "jump to", 60 = "memory position 0x60 (96)"
  • …
  • 0x14:190 (400)
  • …
  • 0x3C:1F4 (500)
  • …
  • 0x50:384 (900)

PC: 0x7 Register: 0x384 (900)

Data vs Code

– Consider program:
  • 0x5:1A 14 (1A = "read from", 14 = "memory position 0x14 (20)"
  • 0x6:1B 3C (1B = "add to", 3C = "memory position 0x3C (60)"
  • 0x7:1E 50 (1E = "store at", 50 = "memory position 0x50 (80)"
  • 0x8:1C 60 (1C = "jump to", 60 = "memory position 0x60 (96)"
  • …
  • 0x14:190 (400)
  • …
  • 0x3C:1F4 (500)
  • 0x50:384 (900)
  • 0x60:1A 50

PC: 0x8 Register: 0x384 (900)
Data vs Code

– Consider program:
  • 0x5:1A 14 (1A = "read from", 14 = "memory position 0x14 (20)"
  • 0x6:1B 3C (1B = "add to", 3C = "memory position 0x3C (60)"
  • 0x7:1E 50 (1E = "store at", 50 = "memory position 0x50 (80)"
  • 0x8:1C 60 (1C = "jump to", 60 = "memory position 0x60 (96)"
  • …
  • 0x14:190 (400)
  • …
  • 0x3C:1F4 (500)
  • 0x50:384 (900)
  • 0x60:1A 50

PC: 0x60  Register: 0x384 (900)

Data vs Code

– What does 0x1C0A mean?
  • 0x0A = 10
  • So, 0x1C0A would mean "jump to byte 10"

– If we can change instruction “0x3C:1F4” to “0x3C:1A7A”…
  • Data is changed to an instruction! We only need to figure out what is 0x1A7A – 0x1F4 = 0x1813
  • Thus, we need to try and overflow 0x3C with 0x1813 to change it to 1A7A
  • Let us assume we were able to do this, now the example would change as follows:
Data vs Code

– Consider program:
  • 0x5:1A 14 (1A = "read from", 14 = "memory position 0x14 (20)"
  • 0x6:1B 3C (1B = "add to", 3C = "memory position 0x3C (60)"
  • 0x7:1E 50 (1E = "store at", 50 = "memory position 0x50 (80)"
  • 0x8:1C 60 (1C = "jump to", 60 = "memory position 0x60 (96)"
  • ....
  • 0x14:190 (400)
  • ....
  • 0x3C:1A7A (6778)
  • 0x50:0
  • 0x60:1A 50
  • 0x0A:Rogue program

PC: 0x5 Register: 0x190 (400)
Data vs Code

– Consider program:
  • 0x5:1A 14 (1A = "read from", 14 = “memory position 0x14 (20)"
  • 0x6:1B 3C (1B = “add to", 3C = "memory position 0x3C (60)
  • 0x7:1E 50 (1E = "store at", 50 = "memory position 0x50 (80)
  • 0x8:1C 60 (1C = "jump to", 60 = "memory position 0x60 (96)
  • …
  • 0x14:190 (400)
  • …
  • 0x3C:1A7A (6778)
  • 0x50:1C0A
  • 0x60:1A 50
  • 0x0A:Rogue program
  
  PC: 0x7    Register: 0x1C0A (7178)
Data vs Code

– Consider program:
  • 0x5:1A 14 (1A = "read from", 14 = "memory position 0x14 (20)"
  • 0x6:1B 3C (1B = "add to", 3C = "memory position 0x3C (60)"
  • 0x7:1E 50 (1E = "store at", 50 = "memory position 0x50 (80)"
  • 0x8:1C 60 (1C = "jump to", 60 = "memory position 0x60 (96)"
  • ...
  • 0x14:190 (400)
  • ...
  • 0x3C:1A7A (6778)
  • 0x50:1C0A
  • 0x60:1A 50
  • 0x0A:Rogue program

PC: 0x60 Register: 0x1C0A (7178)

Data vs Code

– Consider program:
  • 0x5:1A 14 (1A = "read from", 14 = "memory position 0x14 (20)"
  • 0x6:1B 3C (1B = "add to", 3C = "memory position 0x3C (60)"
  • 0x7:1E 50 (1E = "store at", 50 = "memory position 0x50 (80)"
  • 0x8:1C 60 (1C = "jump to", 60 = "memory position 0x60 (96)"
  • ...
  • 0x14:190 (400)
  • ...
  • 0x3C:1A7A (6778)
  • 0x50:1C0A
  • 0x60:1A 50
  • 0x0A:Rogue program

PC: 0x50 Register: 0x1C0A (7178)
Data vs Code

– Consider program:
  • 0x5:1A 14 (1A = "read from", 14 = "memory position 0x14 (20)"
  • 0x6:1B 3C (1B = "add to", 3C = "memory position 0x3C (60)"
  • 0x7:1E 50 (1E = "store at", 50 = "memory position 0x50 (80)"
  • 0x8:1C 60 (1C = "jump to", 60 = "memory position 0x60 (96)"
  • ...
  • 0x14:190 (400)
  • ...
  • 0x3C:1A7A (6778)
  • 0x50:1C0A
  • 0x60:1A 50
  • 0x0A: Rogue program

   PC: 0x0A    Register: 0x1C0A (7178)

Harm from an overflow

• Trial-and-error
• Privilege escalation
• Overwriting memory
  char sample[10] = “AAAAAAAAAAA”;
  sample[10] = ‘B’;
Effects from an overflow attack

- Overwrite the PC
  - When the procedure exits, control returns to a different (rogue program) place
- Overwrite part of code in low memory
  - To substitute code with attacker's code
- Overwrite the PC and data in stack
Vulnerabilities

- Integer overflow
- Off-by-one error
Vulnerabilities

- Unterminated null-terminated strings

FIGURE 6-9 Variable-Length String Representations

From Analyzing Computer Security by Charles P. Pfleeger and Shari Lawrence Pfleeger
(ISBN: 0132789469) Copyright © 2012 Pearson Education, Inc. All rights reserved.
Vulnerabilities

- Parameter length and number
  - Too many parameters
  - Wrong type/size
  - Too long string
- Unsafe utility programs
  - strcpy(dest, src)

Attacks

- Morris worm
- Code Red
- SQL slammer
- Conficker
Countermeasures

- Programmer bounds checking
- Programming language support
- Stack protection/tamper detection
- Hardware protection of executable space
- General access control
  - Directory, list, matrix
FIGURE 6-22 Directory Access Rights

FIGURE 6-24 Access Control Matrix