CS165 – Computer Security

Midterm Review

February 13, 2024

Midterm Structure

- Three sections
  - 10 multiple choice (3pts each)
    - Fill in the blank with specific choices
  - 5 short answer (7pts each)
    - Scenarios that you answer 1 or 2 questions
    - Free form – 2-3 sentences
  - 3 “constructions” (11-12pts each)
    - Scenarios with problem solving
    - 3-4 sub-questions
- Watch the time – answer the questions you know first

Midterm Scope

- Up to and including the “ROP lecture”
  - Does not include the heap lecture
  - We will have a project on the heap
    - Will have ”heap attacks” on the final
  - Should do at least the first attack in P2 for the exam
    - Help make attacks on the stack concrete

Homework

- 1. What is necessary for a software flaw (e.g., memory error)?
  - a) The flaw must be accessible to an adversary.
  - b) An adversary must be able to exploit the flaw.
  - c) Both a) and b)
- 2. Why do we add a “salt” when we compute a hash of a password when storing?
  - a) Because if the machine is compromised, passwords can be stolen directly.
  - b) To make the password stronger.
  - c) To prevent storing the same hash value for two passwords that match.

Homework (Part 1)

- 1. What is necessary for a software flaw (e.g., memory error) to become a vulnerability?
  - a) The flaw must be accessible to an adversary.
  - b) An adversary must be able to exploit the flaw.
  - c) Both a) and b)
- 2. Why do we add a “salt” when we compute a hash of a password when storing?
  - a) Because if the machine is compromised, passwords can be stolen directly.
  - b) To make the password stronger.
  - c) To prevent storing the same hash value for two passwords that match.

Homework (Part 1)

- 1. What is necessary for a software flaw (e.g., memory error) to become a vulnerability?   __C_____ (all accepted)
  - a) The flaw must be accessible to an adversary.
  - b) An adversary must be able to exploit the flaw.
  - c) Both a) and b)
- 2. Why do we add a "salt" when we compute a hash of a password when storing?   ___C_____
  - a) Because if the machine is compromised, passwords can be stolen directly.
  - b) To make the password stronger.
  - c) To prevent storing the same hash value for two passwords that match.

Homework (Part 1)

- 3. Which of the following describes an attack on availability?
  - a) It is hard to notice.
  - b) It can stop legitimate users from using a service
  - c) It can only happen due to a network denial-of-service attack.
- 4. Why is computer security about looking at corner cases of a program?
  - a) Because vulnerabilities are triggered by inputs that are commonly observed in typical workloads.
  - b) Because security problems cannot occur in common cases of a program.
  - c) Because many security vulnerabilities are hidden and hard to discover.

Homework (Part 1)

- 3. Which of the following describes an attack on availability? ______B______
  - a) It is hard to notice.
  - b) It can stop legitimate users from using a service
  - c) It can only happen due to a network denial-of-service attack.
- 4. Why is computer security about looking at corner cases of a program? _____C_______
  - a) Because vulnerabilities are triggered by inputs that are commonly observed in typical workloads.
  - b) Because security problems cannot occur in common cases of a program.
  - c) Because many security vulnerabilities are hidden and hard to discover.

Homework (Part 1)

- 5. Which statement best describes a spatial error like a buffer overflow?
  - a) A referent (i.e., pointer) assigned to an allocated region may be used to read outside that allocated region.
  - b) Allows a memory write outside an allocated region.
  - c) A pointer is used in a memory operation before being assigned to an allocated region.
- 6. Which statement best describes a temporal error?
  - a) A memory region is read before a pointer is assigned to reference that region.
  - b) A pointer is assigned to an allocated region of another data type.
  - c) A pointer is used in a memory operation before being assigned to an allocated region.

Homework (Part 1)

- 5. Which statement best describes a spatial error like a buffer overflow?
  - a) A referent (i.e., pointer) assigned to an allocated region may be used to read outside that allocated region.
  - b) Allows a memory write outside an allocated region.
  - c) A pointer is used in a memory operation before being assigned to an allocated region.
- 6. Which statement best describes a temporal error?
  - a) A memory region is read before being allocated.
  - b) A pointer is assigned to an allocated region of another data type.
  - c) A pointer is used in a memory operation before being assigned to an allocated region.

Homework (Part 1)

- 5. Which statement best describes a spatial error like a buffer overflow? ___A____
  - a) A referent (i.e., pointer) assigned to an allocated region may be used to read outside that allocated region.
  - b) Allows a memory write outside an allocated region.
  - c) A pointer is used in a memory operation before being assigned to an allocated region.
- 6. Which statement best describes a temporal error? ___C__
  - a) A memory region is read before being allocated
  - b) A pointer is assigned to an allocated region of another data type.
  - c) A pointer is used in a memory operation before being assigned to an allocated region.

Homework (Part 1)

- 7. What happens when we cast on object of type A to an object of type B in the C programming language?
  - a) Assign a pointer to the object that interprets the object’s memory layout according to type B.
  - b) Reformat the memory layout of the object (originally of type A) to the format of type B.
  - c) Casts between different types are not allowed in the C programming language.
- 8. What is a security flaw that may be caused because of the limitations of strncpy?
  - a) Cause an illegal information flow.
  - b) Create a string that lacks a null-terminator.
  - c) Write outside the destination’s memory region.

Homework (Part 1)

- 7. What happens when we cast on object of type A to an object of type B in the C programming language?
  - a) Assign a pointer to the object that interprets the object’s memory layout according to type B.
  - b) Reformat the memory layout of the object (originally of type A) to the format of type B.
  - c) Casts between different types are not allowed in the C programming language.
- 8. What is a security flaw that may be caused because of the limitations of strncpy?
  - a) Cause an illegal information flow.
  - b) Create a string that lacks a null-terminator.
  - c) Write beyond the length of the specified limit used in strncpy.

Homework (Part 1)

- 7. What happens when we cast on object of type A to an object of type B in the C programming language? ___A_____
  - a) Assign a pointer to the object that interprets the object’s memory layout according to type B.
  - b) Reformat the memory layout of the object (originally of type A) to the format of type B.
  - c) Casts between different types are not allowed in the C programming language.
- 8. What is a security flaw that may be caused because of the limitations of strncpy? ___B_____ B&C accepted
  - a) Cause an illegal information flow.
  - b) Create a string that lacks a null-terminator.
  - c) Write beyond the length of the specified limit used in strncpy.

Homework (Part 1)

- 9. What is the advantage of applying the “%ms” format identifier in scanf?
    - a) Avoids the program running out of memory.
    - b) Automatically performs all allocations and deallocations for the string object.
    - c) Allocates a larger buffer when the input exceeds the memory allocated for the string.
- 10. What must an adversary modify via a memory error permits to launch a control-flow hijack?
    - a) a function pointer
    - b) a data pointer
    - c) a function’s code

Homework (Part 1)

- 9. What is the advantage of applying the “%ms” format identifier in scanf?
  - a) Avoids the program running out of memory.
  - b) Automatically performs all allocations and deallocations for the string object.
  - c) Allocates a larger buffer when the input exceeds the memory allocated for the string.
- 10. What must an adversary modify via a memory error ~~permits~~ to launch a control-flow hijack?
  - a) a function pointer
  - b) a data pointer
  - c) a function’s code

Homework (Part 1)

- 9. What is the advantage of applying the “%ms” format identifier in scanf? ___C____
  - a) Avoids the program running out of memory.
  - b) Automatically performs all allocations and deallocations for the string object.
  - c) Allocates a larger buffer when the input exceeds the memory allocated for the string.
- 10. What must an adversary modify via a memory error to launch a control-flow hijack? ___A____
  - a) a function pointer
  - b) a data pointer
  - c) a function’s code

Homework (Part II)

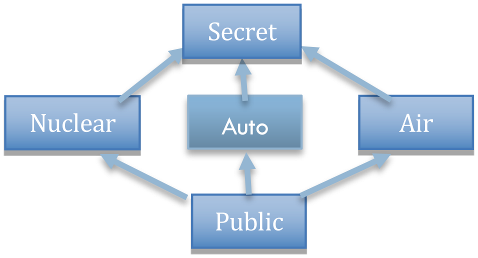
- Estimate the number of guesses needed to crack a password from the information below. (1.5 points each)
  - 1.How many more guesses does it take to guess a 12-character password than an 8-character password, assuming 100 options are available for each character? Try to estimate the answer without a calculator in terms of powers of 10.
  - 2.What is the minimum number of guesses will it take to crack the password "ABC123" given the structures shown in frequency order below and assuming 10 characters for upper case letters and digits?
    - U2D4
    - U1D5
    - U3D3

Homework (Part II)

- Estimate the number of guesses needed to crack a password from the information below. (1.5 points each)
  - 1.How many more guesses does it take to guess a 12-character password than an 8-character password, assuming 100 options are available for each character? Try to estimate the answer without a calculator in terms of powers of 10.
    - Ans: 100^12 - 100^8 = (10*10)^12 - (10*10)^8 = 10^24-10^16 = ~10^24
  - 2.What is the minimum number of guesses will it take to crack the password "ABC123" given the structures shown in frequency order below and assuming 10 characters for upper case letters and digits?
    - U2D4 (first)
    - U1D5 (second)
    - U3D3 (third)
    - Ans: 10^2 * 10^4 + 10^1 * 10^5 + 1 = 2 x 10^6 + 1

Homework (Part III)

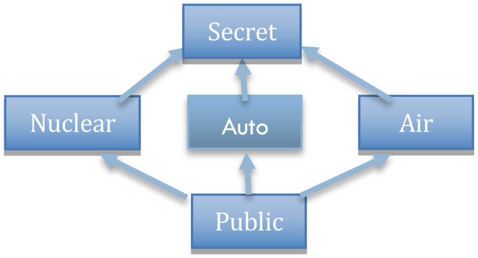
- For the following questions on information flows, assume the following lattice security policy. (1pt each)
  - 1.What is the label of the variable “e” after executing Line 6?
  - 2.What is the label of the variable “a” after executing Line 6?
  - 3.What is the label of “c” after executing Line 8?
  - 4.What is the label of “d” after executing Line 10?
  - 5.Is the operation in Line 10 legal given the resultant information flows?



```
1: Int {Auto} a;

2: Int {Public} b, d;
3: Int c, e;

4: a = 7;
5: b = 2;
6: e = a+b;

7: if (a > 0) {
8:         c = b;
9: }
10: d = c;
```

Homework (Part III)

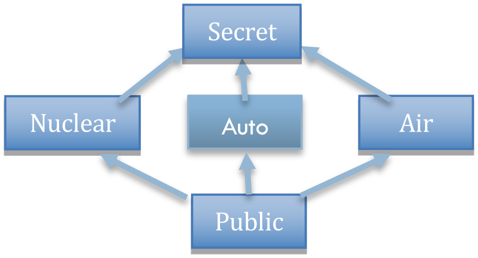
- For the following questions on information flows, assume the following lattice security policy. (1pt each)
  - 1.What is the label of the variable “e” after executing Line 6? Auto – explicit flow from *a* (Auto) and *b* (Public) – LUB of Auto and Public is “Auto”
  - 2.What is the label of the variable “a” after executing Line 6?
  - 3.What is the label of “c” after executing Line 8?
  - 4.What is the label of “d” after executing Line 10?
  - 5.Is the operation in Line 10 legal given the resultant information flows?



```
1: Int {Auto} a;

2: Int {Public} b, d;
3: Int c, e;

4: a = 7;
5: b = 2;
6: e = a+b;

7: if (a > 0) {
8:        c = b;
9: }
10: d = c;
```

Homework (Part III)

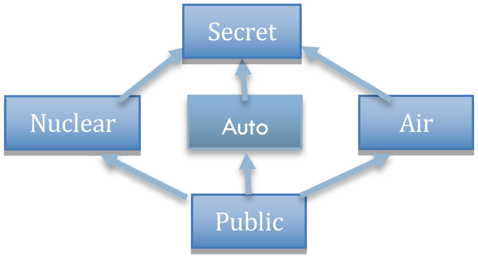
- For the following questions on information flows, assume the following lattice security policy. (1pt each)
  - 1.What is the label of the variable “e” after executing Line 6? Auto – combo (join) of *a* (Auto) and *b* (Public)
  - 2.What is the label of the variable “a” after executing Line 6? Auto – assigned labels are fixed
  - 3.What is the label of “c” after executing Line 8?
  - 4.What is the label of “d” after executing Line 10?
  - 5.Is the operation in Line 10 legal given the resultant information flows?



```
1: Int {Auto} a;

2: Int {Public} b, d;
3: Int c, e;

4: a = 7;
5: b = 2;
6: e = a+b;

7: if (a > 0) {
8:         c = b;
9: }
10: d = c;
```

Homework (Part III)

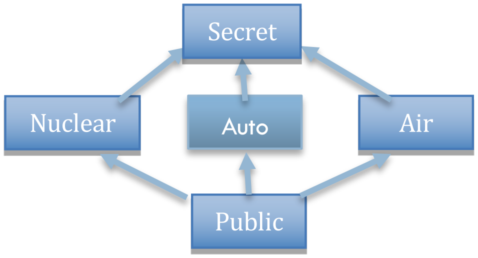
- For the following questions on information flows, assume the following lattice security policy. (1pt each)
  - 1.What is the label of the variable “e” after executing Line 6? Auto – combo (join) of *a* (Auto) and *b* (Public)
  - 2.What is the label of the variable “a” after executing Line 6? Auto – assigned labels are fixed
  - 3.What is the label of “c” after executing Line 8? Auto – implicit flow from a (Auto) to c (unlabeled)
  - 4.What is the label of “d” after executing Line 10?
  - 5.Is the operation in Line 10 legal given the resultant information flows?



```
1: Int {Auto} a;

2: Int {Public} b, d;
3: Int c, e;

4: a = 7;
5: b = 2;
6: e = a+b;

7: if (a > 0) {
8:        c = b;
9: }
10: d = c;
```

Homework (Part III)

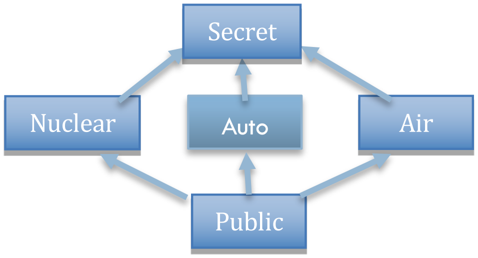
- For the following questions on information flows, assume the following lattice security policy. (1pt each)
  - 1.What is the label of the variable "e" after executing Line 6? Auto – combo (join) of *a* (Auto) and *b* (Public)
  - 2.What is the label of the variable "a" after executing Line 6? Auto – assigned labels are fixed
  - 3.What is the label of "c" after executing Line 8? Auto – implicit flow from a (Auto) to c (unlabeled) and b (Public)
  - 4.What is the label of "d" after executing Line 10? Public – assigned labels are fixed
  - 5.Is the operation in Line 10 legal given the resultant information flows?



```
1: Int {Auto} a;

2: Int {Public} b, d;
3: Int c, e;

4: a = 7;
5: b = 2;
6: e = a+b;

7: if (a > 0) {
8:        c = b;
9: }
10: d = c;
```

Homework (Part III)

- For the following questions on information flows, assume the following lattice security policy. (1pt each)
  - 1.What is the label of the variable “e” after executing Line 6? Auto – combo (join) of *a* (Auto) and *b* (Public)
  - 2.What is the label of the variable “a” after executing Line 6? Auto – assigned labels are fixed
  - 3.What is the label of “c” after executing Line 8? Auto – implicit flow from a (Auto) to c (unlabeled) and b (Public)
  - 4.What is the label of “d” after executing Line 10? Public – assigned labels are fixed
  - 5.Is the operation in Line 10 legal given the resultant information flows? No. "Auto" → "Public" flow is illegal



```
1: Int {Auto} a;

2: Int {Public} b, d;
3: Int c, e;

4: a = 7;
5: b = 2;
6: e = a+b;

7: if (a > 0) {
8:        c = b;
9: }
10: d = c;
```

Homework (Part IV)

- IV. Briefly describe the purpose the following instructions and what they do: (1.5 points each)
  - 1) call
  - 2) leave
  - 3) ret

Homework (Part IV)

- IV. Briefly describe the purpose the following instructions and what they do: (1.5 points each)
  - 1) call
  - Push the return address (address of the next instruction to the call instruction) onto the top of the stack and jump to the target address to execute (%eip changed to the target address specified in call instruction)
  - 2) leave
  - 3) ret

Homework (Part IV)

- IV. Briefly describe the purpose the following instructions and what they do: (1.5 points each)
  - 1) call
  - Push the return address (address of the next instruction to the call instruction) onto the top of the stack and jump to the target address to execute (%eip changed to the target address specified in call instruction)
  - 2) leave
  - Copies the frame pointer %ebp (register) to %esp (register), which releases the stack frame. The old frame pointer (at the top of the stack) is then popped (restored) into %ebp (register).
  - 3) ret

Homework (Part IV)

- IV. Briefly describe the purpose the following instructions and what they do: (1.5 points each)
  - 1) call
  - Push the return address (address of the next instruction to the call instruction) onto the top of the stack and jump to the target address to execute (%eip changed to the target address specified in call instruction)
  - 2) leave
  - Copies the frame pointer %ebp (register) to %esp (register), which releases the stack frame. The old frame pointer (at the top of the stack) is then popped (restored) into %ebp (register).
  - 3) ret
  - Pop the stack (i.e., value referenced by the %esp, which should be the return address) and put it in %eip (so the program jumps to the return address and start executing)

Homework (Part V)

```
1: char *p;

2: p = (char *) malloc(size);

3: len = snprintf(p, size, ``%s'', adv input);

4: free(p);
```

- 1. Is there a spatial or temporal memory error in this code? Why or why not.
- 2. Suppose the statements on lines 3 and 4 are switched? Explain any problem that could be caused.
  - NOTE: Assume the program is multi-threaded.

Homework (Part V)

```
1: char *p;

2: p = (char *) malloc(size);

3: len = snprintf(p, size, ‘‘%s’’, adv input);

4: free(p);
```

- 1. Is there a spatial or temporal memory error in this code? No.

  Why or why not.

  - **Spatial**: snprintf restricts the write to the ‘size’ allocated and ensures a null-terminator is placed – no spatial error

- 2. Suppose the statements on lines 3 and 4 are switched?

  Explain any problem that could be caused.

  - NOTE: Assume the program is multi-threaded.

Homework (Part V)

```
1: char *p;

2: p = (char *) malloc(size);

3: len = snprintf(p, size, ‘‘%s’’, adv input);

4: free(p);
```

- 1. Is there a spatial or temporal memory error in this code? No.

  Why or why not.

  - Spatial: snprintf restricts the write to the ‘size’ allocated and ensures a null-terminator is placed – no spatial error
  - Temporal: no use before initialization of ‘p’. No use after free of ‘p’. No temporal error

- 2. Suppose the statements on lines 3 and 4 are switched?

  Explain any problem that could be caused.

  - NOTE: Assume the program is multi-threaded.

Homework (Part V)

```
1: char *p;

2: p = (char *) malloc(size);

3: len = snprintf(p, size, ``%s’’, adv input);

4: free(p);
```

- 1. Is there a spatial or temporal memory error in this code?

  Why or why not.
  - Spatial: snprintf restricts the write to the ‘size’ allocated and ensures a null-terminator is placed – no spatial error
  - Temporal: no use before initialization of ‘p’. No use after free of ‘p’. No temporal error
  - Requirements of a legal C string are somewhat different than for a spatial error
- 2. Suppose the statements on lines 3 and 4 are switched?

  Explain any problem that could be caused.
  - NOTE: Assume the program is multi-threaded.

Homework (Part V)

```
1: char *p;

2: p = (char *) malloc(size);

3: len = snprintf(p, size, ‘‘%s’’, adv input);

4: free(p);
```

- 1. Is there a spatial or temporal memory error in this code? Why or why not.
- 2. Suppose the statements on lines 3 and 4 are switched? Explain any problem that could be caused.
  - NOTE: Assume the program is multi-threaded.
  - Could perform the write to memory location ‘p’ after it is freed. Why is that a problem?

Homework (Part V)

```
1: char *p;

2: p = (char *) malloc(size);

3: len = snprintf(p, size, ‘‘%s’’, adv input);

4: free(p);
```

- 1. Is there a spatial or temporal memory error in this code?

  Why or why not.
- 2. Suppose the statements on lines 3 and 4 are switched?

  Explain any problem that could be caused.
  - NOTE: Assume the program is multi-threaded.
  - Could perform the write to memory location ‘p’ after it is freed. Why is that a problem?
  - Other thread could allocate memory at p between statements 4 and 3, causing p to be used to write to another object.

Homework (Part VI)

```
char *getline()
{
    char buf[8];
    char *result;
    scanf("%s", buf);
    result = malloc(strlen(buf));
    strcpy(result, buf);
    return result;
}
```

```
08048524 <getline>:
8048524: 55 push %ebp
8048525: 89 e5 mov %esp,%ebp
8048527: 83 ec 10 sub $0x10,%esp
804852a: 56 push %ecx
804852b: 53 push %edi
Diagram stack at this point
804852c: 83 c4 f4 add $0xfffffff4,%esp
804852f: 8d 5d f8 lea 0xfffffff8(%ebp),%ebx
8048532: 53 push %ebx
8048533: e8 74 fe ff ff call 80483ac <_init+0
Modify diagram to show values at this point
```

- Procedure getline is called with the return address equal to 0x804ab62, register %ebp equal to 0xbffffc90, register %edi equal to 0x3, and register %ecx equal to 0x8. You type in the string “01234567890123”.

Homework (Part VI)

```
char *getline()
{
    char buf[8];
    char *result;
    scanf("%s", buf);
    result = malloc(strlen(buf));
    strcpy(result, buf);
    return result;
}
```

```
1 08048524 <getline>:
2 8048524: 55 push %ebp
3 8048525: 89 e5 mov %esp,%ebp
4 8048527: 83 ec 10 sub $0x10,%esp
5 804852a: 56 push %ecx
6 804852b: 53 push %edi
Diagram stack at this point
7 804852c: 83 c4 f4 add $0xfffffff4,%esp
8 804852f: 8d 5d f8 lea 0xfffffff8(%ebp),%ebx
9 8048532: 53 push %ebx
10 8048533: e8 74 fe ff ff call 80483ac <_init+0
Modify diagram to show values at this point
```

- Procedure getline is called with the return address equal to 0x804ab62, register %ebp equal to 0xbffffc90, register %edi equal to 0x3, and register %ecx equal to 0x8. You type in the string "01234567890123".
- (1) Fill in the diagram below indicating as much as you can about the stack just after executing the instruction at line 6 in the disassembly.

Homework (Part VI)

```
+-------------+
|08 04 ab 62  | Return Address
+-------------+
|bf ff fc 90  | Saved %ebp
+-------------+
| buf[4-7)    |
+-------------+
| buf[0-3)    |
+-------------+
|result [4-7] |
+-------------+
|result [0-3] |
+-------------+
| 00 00 00 08 | Saved %ecx
+-------------+
| 00 00 00 03 | Saved %edi,
%esp references this location
+-------------+
```

```
1 08048524 <getline>:
2 8048524: 55 push %ebp
3 8048525: 89 e5 mov %esp,%ebp
4 8048527: 83 ec 10 sub $0x10,%esp
5 804852a: 56 push %ecx
6 804852b: 53 push %edi
Diagram stack at this point
7 804852c: 83 c4 f4 add $0xfffffff4,%esp
8 804852f: 8d 5d f8 lea 0xfffffff8(%ebp),%ebx
9 8048532: 53 push %ebx
10 8048533: e8 74 fe ff ff call 80483ac <_init+0
Modify diagram to show values at this point
```

- Procedure getline is called with the return address equal to 0x804ab62, register %ebp equal to 0xbffffc90, register %edi equal to 0x3, and register %ecx equal to 0x8. You type in the string “01234567890123”.
  - (1) Fill in the diagram below indicating as much as you can about the stack just after executing the instruction at line 6 in the disassembly.

Homework (Part VI)

```
4 char *getline()
5 {
6     char buf[8];
7     char *result;
8     scanf("%s", buf);
9     result = malloc(strlen(buf));
10    strcpy(result, buf);
11    return result;
12 }
```

```
1 08048524 <getline>:
2 8048524: 55 push %ebp
3 8048525: 89 e5 mov %esp,%ebp
4 8048527: 83 ec 10 sub $0x10,%esp
5 804852a: 56 push %ecx
6 804852b: 53 push %edi
Diagram stack at this point
7 804852c: 83 c4 f4 add $0xfffffff4,%esp
8 804852f: 8d 5d f8 lea 0xfffffff8(%ebp),%ebx
9 8048532: 53 push %ebx
10 8048533: e8 74 fe ff ff call 80483ac <_init+0
Modify diagram to show values at this point
```

- Procedure getline is called with the return address equal to 0x804ab62, register %ebp equal to 0xbffffc90, register %edi equal to 0x3, and register %ecx equal to 0x8. You type in the string “01234567890123”.
- (2) Modify your diagram to show the effect of the call to scanf (line 10) on the part of the stack shown.

Homework (Part VI)

```
+-------------+
| 08 00 33 32 | Return Address
+-------------+
| 31 30 39 38 | saved %ebp
+-------------+
| 37 36 35 34 | buf[4-7]
+-------------+
| 33 32 31 30 | buf[0-3]
+-------------+
```

```
1 08048524 <getline>:
2 8048524: 55 push %ebp
3 8048525: 89 e5 mov %esp,%ebp
4 8048527: 83 ec 10 sub $0x10,%esp
5 804852a: 56 push %ecx
6 804852b: 53 push %edi
Diagram stack at this point
7 804852c: 83 c4 f4 add $0xfffffff4,%esp
8 804852f: 8d 5d f8 lea 0xfffffff8(%ebp),%ebx
9 8048532: 53 push %ebx
10 8048533: e8 74 fe ff ff call 80483ac <_init+0
Modify diagram to show values at this point
```

- □ Procedure getline is called with the return address equal to 0x804ab62, register %ebp equal to 0xbffffc90, register %edi equal to 0x3, and register %ecx equal to 0x8. You type in the string “012345678901 23”.
- □ (2) Modify your diagram to show the effect of the call to scanf (line 10) on the part of the stack shown.

Homework (Part VI)

```
+-------------+
| 08 00 33 32 | Return Address
+-------------+
| 31 30 39 38 | saved %ebp
+-------------+
| 37 36 35 34 | buf[4-7]
+-------------+
| 33 32 31 30 | buf[0-3]
+-------------+
```

```
08048524 <getline>:
8048524: 55 push %ebp
8048525: 89 e5 mov %esp,%ebp
8048527: 83 ec 10 sub $0x10,%esp
804852a: 56 push %ecx
804852b: 53 push %edi
Diagram stack at this point
804852c: 83 c4 f4 add $0xfffffff4,%esp
804852f: 8d 5d f8 lea 0xfffffff8(%ebp),%ebx
8048532: 53 push %ebx
8048533: e8 74 fe ff ff call 80483ac <_init+0
Modify diagram to show values at this point
```

- □ Procedure getline is called with the return address equal to 0x804ab62, register %ebp equal to 0xbffffc90, register %edi equal to 0x3, and register %ecx equal to 0x8. You type in the string “01234567890123”.
- □ (3) To what address does the program attempt to return (when getline completes)?

Homework (Part VI)

```
+-------------+
| 08 00 33 32 | Return Address
+-------------+
| 31 30 39 38 | saved %ebp
+-------------+
| 37 36 35 34 | buf[4-7]
+-------------+
| 33 32 31 30 | buf[0-3]
+-------------+
```

```
08048524 <getline>:
8048524: 55 push %ebp
8048525: 89 e5 mov %esp,%ebp
8048527: 83 ec 10 sub $0x10,%esp
804852a: 56 push %ecx
804852b: 53 push %edi
Diagram stack at this point
804852c: 83 c4 f4 add $0xfffffff4,%esp
804852f: 8d 5d f8 lea 0xfffffff8(%ebp),%ebx
8048532: 53 push %ebx
8048533: e8 74 fe ff ff call 80483ac <_init+0
Modify diagram to show values at this point
```

- Procedure getline is called with the return address equal to 0x804ab62, register %ebp equal to 0xbffffc90, register %edi equal to 0x3, and register %ecx equal to 0x8. You type in the string “01234567890123”.
- (3) To what address does the program attempt to return (when getline completes)?
  - 0x8003332

Homework (Part VI)

```
+-------------+
| 08 00 33 32 | Return Address
+-------------+
| 31 30 39 38 | saved %ebp
+-------------+
| 37 36 35 34 | buf[4-7]
+-------------+
| 33 32 31 30 | buf[0-3]
+-------------+
```

```
08048524 <getline>:
8048524: 55 push %ebp
8048525: 89 e5 mov %esp,%ebp
8048527: 83 ec 10 sub $0x10,%esp
804852a: 56 push %ecx
804852b: 53 push %edi
Diagram stack at this point
804852c: 83 c4 f4 add $0xfffffff4,%esp
804852f: 8d 5d f8 lea 0xfffffff8(%ebp),%ebx
8048532: 53 push %ebx
8048533: e8 74 fe ff ff call 80483ac <_init+0
Modify diagram to show values at this point
```

- □ Procedure getline is called with the return address equal to 0x804ab62, register %ebp equal to 0xbffffc90, register %edi equal to 0x3, and register %ecx equal to 0x8. You type in the string “01234567890123”.
- □ (4) What register(s) have corrupted value(s) when getline returns?

Homework (Part VI)

```
char *getline()
{
    char buf[8];
    char *result;
    scanf(“%s”, buf);
    result = malloc(strlen(buf));
    strcpy(result, buf);
    return result;
}
```

```
08048524 <getline>:
8048524: 55 push %ebp
8048525: 89 e5 mov %esp,%ebp
8048527: 83 ec 10 sub $0x10,%esp
804852a: 56 push %ecx
804852b: 53 push %edi
Diagram stack at this point
804852c: 83 c4 f4 add $0xfffffff4,%esp
804852f: 8d 5d f8 lea 0xfffffff8(%ebp),%ebx
8048532: 53 push %ebx
8048533: e8 74 fe ff ff call 80483ac <_init+0
Modify diagram to show values at this point
```

- Procedure getline is called with the return address equal to 0x804ab62, register %ebp equal to 0xbffffc90, register %edi equal to 0x3, and register %ecx equal to 0x8. You type in the string “01234567890123”.
- (4) What register(s) have corrupted value(s) when getline returns?
  - The saved value of register %ebp was changed to 0x31303938, and this will be loaded into %ebp before getline returns. %eip is corrupted because the return of getline() will effectively pop the corrupted return address into %eip.

Homework (Part VI)

```
char *getline()
{
    char buf[8];
    char *result;
    scanf(“%s”, buf);
    result = malloc(strlen(buf));
    strcpy(result, buf);
    return result;
}
```

```
08048524 <getline>:
8048524: 55 push %ebp
8048525: 89 e5 mov %esp,%ebp
8048527: 83 ec 10 sub $0x10,%esp
804852a: 56 push %ecx
804852b: 53 push %edi
Diagram stack at this point
804852c: 83 c4 f4 add $0xfffffff4,%esp
804852f: 8d 5d f8 lea 0xfffffff8(%ebp),%ebx
8048532: 53 push %ebx
8048533: e8 74 fe ff ff call 80483ac <_init+0
Modify diagram to show values at this point
```

- Procedure getline is called with the return address equal to 0x804ab62, register %ebp equal to 0xbffffc90, register %edi equal to 0x3, and register %ecx equal to 0x8. You type in the string “01234567890123”.
- (5) Besides the potential for buffer overflow, what two other things are wrong with the code for getline?
  - The call to malloc should have had strlen(buf)+1 as its argument, and it should also check that the returned value is non-null. Other legit issues will be considered.

Quiz (ROP #1)

a_1: pop ebx; ret

a_2: pop eax; ret

a_3: mov eax, (ebx); ret

a_4: mov ebx, (eax); ret

a_5: add eax, (ebx); ret

a_6: push ebx; ret

a_7: pop esp; ret

Known Gadgets

Draw a stack diagram for a ROP exploit to **store the value 0xBBBBBBB+1 into address 0xAAAAAAA**

Quiz (ROP #1)

a_1: pop ebx; ret

a_2: pop eax; ret

a_3: mov eax, (ebx); ret

a_4: mov ebx, (eax); ret

a_5: add eax, (ebx); ret

a_6: push ebx; ret

a_7: pop esp; ret

Known Gadgets

Draw a stack diagram for a ROP exploit to **store the value 0xBBBBBBB+1 into address 0xAAAAAAA**

A2 | 0x1 | A1 | 0xA | A3 | A2 | 0xB | A5 |

low high

Quiz (ROP #2)

a_1: pop ebx; ret

a_2: pop eax; ret

a_3: mov eax, (ebx); ret

a_4: mov ebx, (eax); ret

a_5: add eax, (ebx); ret

a_6: push ebx; ret

a_7: pop esp; ret

Known Gadgets

Draw a stack diagram for a ROP exploit to **store the value 0xBBBBBBB+1 into address 0xAAAAAAA – then execute from 0xBBBBBBB+1**

Quiz (ROP #2)

a_1: pop ebx; ret

a_2: pop eax; ret

a_3: mov eax, (ebx); ret

a_4: mov ebx, (eax); ret

a_5: add eax, (ebx); ret

a_6: push ebx; ret

a_7: pop esp; ret

Known Gadgets

Draw a stack diagram for a ROP exploit to **store the value 0xBBBBBBB+1 into address 0xAAAAAAA – then execute from 0xBBBBBBB+1**

A2 | 0x1 | A1 | 0xA | A3 | A2 | 0xB | A5 | A7 | 0xA

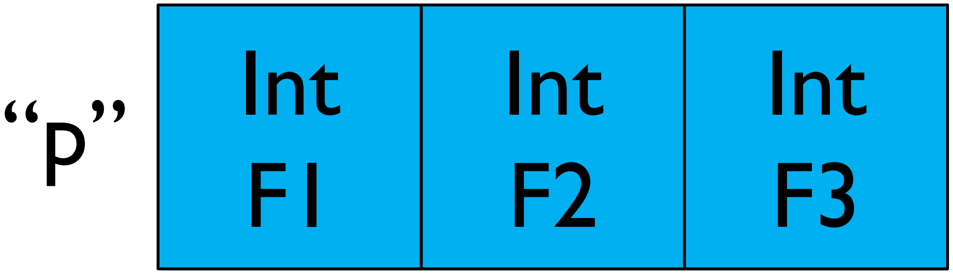
low high

Type Errors

- Errors that permit access to memory according to a multiple, incompatible formats
  - These are called type errors
  - Access using a different “type” than used to format the memory
- Most of these errors are permitted by simple programming flaws
  - Of the sort that you are not taught to avoid
  - Let’s see how such errors can be avoided
- Some of the changes are rather simple

Exploiting Type Errors

- “p” is assigned to an object of type t1



“p”

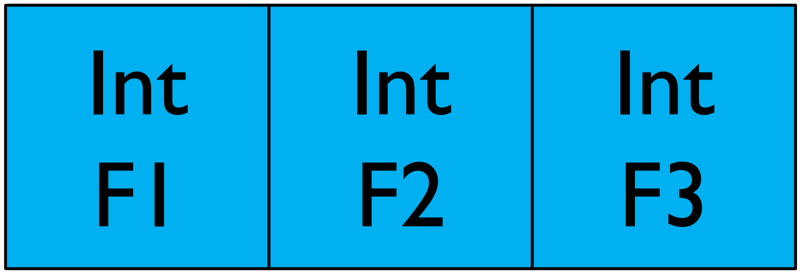
Int F1 | Int F2 | Int F3

- Only memory large enough for t1 is allocated

Exploiting Type Errors

- “p” is assigned to an object of type t1

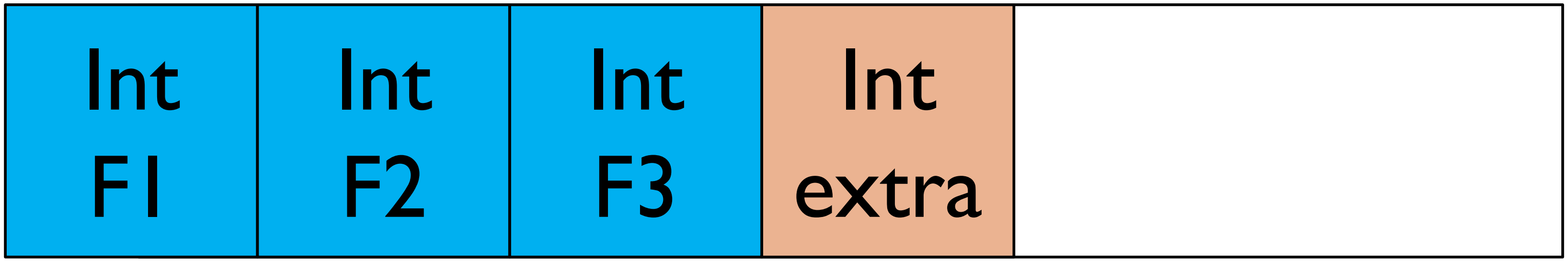
“p”



Int F1	Int F2	Int F3

- But, if we assign a pointer of type t2 to the object

“q”



Int F1	Int F2	Int F3	Int extra	

- This is what can be referenced by “q”
  - ”q” of type t2 thinks it is referencing a larger region

Memory Error Defenses

- We have discussed some
  - Canaries
  - Address Space Layout Randomization
  - Data Execution Protection (No Execute)
- How do these defenses work? Review

Memory Error Defenses

- We have discussed some
  - Canaries
  - Address Space Layout Randomization
  - Data Execution Protection (No Execute)
- These defenses do not prevent ROP attacks
  - Why not?

Memory Error Defenses

- We have discussed some
  - Canaries
  - Address Space Layout Randomization
  - Data Execution Protection (No Execute)
- These defenses do not prevent ROP attacks
  - Why not?
    - Bypass canaries and ASLR
      - Disclose canary values on stack
      - Disclose stack pointer values (EBP)
    - DEP/NX does not prevent execution of code memory

Conclusions

- Structure of exam
  - Multiple choice – fill in blank
  - Short answer – Conceptual questions
    - May be more than one question – be sure to answer all
  - Constructions – Problem solving
    - Multiple sub-parts
- Time management – answer ones you know
- Topics – Covered in these slides
  - Those in this review may be on the exam (up to ROP)
- Readings – good to know more – different angle

Questions