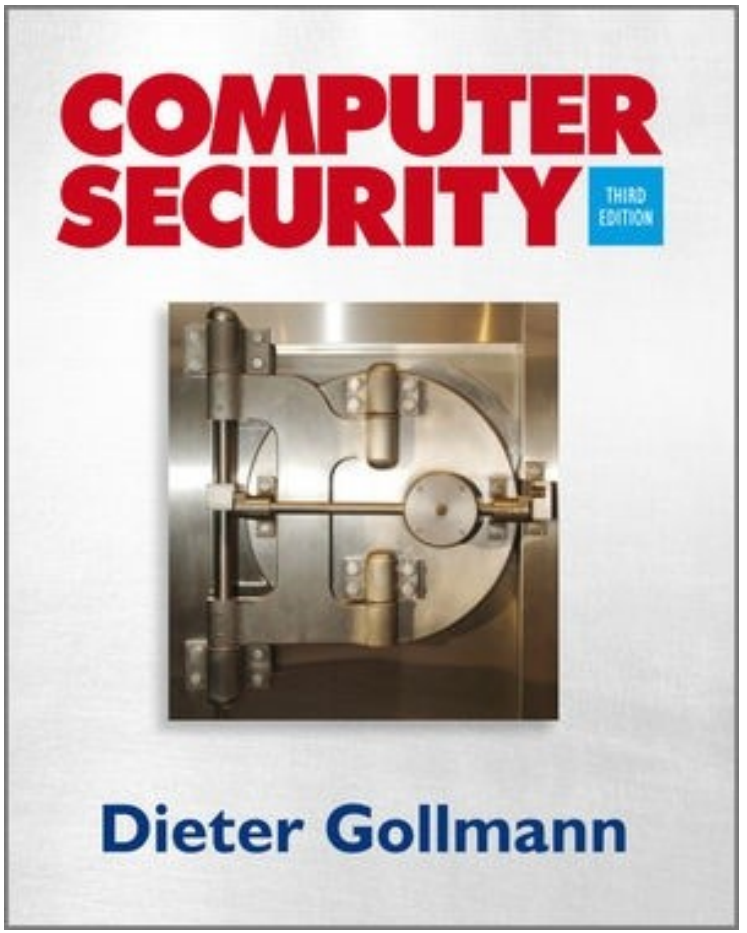
Computer Security 3e

Dieter Gollmann



Chapter 10: Software Security

Secure Software

- Software is secure if it can handle intentionally malformed input; the attacker picks (the probability distribution of) the inputs.
- Secure software: Protect the integrity of the runtime system.
- Secure software ≠ software with security features.
- Networking software is a popular target:
  - Intended to receive external input.
  - May construct instructions dynamically from input.
  - May involve low level manipulations of buffers.

Security & Reliability

- Reliability deals with accidental failures: Failures are assumed to occur according to some given probability distribution.
- The probabilities for failures is given first, then the protection mechanisms are constructed.
- To make software more reliable, it is tested against typical usage patterns: “It does not matter how many bugs there are, it matters how often they are triggered”.

Security & Reliability

- In security, the defender has to move first; the attacker picks inputs to exploit weak defences.
- To make software more secure, it has to be tested against “untypical” usage patterns (but there are typical attack patterns).
- On a PC, you are in control of the software components sending inputs to each other.
- On the Internet, hostile parties can provide input: Do not “trust” your inputs.

Agenda

- Dangers of abstraction
- Input validation
- Integers
- Buffer overflows
- Race conditions
- Defences: Prevention – Detection – Reaction

Preliminaries

- When writing code, programmers use elementary concepts like character, variable, array, integer, data & program, address (resource locator), atomic transaction, …
- These concepts have abstract meanings.
- For example, integers are an infinite set with operations ‘add’, ‘multiply’, ‘less or equal’, …
- To execute a program, we need concrete implementations of these concepts.

Benefits of Abstraction

- Abstraction (hiding ‘unnecessary’ detail) is an extremely valuable method for understanding complex systems.
- We don’t have to know the inner details of a computer to be able to use it.
- We can write software using high level languages and graphical methods.
- Anthropomorphic images explain what computers do (send mail, sign document).

Dangers of Abstraction

- Software security problems typically arise when concrete implementation and the abstract intuition diverge.
- We will explore a few examples:
  - Address (location)
  - Character
  - Integer
  - Variable (buffer overflows)

Input Validation

- An application wants to give users access only to files in directory **`A/B/C/`**.
- Users enter filename as **`input`**; full file name constructed as **`A/B/C/input`**.
- Attack: use **`../`** a few times to step up to root directory first; e.g. get password file with input **`/../../../../etc/passwd`**.
- Countermeasure: input validation, filter out **`../`** (but as you will see in a moment, life is not that easy).
- Do not trust your inputs.

Programming with Integers

- In mathematics integers form an infinite set.
- On a computer systems, integers are represented in binary.
- The representation of an integer is a binary string of fixed length (precision), so there is only a finite number of “integers”.
- Programming languages: signed & unsigned integers, short & long (& long long) integers, …

What will happen here?

```
int i = 1;
while (i > 0)
{
i = i * 2;
}
```

Computing with Integers

- Unsigned 8-bit integers

  255 + 1 = 0 16 ξ 17 = 16

  0 – 1 = 255

- Signed 8-bit integers

  127 + 1 = -128 -128/-1 = -1

- In mathematics: a + b >= a for b >= 0
- As you can see, such obvious “facts” are no longer true.

Two's Complement

- Signed integers are usually represented as 2's complement numbers.
- Most significant bit (sign bit) indicates the sign of the integer:
  - If sign bit is zero, the number is positive.
  - If sign bit is one, the number is negative.
- Positive numbers given in normal binary representation.
- Negative numbers are represented as the binary number that when added to a positive number of the same magnitude equals zero.

Code Example 2

- OS kernel system-call handler; checks string lengths to defend against buffer overruns.

```
char buf[128];
combine(char *s1, size_t len1,
        char *s2, size_t len2)
{
if (len1 + len2 + 1 <= sizeof(buf)) {
strncpy(buf, s1, len1);
strncat(buf, s2, len2);
}
}
```

len1 < sizeof(buf)

len2 = 0xffffffff

len2 + 1 = $2^{32}-1$ + 1

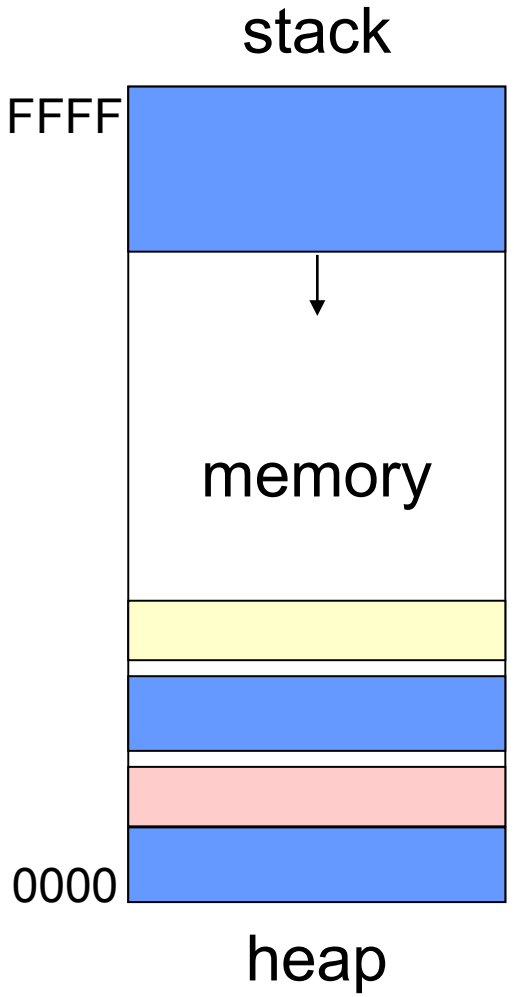
= 0 mod 2^{32}

strncat will be executed

Memory Allocation

Memory configuration

- Stack: contains return address, local variables and function arguments; relatively easy to decide in advance where a particular buffer will be placed on the stack.
- Heap: dynamically allocated memory; more difficult but not impossible to decide in advance where a particular buffer will be placed on the heap.



Variables

- Buffer: concrete implementation of a variable.
- If the value assigned to a variable exceeds the size of the allocated buffer, memory locations not allocated to this variable are overwritten.
- If the memory location overwritten had been allocated to some other variable, the value of that other variable is changed.
- Depending on circumstances, an attacker can change the value of a sensitive variable A by assigning a deliberately malformed value to some other variable B.

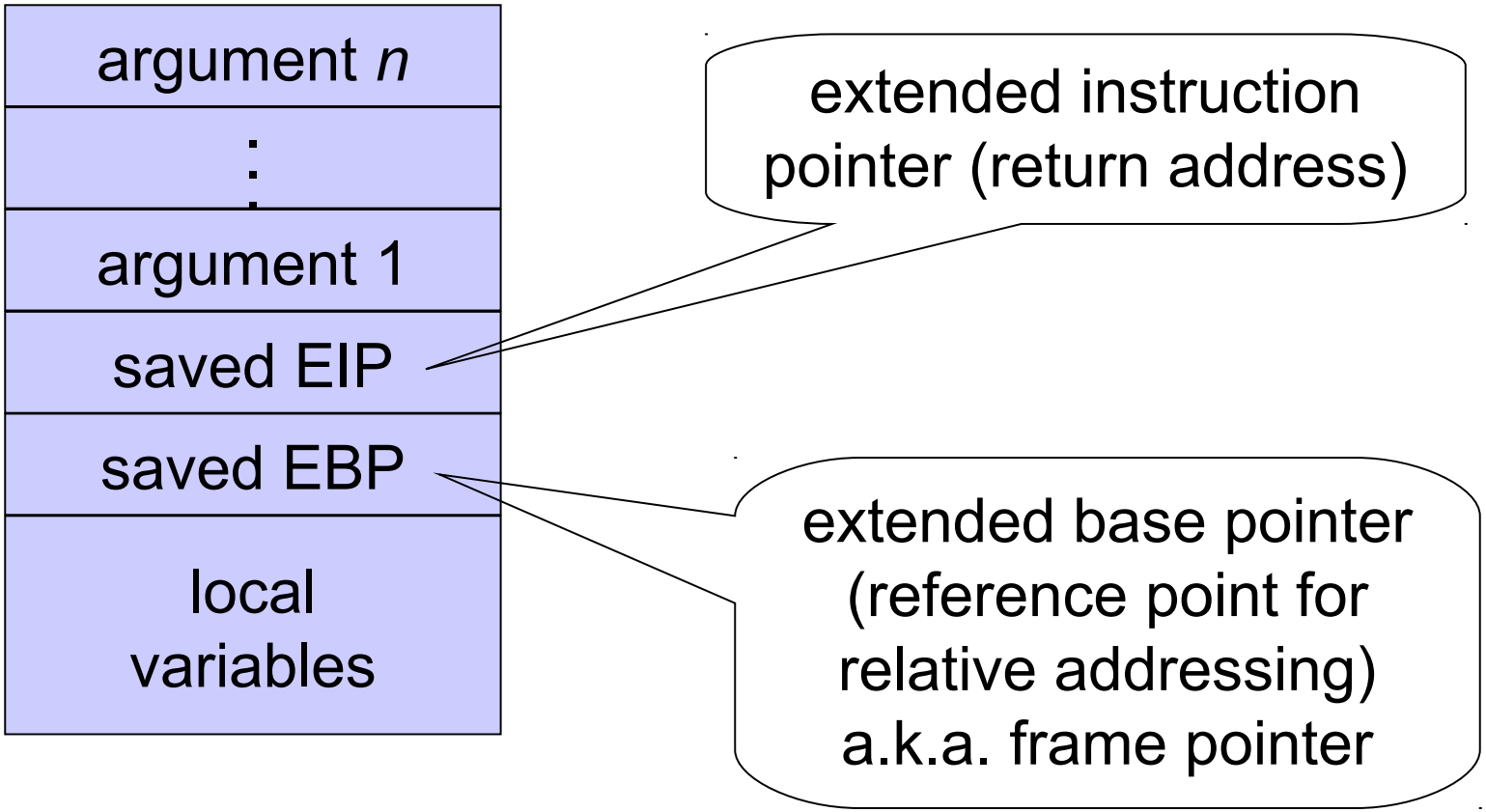
Buffer Overruns

- Unintentional buffer overruns crash software, and have been a focus for reliability testing.
- Intentional buffer overruns are a concern if an attacker can modify security relevant data.
- Attractive targets are return addresses (specify the next piece of code to be executed) and security settings.
- In languages like C or C++ the programmer allocates and de-allocates memory.
- Type-safe languages like Java guarantee that memory management is ‘error-free’.

System Stack

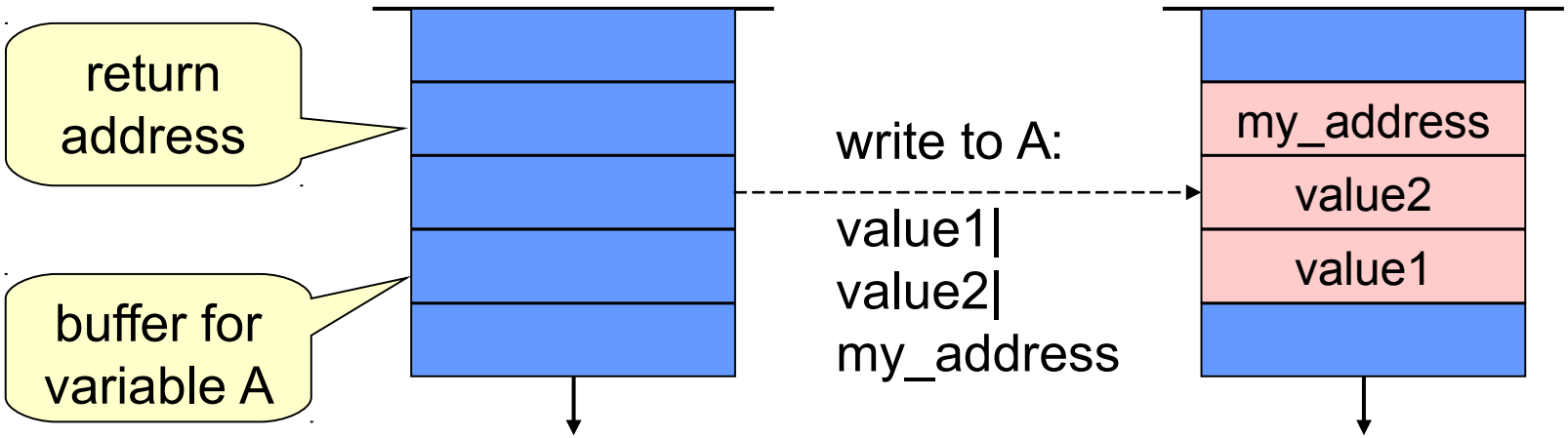
- Function call: stack frame containing function arguments, return address, statically allocated buffers pushed on the stack.
- When the call returns, execution continues at the return address specified.
- Stack usually starts at the top of memory and grows downwards.
- Layout of stack frames is reasonably predictable.

Stack Frame – Layout



Stack-based Overflows

- Find a buffer on the runtime stack of a privileged program that can overflow the return address.
- Overwrite the return address with the start address of the code you want to execute.
- Your code is now privileged too.



Code Example

- Declare a local short string variable

```
char buffer[80];
```

use the standard C library routine call

```
gets(buffer);
```

to read a single text line from standard input and save it into buffer.

- Works fine for normal-length lines, but corrupts the stack if the input is longer than 79 characters.
- Attacker loads malicious code into buffer and redirects return address to start of attack code.

Shellcode

- Overwrite return address so that execution jumps to the attack code ('shellcode').
- Where to put the shellcode?
- Shellcode may be put on the stack as part of the malicious input; a.k.a. argv[]-method.
  - To guess the location, guess distance between return address and address of the input containing the shellcode.
- Details e.g. in Smashing the Stack for Fun and Profit.
- *return-to-libc* method: attack calls system library; change to control flow, but no shellcode inserted.

Overwriting Pointers

- Modify return address with buffer overrun on stack.
  - Attacker can fairly easily guess the location of this pointer relative to a vulnerable buffer.
  - Defender knows which target to protect.
- More powerful attack: overwrite arbitrary pointer with an arbitrary value.
- More targets, hence more difficult to defend against.
- Attacker does not even have to overwrite the pointer!
- Attacker can lure the operating system into reading malformed input and then do the job for the attacker.

Type Confusion

Type Safety – Java

- Type safety (memory safety): programs cannot access memory in inappropriate ways.
- Each Java object has a class; only certain operations are allowed to manipulate objects of that class.
- Every object in memory is labelled with a class tag.
- When a Java program has a reference to an object, it has internally a pointer to the memory address storing the object.
- Pointer can be thought of as tagged with a type that says what kind of object the pointer is pointing to.

Type Confusion

- Dynamic type checking: check the class tag when access is requested.
- Static type checking: check all possible executions of the program to see whether a type violation could occur.
- If there is a mistake in the type checking procedure, a malicious applet might be able to launch a type confusion attack by creating two pointers to the same object-with incompatible type tags.

Type Confusion

- Assume the attacker manages to let two pointers point to the same location.

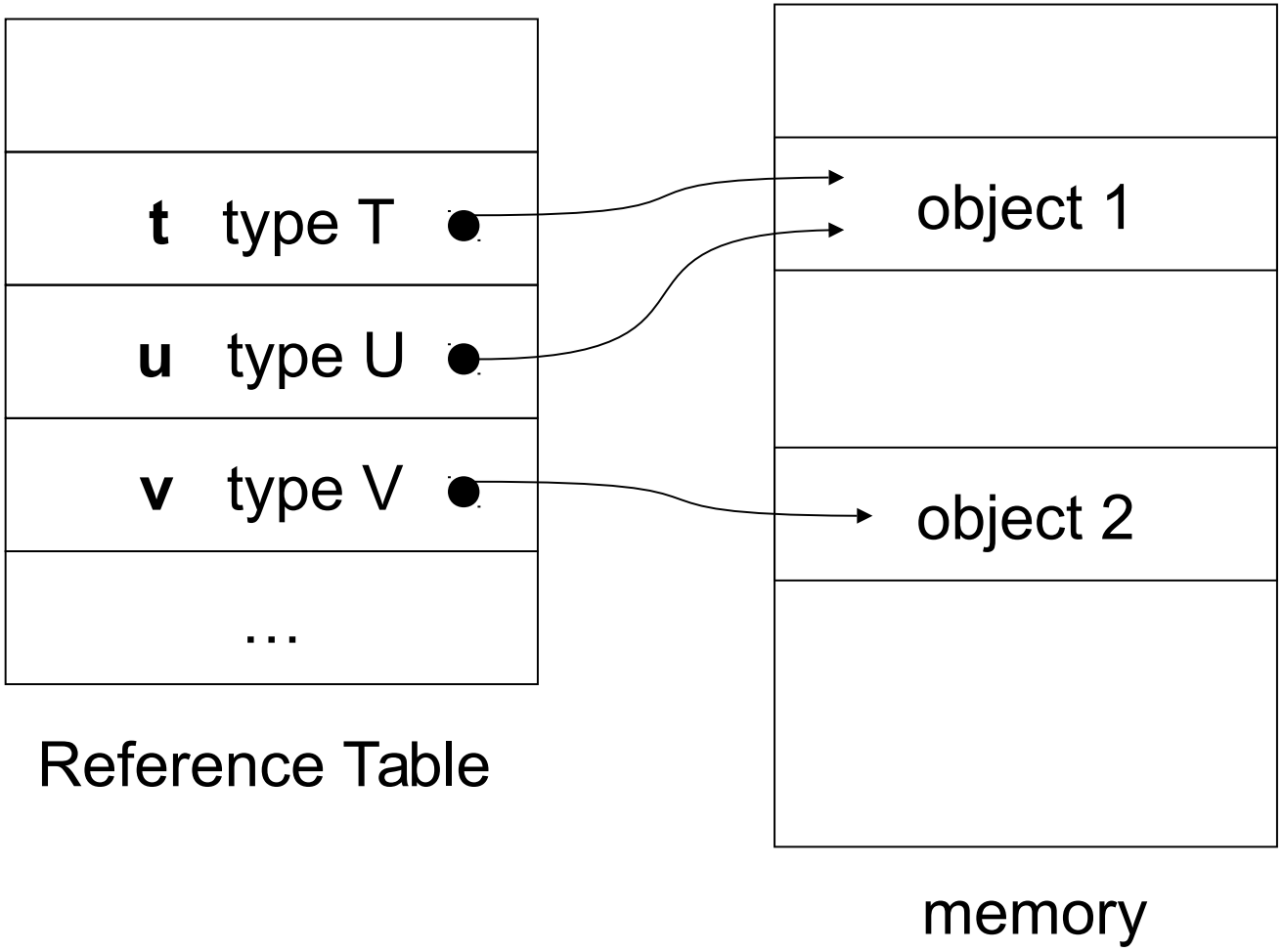
```
class T {
   SecurityManager x;
}

class U {
   MyObject x;
}
```

class definitions

```
T t = the pointer tagged T;
U u = the pointer tagged U;
t.x = System.getSecurity();
MyObject m = u.x;
```

malicious applet

Type Confusion



Data and Code

SQL Injection

- Strings in SQL commands placed between single quotes.
- Example query from SQL database:

  $sql = "SELECT * FROM client WHERE name= '$name'"
- Intention: insert legal user name like 'Bob' into query.
- Attack enters as user name: Bob' OR 1=1 --
- SQL command becomes

  SELECT * FROM client WHERE name = Bob' OR 1=1--
- Because 1=1 is TRUE, name = Bob OR 1=1 is TRUE, and the entire client database is selected; -- is a comment erasing anything that would follow.

SQL Injection

- Countermeasures against code injection:
  - Input validation: make sure that no unsafe input is used in the construction of a command.
  - Change the modus operandi: modify the way commands are constructed and executed so that unsafe input can do no harm.
- Parametrized queries with bound parameters (DBI placeholders in Perl) follow the second approach.
  - Scripts compiled with placeholders instead of user input.
  - Commands called by transmitting the name of the procedure and the parameter values.
    - During execution, placeholders are replaced by the actual input.