Arc-injection Attack

Wei Wang

Sometimes Code-injection can be Difficult

- Sometimes the code-to-injection is too long for the buffer
- Sometimes the code-to-injection is too complicated to write
  - e.g., for a shellcode, the attacker has to inject code to setuid, redirect stdin and start a shell
- Stack is not executable in modern systems
- As a result, code-injection is not always feasible

Arc-injection / Return-to-Libc Attack

- However, a program already has a lot of code that are useful to the attacker
- Particularly, libc has a lot of useful functions
  - libc is the “C standard library”
  - libc provides macros, type definitions, and functions for tasks like string handling, mathematical computations, input/output processing, memory allocation and several other operating system services, e.g., printf

Arc-injection / Return-to-Libc Attack cont'd

- The goal of arc-injection is to inject a jump (e.g., a return address) to redirect the execution flow to some other existing code in the memory, i.e, inject an arc into the execution
- Because libc is usually used for this type of attack, arc-injection is also called return-to-libc attack
  - Strictly speaking, any jump-to-existing-code exploit is an arc-injection attack

An Example of Arc-injection

- Goal: to create a bash shell without injecting any code
- The vulnerable code to exploit:

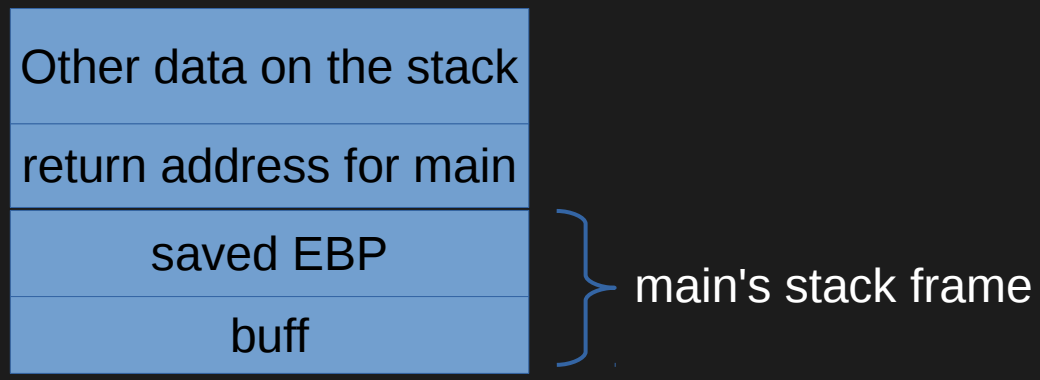
```
Int main(int argc, char *argv[]){
    char buf[4];

    strcpy(buf, argv[1]);

    return 0;
}
```

An Example of Arc-injection cont'd

- The stack frame of main



An Example of Arc-injection cont'd

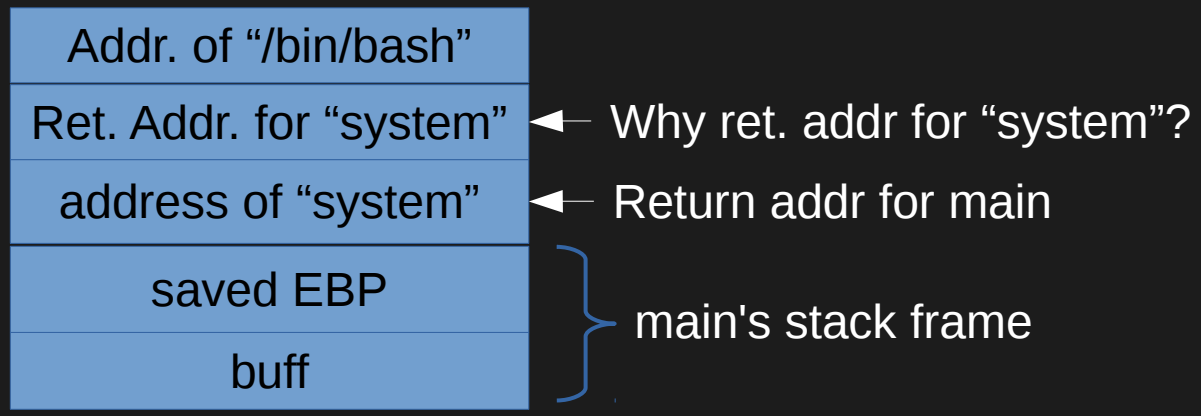
- We want to make a libc function call: system(“/bin/bash”);
- “system” is libc function that executes a command
- The command is passed to “system” as its first parameter

An Example of Arc-injection cont'd

- If the attacker sets the return address to the address of “system” function, and,
- If the attacker sets the first parameter to be address of the string of “/bin/bash”
- The attacker can then complete the attack

An Example of Arc-injection cont'd

- In other words, this is the stack the attacker wants after the buffer is overflew:



- Look! How simple it is compared to a shellcode injection!

An Example of Arc-injection cont'd

- The attacker needs to two things to make the attack possible
  - The address of “system”
  - A string of “/bin/bash” and its address
- For the address of “system”
  - In GDB, do “print system”

An Example of Arc-injection cont'd

- For the string of “/bin/bash”
  - Most of the time, a program has an environment variable of “SHELL=/bin/bash”
  - The attacker can directly use this environment variable
  - In GDB, use the command “x/s *((char **)environ+*i*)” to find the address of an environment variable, where *i* is the *i*'th environment variable
  - The attacker can always inject his environment variable as well

An Example of Arc-injection cont'd

- There are only 25 bytes to generate, which is considerably smaller than a code-injection, which requires about 50 bytes
- Let's try it out.

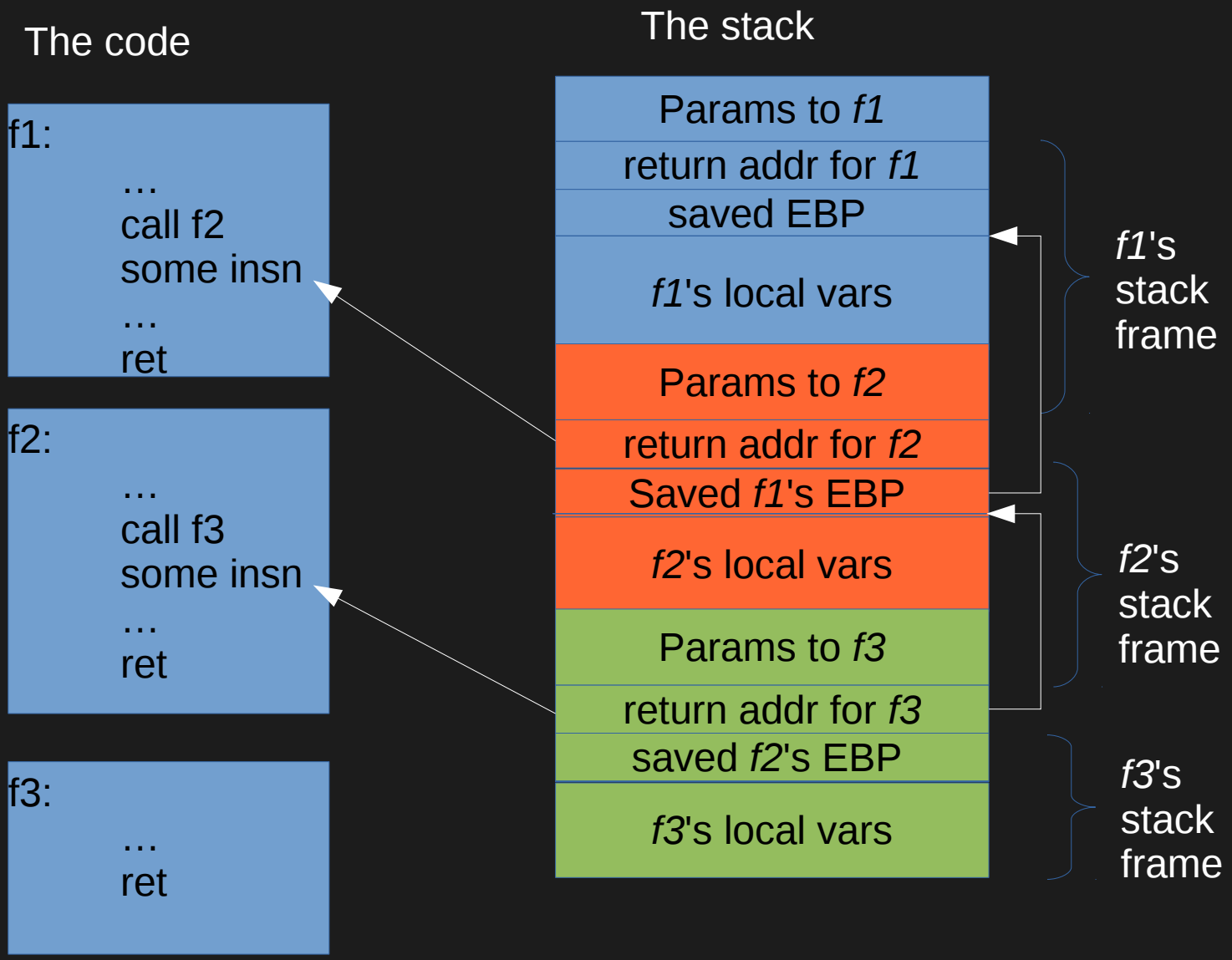
Making Multiple Libc Calls

- To launch a full-scaled attack, the attacker usually needs to make several libc calls
  - e.g., first a “setuid” to pretend to be root, second an “freopen” to redirect the stdin, at last a “system” to create a shell
- To execute multiple functions, the attacker has to supplied valid stack frame for each function, and use return addresses to chain them together

Making Multiple Libc Calls cont'd

- Image, three functions, *f1*, *f2* and *f3; f1* calls *f2, f2* calls *f3*
- What does the stack look like?

Making Multiple Libc Calls cont'd



Making Multiple Libc Calls cont'd

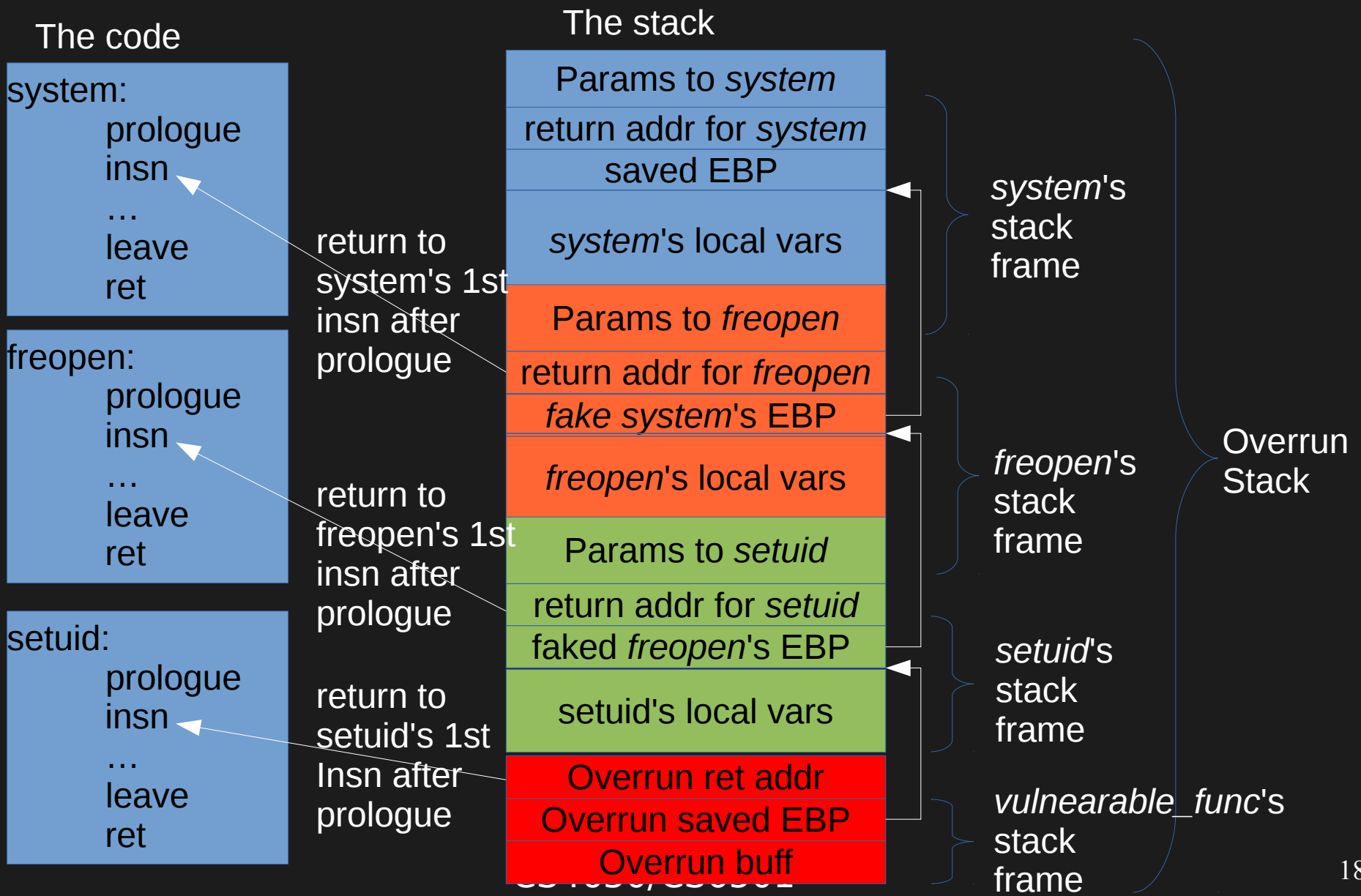
- Similarly, we can construct the attack string to build fake stack frames for multiple libc calles.
- For example, if we want to makes threes calls to setuid, freopen, system, the fake stack frames should looks like the one in the following slide
  - Note that setuid finishes first, then freopen finishes, at last system finishes

Making Multiple Libc Calls cont'd

- Another trick for making multiple libc calls: for each libc function, do not jump to the beginning of the function; instead, jump to the code right after the function prologue
  - The attacker will supply fake stack frame through the attack string, so no need to call the function's prologue to setup the stack frame

```
Setuid:
      # function prologue
      pushl %ebp
      movl %esp, %ebp
      Subl %esp, $some_number # allocate locals
      # end of prologue, jump to here to execute
      … … … … some codes … …
```

Making Multiple Libc Calls cont'd



Making Multiple Libc Calls cont'd

- In summary, to make multiple libc function calls, the attacker has to link the function calls using return addresses
- Attacker may also have to supply stack frames and stack frame pointers, depending on the behavior of the libc function.
  - The location and content of attacker supplied stack frames may get more complex if a libc function wants to manage stack frame itself

Arc-injection Summary

- Due to difficult of injecting large number of instructions and the difficulty of executing code on stack, attackers resort to existing code in memory to attack.
- Therefore, code-injection/stack-code-execution is no longer required
- Because of the large number of valuable functions in libc, usually libc's code is used by the attack. Consequently, arc-injection is also knowns as return-to-libc attack