Memory Corruption

Mustakimur R. Khandaker

Terms

Vulnerability (Bugs): malicious functionality that extends primary, intended design.

- Vulnerabilities may remain invisible until they are exploited.

Exploitation (Attack): identifying input to leverage the vulnerability and either take control of the system or leak sensitive information.

- Software security = risk management.
  - Identify vulnerabilities.
  - Patch vulnerable code.

Management:

- Software auditing (review).
- Security test (automatic).
- Patch development (upgrade).

Software Security Challenge

The foundation of our software stacks is built on top of the unsafe C/C++ programming languages.

- Strong performance.
- Direct access to resources.
- Rich legacy.
- Lack of proactive techniques (e.g. Rust and Go).

Additionally, software are getting complex.

- Demands of more features.
- Compatibility and support of different devices.
- Time gap between vulnerability expose and patch implementation.

Unfortunately, defense mechanism does not consider as feature.

- Expect ignorable performance overhead.
- Generic software solution.

C/C++ Application

Buffer Overflow

Buffer Overflow (BoF)

A buffer overflow occurs when data are written outside of the space allocated to the buffer.

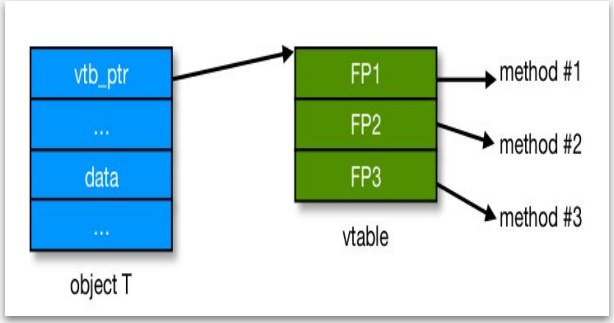
- Unsafe language like C doesn't check writes are in-bound.
- Many unsafe libc functions: strcpy, strcat, gets, scanf.
  - Safe libc versions (e.g., strncpy, strncat) are misleading e.g., strncpy may leave string unterminated.

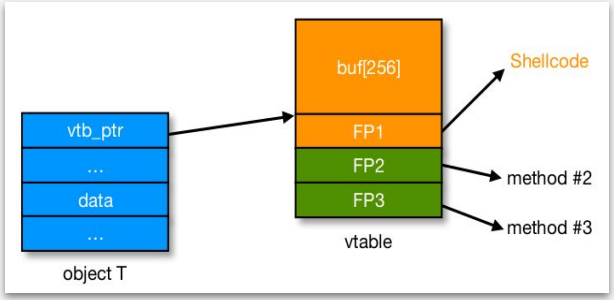
Fuzz testing with address sanitizer is a popular method to find buffer overflows.

Arbitrary Code Execution

Attacker's goal: take over target machine (e.g., web server).
- Execute arbitrary code on target by hijacking control flow.

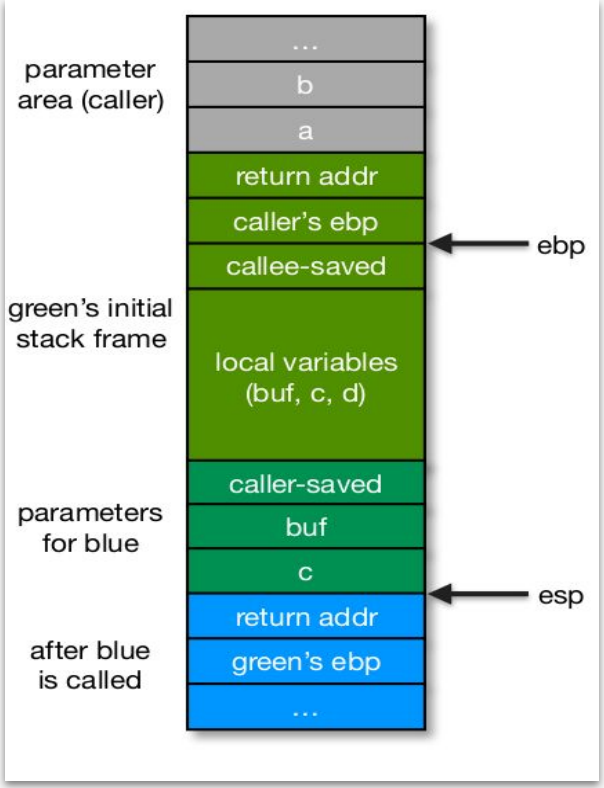
Targets of Control Flow Hijacking:
- Exception handlers.
  - Overwrite address of an exception handler in stack frame.
- Function pointers/return addresses.
  - Overwrite a function pointer on the heap or the stack
- Longjmp buffers: setjmp/longjmp.
  - Longjmp buffer contains EIP.
- Corrupting vtable.
  - Overwrite vtable pointer to malicious buffer.





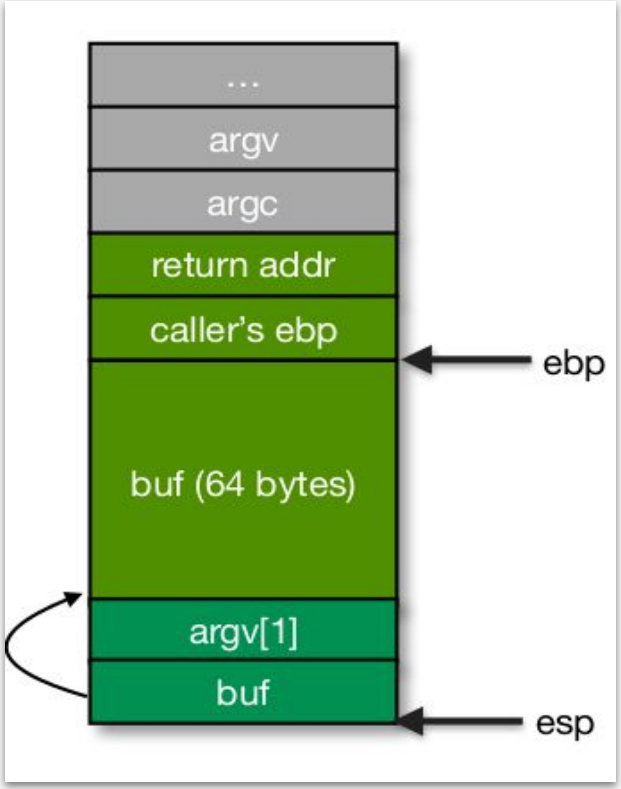
C Program Stack Layout (x86)

```
int green(int a, int b)
{
    char buf[16];
    int c, d;

    if(a > b)
        c = a;
    else
        c = b;

    d = blue(c, buf);
    return d;
}
```



Basic Example

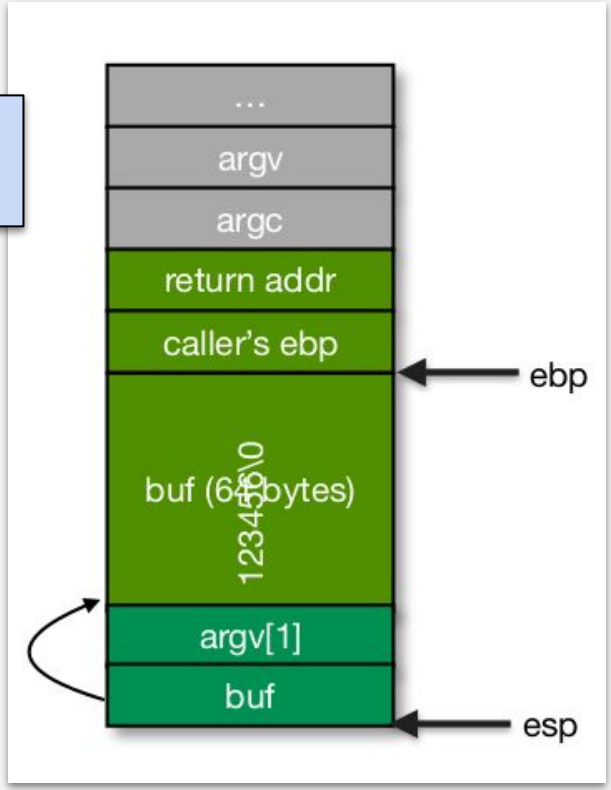
```
#include <string.h>
int main(int argc, char **argv)
{
    char buf[64];
    strcpy(buf, argv[1]);
    return 0;
}

Dump of assembler code for main:
0x080483e4 <+0>:  push   %ebp
0x080483e5 <+1>:  mov    %esp,%ebp
0x080483e7 <+3>:  sub    $72,%esp
0x080483ea <+6>:  mov    12(%ebp),%eax
0x080483ed <+9>:  mov    4(%eax),%eax
0x080483f0 <+12>: mov    %eax,4(%esp)
0x080483f4 <+16>: lea    -64(%ebp),%eax
0x080483f7 <+19>: mov    %eax,(%esp)
0x080483fa <+22>: call   0x8048300 <strcpy@plt>
0x080483ff <+27>: leave
0x08048400 <+28>: ret
```



Expected Input

```
#include <string.h>
int main(int argc, char **argv)
{
    char buf[64];
    strcpy(buf, argv[1]);
    return 0;
}

Dump of assembler code for main:
0x080483e4 <+0>:  push   %ebp
0x080483e5 <+1>:  mov    %esp,%ebp
0x080483e7 <+3>:  sub    $72,%esp
0x080483ea <+6>:  mov    12(%ebp),%eax
0x080483ed <+9>:  mov    4(%eax),%eax
0x080483f0 <+12>: mov    %eax,4(%esp)
0x080483f4 <+16>: lea    -64(%ebp),%eax
0x080483f7 <+19>: mov    %eax,(%esp)
0x080483fa <+22>: call   0x8048300 <strcpy@plt>
0x080483ff <+27>: leave
0x08048400 <+28>: ret
```

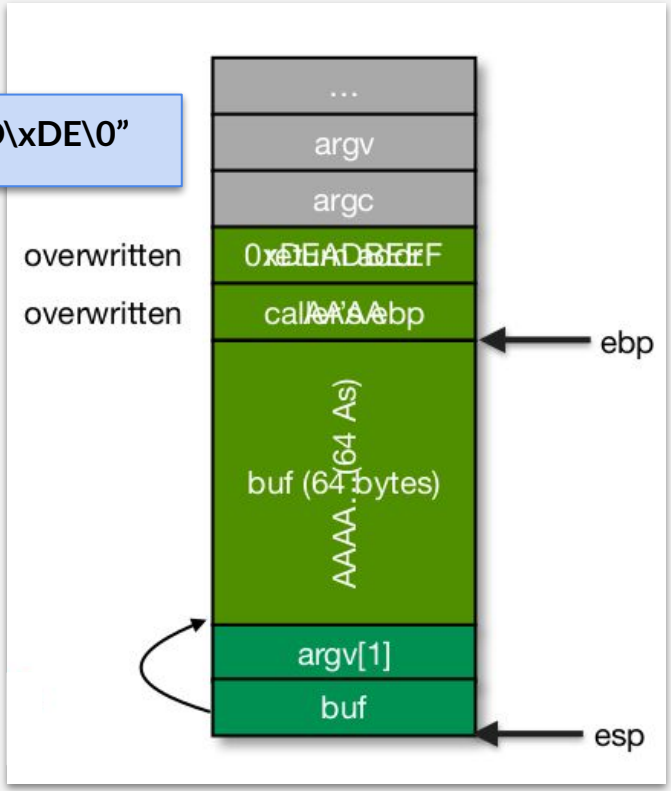
argv[1]: “123456”



Attack Input (Stack Overflow)

```
#include <string.h>
int main(int argc, char **argv)
{
    char buf[64];
    strcpy(buf, argv[1]);
    return 0;
}

Dump of assembler code for main:
0x080483e4 <+0>:  push   %ebp
0x080483e5 <+1>:  mov    %esp,%ebp
0x080483e7 <+3>:  sub    $72,%esp
0x080483ea <+6>:  mov    12(%ebp),%eax
0x080483ed <+9>:  mov    4(%eax),%eax
0x080483f0 <+12>: mov    %eax,4(%esp)
0x080483f4 <+16>: lea    -64(%ebp),%eax
0x080483f7 <+19>: mov    %eax,(%esp)
0x080483fa <+22>: call   0x8048300 <strcpy@plt>
0x080483ff <+27>: leave
0x08048400 <+28>: ret
```

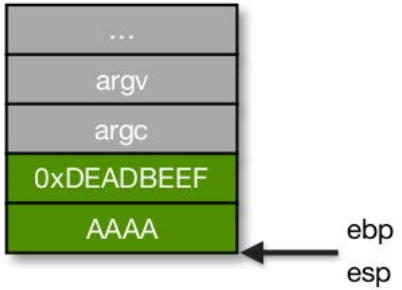
argv[1]: 68 “A”s+“\xEF\xBE\xAD\xDE\0”



Continue ...

```
#include <string.h>
int main(int argc, char **argv)
{
    char buf[64];
    strcpy(buf, argv[1]);
    return 0;
}
```

```
Dump of assembler code for main:
0x080483e4 <+0>:  push   %ebp
0x080483e5 <+1>:  mov    %esp,%ebp
0x080483e7 <+3>:  sub    $72,%esp
0x080483ea <+6>:  mov    12(%ebp),%eax
0x080483ed <+9>:  mov    4(%eax),%eax
0x080483f0 <+12>: mov    %eax,4(%esp)
0x080483f4 <+16>: lea    -64(%ebp),%eax
0x080483f7 <+19>: mov    %eax,(%esp)
0x080483fa <+22>: call   0x8048300 <strcpy@plt>
0x080483ff <+27>: leave
0x08048400 <+28>: ret
```

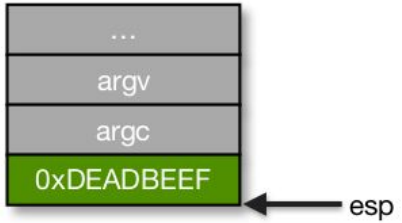


leave:
1. mov %ebp, %esp
2. pop %ebp

Continue ...

```
#include <string.h>
int main(int argc, char **argv)
{
    char buf[64];
    strcpy(buf, argv[1]);
    return 0;
}
```

```
Dump of assembler code for main:
0x080483e4 <+0>:  push   %ebp
0x080483e5 <+1>:  mov    %esp,%ebp
0x080483e7 <+3>:  sub    $72,%esp
0x080483ea <+6>:  mov    12(%ebp),%eax
0x080483ed <+9>:  mov    4(%eax),%eax
0x080483f0 <+12>: mov    %eax,4(%esp)
0x080483f4 <+16>: lea    -64(%ebp),%eax
0x080483f7 <+19>: mov    %eax,(%esp)
0x080483fa <+22>: call   0x8048300 <strcpy@plt>
0x080483ff <+27>: leave
0x08048400 <+28>: ret
```



%ebp = AAAA

leave:
1. mov %ebp, %esp
2. pop %ebp

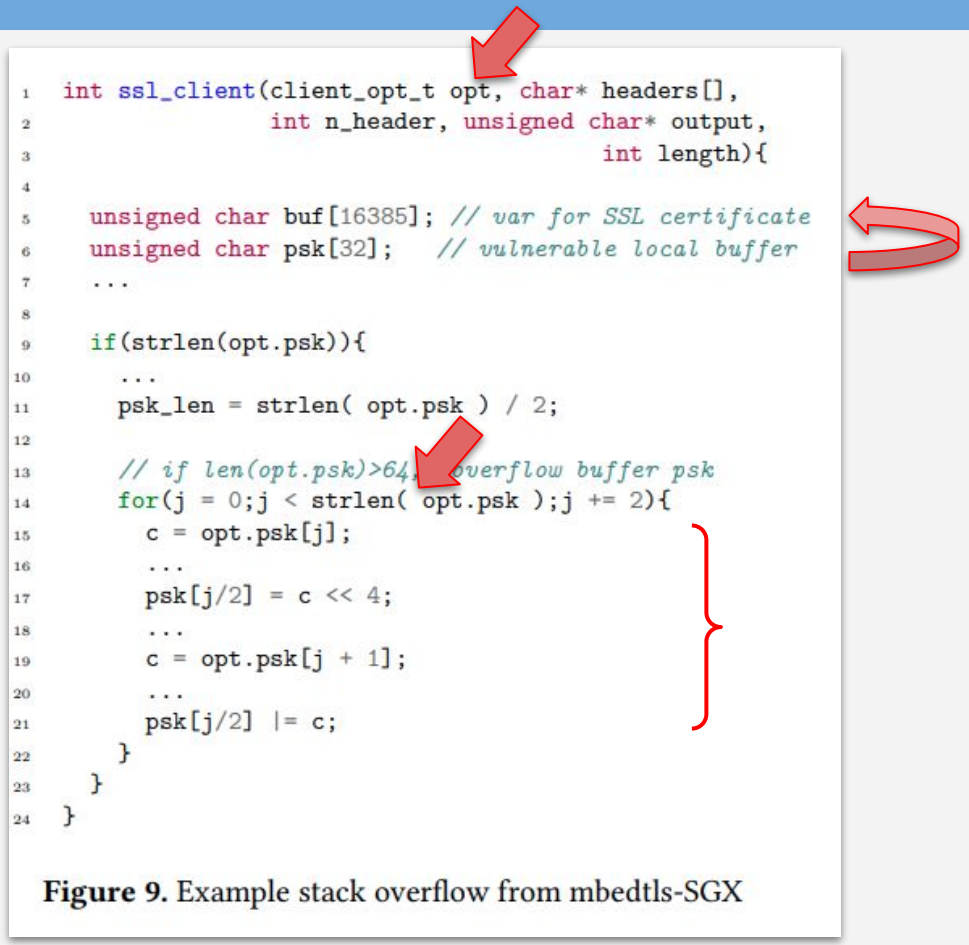
Continue ...

```c
#include <string.h>
int main(int argc, char **argv)
{
    char buf[64];
    strcpy(buf, argv[1]);
    return 0;
}
```


```
Dump of assembler code for main:
0x080483e4 <+0>:  push   %ebp
0x080483e5 <+1>:  mov    %esp,%ebp
0x080483e7 <+3>:  sub    $72,%esp
0x080483ea <+6>:  mov    12(%ebp),%eax
0x080483ed <+9>:  mov    4(%eax),%eax
0x080483f0 <+12>: mov    %eax,4(%esp)
0x080483f4 <+16>: lea    -64(%ebp),%eax
0x080483f7 <+19>: mov    %eax,(%esp)
0x080483fa <+22>: call   0x8048300 <strcpy@plt>
0x080483ff <+27>: leave
0x08048400 <+28>: ret
```

%eip = 0xDEADBEEF
(crash the process)

Real Example

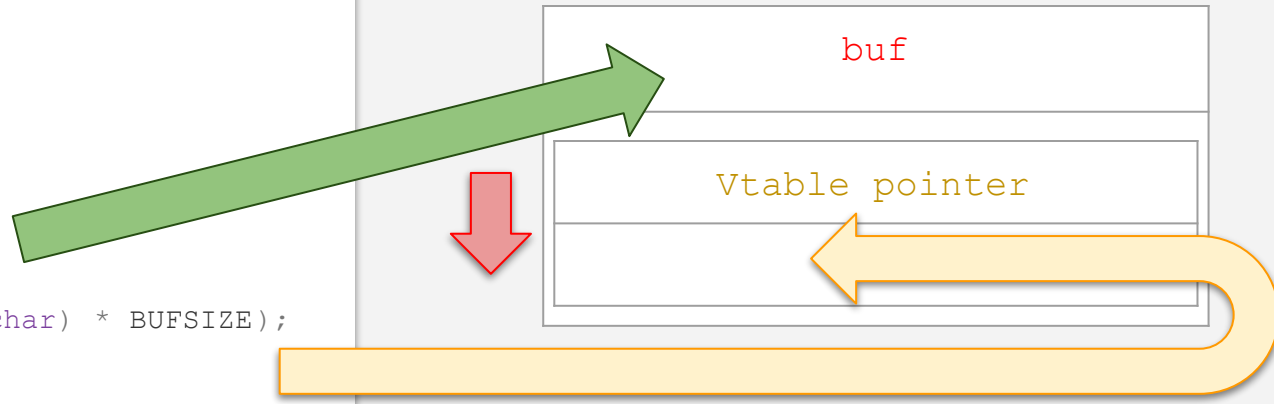


```
1  int ssl_client(client_opt_t opt, char* headers[],
2                 int n_header, unsigned char* output,
3                                       int length){
4
5    unsigned char buf[16385]; // var for SSL certificate
6    unsigned char psk[32];    // vulnerable local buffer
7    ...
8
9    if(strlen(opt.psk)){
10     ...
11     psk_len = strlen( opt.psk ) / 2;
12
13     // if len(opt.psk)>64, overflow buffer psk
14     for(j = 0;j < strlen( opt.psk );j += 2){
15       c = opt.psk[j];
16       ...
17       psk[j/2] = c << 4;
18       ...
19       c = opt.psk[j + 1];
20       ...
21       psk[j/2] |= c;
22     }
23   }
24 }
```

Figure 9. Example stack overflow from mbedtls-SGX

Heap Overflow

```
#define BUFSIZE 256

class C
{
public:
   C() {}
};

int main(int argc, char **argv)
{
   char *buf;
   buf = (char *)malloc(sizeof(char) * BUFSIZE);
   C *c = new C();
   strcpy(buf, argv[1]);
}
```



Type Confusion

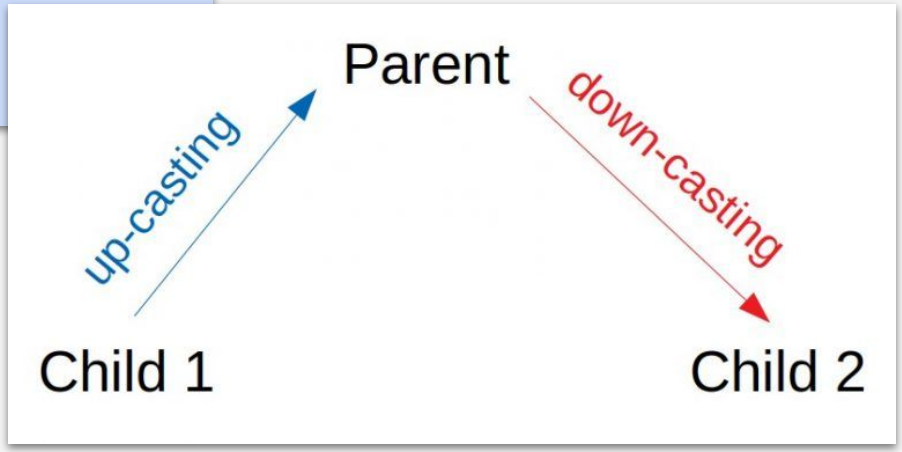
When the program accesses the resource using an incompatible type, this could trigger logical errors because the resource does not have expected properties.

- Memory unsafe language, such as C and C++, type confusion can lead to out-of-bounds memory access.

C++ Casting Operations:

- Static Casting –> static_cast<ToClass>(Object).
- Dynamic Casting –> dynamic_cast<ToClass>(Object).
- C-style Casting –> (ToClass) (Object).

Type Confusion bugs arises from illegal down-casts.



Example


```
#include <iostream>
using namespace std;

class Base{
}; // Parent Class

class Execute : public Base
{ // Child of Base Class
public:
   virtual void exec(const char *program){
       system(program);
   }
};

class Greeter : public Base
{ // Child of Base Class
public:
   virtual void sayHi(const char *str){
       cout << str << endl;
   }
};
```

```
int main(){
   Base *b1 = new Greeter();
   Base *b2 = new Execute();
   Greeter *g;

   g = static_cast<Greeter *>(b1); // Safe Casting to the same type
"Greeter"
   g->sayHi("Greeter says hi!");    // String passed to sayHi()
function

   g = static_cast<Greeter *>(b2); // Unsafe Casting to sibling class
"Execute"
   g->sayHi("/usr/bin/xcalc");      // String passed to exec()
function
                                    // which will turn into a command
to execute calculator

   delete b1;
   delete b2;
   return 0;
}
```

Integer Overflow

C Data Types (x86-32)

short int	16bits	[-32,768; 32,767]
unsigned short int	16bits	[0; 65,535]
unsigned int	32bits	[0; 4,294,967,295]
int	32bits	[-2,147,483,648; 2,147,483,647]
long int	32 bits	[-2,147,483,648; 2,147,483,647]
signed char	8bits	[-128; 127]
unsigned char	8 bits	[0; 255]

Integer Overflow

Problem: what happens when int exceeds max value?

- char c; (8 bits) short s; (16 bits) int m; (32 bits);
- Integer overflow is often related to sign conversion.

```
c = 0x80 + 0x80 = 128 + 128      ⇒  c = 0
s = 0xff80 + 0x80                ⇒  s = 0
m = 0xffffff80 + 0x80            ⇒  m = 0
```

Can integer overflows be exploited?

CVE Reports

Name	Description
CVE-2020-9852	An integer overflow was addressed through improved input validation. This issue is fixed in iOS 13.5 and iPadOS 13.5, macOS Catalina 10.15.5, tvOS 13.4.5, watchOS 6.2.5. A malicious application may be able to execute arbitrary code with kernel privileges.
CVE-2020-9841	An integer overflow was addressed through improved input validation. This issue is fixed in macOS Catalina 10.15.5. An application may be able to execute arbitrary code with kernel privileges.
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 vulnerability. The 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 privileges and execute code in the context of the hypervisor. Was ZDI-CAN-10032.
CVE-2020-8844	This vulnerability allows remote attackers to execute arbitrary code on affected installations of Foxit Reader 9.6.0.25114. User interaction is required to exploit this vulnerability in that the target must visit a malicious page or open a malicious file. The specific flaw exists within the parsing of JPEG files within CovertToPDF. The issue results from the lack of proper validation of user-supplied data, which can result in an integer overflow before writing to memory. An attacker can leverage this vulnerability to execute code in the context of the current process. Was ZDI-CAN-9102.
CVE-2020-6381	Integer overflow in JavaScript in Google Chrome on ChromeOS and Android prior to 80.0.3987.87 allowed a remote attacker to potentially exploit heap corruption via a crafted HTML page.
CVE-2020-6092	An exploitable code execution vulnerability exists in the way Nitro Pro 13.9.1.155 parses Pattern objects. A specially crafted PDF file can trigger an integer overflow that can lead to arbitrary code execution. In order to trigger this vulnerability, victim must open a malicious file.
CVE-2020-6073	An exploitable denial-of-service vulnerability exists in the TXT record-parsing functionality of Videolabs libmicrodns 0.1.0. When parsing the RDATA section in a TXT record in mDNS messages, multiple integer overflows can be triggered, leading to a denial of service. An attacker can send an mDNS message to trigger this vulnerability.
CVE-2020-5310	libImaging/TiffDecode.c in Pillow before 6.2.2 has a TIFF decoding integer overflow, related to realloc.
CVE-2020-4030	In FreeRDP before version 2.1.2, there is an out of bounds read in TrioParse. Logging might bypass string length checks due to an integer overflow. This is fixed in version 2.1.2.
CVE-2020-3641	Integer overflow may occur if atom size is less than atom offset as there is improper validation of atom size in Snapdragon Auto, Snapdragon Compute, Snapdragon Consumer IOT, Snapdragon Industrial IOT, Snapdragon Mobile, Snapdragon Voice & Music, Snapdragon Wearables in APQ8009, APQ8053, APQ8096AU, APQ8098, Kamorta, MDM9206, MDM9207C, MDM9607, MSM8905, MSM8909W, MSM8917, MSM8953, MSM8996AU, MSM8998, QCA6574AU, QCM2150, QCS405, QCS605, QM215, Rennell, SA6155P, Saipan, SDA660, SDM429, SDM429W, SDM439, SDM450, SDM630, SDM632, SDM636, SDM660, SDM845, SDX20, SM6150, SM7150, SM8150, SM8250, SXR2130
CVE-2020-17360	** UNSUPPORTED WHEN ASSIGNED ** An issue was discovered in ReadyTalk Avian 1.2.0. The vm::arrayCopy method defined in classpath-common.h contains multiple boundary checks that are performed to prevent out-of-bounds memory read/write. However, two of these boundary checks contain an integer overflow that leads to a bypass of these checks, and out-of-bounds read/write. NOTE: This vulnerability only affects products that are no longer supported by the maintainer.
CVE-2020-15707	Integer overflows were discovered in the functions grub_cmd_initrd and grub_initrd_init in the efilinux component of GRUB2, as shipped in Debian, Red Hat, and Ubuntu (the functionality is not included in GRUB2 upstream), leading to a heap-based buffer overflow. These could be triggered by an extremely large number of arguments to the initrd command on 32-bit architectures, or a crafted filesystem with very large files on any architecture. An attacker could use this to execute arbitrary code and bypass UEFI Secure Boot restrictions. This issue affects GRUB2 version 2.04 and prior versions.
CVE-2020-15588	An issue was discovered in the client side of Zoho ManageEngine Desktop Central before 10.0.533. An attacker-controlled server can trigger an integer overflow via a crafted header value.
CVE-2020-15137	All versions of HoRNDIS are affected by an integer overflow in the RNDIS packet parsing routines. A malicious USB device can trigger disclosure of unrelated kernel memory to userspace applications on the host, or can cause the kernel to crash. Kernel memory disclosure is especially likely on 32-bit kernels; 64-bit kernels are more likely to crash on attempted exploitation. It is not believed that kernel memory corruption is possible, or that unattended kernel memory disclosure without the collaboration of a userspace program running on the host is possible. The vulnerability is in `HoRNDIS::receivePacket`. `msg_len`, `data_ofs`, and `data_len` can be controlled by an attached USB device, and a negative value of `data_ofs` can bypass the check for `(data_ofs + data_len + 8) > msg_len`, and subsequently can cause a wild pointer copy in the `mbuf_copyback` call. The software is not maintained and no patches are planned. Users of multi-tenant systems with HoRNDIS installed should only connect trusted USB devices to their system.
CVE-2020-15103	In FreeRDP less than or equal to 2.1.2, an integer overflow exists due to missing input sanitation in rdpegfx channel. All FreeRDP clients are affected. The input rectangles from the server are not checked against local surface coordinates and blindly accepted. A malicious server can send data that will crash the client later on (invalid length arguments to a `memcpy`) This has been fixed in 2.2.0. As a workaround, stop using command line arguments /gfx, /gfx-h264 and /network:auto
CVE-2020-14966	An issue was discovered in the jsrsasign package through 8.0.18 for Node.js. It allows a malleability in ECDSA signatures by not checking overflows in the length of a sequence and '0' characters appended or prepended to an integer. The modified signatures are verified as valid. This could have a security-relevant impact if an application relied on a single canonical signature.
CVE-2020-14401	An issue was discovered in LibVNCServer before 0.9.13. libvncserver/scale.c has a pixel_value integer overflow.

Truncation

Truncation errors occur when
- an integer is converted to a smaller integer type and
- the value of the original integer is outside the range of the smaller type

Low-order bits of the original value are preserved and the high-order bits are lost.

```
struct s {
      unsigned short len;
      char buf[];
};

void foo(struct s *p) {
      char buffer[100];
      if (p->len < sizeof buffer)
            strcpy(buffer, p->buf);
      // Use buffer
}
```

```
int main(int argc, char *argv[]) {
      size_t len = strlen(argv[0]);
      struct s *p = malloc(len + 3);
      p->len = len;
      strcpy(p->buf, argv[0]);

      foo(p);
      return 0;
}
```

Arithmetic Overflow

An arithmetic overflow error when the result of an integer operation does not fit within the allocated memory space.

```
void  func( char *buf1, *buf2, unsigned int len1, len2)
{
    char temp[256];

    if  (len1 + len2 > 256)  {return -1}   // length check

    memcpy(temp, buf1, len1);              // cat buffers
    memcpy(temp+len1, buf2, len2);

    do-something(temp);
}
```

What if len1 = 0x80, len2 = 0xffffff80?

- len1 + len2 = 0
- Second memcpy overflows heap!

Signedness Bugs

Compare two signed integers, assuming nonnegativity.

```
if (x < 100)
  do_something();
```

Compare a signed and unsigned integer.

```
if (size < sizeof buffer)
  do_something();
```

Treating a signed negative number as unsigned.

```
void *p = malloc(size); // size < 0
```

Best Practice

Use appropriate types:

- Need a size of a count? Use *size_t*
- Need a specific bit-width? Use *uint8_t*, *uint16_t*, *uint_32_t*, *uint64_t*, etc.
- Need an integer to hold a pointer? Use *intptr_t*.

```c
#include <limits.h>
#include <stdlib.h>

int safe_add(int a, int b)
{
    if (a > 0 && b > INT_MAX - a)
        abort();
    if (a < 0 && b < INT_MIN - a)
        abort();
    return a + b;
}
```

Format String Vulnerability

Format String Vulnerability

> “If an attacker is able to provide the format string to an ANSI C format function in part or as a whole, a format string vulnerability is present.”
>
> –scut/team teso

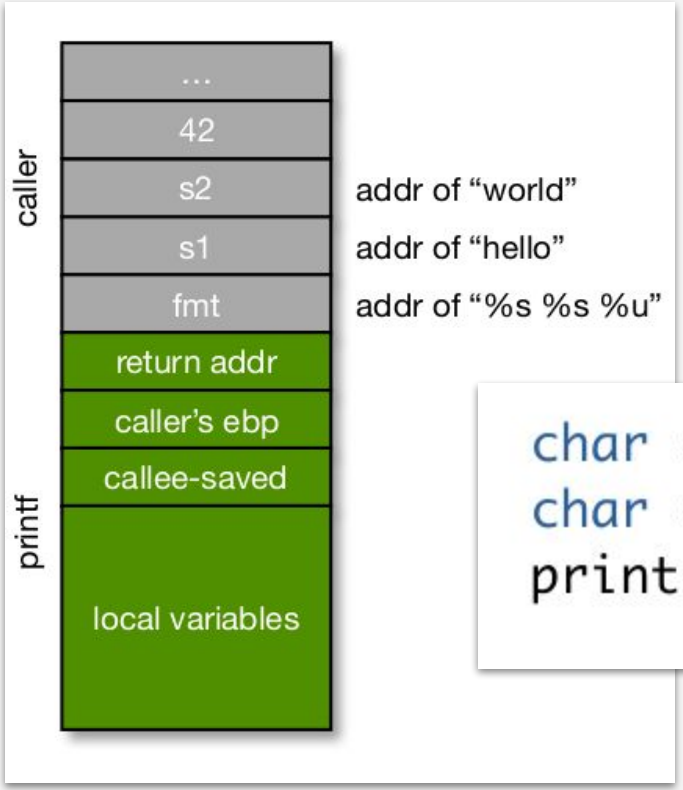
Format String Functions

printf(char *fmt, ...)

- fmt: specifies number and types of arguments.
- ...: variable number of arguments.

Function	Purpose
printf	Output a formatted string
sprintf/snprintf/vsprintf/vsnprintf	Prints to a string/checking the length also
fprintf/vfprintf	Prints into a file
syslog	Writes a message to the system log

Stack Layout for Printf




```
char s1[] = "hello";
char s2[] = "world";
printf("%s %s %u", s1, s2, 42);
```

Example Vulnerable Program

```
int vulnerable (char *user)
{
    printf(user);
}
```

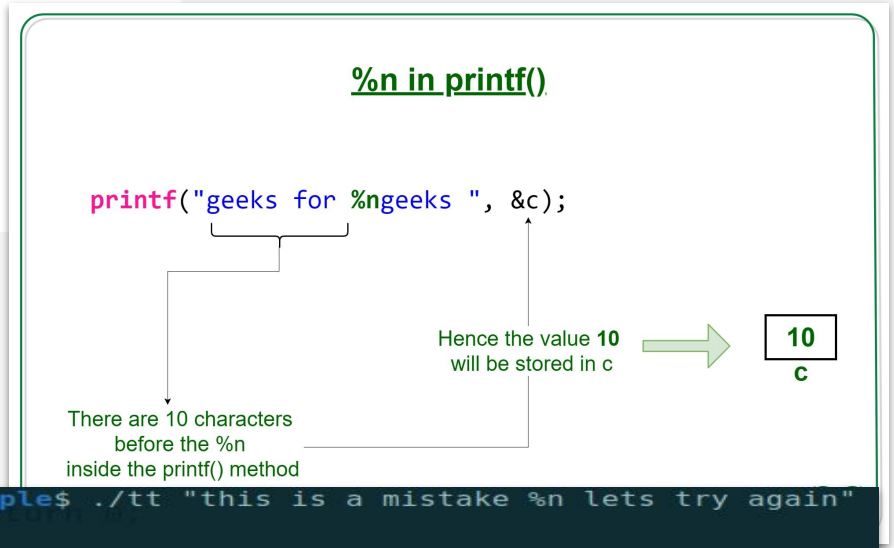
```
int correct (char *user)
{
    printf("%s", user);
}
```

Problem: what if *user = "%s%s%s%s%s%s%s" ?

- Most likely program will crash: DoS
- If not, program will print memory contents: info leak
- Full exploit using user = "%n"

Use of %n in printf()

```c
#include <stdio.h>

int main(int argc, char** argv){
   int c;
   printf("Hello world %n before specifier", &c);
   printf("\n%d\n", c);
   printf(argv[1], &c);
   printf("\n%d\n", c);
   return 0;
}
```




```
mustakim@panda-home:~/teaching/sample$ ./tt "this is a mistake %n lets try again"
Hello world  before specifier
12
this is a mistake  lets try again
18
```

Use-After-Free

Dangling Pointer

A dangling pointer is a pointer variable through which the freed memory is accessed.

```c
char* p = malloc(100); // memory allocated, p is valid
free(p); // p is freed
puts(p); // when puts is called p is a dangling pointer
```

Also known as stale pointer and wild pointer.

Use-After-Free

A class of vulnerability where data on the heap is freed, but a leftover reference or '**dangling pointer**' is used by the code as if the data were still valid.

Causes:

- Wrongly handled error conditions.
- Unaccounted for program states.
- Confusion over which part of the program is responsible for freeing the memory.

An use-after-free bug is exploitable if the program can be brought in a state in which it can allocate memory over the freed area.

Example


```
char *ptr = (char *)malloc(SIZE);
... if (err)
{
    abrt = 1;
    free(ptr);
}
... if (abrt)
{
    logError("operation aborted before commit", ptr);
}
```

Usage & Detection

Why are they so well liked?

- Doesn't require any memory corruption to use.
- Can be used for info leaks.
- Can be used to trigger memory corruption or get control of EIP.

From the defensive perspective, trying to detect use after free vulnerabilities in complex applications is very difficult, even in industry.

Why?

- UAF's only exist in certain states of execution, so statically scanning source for them won't go far (+ multithreaded environment).
- Symbolic execution and constraint solvers are helping find these bugs faster.

Prevention

Set all freed pointers to NULL.

- Check null-pointer before using it.
- Nowadays, OS has built-in defense for null-pointer dereference.

Complexity:

- Need to set all aliased pointers to NULL.
- Free(memory) + Set pointer to NULL needs to be atomic operation.

```
char *message;
int message_type;
/* Initialize message and message_type */
if (message_type == value_1) {
 /* Process message type 1 */
 free(message);
 message = NULL;
}
/* ... */
if (message_type == value_2) {
 /* Process message type 2 */
 free(message);
 message = NULL;
}
```

Double Free

Double Free

Double Free: Freeing the same chunk of memory twice, without it being reallocated in between.

An exploitation occurs when the program calls free on a region that contains data set by the attacker:

- free(q) will try to use the chunk tag located just before the address pointed to by q.
- In this case, the “chunk tag” is now actually part of the attacker’s string.
- The attacker can set the values in their “chunk tag” such that free will overwrite a memory location chosen by the attacker with a value chosen by the attacker.

Real Example

```
1  int
2  ecall_change_master_password(const char* old_password,
3                               const char* new_password) {
4    if (strlen(new_password) < 8
5              || strlen(new_password)+1 > MAX_ITEM_SIZE)
6      ...
7    call_status = ocall_load_wallet(&ocall_ret,
8                              sealed_data, sealed_size);
9    ...
10   sealing_status = unseal_wallet(
11                (sgx_sealed_data_t*)sealed_data,
12                               wallet, plaintext_size);
13   ...
14   if (strcmp(wallet->master_password,
15                        old_password) != 0) {
16         ...
17   }
18   strncpy(wallet->master_password, new_password,
19                               strlen(new_password)+1);
20   ...
21   sealing_status = seal_wallet(wallet,
22                 (sgx_sealed_data_t*)sealed_data,
23                              sealed_size);
24   free(wallet); // first free
25   if (sealing_status != SGX_SUCCESS) {
26     free(wallet);   // second free
27     ...
28   }
29   ...
30 }
```

CVE Reports

Name	Description
CVE-2020-9844	A double free issue was addressed with improved memory management. This issue is fixed in iOS 13.5 and iPadOS 13.5, macOS Catalina 10.15.5. A remote attacker may be able to cause unexpected system termination or corrupt kernel memory.
CVE-2020-9795	A use after free issue was addressed with improved memory management. This issue is fixed in iOS 13.5 and iPadOS 13.5, macOS Catalina 10.15.5, tvOS 13.4.5, watchOS 6.2.5. An application may be able to execute arbitrary code with kernel privileges.
CVE-2020-9783	A use after free issue was addressed with improved memory management. This issue is fixed in iOS 13.4 and iPadOS 13.4, tvOS 13.4, Safari 13.1, iTunes for Windows 12.10.5, iCloud for Windows 10.9.3, iCloud for Windows 7.18. Processing maliciously crafted web content may lead to code execution.
CVE-2020-9768	A use after free issue was addressed with improved memory management. This issue is fixed in iOS 13.4 and iPadOS 13.4, tvOS 13.4, watchOS 6.2. An application may be able to execute arbitrary code with system privileges.
CVE-2020-9633	Adobe Flash Player Desktop Runtime 32.0.0.371 and earlier, Adobe Flash Player for Google Chrome 32.0.0.371 and earlier, and Adobe Flash Player for Microsoft Edge and Internet Explorer 32.0.0.330 and earlier have an use after free vulnerability. Successful exploitation could lead to arbitrary code execution.
CVE-2020-9607	Adobe Acrobat and Reader versions 2020.006.20042 and earlier, 2017.011.30166 and earlier, 2017.011.30166 and earlier, and 2015.006.30518 and earlier have an use-after-free vulnerability. Successful exploitation could lead to arbitrary code execution .
CVE-2020-9606	Adobe Acrobat and Reader versions 2020.006.20042 and earlier, 2017.011.30166 and earlier, 2017.011.30166 and earlier, and 2015.006.30518 and earlier have an use-after-free vulnerability. Successful exploitation could lead to arbitrary code execution .
CVE-2020-9567	Adobe Bridge versions 10.0.1 and earlier version have an use after free vulnerability. Successful exploitation could lead to arbitrary code execution .
CVE-2020-9566	Adobe Bridge versions 10.0.1 and earlier version have an use after free vulnerability. Successful exploitation could lead to arbitrary code execution .
CVE-2020-9543	OpenStack Manila <7.4.1, >=8.0.0 <8.1.1, and >=9.0.0 <9.1.1 allows attackers to view, update, delete, or share resources that do not belong to them, because of a context-free lookup of a UUID. Attackers may also create resources, such as shared file systems and groups of shares on such share networks.
CVE-2020-9320	Avira AV Engine before 8.3.54.138 allows virus-detection bypass via a crafted ISO archive. This affects versions before 8.3.54.138 of Antivirus for Endpoint, Antivirus for Small Business, Exchange Security (Gateway), Internet Security Suite for Windows, Prime, Free Security Suite for Windows, and Cross Platform Anti-malware SDK.
CVE-2020-9273	In ProFTPD 1.3.7, it is possible to corrupt the memory pool by interrupting the data transfer channel. This triggers a use-after-free in alloc_pool in pool.c, and possible remote code execution.
CVE-2020-9262	HUAWEI Mate 30 with versions earlier than 10.1.0.150(C00E136R5P3) have a use after free vulnerability. There is a condition exists that the system would reference memory after it has been freed, the attacker should trick the user into running a crafted application with high privilege, successful exploit could cause code execution.
CVE-2020-9065	Huawei smart phone Taurus-AL00B with versions earlier than 10.0.0.203(C00E201R7P2) have a use-after-free (UAF) vulnerability. An authenticated, local attacker may perform specific operations to exploit this vulnerability. Successful exploitation may tamper with the information to affect the availability.
CVE-2020-8961	An issue was discovered in Avira Free-Antivirus before 15.0.2004.1825. The Self-Protection feature does not prohibit a write operation from an external process. Thus, code injection can be used to turn off this feature. After that, one can construct an event that will modify a file at a specific location, and pass this event to the driver, thereby defeating the anti-virus functionality.
CVE-2020-8945	The proglottis Go wrapper before 0.1.1 for the GPGME library has a use-after-free, as demonstrated by use for container image pulls by Docker or CRI-O. This leads to a crash or potential code execution during GPG signature verification.
CVE-2020-8649	There is a use-after-free vulnerability in the Linux kernel through 5.5.2 in the vgacon_invert_region function in drivers/video/console/vgacon.c.
CVE-2020-8648	There is a use-after-free vulnerability in the Linux kernel through 5.5.2 in the n_tty_receive_buf_common function in drivers/tty/n_tty.c.
CVE-2020-8647	There is a use-after-free vulnerability in the Linux kernel through 5.5.2 in the vc_do_resize function in drivers/tty/vt/vt.c.
CVE-2020-8600	Trend Micro Worry-Free Business Security (9.0, 9.5, 10.0) is affected by a directory traversal vulnerability that could allow an attacker to manipulate a key file to bypass authentication.
CVE-2020-8598	Trend Micro Apex One (2019), OfficeScan XG and Worry-Free Business Security (9.0, 9.5, 10.0) server contains a vulnerable service DLL file that could allow a remote attacker to execute arbitrary code on affected installations with SYSTEM level privileges. Authentication is not required to exploit this vulnerability.
CVE-2020-8470	Trend Micro Apex One (2019), OfficeScan XG and Worry-Free Business Security (9.0, 9.5, 10.0) server contains a vulnerable service DLL file that could allow an attacker to delete any file on the server with SYSTEM level privileges. Authentication is not required to exploit this vulnerability.
CVE-2020-8468	Trend Micro Apex One (2019), OfficeScan XG and Worry-Free Business Security (9.0, 9.5, 10.0) agents are affected by a content validation escape vulnerability which could allow an attacker to manipulate certain agent client components. An attempted attack requires user authentication.
CVE-2020-8447	In OSSEC-HIDS 2.7 through 3.5.0, the server component responsible for log analysis (ossec-analysisd) is vulnerable to a use-after-free during processing of syscheck formatted msgs (received from authenticated remote agents and delivered to the analysisd processing queue by ossec-remoted).

< Memory Corruption />