An answer.pdf for 2018 Fall Midsem. Scored 98%.

24 October 2018
Midterm CEG-4420 Host Computer Security

Start time: 3:00 pm
End time: 4:40 pm

Part 1

1. (6 min) **Size**: Size is a utility program which tells the sizes of the segments in a program file. For instance, the /bin/ls in the question has about 126,379 bytes of code (text) and 4,728 bytes of constants and global variable (data).

If /bin/ls changes size, it is possible that it has been replaced with a trojaned version, so the statement is true as long as the change in size was not an update done by a system administrator.

2. (5 min) **sha512sum** is a program which computes a cryptographic SHA-512 has of a file. A hash takes an object of some (often arbitrary) length and produces a small fixed size output based on the input. If a file is modified, it is extremely unlikely to produce the same hash as the original. SHA-512 is a cryptographic hash which means that is is (as far as we know) computationally infeasible to systematically modify a file to produce the same hash as the original file.

An executable file like /bin/ls which should not change on a regular basis producing a different hash indicates that someone modified the file and would indicate a compromise.

3. (4 min) **Rootkit** A rootkit is a set of programs which can be used by an attacker to make it easier to regain access to a compromised system and hide his activities from the system's owners. Rootkits may contain trojans of common system programs such as login (to give a backdoor), ls (to hide files), ps, (to hide processes).

The statement as written is not true. A rootkit is a collection of programs used by an intruder to hide damage from the system administrator.

4. (6 min) **/proc/1** Proc is a pseudo-filesystem created by the kernel which allows access to information on processes by creating a directory structure to represent each process. The /1 indicates the directory is for the process with PID 1, aka the init process.

The statement is basically true. The command used to start the init process will be stored in the file /proc/1/cmdline. Reading this file will give the path telling which program was invoked as init.

5. **suid** (10 min) Suid (set user ID) root means that a program will run as if it were owned by root regardless of who invoked the program. This is different than normal programs where they will run with the same permissions as the user who invoked them. Suid programs are a common target for exploitation because a successful exploit will allow an attacker to utilize the root privileges of the suid processes.

/bin/mount
/bin/umount

Mount and umount are used to add and remove filesystems. They are suid because adding and removing filesystems is a privileged task. A normal user should not have the ability to mount filesystems over say /bin as they could replace important system programs with their own trojaned versions.

/usr/bin/sudo => sudo allows a user to execute commands as if they were another user. This is only possible if sudo is a suid binary as, otherwise, users could only execute commands as themselves. Because sudo is suid, administrators must be careful what privileged commands they allow users to run by editing /etc/sudoers.

/bin/cat => Cat is a program which concatenates files and writes them to standard output. This should definitely not be suid because it would allow any user to view any file on the system including sensitive ones like /etc/shadow. Users should only be able to read files which they have the proper privileges to read.

Part 2

1.

(i) (5 min)

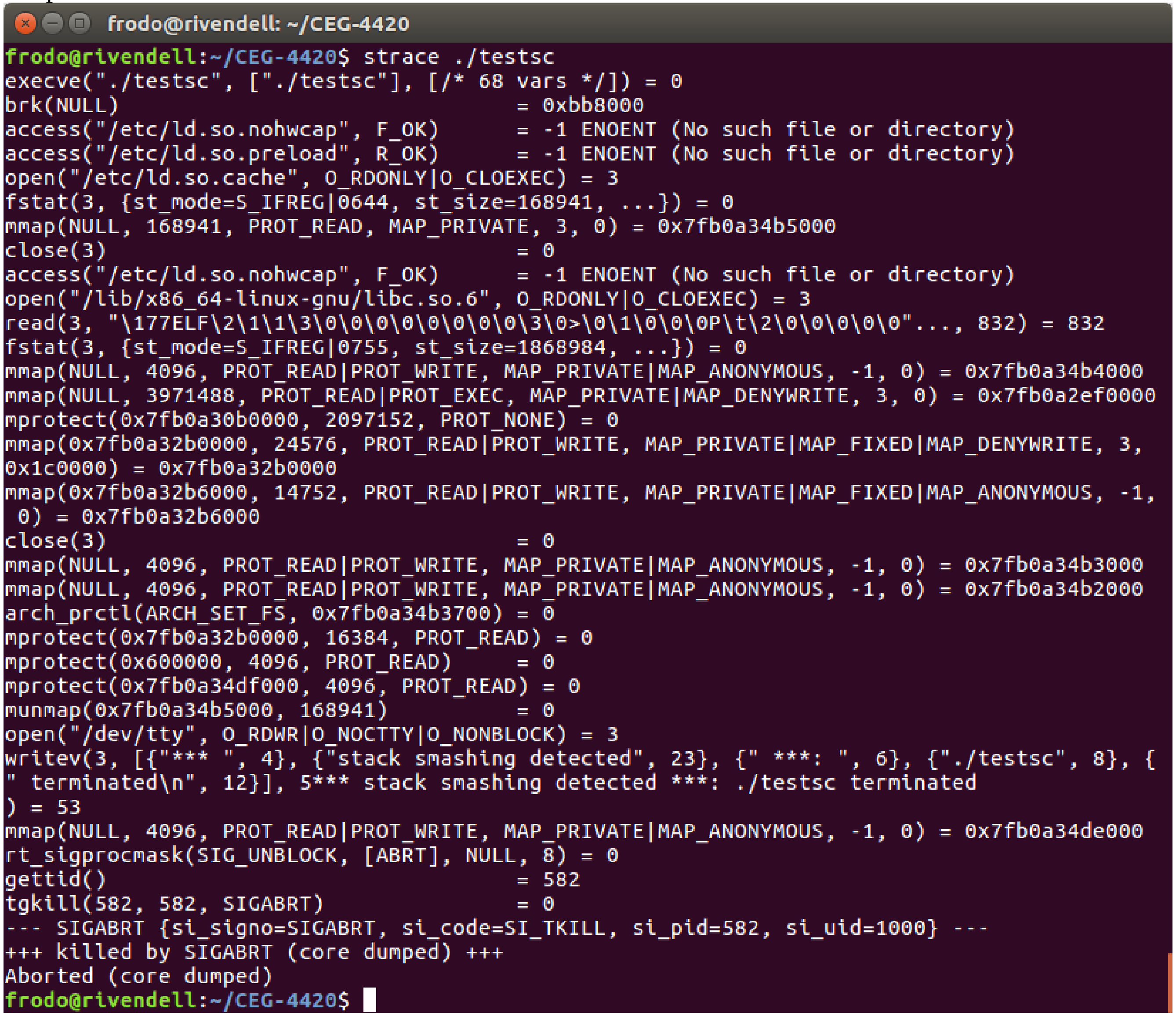
E7: BIOS/UEFI finds the boot device
E8: OS boot loader is discovered
E3: OS boot loader reads the kernel image
E4: OS boot loader invokes the kernel
E1: Root volume (initrd) is mounted by the kernel
E2: Init process is created
E5: Several more processes are started
E6: Several processes (the currently running ones) are started
E9: All file volumes are unmounted Reboot
E10: Init is terminated

(ii) (6 min)
E9: All file volumes are unmounted. When shutting down the computer, the kernel must unmount all file volumes to ensure that all data is flushed to from memory buffers into the filesystems and that all file operations are completed before powering down. This prevents filesystem damage or lost data that could result from incomplete or interrupted file operations.

Security may be breeched if filesystems are modified in an bad way by an attacker. This could include things like overwriting the boot partition to make the system load a bad kernel the next time it is started.

2.
Note: I ran this on 64-bit Ubuntu 16.04

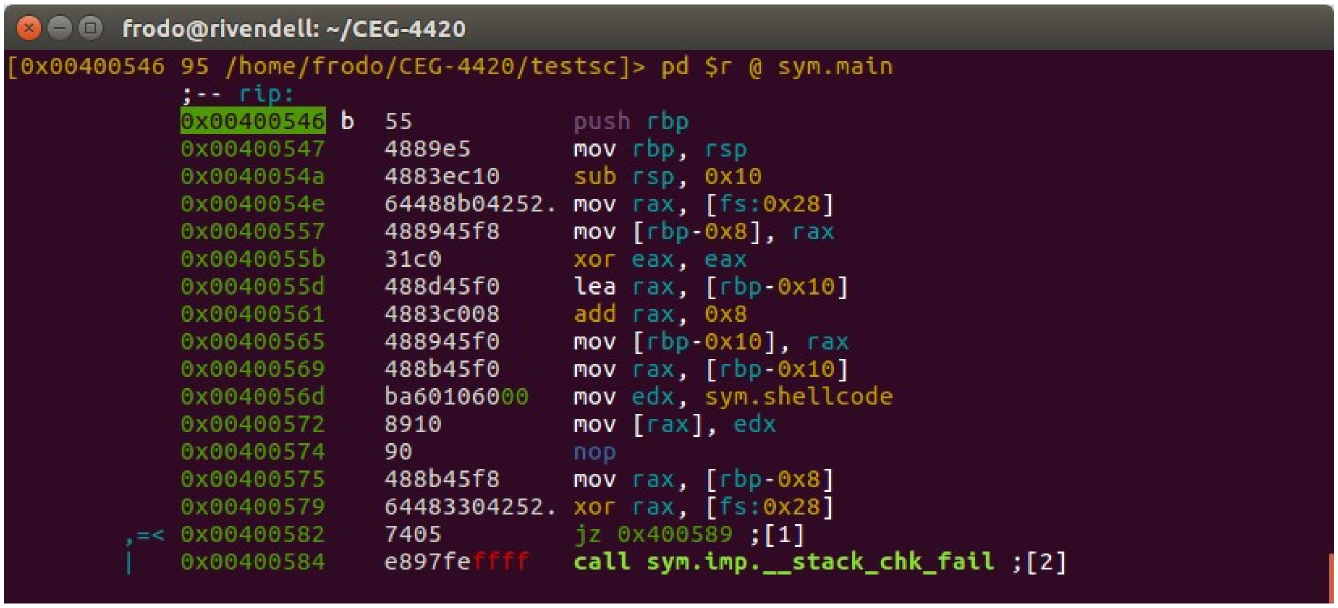
Output of strace


frodo@rivendell: ~/CEG-4420

```
frodo@rivendell:~/CEG-4420$ strace ./testsc
execve("./testsc", ["./testsc"], [/* 68 vars */]) = 0
brk(NULL)                               = 0xbb8000
access("/etc/ld.so.nohwcap", F_OK)      = -1 ENOENT (No such file or directory)
access("/etc/ld.so.preload", R_OK)      = -1 ENOENT (No such file or directory)
open("/etc/ld.so.cache", O_RDONLY|O_CLOEXEC) = 3
fstat(3, {st_mode=S_IFREG|0644, st_size=168941, ...}) = 0
mmap(NULL, 168941, PROT_READ, MAP_PRIVATE, 3, 0) = 0x7fb0a34b5000
close(3)                                = 0
access("/etc/ld.so.nohwcap", F_OK)      = -1 ENOENT (No such file or directory)
open("/lib/x86_64-linux-gnu/libc.so.6", O_RDONLY|O_CLOEXEC) = 3
read(3, "\177ELF\2\1\1\3\0\0\0\0\0\0\0\0\3\0>\0\1\0\0\0P\t\2\0\0\0\0\0"..., 832) = 832
fstat(3, {st_mode=S_IFREG|0755, st_size=1868984, ...}) = 0
mmap(NULL, 4096, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x7fb0a34b4000
mmap(NULL, 3971488, PROT_READ|PROT_EXEC, MAP_PRIVATE|MAP_DENYWRITE, 3, 0) = 0x7fb0a2ef0000
mprotect(0x7fb0a30b0000, 2097152, PROT_NONE) = 0
mmap(0x7fb0a32b0000, 24576, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3,
0x1c0000) = 0x7fb0a32b0000
mmap(0x7fb0a32b6000, 14752, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_ANONYMOUS, -1,
 0) = 0x7fb0a32b6000
close(3)                                = 0
mmap(NULL, 4096, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x7fb0a34b3000
mmap(NULL, 4096, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x7fb0a34b2000
arch_prctl(ARCH_SET_FS, 0x7fb0a34b3700) = 0
mprotect(0x7fb0a32b0000, 16384, PROT_READ) = 0
mprotect(0x600000, 4096, PROT_READ)     = 0
mprotect(0x7fb0a34df000, 4096, PROT_READ) = 0
munmap(0x7fb0a34b5000, 168941)          = 0
open("/dev/tty", O_RDWR|O_NOCTTY|O_NONBLOCK) = 3
writev(3, [{"*** ", 4}, {"stack smashing detected", 23}, {" ***: ", 6}, {"./testsc", 8}, {
" terminated\n", 12}], 5*** stack smashing detected ***: ./testsc terminated
) = 53
mmap(NULL, 4096, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x7fb0a34de000
rt_sigprocmask(SIG_UNBLOCK, [ABRT], NULL, 8) = 0
gettid()                                = 582
tgkill(582, 582, SIGABRT)               = 0
--- SIGABRT {si_signo=SIGABRT, si_code=SI_TKILL, si_pid=582, si_uid=1000} ---
+++ killed by SIGABRT (core dumped) +++
Aborted (core dumped)
frodo@rivendell:~/CEG-4420$ 
```

(i) (10 min)

The shellcode did not execute as the code resulted in a segmentation fault instead of a shell. A message "*** stack smashing detected ***: ./testsc terminated" was printed when I ran the program (see the writev syscall). This message was generated by the stack canary check which is included by default in modern compilers to help detect stack overflows. The stack canary works by putting a number on the stack and then checking if the number changed right before returning from a function. See the below disassembly of testsc's main function. The xor rax,[fs:0x28] line is checking the canary value and the call sym.imp.__stack_chk_fail is what inovkes the function that produced the error message when the canary did not match.



```
[0x00400546 95 /home/frodo/CEG-4420/testsc]> pd $r @ sym.main
            ;-- rip:
            0x00400546 b  55            push rbp
            0x00400547    4889e5        mov rbp, rsp
            0x0040054a    4883ec10      sub rsp, 0x10
            0x0040054e    64488b04252.  mov rax, [fs:0x28]
            0x00400557    488945f8      mov [rbp-0x8], rax
            0x0040055b    31c0          xor eax, eax
            0x0040055d    488d45f0      lea rax, [rbp-0x10]
            0x00400561    4883c008      add rax, 0x8
            0x00400565    488945f0      mov [rbp-0x10], rax
            0x00400569    488b45f0      mov rax, [rbp-0x10]
            0x0040056d    ba60106000    mov edx, sym.shellcode
            0x00400572    8910          mov [rax], edx
            0x00400574    90            nop
            0x00400575    488b45f8      mov rax, [rbp-0x8]
            0x00400579    64483304252.  xor rax, [fs:0x28]
        ,=< 0x00400582    7405          jz 0x400589 ;[1]
        |   0x00400584    e897feffff    call sym.imp.__stack_chk_fail ;[2]
```

(ii) (5 min)
System calls explained
open() is used to give a process access to a file. The kernel will place put a reference to the file in the process's file table and return a file descriptor (index in the table) for the process to use to reference that particular file.

mmap() is used to allocate pages of virtual memory for a process. If a process tries to access memory which has not been allocated to it via mmap, a segmentation fault will occur.

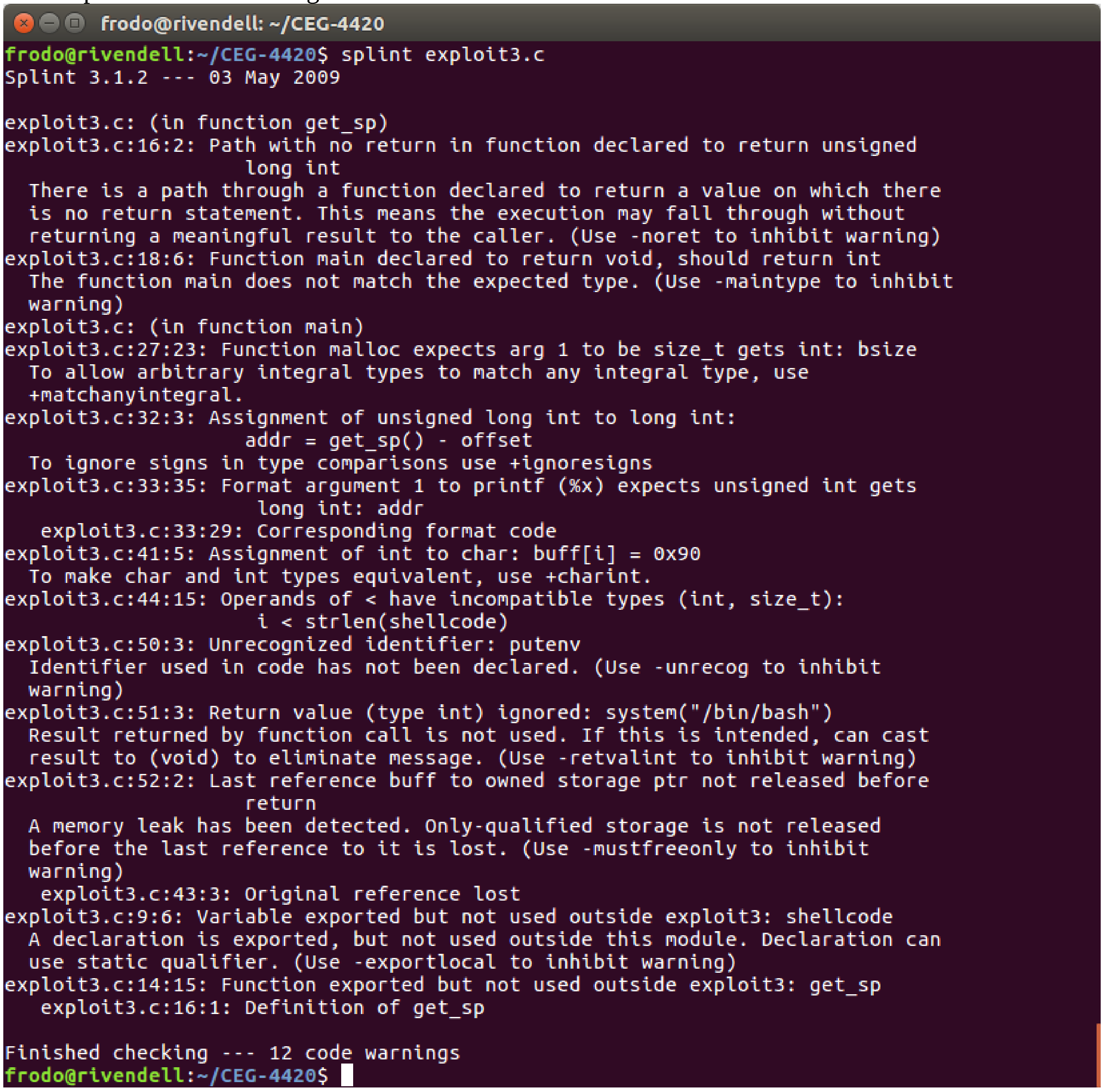
3. (5 min)
The byte 0x00 is also known as the nul character. The nul character is used to indicate the end of a string in C. AlephOne is trying to exploit C functions like strcpy() which copies data from a source buffer to a destination buffer until the it encounters a nul character in the source buffer. The weakness of this function is that it looks copies until a nul character instead of copying a set number of bytes. This allows for buffer overflows when the source buffer is made larger than the destination one. If a string has nuls in the middle of it, only the first part of the string will be copied meaning that some essential parts of the shellcode would be lost. 0x00 can be avoided by changing which assembly language instructions are used. For example instead of saying "mov eax, 0x00" (which contains 0x00), you can use "xor eax, eax" which has the same result but does not require any 0x00 bytes.

4. (20 min)

The Cs stand for a call instruction. The function being called is S, which is the code to set up and invoke execve("/bin/sh"). The arrow 3 represents that the call invokes S. The call is used in order to set up a stack frame for the shellcode to push arguments to. The arrow is a relative call based on the size of the shellcode because the exact location of the shellcode in memory is not known at the time of writing the exploit.

5. (25 min)

Initial splint run – 12 warnings

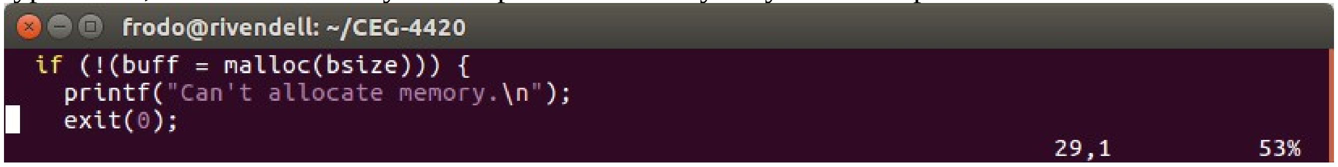


frodo@rivendell: ~/CEG-4420

```
frodo@rivendell:~/CEG-4420$ splint exploit3.c
Splint 3.1.2 --- 03 May 2009

exploit3.c: (in function get_sp)
exploit3.c:16:2: Path with no return in function declared to return unsigned
                    long int
  There is a path through a function declared to return a value on which there
  is no return statement. This means the execution may fall through without
  returning a meaningful result to the caller. (Use -noret to inhibit warning)
exploit3.c:18:6: Function main declared to return void, should return int
  The function main does not match the expected type. (Use -maintype to inhibit
  warning)
exploit3.c: (in function main)
exploit3.c:27:23: Function malloc expects arg 1 to be size_t gets int: bsize
  To allow arbitrary integral types to match any integral type, use
  +matchanyintegral.
exploit3.c:32:3: Assignment of unsigned long int to long int:
                    addr = get_sp() - offset
  To ignore signs in type comparisons use +ignoresigns
exploit3.c:33:35: Format argument 1 to printf (%x) expects unsigned int gets
                     long int: addr
   exploit3.c:33:29: Corresponding format code
exploit3.c:41:5: Assignment of int to char: buff[i] = 0x90
  To make char and int types equivalent, use +charint.
exploit3.c:44:15: Operands of < have incompatible types (int, size_t):
                     i < strlen(shellcode)
exploit3.c:50:3: Unrecognized identifier: putenv
  Identifier used in code has not been declared. (Use -unrecog to inhibit
  warning)
exploit3.c:51:3: Return value (type int) ignored: system("/bin/bash")
  Result returned by function call is not used. If this is intended, can cast
  result to (void) to eliminate message. (Use -retvalint to inhibit warning)
exploit3.c:52:2: Last reference buff to owned storage ptr not released before
                    return
  A memory leak has been detected. Only-qualified storage is not released
  before the last reference to it is lost. (Use -mustfreeonly to inhibit
  warning)
   exploit3.c:43:3: Original reference lost
exploit3.c:9:6: Variable exported but not used outside exploit3: shellcode
  A declaration is exported, but not used outside this module. Declaration can
  use static qualifier. (Use -exportlocal to inhibit warning)
exploit3.c:14:15: Function exported but not used outside exploit3: get_sp
   exploit3.c:16:1: Definition of get_sp

Finished checking --- 12 code warnings
frodo@rivendell:~/CEG-4420$
```

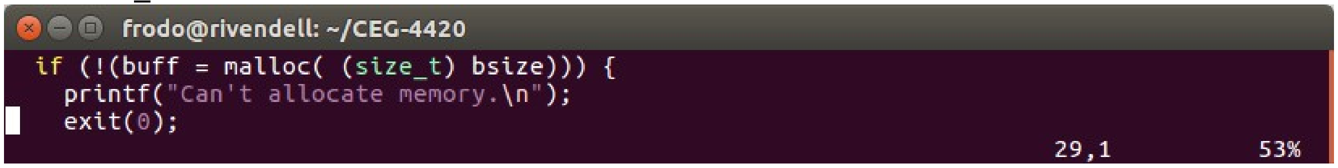
Warning 1: “exploit3.c:27:23 Function malloc expects arg 1 to be size_t gets int: bsize”

The line in question is shown below. The warning exists because int and size_t may not be equivalent types. In C, the size of int is system dependent and may vary from one processor to another.



```
frodo@rivendell: ~/CEG-4420
  if (!(buff = malloc(bsize))) {
    printf("Can't allocate memory.\n");
    exit(0);
                                                        29,1          53%
```

To fix this, I added a typecast to bsize which explicitly tells the compiler that I want to coerce bsize into a size_t.

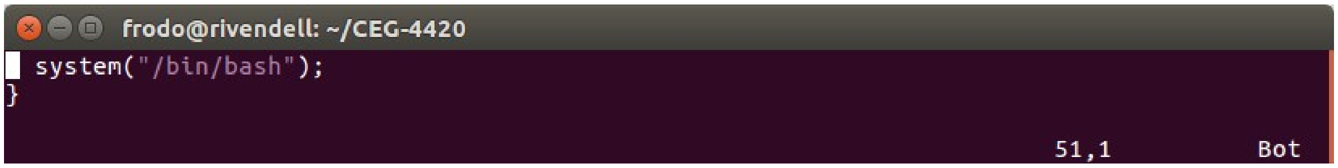


```
frodo@rivendell: ~/CEG-4420
  if (!(buff = malloc( (size_t) bsize))) {
    printf("Can't allocate memory.\n");
    exit(0);
                                                        29,1          53%
```

Warning 2: “exploit3.c:51:3 Return value (type int) ignored: system(“/bin/sh”);”

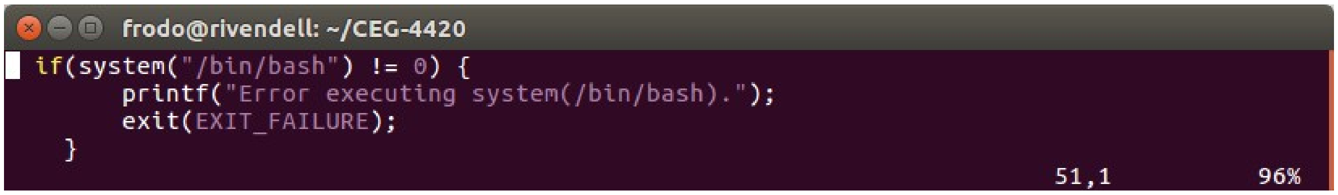
Because system() makes a system call, it returns an int which contains the return value of the system call. This value can be checked to ensure that the system call did not result in an error. It is good coding practice to check all errors to avoid problems that could result from assuming that a function ran with no error.

Before modification:



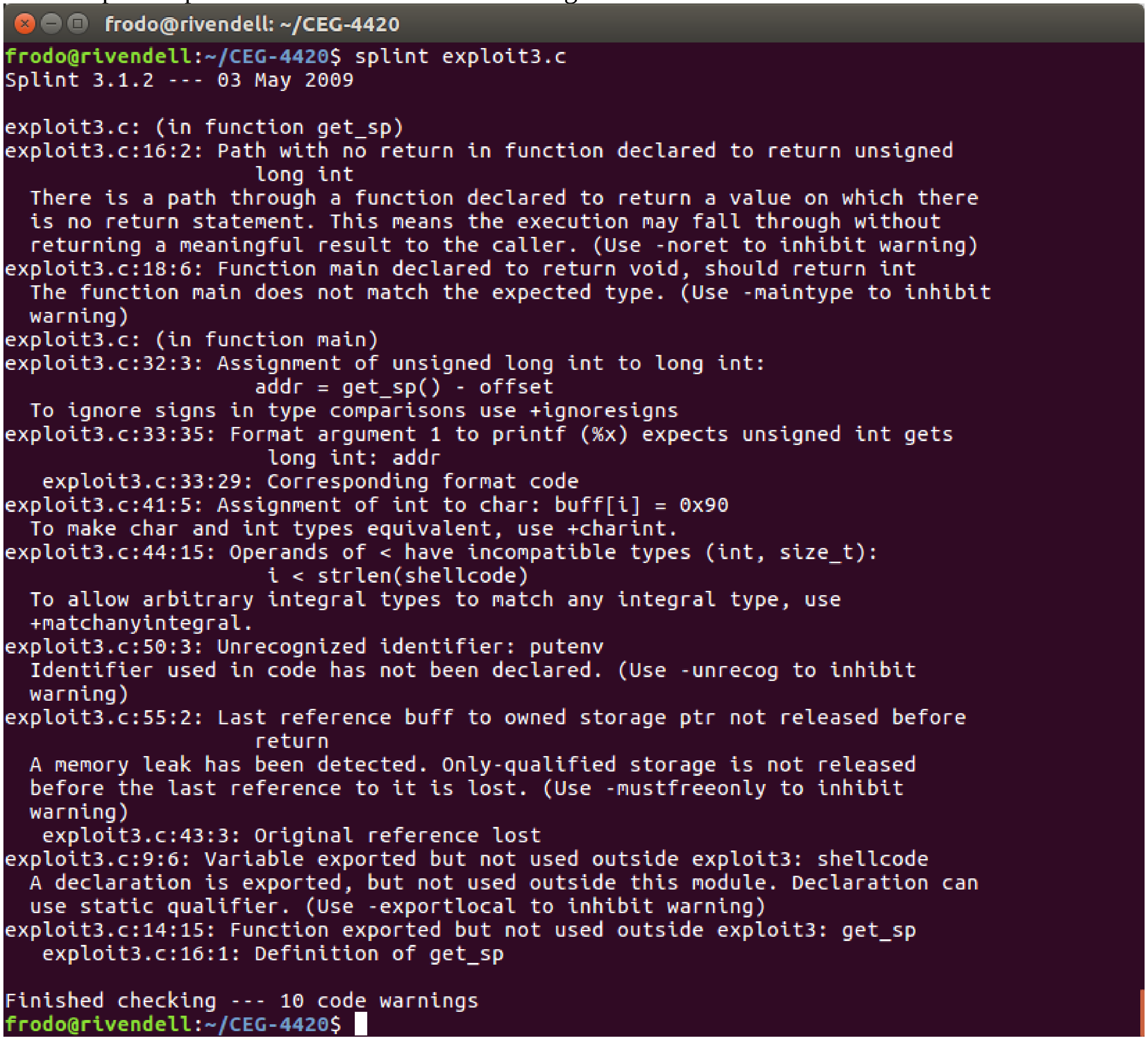
```
frodo@rivendell: ~/CEG-4420
  system("/bin/bash");
}

                                                        51,1          Bot
```

After modification: I used an if statement to check the return value for an error



```
frodo@rivendell: ~/CEG-4420
  if(system("/bin/bash") != 0) {
        printf("Error executing system(/bin/bash).");
        exit(EXIT_FAILURE);
    }
                                                        51,1          96%
```

Final output of splint after modification: 10 warnings



```
frodo@rivendell:~/CEG-4420$ splint exploit3.c
Splint 3.1.2 --- 03 May 2009

exploit3.c: (in function get_sp)
exploit3.c:16:2: Path with no return in function declared to return unsigned
                    long int
  There is a path through a function declared to return a value on which there
  is no return statement. This means the execution may fall through without
  returning a meaningful result to the caller. (Use -noret to inhibit warning)
exploit3.c:18:6: Function main declared to return void, should return int
  The function main does not match the expected type. (Use -maintype to inhibit
  warning)
exploit3.c: (in function main)
exploit3.c:32:3: Assignment of unsigned long int to long int:
                    addr = get_sp() - offset
  To ignore signs in type comparisons use +ignoresigns
exploit3.c:33:35: Format argument 1 to printf (%x) expects unsigned int gets
                     long int: addr
   exploit3.c:33:29: Corresponding format code
exploit3.c:41:5: Assignment of int to char: buff[i] = 0x90
  To make char and int types equivalent, use +charint.
exploit3.c:44:15: Operands of < have incompatible types (int, size_t):
                     i < strlen(shellcode)
  To allow arbitrary integral types to match any integral type, use
  +matchanyintegral.
exploit3.c:50:3: Unrecognized identifier: putenv
  Identifier used in code has not been declared. (Use -unrecog to inhibit
  warning)
exploit3.c:55:2: Last reference buff to owned storage ptr not released before
                    return
  A memory leak has been detected. Only-qualified storage is not released
  before the last reference to it is lost. (Use -mustfreeonly to inhibit
  warning)
   exploit3.c:43:3: Original reference lost
exploit3.c:9:6: Variable exported but not used outside exploit3: shellcode
  A declaration is exported, but not used outside this module. Declaration can
  use static qualifier. (Use -exportlocal to inhibit warning)
exploit3.c:14:15: Function exported but not used outside exploit3: get_sp
   exploit3.c:16:1: Definition of get_sp

Finished checking --- 10 code warnings
frodo@rivendell:~/CEG-4420$
```