

Question #1

When it comes to “Smash the Stack”, what are stack canaries? Give a definition and an example.

Answer #1

Stack Canaries (also called security cookies, an analogy to the harbinger canary birds used in coal mines) are secret values added to binaries during compilation to protect critical stack values like the Return Pointer against buffer overflow attacks.

They detect stack buffer overflows to prevent the execution of possibly malicious code by placing the secret value on the stack which changes every time the program is started. Prior to a function return, the stack canary is checked and if it appears to be modified, the program exits immediately.

Stack canaries make the exploitation of the “Smash the Stack” vulnerability more difficult, but not impossible.

Example:

Assembly without Canary	Assembly with Canary
<inside askUser function>	<inside askUser function>
	0x080485d9 mov eax, <canary>
	0x080485dc xor eax, <right canary val>
	0x080485e3 je 0x080485ea
	0x080485e5 call <__stack_chk_fail@plt>
0x0804856e leave	0x0804856e leave
0x0804856f ret	0x0804856f ret

```

The program being debugged has been started already.
Start it from the beginning? (y or n) y
Starting program: /home/osboxes/canaries/user3_input <<(python -c 'print "Hello
" + "\n" + "A"*100 + "C" * 4 + "D" * 12 + "BBBB"')
Tell me your name, please
Hi Hello
What color is your hat?

Breakpoint 1, 0x0804859b in askUser ()
(gdb) x/24x $esp+160
0xbffffef60:    0x00000000    0xbffff004    0xb7fbb000    0x0000f017
0xbffffef70:    0xffffffff    0x0000002f    0xb7e14dc8    0xb7fd51b0
0xbffffef80:    0x00008000    0xb7fbb000    0xb7fb9244    0xb7e200fc
0xbffffef90:    0x00000001    0x00000000    0xb7e36a60    0x08a5f900
0xbffffefa0:    0x00000001    0xbffff064    0xbffffefb8    0x08048511
0xbffffefb0:    0xb7fbb3dc    0xbffffefd0    0x00000000    0xb7e20647
(gdb) c
Continuing.

Breakpoint 2, 0x080485a0 in askUser ()
(gdb) x/24x $esp+160
0xbffffef60:    0x41414141    0x41414141    0x41414141    0x41414141
0xbffffef70:    0x41414141    0x41414141    0x41414141    0x41414141
0xbffffef80:    0x41414141    0x41414141    0x41414141    0x41414141
0xbffffef90:    0x41414141    0x41414141    0x41414141    0x43434343
0xbffffefa0:    0x44444444    0x44444444    0x44444444    0x42424242
0xbffffefb0:    0xb7fbb300    0xbffffefd0    0x00000000    0xb7e20647
(gdb)

```

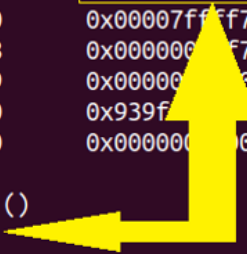
Note here that two distinct executions of the same program yields different stack canary values (in orange), preceding their respective return function pointers (in yellow).

```

Starting program: /home/osboxes/canaries/user3_input_can < <(python -c 'print "A"*100')
/bin/bash: python: command not found
Tell me your name, please

Breakpoint 1, 0x000055555555220 in askUser ()
(gdb) x/16x $rbp-32
0x7fffffff010: 0x0000000000000000      0x0000000000000000
0x7fffffff020: 0x0000555555552d0      0x75c55e80bc05af00
0x7fffffff030: 0x00007fffffff040      0x0000555555551db
0x7fffffff040: 0x0000555555552d0      0x00007ffff7df3cb2
0x7fffffff050: 0x00007fffffff138      0x00000000f7df3ad3
0x7fffffff060: 0x0000555555551c9      0x0000000000000000
0x7fffffff070: 0x0000000000000000      0x939f0000536ccb
0x7fffffff080: 0x0000555555550e0      0x0000000000000000
(gdb) bt
#0  0x000055555555220 in askUser ()
#1  0x0000555555551db in main ()
(gdb)

```



Note here that stack canaries can also be 64-bits (quad-words) in values, which exponentially increases the permutations required for a brute force attack.

Question #2

The following code contains errors that could be exploited. How would you fix it?

```
hashOut.data = hashes + SSL_MD5_DIGEST_LEN;
hashOut.length = SSL_SHA1_DIGEST_LEN;
if ((err = SSLFreeBuffer(&hashCtx)) != 0)
    goto fail;
if ((err = ReadyHash(&SSLHashSHA1, &hashCtx)) != 0)
    goto fail;
if ((err = SSLHashSHA1.update(&hashCtx, &clientRandom)) != 0)
    goto fail;
if ((err = SSLHashSHA1.update(&hashCtx, &serverRandom)) != 0)
    goto fail;
if ((err = SSLHashSHA1.update(&hashCtx, &signedParams)) != 0)
    goto fail;
    goto fail;
if ((err = SSLHashSHA1.final(&hashCtx, &hashOut)) != 0)
    goto fail;

err = sslRawVerify(...);
```

Answer #2

The error in the above code is fixed by discarding the duplicate '**goto fail;**' line of code present in line number 13.

Question #3

How could you catch this error in the future?

Answer #3

To avoid this error in the future, the user could:

- avoid insecure networks
- use a web filtering product that can scan HTTPS traffic (for example, Sophos UTM)
- switch to an alternative web browser (for example, Firefox or Chromium)

This error could be caught

- through negative testing in test cases (via dynamic analysis)
- through coverage analysis (via CI/CD tools)
- by properly detecting and checking unreachable code (by setting warning flags, via static analysis)
- by performing a duplicate lines detection (via static code analysis)
- by performing a manual code review
- by implementing the use of braces in conditions and loops (during development)
- by avoiding misleading indentation (during development)
- by setting the default value for the error code (0) as 'access failure'

Question #4

Heartbleed bug could be traced to a single line of code. What was that line of code?

Answer #4

The following line of code caused the Heartbleed bug:

```
memcpy (bp, pl, payload);
```