Question #1

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

Answer #1

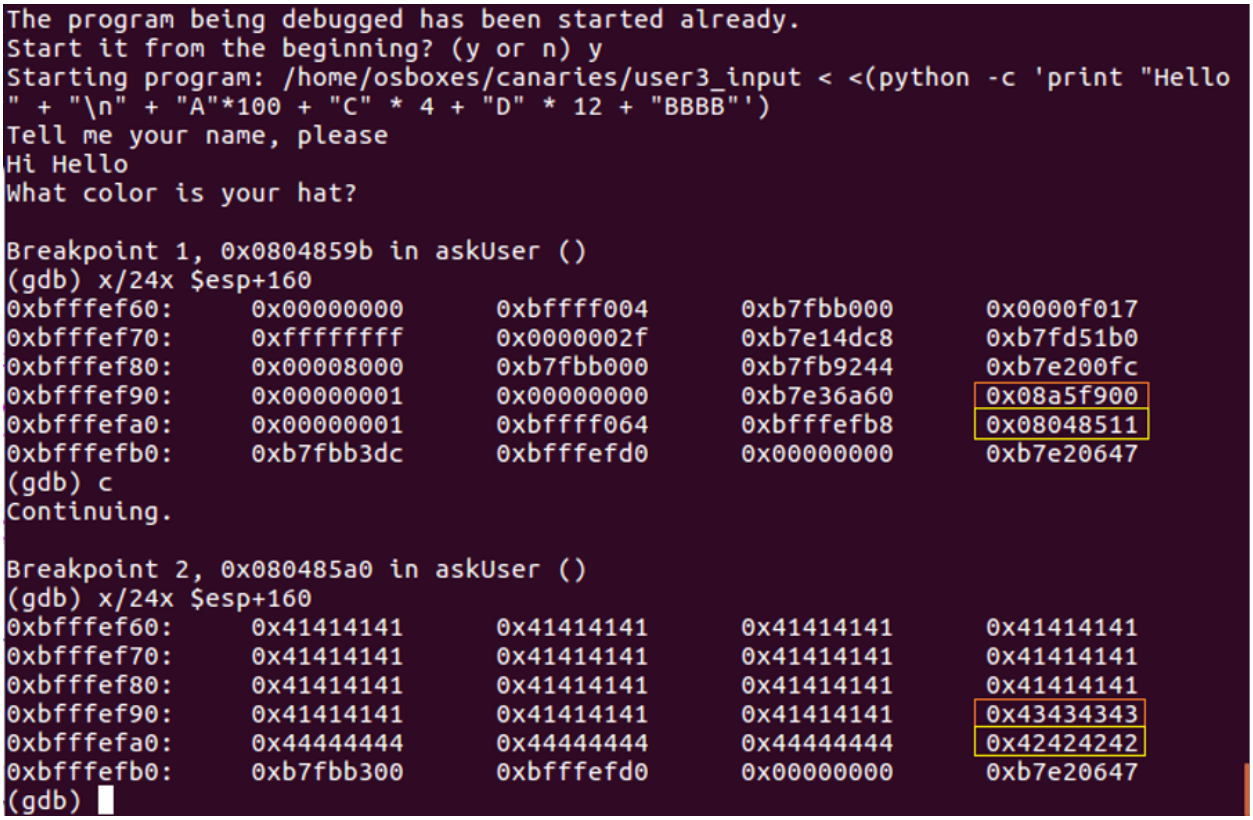
Stack Canaries (also called security cookies, an analogy to the harbinger canary birds used in coal mines) are secret values added to binaries during compilation to protect critical stack values like the Return Pointer against buffer overflow attacks.

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.

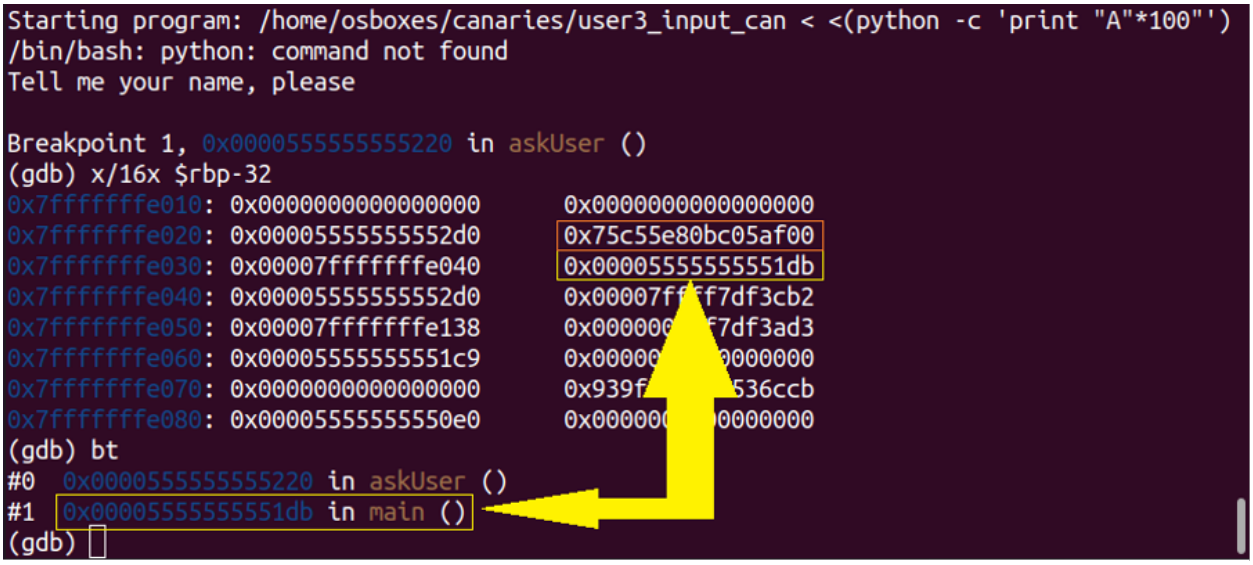
Stack canaries make the exploitation of the “Smash the Stack” vulnerability more difficult, but not impossible.

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



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).



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);
```