**Zadanie 6: Zraniteľnosť programov**

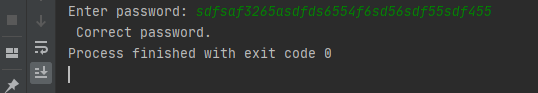
Cieľom šiesteho zadania je oboznámiť sa s problematikou zraniteľnosti aplikácií a možnosťami detekcie daných zraniteľnosti.

# Buffer overflow zraniteľnosť

Prvý príklad zranitelnosti som si vybral samozrejme buffer overflow.

static void vulnerable\_function() {  
 int password\_check = 0;  
 char password\_buf[MAX\_PASSWORD\_LENGTH] = {0};  
  
 printf("Enter password: ");  
 gets(password\_buf);  
  
 if (strcmp(password\_buf, DEFAULT\_PASSWORD\_PLAIN) == 0)  
 password\_check = 1;  
  
 if(password\_check)  
 printf("Correct password.");  
 else  
 printf("Incorrect password.");  
}

Po zadani dostatocne dlheho hesla (input string) pretecie premena password\_buf. Tymto sa prepise miesto v pamati kde je premnea password\_check a teda podminka a kontrtola bude vzdy spravna ked hodnota bude ina ako nula.

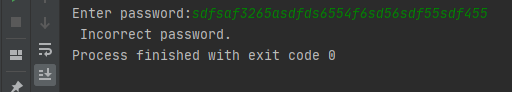


# Buffer overflow ošetrenie

Hlavnym problemom bolo pouzitie funnkcie gets, ktora nekontroluje dlzku vstupu. Funksia fgets riesi tento nedostatok.

static void secure\_function(){  
 char password\_buf[MAX\_PASSWORD\_LENGTH] = {0};  
  
 printf("Enter password: ");  
 fgets(password\_buf, MAX\_PASSWORD\_LENGTH, stdin);  
  
 if (strncmp(password\_buf, DEFAULT\_PASSWORD\_PLAIN, MAX\_PASSWORD\_LENGTH) == 0)  
 printf("Correct password.");  
 else  
 printf("Incorrect password.");  
  
}

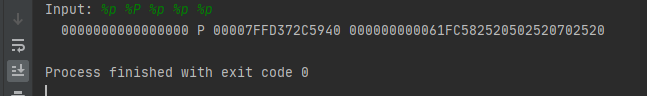
Po osetreni vidime, ze heslo je spravne detekovane ako nespravne.



# Format string zraniteľnosť

Jazyk C ma zranitelnost pri formatovani stdout cez printf, utocnik moze zadat formaty ako 5p alebo 5n ktorym si necha vypisat call stack pamatove hodnoty. Vdaka comu moze neskor executovat prave podla tychto hodnout a volat funkcie a instrukcie programu.

static void vulnerable\_function() {  
 char user\_input[MAX\_INPUT\_LENGTH] = {0};  
  
 printf("Input: ");  
 fgets(user\_input, MAX\_INPUT\_LENGTH, stdin);  
  
 printf(user\_input);  
}

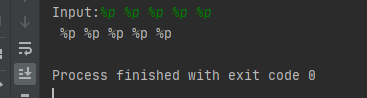


# Format string ošetrenie

Riesenie pouzie spravne fprintf funkcie pomocou %s .

static void secure\_function(){  
 char user\_input[MAX\_INPUT\_LENGTH] = {0};  
  
 printf("Input: ");  
 fgets(user\_input, MAX\_INPUT\_LENGTH, stdin);  
  
 printf("%s", user\_input);  
}

Po ostrenie sa zadany input neprejavy ako format ale ako hodnota charakterov.



# Sprintf zraniteľnosť

Sprintf je nebezpecna funkcia v jazyku C.

static void vulnerable\_function() {  
 char buffer[BUFFER\_SIZE];  
 int check = 0;  
  
 sprintf(buffer, "%s", "This string is too long!");  
  
 printf("check: %d", check);  
}

Nastalo pretecenie a tam, kde by mala byt hodnota 0 je 1936269415.



# Sprintf ošetrenie

Lepsie je pouzivat snprintf.

static void secure\_function(){  
 char buffer[BUFFER\_SIZE];  
  
 int length = snprintf(buffer, BUFFER\_SIZE, "%s%s", "long-name", "suffix");  
  
 if (length >= BUFFER\_SIZE) {  
 printf("String truncation!");  
 }  
}

# File opening zraniteľnosť

Ak nekontrolujeme ci file existuje, utocnik moze vytvorit symbol link, napriklad na subor s heslami.

static int vulnerable\_function() {  
 if (!access(MY\_TMP\_FILE, F\_OK)) {  
 printf("File exists!\n");  
 return EXIT\_FAILURE;  
 }  
 /\* At this point the attacker creates a symlink from /tmp/file.tmp to /etc/passwd \*/  
 FILE \* tmpFile = fopen(MY\_TMP\_FILE, "w");  
 if (tmpFile == NULL) {  
 return EXIT\_FAILURE;  
 }  
 fputs("Some text...\n", tmpFile);  
 fclose(tmpFile);  
  
 return 0;  
}

# File opening zraniteľnosť

Kontorlovat ci file existuje.

static int secure\_function(){  
 int fd;  
 FILE\* f;  
  
 /\* Odstranit mozne symlinks \*/  
 unlink(MY\_TMP\_FILE);  
  
 fd = open(MY\_TMP\_FILE, O\_WRONLY|O\_CREAT|O\_EXCL, FILE\_MODE);  
 if (fd == -1) {  
 perror("Failed to open the file");  
 return EXIT\_FAILURE;  
 }  
  
 f = fdopen(fd, "w");  
 if (f == NULL) {  
 perror("Failed to associate file descriptor with a stream");  
 return EXIT\_FAILURE;  
 }  
 fprintf(f, "Hello, world\n");  
 fclose(f);  
  
 return EXIT\_SUCCESS;  
  
}