Experiment 10

Date of Performance : 17-04-2023

Date of Submission : 07-05-2023

SAP Id: 60004200097

Name : Marwin Shroff

Div: B

Batch : B1

Aim of Experiment

Implement Buffer Overflow Attack.

Theory / Algorithm / Conceptual Description

Buffer Overflow Attack: Attackers exploit buffer overflow issues by overwriting the memory of an application. This changes the execution path of the program, triggering a response that damages files or exposes private information. For example, an attacker may introduce extra code, sending new instructions to the application to gain access to IT systems. If attackers know the memory layout of a program, they can intentionally feed input that the buffer cannot store, and overwrite areas that hold executable code, replacing it with their own code. For example, an attacker can overwrite a pointer (an object that points to another area in memory) and point it to an exploit payload, to gain control over the program. Stack-based buffer overflows are more common, and leverage stack memory that only exists during the execution time of a function. Heap-based attacks are harder to carry out and involve flooding the memory space allocated for a program beyond memory used for current runtime operations.

Ollydbg: OllyDbg (named after its author, Oleh Yuschuk) is an x86 debugger that emphasizes binary code analysis, which is useful when source code is not available. It traces registers, recognizes procedures, API calls, switches, tables, constants and strings, as well as locates routines from object files and libraries. It has a user friendly interface, and its functionality can be extended by third-party plugins. OllyDbg is often used for reverse engineering of programs. It is often used by crackers to crack software made by other developers. For cracking and reverse engineering, it is often the primary tool because of its ease of use and availability; any 32-bit executable can be used by the debugger and edited in bitcode/assembly in realtime.It is also useful for programmers to ensure that their program is running as intended, and for malware analysis purposes.

Splint: Splint is a tool for statically checking C programs for security vulnerabilities and coding mistakes. With minimal effort, Splint can be used as a better lint. If additional effort is invested adding annotations to programs, Splint can perform stronger checking than can be done by any standard lint. Splint has the ability to interpret special annotations to the source code, which gives it stronger checking than is possible just by looking at the source alone. Splint is used by gpsd as part of an effort to design for zero defects.

Cppcheck: Cppcheck is a static code analysis tool for the C and C++ programming languages. It is a versatile tool that can check non-standard code. Cppcheck supports a wide variety of static checks that may not be covered by the compiler itself. These checks are static analysis checks that can be performed at a source code level. The program is directed towards static analysis checks that are rigorous, rather than heuristic in nature. Some of the checks that are supported include:

- Automatic variable checking
- Bounds checking for array overruns
- Classes checking (e.g. unused functions, variable initialization and memory duplication)

Program

Code with Buffer Overflow

```
#include <stdio.h>
#include <string.h>
#define UP_MAXLEN 20
#define UP_PAIR_COUNT 3
int main() {
int flag;
char termBuf;
char username[UP_MAXLEN];
char cpass[UP_MAXLEN];
char npass[UP_MAXLEN];
char keys[UP_PAIR_COUNT][2][UP_MAXLEN] = {
{
"Admin",
"pass3693"
},
{
"Marwin",
"60004200097"
},
{
"Sally",
"Usfsmfs"
}
};
while (1) {
flag = 0;
printf("Change Password\n");
printf("Enter Username: ");
```

```
gets(username);
printf("Enter Current Password: ");
gets(cpass);
for (int i = 0; i < UP_PAIR_COUNT; i++) {
if (strcmp(keys[i][0], username) == 0 && strcmp(keys[i][1],
cpass) == 0) {
printf("Enter New Password: ");
gets(npass);
strcpy( & keys[i][1][0], npass);
for (int j = 0; j < UP_PAIR_COUNT; j++) printf("%s |
%s\n", keys[j][0], keys[j][1]);
printf("Password Changed!\n");
printf("Continue? Y/N: ");
gets( & termBuf);
if (termBuf != 'Y') return 0;
else flag = 1;
}
}
if (flag == 1) continue;
printf("Incorrect Username and Password. Enter Y to
continue.\n");
gets( & termBuf);
if (termBuf != 'Y') return 0;
}
}
```

Output:

```
Change Password
Enter Username: Marwin
Enter Current Password: 60004200097
Enter New Password: hellothisismarwin
Admin | pass3693
Marwin | hellothisismarwin
Sally | Usfsmfs
Password Changed!
Continue? Y/N: Y
Change Password
Enter Username: Marwin
Enter Current Password: hdvghdsgf
Incorrect Username and Password. Enter Y to continue.
Y
```

Program

Code after fixing the Buffer Overflow Vulnerability

```
#include <stdio.h>
#include <string.h>
#define UP_MAXLEN 20
#define UP_PAIR_COUNT 3
int main() {
int flag;
char termBuf;
char username[UP_MAXLEN];
char cpass[UP_MAXLEN];
char npass[UP_MAXLEN];
char keys[UP_PAIR_COUNT][2][UP_MAXLEN] = {
{
"Admin",
"pass3693"
},
{
"Max",
"Qqkaif"
},
{
"Sally",
"Usfsmfs"
}
};
while (1) {
flag = 0;
printf("Change Password\n");
printf("Enter Username: ");
fgets(username, UP_MAXLEN, stdin);
username[strcspn(username, "\r\n")] = 0;
printf("Enter Current Password: ");
fgets(cpass, UP_MAXLEN, stdin);
```

```
cpass[strcspn(cpass, "\r\n")] = 0;
for (int i = 0; i < UP_PAIR_COUNT; i++) {
if (strcmp(keys[i][0], username) == 0 && strcmp(keys[i][1],
cpass) == 0) {
printf("Enter New Password: ");
fgets(npass, UP_MAXLEN, stdin);
npass[strcspn(npass, "\n")] = 0;
strcpy( & keys[i][1][0], npass);
for (int j = 0; j < UP_PAIR_COUNT; j++) printf("%s |
%s\n", keys[j][0], keys[j][1]);
printf("Password Changed!\n");
printf("Continue? Y/N: ");
scanf("%c", & termBuf);
if (termBuf != 'Y') return 0;
else flag = 1;
while ((termBuf = getchar()) != '\n' && termBuf != EOF);
}
}
if (flag == 1) continue;
printf("Incorrect Username and Password. Enter Y to
continue.\n");
scanf("%c", & termBuf);
if (termBuf != 'Y') return 0;
while ((termBuf = getchar()) != '\n' && termBuf != EOF);
}
}
```

Output:

```
Change Password
Enter Username: Max
Enter Current Password: Qqkaif
Enter New Password: marwinisgod
Admin | pass3693
Max | marwinisgod
Sally | Usfsmfs
Password Changed!
```

CONCLUSION

Thus, Buffer Overflow Attack has been successfully demonstrated and prevented using the Splint programming tool