электронный журнал
МОЛОДЕЖНЫЙ НАУЧНО-ТЕХНИЧЕСКИЙ ВЕСТНИК
Издатель ФГБОУ ВПО "МГТУ им. Н.Э. Баумана". Эл No. ФС77-51038.

 # 07, июль 2017

УДК 004.491.22

Перехват управления программой посредством атаки «Переполнение буфера»

Ковалевский А.Е., *студент*
Россия, 105005, г. Москва, МГТУ им. Н.Э. Баумана,
кафедра «Защита информации»

Научный руководитель: Томах О.С., ассистент
Россия, 105005, г. Москва, МГТУ им. Н.Э. Баумана,
кафедра «Защита информации»

Поставим себе целью продемонстрировать уязвимость программ посредством переполнения буфера и дальнейшим перехватом управления программой. В качестве стенда (Build, Host & Target systems) используется: ОС Windows 7 SP1 x64 с MS Visual Studio 2013. Для начала напишем простую программу, подав которой неправильный аргумент, можно легко переполнить стек (таблица 1).

Таблица 1

Код программы

```
#include <stdio.h>
void print_array(int size, char array[])
{
	char buffer[32]; int i;
	for (i = 0; i < size; i++) buffer[i] = array[i];
	printf(buffer);
}

int main()
{
	char temp[] = "1234567890";
	print_array(10, temp);
	return 0;
}
```

Как мы видим в этой программе функция print_array просто выводит массив на экран. Конечно же, здесь очевидна ошибка в виде отсутствия проверки на переполнение буфера, но для исследования переполнения стека самое то. Ведь данную программу можно назвать моделью переполнения буфера в большой и сложной программе, например, операционной системе. Программисты крайне не любят подобные ошибки. «Чаще всего вызывает опасения результирующая строка BUFFER OVERFLOW (переполнение буфера)» [1, с. 349]. Очевидно, что, если функции подать массив, превышающий размер буфера, то произойдет перезапись памяти за границами переменной buffer.

Проверим, что же случится при переполнении буфера. Мы вызовем функцию с заведомо большим размером строки, чем буфер: char temp[] = "1111122222333334444455555666667777788888999990"

При запуске программы наша IDE столкнется с ошибкой. «Она выводит окно, указывающее на то, что поток в процессе вызвал необрабатываемое им исключение, и предлагает либо закрыть процесс, либо начать его отладку» [2, с.623]. Run-Time Check Failure #2 - Stack around the variable 'buffer' was corrupted. Необработанное исключение по адресу 0x39393939 в Переполнение буфера.exe: 0xC0000005: нарушение прав доступа при исполнении по адресу 0x39393939.

Также просмотрим значения регистров в этот момент: EAX = 0000002E EBX = 7EFDE000 ECX = FEC8F1B6 EDX = 0FEF7310 ESI = 00A45887 EDI = 003EFE47 EIP = 39393939 ESP = 003EFE0C EBP = 38383838 EFL = 00010246

Теперь стоит открыть таблицу DEC-HEX чисел и увидеть там, что 39h - это код символа "9", 35h - код символа "5", а 30h - код символа "0", можно догадаться, что куски строки ("111 ... 00"), которые не поместились в нашем буфере, оказались в разных регистрах, а последние символы остались на стеке.

Разберемся, что произошло. Посмотрим, как происходит вызов нашей функции и передача ей аргументов (рис. 1).

```
        print_array(47, temp);
012D10E9 8D 45 CC       lea     eax,[temp]
012D10EC 50             push    eax
012D10ED 6A 2F          push    2Fh
012D10EF E8 11 FF FF FF call    print_array (012D1005h)
```

Рис. 1. Вызов функции

То есть мы видим, что аргументы передаются в обратном порядке: сначала указатель на строку, а потом и число 47 == $2F_{16}$. Перед тем, как передать управление функции print_array, инструкция CALL помещает на стек значение регистра EIP, т.е. адрес возврата. Поэтому перед выполнением print_array стек находится в следующем состоянии (рис. 2).

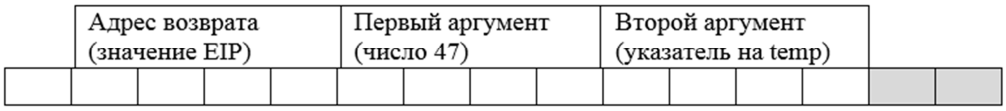

Рис. 2. Стек в момент вызова функции

Далее управление переходит к print_array и перед началом работы процедуры происходит примерно следующая последовательность инструкций (рис. 3).

```
void _stdcall print_array(int size, char array[])
{
01361020 55             push    ebp
01361021 8B EC          mov     ebp,esp
01361023 83 EC 2C       sub     esp,2Ch
01361026 56             push    esi
01361027 57             push    edi
```

Рис. 3. Заголовок функции

То есть первым делом на стеке сохраняется значение EBP, затем в EBP помещается значение ESP и уже потом от ESP вычитается 44. Операции с EBP мы рассматривать не будем; достаточно сказать, что относительно EBP адресуются локальные переменные. Нам интересна команда sub esp,2Ch . Тем самым функция резервирует на стеке место для своих локальных переменных. Их у неё две - char buffer[32] и int i. Массив buffer занимает 32 байта, целое число i - 4 байта. Итого 36 байт. Но ввиду особенностей компилятора между i и buffer[32] резервируются 4 байта и еще между стеком и buffer[32]. В результате стек выглядит так, как показано на рис. 4.

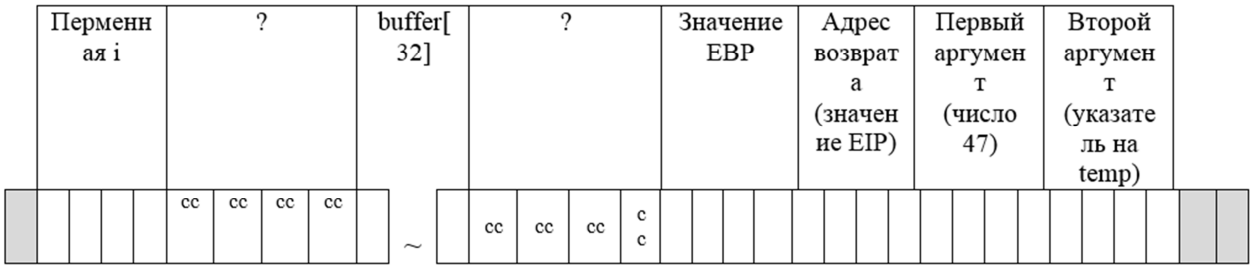

Рис. 4. Состояние стека в момент переполнения

«Теперь должно быть понятно, куда попадают байты, не поместившиеся в буфер. Они записываются на место сохранённого ранее EBP, переписывают адрес возврата и так далее пока их хватит» (рис. 5) [3].

```
00391095 8B E5          mov     esp,ebp
00391097 5D             pop     ebp
00391098 C2 08 00       ret     8
```

Рис. 5. Возврат из функции

Таким образом, правильно подобрав строку, мы можем осознанно указать адрес возврата и перехватить управления программой. Очевидно, что адрес возврата должен находиться в рамках адресного пространства данной программы. Для этого можно использовать внедрение в адресное пространство процесса своей (пользовательской) DLL-библиотеки. В которой может быть реализована программа любой сложности. Для внедрения DLL могут использоваться разные способы:

1. Правка раздела импорта программы
2. Добавление пользовательской библиотеки для загрузки по-умолчанию ко всем процессам.
3. Запуск программы из контекста другой (лаунчер)

Однако уже сейчас в компиляторах применяются автоматические проверки на сохранность стека и защиту от переполнения буфера. Стоит отметить, что для демонстрации данного примера пришлось отключить пару проверок компилятора, поскольку он сразу же блокировал выполнение программы при повреждении стека. Тем не менее в настоящее время используются тысячи программ, в которых уязвимости различного рода не были еще найдены, и за нахождение которых, заинтересованные сайта или сами компании-производители дают немалую награду.

Список литературы

[1]. Соломон Д., Руссинович М. Внутреннее устройство Microsoft Windows 2000. Мастер-класс. СПб.: Питер; М.: Издательско-торговый дом «Русская редакция» 2004. 746 с.

[2]. Рихтер Дж. Windows для профессионалов: создание эффективных Win32 приложений с учетом специфики 64-разрядной версии Windows. 4-е изд. СПб; Питер; М.: Издательско-торговый дом «Русская Редакция», 2001. 752 с.

[3]. Третьяков К. Переполнение буфера. Режим доступа: http://www.codenet.ru/progr/asm/overflow.php (дата обращения 11.04.2017).