Пример использования техники переполнения стека для организации атаки

Технику использования переполнения стека для организации атаки мы хотим проиллюстрировать ставшим уже классическим примером программы на языке C из книги Ховарда и Лебланка “Защищенный код” [1]. Этот пример (в котором английские комментарии заменены на русские) состоит из тела программы `StackOverun.c` (функция `main`) и двух функций `foo` и `bar`. Функция `foo` является атакуемой программой, а функция `bar` – вредоносной программой, которую злоумышленник пытается выполнить.

```
/*
 StackOverrun.c
Эта программа является примером того, как переполнение стека
 может быть использовано для выполнения произвольного кода.
Для достижения этой цели нужно найти вид строки исходных данных, ввод которой
приведет к выполнению функции bar
 */
#include <string.h>
#include <stdio.h>
void foo(const char* input)
{
   char buf[10];




   printf("Мой стек выглядит так:\n%p\n%p\n%p\n%p\n%p\n%p\n% p\n\n");
   strcpy(buf, input);
   printf("%s\n", buf);
   printf("Теперь мой стек выглядит так:\n%p\n%p\n%p\n%p\n%p\n%p\n%p\n\n");
}
void bar(void)
{
   printf("Aaa! Меня взломали!\n");
}
int main(int argc, char* argv[])
{
    printf("Address of foo = %p\n", foo);
   printf("Address of bar = %p\n", bar);
   if (argc != 2)
```

[1] http://www.ozon.ru/context/detail/id/1387492/

```
{
    printf("Пожалуйста введите строку как аргумент!\n");
    return -1;
        }
foo(argv[1]);
  return 0;
}
```

Функция `foo (const char* input)` выполняет полезную работу – она печатает строку, передаваемую ей в качестве аргумента. Для этого она сначала копирует аргумент в локальный массив символов `buf[10]`, который вмещает не более 10 символов. При этом функция `foo` не проверяет длину строки-аргумента, чем воображаемый злоумышленник и воспользуется в нашем примере. Функция `bar` нигде явно не вызывается, задача злоумышленника состоит в вызове ее за счет переполнения стека. Если это удастся, то на экране должен появиться текст «Ааа! Меня взломали!»

Для целей демонстрации в `main` и `foo` добавлены строки, печатающие на экране вспомогательную информацию: адреса функций `foo` и `bar`, а также состояние стека до копирования строки аргумента в буфер и после. Информация выводится с использованием библиотечной функции `printf`, которая при задания формата %p печатает адрес имени переменной или функции, а не ее значение, например

```
printf("Address of foo = %p\n", foo);
```

печатает адрес функции `foo`.

Та же функция `printf`, вызванная без второго параметра, распечатывает содержимое стека при работе функции `foo` – первый раз распечатывается 6 первых слов (в 32-разрядной машине это соответствует 4 однобайтным символам кода ASCII) стека до копирования строки-аргумента в локальный массив `buf[10]`, а второй раз –после.

Ниже показаны результаты работа программы `StackOverrun` при задании ей в качестве аргумента строки “`Hello`”. Эта строка состоит из 5 символов, так что она умещается в массив `buf[10]` без проблем. Мы видим результат такого копирования в первых двух словах стека, распечатанного функцией `foo` – первое слово содержит шестнадцатеричные коды 48 (H), 65 (e), 6C (l), 6C (l), а второе содержит код 6F (o).

```
C:\Secureco2\Chapter05>StackOverrun.exe Hello
Address of foo = 00401000
Address of bar = 00401045
Мой стек выглядит так:
00000000
00000000
7FFDF000
0012FF80
0040108A <-- адрес возврата из foo, который мы пытаемся переписать
00410EDE
Hello
```

А сейчас мой стек выглядит так:

6C6C6548 <-- Сюда скопирована строка "Hello”

0000006F

7FFDF000

0012FF80

0040108A

00410EDE

Адрес возврата из `foo` в `main` равен 0040108A, и между ним и кодами “`Hello`” имеется достаточно безопасный зазор. Адрес функции `bar` равен 00401045, и наша цель будет достигнута, если мы сможем заменить адрес возврата на это значение.

Сначала проверим, можно ли нарушить нормальную работу программы, если длина строки-аргумента будет больше 10 символов. Вызовем ее с аргументом “AAAAAAAAAAAAAAAAAAAAAAAA”, который состоит из 24 символов с шестнадцатеричным кодом 41(A). Результат такого вызова показан ниже. Видно, что весь стек оказался заполнен кодом 41, в том числе таким оказался и адрес возврата из функции `foo`. После распечатки результатов программа аварийно завершается операционной системой с ошибкой недопустимого обращения к адресу 414141.

C:\Secureco2\Chapter05>StackOverrun.exe AAAAAAAAAAAAAAAAAAAAAAAA

Address of foo = 00401000

Address of bar = 00401045

Мой стек выглядит так:

00000000

00000000

7FFDF000

0012FF80

0040108A <-- адрес возврата из foo, который мы пытаемся переписать

00410EDE

AAAAAAAAAAAAAAAAAAAAAAAA

А сейчас мой стек выглядит так:

41414141 <-- Сюда скопирована строка ” AAAAAAAAAAAAAAAAAAAAAAAA”

41414141

41414141

41414141

41414141<-- адрес возврата из foo

41414141

Теперь, когда видно, что программа действительно уязвима из-за переполнения стека, осталось только подобрать нужную строку-аргумент, чтобы в нужном месте стека оказался адрес функции bar (00401045). Такой строкой будет, например строка “ABCDEFGHIJKLMNOP\0x40\0x10\0x45”, в которой последние три символа заданы в шестнадцатеричном виде. Мы опустим детали задания строки, содержащей спецсимвол 0x10 (авторы делают это с помощью простого perl-скрипта) и ниже покажем результат работы `StackOverrun` с такой строкой-аргументов.

C:\Secureco2\Chapter05>StackOverrun.exe “ABCDEFGHIJKLMNOP\0x40\0x10\0x45”

Address of foo = 00401000

Address of bar = 00401045

Мой стек выглядит так:

00000000

00000000

7FFDF000

0012FF80

0040108A <-- адрес возврата из foo, который мы пытаемся переписать

00410EDE

ABCDEFGHIJKLMNOP@ E

А сейчас мой стек выглядит так:

44434241 <-- Сюда скопирована строка “ABCDEFGHIJKLMNOP\0x40\0x10\0x45”

48474645

4C4B4A49

504F4E4D

00401045<-- адрес возврата из foo

00410EDE

Ааа! Меня взломали!

Так как функция `bar` распечатала свою строку Ааа! Меня взломали!, значит, мы добились нужного результата – смогли заменить адрес возврата из функции `foo` в главную программу `main` на адрес функции `bar`. Об этом говорит и адрес возврата, распечатанный функцией `foo`.