ДИСЦИПЛИНА

"Программно-аппаратные средства обеспечения информационной безопасности (ПАСО)"

Тема 3. Защита от вредоносного программного обеспечения

Занятие 3/3. Групповое занятие

Тема:

Механизмы реализации атак "переполнения буфера"

Учебные вопросы:

1. Классификация атак "переполнение буфера"
2. Особенности реализации различных типов атак "переполнение буфера"

Цели занятия

- Уяснить сущность атаки "переполнения буфера"
- Изучить классификацию и способы реализации атак "переполнение буфера"

Литература

Основная

1. Программно-аппаратные средства обеспечения информационной безопасности. В 2 ч. Ч. 1. Защита от разрушающих программных средств : пособие / А. Г. Мацкевич, С. В. Снигирев, Д. А. Свечников. – Орёл : Академия ФСО России, 2011. – 141 с.

2. Программно-аппаратные средства обеспечения информационной безопасности. В 2 ч. Ч. 2. Методы и средства локальной защиты ПЭВМ / А. В. Козачок [и др.]. – Орёл : Академия ФСО России, 2015. – 143 с.

3. Методы и протоколы аутентификации: пособие: в 2 ч. Ч. 1 / Д. Е. Шугуров [и др.]. – Орел : Академия ФСО России, 2013. – 219.

Дополнительная

1. Панов, А. С. Реверсинг и защита программ от взлома. – Санкт-Петербург : БХВ-Петербург, 2006. – 256 с.: ил. ISBN 5-94157-889-X

2. Оголюк, А. А. Защита приложений от модификации : учебное пособие. – Санкт-Петербург : СПбГУ ИТМО, 2016. – 56 с.

3. Проскурин, В. Г. Защита программ и данных : учеб. пособие для студ. учреждений высш. проф. образования / В. Г. Проскурин. – 2-е изд., стер. – Москва : Издательский центр «Академия», 2012. – 208 с. ISBN 978-5-7695-9288-1

4. Программно-аппаратная защита информации: уч. пособие / П. Б. Хорев. – Москва : ИД «ФОРУМ», 2009.

Примерные вопросы для контроля готовности к занятию

1. Дать определение ВПО, классы ВПО.
2. Виды нелегитимных операций, признаки наличия ВПО.
3. Классификация ВПО по объекту заражения.
4. Классификация ВПО по способу внедрения.
5. Методы встраивания программных закладок.
6. Потенциально опасные функции программных закладок.
7. Этапы жизненного цикла ВПО.
8. Этапы жизненного цикла вирусов, стадии исполнения вирусов.
9. Методы самозащиты ВПО.
10. Дать определения и указать отличия "сетевых червей" от средств проникновения в удаленные системы.

ВОПРОС 1

Классификация атак
"переполнение буфера"

Основные понятия

Переполнение буфера (Buffer Overflow) – явление, возникающее, когда компьютерная программа записывает (читает) данные за пределами выделенного в памяти буфера.

Атаки переполнения буфера – это класс компьютерных атак, основанных на возможности изменить ход исполнения атакуемой программы с помощью специально сформированных некорректных входных данных, вызывающих переполнение буфера.

Основа атак переполнения буфера – это получение прав и привилегий атакуемой программы после изменения хода выполнения программы и передачи управления на новый код злоумышленника.

Цели атак переполнения буфера

1) Чтение секретных данных
2) Модификация секретных переменных
3) Передача управления на секретную функцию программы
4) Передача управления на код, переданный жертве самим злоумышленником

Реализация атаки требует решения 2-х задач

1) **подготовка кода**, который будет выполняться в контексте атакуемой программы;

2) **изменение последовательности выполнения программы** с передачей управления подготовленному коду.

Варианты решения задачи подготовки кода

1) подготавливаемый код представляет собой исполняемые машинные инструкции некоторого процессора и может передаваться в программу в качестве ее параметра или команды;

Этот код сохраняется в некоторой **области памяти**:

1) стек (такие атаки называются - stack buffer overflow);

2) куча (такие атаки называются - heap buffer overflow);

3) в области статических данных.

2) код для атаки уже присутствует в программе или ее адресном пространстве, требуется его параметризация;

3) параметризованный код уже расположен в программе или ее адресном пространстве, т.е. подготовка кода не нужна.

Варианты решения задачи передачи управления подготовленному коду

1. искажение адреса возврата из функции
2. искажение указателя функции
3. искажение таблиц переходов
4. искажение указателей данных

1

Суть реализации атаки переполнения буфера посредством искажения адреса возврата из функции

1

Суть реализации атаки переполнения буфера посредством искажения адреса возврата из функции

Программа:

```
1: int test(char *big)
2: {
3:       char buffer[4];      // переполняемый буфер
4:       strcpy(buffer,big);  // переполнение буфера
5:       return 0;
6: }


7:  int main (int argc, char *argv[])
8:  {
9:  char big[1024];
10: gets(big);    //  ввод строки
11: test(big);    //  вызов уязвимой функции
12: ret_address_form_test:
13: return 0;
14: }
```

Состояние стека:

(формирование происходит по ходу выполнения программы)

```
// аргументы функции main
0x0012FFFC:  argc
0x0012FFF8:  argv
// адрес возврата из main
0x0012FFF4:  ret_address_from_main
// локальные переменные функции main
0x0012FFF0 - 0x0012FBF0: big[1024]
// аргументы функции test
0x0012FBEC:    big
// адрес возврата из test
0x0012FBE8:    ret_address_from_test
// локальные переменные функции test
0x0012FBE4 - 0x0012FE0: buffer[4]
```

Если строка **big** длиннее 4 символов, то будет перезаписан ад возврата ret_addresss

1 Суть реализации атаки переполнения буфера посредством искажения адреса возврата из функции

Программа на языке Ассемблера

```
                …
                mov     eax, 0
EIP →           push    0x1111 2222
                push    0x3333 4444
                call    checkX
ret_address:
                cmp     eax, 1
                ret
checkX proc
                sub     esp, 8
                mov     ecx, 12
                mov     esi, 0x5000 0000
                mov     edi, 0x4000 0000
                rep     movsb
                add     esp, 8
                ret
checkX endp
```

Состояние стека

	Адрес	*Значение*
	0x0012 FFFF	вершина стека
	…	…
ESP →	0x0012 FFC8	0
	0x0012 FFC4	0x1111 2222
	0x0012 FFC0	0x3333 4444
	0x0012 FFBC	ret_address
	0x0012 FFB8	???
	0x0012 FFB4	???
	0x0012 FFB0	
	…	
	0x0000 0000	

Передаем управление на инструкцию по адресу возврата

2

Суть реализации атаки переполнения буфера посредством искажения указателя функции

```
1:    void dummy(void)
2:    {
3:       printf("Hellow World !!!");
4:    }


5:    int main(int argc, char ** argv)
6:    {
7:       void (*dummyptr)();              // в стеке выделить 4 байта
8:       char buffer[10];                 // в стеке выделить 10 байт
9:       dummyptr = dummy;
10:      strcpy(buffer, argv[1]);         // реализация уязвимости
11:      (*dummyptr)();
12:      return 0;
13:   }
```

Если строка **argv[1]** длиннее 10 символов, то будет перезаписан адрес функции **dummy**

4

Суть реализации атаки переполнения буфера посредством искажения указателя на данные

```
1:void dummy(char * arg)
2:{
3:   char * p = arg;   // в стеке выделено 4 байта под указатель
4:   char   a[10];     // в стеке выделено под строку 10 байт
5:   gets(a);      // записать в буфер a строку
6:   gets(p);      // записать по адресу p еще одну строку
7:}
```

Если строка **a** длиннее 10 символов, то будет перезаписан адрес, хранящийся в переменной **p**

Классификация атак переполнения буфера

	Внедрение кода	Внедрение параметров	Внедрение кода и параметров не требуется
Искажение адреса возврата из функции	Атака «срыв стека»	Атака «срыв стека» с параметризацией	Атака «срыв стека» с передачей управления
Искажение указателей функций	Атака на указатели функций	Атака на указатели функций с параметризацией	Атака на указатели функций с передачей управления
Искажение таблиц переходов	Атака на таблицы переходов	Атака на таблицы переходов с параметризацией	Атака на таблицы переходов с передачей управления
Искажение указателей данных	Атака с искажением указателей данных	Атака с искажением указателей данных с параметризацией	Атака с искажением указателей данных с оригинальным кодом

ВОПРОС 2

Особенности реализации различных типов атак "переполнение буфера"

Ограничения на реализацию атак переполнения буфера

1) строковые переполняющиеся буфера не позволяют внедрять символы 0
2) размер переполняющихся буферов обычно очень малы
3) абсолютный адрес буфера обычно не известен, по этому приходится оперировать относительными адресами
4) базовые адреса системных функций изменяются от одной версии операционной системы к другой меняются (а в ОС Windows Vista базовый адрес системной библиотеки всегда разный)
5) адреса уязвимых мест программ зачастую непостоянны
6) требуются глубокие знания спецификаций процессоров, особенностей функционирования ОС, компиляторов различных производителей

Трудности реализации систем защиты от атак переполнения буфера

1) не существует надежных методик автоматического поиска переполняющихся буферов как на этапе разработки ПО, так и на этапе функционирования ПО
2) все разработанные методики противодействия атакам переполнения буфера существенно замедляют функционирование программ
3) межсетевые экраны отсекают лишь примитивные атаки переполнения буфера на этапе доставки кода атаки на целевую систему, методики поиска некорректных буферов, передаваемых в сетевых пакетах крайне примитивны

Схема построения атаки переполнения буфера

1) подготавливается мусор и вычисляются номера байтов, которые переписывают адрес возврата из функции

2) подготавливается исполнимый код в качестве буфера

3) вычисляется адрес возврата при помощи отладчика, так чтобы он указывал на исполнимый код в стеке

4) подготовленный буфер содержащий мусорные байты, новый адрес возврата и исполняемый код передаются уязвимой программе

Пример последовательно переполнения буфера при записи

```
seq_write(char *p)
{
   char buff[8];
   …
   strcpy(buff, p);
}
```

Пример индексного переполнения буфера при чтении

```
idx_write(int i)
{
   char buff[]="0123456789";
   …
   return buff[i];
}
```

В зависимости от своего местоположения буфера делаться на три независимые категории

- локальные буфера, расположенные в стеке и часто называемые автоматическими переменными
- статичные буфера, расположенные в секции (сегменте) данных
- динамические буфера, расположенные в куче

ПРИМЕР

```
void show_array(int arrlen, char array[])
{
    char buffer[32];
    int i;
    for (i = 0; i < arrlen; i++) buffer[i] = array[i];
    printf(buffer);
}

int main()
{
    char mystr[] = "To be, or not to be...";
    show_array(23, mystr);
    return 0;
}
```

Что же будет, если mystr длиннее 32-х байт?

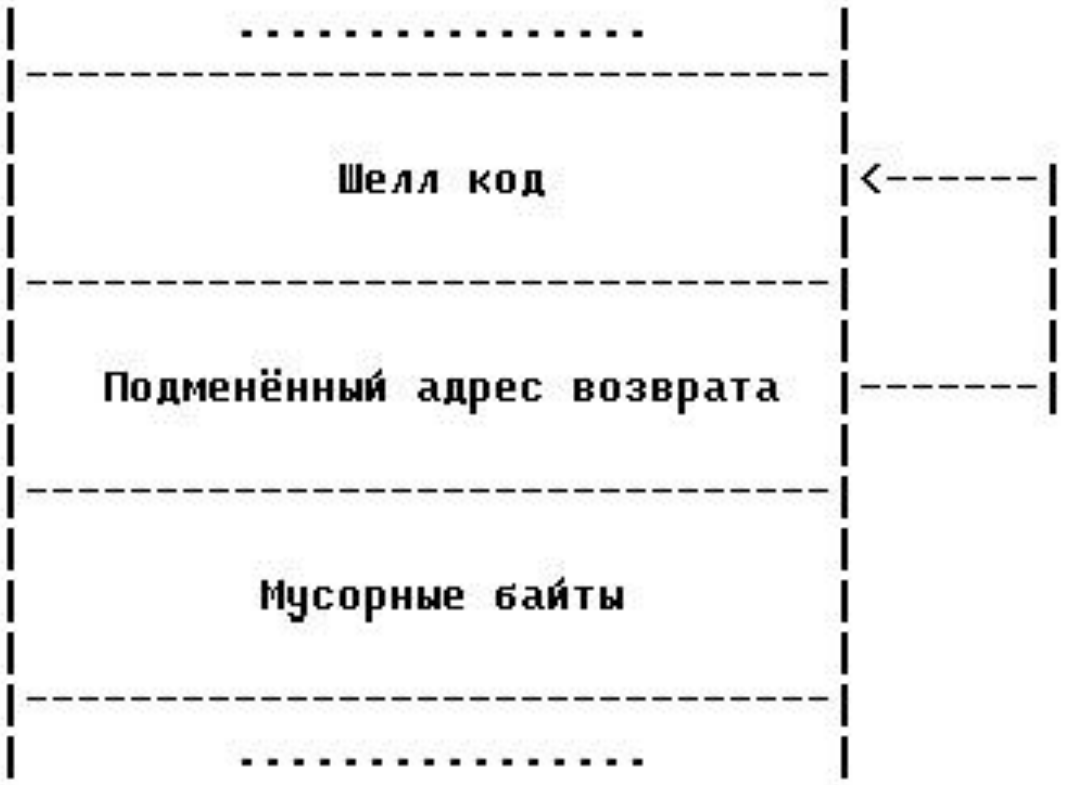
Задание на самоподготовку:

1) Углубить знания по подходам к реализации атак переполнения буфера.

Следующее занятие посвящено изучению вопросов защиты приложений от атак переполнения буфера.

Переполнение буфера через неисполнимый стек в Windows

Стек уязвимой программы в момент проведения обычной атаки на срыв стека выглядит следующим образом



ПРИМЕР

```
#include <windows.h>
void main ()
{
    WinExec("cmd",1);
}
```

```
.386
.model flat, stdcall
    extrn ExitProcess:proc
    extrn WinExec:proc
    .code
start:
    push 1
    push offset comm1
    call WinExec ; !!!!!
    push 0
    call ExitProcess
    comm1 db 'cmd',0
    end start
    end
```

Схема стека внутри WinExec

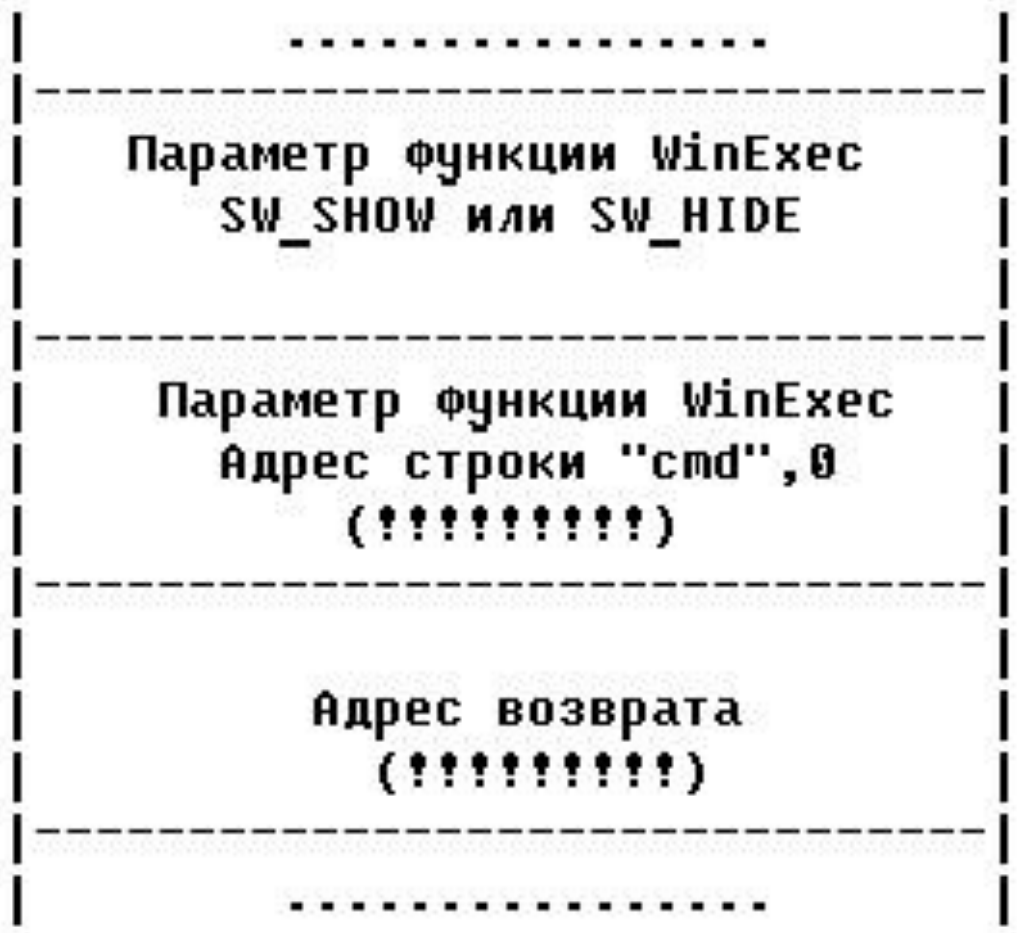
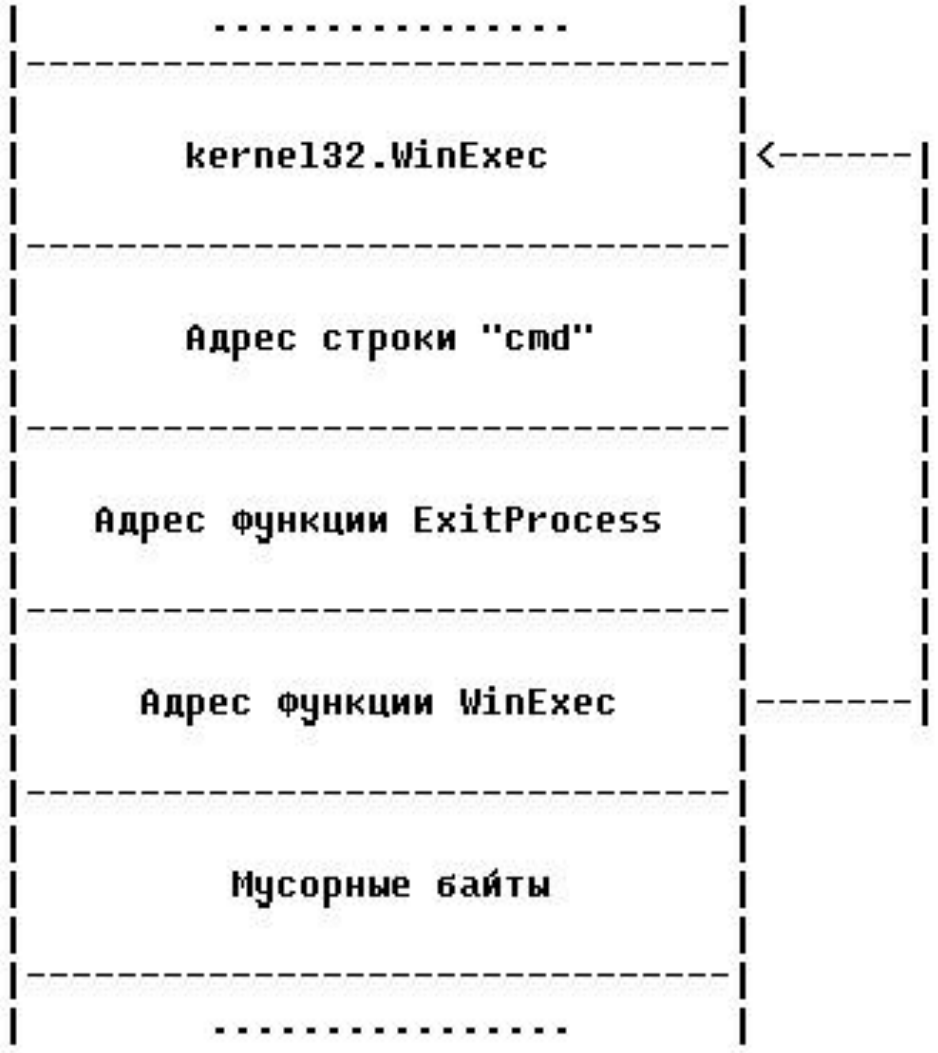


Схема построения атаки

1. Подготавливается имя программы ("cmd"), мусор и вычисляются номера байтов, которые перетирают адрес возврата
2. Вычисляется адрес строки при помощи отладчика, так чтобы он указывал на строку "cmd" в стеке;
3. Вычисляются адреса функций: WinExec, ExitProcess;
4. Подготовленный буфер содержащий имя программы ("cmd"), мусорные байты, новый адрес строки и адреса функций передаётся уязвимой программе.

Схема стека при реализации атаки



Как можно увидеть по схеме стека, в момент атаки был упущен первый параметр функции WinExec.

А так же был поставлен адрес функции ExitProcess в качестве адреса возврата из функции WinExec.

После того, как консоль будет запущена, управление будет передано на функцию ExitProcess, которая корректно завершит процесс.