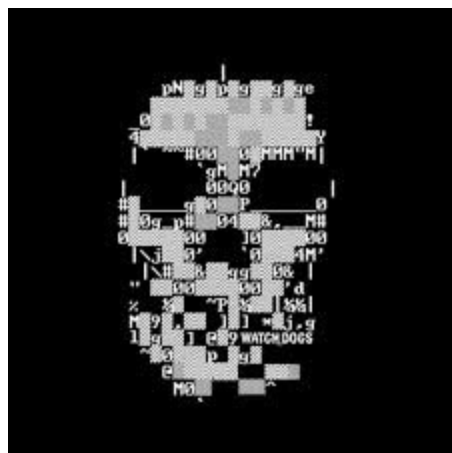


Уязвимость форматной строки. Основы.

https://telete.in/hacker_sanctuary • September 09, 2018



*В случае заимствования данной информации, указывайте авторство -
Telegram-канал "Убежище Хакера".*

Данный пост будет носить **теоретическо-практический** характер. В нём мы попытаемся разобраться в уязвимости форматной строки.

Перед прочтением стоит ознакомиться с постом об основах бинарных уязвимостей (вот [тут](#)). А также, с основами обратной разработки

https://telete.in/hacker_sanctuary/42

https://telete.in/hacker_sanctuary/43

https://telete.in/hacker_sanctuary/45

https://telete.in/hacker_sanctuary/46

https://telete.in/hacker_sanctuary/47

https://telete.in/hacker_sanctuary/80

https://telete.in/hacker_sanctuary/82

Предисловие.

Не так давно на канале был разбор задания на эксплуатацию уязвимости форматной

строки (вот тут пост - https://telete.in/hacker_sanctuary/105). В нём было обещано, что скоро выйдет пост об уязвимости форматной строки. Ничего сверх нового и умного здесь не будет, просто основы.

Начинаем.

Для начала нужно отметить, что данная уязвимость по большей части применима к приложениям написанным на чистом Си, в которых активно используются функции форматного вывода. Поговорим о форматном выводе.

Итак, у нас есть некоторая целочисленная переменная `var1`, в которую записано число, допустим, что это число 91.

На языке Си это выглядит примерно так.

```
1  #include <stdio.h>
2
3  int main()
4  {
5      int var1 = 91;
6      return 0;
7  }
```

Теперь представим, что мы хотим вывести это число на экран и будем использовать для этого функцию `printf()`. Одной из особенностей `printf`, является то, что она может выводить переменные в различном формате. Что это значит?

Например, мы выводим переменную `var1`, при этом указывая, что она должна быть выведена как 16-ричное число, или вообще как символ из ASCII таблицы. Такое указание может быть дано с помощью специальных сочетаний символов, которые указывают на формат. Основные сочетания представлены ниже:

`%x` - указание на 16-ричный вывод

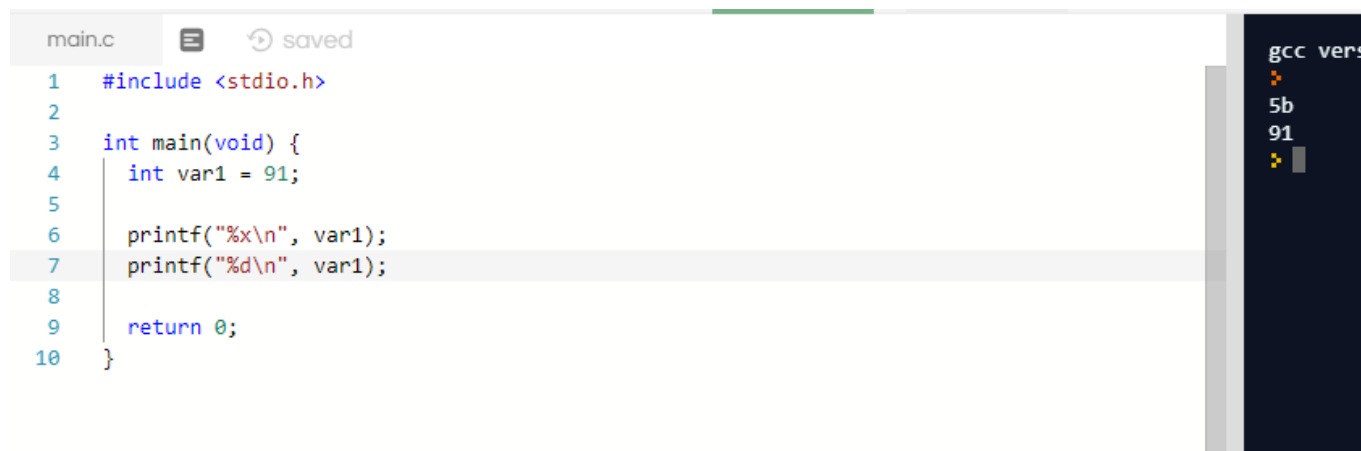
`%d` - указание на 10-тичный вывод

`%c` - указание на вывод одного символа

`%s` - указание на вывод строки

Подробнее про `printf()` и форматы вы можете почитать самостоятельно.

Покажем данную теорию на примере, чтобы стало понятно. Для этого используем онлайн компилятор, чтобы не возиться у себя на машине.



```
main.c saved
1  #include <stdio.h>
2
3  int main(void) {
4      int var1 = 91;
5
6      printf("%x\n", var1);
7      printf("%d\n", var1);
8
9      return 0;
10 }
```

gcc version 4.8.3
5b
91

Как можно заметить, мы указываем некоторую строку с форматом и символов перевода строки. В формат подставляется аргумент. Форматные строки довольно удобны, например, нам может понадобится вывести какую-либо информацию, где посреди строки будет некоторая переменная, рассчитанная заранее. Вот пример.



```
1  #include <stdio.h>
2
3  int main(void) {
4      int var1 = 91;
5
6      printf("Hello, var1 is %d, but you can change it\n", var1);
7
8      return 0;
9  }
```

gcc version 4.8.3
Hello, var1 is 91, but you can change it

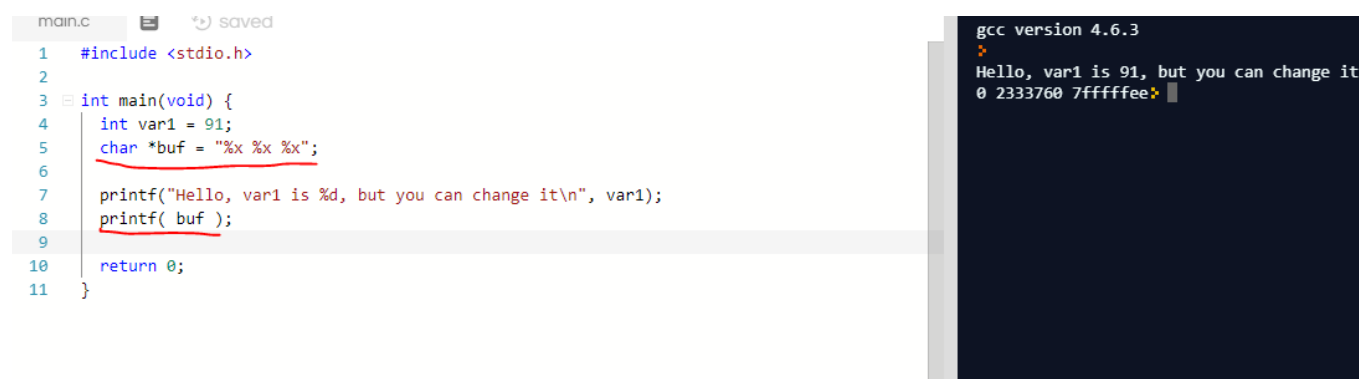
Надеюсь понятно, как это работает.

Суть уязвимости.

Теперь перейдём к уязвимости. Её суть заключается в том, что если вы используете функцию `printf()` для вывода некоторой переменной, без указания формата - это может привести к чтению памяти со стека и записи по произвольному адресу. Почему так?

Всё дело в том, что если вы не указываете формат, вы не сможете вывести обычное целое число, вы обязательно должны будете выводить строку, а в строке могут быть символы формата. Если они там есть и при этом никаких аргументов больше не

передаётся, то аргументы будут взяты со стека, что позволяет читать память со стека. Вот самый простой пример.



The image shows a code editor on the left and a terminal window on the right. The code editor displays a C program named `main.c` with the following content:

```
1 #include <stdio.h>
2
3 int main(void) {
4     int var1 = 91;
5     char *buf = "%x %x %x";
6
7     printf("Hello, var1 is %d, but you can change it\n", var1);
8     printf( buf );
9
10    return 0;
11 }
```

The terminal window shows the output of the program:

```
gcc version 4.6.3
Hello, var1 is 91, but you can change it
0 2333760 7ffffffe
```

Чтение памяти - не самое печальное в данной уязвимости, так как мы можем и писать в память, с помощью специального указания на формат. Про запись в память с помощью данной уязвимости мы поговорим в другом посте, но стоит сказать, что с помощью данной уязвимости можно обходить "стековую канарейку", которая защищает адрес возврата от перезаписи.

Какой профит?

Даже от чтения памяти можно получить некоторый профит, например, если мы можем контролировать строку, которая будет напечатана без указания формата, то мы можем осуществить атаку типа DoS, т.к. есть последовательности форматов, которые приводят к аварийному завершению работы.

Также чтение стека позволяет нам понять устройство адресного пространства и реализовывать другие атаки (например, `ret2libc`).

Данная уязвимость не особо популярна, однако раз на раз не приходится и умение её эксплуатировать может пригодиться.

Если у Вас будет интерес к данной уязвимости - напишите в наш ответчик - `@hackersanctuary_bot`, и в следующих постах данная уязвимость будет разобрана подробнее (с примерами кода и работой в отладчике), а также будут разобраны способы записи произвольной памяти с помощью данной уязвимости.

Создано с помощью [Tgraph.io](https://tgraph.io)