
CTF Code

Writeups

Forensics

24 октября 2021 г.

Оглавление

Easy	1
1 Baby keyboard	1
2 Just log in	4
Medium	5
1 Find file	5
2 Do you hear it?	6
Hard	7
1 Random xor	7
Real life	9
1 How about history?	9

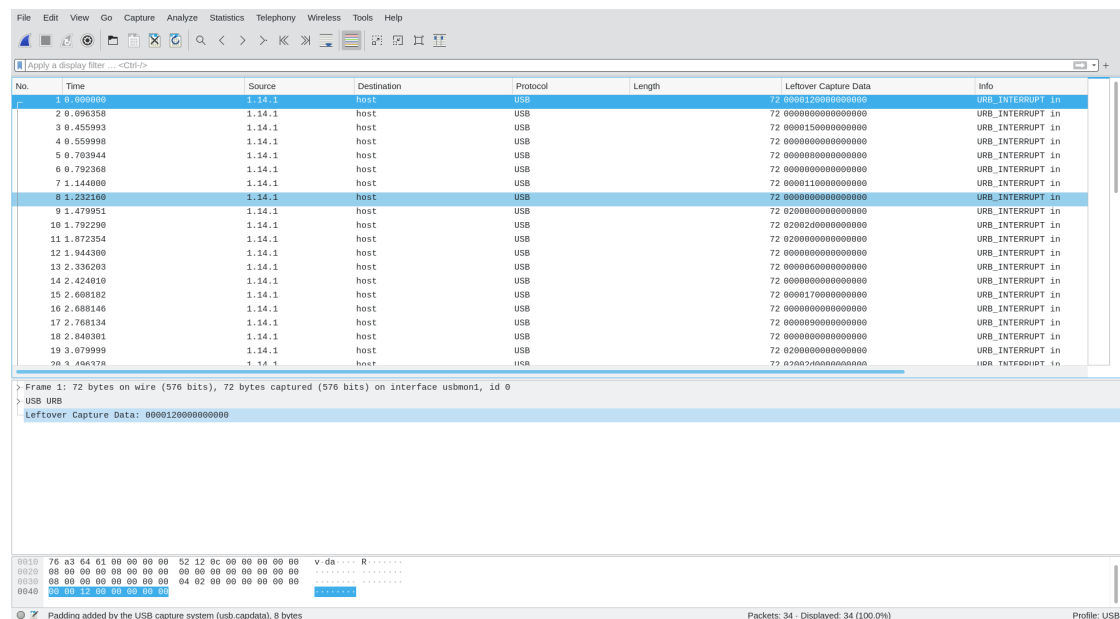
Easy

1 Baby keyboard

Теги: Hardware capture

У нас есть какой-то оооочень странный дамп.

Открываем дамп и видим, что это просто sniff данных с USB-порта. По условию таска это клавиатура, из чего и будем исходить. Если присмотреться можно увидеть, что все пакеты по 8 байт. Значит это точно клавиатура, либо что-то похожее и передающее данные на маленькой скорости (подробнее про это можно прочитать [вот здесь](#)). Теперь остается только достать полезную нагрузку и добавить ее как столбец:



Экспортируем данные в CSV и убираем заусенцы:

Easy

```
[anykeyshik@Irisu Downloads]$ cat dump.csv | cut -d "," -f 7 | cut -d "\"" -f 2
Leftover Capture Data
000012000000000000
000000000000000000
000015000000000000
000000000000000000
000008000000000000
000000000000000000
000011000000000000
000000000000000000
020000000000000000
02002d000000000000
020000000000000000
000000000000000000
000006000000000000
000000000000000000
000017000000000000
000000000000000000
000009000000000000
000000000000000000
020000000000000000
02002d000000000000
020000000000000000
02000a000000000000
020000000000000000
02000b000000000000
020000000000000000
020012000000000000
020000000000000000
020016000000000000
020000000000000000
020017000000000000
020000000000000000
02001e000000000000
020000000000000000
000000000000000000
```

После чего не составляет проблемы написать простенький питоновский скрипт для парса этой штуки и получения флага:

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
```

```
keyboardMap = {
    2: "PostFail",
    4: "a",
    5: "b",
    6: "c",
    7: "d",
    8: "e",
    9: "f",
    10: "g",
```

```
11: "h",
12: "i",
13: "j",
14: "k",
15: "l",
16: "m",
17: "n",
18: "o",
19: "p",
20: "q",
21: "r",
22: "s",
23: "t",
24: "u",
25: "v",
26: "w",
27: "x",
28: "y",
29: "z",
30: "1",
31: "2",
32: "3",
33: "4",
34: "5",
35: "6",
36: "7",
37: "8",
38: "9",
39: "0",
40: "Enter",
44: "Space",
45: "_",
47: "[",
48: "]",
57: "CapsLock"
}

with open('hexoutput', 'r') as keys:
    for line in keys:
        byteArray = bytearray.fromhex(line.strip())

        for byte in byteArray:
            if byte != 0:
                value = int(byte)
```

```
if value in keyboardMap:
    print(keyboardMap[value])
else:
    print("Unknown_key_{}".format(value))
```

2 Just log in

Теги: Виртуальная машина, сброс пароля, дампы памяти

Мы накрыли очень опасного хакера. Помоги достать данные из его системы.
Только, для начала, как бы попасть внутрь...

Нам дается запароленная виртуалка с Windows 7. Хакер, который ее использовал был явно не самым аккуратным человеком, это видно, стоит только зайти. Ну кто будет хранить флаги на рабочем столе? По сути, вся задача сводится к гуглингу "как сбросить пароль на Windows 7" или, более умный путь решения, знания из которого пригодятся в последней задаче ветки - вытащить из данного дампа памяти с помощью Volatility и плагина mimikatz.

```
$ volatility --plugins=%plugin_folder% -f dump.vmem --profile=Win7SP1x64 mimikatz
```

```
Volatility Foundation Volatility Framework 2.6.1
```

Module	User	Domain	Password
--------	------	--------	----------

wdigest	Kirino	OreImo	KirinoAyaseBestLove
---------	--------	--------	---------------------

Подробнее про Volatility можно почитать [вот здесь](#).

Флаг: oren_ctf_Shellshock!

Medium

1 Find file

Теги: FAT32, циклическая система, дамп

Перед тем, как мы его поймали, хакер успел разбить свою флешку. Но где-то на его компьютере лежит дамп ее файловой системы.

Нам дан FAT32 дамп какого-то диска. Если немного побродить по папкам становится понятно, что система закольцована. Но должно же быть решение! Если проанализировать дамп **fatcat**'ом (или чем-то похожим), то можно увидеть, что количество ссылок на 0 немного отличается. Стоп, но и флаги начинаются с 0. Провалившись вниз еще на несколько уровней становится понятно, что это флаг, зашитый в закольцованный дамп. Дальше можно или ручками пройти весь путь или написать простенький баш-скрипт, который найдет нужный путь:

```
#!/bin/sh
```

```
for i in {1..660}
do
    fatcat dump -L $i 2>/dev/null | grep '^f' >> files
done
```

Содержимое файла весьма лаконично:

f	1/1/1980	00:00:00	MATCH	c=74	s=0	(0B)
f	1/1/1980	00:00:00	MATCH	c=74	s=0	(0B)
f	1/1/1980	00:00:00	MATCH	c=74	s=0	(0B)
f	1/1/1980	00:00:00	MATCH	c=74	s=0	(0B)

Воспользуемся прямым путем до флага: **fatcat -k 74 dump** и получим **/SPACE/0/R/E/N/_/C/T/F/_/V/E/N/O/M/!/MATCH**. Немного побрутив регистр одного слова (все остальные фиксированы) получим наш флаг: **oren_ctf_VENOM!**.

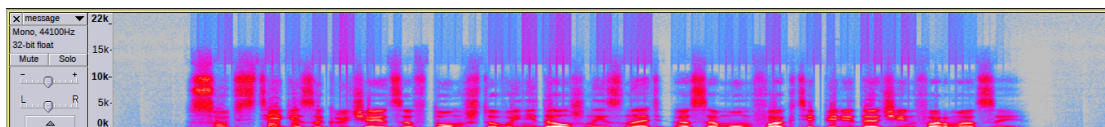
2 Do you hear it?

Теги: Фазовое кодирование, стеганография

Наш задержанный увлекался стеганографией. Неизвестно, успел ли он что-то спрятать, но вдруг?

В аудиозаписи автоматический голос сообщает: "где-то в этом сообщении спрятан флаг". Никакой другой полезной информации голос не сообщает.

Слушаем файл и слышим характерные "потрескивания" на фоне голоса. Эти же потрескивания видны и в спектрограмме (вертикальные линии):



Для тех, кто знаком со звуковой стеганографией, эти потрескивания сразу же напомним фазовое кодирование — один из самых простых и эффективных методов скрытия информации. Этот метод довольно популярен, и его часто описывают в исследованиях.

Чтобы решить задачу, нужно написать (или найти в интернете) код, который умеет разворачивать фазовое кодирование, и перебрать длину сегмента. По простому запросу **такой** код на Python находится на 4 строке выдачи. Кстати, как раз с помощью этого репозитория и был спрятан флаг!

Если взять тот код, его придётся немного подправить для конкретного WAV файла — он умеет работать только со звуком определённого формата.

Прогоняем обратно, получаем флаг: **oren_ctf_Heartbleed!**

Hard

1 Random xor

Теги: Break random

Кажется, он не так прост, как кажется. В архиве лежит какая-то информация, которую он хочет спрятать. Настолько, что даже непоправимо ее испортил. Получится ли у вас ее восстановить?

Нам дается архив с текстом программы и зашифрованным флагом. Казалось бы, на первый взгляд задача нерешаемая. Но если немного подумать, то становится понятно, что zip-архив дан чтобы восстановить время в которое был проведен хог. Тогда очень просто написать декриптор и получить флаг:

```
emph# emphinclude <iostream>
emph# emphinclude <fstream>
emph# emphinclude <ctime>
emph# emphinclude <cstdlib>

int main()
{
    std::fstream in("flag.enc", std::fstream::in);
    std::fstream out("out", std::fstream::out);

    std::string flag;
    std::string wtf;
    time_t start_time;

    in >> flag;
    std::cin >> start_time;

    while(start_time > 0)
    {
        srand(start_time);
        for(int i = 0; i < flag.size(); i++)
        {
            char ch = flag[i] ^ rand() % 255;
```

Hard

```
    if (!isprint(ch)) {  
        break;  
    }  
  
    wtf.push_back(ch);  
    }  
  
    wtf.push_back( '\n' );  
    out << wtf;  
    start_time -= 1;  
    }  
}
```

Грепаем выходной файл и получаем флаг `oren_ctf_goto_fail!`

Real life

1 How about history?

Теги: Анализ памяти

Кажется, в истории браузера тоже было что-то подозрительное. Сегодня на допросе мы не смогли получить ничего полезного, но, судя по тому, как он занервничал - там есть что-то опасное.

Задача - вытащить удаленную историю из хрома. На первый взгляд, все выглядит черезчур сложно, но, благо, нам дан дамп оперативной памяти. Просто залезаем плагином в историю хрома и, внезапно, получаем ссылку на pastebin.com, где лежит последний флаг серии.

```
volatility --plugins=%plug_folder% -f dump --profile=Win7SP1x64 chromehistory
```

Флаг: oren_ctf_Badlock!