POSITIVE TECHNOLOGIES

POSITIVE TECHNOLOGIES

Разбор заданий PHDays CTF/Quals

Дмитрий Скляров

ведущий аналитик отдела перспективных разработок

Positive Technologies

Вебинар Positive Technologies, 17 апреля 2014 г.

Что такое CTF?

CTF: Capture The Flag

- Командная игра, помогающая участникам приобрести опыт решения задач ИБ в форме соревнования
- Задания обычно основаны на реальных прототипах
- Два основных стиля проведения:
  - Classic (attack + defense)
  - Jeopardy (task-based)
- Подробнее можно почитать на https://ctftime.org/

Task-based CTF: категории заданий

- Forensic – компьютерно-криминалистическая экспертиза
- Reverse – анализ бинарного кода (reverse-engineering)
- Pwn – эксплуатация уязвимостей
- Admin – навыки администрирования
- Network – знание сетевой инфраструктуры и протоколов
- Crypto – криптография
- Segano – стеганография
- PPC — олимпиадное программирование (professional programming and coding)
- Web – поиск и использование Web-уязвимостей
- Misc – всякая всячина

Positive Hack Days CTF

- Проводится в рамках PHDays – международного форума, посвященного практическим вопросам информационной безопасности

- За три года успешно проведено три CTF (2011, 2012, 2013)

- Отборочный тур PHDays IV CTF прошел 25-27 января 2014

- PHDays IV CTF состоится 21-22 мая 2014

  http://www.phdays.ru/ctf/

  http://www.phdays.com/ctf/

Неочевидные задачи

mp3 me

«And do you know that sometimes music stores a hidden message?»

```
0000000000: 49 44 33 04 00 00 00 00 | 1F 76 54 52 43 4B 00 00  ID3♦     ▼vTRCK
0000000010: 00 09 00 00 01 FF FE 30 | 00 31 00 00 00 52 47 42   ○  ☺яю0 1    RGB
0000000020: 37 00 00 00 0D 00 00 03 | 35 2C 31 38 33 2C 20 4E  7   ♪  ♥5,183, N
0000000030: 55 4C 4C 00 52 47 42 36 | 00 00 00 0A 00 00 03 30  ULL RGB6   ◙  ♥0
0000000040: 2C 34 32 2C 31 35 39 00 | 52 47 42 35 00 00 00 0C  ,42,159 RGB5   ♀
0000000050: 00 00 03 31 39 34 2C 32 | 34 34 2C 36 38 00 52 47    ♥194,244,68 RG
0000000060: 42 34 00 00 00 09 00 00 | 03 34 37 2C 37 37 2C 36  B4   ○  ♥47,77,6
0000000070: 00 52 47 42 33 00 00 00 | 0B 00 00 03 34 34 2C 37   RGB3   ♂  ♥44,7
0000000080: 33 2C 31 34 31 00 52 47 | 42 32 00 00 00 0C 00 00  3,141 RGB2   ♀
0000000090: 03 31 34 30 2C 32 30 37 | 2C 37 32 00 52 47 42 31  ♥140,207,72 RGB1
00000000A0: 00 00 00 0D 00 00 03 31 | 32 30 2C 31 35 36 2C 32     ♪  ♥120,156,2
00000000B0: 30 33 00 00 00 00 00 00 | 00 00 00 00 00 00 00 00  03
00000000C0: 00 00 00 00 00 00 00 00 | 00 00 00 00 00 00 00 00
00000000D0: 00 00 00 00 00 00 00 00 | 00 00 00 00 00 00 00 00
```

```
RGB7   5,183, NULL RGB6   0,42,159 RGB5   194,244,68 RGB4   47,77,6 RGB3
44,73,141 RGB2   140,207,72 RGB1   120,156,203
```

mp3 me: Python в помощь!

RGB7	**5,183, NULL**	**RGB6**	**0,42,159**
RGB5	**194,244,68**	**RGB4**	**47,77,6**
RGB3	**44,73,141**	**RGB2**	**140,207,72**
RGB1	**120,156,203**		

```
>>> a = [120,156,203, 140,207,72, 44,73,141, 47,77,6,
194,244,68, 0,42,159, 5,183]
>>> print "".join(map(chr, a)).encode("hex")
789ccb8ccf482c498d2f4d06c2f444002a9f05b7

>>> import zlib
>>> print zlib.decompress("".join(map(chr, a)))
i_hate_ucucuga
```

Что такое ucucuga?

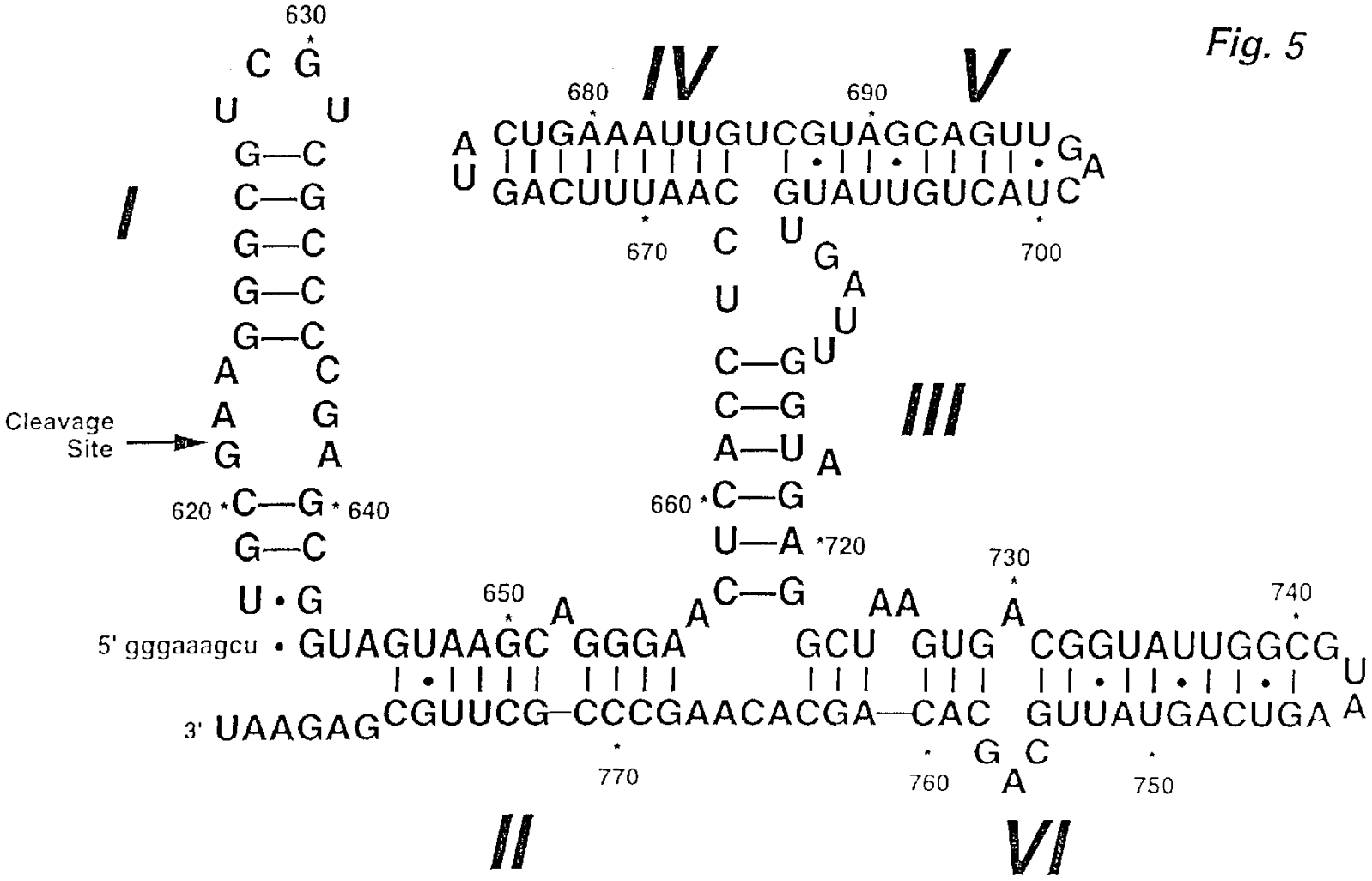


Fig. 5

Неправильная криптография

mars

«We've intercepted the following communication. What they are talking about?»

```
> REQUEST EXCHANGE
< CONFIRMED
> KEY 7abed4245dec0a5df17c672d799488c7b5ef5b9110c43e0c3eae586c21b7369095569fc3aa0cc2cde94
a362cc6e7902b8bb1bd6d32f358d8e7091e7e8c48b9fa2aabe1579328cfa18e55927c5e9c7d50ccba5b5f4e17
3b606b65a95c6f6c27b9fa8b482c45ce9605700856869b2beb7f663bf6244902ab57bf681a31fdbe86d7107da
a7cc53dcff6633cb7a591508da29e1016dc85211f83052efc8e987e4d5cf5bac407ee187795dbba722e8b3650
c555d119c55676a5ff90af6a0eabbba477/010001
< KEY 55b0a8cbba652468362b92444a4336da542eb8641ac83fc85e4e44bc77965842aa18f783d150a03a084
f04e6809ee60e5bfe8379397a665936d91b5f3433b48339d22b8b3de2544898376c87ddc385b8e6994eda6e38
5996a102a6cf5e241dbb0c78eae345c01a1bdad9d30d9bf84c02d976965654ec028d965833bd7072fbeb8c576
b49cc9f666757300fca0877f88a0975b3b5c67555f1de17aa153d2b9b54c6a51bcb35ca840e7f5cabdb1bd4ea
b542cbae8a2dd5a59f8e84785089bdd383/010001
> MSG 2cb5e614c40eb325cdcec0410884b5e4b20cb63c39f93b720710b2b4a2d66b76588129d24172faf0ef8
4ca02b3f89b718eaf2a7ba7a13e444a7772ed79b3db6c621c02a0d32ec09c510fcd39b1c76d715507de8634d2
17dceb33cb4d0cb0309b58bf89e4e27e6ab0a24c32f97d820013cefcd35408e12d11644928a368526d1482503
995c0d3921164fc0b301329ffb1aac13f66dcaa21a6f63e371308be12366de0e68db27751ec12f9e02976bc52
b2a2888f07608d26a637a46f03a6f1a1ab
< MSG 3c94c8dffb44880a13acc3e40826d021e4da41aca832cb6d396e83f528eb6fdd4aed8f59ed837d07223
0b0d6e38e57d90dd409de1c2e867a5b6a3962a83aa330b12516c313b0ba4820887fc7761c673d223b51f4f97e
d1a7a90a009a880f7317b3681a8ca64b53f2416d7daf43b8da2358d19d7e38d10c22bf7c55cf8fb587d33b9ee
87492149ad959c0154b4708c5961705f08a45423d4f5cabcbb3b2ec43266589ac3dbaf72a88377b9fe2afa84b
e2feacdb622f508d5d2874fa3106334bbd
```

mars ≡ Modified RSA

- 0x010001 == 65537 – публичная экспонента RSA (e)
- Сначала обмен открытыми ключами (n_1/e_1, n_2/e_2), потом обмен зашифрованными на них сообщениями (c_1, c_2)
- $c_1 = pow(m_1, e_1, n_1)$; $c_2 = pow(m_2, e_2, n_2)$
- требуется найти m_1 и m_2

- $m_i = pow(c_i, d_i, n_i)$
- $d_i * e_i \equiv 1 \bmod \phi(n_i)$
- n_i – произведение нескольких простых чисел
- n_1, n_2 – 1535 бит, не факторизуется ;(

mars: Евклид на Python-е

```
def egcd(a, b): # Extended Greatest Common Divisor
  if a == 0: return (b, 0, 1)
  else:
    g, y, x = egcd (b % a, a)
    return (g, x - (b // a) * y, y)

gcd = egcd(n1,n2)[0] # 1024 бита
p1 = n1 / cd # 512 бит
p2 = n2 / cd # 512 бит
```

- p_1 и p_2 простые числа
- gcd – составное число длиной 1024 бита (скорее всего 512*512), все равно не факторизуется ;(

mars: а если числа маленькие?

- пусть n = p*q*r
- тогда при 0 < m < p будет справедливо:

$$pow(m, e, n) \% p == pow(m, e, p)$$

- следовательно

$$e*d \equiv 1 \bmod \phi(p)$$
$$d = modinv(e, \phi(p))$$
$$\phi(p) == p-1,$$

- вычисление алгебраического дополнения

```
def modinv(a, m): # Modular inversion
  g, x, y = egcd (a, m)
  if g == 1: return x % m
  raise Exception("modular inverse does not exist")
```

mars: и, наконец, результат

```
def showX(v):
  print ("%0256X" % v).decode("hex").split('\0')[-1]

d1 = modinv(e, p1-1)
d2 = modinv(e, p2-1)
showX(pow(c1, d1, p1))
showX(pow(c2, d2, p2))
```

REQUEST: GET_FLAG (SIGNATURE: 5e2d5e0323591b1c).

RESPONSE: its_n0t_ab0ut_p4dd1ng

secc

«This key verification scheme is built on elliptic crypto, bet this points are impossible.

nc 195.133.87.171 5555

password: secch4l*»

- `source.tar.gz:`
  - `ecc.py`
  - `task.py`

secc: task.py/main

```
def main():
    print "Auth:“
    auth = raw_input()
    if hashlib.sha1(auth).hexdigest() !=
    "375d5c01ca1b8c3863024d10aac7713472eb5033":  # secch4l*
        print "nope“
        return

    prefix = os.urandom(8)
    print "Proof of work, please“
    print "Prefix is (hexed) ", prefix.encode("hex")
    test = raw_input().decode("hex")
    if not test.startswith(prefix) or len(test) > 16:
        print "nope“
        return
    h = hashlib.sha1(test).hexdigest()
    if not h.startswith("000000"):
        print "nope“
        return

    goflag()
```

secc: Proof of work, please

```
def readLn(sock):
  a = []
  while True:
    c = sock.recv(1)
    if '\n' == c: return "".join(a)
    a.append(c)

print readLn(sock)          # Auth:
sock.send("secch4l*\n")
print readLn(sock)          # Proof of work, please
s = readLn(sock)
print s                     # Prefix is (hexed)  0b3997e62b9ffbf4

prefix = s.split()[-1].decode("hex")
for i in xrange(0x7FFFFFFF):
  s = "%s%X" % (prefix, i)
  if hashlib.sha1(s).digest()[:3] == '\0\0\0': break
sock.send(s + '\n')
```

secc: task.py/goflag, ecc.py/derive

```
def goflag():
    print "EC PASSWORD CHECK"
    r = random.randint(31337, 1 << 250)
    R = p256.power(G, r)
    print "R =", R
    print "SHARED SECRET = R ^ PASSWORD"
    S = p256.power(R, PASSWORD)
    key = p256.derive(S)
    cipher = encrypt(FLAG, key)
    print "ENCRYPTED MESSAGE:", cipher.encode("hex")

def encrypt(msg, key):
    iv = os.urandom(8)
    stream = hashlib.sha256(iv + key).digest()
    stream = hashlib.sha256(stream + iv + key).digest()
    cipher = iv + xor(msg, stream)
    return cipher

def derive(self, p):
  return hashlib.sha256(str((p[0] << 10) / p[1])).digest()
```

secc: расшифрование

EC PASSWORD CHECK

R = 57211521812416894852507836254716617244582021770556870735566942430422483211 4

SHARED SECRET = R ^ PASSWORD

ENCRYPTED MESSAGE: 7a93846a011e0d0382e94f32d705239e6298169dcec20da5d6

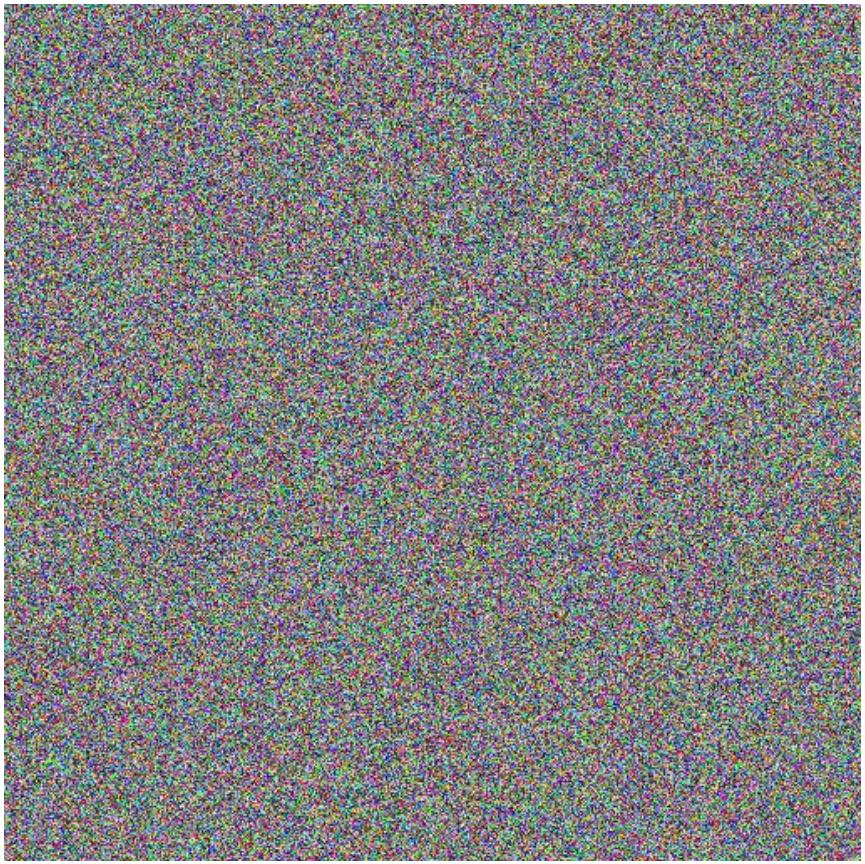
```
import hashlib, ecc

enc = "7a93846a011e0d0382e94f32d705239e6298169dcec20da5d6".decode("hex")
iv = enc[:8]

def decrypt(key):
  stream = hashlib.sha256(iv + key).digest()
  stream = hashlib.sha256(stream + iv + key).digest()
  return ecc.xor(enc[8:], stream)

for i in xrange(0x7FFFFFFF):
  s = decrypt(hashlib.sha256(str(i)).digest())
  for c in bytearray(s):
    if c < 32 or c >= 128: break
  else:
    print s                                  # ecc_is_too_s3cure
    break
```

Reverse Engineering

ShadeIt9000

«We've just searched Paul_Axe's Moscow apartment and found an encrypted image named 'derrorim_enc.bmp' on his desktop. Also there was some utility for image encryption - ShadeIt9000, but we couldn't find a decryptor. Please, help us to decrypt this image.»

derrorim_enc.bmp



ShadeIt9000.exe: заглянем внутрь

- Приложение использует OpenGL
- Строчка “inflate 1.2.8 Copyright 1995-2013 Mark Adler”

```
00416340 sub_416340      proc near               ; DATA XREF: .rdata:00464674↓o
00416340                 push    edi
00416341                 mov     edi, ecx
00416343                 push    1D00h           ; mode
00416348                 mov     dword ptr [edi+38h], 0
0041634F                 call    ds:glShadeModel
00416355                 push    0B71h           ; cap
0041635A                 call    ds:glEnable
00416360                 push    0DE1h           ; cap
00416365                 call    ds:glEnable
0041636B                 lea     eax, [edi+2044h]
00416371                 mov     ecx, edi
00416373                 push    eax
00416374                 push    offset byte_47F660
00416379                 push    offset byte_47F7B8
0041637E                 call    sub_415EB0
00416383                 pop     edi
00416384                 retn
00416384 sub_416340      endp
```

- По адресам 0x47F660 и 0x47F7B8 расположены массивы данных, упакованные ZLib

ShadeIt9000.exe: распакуем ZLib

```
from zlib import decompress as unZ
base = 0x47C000 - 0x7AE00 # data section base
ab=open("ShadeIt9000.exe", "rb").read()
open("1.txt", "w").write(unZ(ab[0x47F660-base:],-15))
open("2.txt", "w").write(unZ(ab[0x47F7B8-base:],-15))
```

- файл **`2.txt`**: вершинный шейдер

```
attribute vec3 a_param;

varying vec4 texCoord0;
varying vec3 v_param;

void main(void)
{
        gl_Position = gl_ModelViewProjectionMatrix * gl_Vertex;
        texCoord0 = gl_MultiTexCoord0;
        v_param = a_param;
}
```

ShadeIt9000.exe: извлечем шейдеры

- файл **`1.txt`**: пиксельный шейдер

```
#version 330
uniform sampler2D u_texture;
uniform sampler2D u_gamma;
varying vec4 texCoord0;
varying vec3 v_param;

uint func(vec3 co){
    return uint(fract(sin(dot(co ,vec3(17.1684, 94.3498, 124.9547))) * 68431.4621) * 255.);
}

uvec3 rol(uvec3 value, int shift) {
    return (value << shift) | (value >> (8 - shift));
}

const uvec3 m = uvec3(0xff);

void main()
{
            uvec3 t = uvec3(texture2D(u_texture, vec2(texCoord0)).rgb * 0xff) & m;
            uvec3 g = uvec3(texture2D(u_gamma, vec2(texCoord0)).rgb * 0xff) & m;
            int s = int(mod(func(v_param), 8));
            t = rol(t, s);
            vec3 c = vec3((t ^ g) & m) / 0xff;
            gl_FragColor = vec4(c, 1.);
}
```

ShadeIt9000.exe: главное о шейдере

```
void main()
{
  uvec3 t = uvec3(texture2D(u_texture,
      vec2(texCoord0)).rgb * 0xff) & m;
  uvec3 g = uvec3(texture2D(u_gamma,
      vec2(texCoord0)).rgb * 0xff) & m;
  int s = int(mod(func(v_param), 8));
  t = rol(t, s);
  vec3 c = vec3((t ^ g) & m) / 0xff;
  gl_FragColor = vec4(c, 1.);
}
```

ShadeIt9000.exe: генератор гаммы

```
unsigned char *pbGamma = malloc(cbGamma);
srand(time(0));
for (i = 0; i < cbGamma; i++) {
  pbGamma[i] = rand();
}
```

- **derrorim_enc.bmp** создан 21.01.2014 18:37:52

```
>>> import time
>>> print hex(int(time.mktime((2014,1,21,
     18,37,52, 0,0,0))))
```

0x52de8640

- Копируем и исправляем файл **ShadeIt9000_f.exe**:

```
00015557: E8A5310100 => B84086DE52
```

call _time => mov eax,52de8640h

ShadeIt9000: готовим константы

```
import os
bmp=open("derrorim_enc.bmp", "rb").read()
hdr = bmp[:0x36]
abData = bytearray(bmp[0x36:])
cbBody = len(bmp) - len(hdr)
open("00.bmp", "wb").write(hdr + '\0'*cbBody)
open("XX.bmp", "wb").write(hdr + '\2'*cbBody)
os.system("ShadeIt9000_f.exe 00.bmp")
os.system("ShadeIt9000_f.exe XX.bmp")
```

- в файлах **`00_enc.bmp`** и `XX_enc.bmp` гамма и сдвиги по-XOR-енные с гаммой

ShadeIt9000: расшифровываем

```
def rol(v,i): return (((v<<i) & 0xFF) | ((v>>(8-i)) & 0xFF))
def ror(v,i): return (((v>>i) & 0xFF) | ((v<<(8-i)) & 0xFF))
dRot = {rol(1,i):i for i in xrange(8)}

bmp=open("derrorim_enc.bmp", "rb").read()
hdr = bmp[:0x36]
abData = bytearray(bmp[0x36:])

abGamma = bytearray(open("00_enc.bmp", "rb").read()[0x36:])
abRot = bytearray(open("XX_enc.bmp", "rb").read()[0x36:])
for i,b in enumerate(abGamma): abRot[i] = dRot[abRot[i] ^ b]
for i,b in enumerate(abGamma): abData[i] =
      ror(abData[i] ^ b, abRot[i])

open("derrorim.bmp", "wb").write(hdr + str(abData))
```

ShadeIt9000: `derrorim.bmp`

ShadeIt9000: короткий путь

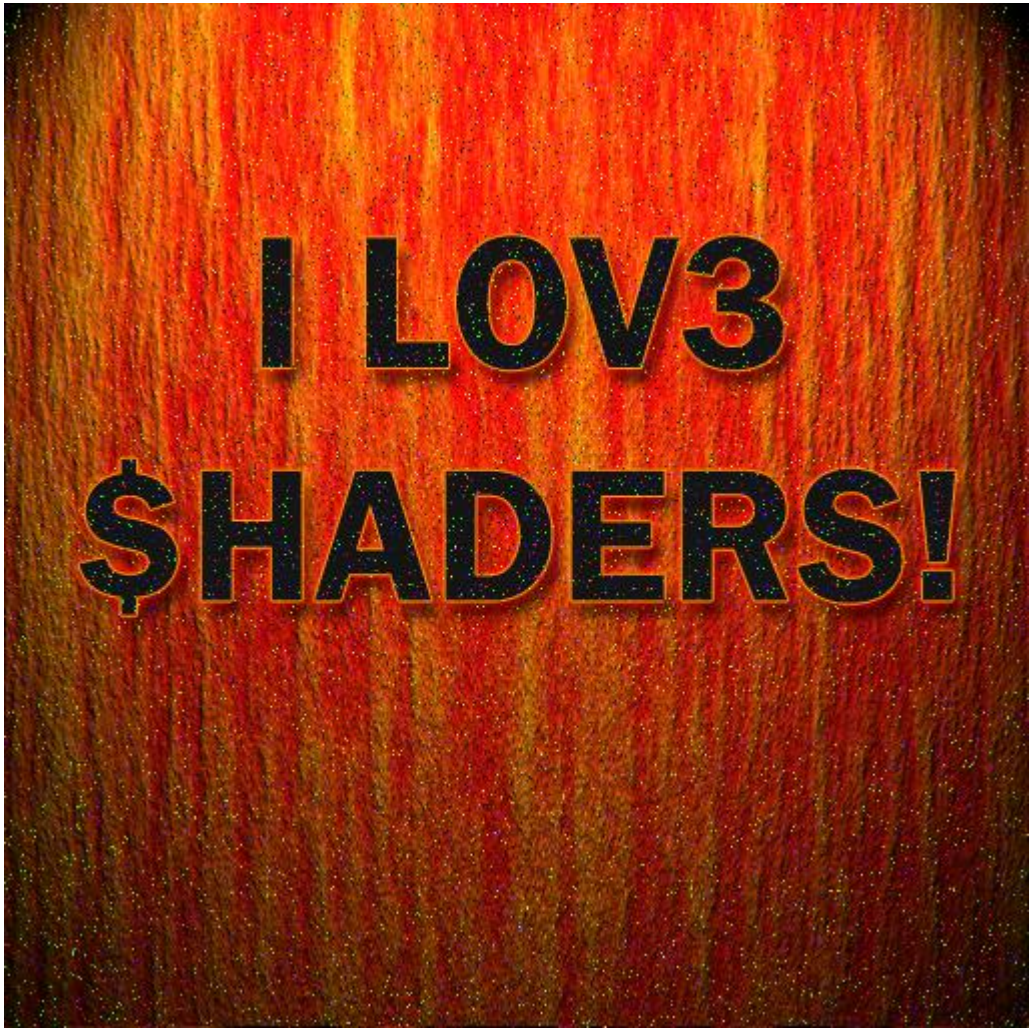
- Сразу за вершинным шейдером по адресам 0x47F848 и 0x47F9A0 лежит упакованный ZLib код пиксельного и вершинного шейдера для выполнения обратного преобразования (расшифрования)
- Код вершинного шейдера для зашифрования и расшифрования идентичен
- А что если подменить пиксельный шейдер?

- Копируем **ShadeIt9000_f.exe** в **ShadeIt9000_d.exe** и исправляем его:

```
00015775: 60F6 => 48F8
```

- Запускаем **ShadeIt9000_d.exe derrorim_enc.bmp**

ShadeIt9000: `derrorim_enc_enc.bmp`



Архив заданий PHDays IV CTF Quals

http://ctfarchive.phdays.com/phd4quals/

Конец рассказа
Спасибо за внимание

Дмитрий Скляров

ведущий аналитик отдела перспективных разработок

Positive Technologies

DSklyarov@ptsecurity.ru

POSITIVE TECHNOLOGIES