v0id s3curity

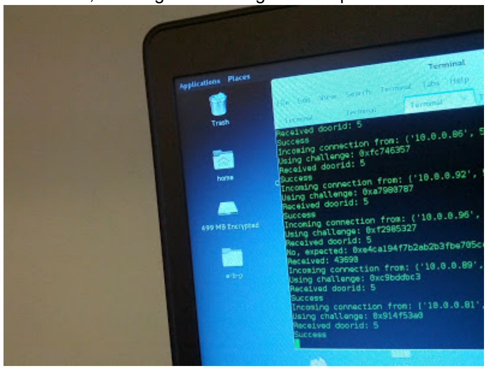
Yet another blog by a security enthusiast !

WEDNESDAY, MARCH 5, 2014

Boston Key Party CTF 2014 - Door 1 - Crypto 500 - [Team SegFault]

Description:
You need to open door no 5 in a secret facility. We have been trying to brute-force the door without success. One of our agents was able to infiltrate the control room and take a picture. The server is known to use some kind of problematic random number generator for the authentication. Open the door. The service is accessible on 54.186.6.201:8901, good luck with your mission. UPDATE/HINT: LFSR.

We were given an image file with some info, showing the challenge and response communication.



```
Received doorid: 5
Success
Incoming connection from: ('10.0.0.86', 5
Using challenge: 0xfc746357
Received doorid: 5
Success
Incoming connection from: ('10.0.0.92',
Using challenge: 0xa7908787
Received doorid: 5
Success
Incoming connection from: ('10.0.0.96',
Using challenge: 0xf2985327
Received doorid: 5
No, expected: 0xe4ca194f7b2ab2b3fbe705c
Received: 43698
Incoming connection from: ('10.0.0.89',
Using challenge: 0xc9bddbc3
Received doorid: 5
Success
Incoming connection from: ('10.0.0.81',
Using challenge: 0x914f53a3
Received doorid: 5
Success
```

To start with, it was my friend who asked me to look into this problem and said, the challenge received might be seed for LFSR and output sequence might be the expected reply. Lets see.

[*] The challenge received is a 4 byte value. If it has to be seed for LFSR, then the LFSR should have 32 registers to deal with 4 bytes
[*] The screen capture shows a big number which might be a possible output sequence

Below is the number:

```
0xe4ca194f7b2ab2b3fbe705c
```

This is how the bitstream looked like:

```
111001001100101000011001010011110111101100101010101100101011001111111101111110011100000101
1100
```

With 92 bits of output sequence in hand and a possible 32 register LFSR, we can find the LFSR feedback function by feeding the Berlekamp-Massey algorithm with 64 bits(2 * length of LFSR) of the output sequnce.

The feedback function found is:

```
1 + x + x^4 + x^5 + x^6 + x^16 + x^19 + x^31 + x^32
```

The degree of the polynomial is 32, which supports the possible length of LFSR being 32.

Now, if the sequence 0xe4ca194f7b2ab2b3fbe705c can be generated using the above function and seed 0xf2985327, then we are on right track.

```
#!/usr/bin/env python

chall = 0xf2985327
seed = [int(i) for i in bin(chall)[2:]]
# x + x^4 + x^5 + x^6 + x^16 + x^19 + x^31 + x^32 + 1
feedback_function = [1, 0, 0, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 1, 1]

output = ''
for i in range(96):
    feedback = 0
    output += str(seed[-1])
    for i in range(len(seed)):
        feedback = (seed[i] * feedback_function[i] + feedback) % 2
    seed = [feedback] + seed[:-1]
```

ABOUT ME

Reno Robert

View my complete profile

BLOG ARCHIVE

- ► 2016 [4]
- ► 2015 [23]
- ▼ 2014 [21]
  - ► December [2]
  - ► October [1]
  - ► September [3]
  - ► August [3]
  - ► July [2]
  - ► May [2]
  - ▼ March [2]
    - RuCTF Quals 2014 - Aggregator - Vuln 200 - [Team S...
    - Boston Key Party CTF 2014 - Door 1 - Crypto 500 - ...
  - ► February [2]
  - ► January [4]
- ► 2013 [41]
- ► 2012 [20]

LABELS

Exploitation
Crypto
2014
Reverse
2015
Exploit Exercise
ROP
Hackim
CSAW15
Hack You CTF
Cscamp CTF Quals
2016
Boston Key Party CTF
Codegate CTF
Hack You Too CTF
Plaid CTF
RSA
Bug
EB CTF
Volga CTF
32C3 CTF
DIMVA CTF
Defcon
InCTF
PoliCTF
Pwnium CTF
SIGINT CTF
UAF
29C3 CTF
AES
Backdoor CTF
Balt CTF
CSAW CTF
ForbiddenBITS CTF
GITS CTF
Information Leak

```
print hex(int(output,2))
```

```
[ctf@renorobert BKP]$ python cry500.py
0xe4ca194f7b2ab2b3fbe705ccL
```

The generated output sequence is equal to the one in the screen capture, so we got the LFSR design right!
Now the amount of bits to be sent as response should be bruteforced. I incremented byte by byte from 96 bits, but soon I was told in IRC that 296 bits is all thats needed. Sending a 296 bits reply gave us the flag.

```
0x8d3c22ea
0x57443cb1e7daf6a45be190d11d3a1b4902633133d62d5900fd134c3caa91d5c233342fce62
response
> Door open: n0t_s0_r4nd0m_029210
```

Flag for the challenge is **n0t_s0_r4nd0m_029210**

Posted by Reno Robert at 3:01 AM

Labels: 2014 , Crypto

No comments :

Post a Comment

Enter your comment...

Comment as: Select profile...

Publish Preview

Newer Post Home Older Post

RuCTF Quals

UFO CTF

Web

POPULAR POSTS

Defeating anti-debugging techniques using IDA and x86 emulator plugin

Recently I was trying to reverse a small crackme by Tosh. The challenge was to find a password by reversing the binary, bypassing t...

Exploit Exercise - Python Pickles

Level [17] in nebula is pretty straight forward. The first look of it reveals the use of python's potentially vulnerable funct...

Exploiting PHP Bug #66550 - SQLite prepared statement use-after-free - [A local PHP exploit]

As the title says, this bug is useful only for post exploitation to bypass protections when the attacker already has arbitrary PHP c...

Exploit Exercise - PHP preg_replace

This level has a setuid binary which acts as a wrapper to execute a php script. The php script uses preg_replace with "e" ...

Revisiting Defcon CTF Shitsco Use-After-Free Vulnerability - Remote Code Execution

Defcon Quals 2014 Shitsco was an interesting challenge. There were two vulnerability in the binary - strcmp information leak and an...

Return to VDSO using ELF Auxiliary Vectors

This post is about exploiting a minimal binary without SigReturn Oriented Programming (SROP). Below is demo code, thanks to my friend...

Exploit Exercise - Orphan Process and Race Condition

Lets see how to solve level [19] in Nebula. The code given checks whether its parent process is root using /proc entry. int main(...

Codegate 2013 Quals - Vuln 100 [Failed Attempt] & Updated Solution

As the title says this is not a working solution for vuln 100 but just a description of my approach to exploit the given x86_64 binar...

Data Structure Recovery using PIN and PyGraphviz

This is a simple POC PIN tool to recover data structures in dynamically linked stripped executables, mainly for analyzing small prog...

Some universal gadget sequence for Linux x86_64 ROP payload

Unlike x86, x86_64 ROP payload needs the parameters to function call in registers. __libc_csu_init functions provides a few nice gad...

Powered by Blogger.