More Smoked Leet Chicken

We pwn CTFs

Oct
25

Hack.lu 2013 CTF — Crypto 350 (BREW'r'Y)

- Writeups

by hellman

> BREW'r'Y (Category: Crypto) Author(s): dwuid
> Finally, the robots managed to sneak into one of our breweries. I guess I won't have to explain how bad that really is. That darn non-physical ones even shutdown our login system. Shiny thing, advanced technology, all based on fingerprints. Been secure as hell. If only it was running. Well, basically, we're screwed.
> But wait, there's hope. Seems like they didn't shutdown our old login system. Backward compatibility's a bitch, eh? Unfortunately, we got like _zero_ knowledge about the protocol. I mean come on, the last time we used that thingy was like decades ago. If we are lucky, the old authentication method is buggy.
> So, I heard you're kinda smart? Have a look at it. We desperately need to get drunk^W supply. You'll find the old system at ctf.fluxfingers.net:1335. Good luck.
>
> Hint: Data is — and is expected to be — compressed using zlib.
> Hint: The challenge text gives hints about the protocol involved.

Summary: Hamilton path bugged check

Let's connect and see what's there:

```
$ nc -v ctf.fluxfingers.net 1335
ctf.fluxfingers.net [149.13.33.74] 1335 (?) open
```

```
BREW'r'Y graphICAL LOGIN PANEL - (Stage 1)
[ "Beer you, beer me, beer us together." ]
_____________________________________________________________________________

Session ticket #267:
eJw1l4dxxDAMBFtRCaKY+2/Mt4v3jH1HUIyI/Pa07z7taf0D32Z7gu0Ev07zS+O5N9DeHjy2xIY8+d7b05+

Commit round authorization:
```

We see another hint — "graph". So let's decode base64 and zlib:

```
In [3]: zlib.decompress( s.decode("base64") )
Out[3]: '1 129 1 132 1 101 1 135 1 118 1 23 1 121 2 99 2 103
2 8 2 108 2 113 2 51 2 31 3 87 3 5 3 119 3 146 3 23 3 26 3 29
4 128 4 101 4 140 4 45 4 18 4 116 4 58 5 137 5 10 5 110 5 83
5 85 5 57 6 68 6 137 6 42 6 77 6 36 6 93 6 126 7 115 7 36 7
...
```

We see a bunch of numbers. Remember it should be a graph: we can pair the numbers so it will be a list of edges. Also we see that edges connect smaller numbers to bigger — so most probably it's an undirected graph.

Remember the hint about zero knowledge protocol (google and find this). Seems we need to prove we know a Hamilton path (Hamilton cycle here actually) for a given graph. (Hamilton path is a path that visits all the points in graph bu each point only once) We need to send some graph to the server and the server will ask either a permutation (which must map original graph to the one we sent) or a Hamilton path on that graph.

We can send some simple graph in which we know a Hamilton path and if the server asks for it we will know it. Or we can make a random permutation of the original graph. The chance is 50% in both cases.

Another problem is to guess format of the answers. Permutation is sent in form `[old_point1, new_point1, old_point2, new_point2, ...]`. Hamilton path is send as just list of points. Both lists are encoded in the same manner (numbers separated by spaces, compressed and base64'd).

Actually, there are 3 stages in the challenge:

- Stage 1: 1 request with 150 points graph. (We can easily win by chance)
- Stage 2: 20 requests with 50 points graph. (Random path in such graph is rather to be a Hamiltonian, so we indeed can prove it)
- Stage 3: 20 requests with 150 points graph. (Oh.)

3rd stage introduces masked authorization. Guess that mask is simple xor. We need to mask our graph with random numbers xor, and after sending a permutation or a path we must show our mask. Well, I think we can use that and generate different masks for different request types.

So I guess it was intended to move through the stages one-by-one and apply different methods for each stage.

But aactually there's a bug in check so we can solve all the 3 stages using that bug and avoid implementing different techniques for each stage.

The bug is rather easy to find — if you send an extended graph (with additional edges) — it is accepted as a valid permutation of the original one. So we can easily send both a permutation and a Hamiltonian path — just add any edges you want.

Here's the code:

```
import random, zlib
from sock import *

def decode(s):
    s = zlib.decompress( s.decode("base64") )
    return map(int, s.split())

def encode(lst):
    s = zlib.compress(" ".join(map(str, lst)), 9).encode("base64")
    # fuck python! I've wasted lots of time on this
    s = s.replace("\n", "")
    return s


def encode_perm(perm):
    lst = []
    for i, v in enumerate(perm):
        lst.append(i + 1)
        lst.append(v)
    return encode(lst)

def do_graph(graph):
    LEN = max(graph)

    perm = range(1, LEN + 1)
    random.shuffle(perm)

    graph = [perm[i-1] for i in graph]
    edges = zip(graph[::2], graph[1::2])

    ham_path = range(1, LEN + 1)
    random.shuffle(ham_path)

    for edge in zip(ham_path, ham_path[1:] + ham_path[:1]):
        if edge not in edges and edge[::-1] not in edges:
            edges.append(edge)

    newgraph = [p for edge in edges for p in edge]
    return newgraph, perm, ham_path

def read_stage_head():
    f.read_until("ticket #")
    f.read_until("\n")
    s = f.read_until("\n")
    return decode(s)

def do_step(graph, stage):
    f.read_until("authorization:\n")

    newgraph, perm, ham_path = do_graph(graph)
    if stage == 3:
        newgraph = [0x41 ^ c for c in newgraph]
    f.send(encode(newgraph) + "\n")
```

```
    req = f.read_until(":\n")
    print "STAGE %d request:" % stage, req.strip()
    if "type A" in req:
        f.send(encode_perm(perm) + "\n")
    else:
        f.send(encode(ham_path) + "\n")

    if stage == 3:
        f.read_until("mask:\n")
        f.send(encode([0x41 for c in newgraph]) + "\n")


f = Sock("149.13.33.74", 1335)

graph = read_stage_head()
do_step(graph, stage=1)

graph = read_stage_head()
for i in range(20):
    do_step(graph, stage=2)

graph = read_stage_head()
for i in range(20):
    do_step(graph, stage=3)


print f.read_all()
```

```
$ py wup.py
STAGE 1 request: Requesting round authorization ticket, type A:
STAGE 2 request: Requesting round authorization ticket, type B:
...(20 requests)
STAGE 2 request: Requesting round authorization ticket, type A:
STAGE 3 request: Requesting round authorization ticket, type A:
...(20 requests)
STAGE 3 request: Requesting round authorization ticket, type A:

Welcome to our main interface. Your BREW'r'Y identification is: iN.D3sPER4T3.n3Ed.0

(Closing connection.)
```

The flag: **iN.D3sPER4T3.n3Ed.0F.PHRiTZk3Wli**

Tags: 2013, ctf, graph, hack.lu, Hamilton path, mask, permutation, python, xor, zero knowledge proof

1 comment

1. 

   give me your contact pls says:

   October 31, 2013 at 21:59 (UTC 3)

   Reply

   give me your contact pls

Leave a Reply

Your email address will not be published.

Message:

You may use these HTML tags and attributes: `<a href="" title=""> <abbr title=""> <acronym title=""> <b> <blockquote cite=""> <cite> <code> <del datetime=""> <em> <i> <q cite=""> <s> <strike> <strong>`

Name:

Email:

Website:

Submit Comment

Submit Comment

Archives

- March 2017 (3)
- October 2016 (6)
- September 2016 (3)
- May 2016 (5)
- April 2016 (2)
- March 2016 (5)
- September 2015 (2)
- May 2015 (4)
- April 2014 (4)
- March 2014 (1)
- February 2014 (4)
- January 2014 (2)
- December 2013 (1)
- October 2013 (1)
- June 2013 (6)
- April 2013 (1)
- February 2013 (2)
- November 2012 (6)
- October 2012 (7)
- May 2012 (6)
- April 2012 (2)

- March 2012 (4)
- February 2012 (17)
- January 2012 (12)
- December 2011 (6)
- October 2011 (5)
- September 2011 (10)
- August 2011 (8)
- July 2011 (3)
- June 2011 (5)
- April 2011 (10)
- March 2011 (7)
- January 2011 (2)
- December 2010 (1)
- November 2010 (1)
- October 2010 (11)
- September 2010 (11)

Meta

- Register
- Log in
- Entries RSS
- Comments RSS
- WordPress.org

Tags

2010 2011 2012 2013 2014 2016 aes aslr binary bruteforce c++ codegate crt crypto ctf defcon exploit exploitation formatstring gits hack.lu hacklu hash ictf leetmore libnum nuit du hack nx pctf plaid plaidctf ppp python quals reverse reversing rop rsa sage shellcode vm web writeup x64 xor

Valid XHTML 1.0 Strict Valid CSS Level 2.1

More Smoked Leet Chicken uses Graphene theme by Syahir Hakim.