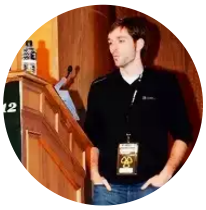


</pwntester>

Search

Subscribe via RSS

GitHub LinkedIn Twitter Contact

17 Jan 2014 on CTF | HackYou2014 | Crypto

#hackyou2014 Crypto400 write-up

In this level we are said that:

> *We have intercepted communication in a private network. It is used a strange protocol based on RSA cryptosystem.*
>
> *Can you still prove that it is not secure enough and get the flag?*

We are given a pcap file with a bunch of transmissions generated with this script:

```
#!/usr/bin/python
import sys
import struct
import zlib
import socket
```

```
class Client:
    def __init__(self, ip):
        #init
        self.ip = ip
        self.port = 0x1337
        #connect
        self.conn = socket.socket(socket.AF_INET, so
        self.conn.connect((self.ip, self.port))
        #recieve e
        self.e = self.Recv()
        #recieve n
        self.n = self.Recv()
        self.e, self.n = int(self.e), int(self.n)

    def Recv(self):
        #unpack data
        length = struct.unpack('!H', self.conn.recv(
        data = zlib.decompress(self.conn.recv(length
        return data

    def Pack(self, data):
        #compress data
        data = zlib.compress('%s' % data)
        length = struct.pack('!H', len(data))
        return '%s%s' % (length, data)

    def Send(self, msg):
        #send message
        msg = int(msg.encode('hex'),16)
        assert(msg < self.n)
        msg = pow(msg, self.e, self.n)
        self.conn.send(self.Pack(msg))
        print '[+] Message send'

    def __del__(self):
        #close connection
        self.conn.close()

if len(sys.argv) != 2:
    print 'Usage: %s <ip>' % sys.argv[0]
    sys.exit(0)

flag = open('message.txt').readline()
test = Client(sys.argv[1])
test.Send(flag)
```

Analyzing the protocol it looks like it goes something like:

- Message to be send is read from external file
- Connection is established against given IP on port 4919 (0x1337)
- Server sends "e" packet [data lengt + zlib(data)]
- Server sends "n" packet [data lengt + zlib(data)]
- Both "e" and "n" are casted to integers

- Client encodes message as integer
- Client verifies message < n
- Client encrpyts message using: enc = pow(message, e, n)
- Client sends encrypted message packet [data lengt + zlib(data)]

If we explore the pcap with wireshark we can see a bunch of transmissions from Client to Server. We will need to process it in order to extract the message, the following python+scapy script will read all the interesting elements being transmited for each communication: e, n and flag:

```
#!/usr/bin/python
from scapy.all import *
from struct import *
import zlib

pkts = PcapReader("packets.pcap")
for p in pkts:
    pkt = p.payload
    if pkt.getlayer(Raw):
        raw = pkt.getlayer(Raw).load
        print "{0}:{1} -> {2}:{3}".format(pkt.src,pk
        if str(pkt.sport) == "4919":
            elength = struct.unpack("!H",raw[0:2])[0
            ezip = raw[2:2 + elength]
            e = int(zlib.decompress(ezip))
            nlength = struct.unpack("!H",raw[elength
            nzip =  raw[elength + 4:elength + 4 + nl
            n = int(zlib.decompress(nzip))
            print "e = {0}".format(e)
            print "n = {0}".format(n)
        if str(pkt.dport) == "4919":
            flaglength = struct.unpack("!H",raw[0:2]
            flagzip = raw[2:2 + flaglength]
            encflag = int(zlib.decompress(flagzip))
            print "encflag = {0}".format(encflag)
```

As an example of one communication:

```
alvaro@winterfell ~/D/h/crypto400> python decrypter.
WARNING: No route found for IPv6 destination :: (no
192.168.1.10:4919 -> 192.168.1.5:41260
e = 17
n = 2765806067803871578047042957059798714454221387 51
192.168.1.5:41260 -> 192.168.1.10:4919
encflag = 114334886129919907685360866989651801465503
```

Analyzing the traffic, there are 19 communications with different modulos but always same exponent (e=17) which simplifies the problem

```
encrypted = (flag^17) % modulo
```

We should only care to find F so that:

```
encflag1 = F % n1
encflag2 = F % n2
...
...
encflag18 = F % n18
encflag19 = F % n19
```

In order to solve the equation we can use the Chinese Remainder Theorem (CRT). For which I found a Python implementation that I needed to adjust a little bit:

```
from operator import mod

def eea(a,b):
    """Extended Euclidean Algorithm for GCD"""
    v1 = [a,1,0]
    v2 = [b,0,1]
    while v2[0]<>0:
       p = v1[0]//v2[0] # floor division
       v2, v1 = map(lambda x, y: x-y,v1,[p*vi for vi
    return v1

def inverse(m,k):
     """
     Return b such that b*m mod k = 1, or 0 if no so
     """
     v = eea(m,k)
     return (v[0]==1)*(v[1] % k)

def crt(ml,al):
     """
     Chinese Remainder Theorem:
     ms = list of pairwise relatively prime integers
     as = remainders when x is divided by ms
     (ai is 'each in as', mi 'each in ms')

     The solution for x modulo M (M = product of ms)
     x = a1*M1*y1 + a2*M2*y2 + ... + ar*Mr*yr (mod M
     where Mi = M/mi and yi = (Mi)^-1 (mod mi) for 1
     """

     M  = reduce(lambda x, y: x*y,ml)        # multi
```

```
    Ms = [M/mi for mi in ml]   # list of all M/mi
    ys = [inverse(Mi, mi) for Mi,mi in zip(Ms,ml)]
    return reduce(lambda x, y: x+y,[ai*Mi*yi for ai

F = crt(modulos,remainders)
```

Once we find F, we can calculate its 17th root in order to find the integer version of the flag:

```
def root(x,n):
    """Finds the integer component of the n'th root
    an integer such that y ** n <= x < (y + 1) ** n.
    """
    high = 1
    while high ** n < x:
        high *= 2
    low = high/2
    while low < high:
        mid = (low + high) // 2
        if low < mid and mid**n < x:
            low = mid
        elif high > mid and mid**n > x:
            high = mid
        else:
            return mid
    return mid + 1

intflag = root(F,17)
```

And from there its easy to get the flag:

```
flag = hex(intflag)[2:-1].decode('hex')
```

Running our script returns:

```
alvaro@winterfell ~/D/h/crypto400> python decrypter.
WARNING: No route found for IPv6 destination :: (no
Secret message! CTF{336b2196a2932c399c0340bc41cd362d
```

Cool!!!!

This is the full script:

```
#!/usr/bin/python
from scapy.all import *
from struct import *
import zlib
from operator import mod

def eea(a,b):
    """Extended Euclidean Algorithm for GCD"""
    v1 = [a,1,0]
    v2 = [b,0,1]
    while v2[0]<>0:
       p = v1[0]//v2[0] # floor division
       v2, v1 = map(lambda x, y: x-y,v1,[p*vi for vi
    return v1

def inverse(m,k):
     """
     Return b such that b*m mod k = 1, or 0 if no so
     """
     v = eea(m,k)
     return (v[0]==1)*(v[1] % k)

def crt(ml,al):
     """
     Chinese Remainder Theorem:
     ms = list of pairwise relatively prime integers
     as = remainders when x is divided by ms
     (ai is 'each in as', mi 'each in ms')

     The solution for x modulo M (M = product of ms)
     x = a1*M1*y1 + a2*M2*y2 + ... + ar*Mr*yr (mod M
     where Mi = M/mi and yi = (Mi)^-1 (mod mi) for 1
     """

     M  = reduce(lambda x, y: x*y,ml)        # multi
     Ms = [M/mi for mi in ml]   # list of all M/mi
     ys = [inverse(Mi, mi) for Mi,mi in zip(Ms,ml)]
     return reduce(lambda x, y: x+y,[ai*Mi*yi for ai

def root(x,n):
    """Finds the integer component of the n'th root
    an integer such that y ** n <= x < (y + 1) ** n.
    """
    high = 1
    while high ** n < x:
        high *= 2
    low = high/2
    while low < high:
        mid = (low + high) // 2
        if low < mid and mid**n < x:
            low = mid
        elif high > mid and mid**n > x:
            high = mid
        else:
            return mid
    return mid + 1

pkts = PcapReader("packets.pcap")
modulos = []
remainders = []
```

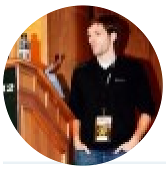
```
    exponents = []
    for p in pkts:
        pkt = p.payload
        if pkt.getlayer(Raw):
            raw = pkt.getlayer(Raw).load
            if str(pkt.sport) == "4919":
                elength = struct.unpack("!H",raw[0:2])[0
                ezip = raw[2:2 + elength]
                e = int(zlib.decompress(ezip))
                nlength = struct.unpack("!H",raw[elength
                nzip =  raw[elength + 4:elength + 4 + nl
                n = int(zlib.decompress(nzip))
                modulos.append(n)
                exponents.append(e)
            if str(pkt.dport) == "4919":
                flaglength = struct.unpack("!H",raw[0:2]
                flagzip = raw[2:2 + flaglength]
                encflag = int(zlib.decompress(flagzip))
                remainders.append(encflag)

    F = crt(modulos,remainders)
    intflag = root(F,17)
    flag = hex(intflag)[2:-1].decode('hex')
    print flag
```

Thanks for reading!

References:

- Chinese Remainder Theorem
- Hastad's broadcast attack



pwntester

Share this post