Weak RSA

28th 2022 / Document No. D23.102.39

Prepared By: `Machina`

Challenge Author(s): `tomtoump`

Difficulty: Easy

Classification: Official

Synopsis

- Perform Wiener's attack on a public RSA key.

Description

- Can you decrypt the message and get the flag?

Skills Required

- Basic research skills.
- Basic mathematical skills.
- Basic understanding of how RSA works.

Skills Learned

- Better understanding of the RSA cryptosystem and its weaknesses.
- Better understanding of small private exponent security risks on RSA.

Enumeration

Analyzing the given files

Once we get our hands on the challenge files, we will find that no source code is provided. All we get are `flag.enc` and `key.pub`, the encrypted flag in a binary file and the public key used to encrypt it. From the title of the challenge, we know that it is an RSA public key. We can use the [pycryptodome] library to load the RSA key and extract its values. Since it is a public key, we expect to get `N`, the public modulus, and `e`, the public exponent:

```python
from Crypto.PublicKey import RSA

def get_pubkey(f):
    with open(f) as pub:
        key = RSA.importKey(pub.read())
    return (key.n, key.e)

N, e = get_pubkey('./key.pub')
print(f'{e = }')
```

Which gives us the 2 numbers:

```
e =
68180928631284147212820507192605734632035524131139938618069575375591806315288775310503696874509130847
52957246260872801929071014966130024613803657934207958043477734411124549518792788113213835795874497424
33659622048350897539876673955116828293912767143595820552901406177978144435307971540406859782299369072
06605
```

Usually the public exponent is a much smaller number, with `0x10001` being a very common choice. So this is something very interesting to research.

Solution

Finding the vulnerability

Since e is very large, it can lead to a small private exponent d. However, a small d leads to a broken RSA encryption as described in [this] paper.

Exploitation

Loading key and ciphertext

We will use the above function `getPubkey` to get the public key values, and we will write another function `getCiphertext` to load the ciphertext and convert it to a number so we can work with it:

```python
def get_pubkey(f):
    with open(f) as pub:
        key = RSA.importKey(pub.read())
    return (key.n, key.e)

def get_ciphertext(f):
    with open(f, 'rb') as ct:
        return bytes_to_long(ct.read())
```

Performing Wiener's attack

There is a python module, owiener, that implements the Wiener attack:

```python
d = owiener.attack(e, N)
```

Decrypting RSA

Now that we have the private exponent, we have everything we need to decrypt RSA and get our plaintext:

```python
def decrypt_rsa(N, e, d, ct):
    pt = pow(ct, d, N)
    return long_to_bytes(pt)
```

Getting the flag

A final summary of all that was said above:

1. We have extracted the public values from the key.
2. We have recovered the private exponent d through the Wiener attack.
3. We have decrypted the ciphertext.

This recap can be represented by code with the `pwn` function:

```python
def pwn():
    N, e = get_pubkey('./key.pub')
    ct = get_ciphertext('./flag.enc')
    d = owiener.attack(e, N)
    flag = decrypt_rsa(N, e, d, ct)
    print(flag)

if __name__ == '__main__':
    pwn()
```