VCTM Practice CTF Challenge-1 (Crypto) Writeup

The challenge is based on exploiting a weakness of RSA algorithm. There are 2 tracks to solve this challenge –

1. Easy – There is a simple tool available to attack RSA encryption. The tool is capable of retrieving private key from weak public key and encrypted data. Link to download the tools is –

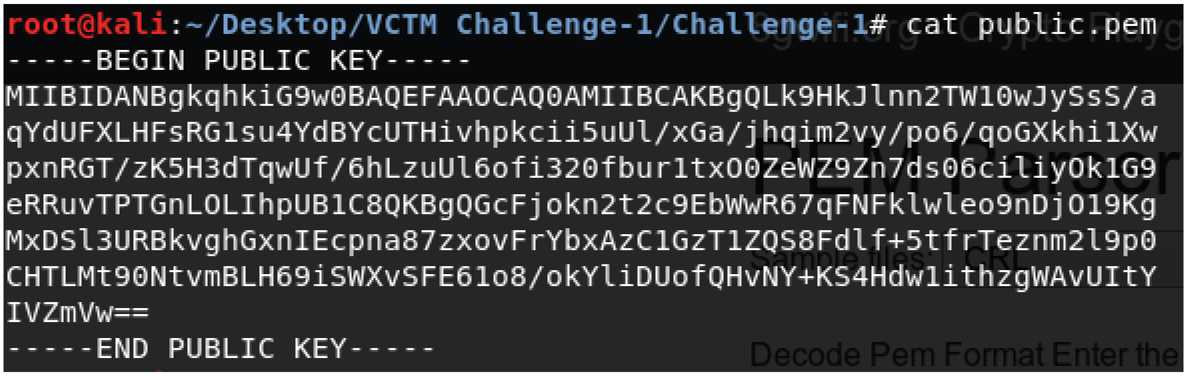
   https://github.com/Ganapati/RsaCtfTool

   A screenshot of the private key retrieval process is given below –

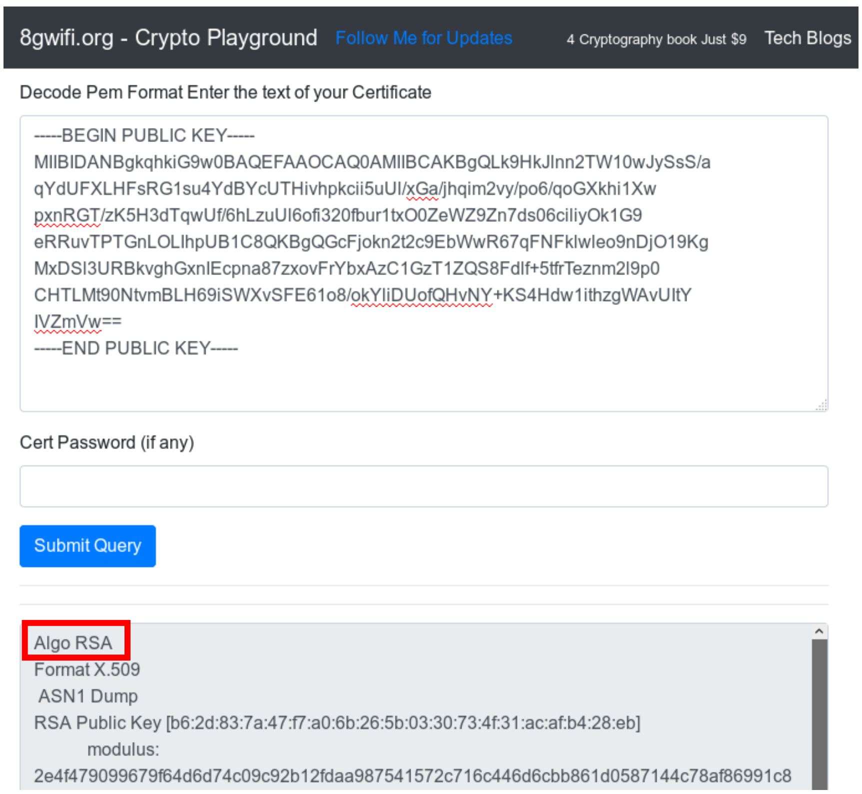
```
root@Kalinux: ~/Desktop/crypto/RsaCtfTool
File  Edit  View  Search  Terminal  Help
oot@Kalinux:~/Desktop/crypto/RsaCtfTool# python RsaCtfTool.py --publickey  /root/D
esktop/crypto/public.pem --private
-----BEGIN RSA PRIVATE KEY-----
MIICOQIBAAKBgQLk9HkJlnn2TW10wJySsS/aqYdUFXLHFsRG1su4YdBYcUTHivhp
kcii5uUl/xGa/jhqim2vy/po6/qoGXkhi1XwpxnRGT/zK5H3dTqwUf/6hLzuUl6o
fi320fbur1txO0ZeWZ9Zn7ds06ciliyOk1G9eRRuvTPTGnLOLIhpUB1C8QKBgQGc
Fjokn2t2c9EbWwR67qFNFklwleo9nDjO19KgMxDSl3URBkvghGxnIEcpna87zxov
FrYbxAzC1GzT1ZQS8Fdlf+5tfrTeznm2l9p0CHTLMt90NtvmBLH69iSWXvSFE61o
8/okYliDUofQHvNY+KS4Hdw1ithzgWAvUItYIVZmVwIgVsK1P6gf1kain0MzE0i0
Dcz7Ugl7i8BOb9H/PYuK+lcCQQE8X2P0BZHwMO9cVwz1OdaptTfjpeonCms1lLjh
xiCuxCfmu+DIpiMVztgXZs4S7OawXMgDGugbdgxhHraQBa0NAkECV49x/qCDywiB
LZHINLB1+2l+zRK1u252AkTsV199cnyZUygV0F87CjRGw2T3XqbsShCfCvsYv0Dm
KhF6KIbcdQIgVsK1P6gf1kain0MzE0i0Dcz7Ugl7i8BOb9H/PYuK+lcCIFbCtT+o
H9ZGop9DMxNItA3M+1IJe4vATm/R/z2LivpXAkAqLWTm+cRGDxGj5qkca54PObqT
3urBAuAoQ3OkSmywEMfiHZSARMDqnVZERKVFk1LguMtFaGaRRsgYIyix4dzC
-----END RSA PRIVATE KEY-----
oot@Kalinux:~/Desktop/crypto/RsaCtfTool# openssl  rsautl -decrypt -inkey /root/Des
ktop/crypto/private.pem -in /root/Desktop/crypto/flag.bin -out flag.txt
oot@Kalinux:~/Desktop/crypto/RsaCtfTool# cat flag.txt
VCTM{h3ll0_th3r3_y0u_g0t_th3_flag}
oot@Kalinux:~/Desktop/crypto/RsaCtfTool# 
```

2. Fun and Learning – Although the solution by oping easy track is fine but it is very important to understand what weakness the tool exploited and how. Hence, we can solve this challenge by performing manual steps such as – writing script and using python implementation of wiener attack. More details about wiener attack are available here. Let's go through the steps involved in this track –

a. *Step 1* - Check public.pem file content and try to figure out the encryption algorithm.



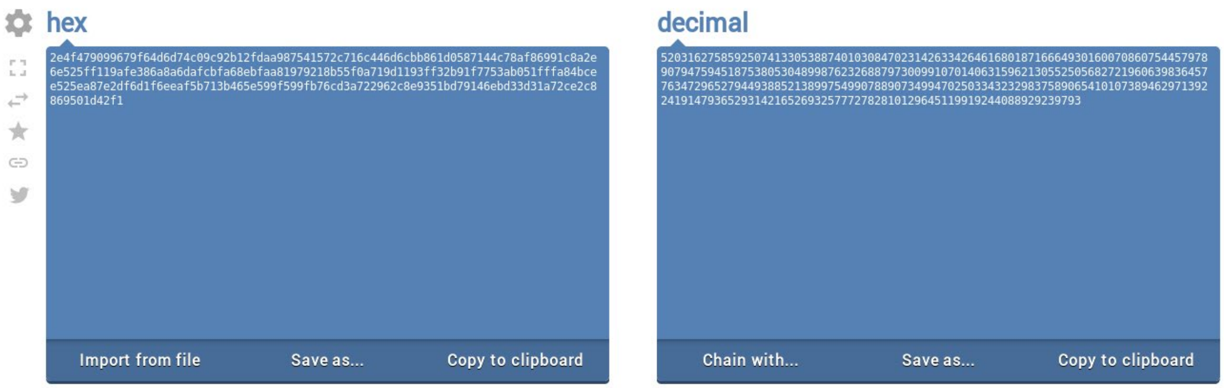
It is base64 encoded, let's try another tool (https://8gwifi.org/PemParserFunctions.jsp) to decode the .pem file.



So now we know that the RSA algorithm is used to generate the public private key pair. Also, the tool provides modulus (m) and exponent (e) values in hex format.

modulus:
2e4f479099679f64d6d74c09c92b12fdaa987541572c716c446d6cbb861d0587144c78af86991c8
a2e6e525ff119afe386a8a6dafcbfa68ebfaa81979218b55f0a719d1193ff32b91f7753ab051fffa84bc
ee525ea87e2df6d1f6eeaf5b713b465e599f599fb76cd3a722962c8e9351bd79146ebd33d31a72c
e2c8869501d42f1
public exponent:
19c163a249f6b7673d11b5b047aeea14d16497095ea3d9c38ced7d2a03310d2977511064be084
6c672047299daf3bcf1a2f16b61bc40cc2d46cd3d59412f057657fee6d7eb4dece79b697da740874
cb32df7436dbe604b1faf624965ef48513ad68f3fa246258835287d01ef358f8a4b81ddc358ad8738
1602f508b5821566657

b. *Step 2* – n and e are in hex format. Let's generate the decimal equivalent using another tool (https://onlinehextools.com/convert-hex-to-decimal).



n =
520316275859250741330538874010308470231426334264616801871666493016007086075445797890794759451875380530489987623268879730099107014063159621305525056827219606398364577634729652794493885213899754990788907349947025033432329837589065410107389462971392241914793652931421652693257772782810129645119919244088929239793

e =
2893772092645268558207272744502598594574213847213464562302931714917536364435654332357622578107628083739265756246236014186382855511847296741216759203441027053438164377210787726999073799777756391170963412155903375472723201367411766424299573666266861590717672059138063419836213941087886891683591717228206069120 87

c. *Step 3* – Our objective is to generate the private key from public key and encrypted text. We have values of n and e, what else is required to generate private key – if we g into details of RSA algorithm, the decryption used below formula –
$m = c^d \bmod n$

We know c (cipher text) and n (modulus) so all we need is d. Let's apply wiener attack, it tries to generate private key from public key in the cases the encryption exponent (e) is too small. Clone the git repository of [python implementation of wiener attack]() in your local machine and modify the *RSAwienerHacker.py* file to provide the values of n and e. The modified code is given below –

```
#!/usr/bin/env python3

import ContinuedFractions, Arithmetic

def hack_RSA(e,n):

    frac = ContinuedFractions.rational_to_contfrac(e, n)

    convergents = ContinuedFractions.convergents

    from_contfrac(frac) for (k,d) in convergents:

        #check if d is actually the key

        if k!=0 and (e*d-1)%k == 0:

            phi = (e*d-1)//k s = n - phi + 1

            # check if the equation x^2 - s*x + n = 0

            # has integer roots

            discr = s*s - 4*n

            if(discr>=0):

                t = Arithmetic.is_perfect_square(discr)

                if t!=-1 and (s+t)%2==0:

                    return d

print(hack_RSA(289377209264526855820727274450259859457421384721346456230293171491753636443565433235762257810762808373926575624623601418638285551184729674121675920344102705343816437721078772699907379977775639117096341215590337547272320136741176642429957366626686159071767205913806341983621394108788689168359171722820606912087,
520316275859250741330538874010308470231426334264616801871666493016007086075445797890794759451875380530489987623268879730099107014063159621305525056827219606398364577634729652794493885213899754990788907349947025033432329837589065410107389462971392241914793652931421652693257772782810129645119919244088929239793))
```

This program will return in the value of d.

d =
39242924240998887071343792618081289828911742678942932818058940576925353704023

d. *Step 4* – Let's use the RSA python implementation library to write a small code to utilize values of n, e and d to generate private key.

```
#!/usr/bin/python

from Crypto.PublicKey import RSA

n =
520316275859250741330538874010308470231426334264616801871666493016007086075445797890794759451875380530489987623268879730099107014063159621305525056827219606398364577634729652794493885213899754990788907349947025033432329837589065410107389462971392241914793652931421652693257772782810129645119919244088929239793

e =
289377209264526855820727274450259859457421384721346456230293171491753636443565433235762257810762808373926575624623601418638285551184729674121675920344102705343816437721078772699907379977775639117096341215590337547272320136741176642429957366626686159071767205913806341983621394108788689168359171722820606912087

d =
392429242409988870713437926180812898289117426789429328180589405769253537040 23

private_key = RSA.construct((n,e,d)).exportKey("PEM")

f = open('private.pem','w')

f.write(private_key)

f.close()
```

This step will generate the private key which is stored in private.pem file.

e. *Step 5* – Now with the private key, we can use openssl to decrypt the flag in flag.bin file. ***openssl rsautl -decrypt -in flag.bin -inkey private.pem -out flag.txt***

The flag is - **VCTM{h3ll0_th3r3_y0u_g0t_th3_f1ag}**