RACTF 2020 Crypto - Really Speedy Algirithm

deuterium — 2020-06-09

Canonical HTML: https://deut-erium.github.io/WriteUps/2020/ractf/crypto/Really˙Speedy˙Algorithm/2020-06-09-RACTF-2020-Really-Speedy-Algorithm.html

Abstract

CTF writeup: RACTF 2020's Really Speedy Algorithm puts RSA on a shot clock: parse the prompts and script the missing parameters for 100 timed questions.

#Really Speedy Algorithm

```
Connect to the network service to get the flag. The included template script may be of
use.

Warning: The challenge server is currently under heavy load, so
is miss-judging how fast solutions are being sent. If your script
is "too slow", this may be the issue. We're working to resolve it.
If your script is timing out a lot, DM @Bottersnike#3605 on
Discord with your script and I can run it for you.
```

This challenge is all about scripting your RSA knowledge, you are given a duration of about a second, to reply with the desired response 100 times in a row to obtain the flag.

Quickly netcatting the server one could see

```
[*] Welcome to my little RSA game.
[*] You will be presented with a number of questions.
[*] You have at most 1000ms to solve any given question.
[*] That should be plenty for any human good at crypography.
[*] Good luck.
[*]
[c] Challenge 1:
[:] p:
112595320707490835575378630950829120586417040569541163533076442603419254278936061428345858103608910483696816094789930962562146339201677582086042938631432 79
[:] phi:
131064587799066203944005827312562617566669586196802373713166674946117857476631324909300704603924316375524117034979774573457951330106727138654981406704336485209955642518891120054845088878754681912501769601803479805113260316058019238267654072352658937053161061649042885639007433350438630146424838391623764024288
[:] e: 65537
[:] ct:
128888444188911299316060627894674165443895799030567545881798184573905727932049715322993601264858919464128364177444155939775493201453351612767170937300333987078025128111979600272358211481975171316360005673613236601924964071568574643794655671843495059544037652057137373486252046141981703380542284670110596876893
[?] pt:
[!] A good cryptologist should be faster than that!
```

It gives some parameters, and we have to provide asked parameter based on the given parameters.

All the challenges can be summarized in one of the following forms:-

- Calculate `pt`, given `e`, `phi`, `p`
  - `d = inverse of e under phi`
  - `q is phi/(p-1) + 1` as `phi = (p-1)*(q-1)`
  - `n = p*q`
  - `pt = (ct^d) % n`
- Calculate `ct`, given `pt`, `p`, `q` and `e`
  - `n = p*q`
  - `ct = (pt^e) % n`
- Calculate `q`, given `n` and `p`
  - `q = n//p`
- Calculate `p`, given `n` and `q`
  - `p = n//q`
- Calculate `n`, given `p` and `q`
  - `n = p*q`
- Calculate `d`, given `p`, `q` and `e`
  - `phi = (p-1) * (q-1)`
  - `d = inverse of e under phi`

Input format begins with specific tags

- Provided parameter begins with the tag `[:]`
- Asked parameter begins with the tag `[?]`
- Challenge number begins with tag `[c]`
- Crucial information begins with tag `[!]`
- General statements/information begins with tag `[*]`

Putting all the above information in a really hacky script

```
from pwn import remote
from gmpy2 import invert

#HOST, PORT = "95.216.233.106" ,62467
HOST, PORT = "139.59.190.222", 31337
REM = remote(HOST, PORT)
```

```
def recieve():
    n = None  # only the recieved parameters returned
    p = None
    q = None
    ct = None
    pt = None
    e = None
    d = None
    phi = None
    param = None  # The parameter specified in a challenge to find
    while True:
        data = REM.recvn(3).decode()
        print(data, end="")
        if data.startswith('[!]'): # line of critical information
            print(REM.recvline().decode().strip())
            print(REM.recvline().decode().strip())
        elif data.startswith('[:]'):  # line of parameter specification type
            param_name = REM.recvuntil(b': ').decode().strip()[:-1]
            print(param_name)
            val = int(REM.recvline().decode().strip())
            if param_name == 'n':
                n = val
            elif param_name == 'p':
                p = val
            elif param_name == 'q':
                q = val
            elif param_name == 'ct':
                ct = val
            elif param_name == 'pt':
                pt = val
            elif param_name == 'phi':
                phi = val
            elif param_name == 'e':
                e = val
            elif param_name == 'd':
                d = val
        elif data.startswith('[?]'): # line specifying which parameter to find
            param = REM.recvuntil(b': ').decode().strip()[:-1]
            print(param)
            return (n, p, q, ct, pt, e, d, phi, param)
            break
        elif data.startswith('[*]') or data.startswith('[c]'): # random informative
line
            print(REM.recvline().decode().strip())
        else:
            print(data)
            print(REM.recvall().decode().strip())
    return (n, p, q, ct, pt, e, d, phi, param)


def solve(values):
    """
    Takes the parameters and calculates the desired parameter
    """
    # print(values)
    n, p, q, ct, pt, e, d, phi, param = values
    if param == 'pt':
        d = invert(e, phi)
        q = phi // (p - 1) + 1
        n = p * q
```

```
        pt = pow(ct, d, n)
        return pt
    elif param == 'ct':
        ct = pow(pt, e, p * q)
        return ct
    elif param == 'q':
        return n // p
    elif param == 'p':
        return n // q
    elif param == 'n':
        return p * q
    elif param == 'd':
        phi = (p - 1) * (q - 1)
        return int(invert(e, phi))
    else:
        print("NOT IMPLEMENTED", param)


def send(val):
    REM.sendline(str(val).encode())


while True:
    rec = solve(recieve())
    print(rec)
    send(rec)
```

Having bad connection with high latency, I requested one of the admins to run the script and got flag in return 😄

ractf–F45t35tCryp70gr4ph3rAr0und"

> Interactive controls are omitted from this PDF. See the canonical HTML version.