RSA Broadcast RUS

7 февраля 2020 г.

1 Атака на широковещательный вариант RSA (RSA Broadcast attack)

Представьте, что вы используете открытую экспоненту $e = 3$ (на так давно она была достаточно популярна), но вы защифроваываете только сообщения, которые переаолняют модуль N при возведении в третью степень, так что нельзя просто взять и извлечь корень третьей степени. Правда ли это безопасно? Есть довольно простой сценарий, который наглядно проказывает проблему небольшой открытой экспоненты. Представьте, что пользователь использует RSA, чтобы посылать сообщения на несколько серверов (все с экспонентой 3, но разными модулями N). Пусть таких серверов 3. Одно и то же сообщение шифруется 3 раза: с ключами первого, второго и третьего серверов.

$$C_1 = M^3 mod N_1$$

$$C_2 = M^3 mod N_2$$

$$C_3 = M^3 mod N_3$$

Почти невозможно дешифровать каждый из шифротекстов C_i по одинчке, но с тремя Марвин(или Мэллори, как вам больше нравится) можно решить эту задачу. ## Китайская теорема об остатках Если известны остатки от Евклидова деления целого числа n на несколько целых чисел, то можно определить уникальный остаток от деления n на произведение этих целых чисел, при условии, что все делители (модули, по которым брали остатки) попарно простые.

Как мы можем использовать эту теорему? Во-первых, нам необходимо проверить, что все делители (N_1, N_2, N_3) взаимно (попарно) простые. Но, если вдруг они не взаимно простые, то наибольший общий делитель двух из них больше единицы и либо они равны (и мы не можем их испольовать), либо они имеют в составе одно и то же простое число -их . В последнем случае мы и так можем расшифровать сообщение. Так что будем считать, что они попарно взаимнопростые. Это значит, что существует такой X, что:

$$X < N_1 N_2 N_3$$

$$X = C_1 mod N_1$$

$$X = C_2 mod N_2$$

$$X = C_3 mod N_3$$

и такой X уникален.

Давайте рассмотрим $C = M^3$. Так как $M < N_1$ и $M < N_2$ и $M < N_3$, то $C = M^3 < N_1 N_2 N_3$. А ещё:

$$C = C_1 mod N_1$$

$$C = C_2 mod N_2$$

$$C = C_3 mod N_3$$

Так что используя критайскую теорему об остатках можно найти C и всё, что останется сделать - это извлечь кубический корень. ## Как получить C?

Пусть $N_i, i = 1, k$ - модули, а $c_i, i = 1, k$ - остатки по делению. $N = N_1 N_2 ... N_k$, а $M_i = N/N_i$ Тогда $C = (\sum_{i=0}^{k} C_i M_i (M_i^{-1} mod N_i)) mod N$

Воспользуйтесь выражением и вытащите флаг из трёх зашифрованных сообщений, полученных с сервера. Удачи!

[1]:
```
import socket
import re
from Crypto.Util.number import inverse,long_to_bytes,bytes_to_long
class VulnServerClient:
    def __init__(self,show=True):
        """Initialization, connecting to server"""
        self.s=socket.socket(socket.AF_INET,socket.SOCK_STREAM)
        self.s.connect(('cryptotraining.zone',1339))
        if show:
            print (self.recv_until().decode())
    def recv_until(self,symb=b'\n>'):
        """Receive messages from server, by default till new prompt"""
        data=b''
        while True:

            data+=self.s.recv(1)
            if data[-len(symb):]==symb:
                break
        return data
    def get_public_keys(self,show=True):
        """Receive public keys from the server"""
        self.s.sendall('public\n'.encode())
        response=self.recv_until().decode()
        if show:
            print (response)
        e1=int(re.search('(?<=e1: )\d+',response).group(0))
        N1=int(re.search('(?<=N1: )\d+',response).group(0))
        e2=int(re.search('(?<=e2: )\d+',response).group(0))
        N2=int(re.search('(?<=N2: )\d+',response).group(0))
        e3=int(re.search('(?<=e3: )\d+',response).group(0))
        N3=int(re.search('(?<=N3: )\d+',response).group(0))

        return [(e1,N1),(e2,N2),(e3,N3)]

    def get_ciphertexts(self,show=True):
        """Receive ciphertexts from the server"""
```

```
        self.s.sendall('ciphertext\n'.encode())
        response=self.recv_until().decode()
        if show:
            print (response)
        c1=bytes_to_long(bytes.fromhex(re.search('(?<=ciphertext1:␣
↪)[0-9a-f]+',response).group(0)))
        c2=bytes_to_long(bytes.fromhex(re.search('(?<=ciphertext2:␣
↪)[0-9a-f]+',response).group(0)))
        c3=bytes_to_long(bytes.fromhex(re.search('(?<=ciphertext3:␣
↪)[0-9a-f]+',response).group(0)))
        return (c1,c2,c3)

    def __del__(self):
        self.s.close()
```

[2]:
```
vs=VulnServerClient()
pk_list=vs.get_public_keys()
(c1,c2,c3)=vs.get_ciphertexts()
```

```
Welcome to RSA broadcast task
Available commands:
help - print this help
public - show public keys
ciphertext - show ciphertexts
quit - quit
>
e1: 3
N1: 20287982006618431876793244706487063574769448388426702838915722457901061849764
47243626039531795325398275543653293234830132766488917963785071035930009368913228
46098718727133325953848182431476448546554772557085987908949429403596359635314342
61288990889827227232217334114165156730168742722649122944238549378576569971598646
24831244231716527562039198797157055907715253054467883225128444276488229226822053
88423707896633544989180321378196798302096862401020103125458117084856441433418990
68127432781004629189088988249662139540320855429177722770638935536567888515701105
24658930707312433607838782424604007019355796827163459647835 81
e2: 3
N2: 19461826656993775602233892694656909792660373232000384211810816270789889052584
94698160962108833920376795505059935112703645007973713125768662138084135228774544
84246665882296260574836114845774508160454919015698629693496502771810686240938284
17156980569552655552046208392613888308267963491767042426692994638308425125646883
72733269120922089406251010437634980936383704472101088630786058537742421729307186
22708737748976038337571022634250268274774822635019271377670016639627161725800462
58359389697367042361825252217879227323662108134646132859434584008363584348464551
00282160172155484438176350214865799878458490158345252484035 69
e3: 3
N3: 21576795955319933162780162589243062240645136282924505451251041003657099282864
96733559781987591118543333674003519120138899019875391980134365196187463909651874
```

```
94910390993435720830028431908565639307591762759339257606796030370738921636096216
64570954521583140185192439160383490958754771796588603890957120549323502385606270
56760553605426061976992709566236120594344688089130142528737116646690360014768760
33466801652854702139322954477234402112795351169729172489093923989602378894935730
69202923399447965912580489756649337138758741588146119135480820729610142893116374
233612478079989267341580066616869035545850231761766594993754l
>
ciphertext1: 30d097f552419a68a8611903f6f0cf3d1baf14672d28151eab8307835fd78fa3347
d250555beada3bcdae7d796f45139f1127db71c591a745246118139ad3a06a8ded25526e8af11723
2b3a75b7eee7fd197b3fafb83d5a41dd091752138c734ddfafc92bb68f1232412d6be328a096f615
22fe4bda4e7d521b68c411c39929e2bfa851d9619eddcbd6782a4f3f667300960e2f683ccb75fa8c
fc0ebd624701b05203f1e7981d8d39866cd35ddc6b42fdd9a5fe4dab257069cd756b3b59917fb4c1
d02709d76999db9afc96bf2bddda6115382d555816756f7cbbb6a3510ab7521dbfa09903c2642ee0
de5bd97493a8f3c80edeff79999552f69f20f6042afb7
ciphertext2: 6c304723aae36f6613e75546a851364ddb241ae531c4700dccf3e68483c44c87788
af7972b5526bf3d1d49884405010c1c41e4e086f1c73a070e8311add5041be1b8348c728220a4a56
9b1298e74a4ada1a675aa76d8dbf80ae127772556efbdfc3b8451e130d65513437a4a3f6a9feeec0
cb68007739462d3b02d35ff5709c7cf5ce11d6d7f00d72044c7a902ff3ad5f8e2552891af24daf96
d3afe83e0f349f3d46930b3b62e0a7ab63b7f87711da3e3834500f97fa91fb074e3f2c2342123f94
aa2c777bedfb0cac3b975add559f32731563ee2a628c58af737d3fee8d0292f3ce9e8bec5554f29f
0fcd38e3bc72bac31567d6d9bed1450556681a2ab7ce8
ciphertext3: 41d6a25a3bcae7cdd3140282f2d979d428c49bd314cbb28381956358a2f6e83dae2
f882a74ef041421ee3957c4704174b8f295a774e09e099028a79b2599d2927b3603c2c5c1748a09c
cfe0a84dd33b120f312e55a456f7a9f32f158b3c85b70ccbb1f36511557d01d3be6dea32aa2e9493
32f2ae6600d45f1abd319d4054263bce01b176adb604c132493eaf57741696e073874df075be3e50
270b0333cff20bcd7c1fd99f476aada0f420de07a2aa049681acc2ebc60507f5090fb47e27bc441b
64075970e4ff5aed0770240f8de6f01fdb4305e25524d964b466ec5d5eabebdca58d1c189c199a7a
d335ad1803b80c2bf4d39ddf3d2d51f61b886272e59b1
>
```

[ ]: