Wieners Attack RUS

7 февраля 2020 г.

1 Атака Винера

Расшифрование или создание подписи при помощи RSA обычно достаточно энергоёмкий и долгий процесс, так как мы не можем просто выбрать красивую закрытую экспоненту d с низким весом Гэмминга (её очень просто было бы угадать). Поэтому у людей иногда появляется позыв выбрать такую открытую экспоненту e, что длина d в битах в несколько раз меньше, чем длина модуля N, например, N - 2048, а d - 500 бит. Интуитивно может показаться, что раз d сложно угадать (всё-таки 500 бит), можно быть спокойным за безопасной криптосистемы. К сожалению, это очень опасная ошибка. ## Теорема (M. Wiener) Пусть $N = pq$ с $q < p < 2q$. Пусть $d < \frac{1}{3}N^{\frac{1}{4}}$. Зная (N, e), у которого $ed = 1 mod \varphi(N)$, Марвин может эффективно восстановить d.

1.0.1 Доказательство (взято из Twenty Years of Attacks on the RSA Cryptosystem в авторстве Dan Boneh)

Доказательство основывается на приближенных значениях при использовании непрерывных дробей. Поскольку существует $ed = 1 mod \varphi(N)$, то сучществует такое k, что $ed - k\varphi(N) = 1$. Следовательно,

$$|\frac{e}{\varphi(N)} - \frac{k}{d}| = \frac{1}{d\varphi(N)}$$

Значит, $\frac{k}{d}$ - это приближенное значение к $\frac{e}{\varphi(N)}$. Хоть Марвин и не знает $\varphi(N)$, он може воспользоваться N, чтобы получить его примерную оценку. Действительно, раз $\varphi(N) = N - p - q + 1$ и $p + q - 1 < 3\sqrt{N}$, получается, что $|N - \varphi(N)| < 3\sqrt{N}$

Подставляя N вместо $\varphi(N)$ получаем:

$$|\frac{e}{N} - \frac{k}{d}| = |\frac{ed - k\varphi(N) - kN + k\varphi(N)}{Nd}| = |\frac{1 - k(N - \varphi(N))}{Nd}| \leq |\frac{3k\sqrt(N)}{Nd}| = \frac{3k}{d\sqrt{N}}$$

Заметим, что $k\varphi(N) = ed - 1 < ed$. Так как $e < N$, видно, чтот $k < d < \frac{1}{3}N^{\frac{1}{4}}$. Поэтому получаем:

$$|\frac{e}{N} - \frac{k}{d}| \leq \frac{1}{dN^{\frac{1}{4}}} < \frac{1}{2d^2}$$

Это классическое отношение приближения. Количество дробей $\frac{k}{d}$ with $d < N$, дающих приближенное значение $\frac{e}{N}$ с такой точностью, ограничено $log_2 N$ На самом деле, все такие дроби могут быть получены как подходящие дроби непрерывной дроби $\frac{e}{N}$. Всё, что надо сделать, это посчитать $logN$ подходящих дробей непрерывной дроби $\frac{e}{N}$. Одна из них будет равна $\frac{k}{d}$. Так как $ed - k\varphi(N) = 1$, то $(k, d) = 1$ и значит $\frac{k}{d}$ - это сокращенная дробь. Это и есть линейный по времени алгоритм восстановления закрытого ключа d. ЧТД

1.1 Непрерывные и подходящие дроби

Непрерывная дробь - это выражение, полученное при помощи последовательного процесса представления числа как суммы его целочисленной части и обратного элемента по умножению к оставшейся части, далее представляя эту оставшуюся часть, как сумму целочисленной части и обратного элемента и так далее. Пример: Для действительного числа $x > 0$ и целых положительных a_i, for $i = 1, ..., n$

$$x = a_0 + \cfrac{1}{a_1 + \cfrac{1}{a_2 + \cfrac{1}{\ddots + \cfrac{1}{a_n}}}}$$

- это непрерывная дробь. Целые числа a_0, a_1, и т.д., называются элементами или неполными частными непрерывной дроби. Есть много способов записи непрерывных дробей, мы будем использовать следующую:

$$x = [a_0; a_1, a_2, ...]$$

Непрерывная дробь может сформировать своё приближение при помощи первых элементов. Таки приближения называются подходящими дробями. Очевидно, что для рационального x количество таких подходящих дробей конечно finite. Первые четыре подходящие дроби непрерывной дроби:

$$\frac{a_0}{1}, \frac{a_1a_0 + 1}{a_1}, \frac{a_2(a_1a_0 + 1) + a_0}{a_2a_1 + 1}, \frac{a_3(a_2(a_1a_0 + 1) + a_0) + (a_1a_0 + 1)}{a_3(a_2a_1 + 1) + a_1}$$

Если существуют последующие подходящие дроби, они могут быть вычислены рекурсивно. h_i обозначим числители, а k_i - знаменатели. Тогда ряд подходящих дробей может быть вычислен следующим образом:

$$\frac{h_i}{k_i} = \frac{a_ih_{i-1} + h_{i-2}}{a_ik_{i-1} + k_{i-2}}$$

Реализуйте алгоритм для нахождения элементов и подходящих дробей для любого рационального числа.

Подсказка: вы можете использовать алгоритм Евклида для вычисления элементов

[ ]:
```

```

Теперь, когда вы написали алгоритм поиска подходящих дробей, давайте рассмотрим, как при их помощи найти p и q. Поскольку $\frac{k}{d}$ - это сокращенная дробь, знаменатель одной из подходящих дробей - это и есть закрытая экспонента d. Этого уже достаточно, чтобы расшифровать шифртекст, но мы может и факторизовать N. Мы также знаем k, а $\varphi(N) = \frac{ed-1}{k}$, значит мы можем вычислить $\varphi(N)$.

$$N - \varphi(N) = pq - (p-1)(q-1) = pq - pq + p + q - 1 = p + q - 1$$

$$q = N - \varphi(N) - p + 1$$

$$N = pq = p(N - \varphi(N) - p + 1) = pN - p\varphi(N) - p^2 + p$$
$$p^2 - p(N - \varphi(N) + 1) + N = 0$$

Получаем красивое квадратное урвавнение, корнями которого и будут p и q

Давайте используем полученные знания на уязвимом сервере. Задача такая же, как и в blinding таске. Нужно отправить на сервер правильную подпись для сообщения 'flag', чтобы его получить. Шаги: 1. Найти d хитроумно используя непрерывные дроби 2. Найти p и q 3. Подписать сообщение и послать на сервер 4. Profit

[1]:
```
import socket
import re
from Crypto.Util.number import inverse,long_to_bytes,bytes_to_long
class VulnServerClient:
    def __init__(self,show=True):
        """Инициализация, подключаемся к серверу"""
        self.s=socket.socket(socket.AF_INET,socket.SOCK_STREAM)
        self.s.connect(('cryptotraining.zone',1338))
        if show:
            print (self.recv_until().decode())
    def recv_until(self,symb=b'\n>'):
        """Получение сообщения с сервера, по умолчанию до приглашения к вводу␣
 ↪команды"""
        data=b''
        while True:

            data+=self.s.recv(1)
            if data[-len(symb):]==symb:
                break
        return data
    def get_public_key(self,show=True):
        """Получение открытого ключа с сервера"""
        self.s.sendall('public\n'.encode())
        response=self.recv_until().decode()
        if show:
            print (response)
        e=int(re.search('(?<=e: )\d+',response).group(0))
        N=int(re.search('(?<=N: )\d+',response).group(0))
        self.num_len=len(long_to_bytes(N))
        self.e,self.N=e,N
        return (e,N)

    def checkSignatureNumber(self,c,show=True):
        """Проверка сигнатуры (на сервере) для подписи в числовом␣
 ↪представлении"""
        try:
            num_len=self.num_len
        except KeyError:
```

```
            print ('You need to get the public key from the server first')
            return
        signature_bytes=long_to_bytes(c,num_len)
        self.checkSignatureBytes(signature_bytes,show)

    def checkSignatureBytes(self,c,show=True):
        """Проверка сигнатуры (на сервере) для подписи в байтовом␣
↪представлении"""
        try:
            num_len=self.num_len
        except KeyError:
            print ('You need to get the public key from the server first')
            return
        if len(c)>num_len:
            print ("The message is too long")
            return

        hex_c=c.hex().encode()
        self.s.sendall(b'flag '+hex_c+b'\n',)
        response=self.recv_until(b'\n').decode()

        if show:
            print (response)

        if response.find('Wrong')!=-1:
            print('Wrong signature')
            x=self.recv_until()
            if show:
                print (x)
            return
        flag=re.search('CRYPTOTRAINING\{.*\}',response).group(0)
        print ('FLAG: ',flag)
    def setPrivateKey(self,p,q):
        """Выставить закрытый ключ"""
        self.p=p
        self.q=q
        self.d=inverse(self.e,(p-1)*(q-1))

    def signMessageBytes(self,m):
        """Подписать сообщение, после того как найден закрытый ключ"""
        try:
            num_len=self.num_len
        except KeyError:
            print ('You need to get the public key from the server first')
            return
        if len(m)>num_len:
            print('m too long')
```

```
        if len(m)<num_len:
            m=bytes([0x0]*(num_len-len(m))) + m
        signature_bytes=long_to_bytes(pow(bytes_to_long(m),self.d,self.N))
        return signature_bytes

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

[2]:
```
vs=VulnServerClient()
(e,N)=vs.get_public_key()
```

```
Welcome to the Wiener attack task
 Private exponent d is just 500 bits, so you should be able to find it
Available commands:
help - print this help
public - show public key
flag <hex(signature(b'flag'))> - print flag
quit - quit
>
e: 883955104397844360839802589678079320200301053675860631429632241079629322700900
637792870473311552753056833515750354593566814028068767764438482882777434179174
1675311017053491920564415641618138819406621038589908795280642494559623024398109
0686122984068829991815467478093723627539243672428333723190058091506420994459082
9593179768699333704428744236644694463302339528604487059166123052119894425485654
0088737753911491026348538755316415798095543587870314159684746164449762012523139
0330179843055661987371450397018342663325443054774257588676997705586499693980142
841132838109639976782671895045211865978415489119582699166192296 15
N: 2125632243008959885433870068920027190367541374095231465087756330559520029821
4795840279408610649497535155919886199494094094113946327038855028353113112375895
8799623003985099830415299465089729855511075800086427546384784838006984961812360
3275447786309956087764630457865643426499984650766405764962904115568809167308773
7339579837961500153140350438182833535429065058704101283648761716829403428808106
0340011418775963432932239367388470383293641635502826193087609782306888074314016
4934435569101657199682107768339106258747384129656119582367486258484881517847096
90389684573576432874392053703896310262986981095078473647234183 09709
>
```

[ ]: