ZIP Password Cracker

Team18 :)
110550085 房天越
109652013 雷其諭
0816085 曾嘉佑

Outline

1. Intro. to ZIP
2. Common Attacks
3. ZIPCrypto
4. CRC32
5. Plaintext Attack
6. Conclusion
7. Reference

Intro. to ZIP

1. 開源、被廣泛支援
2. 支援ZipCrypto及AES-256加密
3. 僅對檔案內容加密
4. 修改及刪除檔案不需密碼

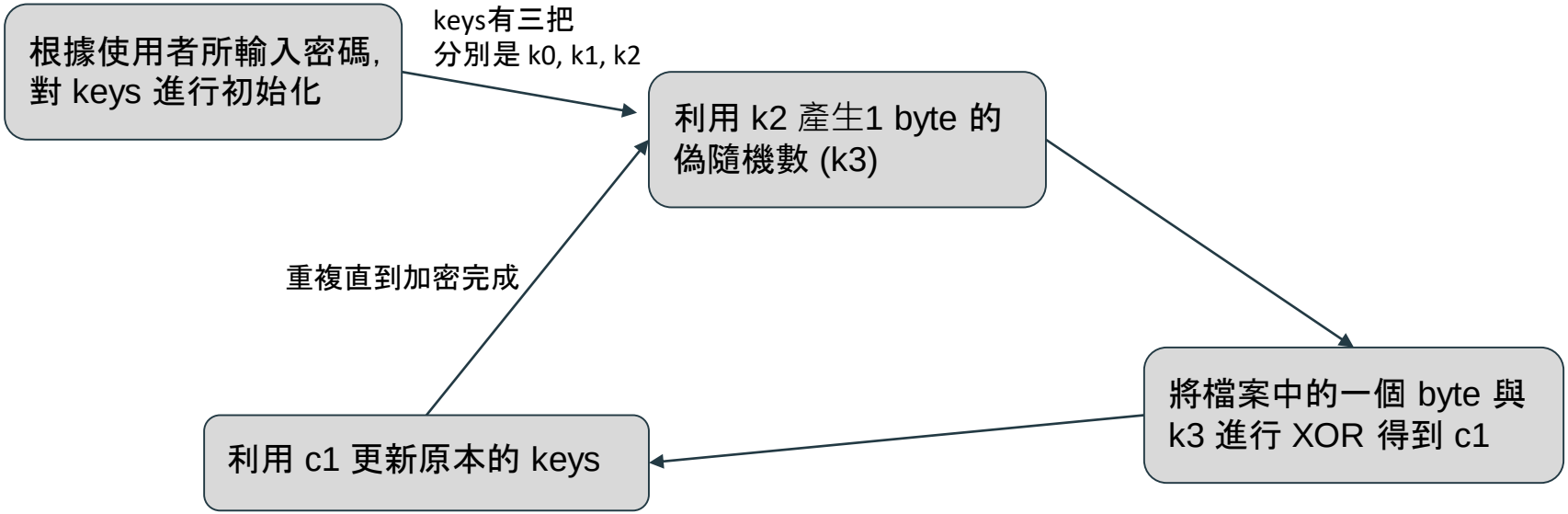
Common Attacks

一些常見的手段包括以下這些：

1. 暴力破解 (BruteForce Attack)
2. 字典攻擊 (Dictionary Attack)
3. 已知明文攻擊 (Plaintext Attack)

我們將主要以Plaintext Attack為重點進行探討

ZIPCrypto


根據使用者所輸入密碼,
對 keys 進行初始化
keys有三把
分別是 k0, k1, k2
利用 k2 產生1 byte 的
偽隨機數 (k3)
將檔案中的一個 byte 與
k3 進行 XOR 得到 c1
利用 c1 更新原本的 keys
重複直到加密完成

Pros and Cons of ZIPCrypto

優點：

開源、簡單好用、泛用性高

缺點：

使用對稱式加密，安全性有極大疑慮

Keys

這是 keys 的初始化方式 →

```
void InitialKeys(string pwd)
{
    for (auto c: pwd)
        UpdateKeys(c);
}
```

```
uint32_t k0 = 0x12345678;
uint32_t k1 = 0x23456789;
uint32_t k2 = 0x34567890;
```

← 這是 k0, k1, k2 的預設值

這是 keys 的更新方式 →
134775813為magic number

```
void UpdateKeys(uint8_t p) {
    k0 = crc::Crc32(k0, p);
    k1 = (k1 + crc::lsb(k0)) * 134775813 + 1;
    k2 = crc::Crc32(k2, crc::msb(k1));
}
```

```
uint8_t k3()
{
    uint16_t tmp = k2 | 3;
    return crc::lsb((tmp * (tmp ^ 1)) >> 8);
}
```

← 這是 k3 的計算方式

CRC32

概念：

將原本訊息末尾加上32位數的0

整個訊息對CRC32定義的primitive polynomial進行GF(2)的除法

校驗碼即為其餘數

CRC32

ZIPCrypto採用版本為CRC32-IEEE 802.3

預先定義的多項式如下:

$$x^{32}+x^{26}+x^{23}+x^{22}+x^{16}+x^{12}+x^{11}+x^{10}+x^{8}+x^{7}+x^{5}+x^{4}+x^{2}+x+1$$

LSB MSB函數定義如右→

B代表的是 byte而非bit

```
constexpr uint8_t msb(uint32_t value)
{
    return static_cast<uint8_t>((value >> 24) & 0xFF);
}

constexpr uint8_t lsb(uint32_t value)
{
    return static_cast<uint8_t>(value & 0xFF);
}
```

CRC32加速

CRC32在運算時可以由事先運算建表後查表來加速

```
uint32_t Crc32(uint32_t pval, uint8_t b) {
    return pval >> 8 ^ CrcTable[lsb(pval) ^ b];
}
```

```
uint32_t Crc32Inv(uint32_t pval, uint8_t b) {
    return pval << 8 ^ CrcInvTable[msb(pval) >> 24] ^ b;
}
```

其中表格是由

```
void InitTable()
{
    for (int i = 0; i < 256; ++i)
    {
        uint32_t crc = i;
        for (int j = 0; j < 8; ++j)
        {
            if (crc & 1)
                crc = crc >> 1 ^ 0xedb88320;
            else
                crc = crc >> 1;
        }
        crc::CrcTable[i] = crc;
        crc::CrcInvTable[crc::msb(crc)] = crc << 8 ^ i;
    }
}
```

所建立

Encrypt and Decrypt

```
void Encrypt(std::vector<uint8_t>& data)
{
    for (uint8_t& p : data)
    {
        uint8_t c = p ^ k3();
        UpdateKeysBackward(c);
        p = c;
    }
}
```

```
void Decrypt(std::vector<uint8_t>& data)
{
    for (uint8_t& c : data)
    {
        uint8_t p = c ^ k3();
        UpdateKeys(p);
        c = p;
    }
}
```

PlainText Attack

Step1. 找到k3

右圖為 ZIPCrypto 的加密原理

由於 $c = p \oplus k3$

$k3 = p \oplus c$

```
void Encrypt(std::vector<uint8_t>& data)
{
    for (uint8_t& p : data)
    {
        uint8_t c = p ^ k3();
        UpdateKeysBackward(c);
        p = c;
    }
}
```

Step2-1.

```
uint8_t k3()
{
    uint16_t tmp = k2 | 3;
    return crc::lsb((tmp * (tmp ^ 1)) >> 8);
}
```

附圖是 k3的生成方式。

我們可以從中推斷

(1):k3 長度為 8 bits， 且 tmp 長度為 16 bits

(2):因為 tmp= k2 OR 3， 所以我們 有tmp之後可以確定 k2 的 [2, 15] bits

Step2-2.

```
uint8_t k3()
{
    uint16_t tmp = k2 | 3;
    return crc::lsb((tmp * (tmp ^ 1)) >> 8);
}
```

進一步推導:

(1):因為 k3 只有 8 bits， k3 最多有 2^8 種可能

(2): tmp 的 [0, 1]bits 都是 1， 因此有 2^14 種可能的 tmp

(3): 給定 k3 後， 可以找到 2^14/2^8 = 2^6 種可能的tmp

(4): tmp 運算只用到 k2 的[2,16)， 因此剩下的k2 [16,31] 有 2^16種可能

(5):給定k3後， 可以得到 2^6 (k2[0,16))*2^16(k2[16,31]) = 2^22種可能的 k2

Step2-3.

```
void UpdateKeys(uint8_t p) {
    k0 = crc::Crc32(k0, p);
    k1 = (k1 + crc::lsb(k0)) * 134775813 + 1;
    k2 = crc::Crc32(k2, crc::msb(k1));
}
```

讓 ki_{n+1}代表迭代了 $n+1$ 次的 ki，我們今天假定一個任意的 $k2_{n+1}$

由算法公式可以得到 $k2_{n+1} = Crc32(k2_n, msb(k1_{n+1}))$

再藉由CRC32的逆運算，我們可以得到

$k2_n = (k2_{n+1} << 8) \bigoplus CrcInvTable[MSB(k2_{n+1})] \bigoplus MSB(k1_{n+1})$

很顯然 $k2_{n+1}$的$[2, 32)$是已知的，因此$k2_{n+1} << 8$ 的 $[10, 32)$是已知

整個 CrcInvTable 都是已知且$MSB(k1_{n+1})$的 $[8, 32)$ 位都是 0

再者 $k2_n$ 也可以用上一頁同樣的方式找到第$[2, 16)$位

$$k2_n = (k2_{n+1} << 8) \oplus CrcInvTable[MSB(k2_{n+1})] \oplus MSB(k1_{n+1})$$

Step2-4.

因此我們可以簡單建立表格

由表格可知，左右兩式的[10,16)皆為已知，不同的點在於左式中有64種可能而右式中只有一種

因此我們可以事先排除[10,16)不同的可能性

平均每64種可能性會被縮減至一種

而我們可以依此類推

縮小 $k2_{n-1}, k2_{n-2}$ ...的範圍

SIDE	表達式	已知位數	分布範圍	可能數量
LHS	$k2_n$	14	[2,16)	64
RHS	$(k2_{n+1} << 8)$	22	[10,32)	1
	$CrcInvTable[MSB(k2_{n+1})]$	32	[0,32)	1
	$MSB(k1_{n+1})$	24	[8,32)	1
	左側總計	14	[2,16)	64
	右側總計	22	[10,32)	1
	左右聯集	30	[2,32)	
	左右交集	6	[10,16)	

Step3-1.

```
void UpdateKeys(uint8_t p) {
    k0 = crc::Crc32(k0, p);
    k1 = (k1 + crc::lsb(k0)) * 134775813 + 1;
    k2 = crc::Crc32(k2, crc::msb(k1));
}
```

在前幾頁裡，我們用到了以下這條式子

$$k2_n = (k2_{n+1} << 8) \bigoplus CrcInvTable[MSB(k2_{n+1})] \bigoplus MSB(k1_{n+1})$$

我們只要簡單將他移項便可得到

$$MSB(k1_{n+1}) = (k2_{n+1} << 8) \bigoplus CrcInvTable[MSB(k2_{n+1})] \bigoplus k2_n$$

因此我們可以很輕易地取得$MSB(k1_i)$ for $i = 3, 4, \ldots\ldots, n+1$

再來我們可以觀察$k1$的關係式，可以得到

$$k1_{n-1} + LSB(k0_n) = (k1_n - 1) * 134775813^{-1}$$

其中134775813^{-1} 是134775813在 mod32 的 inverse，具體值為0xd94fa8cd

Step3-2.

```
void UpdateKeys(uint8_t p) {
    k0 = crc::Crc32(k0, p);
    k1 = (k1 + crc::lsb(k0)) * 134775813 + 1;
    k2 = crc::Crc32(k2, crc::msb(k1));
}
```

$$k1_{n-1} + LSB(k0_n) = (k1_n - 1) * 134775813^{-1}$$

同時對左右兩邊取 MSB，因為 $LSB(k0_n)$ 只有一個 byte 因此取完 MSB之後對結果的影響並不大 (± 1)

至此我們可以得到 $k1_{n-1}$ 和 $k1_n$ 之間的關係

我們只需要像之前找k2那樣把k1找出來就行了

Step3-3.

```
void UpdateKeys(uint8_t p) {
    k0 = crc::Crc32(k0, p);
    k1 = (k1 + crc::lsb(k0)) * 134775813 + 1;
    k2 = crc::Crc32(k2, crc::msb(k1));
}
```

$k1_{n-1} + LSB(k0_n) = (k1_n - 1) * 134775813^{-1}$

我們透過上式能得知$LSB(k0_{n+1})$

也就是$k0_{n+1}$的$[0, 8)$。觀察$k0$的關係式，可以得到

$k0_{n+1} = Crc32(k0_n, P_i) = (k0_n >> 8) \bigoplus CrcTable[LSB(k0_n) \bigoplus P_i]$

從這條式子裡我們可以得到$k0_{n+1}$的$[24, 32)$，因為$CrcTable[LSB(k0_n) \bigoplus P_i]$是已知，而$k0_n >> 8$ 的$[24, 32)$ 都是0

再回去看$k0_n = Crc32(k0_{n-1}, P_i) = (k0_{n-1} >> 8) \bigoplus CrcTable[LSB(k0_{n-1}) \bigoplus P_i]$

這次我們可以得到$k0_n$的$[0, 8)$和$[16, 32)$，因為條件多了$[24, 32)$

再做兩次我們就可以得到一整串完整的 $k0$

Step4.

至此推導大致完成。

我們只需要在前幾頁推導的範圍內窮舉並配合明文

進行Update keys 的逆運算找到原始的 keys

就可以用原始的 keys 對密文進行解密

Severity

掌握越多明文，所需複雜度越低

掉的速度很快！

bytes	key2 list entries
1	$2^{22} = 4194304$
2	3473408
3	2152448
4	1789183
5	1521084
10	798169
15	538746
20	409011
25	332606
30	283930
40	213751
50	174471
100	88248
200	43796
300	31088
500	16822
1000	7785
2000	5196
4000	3976
6000	3000
8000	1296
10000	1857
12000	243
12289	801

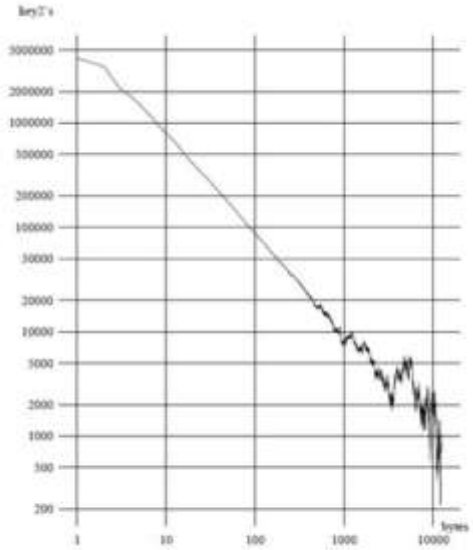


Fig. 1. Decrease in the number of key2 candidates using varying amounts of known plaintext. These results are for the PKZIP contest file and are fairly typical, though the entry 12000 is unusually low. (logarithmic scaling).

Conclusion

References

https://www.aloxaf.com/2019/04/zip_plaintext_attack/#%E5%8F%8

https://flandre-scarlet.moe/blog/1685/

https://link.springer.com/content/pdf/10.1007/3-540-60590-8_12.pdf


danke
thank you
merci
gracias
obrigado
bedankt
go raibh maith agat
teşekkür ederim
dziękuję
arigato
sukriya
tapadh leat
kop khun krap
ngiyabonga
dakujem
hvala
terima kasih
mauruuru
grazie
takk
taiku
спасибо
мерси
sagolun
mochchakkeram
谢謝
Баярлалаа
Adobe Stock | #41285353