A probabilistic attack against encrypted ZIP archives

This document presents a probabilistic attack against encrypted ZIP archives created with WinZip 8.1. An implementation is described and we demonstrate how the attack can be exploited. If one has access to the computer, the attack runs in a few minutes, otherwise in several days. The attack is based on the attack published by Stay [1].

1 Introduction

The encryption cipher described in the ZIP format specification [2] is byte-oriented and has a 96-bit internal state. The user-provided password is used to initialize the internal state, prior to start the encryption. The *compressed* data are prepended with 10 random bytes and then encrypted with the cipher. This means that what is referred as the *plaintext* is actually composed of unintelligible data. Each file in the archive can be encrypted/decrypted individually. The specification do not impose all files in an archive to be encrypted with the same password, however this is frequently the case.

1.1 Stay's attack

Assuming that all files in the archive are encrypted with the same password, if we can recover the first 10 bytes of plaintext of at least 5 files, the whole archive can be decrypted with the attack published by Stay [1].

1.2 Vulnerable programs

The programs vulnerable to this attack are WinZip 8, InfoZip and NetZip, that all contain the same flaw. These programs encrypt the 10 random bytes twice and reset the internal state between each encryption. Because the encryption uses XOR (denoted $\oplus$), the first random byte is not encrypted at all[1]. They also use a linear congruent generator, the function `rand()` of the libC.

[1] XOR is its own inverse: $a \oplus b \oplus b = a$

If the archive has 5 files, we know 5 random bytes. These known random bytes correspond to the $1^{st}, 11^{th}, 21^{st}, 31^{st}$ and 41^{st} call to `rand()` which suffices to recover the 32-bit seed of the generator[2]. Once the original seed is known, the 10 random bytes of each file can be recovered and the attack can be exploited.

1.3 WinZip 8.1

The pseudo random number generator has changed in WinZip 8.1 and a custom algorithm is used instead. The generator is initialized using two 32-bit words: *pid*, the process ID and, $ticks + time$, where *ticks* is the number of milliseconds since the computer started and *time* is the number of seconds since January, 1^{st} 1970. The generator is re-initialized immediately after each usage and the random bytes are not encrypted twice anymore.

2 Attack

Let's consider that the archive has k files. The i-th random byte of file j is denoted $r_{i,j}$ and the i-th encrypted byte of file j is denoted $e_{i,j}$. The cipher is byte-oriented and each byte $r_{i,j}$ is encrypted with $e_{i,j} = r_{i,j} \oplus key_{i,j}$, where $key_{i,j}$ is one byte derived from the 96-bit internal state. Assuming that all files are encrypted with the same password, $key_{1,j}$ is identical for all j in $(1, \ldots, k)$. The time taken to compress and encrypt file j is called t_j.

The recovery of the random bytes works as follows. First, m bits of *pid* and $ticks + time$ are guessed for a total of 2^m possibilities. This corresponds to the first initialization of the generator. The random bytes $r_{1,1}, \ldots, r_{10,1}$ are generated and $key_{1,1}$ is computed with $key_{1,1} = r_{1,1} \oplus e_{1,1}$.

The generator is re-initialized after each usage. This means that $r_{1,2}, \ldots, r_{10,2}$ can be generated using the same value for $ticks + time$, if we make the realistic assumption that the generation is immediate. $r_{1,2} \oplus key_{1,2}$ is then compared against $e_{1,2}$. About 2^{m-8} possibilities survive this filter.

d_1 values of t_1 are then guessed and represent $n = log_2(d_1)$ bits, leading to a total of 2^{m+n-8} possibilities. $ticks + time$ is updated accordingly and bytes $r_{1,3}, \ldots, r_{10,3}$ are generated. $r_{1,3} \oplus key_{1,3}$ is then compared against $e_{1,3}$, which filters again about 2^{-8} possibilities. There remain 2^{m+n-16} possibilities. If $n < 8$, the set of possibilities has been reduced.

This step is repeated $k - 1$ times, and each file is used to further reduce the set of possibilities. Only one candidate remains at the end, and the ten random bytes $r_{1,j}, \ldots, r_{10,j}$ have been recovered for each file j in $(1, ..., k)$, as well as the sequence $t_1, \ldots, t_{k-1}$ that corresponds to the time taken to compress and encrypt each file, except the last one.

[2]The seed can be recovered with a "brute-force" of the 2^{32} values. Only one seed will produce the expected sequence of random bytes.

2.1 Conplexity and convergence of the attack

The *complexity* of the attack represents the overall number of guesses made throughout the attack and can be computed with

$$\sum_{i=1}^{k-1} 2^{m-8i} D_i$$

where $D_i = d_1 d_2 \cdots d_i$.

The attack starts with a set of 2^m possibilities and *converge* then towards the final candidate. The convergence will succeed only if less than 2^8 guesses are made per file, and if there are enough files. More formally, the following relationship between k and $d_1, \ldots, d_{k-1}$ must hold to ensure the convergence of the attack:

$$2^m \prod_{i=1}^{k-1} d_i \leq 2^{8k}$$

2.2 Educated guesses

To reduce the complexity and improve the convergence of the attack, “educated guesses” must be done for *pid*, $ticks + time$ and t_i.

The time t_i can be estimated using s_i, the size of the file i. The bigger the file, the longer is takes to compress and encrypt. Empirical tests [3] showed that the number of required guesses grow lineraly with the size of the file. The hardware and the file type (image, text, video, etc.) has also an impact.

Empirical tests [3] showed also that the timer precision of most workstation running Windows XP was 10ms, so that only 1/10th of the values need actually to be tested.

The *pid* is also bounded, and only about 1000 values need to be considered.

The original value of *time* can be estimated using the date stored in the archive. If one has access to the workstation, the orignal value of *ticks* can also be estimated using the time stored in the archive. In this case, only a few thousands guesses are necessary for $ticks + time$. If the boot time is unknown, a maximal “uptime” of the workstation needs to be assumed. A maximal uptime of 8 hours means for instance that 2^{25} values of *ticks* will need to be tested.

2.3 Implementation

The attack was implemented using C and assembler.

The table below compares the attack time with the number of possibilities for $ticks + time$. The archive had 6 files of 50kb, 1000 values were tested for *pid*, and 3 values were tested for t_i with i in $(1, \ldots, 6)$. Reference hardware was a 1,5 GHz CPU.

Number of guesses for $ticks + time$	Maximal uptime (hrs)	Approximation of attack time (hrs)	Order
2^{32}	infinite	23'900	year
2^{28}	74	1493	month
2^{26}	18	373	weeks
2^{25}	9,3	186	days

Using the boot time to guess $ticks + time$ more accurately, that attack time is reduced by several orders of magnitude.

Number of guesses for $ticks + time$	Accuracy of $ticks + time$ (sec)	Approximation of attack time (sec)	Order
2^{16}	65	1307	minutes
2^{10}	1	20	seconds

3 Conclusion

We have presented and implemented a novel attack against encrypted ZIP archives created with WinZip 8.1. If the workstation is accessible, the boot time is known and the attack runs in several minutes with a high probability of success. If the original workstation is unknown, a maximal uptime of 8 hours can be assumed, and the attack runs in a couple of days in this case. The attack will always be probabilistic, but we have demonstrated that it is exploitable against archives containing small files.

This attack shows also that lessons are not always learned from the past. The new pseudo number generator of WinZip 8.1 has still weaknesses, because it fails to generate enough entropy. Instead of initializing the generator using temporal information, the WinZip authors could have used for instance the hash of the file itself.

4 References

[1] Michael Stay, *ZIP Attacks with Reduced Known Plaintext*, Fast Software Encryption, LNCS 2355, 125-134. Springer-Verlag, 2002.

[2] PKWARE Inc., *ZIP File Format Specification*, www.pkware.com, 2004.

[3] Wernli Erwann, *On the security of ZIP files*, Diploma thesis, 2004.