ZIP file encryption weaknesses

Krystian Matusiewicz
@kmatusie

Natalia Wochtman
@nwochtman

Intel Poland

Security PWNing 2017

Reading the specs...

WinZip 9.0 and WinZip 10.0 stored all AES-encrypted files using the AE-2 file format, which does not store the encrypted file's CRC.

Reading the specs...

WinZip 9.0 and WinZip 10.0 stored all AES-encrypted files using the AE-2 file format, which does not store the encrypted file's CRC.

WinZip 11 instead uses the AE-1 file format, which does store the CRC, for most files.

Reading the specs...

WinZip 9.0 and WinZip 10.0 stored all AES-encrypted files using the AE-2 file format, which does not store the encrypted file's CRC.

WinZip 11 instead uses the AE-1 file format, which does store the CRC, for most files.

Reading the specs...

WinZip 9.0 and WinZip 10.0 stored all AES-encrypted files using the AE-2 file format, which does not store the encrypted file's CRC.

WinZip 11 instead uses the AE-1 file format, which does store the CRC, for most files.

This provides an extra integrity check against the possibility of hardware or software errors that occur during the actual process of file compression/encryption or decryption/decompression. (...)

Reading the specs...

WinZip 9.0 and WinZip 10.0 stored all AES-encrypted files using the AE-2 file format, which does not store the encrypted file's CRC.

WinZip 11 instead uses the AE-1 file format, which does store the CRC, for most files.

This provides an extra integrity check against the possibility of hardware or software errors that occur during the actual process of file compression/encryption or decryption/decompression. (...)

Reading the specs...

Because for some very small files the CRC can be used to determine the exact contents of a file, regardless of the encryption method used, WinZip 11 continues to use the AE-2 file format, with no CRC stored, for files with an uncompressed size of less than 20 bytes.

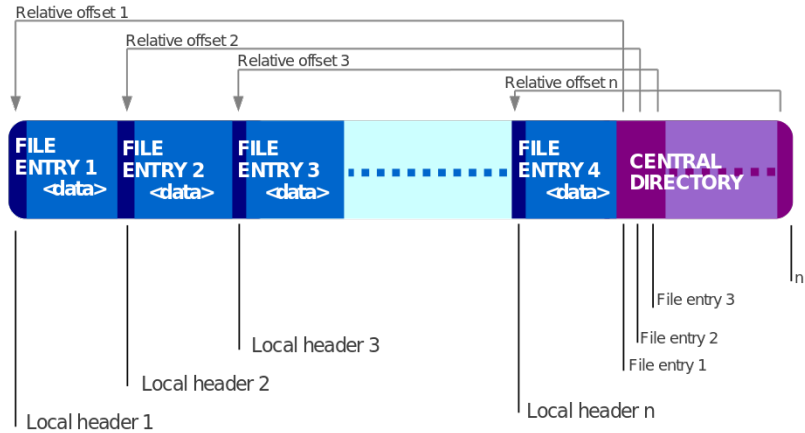
Reading the specs...

Because for some very small files the CRC can be used to determine the exact contents of a file, regardless of the encryption method used, WinZip 11 continues to use the AE-2 file format, with no CRC stored, for files with an uncompressed size of less than 20 bytes.

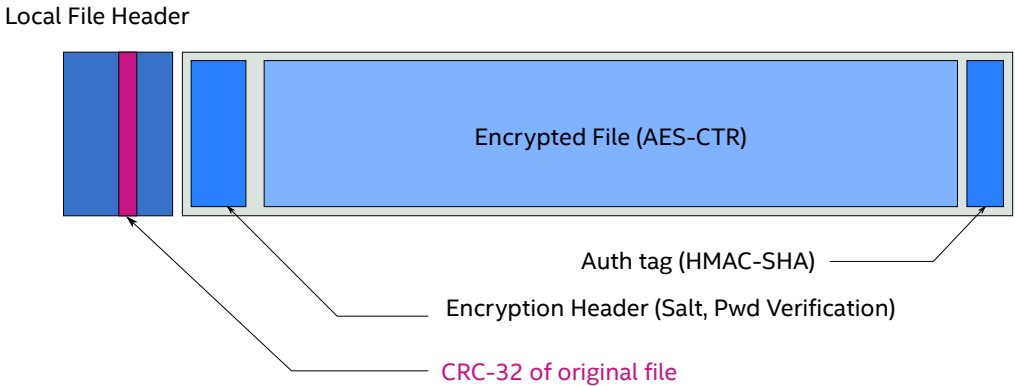
Reading the specs...

Because for some very small files the CRC can be used to determine the exact contents of a file, regardless of the encryption method used, WinZip 11 continues to use the AE-2 file format, with no CRC stored, for files with an uncompressed size of less than 20 bytes.

Structure of ZIP files



Structure of File Entry



Cyclic Redundancy Codes

Cyclic Redundancy Codes (in three easy steps)

Cyclic Redundancy Codes (in three easy steps)

Step 1: Take data* bits $d_0, d_1, d_2, \ldots, d_{n-1}$ as coefficients of a polynomial over the binary field

$$D(x) = d_0 + d_1 \cdot x + d_2 \cdot x^2 + \ldots + d_{n-1}x^{n-1}$$

Cyclic Redundancy Codes (in three easy steps)

Step 1: Take data* bits $d_0, d_1, d_2, \ldots, d_{n-1}$ as coefficients of a polynomial over the binary field

$$D(x) = d_0 + d_1 \cdot x + d_2 \cdot x^2 + \ldots + d_{n-1}x^{n-1}$$

Step 2: Compute the remainder $C(x)$ of a polynomial division of $D(x)$ by a fixed reduction polynomial $R(x) = r_0 + r_1 x + \cdots + x^u$

$$C(x) = D(x) \bmod R(x)$$

Cyclic Redundancy Codes (in three easy steps)

Step 1: Take data* bits $d_0, d_1, d_2, \ldots, d_{n-1}$ as coefficients of a polynomial over the binary field

$$D(x) = d_0 + d_1 \cdot x + d_2 \cdot x^2 + \ldots + d_{n-1}x^{n-1}$$

Step 2: Compute the remainder $C(x)$ of a polynomial division of $D(x)$ by a fixed reduction polynomial $R(x) = r_0 + r_1 x + \cdots + x^u$

$$C(x) = D(x) \text{ mod } R(x)$$

Step 3: Take coefficients* c_0, ..., c_{u-1} of polynomial $C(x)$ as the checksum bits.

Cyclic Redundancy Codes (in three easy steps)

Step 1: Take data* bits $d_0, d_1, d_2, \ldots, d_{n-1}$ as coefficients of a polynomial over the binary field

$$D(x) = d_0 + d_1 \cdot x + d_2 \cdot x^2 + \ldots + d_{n-1}x^{n-1}$$

Step 2: Compute the remainder $C(x)$ of a polynomial division of $D(x)$ by a fixed reduction polynomial $R(x) = r_0 + r_1 x + \cdots + x^u$

$$C(x) = D(x) \bmod R(x)$$

Step 3: Take coefficients* c_0, ..., c_{u-1} of polynomial $C(x)$ as the checksum bits.

* almost: needs data padding, initial and final xors

Polynomial modulo reduction

- Computing something mod $R(x)$ means $R(x) \equiv 0$,
- Over the binary field, $a + a = 0$.

Polynomial modulo reduction

- Computing something mod $R(x)$ means $R(x) \equiv 0$,
- Over the binary field, $a + a = 0$.

For $R(x) = r_0 + r_1 x + \cdots + x^u$, it means

$$x^u \equiv r_0 + r_1 x + \cdots + r_{u-1} x^{u-1} \tag{1}$$

Polynomial modulo reduction

- Computing something mod $R(x)$ means $R(x) \equiv 0$,
- Over the binary field, $a + a = 0$.

For $R(x) = r_0 + r_1 x + \cdots + x^u$, it means

$$x^u \equiv r_0 + r_1 x + \cdots + r_{u-1} x^{u-1} \tag{1}$$

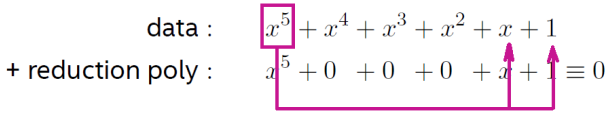
Consequence:

- Can iteratively reduce polynomial of any degree, replacing x^u according to (1)

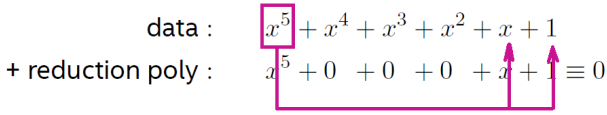
Modulo reduction: example

$$
\begin{aligned}
\text{data :} \quad & x^5 + x^4 + x^3 + x^2 + x + 1 \\
\text{+ reduction poly :} \quad & x^5 + 0 \ + 0 \ + 0 \ + x + 1 \equiv 0 \\
\hline
\text{result :} \quad & 0 \ + x^4 + x^3 + x^2 + 0 + 0
\end{aligned}
$$

Linearity of CRC



Linearity of CRC



Example:

$$c = \mathsf{CRC32}(x), \qquad |x| = 32$$

Bits c_i can be represented as an affine function of 32 input bits x_i:

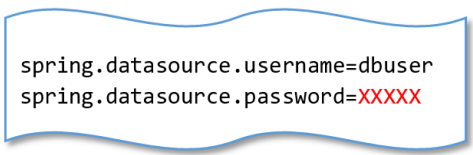
$$
\begin{aligned}
c_0 &= x_0 + x_1 + x_2 + x_3 + x_4 + x_6 + x_7 + x_8 + x_{16} + x_{20} + x_{22} + x_{23} + x_{26}, \\
c_1 &= x_1 + x_2 + x_3 + x_4 + x_5 + x_7 + x_8 + x_9 + x_{17} + x_{21} + x_{23} + x_{24} + x_{27}, \\
c_2 &= x_0 + x_2 + x_3 + x_4 + x_5 + x_6 + x_8 + x_9 + x_{10} + x_{18} + x_{22} + x_{24} + x_{25} + x_{28} + 1, \\
c_3 &= x_1 + x_3 + x_4 + x_5 + x_6 + x_7 + x_9 + x_{10} + x_{11} + x_{19} + x_{23} + x_{25} + x_{26} + x_{29} + 1, \\
&\vdots \\
c_{31} &= x_0 + x_1 + x_2 + x_3 + x_5 + x_6 + x_7 + x_{15} + x_{19} + x_{21} + x_{22} + x_{25} + x_{31}
\end{aligned}
$$

We like linearity

$$\begin{bmatrix} c_0 \\ c_1 \\ \vdots \\ c_{n-1} \end{bmatrix} = \begin{bmatrix} q_{0,0} & q_{0,1} & \cdots & \cdots & \cdots & q_{0,m-1} \\ q_{1,0} & q_{1,1} & \cdots & \cdots & \cdots & q_{1,m-1} \\ \vdots & \vdots & & & & \vdots \\ q_{n-1,0} & q_{n-1,1} & \cdots & \cdots & \cdots & q_{n-1,m-1} \end{bmatrix} \cdot \begin{bmatrix} d_0 \\ d_1 \\ \vdots \\ \vdots \\ \vdots \\ d_{m-1} \end{bmatrix} + \begin{bmatrix} b_0 \\ b_1 \\ \vdots \\ b_{n-1} \end{bmatrix}$$

Scenario 1

Extracting bits from mostly known file: basics



- We have 32 equations
- We can solve for unknown bits of the data
- Can recover up to 32 bits, depending on the rank of the matrix

Extracting bits from mostly known file: in general

What if the secret is longer?

- We get linear "check" relations among unknown bits

Extracting bits from mostly known file: in general

What if the secret is longer?

- We get linear "check" relations among unknown bits
- We can reduce the search space by up to 32 bits, i.e. recover n-bit value in 2^{n-d}, $d \lessapprox 32$

Extracting bits from mostly known file: in general

What if the secret is longer?

- We get linear "check" relations among unknown bits
- We can reduce the search space by up to 32 bits, i.e. recover n-bit value in 2^{n-d}, $d \lessapprox 32$

Extracting bits from mostly known file: in general

What if the secret is longer?

- We get linear "check" relations among unknown bits
- We can reduce the search space by up to 32 bits, i.e. recover n-bit value in 2^{n-d}, $d \lessapprox 32$

Example: If the only unknown part of the file is a password:

- we have a good chance of determining it uniquely if the entropy is $\leq$ 32 bits,

Extracting bits from mostly known file: in general

What if the secret is longer?

- We get linear "check" relations among unknown bits
- We can reduce the search space by up to 32 bits, i.e. recover n-bit value in 2^{n-d}, $d \lessapprox 32$

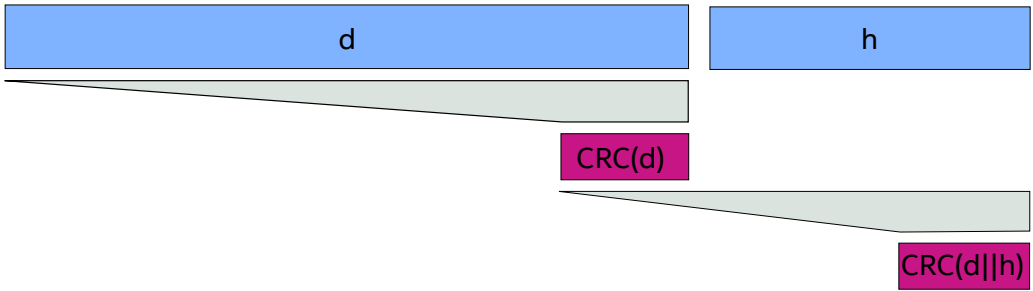
Example: If the only unknown part of the file is a password:

- we have a good chance of determining it uniquely if the entropy is ≤ 32 bits,
- we can sieve out vast majority of potential passwords from our wordlist.

Scenario 2

Incremental files

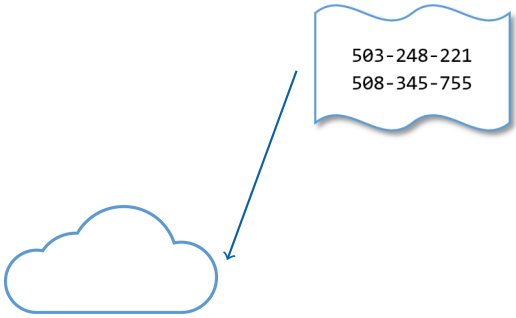
- CRC computation is “incremental”
- Easy to compute $c' = CRC(d||h)$ knowing $c = CRC(d)$ and h, without the knowledge of d



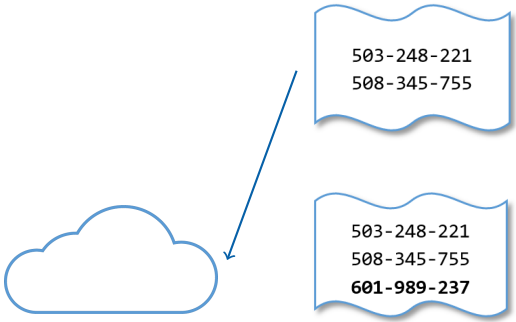
Recovering incremental files

Recovering incremental files

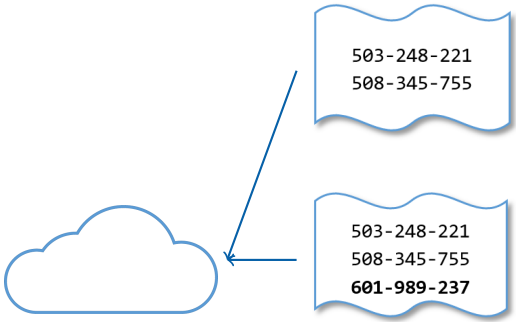
Recovering incremental files



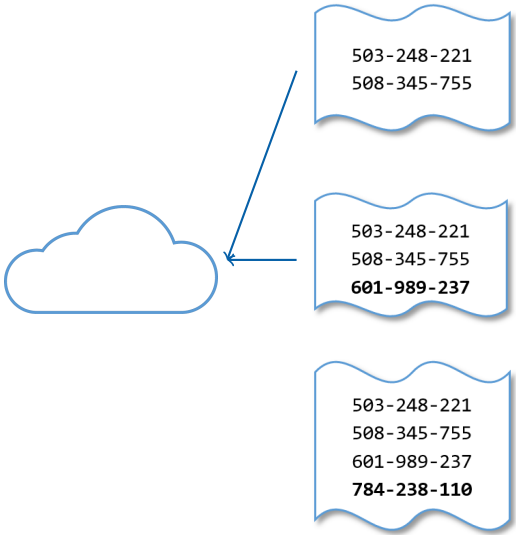
Recovering incremental files



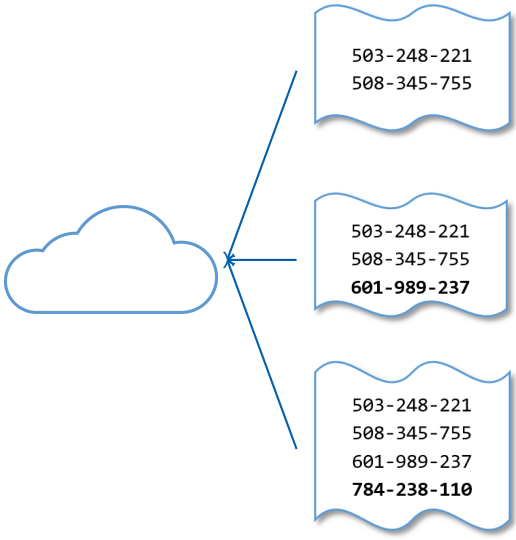
Recovering incremental files



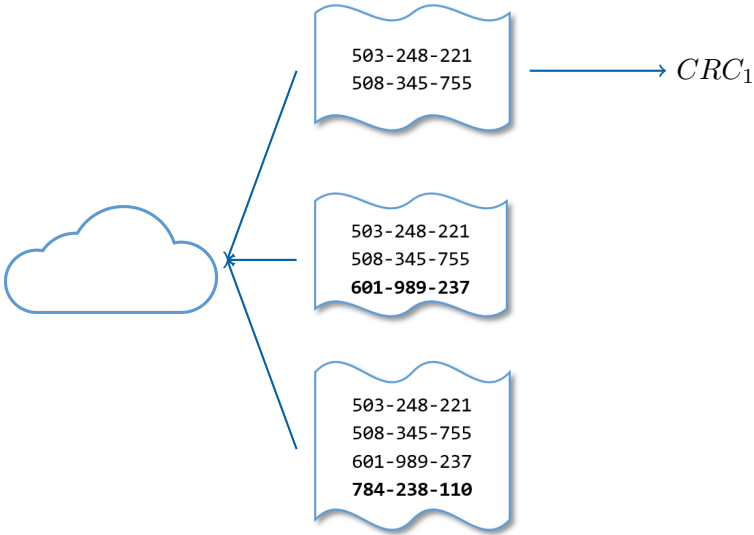
Recovering incremental files



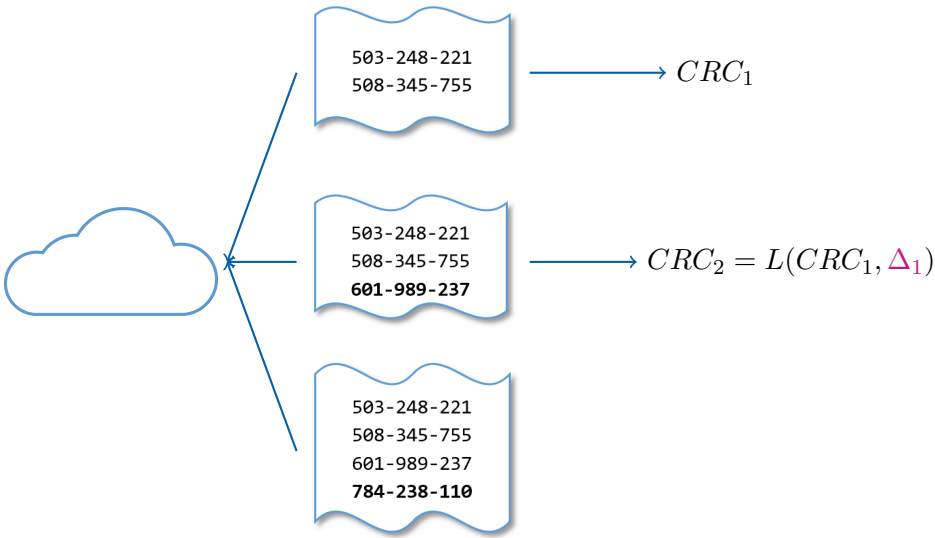
Recovering incremental files



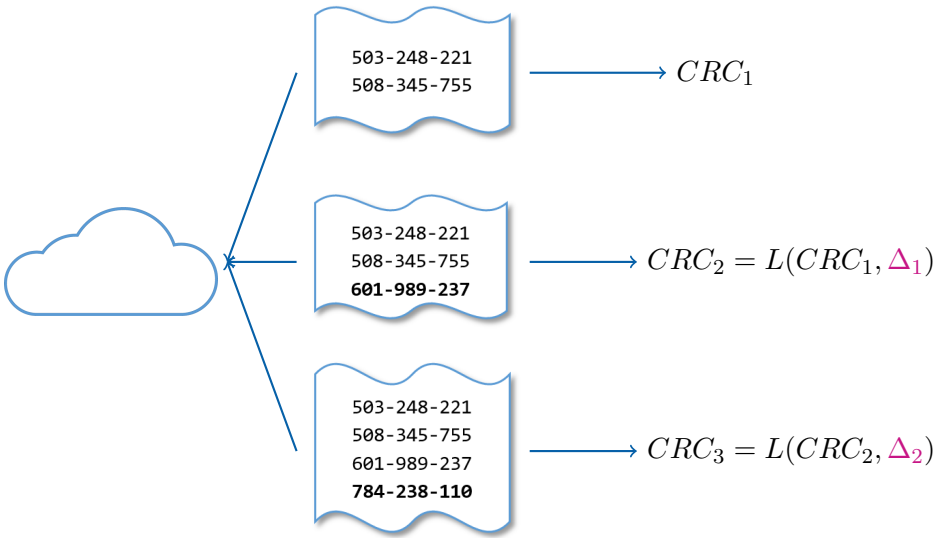
Recovering incremental files



Recovering incremental files



Recovering incremental files



Recovering incremental files

- If data increment Δ is small (≤ 32 bits of entropy), we can find $\Delta_i = L_{\text{inv}}(CRC_{i-1}, CRC_i)$
- Recover the increments Δ_i and reconstruct file contents

Scenario 3

Information tagging

- CRC values of unencrypted data can be extracted from encrypted ZIP files

Information tagging

- CRC values of unencrypted data can be extracted from encrypted ZIP files
- CRC values are $\approx$ uniformly distributed

Information tagging

- CRC values of unencrypted data can be extracted from encrypted ZIP files
- CRC values are $\approx$ uniformly distributed
- Probability of collision is 2^{-16} for CRC-32

Information tagging

- CRC values of unencrypted data can be extracted from encrypted ZIP files
- CRC values are $\approx$ uniformly distributed
- Probability of collision is 2^{-16} for CRC-32
- If your chance of being wrong is $1/65000$, would you get a warrant?

The Fix

- Easy to fix: don’t store CRC for encrypted files
- Reported our findings to WinZip (Corel), issue acknowledged
- “Fixed” in WinZip 22: registry setting to switch between AE-1 AE-2

Conclusions

- Always watch out for small details

Conclusions

- Always watch out for small details
- CRC codes are linear!

Conclusions

- Always watch out for small details
- CRC codes are linear!
- Linearity is always a friend of a cryptanalyst

Conclusions

- Always watch out for small details
- CRC codes are linear!
- Linearity is always a friend of a cryptanalyst
- Potential improvements: uncompressed/compressed size available, go beyond 32 bits

Thank you!

Backup

CRC-32 Source code

```
uint32_t crc32(unsigned char * buffer, size_t len)
{
  const uint32_t INITXOR = 0xFFFFFFFFU;
  const uint32_t FINALXOR = 0xFFFFFFFFU;
  const uint32_t CRCPOLY = 0xEDB88320U;
  uint32_t crcreg = INITXOR;
  // for each byte of the buffer
  for (size_t j = 0; j < len; ++j) {
    unsigned char b = buffer[j];
    // for each bit of the current byte b
    for (size_t i = 0; i < CHAR_BIT; ++i) {
      unsigned int lastBit = (crcreg ^ b) & 1;
      crcreg = crcreg >> 1;
      crcreg ^= lastBit ? CRCPOLY : 0;
      b >>= 1;
    }
  }
  return crcreg ^ FINALXOR;
}
```