Cryptography - Attack Plan

lrfk99

Version 2021.01.01

1 Attack Plan

My plan to crack the code involves exploiting the weakness of the Electronic Code Book (ECB) mode of operation, and the limited number of possible what3words addresses.

First of all we can establish for certain that the what3words address which was encrypted can be no more than 16 letters long (including '.' dividers). We know this because the ciphertext is 16 bytes long (32 hexadecimal characters) and using DES in ECB mode of operation does not itself add any padding or parity bits. It is possible that the plaintext was less than 16 letters long and padding was used in the conversion to hexadecimal, however I assumed that it was exactly 16 letters since there was no mention of padding in the assignment specification, and the 'encrypt.exe' program did not add any padding to the input.

Next, I researched into the what3words word selection criteria and found here that in English, the shortest words are four letters long. Knowing this means that the longest possible word used in the plaintext w3w address here was six letters. Since there are two '.' dividers in the 16 characters, meaning that the combined length of the three words is 14.

I then took a dictionary of 65,000 UK English words from here and filtered them down to words of length 4, 5, or 6. I also filtered out words which contained an apostrophe, because I read that part of the word selection criteria is that the words do not contain any punctuation. This left me with 14,657 possible words which could make up this 16-letter long address.

The next stage involved exploiting the weakness of ECB in this situation, since the key was 8 bytes long, yet the input and output were 16 bytes, we know that the same key was used for each half of the ciphertext and that the two halves are independent of each other. This is the biggest weakness of ECB. So, we can split the ciphertext in half and try to work out each half of the plaintext individually.

Using the list of 14,657 words, I then set about producing all of the possible first halves of the plaintext. Each half must contain one full word, a '.' divider, and part of another word. If I naively joined each word with every other word (with a '.' divider) then I would have produced 214,827,649 (14657^2) possibilities. However, I also took into consideration the fact that some combinations of certain word lengths are not possible, such as two 6-letter words, or a 5 letter and a 6-letter word, since this would make the overall w3w address length more than 16 letters. Using this additional criterion, as well as eliminating combinations which were duplicates when considering the first 8 letters only, I produced 7,109,081 possibilities for the first half (8 bytes) of the plaintext. This is an improvement over the naive approach by a factor of 30, and also a tiny fraction of the total number of w3w addresses ($57 * 10^{12}$). I also converted each of these strings to hexadecimal, the format required by the 'encrypt.exe' program.

The next task was to try each of these possibilities on the 'encrypt.exe' program to see which of them, when encrypted, produced '903408ec4d951acf', the first half of the ciphertext. Initially I tried sequentially testing each possibility, but I soon realised that there was a delay built into the executable of 0.5 seconds, making running in 7 million times infeasible. I then realised that I could concatenate the hexadecimal strings together and run them in batches, in order to speed up this process. After experimenting with different lengths of input I found that the maximum number I could test at once was 2047. This meant that I only needed to run the program a maximum of 3473 times, which was much more feasible. This is possible because each 16-character hexadecimal string (8 bytes) in encrypted independently and then output in the same order, meaning that I can take

the output, split it into 2047 lots of 16-character hexadecimal ciphertext strings, and match each one to the corresponding 16-character hexadecimal plaintext input string. All I would then have to do is check each 16-character ciphertext to see if it matched '903408ec4d951acf', and when I found a match, I would be able to find the plaintext which produced it.

Running this script on a Dell Precision 5530 Laptop with an Intel Core i7-8850H and 64GB RAM took less than an hour, with each of the 3473 batches taking around a second to concatenate, execute the command line program, and then check the output for matches. This gave me the first half of the plaintext in hexadecimal, which was '74696c652e62696c'. When converted into a UTF-8 string, this gave the first half of the w3w address, 'tile.bil'.

Having worked out the first half of the location, I then set about working out the second half. I knew that the second word of the w3w address began with 'bil', and that the sum of the lengths of the second and third words must be 10, since the first word was 4 letters long and there are also two '.' characters which all make up a 16 letter address. Using the same 14,657 word list with only 4, 5, or 6 letter words from before, I produced all of the possible combinations of second and third words (with a '.' separator). There were only 10 words from my list which began with 'bil', so the possibly second words were: 'bile','bilge','bilges','bilked','bill','billed','billet','billow','bills','billy'. When combining each of these words with every other word from my list, I ensured that their lengths summed to 10, and I eliminated combinations which were duplicates when considering the last 8 letters only. This produced a list of 41,361 possibilities for the second half of the plaintext. I also converted each of these to hexadecimal.

Using the same program as before, but this time checking for a match with the second half of the ciphertext ('aeb47ca88390c475'), to batch test 2047 possibilities at a time, took around 20 seconds when using the same machine as before. This gave me the second half of the plaintext in hexadecimal, which was '6c732e7072696e74'. When converted into a UTF-8 string, this gave the second half of the w3w address, 'ls.print'.

Putting the two halves together, the entire plaintext in hexadecimal is '74 69 6c 65 2e 62 69 6c 6c 73 2e 70 72 69 6e 74', which, in UTF-8, is 'tile.bills.print'. When entered into what3words.com, the location is Four Seasons Total Landscaping, 7346 Melrose Street, Philadelphia, Pennsylvania, United States. This is the place where Rudy Giuliani infamously held a press conference on 7 November 2020.