

CTF杂项之BubbleBabble加密算法

转载

[digupang7059](#) 于 2019-03-28 17:44:00 发布 835 收藏 1

文章标签: [python](#) [运维](#)

原文链接: <http://www.cnblogs.com/mke2fs/p/10616588.html>

版权

这题很坑, 刚开始我拿到就分析不出来了 (/无奈), 关键是不知是什么加密算法, 后来看题目描述的bubble, 猜测是bubble

这种算法 (听都没听说过。。。。)

上图

```
flag.enc
Offset(h) 00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F Decoded text
00000000 78 69 6E 69 6B 2D 73 61 6D 61 6B 2D 6C 75 76 61 xlinik-samak-luvag
00000010 67 2D 68 75 74 61 66 2D 66 79 73 69 6C 2D 6E 6F g-hutaf-fysil-notok
00000020 74 6F 6B 2D 6D 65 70 65 6B 2D 76 61 6E 79 68 2D tok-mepek-vanyh-
00000030 7A 69 70 65 66 2D 68 69 6C 6F 6B 2D 64 65 74 6F zipef-hilok-detok
00000040 6B 2D 64 61 6D 69 66 2D 63 75 73 6F 6C 2D 66 65 k-damif-cusol-fe
00000050 7A 79 78 zyx
```

这串编码

```
xlinik-samak-luvag-hutaf-fysil-notok-mepek-vanyh-zipef-hilok-detok-damif-cusol-fezyx
```

百度也没找到哪里有, 后面看到有大佬写的writeup, 于是就借鉴一下下, 233

这里是这种加密算法的解释 (为了方便大家, 加密算法我复制到文末), 链接传送: http://wiki.yak.net/589/Bubble_Babble_Encoding.txt

附上解题代码, 楼主用的python3.7, 另外bubblepy这个库需要导入一下, 网址附上 <https://pypi.python.org/pypi/bubblepy/>

```
from bubblepy import BubbleBabble
#导入包bubblepy
str='xlinik-samak-luvag-hutaf-fysil-notok-mepek-vanyh-zipef-hilok-detok-damif-cusol-fezyx'
#str是待解密字符
Str=BubbleBabble()
print(Str.decode(str))
```

最后, get Flag

```
file Edit View Navigate Code Refactor Run Tools VCS Window Help
Spider bubble_babble_Decode.py
bubble_babble_Decode.py x
1 from bubblepy import BubbleBabble
2 #导入包
3 str='xinik-samak-luvag-hutaf-fysil-notok-mepek-vanyh-zipef-hilok-detok-damif-cusol-fezyx'
4 #str是待解密字符
5 Str=BubbleBabble()
6 print(Str.decode(str))
7
Run: bubble_babble_Decode x
E:\Users\MKe2fs\PycharmProjects\Spider\venv\Scripts\python.exe E:/Users/MKe2fs/PycharmProjects/Spic
b'flag{Ev3ry7hing_i5_bubb13s}'
```

这里是BubbleBabble加密算法

```
authors==Huima
status==Experimental
title==The Bubble Babble Binary Data Encoding
number==Internet Draft
date==April 2000
Network Working Group                                Antti Huima
Internet Draft                                     SSH Communications Security
draft-huima-babble-01.txt                            April 2000
```

The Bubble Babble Binary Data Encoding

Status of this Memo

This memo provides information for the Internet community. It does not specify an Internet standard of any kind. Distribution of this memo is unlimited.

Copyright Notice

Copyright (C) The Internet Society (2000). All Rights Reserved.

Abstract

This document describes a new encoding method for binary data that is intended to be used in conjunction with fingerprints of security-critical data.

1. Introduction

Hash values of certificates and public keys, known as fingerprints or thumbprints, are commonly used for verifying that a received security-critical datum has been received correctly. Fingerprints

are binary data and typically encoded as series of hexadecimal digits. However, long strings hexadecimal digits are difficult for comprehend and cumbersome to translate reliably e.g. over phone.

The Bubble Babble Encoding encodes arbitrary binary data into pseudowords that are more natural to humans and that can be pronounced relatively easily. The encoding consumes asymptotically the same amount of space as an encoding of the form

HH HH HH HH ...

where 'H' is a hexadecimal digit, i.e. carries 16 bits in six characters. However, the Bubble Babble Encoding includes a checksumming method that can sometimes detect invalid encodings. The method does not increase the length of the encoded data.

2. Encoding

Below, $\lfloor X \rfloor$ denotes the largest integer not greater than X.

Let the data to be encoded be $D[1] \dots D[K]$ where K is the length of the data in bytes; every $D[i]$ is an integer from 0 to $2^8 - 1$. First define the checksum series $C[1] \dots C[\lfloor K/2 \rfloor]$ where

$$C[1] = 1$$

$$C[n] = (C[n - 1] * 5 + (D[n * 2 - 3] * 7 + D[n * 2 - 2])) \bmod 36$$

The data is then transformed into $\lfloor K/2 \rfloor$ 'tuples' $T[1] \dots T[\lfloor K/2 \rfloor]$ and one 'partial tuple' P so that

$$T[i] = \langle a, b, c, d, e \rangle$$

where

$$\begin{aligned} a &= (((D[i * 2 - 3] \gg 6) \& 3) + C[i]) \bmod 6 \\ b &= (D[i * 2 - 3] \gg 2) \& 15 \\ c &= (((D[i * 2 - 3]) \& 3) + \lfloor C[i] / 6 \rfloor) \bmod 6 \\ d &= (D[i * 2 - 2] \gg 4) \& 15; \text{ and} \\ e &= (D[i * 2 - 3]) \& 15. \end{aligned}$$

The partial tuple P is

$$P = \langle a, b, c \rangle$$

where if K is even then

$$\begin{aligned} a &= (C[i]) \bmod 6 \\ b &= 16 \\ c &= \lfloor C[i] / 6 \rfloor \end{aligned}$$

but if it is odd then

$$\begin{aligned} a &= (((D[K] \gg 6) \& 3) + C[i]) \bmod 6 \\ b &= (D[K] \gg 2) \& 15 \\ c &= (((D[K]) \& 3) + \lfloor C[i] / 6 \rfloor) \bmod 6 \end{aligned}$$

The 'vowel table' V maps integers between 0 and 5 to vowels as

$$0 - a$$

- 1 - e
- 2 - i
- 3 - o
- 4 - u
- 5 - y

and the 'consonant table' C maps integers between 0 and 16 to consonants as

- 0 - b
- 1 - c
- 2 - d
- 3 - f
- 4 - g
- 5 - h
- 6 - k
- 7 - l
- 8 - m
- 9 - n
- 10 - p
- 11 - r
- 12 - s
- 13 - t
- 14 - v
- 15 - z
- 16 - x

The encoding $E(T)$ of a tuple $T = \langle a, b, c, d, e \rangle$ is then the string

$V[a] C[b] V[c] C[d] \text{'-'} C[e]$

where there are five characters, and '-' is the literal hyphen.

The encoding $E(P)$ of a partial tuple $P = \langle a, b, c \rangle$ is the three-character string

$V[a] C[b] V[c]$.

Finally, the encoding of the whole input data D is obtained as

$\text{'x'} E(T[1]) E(T[2]) \dots E(T[\lfloor K/2 \rfloor]) E(P) \text{'x'}$

where 'x's are literal characters.

3. Decoding

Decoding is obviously the process of encoding reversed.

To check the checksums, when a tuple $\langle a, b, c, d, e \rangle$ or partial tuple $\langle a, b, c \rangle$ has been recovered from the encoded string, an implementation should check that $((a - C[i]) \bmod 6) < 4$ and that $((c - C[i]) \bmod 6) < 4$. Otherwise the encoded string is not a valid encoding of any data and should be rejected.

4. Checksum Strength

Every vowel in an encoded string carries 0.58 bits redundancy; thus the length of the 'checksum' in the encoding of an input string containing K bytes is $0.58 * K$ bits.

5. Test Vectors

ASCII Input	Encoding

`' (empty string)`	`xexax`
`1234567890`	`xesef-disof-gytuf-katof-movif-baxux`
`Pineapple`	`xigak-nyryk-humil-bosek-sonax`

6. Author's Address

Antti Huima
SSH Communications Security, Ltd.
[XXX]

7. Full Copyright Statement

Copyright (C) The Internet Society (2000). All Rights Reserved.

This document and translations of it may be copied and furnished to others, and derivative works that comment on or otherwise explain it or assist in its implementation may be prepared, copied, published and distributed, in whole or in part, without restriction of any kind, provided that the above copyright notice and this paragraph are included on all such copies and derivative works. However, this document itself may not be modified in any way, such as by removing the copyright notice or references to the Internet Society or other Internet organizations, except as needed for the purpose of developing Internet standards in which case the procedures for copyrights defined in the Internet Standards process must be followed, or as required to translate it into languages other than English.

The limited permissions granted above are perpetual and will not be revoked by the Internet Society or its successors or assigns.

This document and the information contained herein is provided on an "AS IS" basis and THE INTERNET SOCIETY AND THE INTERNET ENGINEERING TASK FORCE DISCLAIMS ALL WARRANTIES, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO ANY WARRANTY THAT THE USE OF THE INFORMATION HEREIN WILL NOT INFRINGE ANY RIGHTS OR ANY IMPLIED WARRANTIES OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE.