编码分析

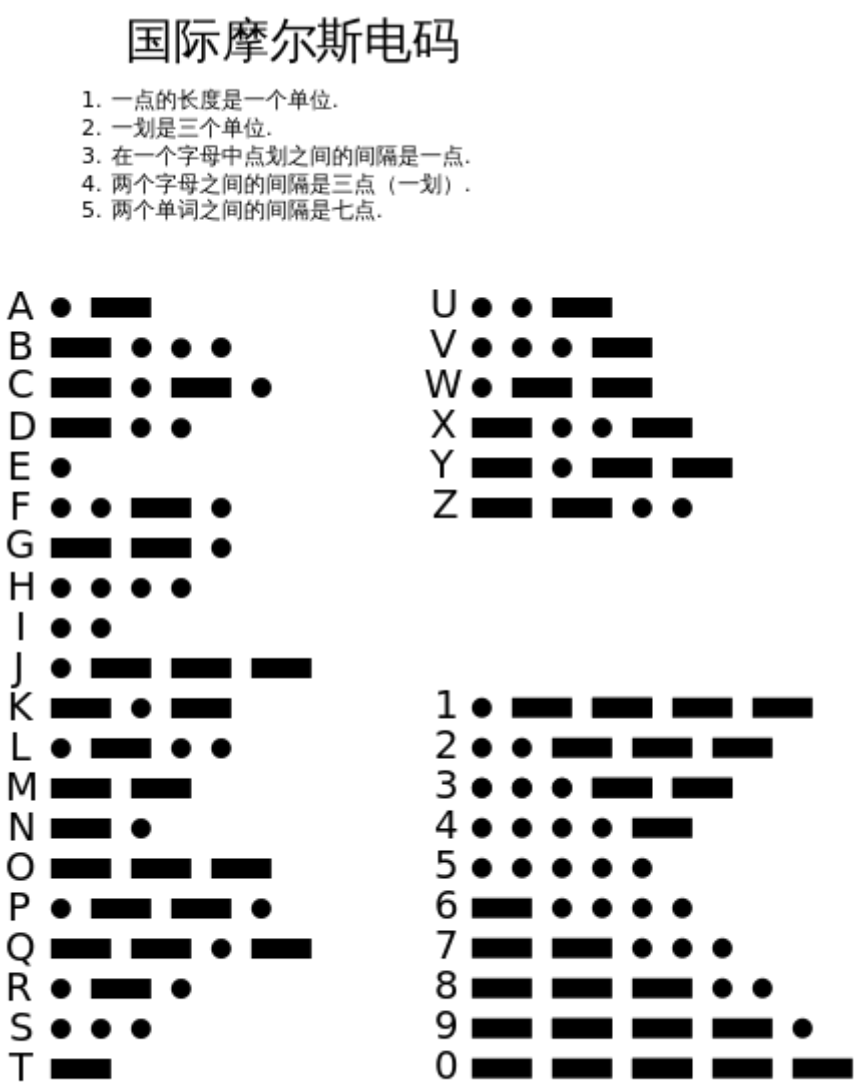
通信领域常用编码

电话拨号编码

1-9 分别使用 1-9 个脉冲，0 则表示使用 10 个脉冲。

Morse 编码

参见 摩尔斯编码 - 维基百科，对应表如下



特点

- 只有 `.` 和 `-`；
- 最多 6 位；
- 也可以使用 `01` 串表示。

工具

- CTF在线工具：莫尔斯电码

> 题目 JarvisOJ - BASIC题 `-.-  字符串`

敲击码

敲击码（Tap code）是一种以非常简单的方式对文本信息进行编码的方法。因该编码对信息通过使用一系列的点击声音来编码而命名，敲击码是基于 5 ×5 方格波利比奥斯方阵来实现的，不同点是是用 K 字母被整合到 C 中。

	A	B	C	D	E	F
1	Tap Code	1	2	3	4	5
2	1	A	B	C/K	D	E
3	2	F	G	H	I	J
4	3	L	M	N	O	P
5	4	Q	R	S	T	U
6	5	V	W	X	Y	Z

源文本	F	O	X
位置	2,1	3,4	5,3
敲击码	.. .	... .	....

曼彻斯特编码

- 曼彻斯特编码 - 维基百科

格雷编码

- 格雷码 - 维基百科

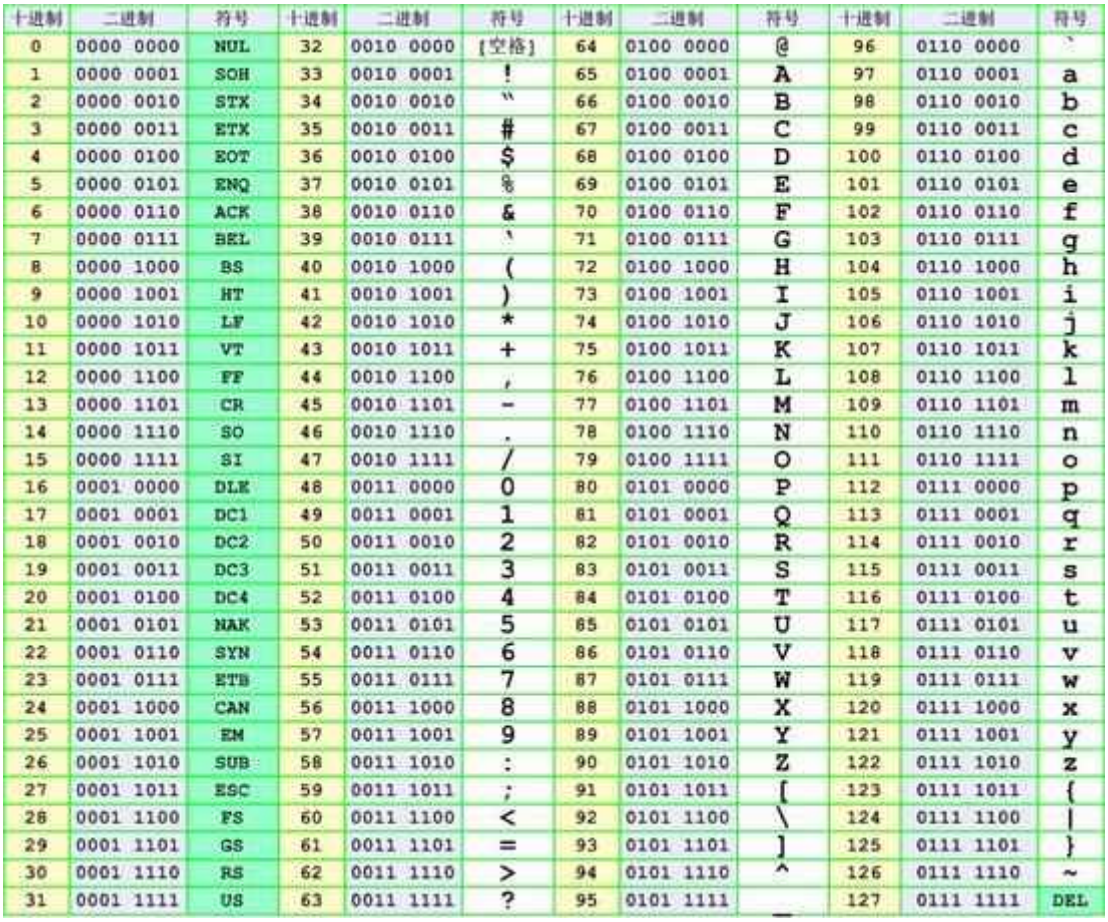
计算机相关的编码

本节介绍一些计算机相关的编码。

字母表编码

- `A~Z/a~z` 对应 `1~26` 或者 `0~25`

ASCII 编码



十进制	二进制	符号	十进制	二进制	符号	十进制	二进制	符号	十进制	二进制	符号
0	0000 0000	NUL	32	0010 0000	[空格]	64	0100 0000	@	96	0110 0000	`
1	0000 0001	SOH	33	0010 0001	!	65	0100 0001	A	97	0110 0001	a
2	0000 0010	STX	34	0010 0010	"	66	0100 0010	B	98	0110 0010	b
3	0000 0011	ETX	35	0010 0011	#	67	0100 0011	C	99	0110 0011	c
4	0000 0100	EOT	36	0010 0100	$	68	0100 0100	D	100	0110 0100	d
5	0000 0101	ENQ	37	0010 0101	%	69	0100 0101	E	101	0110 0101	e
6	0000 0110	ACK	38	0010 0110	&	70	0100 0110	F	102	0110 0110	f
7	0000 0111	BEL	39	0010 0111	'	71	0100 0111	G	103	0110 0111	g
8	0000 1000	BS	40	0010 1000	(	72	0100 1000	H	104	0110 1000	h
9	0000 1001	HT	41	0010 1001	)	73	0100 1001	I	105	0110 1001	i
10	0000 1010	LF	42	0010 1010	*	74	0100 1010	J	106	0110 1010	j
11	0000 1011	VT	43	0010 1011	+	75	0100 1011	K	107	0110 1011	k
12	0000 1100	FF	44	0010 1100	,	76	0100 1100	L	108	0110 1100	l
13	0000 1101	CR	45	0010 1101	-	77	0100 1101	M	109	0110 1101	m
14	0000 1110	SO	46	0010 1110	.	78	0100 1110	N	110	0110 1110	n
15	0000 1111	SI	47	0010 1111	/	79	0100 1111	O	111	0110 1111	o
16	0001 0000	DLE	48	0011 0000	0	80	0101 0000	P	112	0111 0000	p
17	0001 0001	DC1	49	0011 0001	1	81	0101 0001	Q	113	0111 0001	q
18	0001 0010	DC2	50	0011 0010	2	82	0101 0010	R	114	0111 0010	r
19	0001 0011	DC3	51	0011 0011	3	83	0101 0011	S	115	0111 0011	s
20	0001 0100	DC4	52	0011 0100	4	84	0101 0100	T	116	0111 0100	t
21	0001 0101	NAK	53	0011 0101	5	85	0101 0101	U	117	0111 0101	u
22	0001 0110	SYN	54	0011 0110	6	86	0101 0110	V	118	0111 0110	v
23	0001 0111	ETB	55	0011 0111	7	87	0101 0111	W	119	0111 0111	w
24	0001 1000	CAN	56	0011 1000	8	88	0101 1000	X	120	0111 1000	x
25	0001 1001	EM	57	0011 1001	9	89	0101 1001	Y	121	0111 1001	y
26	0001 1010	SUB	58	0011 1010	:	90	0101 1010	Z	122	0111 1010	z
27	0001 1011	ESC	59	0011 1011	;	91	0101 1011	[	123	0111 1011	{
28	0001 1100	FS	60	0011 1100	<	92	0101 1100	\	124	0111 1100	|
29	0001 1101	GS	61	0011 1101	=	93	0101 1101	]	125	0111 1101	}
30	0001 1110	RS	62	0011 1110	>	94	0101 1110	^	126	0111 1110	~
31	0001 1111	US	63	0011 1111	?	95	0101 1111	_	127	0111 1111	DEL

特点

我们一般使用的 ascii 编码的时候采用的都是可见字符，而且主要是如下字符

- `0~9`：`49~57`
- `A~Z`：`65~90`
- `a~z`：`97~122`

二进制编码，将 ascii 码对应的数字换成二进制表示形式。

- 只有 0 和 1
- 不大于 8 位，一般 7 位也可以，因为可见字符到 127 `(0 ~ 2^7-1)`。
- 其实是另一种 ascii 编码。

十六进制编码，将 ascii 码对应的数字换成十六进制表示形式。

- `A~Z`:`41~5A`
- `a~z`:`61~7A`

工具

- CTF在线工具：进制转换

> 题目 2018 DEFCON Quals ghettohackers: Throwback

题目描述如下

`Anyo!e!howouldsacrificepo!icyforexecu!!onspeedthink!securityisacomm!ditytop!urintoasy!tem!`

第一直觉应该是我们去补全这些叹号对应的内容，从而得到 flag，但是补全后并不行，那么我们可以把源字符串按照 ! 分割，然后字符串长度 1 对应字母 a，长度 2 对应字母 b，以此类推

Go

```
ori = 'Anyo!e!howouldsacrificepo!icyforexecu!!onspeedthink!securityisacomm!ditytop!urintoasy!tem!'
sp = ori.split('!')
print repr(''.join(chr(97 + len(s) - 1) for s in sp))
```

进而可以得到，这里同时需要假设 0 个字符为空格。因为这正好使得原文可读。

`dark logic`

> 题目 JarvisOJ - BASIC题 `德军的密码`

Base 编码

`base xx` 中的 `xx` 表示的是采用多少个字符进行编码，比如说 base64 就是采用以下 64 个字符编码，由于 2 的 6 次方等于 64，所以每 6 个比特为一个单元，对应某个可打印字符。3 个字节就有 24 个比特，对应于 4 个 Base64 单元，即 3 个字节需要用 4 个可打印字符来表示。它可用来作为电子邮件的传输编码。在 Base64 中的可打印字符包括字母 `A-Z`、`a-z`、数字 `0-9`，这样共有 62 个字符，此外两个可打印符号在不同的系统中而不同。

	A	B	C	D	E	F	G	H
1	Binary	Char	Binary	Char	Binary	Char	Binary	Char
2	0	A	10000	Q	100000	g	110000	w
3	1	B	10001	R	100001	h	110001	x
4	10	C	10010	S	100010	i	110010	y
5	11	D	10011	T	100011	j	110011	z
6	100	E	10100	U	100100	k	110100	0
7	101	F	10101	V	100101	l	110101	1
8	110	G	10110	W	100110	m	110110	2
9	111	H	10111	X	100111	n	110111	3
10	1000	I	11000	Y	101000	o	111000	4
11	1001	J	11001	Z	101001	p	111001	5
12	1010	K	11010	a	101010	q	111010	6
13	1011	L	11011	b	101011	r	111011	7
14	1100	M	11100	c	101100	s	111100	8
15	1101	N	11101	d	101101	t	111101	9
16	1110	O	11110	e	101110	u	111110	+
17	1111	P	11111	f	101111	v	111111	/
18	Padding	=						

具体介绍参见 Base64 - 维基百科。

以 `Man` → `TWFu` 为例：

文本	M								a								n							
ASCII编码	77								97								110							
二进制位	0	1	0	0	1	1	0	1	0	1	1	0	0	0	0	1	0	1	1	0	1	1	1	0
索引	19								22								5							
Base64编码	T								W								F							
索引	46																							
Base64编码	u																							

如果要编码的字节数不能被 3 整除，最后会多出 1 个或 2 个字节，那么可以使用下面的方法进行处理：先使用 0 值在末尾补足，使其能够被 3 整除，然后再进行 base64 的编码。在编码后的 base64 文本后加上一个或两个 = 号，代表补足的字节数。也就是说，当最后剩余一个八位字节（一个 byte）时，最后一个 6 位的 base64 字节块有四位是 0 值，最后附加上两个等号；如果最后剩余两个八位字节（2 个 byte）时，最后一个 6 位的 base 字节块有两位是 0 值，最后附加一个等号。参考下表：

文本（1 Byte）	A																							
二进制位	0	1	0	0	0	0	0	1																
二进制位（补0）	0	1	0	0	0	0	0	1	0	0	0	0												
Base64编码	Q						Q																	
文本（2 Byte）	B								C															
二进制位	0	1	0	0	0	0	1	0	0	1	0	0	0	0	1	1			x	x	x	x	x	x
二进制位（补0）	0	1	0	0	0	0	1	0	0	1	0	0	0	0	1	1	0	0	x	x	x	x	x	x
Base64编码	Q						k						M											

由于解码时补位的 0 并不参与运算，可以在该处隐藏信息。

与 base64 类似，base32 使用 32 个可见字符进行编码，2 的 5 次方为 32，所以每 5 bit 为 1 个分组。5 字节为 40 bit，对应于 8 个 base32 分组，即 5 个字节用 8 个 base32 中字符来表示。但如果不足 5 个字节，则会先对第一个不足 5 bit 的分组用 0 补足了 5 bit ，对后面剩余分组全部使用 “=” 填充，直到补满 5 个字节。由此可知，base32 最多只有 6 个等号出现。例如：

文本 (1 Byte)	A																																							
二进制位	0	1	0	0	0	0	0	1																																
二进制位 (补0)	0	1	0	0	0	0	0	1	0	0																														
Base32编码	I					E					=					=					=					=					=					=				

特点

- base64 结尾可能会有 `=` 号，但最多有 2 个
- base32 结尾可能会有 `=` 号，但最多有 6 个
- 根据 base 的不同，字符集会有所限制
- 有可能需要自己加等号（十六进制 `0x3d`）
- 更多内容请参见 RFC4648

工具

- CTF在线工具：BaseXX编码
- Python库 base64

Base64隐写

从Base64编码的原理我们很自然推出Base64解码的过程：

- 丢掉末尾的所有‘=’；
- 每个字符**查表**后，转换为对应的6位二进制，得到一串二进制；
- 从二进制串的头部开始，每次取8位转换为对应的ASCII字符，如果不足8位则丢弃。

在解码的第3步中，会有部分数据被丢弃（但不会影响解码结果），这些数据正是我们在编码过程中补的0。也就是说，如果我们在编码过程中不全用0填充，而是用其他的数据填充，仍然可以正常编码解码，因此这些位置可以用于隐写。

解开隐写的方法就是通过**查表**将这些不影响解码结果的位提取出来组成二进制串，然后转换成ASCII字符串。 下面是一个Base64隐写的题目。

> 题目：从文本中提取隐写信息 点此下载 。

以最后三行的文本为例，展示提取隐写信息的过程：

Bash

```
IEEtWn== #编码时补了4位，取'n'(100111)的低4位
IC3=     #编码时补了2位，取'3'(110111)的低2位
IGFuZCBfIGFzIGFkZGl0aW9uYWwgY2hhcmFjdGVycy5=
#编码时补了2位，取'5'(111001)的低2位。
#拼接成二进制串：01111101(ASCII码'}')，这意味者每处理4个'='就能拿到一个隐写字符。
```

使用脚本读取文件中的隐写信息。

Python

```
import base64

def deStego(stegoFile):
    b64table = "ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789+/"
    with open(stegoFile,'r') as stegoText:
        message = ""
        for line in stegoText:
            try:
                text = line[line.index("=") - 1:-1]
                message += "".join([ bin( 0 if i == '=' else b64table.find(i))[2:].zfill(6) for i in text])[2 if text.count('=') ==2 else 4:6]
            except:
                pass
    return "".join([chr(int(message[i:i+8],2)) for i in range(0,len(message),8)])

print(deStego("data.txt"))
```

输出:`flag{BASE64_i5_amaz1ng}`

霍夫曼编码

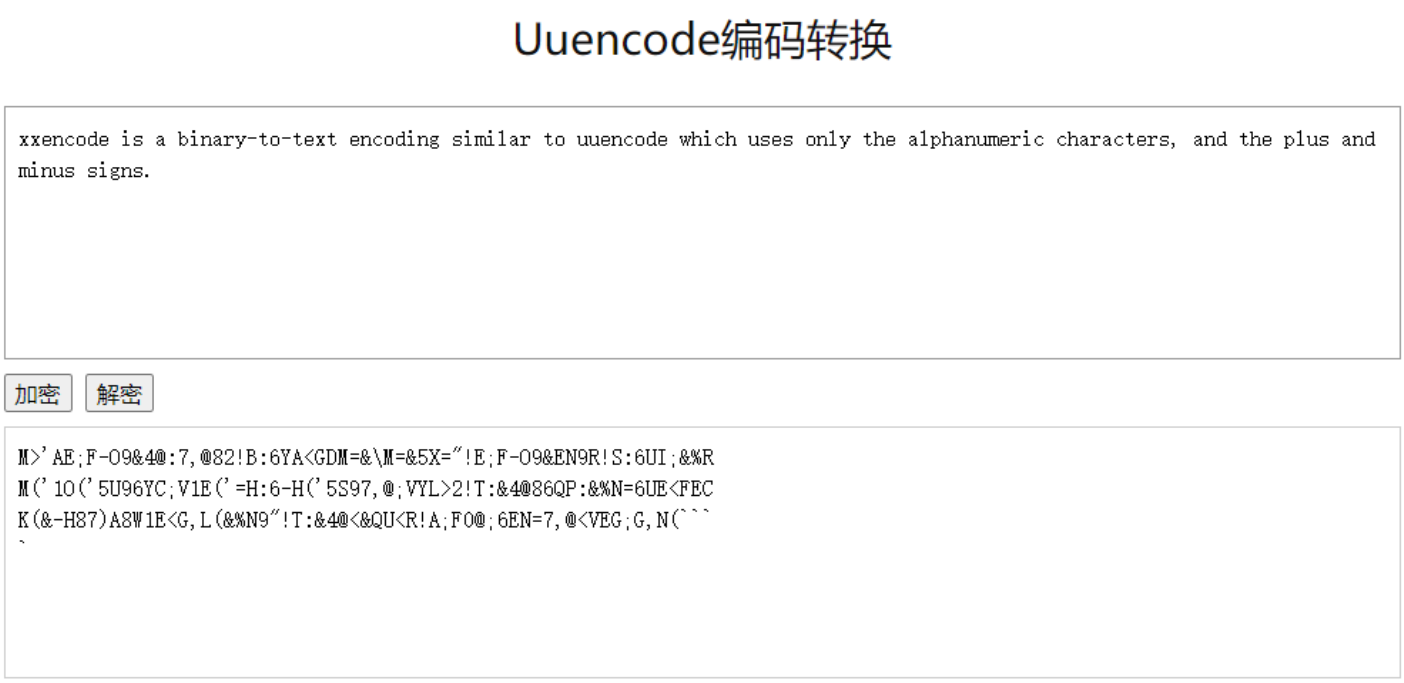
参见 霍夫曼编码 - 维基百科。

UUencode

Uuencode是二进制信息和文字信息之间的转换编码，也就是机器和人眼识读的转换。Uuencode编码方案常见于电子邮件信息的传输，目前已被多用途互联网邮件扩展（MIME）大量取代。

Uuencode将输入文字以每三个字节为单位进行编码，如此重复进行。如果最后剩下的文字少于三个字节，不够的部份用零补齐。这三个字节共有24个Bit，以6-bit为单位分为4个群组，每个群组以十进制来表示所出现的数值只会落在0到63之间。将每个数加上32，所产生的结果刚好落在ASCII字符集中可打印字符（32-空白...95-底线）的范围之中。

Uuencode编码每60个将输出为独立的一行（相当于45个输入字节），每行的开头会加上长度字符，除了最后一行之外，长度字符都应该是“M”这个ASCII字符（77=32+45），最后一行的长度字符为32+剩下的字节数目这个ASCII字符。



工具

- UUencode编码转换

XXencode

XXencode将输入文本以每三个字节为单位进行编码。如果最后剩下的资料少于三个字节，不够的部份用零补齐。这三个字节共有24个Bit，以6bit为单位分为4个组，每个组以十进制来表示所出现的数值只会落在0到63之间。以所对应值的位置字符代替。它所选择的可打印字符是：

`+-0123456789ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz`，一共64个字符。跟base64打印字符相比，就是uuencode多一个“-” 字符，少一个”/” 字符。 但是，它里面字符顺序与Base64完全不一样。与UUencode比较，这里面所选择字符，都是常见字符，没有特殊字符。

每60个编码输出（相当于45个输入字节）将输出为独立的一行，每行的开头会加上**长度字符**，除了最后一行之外，长度字符都应该是“h”这个字符（45，刚好是64字符中，第45位’ h’ 字符），最后一行的长度字符为剩下的字节数目在64字符中位置所代表字符。

具体信息参见xxencoding - 维基百科

特点

- 包括只有数字，大小写字母和 `+-` 字符

工具

- XXencode编码转换

URL 编码

参见 URL 编码 - 维基百科 和 关于URL编码。

特点

- 大量的百分号

工具

- CTF在线工具：URL编码

Unicode 编码

参见 Unicode - 维基百科 和 字符编码笔记：ASCII，Unicode 和 UTF-8。

注意，它有四种表现形式。

以文本 `The` 的编码为例：

表现形式	格式	转换结果
`UTF-16`	`\uXXXX`	`\u0054\u0068\u0065`
`UTF-32`	`\uXXXXXXXX`	`\U00000054\U00000068\U00000065`
`Unicode`	`&#DDDDD;`	`The`
`Hex`	`&#xXXXX;`	`The`

工具

- CTF在线工具：ASCII编码转换

HTML 实体编码

什么是HTML实体?

> 以“&”开头和以“;”结尾的字符串。
>
> 使用实体代替解释为HTML代码的保留字符（&，<，>，''），不可见字符（如不间断空格）和无法从键盘输入的字符（如©）。

HTML实体 `Foo © bar 𝌆 baz ☃ qux`

转为对应字符 `Foo © bar 𝌆 baz ☃ qux`

工具

- HTML entity encoder/decoder

条形码

- 宽度不等的多个黑条和空白，按照一定的编码规则排列，用以表达一组信息的图形标识符
- 国际标准
- EAN-13 商品标准，13 位数字
- Code-39：39 字符

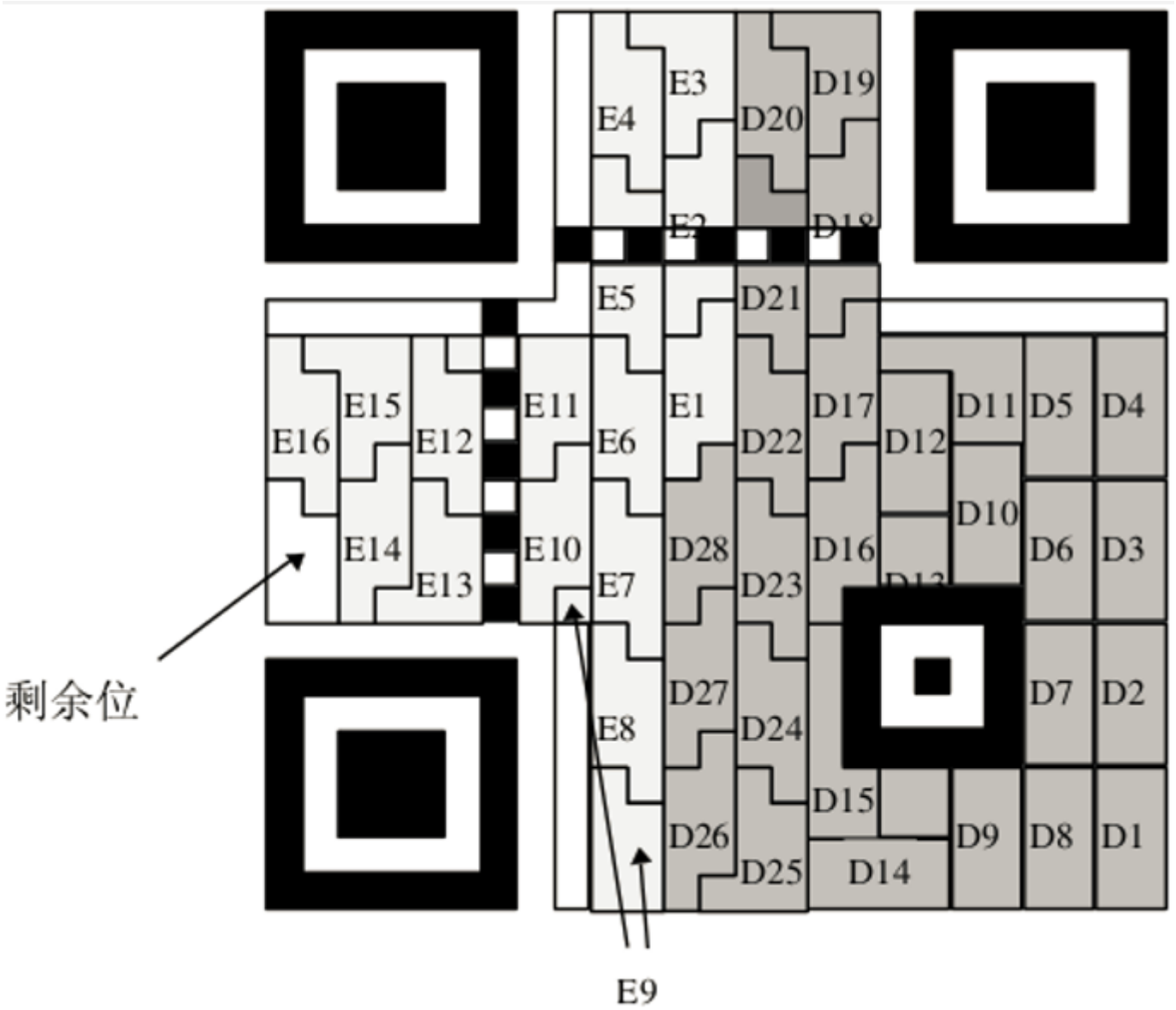
- Code-128：128 字符

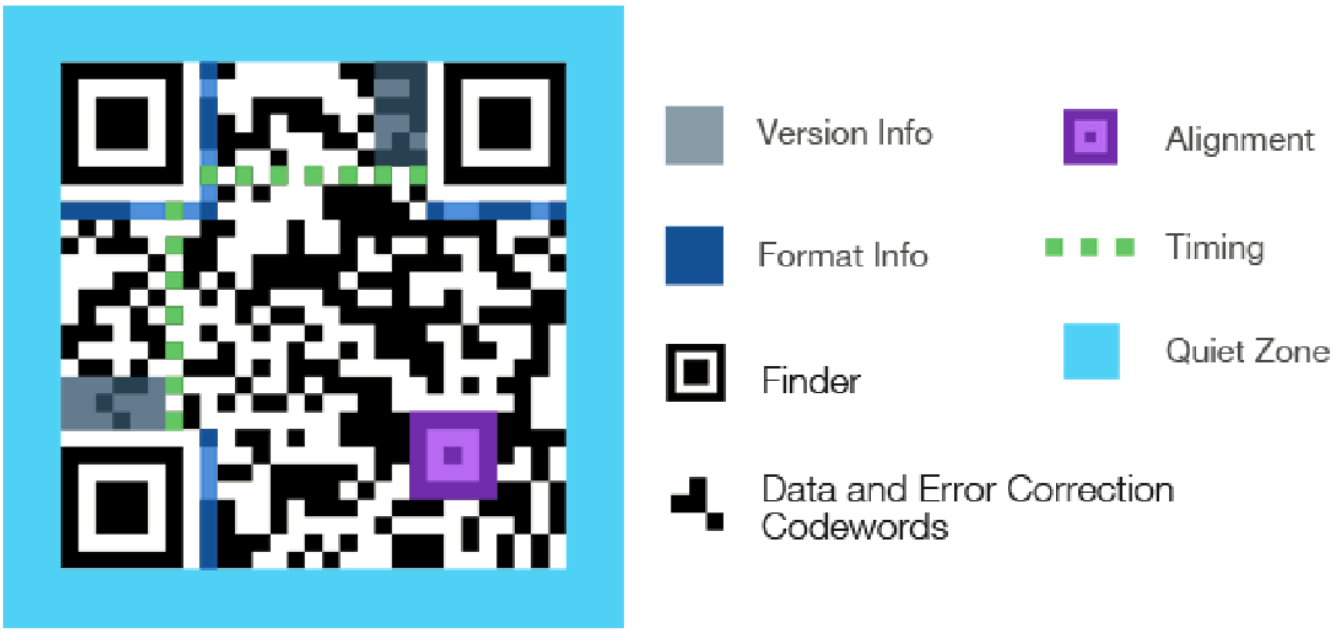
工具

- 条形码在线识别

二维码

- 用某种特定几何图形按一定规律在平面分步的黑白相间的图形记录数据符号信息
- 堆叠式 / 行排式二维码：Code 16 k、Code 49、PDF417
- 矩阵式二维码：QR CODE





工具

- 二维码生成与解码
- QR_Research

零宽字节隐写

零宽度字符隐写术（Zero-Width Space Steganography）将隐藏消息编码和解码为不可打印/可读字符。

字符包括：零宽度空格 `\u200b`，零宽度非连接符 `\u200c`，零宽度连接符 `\u200d`，从左至右书写标记 `\u200e`，从右至左书写标记 `\u200f`。

参见 http://www.ga1axy.top/index.php/archives/20/

工具

- Unicode Steganography with Zero-Width Characters

Npiet

Piet（以画家Piet Mondrian的名字命名）是最著名的深奥编程语言之一，它使用图像作为源代码。该语言使用20种颜色，并且命令被编码为相邻像素之间的颜色变化。**npiet** 是 **Piet** 程序的解释器。

工具

- npiet online

BrainFuck

BrainFuck 是最著名的深奥编程语言之一，以其极简主义而著称，只需要八个简单命令。包括

`> < + - . , [ ]`

工具

- BrainFuck

Ook!

Ook! 是对**BrainFuck**的重写。命令对应关系如下。

	A	B
1	**Brainfuck**	**Ook!**
2	**>**	**Ook. Ook?**
3	**<**	**Ook? Ook.**
4	**+**	**Ook. Ook.**
5	**-**	**Ook! Ook!**
6	**.**	**Ook! Ook.**
7	**,**	**Ook. Ook!**
8	**[**	**Ook! Ook?**
9	**]**	**Ook? Ook!**

工具

- Ook!

jother

jother是一种运用于javascript语言中利用少量字符构造精简的匿名函数方法对于字符串进行的编码方式。其中少量字符包括："!"、"+"、"("、")"、"["、"]"、"{"、"}"。只用这些字符就能完成对任意字符串的编码。

参见 jother编码之谜

工具

- J other编码工具

jjencode

jjencode代码，就是将正常的js代码转换成复杂的只有符号的字符串编码，

进行加密 如：[$._$_]+$._$+",\\"+$.$__+$.___+"\\"+$.__$+$.__$+$这样的组合。

工具

- hackvertor

aaencode

aaencode代码，是将正常的js代码转为好玩的特殊网络表情符号。

如:ﾟωﾟﾉ= / ｀ｍ´）ﾉ ~┻━┻ //*´∇｀*/ ['_']; o=(ﾟｰﾟ) 这样的表情文字.

- hackvertor

Vbscript.encode加密

取证隐写前置技能

- 了解常见的编码

能够对文件中出现的一些编码进行解码，并且对一些特殊的编码（Base64、十六进制、二进制等）有一定的敏感度，对其进行转换并得到最终的 flag。

- 能够利用脚本语言（Python 等）去操作二进制数据
- 熟知常见文件的文件格式，尤其是各类文件头、协议、结构等
- 灵活运用常见的工具

Python 操作二进制数据

struct 模块

有的时候需要用 Python 处理二进制数据，比如，存取文件，socket 操作时。这时候，可以使用 Python 的 struct 模块来完成。

struct 模块中最重要的三个函数是pack()、unpack()和calcsize()

- pack(fmt, v1, v2, ...)按照给定的格式（fmt），把数据封装成字符串（实际上是类似于 c 结构体的字节流）
- unpack(fmt, string)按照给定的格式（fmt）解析字节流 string，返回解析出来的 tuple
- calcsize(fmt)计算给定的格式（fmt）占用多少字节的内存

这里打包格式fmt确定了将变量按照什么方式打包成字节流，其包含了一系列的格式字符串。这里就不再给出不同格式字符串的含义了，详细细节可以参照Python2 struct

Go

```
>>> import struct
>>> struct.pack('>I',16)
'\x00\x00\x00\x10'
```

pack的第一个参数是处理指令，`'>I'`的意思是：`>`表示字节顺序是`Big-Endian`，也就是网络序，`I`表示 4 字节无符号整数。

后面的参数个数要和处理指令一致。

读入一个 BMP 文件的前 30 字节，文件头的结构按顺序如下

- 两个字节：BM表示 Windows 位图，BA表示 OS/2 位图
- 一个 4 字节整数：表示位图大小
- 一个 4 字节整数：保留位，始终为 0
- 一个 4 字节整数：实际图像的偏移量
- 一个 4 字节整数：Header 的字节数
- 一个 4 字节整数：图像宽度
- 一个 4 字节整数：图像高度
- 一个 2 字节整数：始终为 1
- 一个 2 字节整数：颜色数

Ruby

```
>>> import struct
>>> bmp =
'\x42\x4d\x38\x8c\x0a\x00\x00\x00\x00\x00\x36\x00\x00\x00\x28\x00\x00\x00\x80\x0
2\x00\x00\x68\x01\x00\x00\x01\x00\x18\x00'
>>> struct.unpack('<ccIIIIIIHH',bmp)
('B', 'M', 691256, 0, 54, 40, 640, 360, 1, 24)
```

bytearray 字节数组

将文件以二进制数组形式读取

JavaScript

```
data = bytearray(open('challenge.png', 'rb').read())
```

字节数组就是可变版本的字节

JavaScript

```
data[0] = '\x89'
```

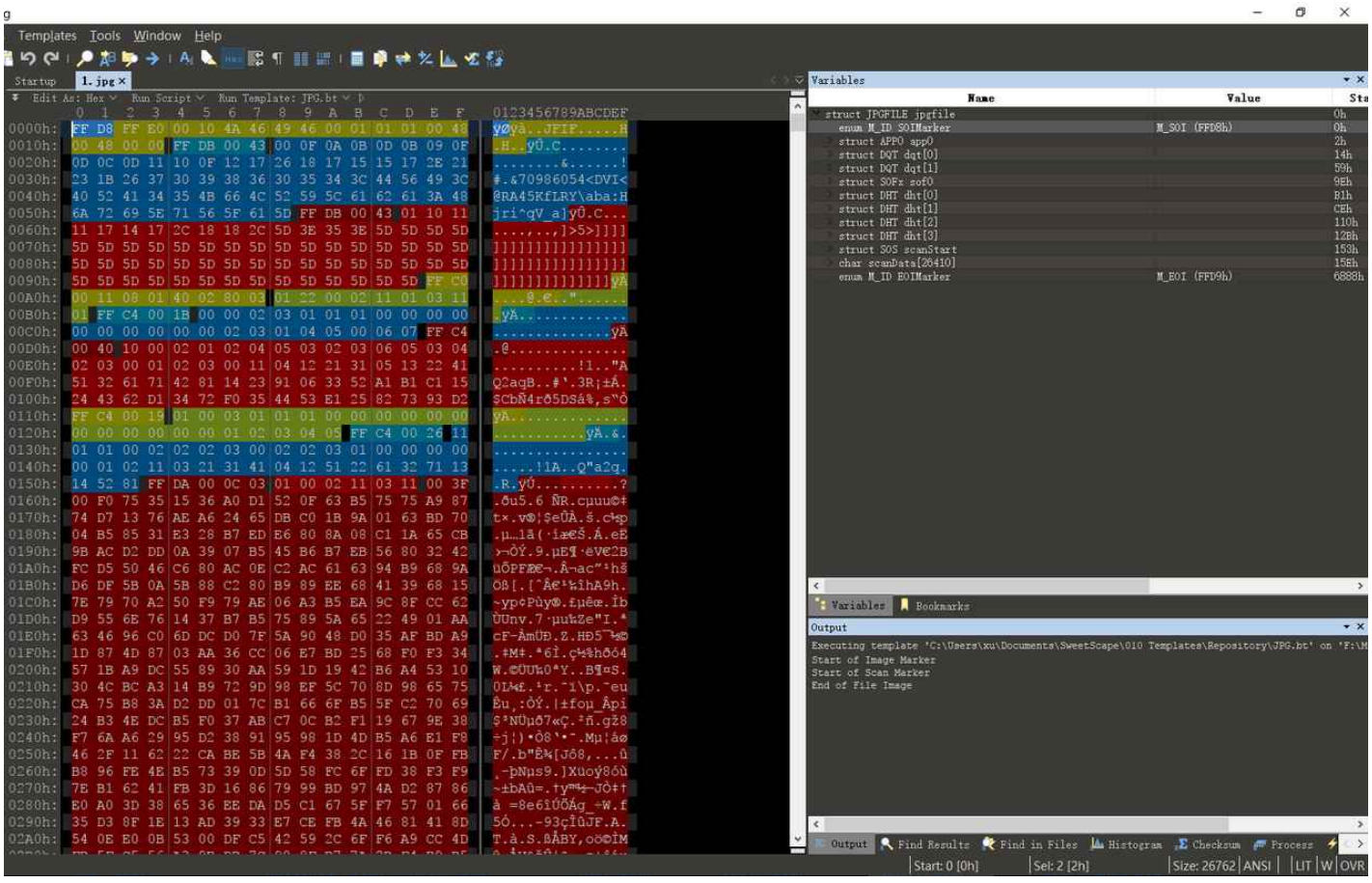
常用工具

010 Editor

SweetScape 010 Editor 是一个全新的十六进位文件编辑器，它有别于传统的十六进位编辑器在于它可用「范本」来解析二进位文件，从而让你读懂和编辑它。它还可用来比较一切可视的二进制文件。

利用它的模板功能可以非常轻松的观察文件内部的具体结构并且依此快速更改内容。

吾爱破解论坛资源



file命令

file命令根据文件头（魔法字节）去识别一个文件的文件类型。

Shell

```
$ file flag
flag: PNG image data, 450 x 450, 8-bit grayscale, non-interlaced
```

strings命令

打印文件中可打印的字符，经常用来发现文件中的一些提示信息或是一些特殊的编码信息，常常用来发现题目的突破口。

- 可以配合grep命令探测指定信息

Shell

```
$ strings flag|grep -i XXCTF
```

- 也可以配合-o参数获取所有 ASCII 字符偏移

Shell

```
$ strings -o flag|head
     14 IHDR
     45 gAMA
     64  cHRM
     141 bKGD
     157 tIME
     202 IDATx
     223 NFdVK3
     361 |;*-
     410 Ge%<W
     431 5duX@%
```

binwalk命令

binwalk 本是一个固件的分析工具，比赛中常用来发现多个文件粘合再在一起的情况。根据文件头去识别一个文件中夹杂的其他文件，有时也会存在误报率（尤其是对 Pcap 流量包等文件时）。

Shell

```
$ binwalk flag

DECIMAL       HEXADECIMAL     DESCRIPTION
--------------------------------------------------------------------------------
0             0x0             PNG image, 450 x 450, 8-bit grayscale, non-interlaced
134           0x86            Zlib compressed data, best compression
25683         0x6453          Zip archive data, at least v2.0 to extract, compressed size: 675, uncompressed size: 1159, name: readme.txt
26398         0x671E          Zip archive data, at least v2.0 to extract, compressed size: 430849, uncompressed size: 1027984, name: trid
457387        0x6FAAB         End of Zip archive
```

配合-e参数可以进行自动化提取。`$ binwalk -e flag`

也可以结合dd命令进行手动切割。

Shell

```
$ dd if=flag of=1.zip bs=1 skip=25683
431726+0 records in
431726+0 records out431726 bytes (432 kB, 422 KiB) copied, 0.900973 s, 479 kB/s
```