

Códigos

Aprenda várias formas de representar dados no computador e suas aplicações em CTFs.

- Dados e códigos
- Código ASCII
- Base64

Dados e códigos

Nos computadores, todos os dados são guardados da mesma forma: em **código binário**. Esse código é como uma sequência de '0's e '1's, e essas sequências são agrupadas e sofrem modificações para representar tudo o que vemos em um computador.

Mas se tudo são apenas zeros e uns, o que diferencia um texto de uma música? A diferença é o modo como esses dados são **representados**.

Para cada tipo de dado existe uma codificação diferente aplicada a ela, e por isso um mesmo dado pode ter várias interpretações dependendo da codificação que você trabalha.

Um exemplo de uma representação de um valor binário é como um número.

Um número em binário pode ser convertido para um número em **decimal**. Por exemplo, `00101010` é o número `42` em decimal.

Uma representação mais comum para um código binário é a base **hexadecimal**, que tem a vantagem de que cada casa pode representar uma sequência de 4 bits. Por exemplo, o mesmo número `00101010` é `2A` em hexadecimal ($0010 = 2$ e $1010 = A$).

Nas próximas seções, você verá outras formas de representar os dados no computador.

Código ASCII

Nessa seção, vamos apresentar uma das codificações mas famosas para representar texto em um computador: o ASCII.

“ O nome ASCII vem do inglês *American Standard Code for Information Interchange* que significa "Código Padrão Americano para o Intercâmbio de Informação"

O ASCII, originalmente baseado no inglês, codifica 128 caracteres específicos com **7 bits**. Como um computador normalmente trabalha na escala de bytes (8 bits), o ASCII é mais frequentemente encontrado numa representação de **8 bits**.

A tabela abaixo mostra todos os caracteres do código ASCII.

ASCII TABLE

Decimal	Hex	Char	Decimal	Hex	Char	Decimal	Hex	Char	Decimal	Hex	Char
0	0	(NUL)	32	20	(SPACE)	64	40	@	96	60	<
1	1	(START OF HEADING)	33	21	!	65	41	A	97	61	a
2	2	(START OF TEXT)	34	22	"	66	42	B	98	62	b
3	3	(END OF TEXT)	35	23	#	67	43	C	99	63	c
4	4	(FORM FEED)	36	24	\$	68	44	D	100	64	d
5	5	(NEWLINE)	37	25	%	69	45	E	101	65	e
6	6	(POSITIVE ACKNOWLEDGE)	38	26	&	70	46	F	102	66	f
7	7	(BEL)	39	27	'	71	47	G	103	67	g
8	8	(BACKSPACE)	40	28	(72	48	H	104	68	h
9	9	(TAB)	41	29)	73	49	I	105	69	i
10	A	(LINE FEED)	42	2A	*	74	4A	J	106	6A	j
11	B	(VERTICAL TAB)	43	2B	+	75	4B	K	107	6B	k
12	C	(FORM FEED)	44	2C	,	76	4C	L	108	6C	l
13	D	(CARRIAGE RETURN)	45	2D	-	77	4D	M	109	6D	m
14	E	(SHIFT OUT)	46	2E	.	78	4E	N	110	6E	n
15	F	(SHIFT IN)	47	2F	/	79	4F	O	111	6F	o
16	10	(DATA LINK ESCAPE)	48	30	0	80	50	P	112	70	p
17	11	(DEVICE CONTROL 1)	49	31	1	81	51	Q	113	71	q
18	12	(DEVICE CONTROL 2)	50	32	2	82	52	R	114	72	r
19	13	(DEVICE CONTROL 3)	51	33	3	83	53	S	115	73	s
20	14	(DEVICE CONTROL 4)	52	34	4	84	54	T	116	74	t
21	15	(NEGATIVE ACKNOWLEDGE)	53	35	5	85	55	U	117	75	u
22	16	(SYNCHRONOUS IDLE)	54	36	6	86	56	V	118	76	v
23	17	(END OF TRANSMISSION)	55	37	7	87	57	W	119	77	w
24	18	(CANCEL)	56	38	8	88	58	X	120	78	x
25	19	(END OF MESSAGE)	57	39	9	89	59	Y	121	79	y
26	1A	(SUBSTITUTE)	58	3A	:	90	5A	Z	122	7A	z
27	1B	(ESCAPE)	59	3B	;	91	5B	[123	7B	{
28	1C	(FILE SEPARATOR)	60	3C	<	92	5C	\	124	7C	
29	1D	(GROUP SEPARATOR)	61	3D	=	93	5D]	125	7D	}
30	1E	(MESSAGE SEPARATOR)	62	3E	>	94	5E	^	126	7E	~
31	1F	(UNIT SEPARATOR)	63	3F	?	95	5F	_	127	7F	(DEL)

Você pode usar essa tabela para decodificar uma sequência de números em binário, decimal ou hexadecimal para ASCII.

ASCII em Python

Uma maneira mais fácil de manipular a codificação ASCII, em vez de usar manualmente uma tabela, é por meio de códigos. Em Python, usamos as funções `ord()` e `chr()` para isso.

A função `ord()` recebe uma string de tamanho 1 e retorna um inteiro que representa o código da letra, se ela for ASCII, devolverá seu código ASCII. Por exemplo, `ord('a')` devolverá `97`.

Já a função `chr()` é o inverso da anterior. Ela recebe um inteiro e devolve o caractere com o respectivo código ASCII. Por exemplo, `chr(97)` devolverá `'a'`.

Exercícios

WeChall: ASCII

WeChall: URL

Referências

ASCII table

Python Built-in Functions

Base64

Agora que você já teve contato com o código ASCII, vamos conhecer o **Base64**, um método para codificar e decodificar dados binários em caracteres ASCII.

Esse método é utilizado frequentemente para transferir dados em meios que só suportam formatos ASCII, por exemplo, enviar anexos por e-mail (que usa o **MIME**).

O nome base64 origina-se do fato de que esse sistema é constituído de 64 caracteres, representando exatamente 6 bits de dados. Com isso, três bytes de 8 bits podem ser representados por 4 dígitos de 6 bits em Base64.

A tabela abaixo mostra a equivalência entre os valores de um conjunto de 6 bits e os caracteres usados para codificação.

Valor	Caractere	Valor	Caractere	Valor	Caractere	Valor	Caractere
0	A	16	Q	32	g	48	w
1	B	17	R	33	h	49	x
2	C	18	S	34	i	50	y
3	D	19	T	35	j	51	z
4	E	20	U	36	k	52	0
5	F	21	V	37	l	53	1
6	G	22	W	38	m	54	2
7	H	23	X	39	n	55	3
8	I	24	Y	40	o	56	4
9	J	25	Z	41	p	57	5
10	K	26	a	42	q	58	6
11	L	27	b	43	r	59	7
12	M	28	c	44	s	60	8
13	N	29	d	45	t	61	9
14	O	30	e	46	u	62	+
15	P	31	f	47	v	63	/

Quando os bits da mensagem original não são múltiplos de 6, são adicionados zeros como *padding*. Assim, na mensagem codificada é colocado um `=` para cada dois zeros de *padding*. Aliás, esse `=` é uma forma bem comum de reconhecer um texto codificado em Base64.

Abaixo está um exemplo de um texto codificado em Base64:

```
texto:      M      |      a      |      n
8 bits:  0 1 0 0 1 1 0 1 | 0 1 1 0 0 0 0 1 | 0 1 1 0 1 1 1 0
6 bits:  0 1 0 0 1 1 | 0 1 0 1 1 0 | 0 0 0 1 0 1 | 1 0 1 1 1 0
Valor:      19      |      22      |      5      |      46
texto:      T      |      W      |      F      |      u
(Base64)
```

E se tirarmos o `n`, a codificação vai necessitar de um *padding*:

```
texto:      M      |      a      |
8 bits:  0 1 0 0 1 1 0 1 | 0 1 1 0 0 0 0 1 |
6 bits:  0 1 0 0 1 1 | 0 1 0 1 1 0 | 0 0 0 1 0 0 | 0 0 0 0 0 0
Valor:      19      |      22      |      5      | (padding)
texto:      T      |      W      |      E      |      =
(Base64)
```

Ferramentas

Para manipular textos em Base64, pode-se usar o comando Unix `base64`.

Por exemplo, se tivermos uma arquivo `sagan.txt` com o texto `The Cosmos is all that is or ever was or ever will be`. Podemos convertê-lo para um arquivo `sagan64.txt` com o comando

```
base64 sagan.txt > sagan64.txt
```

O resultado será o arquivo `sagan64.txt` com o texto

```
VGhlIENvc21vcyBpcyBhbGwgdGhhdCBpcyBvcjBlbmVvIHdhcyBvcjBlbmVvIHdpbGwgYmUK
```

Agora, para decodificar o arquivo `sagan64.txt`, usamos o mesmo comando com a flag `-d`:

```
base64 -d sagan64.txt
```

Exercícios

OverTheWire: Krypton 0

Decodifique essa mensagem