Lab 6: Ciphers and Digital Certificates

In this lab we will investigate a range of ciphers and also how we can view the details within digital certificates. Along with this, we will setup our

A Ciphers

Use your desktop computer to complete the following:

No	Description	Result
1	Go to: **http://asecuritysite.com/Challenges** Click on the "Start Challenge" button, and see if you can score over 40 points.	Your final score:
2	Now see if you can crack the five-minute cracking challenge for: http://asecuritysite.com/challenges/scramb	Your fastest time:

B Digital Certificates

Now, we will try to crack some certificates, and gain access to their key pairs.

No	Description	Result
1	Try and crack some certificates from: http://asecuritysite.com/Encryption/certcrack What are the passwords for 'bill09.pfx', 'bill18.pfx', and 'country04.pfx'?	bill09.pfx: bill18.pfx: country04.pfx:

2. We can also create a short **Python script** to try to crack the same certificates.

Boot up your Kali VM on your public network, and download the following archive:

http://asecuritysite.com/public/certs.zip

Extract the certificates into the /root folder, and then move into that folder. Now use openssl to try a password:

```
openssl pkcs12 -nokeys -in bill01.pfx -passin pass:orange
```

Did you manage to run the script?

What password is correct for bill01.pfx?

Now implement the Python script given below:

```
from OpenSSL import crypto
words=[]
words.append("coconut")
words.append("mango")
words.append("apples")
words.append("apple")
words.append("oranges")
words.append("orange")
words.append("ankle")
words.append("password")
words.append("bill")
words.append("battery")

for passwd in words:

  try:
    p12 = crypto.load_pkcs12(open("fredpfx.pfx", 'rb').read(), passwd)
    certificate =p12.get_certificate()

    p12.get_privatekey()
    print (certificate.get_serial_number())
    print (certificate.get_issuer().get_components())
    print (certificate.get_signature_algorithm())
    print ("Success: "+passwd)
  except Exception as ex:
    print (".")
```

https://repl.it/@billbuchanan/csn09112digcert01

Can adapt this script to crack some of the other certificates contained in the archive you have downloaded. Bill01.pdf to bill18.pdf are based on fruits (in lowercase), country01.pdf to country06.pdf are based on countries.

Outline the passwords of the certificates:

Can you modify the code so that it shows other details from the certificate, such as its public key, subject, version and “notBefore”, and “notAfter”.

Ref: https://pyopenssl.org/en/0.15.1/api/crypto.html#x509name-objects

C Coursework

The coursework specification is at:
https://github.com/billbuchanan/csn09112/tree/master/coursework

Overall, you must analyse the operation of a bot and a controller, and where the controller waits for a network connection from the bot (Figure 1). Once connected, the pass secret messages to each other. You first task is to analyse the messages they send, and try and crack them. You can either use your AWS instances or vSoC 2. In the second part of the coursework, we will use Snort to detect the presence of the bot.

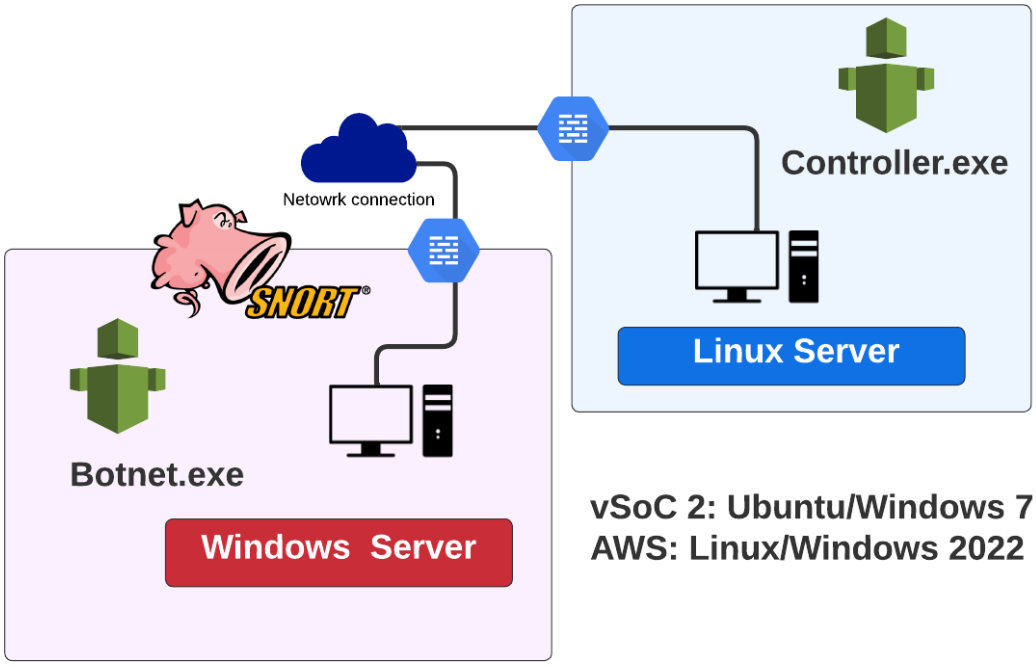

Figure 1: Coursework setup

D Coursework setup (AWS)

You can complete your coursework either on vSoC or within AWS. This section will setup your environment for Snort and the Botnet for Linux and Windows 2022.

D.1 Setup Windows

Setup your Windows 2022 for a remote desktop connection (see a previous lab). The steps are then:

- Install .NET 2.0 and .NET 3.0. For this select Server Manager, and then add ".NET Framework 3.5 Features" (as shown in Figure 2).
- Install WinPCap from https://www.winpcap.org/.
- Install Wireshark from https://www.wireshark.org/download.html.
- Install **Snort 2.9.9.0** from https://www.snort.org/downloads/archive/snort/Snort_2_9_9_0_Installer.exe
- Download Botnet.exe and Controller.exe from https://github.com/billbuchanan/csn09112/blob/master/coursework/c.zip
- Extract Botnet.exe and Controller.exe to the c:\botnet folder.

- Navigate to c:\botnet from the command line, and test that Botnet.exe will run.

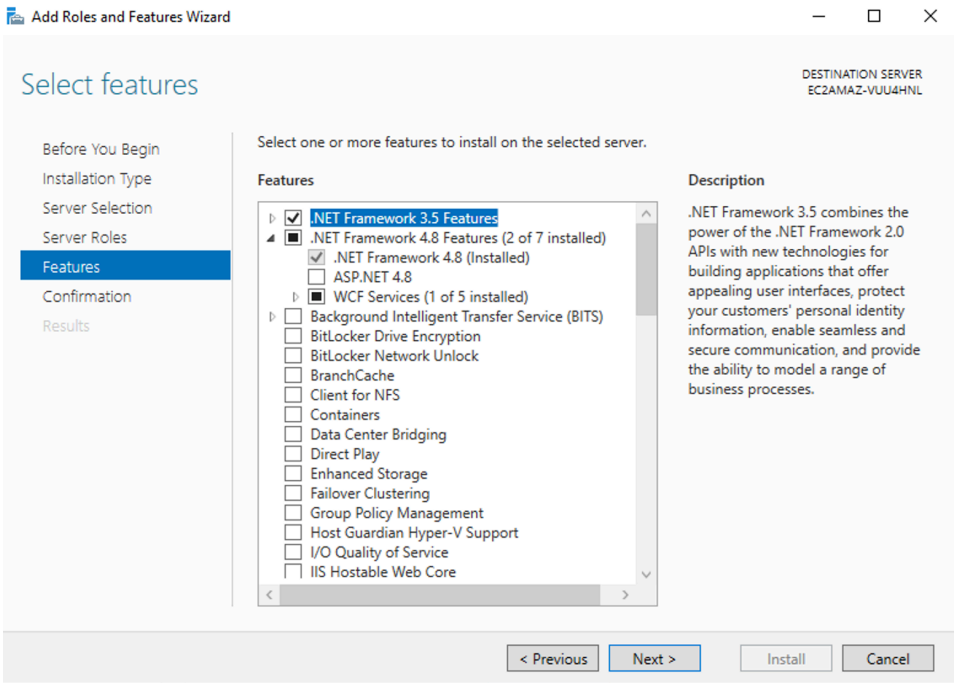

Figure 2: Installing .NET 3.5 Features

D.2 Setup Linux

Setup your Linux AWS instance for a remote SSH connection. The steps are then:

```
# wget 'https://github.com/billbuchanan/csn09112/blob/master/coursework/c.zip?raw=true'
# mv c.zip?raw=true c.zip
# unzip c.zip
```

This should extract the files of botnet.exe and controller.exe. The controller will wait for a connection from the botnet. The ports used for the connection will range from 5,000 to 5,100, so open up the firewall on your Linux AWS instance (Figure 3).

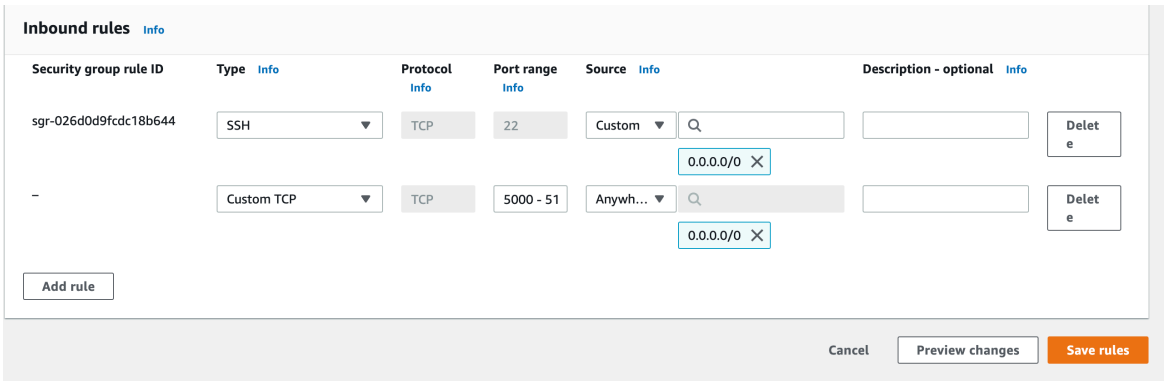

Figure 3: Opening up Ports 5,000 to 5,100 on Linux

Next run controller.exe (Figure 4) with:

```
# mono controller.exe
```

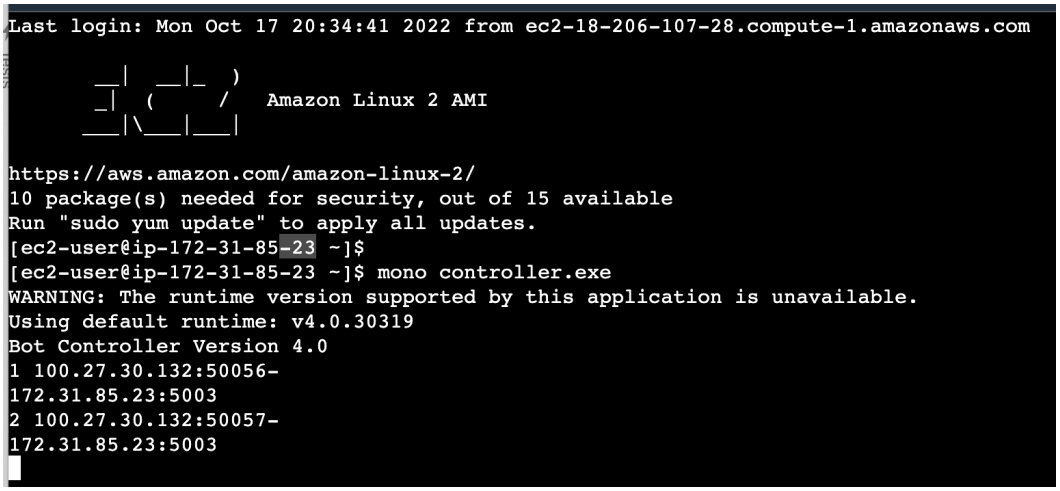

Figure 4: Running the controller.exe

There will be no network connections shown yet, as we now need to run the Botnet from the Windows instance.

D.3 Running the bot

Now we will run the bot, and make a connection. For this determine the public IP address of the Linux instance, and then run the bot with this address as an argument. For example, if the IP address of your Linux instance is 54.205.20.103, the bot can be run with:

```
C:\> botnet 54.205.20.103
```

Make sure that you have made a connection with the controller (as see in Figure 4 and Figure 5).

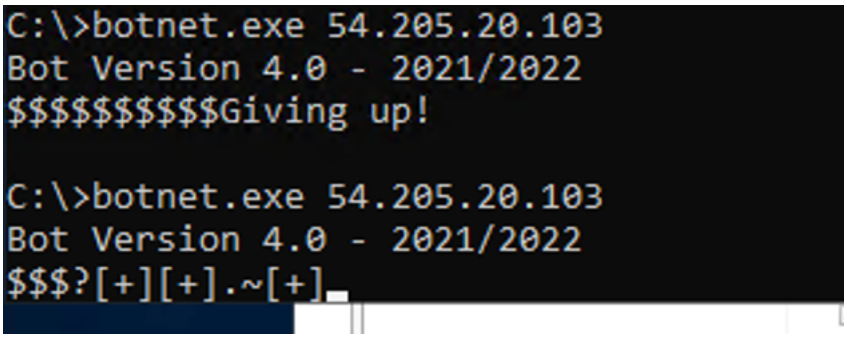

Figure 5: Running the botnet on Windows 2022

Now, stop the bot (with Ctrl-C), and will we now run Wireshark and capture the traffic. Restart our Bot, and make sure you are capturing traffic. After it has finished, stop Wireshark and view the Wireshark trace (see Figure 5). You can use the ip.addr filter to focus on the traffic that relates to your Botnet connection.

From the traffic generated on Wireshark, determine the following:

Which TCP server port has been used for the connection:

Which TCP client port has been used for the connection:

By clicking on a network packet, and selecting “Follow stream” (Figure 6). What are the messages that the bot sends to the controller:

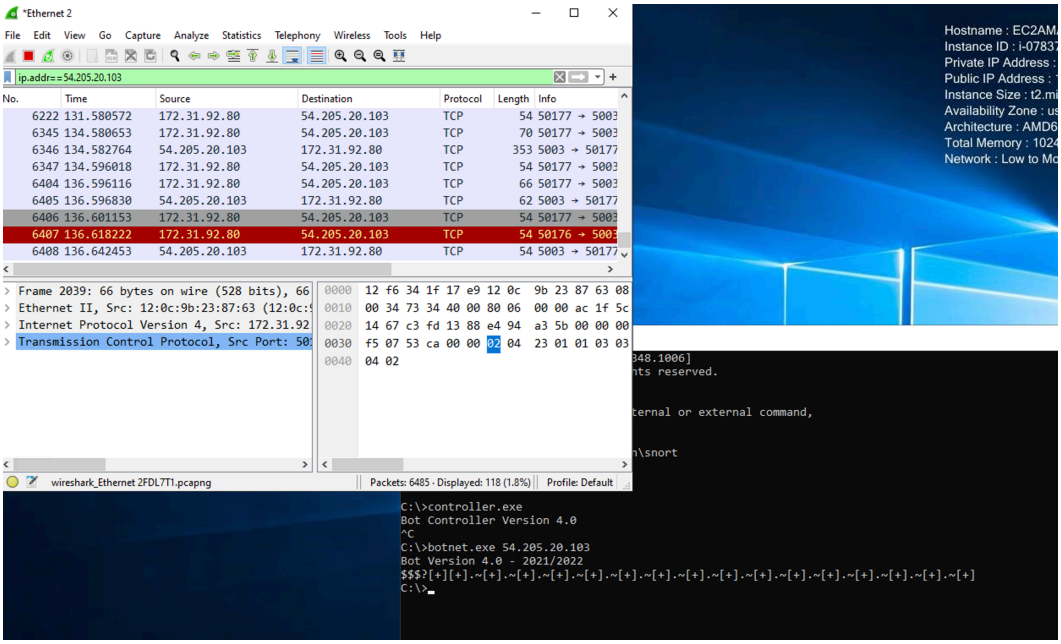

Figure 5: Capturing network traffic from Bot

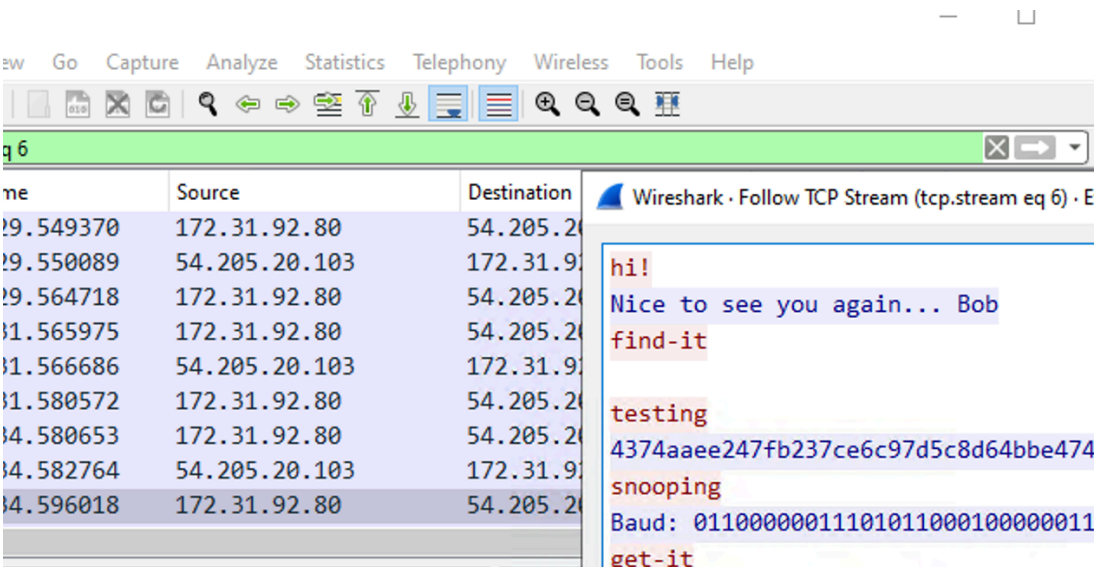

Figure 6: Analysing traffic flow from bot and controller

D.4 Running Snort

Quit the bot, and now we will run Snort on Windows 2022. You can also run it on Linux, if you want. Run create a Snort file (such as with the name 1.snort):

```
alert tcp any any -> any 5000 ( msg:"Sample alert";  sid:1000; rev:1; )
alert tcp any any -> any 5001 ( msg:"Sample alert";  sid:1001; rev:1; )
alert tcp any any -> any 5002 ( msg:"Sample alert";  sid:1002; rev:1; )
alert tcp any any -> any 5003 ( msg:"Sample alert";  sid:1003; rev:1; )
alert tcp any any -> any 5004 ( msg:"Sample alert";  sid:1004; rev:1; )
alert tcp any any -> any 5005 ( msg:"Sample alert";  sid:1005; rev:1; )
alert tcp any any -> any 5006 ( msg:"Sample alert";  sid:1006; rev:1; )
alert tcp any any -> any 5007 ( msg:"Sample alert";  sid:1007; rev:1; )
alert tcp any any -> any 5008 ( msg:"Sample alert";  sid:1008; rev:1; )
alert tcp any any -> any 5009 ( msg:"Sample alert";  sid:1009; rev:1; )


# Some additional pre-processor things
preprocessor stream5_global: track_tcp yes, \
track_udp yes, \
track_icmp no, \
max_tcp 262144, \
max_udp 131072, \
max_active_responses 2, \
min_response_seconds 5
preprocessor stream5_tcp: policy windows, detect_anomalies, require_3whs 180, \
overlap_limit 10, small_segments 3 bytes 150, timeout 180, \
ports client 21 22 23 25 42 53 70 79 109 110 111 113 119 135 136 137 139 143 \
161 445 513 514 587 593 691 1433 1521 1741 2100 3306 6070 6665 6666 6667 6668 6669 \
7000 8181 32770 32771 32772 32773 32774 32775 32776 32777 32778 32779, \
ports both 80 81 82 83 84 85 86 87 88 89 90 110 311 383 443 465 563 591 593 631 636
901 989 992 993 994 995 1220 1414 1830 2301 2381 2809 3037 3057 3128 3443 3702 4343
4848 5250 6080 6988 7907 7000 7001 7144 7145 7510 7802 7777 7779 \
7801 7900 7901 7902 7903 7904 7905 7906 7908 7909 7910 7911 7912 7913 7914 7915 7916
\
7917 7918 7919 7920 8000 8008 8014 8028 8080 8085 8088 8090 8118 8123 8180 8222 8243
8280 8300 8500 8800 8888 8899 9000 9060 9080 9090 9091 9443 9999 10000 11371 34443
34444 41080 50000 50002 55555
preprocessor stream5_udp: timeout 180
```

You save your file, and then run Snort from the command line with (make sure you have created a log folder in the place below where you are running Snort):

```
C:\> c:\snort\bin\snort -i 1 -c 1.snort -k none -K ascii -l log
```

A sample run in shown in Figure 7. Now make the connection between the Bot and the Controller, and wait for data to be transferred. Then stop Snort, and examine the alert.ids file contained in the log folder. A sample run should show the form of:

```
[**] [1:1000:1] Sample alert [**]
[Priority: 0]
10/18-06:02:11.621177 172.31.92.80:49735 -> 54.211.227.176:5000
TCP TTL:128 TOS:0x2 ID:30750 IpLen:20 DgmLen:52 DF
12****S* Seq: 0xD6CEAD40  Ack: 0x0  Win: 0xF507  TcpLen: 32
TCP Options (6) => MSS: 8961 NOP WS: 8 NOP NOP SackOK

[**] [1:1000:1] Sample alert [**]
[Priority: 0]
10/18-06:02:12.122246 172.31.92.80:49735 -> 54.211.227.176:5000
TCP TTL:128 TOS:0x0 ID:30751 IpLen:20 DgmLen:52 DF
******S* Seq: 0xD6CEAD40  Ack: 0x0  Win: 0xF507  TcpLen: 32
TCP Options (6) => MSS: 8961 NOP WS: 8 NOP NOP SackOK

[**] [1:1000:1] Sample alert [**]
[Priority: 0]
10/18-06:02:12.637905 172.31.92.80:49735 -> 54.211.227.176:5000
TCP TTL:128 TOS:0x0 ID:30752 IpLen:20 DgmLen:52 DF
```

```
******S* Seq: 0xD6CEAD40  Ack: 0x0  Win: 0xF507  TcpLen: 32
TCP Options (6) => MSS: 8961 NOP WS: 8 NOP NOP SackOK

[**] [1:1000:1] Sample alert [**]
[Priority: 0]
10/18-06:02:13.153531 172.31.92.80:49735 -> 54.211.227.176:5000
TCP TTL:128 TOS:0x0 ID:30753 IpLen:20 DgmLen:52 DF
******S* Seq: 0xD6CEAD40  Ack: 0x0  Win: 0xF507  TcpLen: 32
TCP Options (6) => MSS: 8961 NOP WS: 8 NOP NOP SackOK
```

```
C:\>
C:\>
C:\>
C:\>c:\snort\bin\snort -i 1 -c 1.snort -k none -K ascii -l log
Running in IDS mode

        --== Initializing Snort ==--
Initializing Output Plugins!
Initializing Preprocessors!
Initializing Plug-ins!
Parsing Rules file "1.snort"
Tagged Packet Limit: 256
Log directory = log
      Max Expected Streams: 768
Stream global config:
    Track TCP sessions: ACTIVE
    Max TCP sessions: 262144
    TCP cache pruning timeout: 30 seconds
    TCP cache nominal timeout: 3600 seconds
    Memcap (for reassembly packet storage): 8388608
    Track UDP sessions: ACTIVE
    Max UDP sessions: 131072
    UDP cache pruning timeout: 30 seconds
    UDP cache nominal timeout: 180 seconds
    Track ICMP sessions: INACTIVE
    Track IP sessions: INACTIVE
    Log info if session memory consumption exceeds 1048576
    Send up to 2 active responses
    Wait at least 5 seconds between responses
    Protocol Aware Flushing: ACTIVE
```

Figure 7: Running Snort

E Test

1. Crack some Caesar codes at:
   http://asecuritysite.com/tests/tests?sortBy=caesar
2. Determine some hex conversions at:
   http://asecuritysite.com/tests/tests?sortBy=hex01
3. Determine some Base64 conversions:
   http://asecuritysite.com/tests/tests?sortBy=ascii01

Shifted alphabet

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

```
 7   G H I J K L M N O P Q R S T U V W X Y Z A B C D E F
 8   H I J K L M N O P Q R S T U V W X Y Z A B C D E F G
 9   I J K L M N O P Q R S T U V W X Y Z A B C D E F G H
10   J K L M N O P Q R S T U V W X Y Z A B C D E F G H I
11   K L M N O P Q R S T U V W X Y Z A B C D E F G H I J
12   L M N O P Q R S T U V W X Y Z A B C D E F G H I J K
13   M N O P Q R S T U V W X Y Z A B C D E F G H I J K L
14   N O P Q R S T U V W X Y Z A B C D E F G H I J K L M
15   O P Q R S T U V W X Y Z A B C D E F G H I J K L M N
16   P Q R S T U V W X Y Z A B C D E F G H I J K L M N O
17   Q R S T U V W X Y Z A B C D E F G H I J K L M N O P
18   R S T U V W X Y Z A B C D E F G H I J K L M N O P Q
19   S T U V W X Y Z A B C D E F G H I J K L M N O P Q R
20   T U V W X Y Z A B C D E F G H I J K L M N O P Q R S
21   U V W X Y Z A B C D E F G H I J K L M N O P Q R S T
22   V W X Y Z A B C D E F G H I J K L M N O P Q R S T U
23   W X Y Z A B C D E F G H I J K L M N O P Q R S T U V
24   X Y Z A B C D E F G H I J K L M N O P Q R S T U V W
25   Y Z A B C D E F G H I J K L M N O P Q R S T U V W X
26   Z A B C D E F G H I J K L M N O P Q R S T U V W X Y
```

ASCII table

(nul)	0	0000	0x00	(sp)	32	0040	0x20	@	64	0100	0x40	`	96	0140	0x60
(soh)	1	0001	0x01	!	33	0041	0x21	A	65	0101	0x41	a	97	0141	0x61
(stx)	2	0002	0x02	"	34	0042	0x22	B	66	0102	0x42	b	98	0142	0x62
(etx)	3	0003	0x03	#	35	0043	0x23	C	67	0103	0x43	c	99	0143	0x63
(eot)	4	0004	0x04	$	36	0044	0x24	D	68	0104	0x44	d	100	0144	0x64
(enq)	5	0005	0x05	%	37	0045	0x25	E	69	0105	0x45	e	101	0145	0x65
(ack)	6	0006	0x06	&	38	0046	0x26	F	70	0106	0x46	f	102	0146	0x66
(bel)	7	0007	0x07	'	39	0047	0x27	G	71	0107	0x47	g	103	0147	0x67
(bs)	8	0010	0x08	(	40	0050	0x28	H	72	0110	0x48	h	104	0150	0x68
(ht)	9	0011	0x09	)	41	0051	0x29	I	73	0111	0x49	i	105	0151	0x69
(nl)	10	0012	0x0a	*	42	0052	0x2a	J	74	0112	0x4a	j	106	0152	0x6a
(vt)	11	0013	0x0b	+	43	0053	0x2b	K	75	0113	0x4b	k	107	0153	0x6b
(np)	12	0014	0x0c	,	44	0054	0x2c	L	76	0114	0x4c	l	108	0154	0x6c
(cr)	13	0015	0x0d	-	45	0055	0x2d	M	77	0115	0x4d	m	109	0155	0x6d
(so)	14	0016	0x0e	.	46	0056	0x2e	N	78	0116	0x4e	n	110	0156	0x6e
(si)	15	0017	0x0f	/	47	0057	0x2f	O	79	0117	0x4f	o	111	0157	0x6f
(dle)	16	0020	0x10	0	48	0060	0x30	P	80	0120	0x50	p	112	0160	0x70
(dc1)	17	0021	0x11	1	49	0061	0x31	Q	81	0121	0x51	q	113	0161	0x71
(dc2)	18	0022	0x12	2	50	0062	0x32	R	82	0122	0x52	r	114	0162	0x72
(dc3)	19	0023	0x13	3	51	0063	0x33	S	83	0123	0x53	s	115	0163	0x73
(dc4)	20	0024	0x14	4	52	0064	0x34	T	84	0124	0x54	t	116	0164	0x74
(nak)	21	0025	0x15	5	53	0065	0x35	U	85	0125	0x55	u	117	0165	0x75
(syn)	22	0026	0x16	6	54	0066	0x36	V	86	0126	0x56	v	118	0166	0x76
(etb)	23	0027	0x17	7	55	0067	0x37	W	87	0127	0x57	w	119	0167	0x77
(can)	24	0030	0x18	8	56	0070	0x38	X	88	0130	0x58	x	120	0170	0x78
(em)	25	0031	0x19	9	57	0071	0x39	Y	89	0131	0x59	y	121	0171	0x79
(sub)	26	0032	0x1a	:	58	0072	0x3a	Z	90	0132	0x5a	z	122	0172	0x7a
(esc)	27	0033	0x1b	;	59	0073	0x3b	[	91	0133	0x5b	{	123	0173	0x7b
(fs)	28	0034	0x1c	<	60	0074	0x3c	\	92	0134	0x5c	|	124	0174	0x7c
(gs)	29	0035	0x1d	=	61	0075	0x3d	]	93	0135	0x5d	}	125	0175	0x7d
(rs)	30	0036	0x1e	>	62	0076	0x3e	^	94	0136	0x5e	~	126	0176	0x7e
(us)	31	0037	0x1f	?	63	0077	0x3f	_	95	0137	0x5f	(del)	127	0177	0x7f

Base 64

Example:

"fred"	01100110	01110010	01100101	01100100		
Split into 6 bits:	011001	100111	001001	100101	011001	00
	Z	n	J	l	Z	A =

=

Val	Encoding	Val	Encoding	Val	Encoding	Val	Encoding
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	/