Orethanos
Tech security

RetroATtualizando

AT COMMANDS

Buenaventura Salcedo

WHOAMI

- Graduado en Ingeniería Informática UNED (2019)
- Máster de Ciberseguridad UNED (en curso TFM, 2021)
- CEO Servicio Técnico de Telefonía Movil e Informática (2004)
- Desarrollador de herramientas forenses para smartphones

nomed1

AGRADECIMIENTOS

- A la Organización de BITUP
- Colaboradores de BITUP

MOTIVACIONES

- Comunicaciones con modems desde Windows 3.X
- Fácil implementación en Python (entre otros)
- Descubrir AT Commands válidos para cada dispositivo
- Interés desde el punto de vista forense
- Podemos considerar vulnerables para exfiltración
- Podemos exponer casos reales

ÍNDICE

AT COMMANDS

Son instrucciones utilizadas para controlar un modem

Lenjuage desarrollado por HAYES Communications

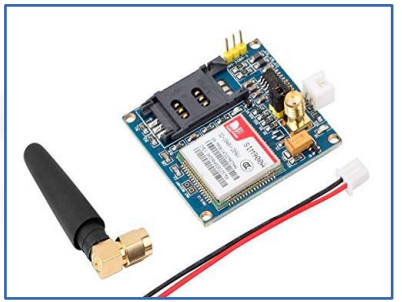
Se convirtió en un estándar abierto para modems

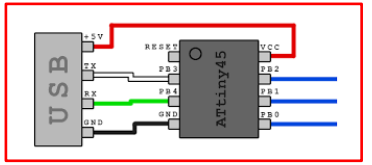
Los fabricantes varian y eligen su repertorio(*) y como es sencillo...

Podemos:

- Gestionar llamadas (colgar, descolgar, controlar volumen,...)
- Gestionar SMS (envio, recepción, ...)
- Consultar estado (info,bateria, fechas, nivel de señal, …)
- Consulta información interna(versiones, llamadas, contactos, …)
- Conseguir instrucciones vulnerables para exfiltrar información

DISPOSITIVOS





CONFIGURACIÓN

- Puerto
- Velocidad
- Bits
- Paridad
- Bits de parada
- Timeout
- Control de flujo software
- Control de flujo hardware

(DEVICE,RATE, bytesize=8, parity='N', stopbits=1, timeout=TIMEOUT, xonxoff=0, rtscts=1)

CONFIGURACIÓN

class serial.**Serial**

__init__(*port=None*, *baudrate=9600*, *bytesize=EIGHTBITS*, *parity=PARITY_NONE*, *stopbits=STOPBITS_ONE*, *timeout=None*, *xonxoff=False*, *rtscts=False*, *write_timeout=None*, *dsrdtr=False*, *inter_byte_timeout=None*)

Parameters:

- **port** – Device name or `None`.
- **baudrate** (*int*) – Baud rate such as 9600 or 115200 etc.
- **bytesize** – Number of data bits. Possible values: `FIVEBITS`, `SIXBITS`, `SEVENBITS`, `EIGHTBITS`
- **parity** – Enable parity checking. Possible values: `PARITY_NONE`, `PARITY_EVEN`, `PARITY_ODD` `PARITY_MARK`, `PARITY_SPACE`
- **stopbits** – Number of stop bits. Possible values: `STOPBITS_ONE`, `STOPBITS_ONE_POINT_FIVE`, `STOPBITS_TWO`
- **timeout** (*float*) – Set a read timeout value.
- **xonxoff** (*bool*) – Enable software flow control.
- **rtscts** (*bool*) – Enable hardware (RTS/CTS) flow control.
- **dsrdtr** (*bool*) – Enable hardware (DSR/DTR) flow control.
- **write_timeout** (*float*) – Set a write timeout value.
- **inter_byte_timeout** (*float*) – Inter-character timeout, `None` to disable (default).

Raises:

- **ValueError** – Will be raised when parameter are out of range, e.g. baud rate, data bits.
- **SerialException** – In case the device can not be found or can not be configured.

CONFIGURACIÓN

Ejemplo con Python para Linux con la clase Serial del paquete pyserial

```
phone = serial.Serial(
        port = "/dev/ttyACM0", #ttyUSBX or windows COMX
        baudrate = 115200, #9600,14400,19200,38400,57600,115200,230400,460800
        bytesize = 8,
        parity = 'N',
        stopbits = 1,
        timeout = 3,
        xonxoff = 0,
        rtscts = 1)
```

SINTAXIS

AT+COMANDO [? | = ? | = valor1,valor2,…,valorN]

AT+COMANDO	EJECUCIÓN
AT+COMANDO?	LECTURA
AT+COMANDO=?	CONSULTA
AT+COMANDO=valores	MODIFICACION

- Los comandos pueden ir en minúsculas
- El espacio de nombres se amplia con los simbolos **[+*!@#$%^&]**
- Esto depende del fabricante y de su propio repertorio de instrucciones podemos buscar las instrucciones en la ROM del dispositivo y crear nuestras bases de datos.

POC 1 – MY CONSOLE AT COMMANDS

Precondiciones via USB (via jtag no tiene porque ser así)

- Test Card SIM o SIM sin PIN dentro
- OK después de conectar el cable
- Permisos al dispositivo en Linux **sudo chmod 666 dev/ttyACM0**

POC 1 – MY CONSOLE AT COMMANDS

Vayamos al codigo fuente:

```
1 #requirements pyserial, you can do pip3 install pyserial
2 import serial
3 import sys
4
5 DEVICE = '/dev/ttyACM0'
6 RATE = 115200 #115200 #1200,2400,4800,9600,14400,19200,38400,57600,115200,230400,460800
7 TIMEOUT = 3 # you can put 0.5
8
9 #inicializate
10 print ("Little at command console by @nomed1")
11
12 phone = serial.Serial(DEVICE,RATE, bytesize=8, parity='N', stopbits=1, timeout=TIMEOUT, xonxoff=0, rtscts=1)
13 command = ""
14
15 while (command != "exit"):
16   command = input("$> ")
17   if (command != "exit"):
18     phone.write(str.encode(command + '\r',"utf-8"))
19     response = bytes.decode(phone.readall())
20     print (response)
21
22 print ("Disconnecting ...")
23 phone.close()
24
```

POC 2 – ATCOMMANDS SOPORTADOS

strings file | grep “AT[+*!@#$%^&]”

POC 3 – PARSE CONTACTS TO VCF

Trabajar con los contactos:

1.- Listar los almacenamientos

2.- Crear un bucle que recorra cada almacenamiento

- 2.1.- Seleccionar el almacenamiento
- 2.2.- Pedir información del almacenamiento
- 2.3.- Solicitar la lectura de los contactos
- 2.4.- Parsear los contactos y guardar

POC 3 – PARSE CONTACTS TO VCF

1.- Listar los almacenamientos:

```
def get_locations():
    '''get locations cleans'''
    command = "+CPBS"
    POST = "=?"
    phone.write(str.encode(PRE + command + POST + '\r',"utf-8"))
    response = bytes.decode(phone.readall())
    s = ""
    if "OK" in response:
        lines = response.replace('\r',"").split('\n')
        for l in lines:
            if command + ": " in l:
                #print (hexa(i))
                s =  l[l.index('(') + 1:l.index(')')]
    else:
        s += "    > ERROR 99: Command error in instruccion " + PRE + command
    return s
```

POC 3 – PARSE CONTACTS TO VCF

Posibles almacenamientos de contactos y llamadas:

SM	SIM
DC	Dialed Calls
MC	Missed Call list
RC	Received call list
ME	Mobile equipment phonebook
FD	SIM fixdialling phonebook
ON	MSISDN list
LD	Lastnumber dialed phonebook
EN	Emergency numbers

POC 3 – PARSE CONTACTS TO VCF

2.2.- Solicitar información del almacenamiento:

```
def get_storage_info():
    '''return a clean a string with storage,number contacts, total space'''
    command = "+CPBS"
    POST = "?"
    phone.write(str.encode(PRE + command + POST + '\r',"utf-8"))
    response = bytes.decode(phone.readall())
    s = ""
    if "OK" in response:
        lines = response.replace('\r',"").split('\n')
        for l in lines:
            if command + ": " in l:
                #print (hexa(i))
                s =  l[l.index(':') + 2:].strip()
    else:
        s += "  > ERROR 99: Command error in instruccion " + PRE + command
    return s
```

Nombre, numero de registros, tamaño

POC 3 – PARSE CONTACTS TO VCF

2.3.- Conseguir los contactos:

```
def get_info(command):
    phone.write(str.encode(PRE + command + '\r',"utf-8"))
    response = bytes.decode(phone.readall())
    s = ""
    if "OK" in response:
        lines = response.replace('\r',"").split('\n')
        for l in lines:
            if ("OK" not in l) and (l != "") and (command not in l):
                #print (hexa(i))
                s += '\n' + l
    else:
        s += "  > ERROR 99: Command error in instruccion " + PRE + command
    return s.strip()
```

Limpieza de los registros

POC 3 – PARSE CONTACTS TO VCF

2.4.- Parsear y guardar a fichero de tarjetas de visitas:

```
def export2vcf(lista,storage):
    '''export list to a vcf 2.1 file format'''
    out = storage.replace("\"","") + time.strftime("%Y%m%d_%H%M%S" + ".vcf")
    s = ''
    for i in lista.split("\n"):
        if "+CPBR:" in i:
            r = i.replace("\"","").split(",")
            print(r)
            s = s + "BEGIN:VCARD\nVERSION:2.1\n"
            s = s + "N:" + r[3] + "\n"
            s = s + "FN:" + r[3] + "\n"
            s = s + "TEL;type=CELL:" + r[1] + "\n"
            s = s + "END:VCARD\n";
    try:
        f = open(out,"w")
        f.write(s)
    finally:
        f.close()
    return out
```

Version 2.1 de vcf compatible con Windows, Outlook, Android, iOS Blackberry, Nokia old, Symbian,.

SMS

Values that May be Assigned to the Parameters of the +CPMS AT Command

Here are the values defined in the SMS specification that may be assigned to the parameters *message_storage1*, *message_storage2* and *message_storage3*:

- **SM**. It refers to the message storage area on the SIM card.
- **ME**. It refers to the message storage area on the GSM/GPRS modem or mobile phone. Usually its storage space is larger than that of the message storage area on the SIM card.
- **MT**. It refers to all message storage areas associated with the GSM/GPRS modem or mobile phone. For example, suppose a mobile phone can access two message storage areas: "SM" and "ME". The "MT" message storage area refers to the "SM" message storage area and the "ME" message storage area combined together.
- **BM**. It refers to the broadcast message storage area. It is used to store cell broadcast messages.
- **SR**. It refers to the status report message storage area. It is used to store status reports.
- **TA**. It refers to the terminal adaptor message storage area.

CONCLUSIONES

- Sencillo de implementar con Python
- Los AT commands los podemos encontrar en la ROM

- Usaremos el patrón: AT[+*!@#$%^&] con strings

LÍNEAS FUTURAS

- Estudiar las vulnerabilidades de ATCommands
- Parsear el resto de contenedores que puedan ser de nuestro interés
- Funcionamiento en Android de los ATCommands

REFERENCIAS

- https://atcommands.org/sec18-tian.pdf
- https://lastminuteengineers.com/a6-gsm-gprs-module-arduino-tutorial/
- https://m2msupport.net/m2msupport/at-commands-to-get-device-information/
- https://doc.qt.io/archives/qtopia4.3/atcommands.html
- https://m2msupport.net/m2msupport/atcpbs-select-phonebook-memory-storage/

PREGUNTAS