General notes about exploiting Windows x64

Sebastian Fernandez
sebastian@immunityinc.com

IMMUNITY Security Research

Who am I?

- Security researcher at Immunity Inc.
  - Exploit development for CANVAS
  - Ported many parts of CANVAS to Windows x64 (shellcodes, backdoors and other “things”)
  - Researching x64 exploitation techniques

x64, what are you talking about?

- x64 (formally x86_64) is an architecture extending the 32bit x86 arch with more registers, instructions and memory range
- Most of the PCs sold over the last few years are based on this arch so most likely your computer supports x64 OSs
- Most software companies have ported their operating system to the platform. Microsoft also did it!
  - Windows XP, 2003, Vista, 2008 and 7 have ports for this arch

No, not really

Why research x64?

- Kernel works entirely on 64 bits.
- Remote/Local exploitation of services.
- Most likely new bugs have been introduced while porting the system.

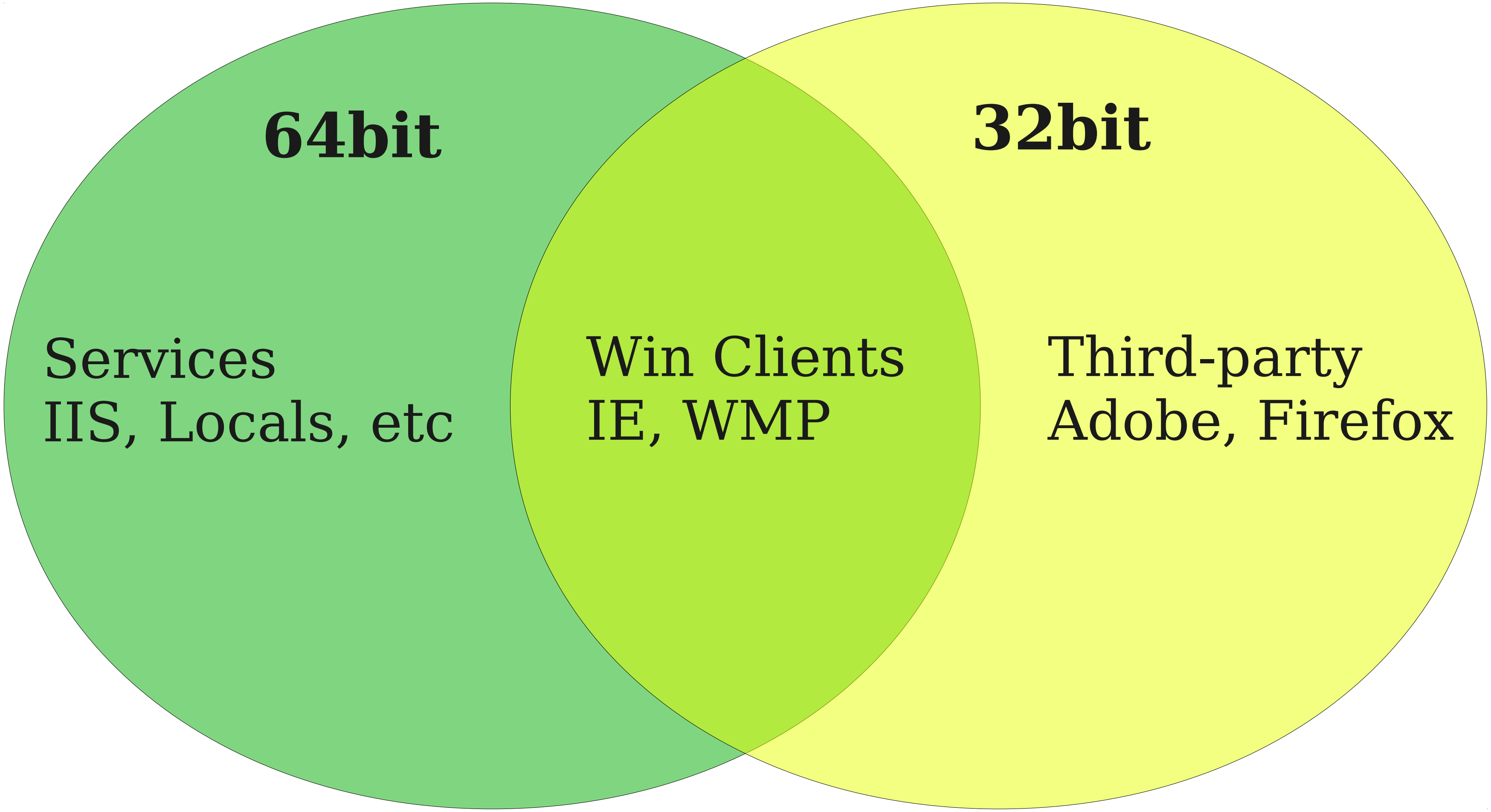
Clientside on 64bit age

- IE is not default, but still available to use.
- When Adobe launches 64bit Flash version in their next major release, IE x64 could become default.

Windows 64

- Services run in 64bits.
- Most applications still don't do it.
- IE and WMP are ported to x64, but by default are launched the 32bit ones.

Windows Applications



64bit

32bit

Services
IIS, Locals, etc

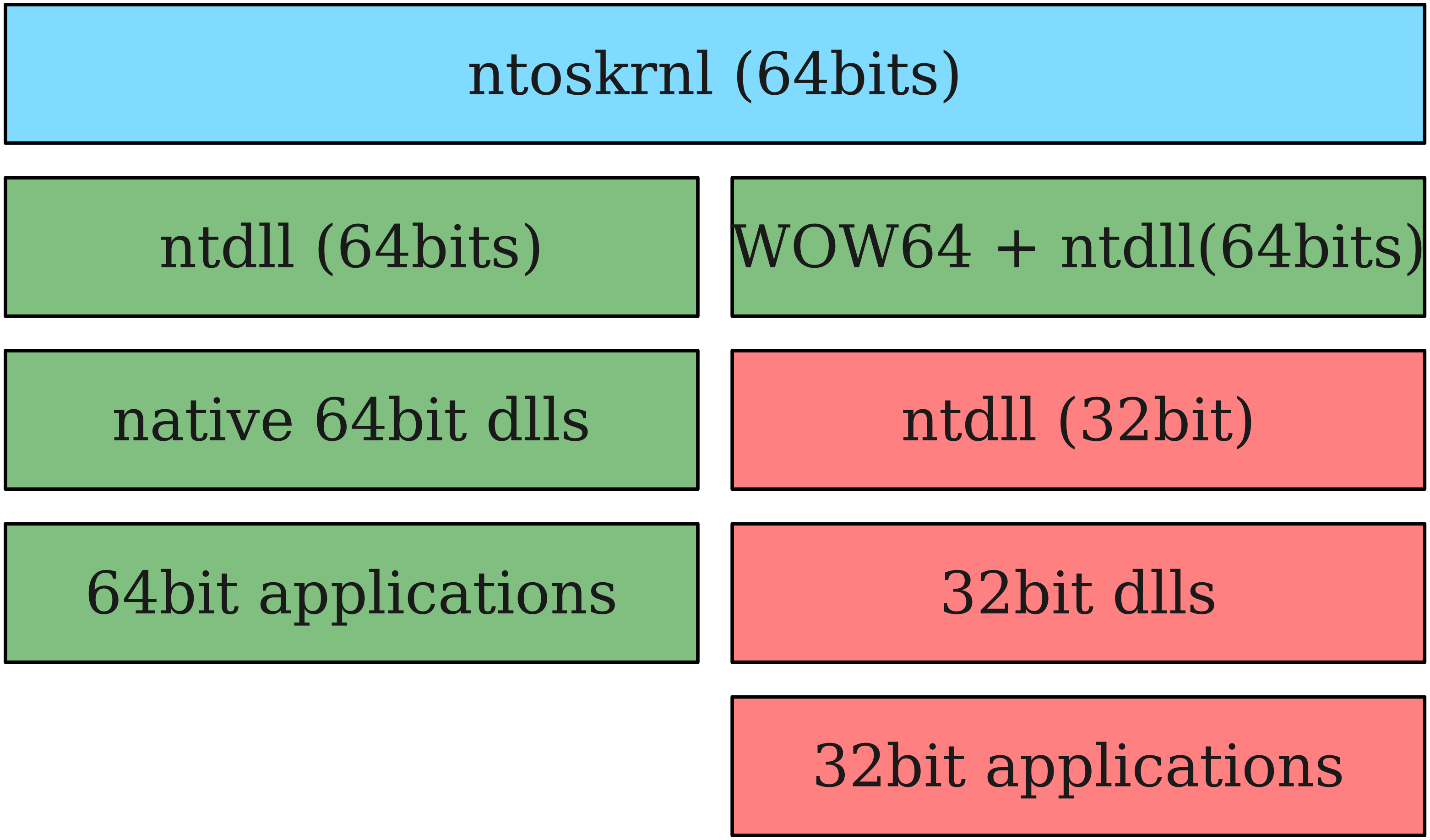
Win Clients
IE, WMP

Third-party
Adobe, Firefox

Windows 64 internals

- Native 64bit with support for 32bit applications using wow64 subsystem.
- No more Ntvdm, 16 bit applications are unsupported.
- Fastcall calling convention.

Windows 64



ntoskrnl (64bits)

ntdll (64bits)

WOW64 + ntdll(64bits)

native 64bit dlls

ntdll (32bit)

64bit applications

32bit dlls

32bit applications

WOW64

- Windows on Windows 64:
  - Abstraction layer to run 32 bit applications on 64bit OS.
  - Patch many ntdll functions for sycall compatibility.
  - Redirect registry access.
  - Environment variables.
  - Switch context to 32bits.

WOW64

ProcessInit

```
....
.text:0000000078BE73C3                call    MapNtdll32
....
```

MapNtdll32:

.....

Loads ntdll from windows/syswow64/

```
.....
.text:0000000078BE7E7D                                        ; MapNtdll32+200j
.text:0000000078BE7E7D                mov     cs:NtDll32Base, ebp
.text:0000000078BE7E83                mov     [rsp+518h+var_498], rbp
.text:0000000078BE7E8B                mov     [rsp+518h+var_490], rbp
.....
.text:0000000078BE7FAE
.text:0000000078BE7FAE loc_78BE7FAE:                          ; CODE XREF: MapNtdll32+334j
.text:0000000078BE7FAE                mov     eax, dword ptr [rsp+518h+var_498]
.text:0000000078BE7FB5                mov     cs:NtDll32Base, eax
.text:0000000078BE7FBB                mov     eax, ds:7FFE0334h
.text:0000000078BE7FC2                mov     cs:Ntdll32LoaderInitRoutine, eax
.text:0000000078BE7FC8                mov     eax, ds:7FFE0338h
.text:0000000078BE7FCF                mov     cs:Ntdll32KiUserExceptionDispatcher, eax
.text:0000000078BE7FD5                mov     eax, ds:7FFE033Ch
.text:0000000078BE7FDC                mov     cs:Ntdll32KiUserApcDispatcher, eax
.text:0000000078BE7FE2                mov     eax, ds:7FFE0340h
.text:0000000078BE7FE9                mov     cs:Ntdll32KiUserCallbackDispatcher, eax
.text:0000000078BE7FEF                mov     eax, ds:7FFE0344h
.text:0000000078BE7FF6                mov     cs:dword_78C1FD98, eax
....
```

Stdcall calling convention

- Each argument is pushed into the stack right-to-left.
- Ret value is on eax.
- Stack aligned to 32 bits.
- Calle cleans stack.

Fastcall Calling convention

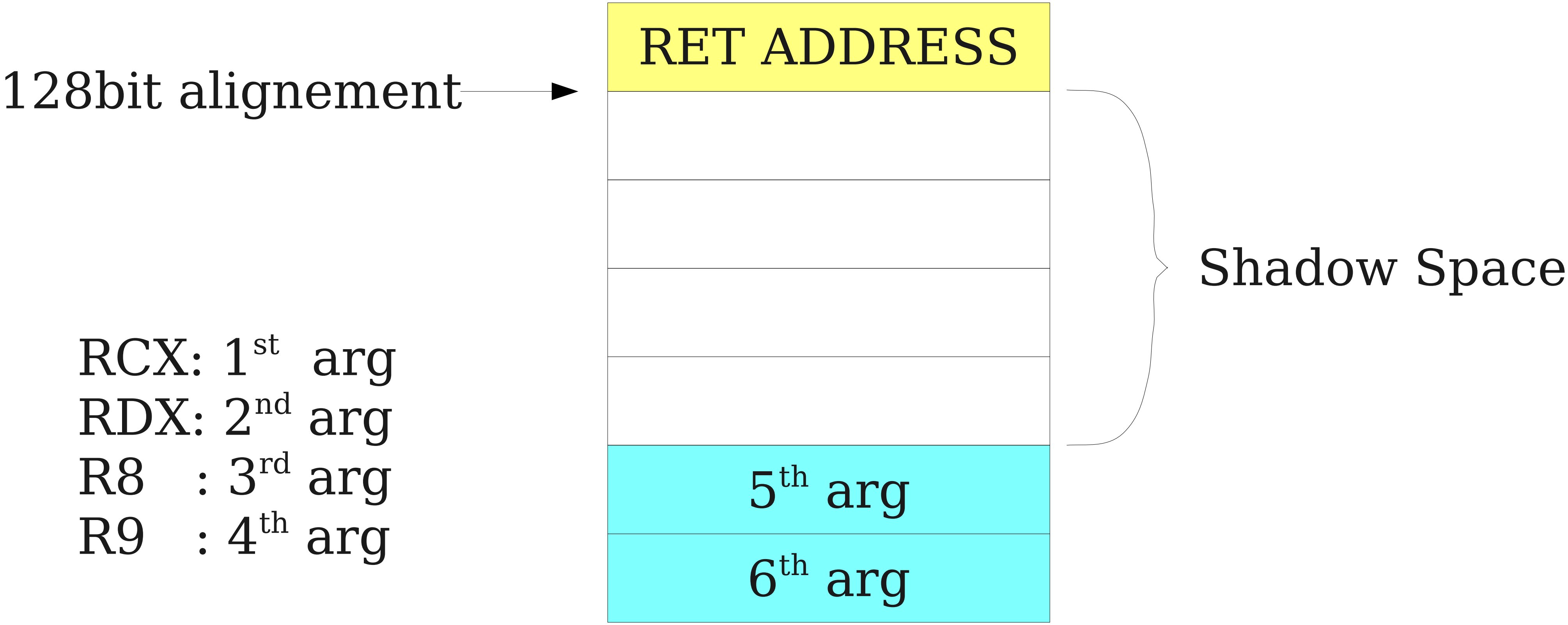
- First 4 arguments are passed in RCX, RDX, R8 and R9.
- The rest of the arguments are pushed in the stack.
- Shadow space must be added in the stack for arguments that have been passed.
- 128 bit stack alignment.

After a call on stdcall

```
int function(arg1,arg2,arg3,arg4,arg5,arg6);
....
    push arg6
    push arg5
    push arg4
    push arg3
    push arg2
    push arg1
    call function
....
```

RET ADDRESS
1st arg
2nd arg
3rd arg
4th arg
5th arg
6th arg

After a call on fastcall



RCX: 1st arg
RDX: 2nd arg
R8 : 3rd arg
R9 : 4th arg

Calling convention

- Shellcoding is easier, less usage of the stack.
- Harder to make ret2libc exploits.

Shellcoding

Shellcode 32bits on Win64

- Can detect WOW64 environment using IsWow64Process function.
- Be aware of not using direct syscalls.
- Other things are basically the same as wow64 sets a friendly environment for running almost every 32bit code.

Shellcodes 64bits on Win64

- Much cleaner since x64 arch let reference RIP (instruction pointer).
- Don't need to use stack (usually), but be aware of 128-bit alignement and shadow space.
- Smaller size of shellcodes because arguments are maintained in registers and half of them are restored by calling functions.

x86 referencing

```
shellcode_init:
    jmp get_str
return_str:
    pop ebx                     ;get address from the stack
...
...
get_str:
    call return_str
    .string “c:\calc.exe”
```

Ugly code

Everybody writes ugly code

```
char *str = “string”;
char *new_str = strcpy(malloc(strlen(str)+1), str);
```

But….

x64 referencing

You don't feel as ugly when writing shellcodes for x64.

```
init_shellcode:
    lea rcx, qword ptr[rel the_str]   ;reference address
...                                   ;using RIP as base.
...
the_str:
    .string “c:\calc.exe”
```

Exploiting

Problems when exploiting

- “Classic” security measures: ASLR, DEP, stack and heap protections.
- All addresses contain at least 2 zero-bytes.
- Calling convention.

ASLR

- Microsoft first implemented it on Windows Vista
- Application/module needs base-dynamic flag to be set at compilation time
- Always enabled on system services
- IE has enabled full ASLR since version 8

Defeating ASLR

- Search for non address-randomized modules.
- No common technique.
- We need an info leak per exploit to defeat data randomization.
- IE8 gives us the opportunity to guess the base address 2 times before warning that someone is hacking us :) .

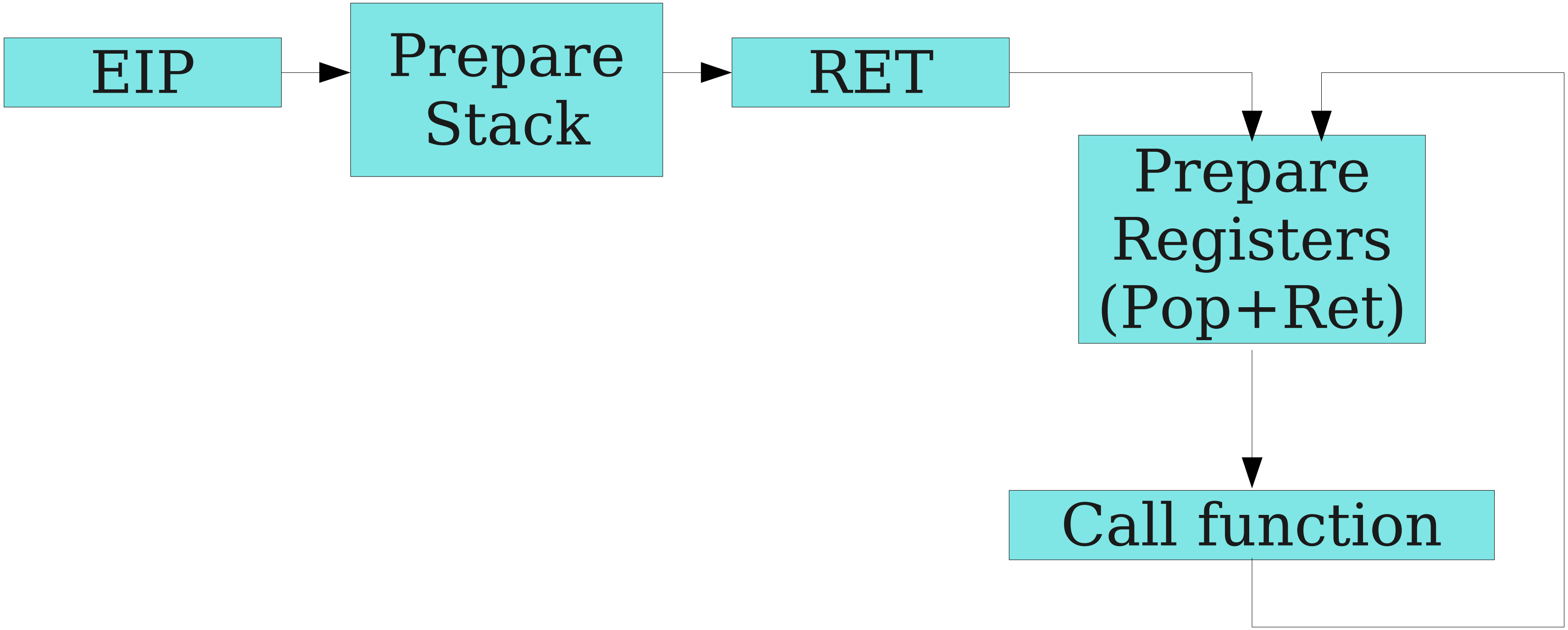
DEP

- No executable data sections (stack, heap, etc).
- No direct ret2libc because of calling convention.
- DEP is enabled automatically on all 64bit applications.

DEP bypass

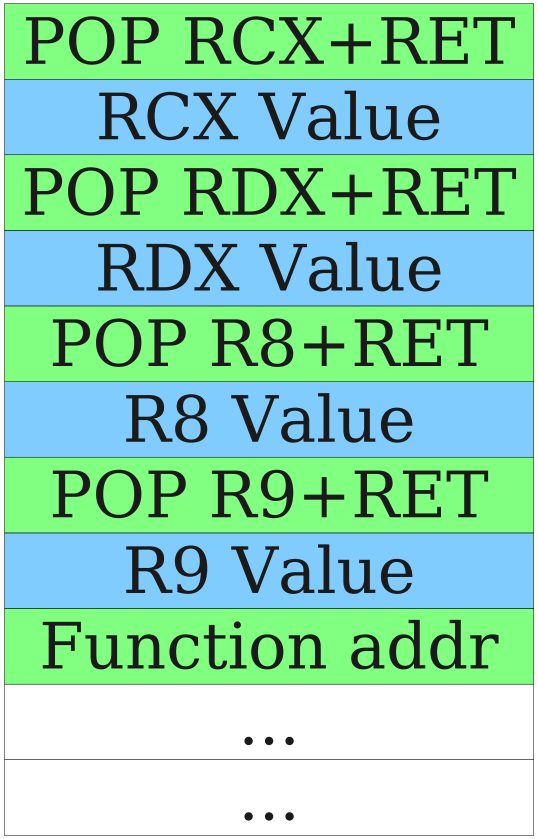
- Build stack with addresses and arguments.
- Use ROP to pop arguments from the stack:
  - POP+RET multiple times
  - POP+Trash_Code+RET
  - Other ways to assign the data in the stack
- Ret2libc.

DEP bypass: ROP



Dep bypass: ROP

Top stack when EIP pointing to a RET instruction.

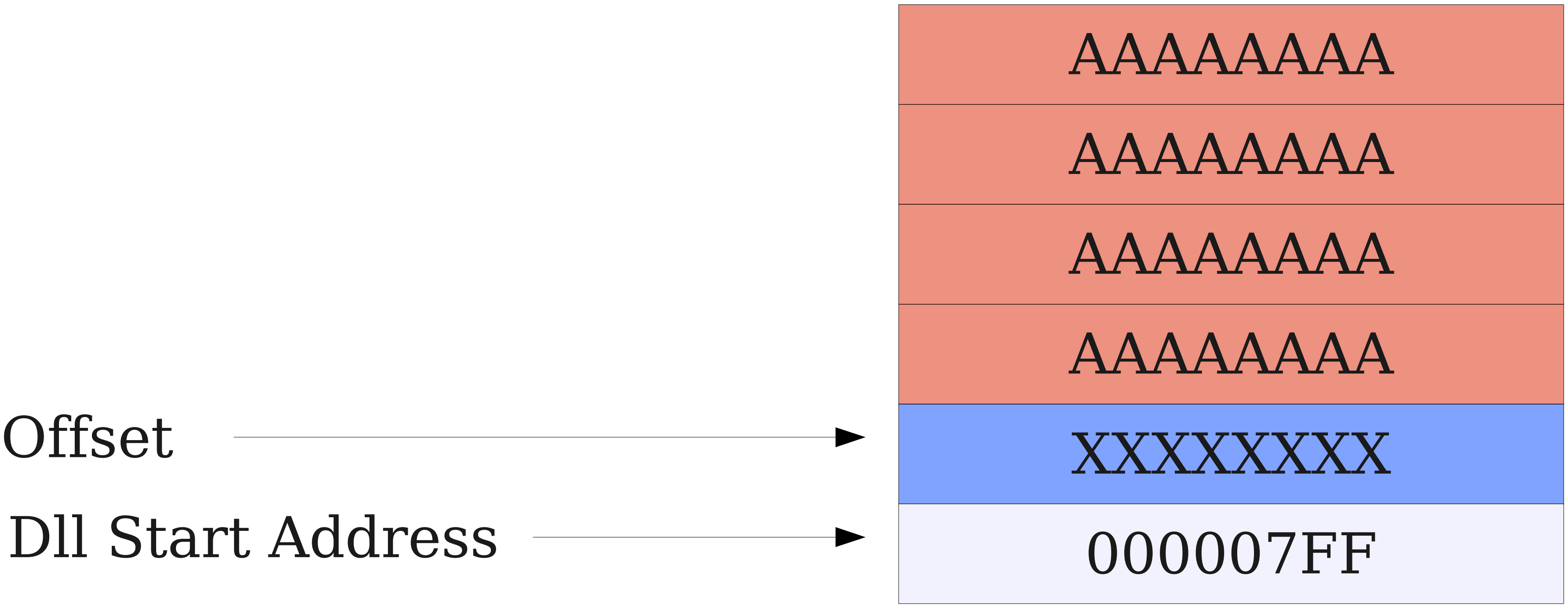


POP RCX+RET
RCX Value
POP RDX+RET
RDX Value
POP R8+RET
R8 Value
POP R9+RET
R9 Value
Function addr
...
...

2 zero-bytes on addresses

- Typical dll base address: 000007FF:XXXXXXXX
- Implies a NULL unicode char
  - Will prevent any wstrcpy/strcpy from being completed
  - On clientside exploits when converting from BSTR to Cstrings, it will cut down the string to the first null

Overwrite less significant bytes



AAAAAAAA

AAAAAAAA

AAAAAAAA

AAAAAAAA

Offset → XXXXXXXX

Dll Start Address → 000007FF

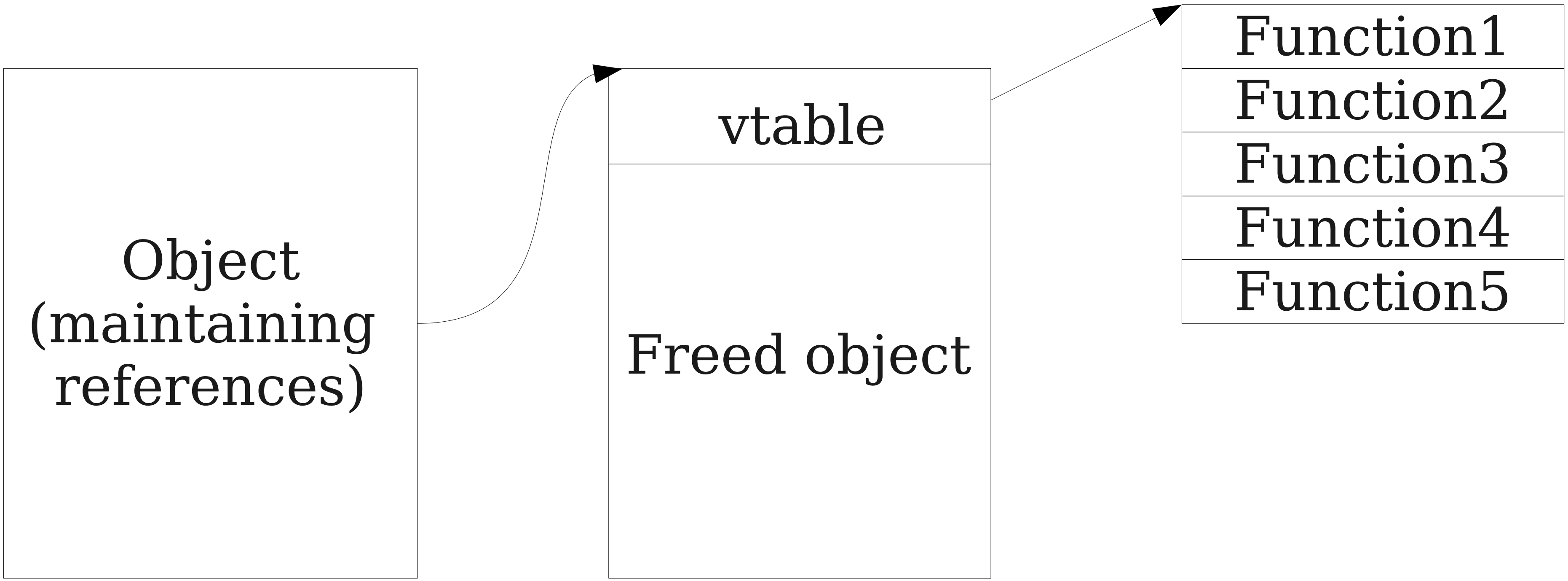
Client side use-after-free

- Very common vulnerability:
  - Aurora (ms10_002)
  - iepeers_set_attribute (ms10_018)
  - CfunctionPointer (ms09_002)
- Exploited replacing freed objects maintaining references to them.

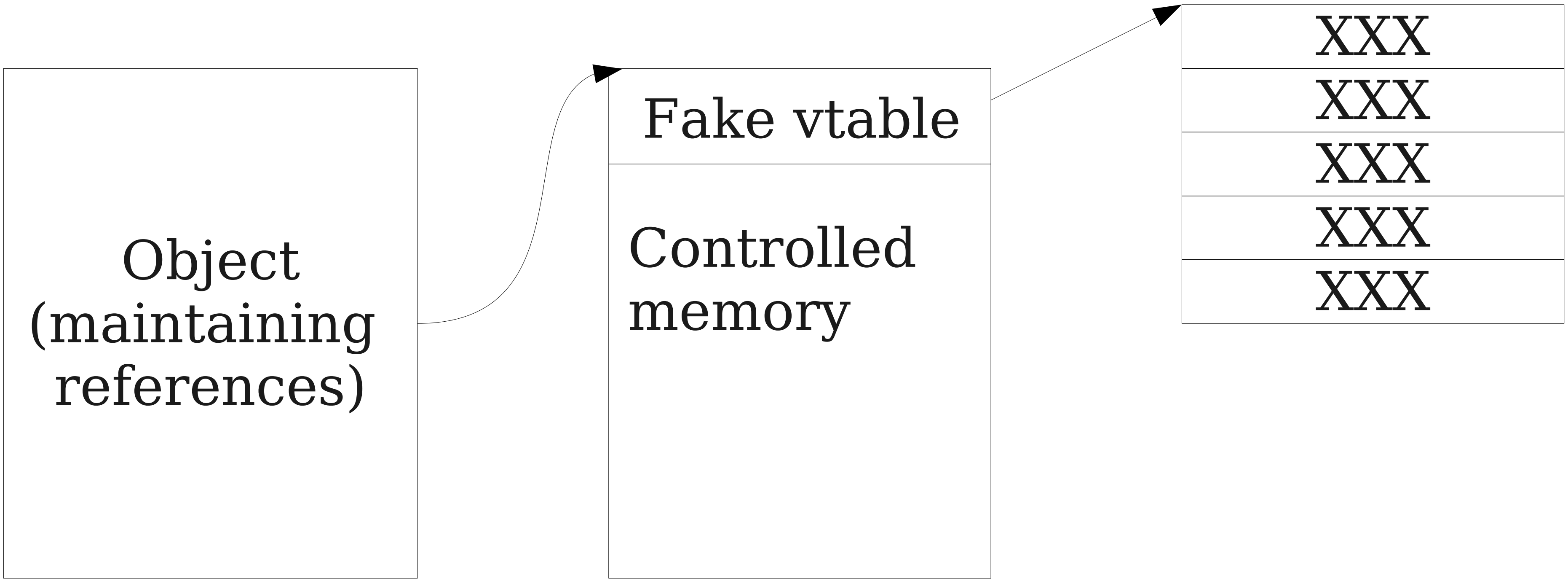
Client side use-after-free

```
;function referencing from an object
;our object is on rcx
mov     rdx,qword ptr [rcx]      ;get vtable
call    qword ptr [rdx+8]        ;call the function
                                    ;from the vtable
```

Client side use-after-free



Client side use-after-free



Object (maintaining references)

Fake vtable

Controlled memory

XXX

XXX

XXX

XXX

XXX

Use-after-free (x86)

- Transform javascript strings to Cstrings for filling vtable.
  - UnicodeStr(unescape(“%u0d0d %u0d0d...”))
  - cstring = “\x0d\x0d\x0d\x0d\x..\x..\x00\x00”
- Use heap spray techniques to create the vtable functions in memory and align it.

Use-after-free(x64)

- There is no way to transform javascript strings cointaining nulls in Cstrings:
  - UnicodeStr(unescape(“%u0d0d%u0d0d %u0000%u0000…”))
  - cstring = “\x0d\x0d\x0d\x0d\x00\x00”
- **Need to load binary data in memory to replace the freed objects.**
- Heap spray to create functions in memory (using conventional heap spray).

Tools for Windows x64

- Windbg.
- WinAppDbg.
- MOSDEF x64.
- IDA64 + IDAPython64
- Next... Immunity Debugger.

The Future

- Look for more interesting bug classes in ported applications
- Next Windows version release will run all the 64bit applications default.
  - Those who don’t ramp up now will be left behind!

Questions?

Thank you for your time

Contact me at:

sebastian@immunityinc.com