

OSEP (Offensive Security Evasion Professional) Notes Overview

PT 1 by Joas



Sumário

Details	3
Laboratory.....	4
Programming Language.....	4
Managed Code	5
Java.....	6
C#.....	7
Assembly language.....	7
Opcode	8
HTML Smuggling.....	9
Office Phishing.....	13
Shellcode Run	34
A Beginner's Guide to Windows Shellcode Execution Techniques.....	34
Shellcode: In-Memory Execution of DLL	43
Running ShellCode in Memory AV Evasion – VBA Version.....	55
Execute Code in a Microsoft Word Document Without Security Warnings	59
AV Evasion Part 2, The disk is lava	66
Powershell Commands.....	71
NATIVE POWERSHELL X86 SHELLCODE INJECTION ON 64-BIT PLATFORMS	71
Low-Level Windows API Access From PowerShell	78
Malicious Office Documents: Multiple Ways to Deliver Payloads	104
POWERSHELL SCRIPTS USED TO RUN MALICIOUS SHELLCODE. REVERSE SHELL VS BIND SHELL	111
JScripT Dropper	119
JScripT Meterpreter.....	119
Payload Delivery for DevOps : Building a Cross-Platform Dropper Using the Genesis Framework, Metasploit and Docker	120
Donut v0.9.2 "Bear Claw" - JScript/VBScript/XSL/PE Shellcode and Python Bindings.....	127
Shellcode: In-Memory Execution of JavaScript, VBScript, JScript and XSL	133
Process Injection Techniques	156
DLL Injection	174
Reflective DLL Injection	182
SharpShooter.....	191
Process Injection	198
Process Hollowing in C#	203
DISCOVERING THE ANTI-VIRUS SIGNATURE AND BYPASSING IT	214

Bypass Antivirus with Metasploit.....	220
MSFEncode.....	227
MSFVenom.....	227
MSFEncrypt	235
AV Bypass Custom Binaries, Veil Evasion and Meterpreter Payload	235
AV Bypass with C# Runner	236
Creating Simple Backdoor Payload by C#.NET	236
Making Encrypted Meterpreter Payload by C#.NET	256
VBA Bypass AV	288
Shellcodes and bypass Antivirus using MacroPack	300
Offensive VBA.....	304
Injection Cobalt Strike Beacon from Office.....	308
AMSI Bypass	313
AMSI Concept.....	313
AMSI Bypass Methods.....	314
Bypass AMSI with powershell	326
Memory Patching AMSI Bypass.....	347
Exploring PowerShell AMSI and Logging Evasion.....	353

Details

This PDF is intended to help those studying for OSEP or seeking resources on Dropout.

All credits for the materials sent have been duly placed.

This is just part 1 as the content would be too extensive

Laboratory

Machines

- | | | | |
|-----------|-------------|-----------------|------------|
| • Legacy | • Sneaky | • Ariekei | • Teacher |
| • Lame | • Haircut | • LaCasadePapel | • FluJab |
| • Optimum | • Shocker | • CrimeStopper | • Waldo |
| • Beep | • Cronos | • Querier | • Hawk |
| • Bastard | • Dropzone | • Zipper | • RedCross |
| • Arctic | • Sunday | • Bart | |
| • Tenten | • Bank | • Tally | |
| • Grandpa | • Popcorn | • Rabbit | |
| • October | • Node | • Access | |
| • Granny | • Brainfuck | • Sizzle | |
| • Charon | • Nineveh | • Giddy | |
| • Lazy | • Mirai | • Reel | |
| • Devel | • Nightmare | • SecNotes | |

<https://www.hackthebox.eu/>

Programming Language

x86 is a family of [complex instruction set computer](#) (CISC) [instruction set architectures](#)^[a] initially developed by [Intel](#) based on the [Intel 8086 microprocessor](#) and its [8088](#) variant. The 8086 was introduced in 1978 as a fully [16-bit](#) extension of Intel's [8-bit 8080](#) microprocessor, with [memory segmentation](#) as a solution for addressing more memory than can be covered by a plain 16-bit address. The term "x86" came into being because the names of several successors to Intel's 8086 processor end in "86", including the [80186](#), [80286](#), [80386](#) and [80486](#) processors.

The term is not synonymous with [IBM PC compatibility](#), as this implies a multitude of other [computer hardware](#). [Embedded systems](#) and general-purpose computers used x86 chips [before the PC-compatible market started](#),^[b] some of them before the [IBM PC](#) (1981) debut.

As of 2022, most [desktop computers](#), [laptops](#) and [game consoles](#) (with the exception of the [Nintendo Switch](#)^[2]) sold are based on the x86 architecture family,^[citation needed] while mobile categories such as [smartphones](#) or [tablets](#) are dominated by [ARM](#); at the high end, x86 continues to dominate compute-intensive [workstation](#) and [cloud computing](#) segments,^[3] while the [fastest](#) supercomputer in 2020 was ARM-based, with the top 4 no longer x86-based in that year.^[4]

In the 1980s and early 1990s, when the [8088](#) and [80286](#) were still in common use, the term x86 usually represented any 8086-compatible CPU. Today, however, x86 usually implies a binary compatibility also with the [32-bit instruction set](#) of the 80386. This is due to the fact that this instruction set has become something of a lowest common denominator for many modern operating systems and probably also because the term became common *after* the introduction of the [80386](#) in 1985.

A few years after the introduction of the 8086 and 8088, Intel added some complexity to its naming scheme and terminology as the "iAPX" of the ambitious but ill-fated [Intel iAPX 432](#) processor was tried on the more successful 8086 family of chips,^[c] applied as a kind of

system-level prefix. An 8086 *system*, including [coprocessors](#) such as [8087](#) and [8089](#), and simpler Intel-specific system chips,^[d] was thereby described as an iAPX 86 *system*.^{[5][e]} There were also terms *iRMX* (for operating systems), *iSBC* (for single-board computers), and *iSBX* (for multimodule boards based on the 8086-architecture), all together under the heading *Microsystem 80*.^{[6][7]} However, this naming scheme was quite temporary, lasting for a few years during the early 1980s.^[f]

Although the 8086 was primarily developed for [embedded systems](#) and small multi-user or single-user computers, largely as a response to the successful 8080-compatible [Zilog Z80](#),^[8] the x86 line soon grew in features and processing power. Today, x86 is ubiquitous in both stationary and portable personal computers, and is also used in [midrange computers](#), [workstations](#), servers, and most new [supercomputer clusters](#) of the [TOP500](#) list. A large amount of [software](#), including a large list of [x86 operating systems](#) are using x86-based hardware.

Modern x86 is relatively uncommon in [embedded systems](#), however, and small [low power](#) applications (using tiny batteries), and low-cost microprocessor markets, such as [home appliances](#) and toys, lack significant x86 presence.^[g] Simple 8- and 16-bit based architectures are common here, although the x86-compatible [VIA C7](#), [VIA Nano](#), [AMD's Geode](#), [Athlon Neo](#) and [Intel Atom](#) are examples of 32- and [64-bit](#) designs used in some *relatively* low-power and low-cost segments.

There have been several attempts, including by Intel, to end the market dominance of the "inelegant" x86 architecture designed directly from the first simple 8-bit microprocessors. Examples of this are the [iAPX 432](#) (a project originally named the *Intel 8800*^[9]), the [Intel 960](#), [Intel 860](#) and the Intel/Hewlett-Packard [Itanium](#) architecture. However, the continuous refinement of x86 [microarchitectures](#), [circuitry](#) and [semiconductor manufacturing](#) would make it hard to replace x86 in many segments. AMD's 64-bit extension of x86 (which Intel eventually responded to with a compatible design)^[10] and the scalability of x86 chips in the form of modern multi-core CPUs, is underlining x86 as an example of how continuous refinement of established industry standards can resist the competition from completely new architectures.

[x86 - Wikipedia](#)

Managed Code

Managed code is computer program code that requires and will execute only under the management of a [Common Language Infrastructure](#) (CLI); [Virtual Execution System](#) (VES); [virtual machine](#), e.g. [.NET](#), [CoreFX](#), or [.NET Framework](#); [Common Language Runtime](#) (CLR); or [Mono](#). The term was coined by [Microsoft](#).

Managed code is the compiler output of [source code](#) written in one of over twenty high-level [programming languages](#), including [C#](#), [J#](#) and [Visual Basic .NET](#).

The distinction between managed and unmanaged code is prevalent and only relevant when developing applications that interact with CLR implementations. Since many^[which?] older programming languages have been ported to the CLR, the differentiation is needed to identify managed code, especially in a mixed setup. In this context, code that does not rely on the CLR is termed "unmanaged".

A source of confusion was created when Microsoft started connecting the .NET Framework with [C++](#), and the choice of how to name the [Managed Extensions for C++](#). It was first named