Section A – Personal Data:

a. **Name**: Alex Ionescu

b. **Handler/Alias**: @aionescu

c. **Email Address**: aionescu@gmail.com <mailto:aionescu@gmail.com>

d. **Company**: Winsider Seminars & Solutions, Inc.

e. Brief biography:

Alex Ionescu is the Vice President of EDR Strategy at CrowdStrike, Inc., where he started as its Chief Architect almost six years ago. Alex is a world-class security architect and consultant expert in low-level system software, kernel development, security training, and reverse engineering. He is coauthor of the last three editions of the Windows Internals series, along with Mark Russinovich and David Solomon. His work has led to the fixing of many critical kernel vulnerabilities, as well as over a few dozen non-security bugs. <br> Previously, Alex was the lead kernel developer for ReactOS, an open source Windows clone written from scratch, for which he wrote most of the Windows NT-based subsystems. During his studies in Computer Science, Alex worked at Apple on the iOS kernel, boot loader, and drivers on the original core platform team behind the iPhone, iPad and AppleTV. Alex is also the founder of Winsider Seminars & Solutions Inc., a company that specializes in low- level system software, reverse engineering and security trainings for various institutions.

Section B – Training Class Data

a. Title of Training Class:

WINDOWS INTERNALS FOR REVERSE ENGINEERS (FIVE-DAY)

b. Brief Description of Training Class:

Learn the internals of the Windows NT kernel & system architecture, including Windows 10 and Server 2016, in order to learn how rootkits, PLA implants, NSA backdoors, and other kernel-mode malware exploit the various system functionalities, mechanisms and data structures to do their dirty work. Also learn how drivers operate and how they can be subject to attack from user-mode callers to elevate their privileges. Finally, learn how CPU architecture deeply ties into OS design, and how Intel, ARM and AMD's mistakes can lead to more pwnage.

We'll cover the many new Windows 10 kernel changes, including the introduction of virtualized secure process sandboxes (Trustlets), Hypervisor-Based Security and Code Integrity (Device Guard), mitigations such as Arbitrary Code Guard (ACG), Control Flow Guard (CFG), and Return Flow Guard (RFG), as well as new Container Support (Silos) and Kernel ASLR. New Windows 8.1 functionality covering SecureBoot and Code Signing requirements will also be covered, and new Windows 8 kernel features relevant to driver operation and exploitation techniques will be discussed, including an overview of over two dozen new security mitigations that have been added to the operating system. If the last system you exploited was running Windwos XP -- don't worry: Windows 7 kernel changes will be discussed too, such as the new Object Manager data structures.

After learning the theory, you will use tools such as WinDBG, SysInternals Tools, Process Hacker & more, to analyze, poke, and prod kernel-mode Windows components, as well as write your scripts, NatVis LINQ commands, and even JavaScript debugger scripts.

Throughout the class, we'll focus on using various techniques and tools to inspect the Windows kernel for consistency, tracing its operation, and editing it, as well as ways in which offensive and defensive attackers can mess with the system's state in unexpected, "clean" ways. We'll also take a look at several examples of malicious and/or buggy drivers in a given Windows system, as well as architectural bugs over Windows' lifetime.

Finally, Windows 10, Windows 8.1 and Windows 8 introduced a plethora of new security features, mitigations, and code signing requirements (as well as support for SecureBoot). We'll see how these changes to the architecture have dramatically constrained exploit techniques.

c. Pre-requisite of Training Class:

IMPORTANT: You should be able to understand x86 assembly to take this course, but knowledge of obfuscation, packing, etc, is not required.

Basic knowledge of Windows, processor architecture, and operating systems is helpful -- you should know what an interrupt is, and what is the difference between user and kernel mode (ring levels), a bit about virtual memory/paging, etc.

Minimum Software to install:
You must have a Windows machine to attend, and you should have the Windows Driver Kit (WDK) 10 released for the Creator's Update (Redstone

2), build 15003 or above, which you can freely grab from MSDN. If you really cannot find a way to obtain it, you may contact the conference and a temporary copy for class use will be made available for you.

A virtual machine (VirtualBox is preferred) is recommended with an installed version of Windows 10. Locally, any version of Windows 7 or Windows 8, 32-bit or 64-bit is fine -- but 64-bit is recommended to keep up with the times. You may even use Mac or Windows if you'd like, but you won't be able to setup network debugging unless you use two virtual machines.

The instructor will use a 64-bit Windows 10 device, with Windows 8.1 and Windows 7 32-bit VMs.

Before the course begins, you will receive a helpful information with information on how to prepare for the course -- these are just here to give you a rough idea.

* IDA/Hexrays helpful, but not required.

d. Daily Class Outline:

Part 1: WinDBG Zero to Hero

You will be presented with many WinDBG/KD commands, as well as how to use the new Language INtegrated Query (LINQ) language part of NatVis and how to write your own scripts, both in the native interpreter as well as the new Java-script based extension & scripting interface. You'll spend a good few hours learning WinDBG tricks that will save you hours of time and effort, and turn your WinDBG-fu from white belt to WinDBG sensei. You'll also never call it "Win DEE BEE GEE" again.

Part 2: Windows OS and CPU Architecture

You'll be exposed to low-level CPU details such as segmentation and task gates, things you probably thought didn't matter since 1986 anymore, but which still drive core Windows architecture today. Never again will you think [fs:18h] is a magic constant. Both x86 and x64 will be covered, up to the latest features such as SGX, UMIP, and CET.

We'll then move our focus toward understanding the behaviour and operation of the Windows Kernel. Core topics such as interrupts, DPCs, APCs, timers, scheduling and memory management will be discussed at the architectural level. You are expected to already be familiar with some of these ideas -- for example, on the topic of memory management, it is not about how to use malloc, but rather about the complexities of and effects of the algorithms used by Windows when managing pages.

You will learn the various data structures used by bookkeeping and internal consistency and management of key system components such as the ALPC subsystem that powers RPC, as well as core mechanisms and objects such as threads, processes, and jobs, and how these fields can affect the operation of the system in ways that malicious code can control.

Part 3: Windows Kernel Mode Reverse Engineering, Vulnerability Analysis, and Forensics

You'll learn about the Windows Memory Manager and its many bookkeeping data structures both at the virtual memory and physical memory layer and how they can both be used to identify hidden code at runtime, as well as used forensically for gathering data about an attack. We'll also cover shared memory objects, and the dangers that lie within -- including a few live samples of bugs nobody cares to fix.

Finally, it's not just all about the kernel! You'll also discover how Windows' Subsystem Model affects the execution of code and abstraction of various system components and we'll take a look at the Windows Subsystem specifically, and how past and present CSRSS vulnerabilities can be used to escape sandboxes and attack the kernel. The Windows Subsystem extends its claws into the kernel through the window manager (Win32k.sys) and its data structures/behaviors will also be in the purview of our class. We will be peering into aspects of Win32k.sys that few have looked at before -- except those that find new kernel vulnerabilities in it by the minute. We'll top it off with some forensic spellunking in the configuration manager where the registry lives, as well as the many changes in the user-mode loader data structures.

Finally, we will take a look at certain examples of plain C code, that contains well-known bugs, as well as certain harder-to-spot bugs within the specific context of running in a Windows kernel environment.