Starting with Windows Kernel Exploitation – part 1 – setting up the lab

Posted on May 28, 2017

Recently I started learning Windows Kernel Exploitation, so I decided to share some of my notes in form of a blog.

This part will be about setting up the lab. In further parts I am planning to describe how to do some of the exercises from HackSysExtremeVulnerableDriver by Ashfaq Ansari.

I hope someone will find this useful!

What I use for this part:

- Kali Linux – as a host system (you can use anything you like)
- VirtualBox
- 2 Virtual Machines: Windows 7 32 bit (with VirtualBox Guest Additions installed) – one will be used as a Debugger and another as a Debugee
- WinDbg (you can find it in Windows SDK)

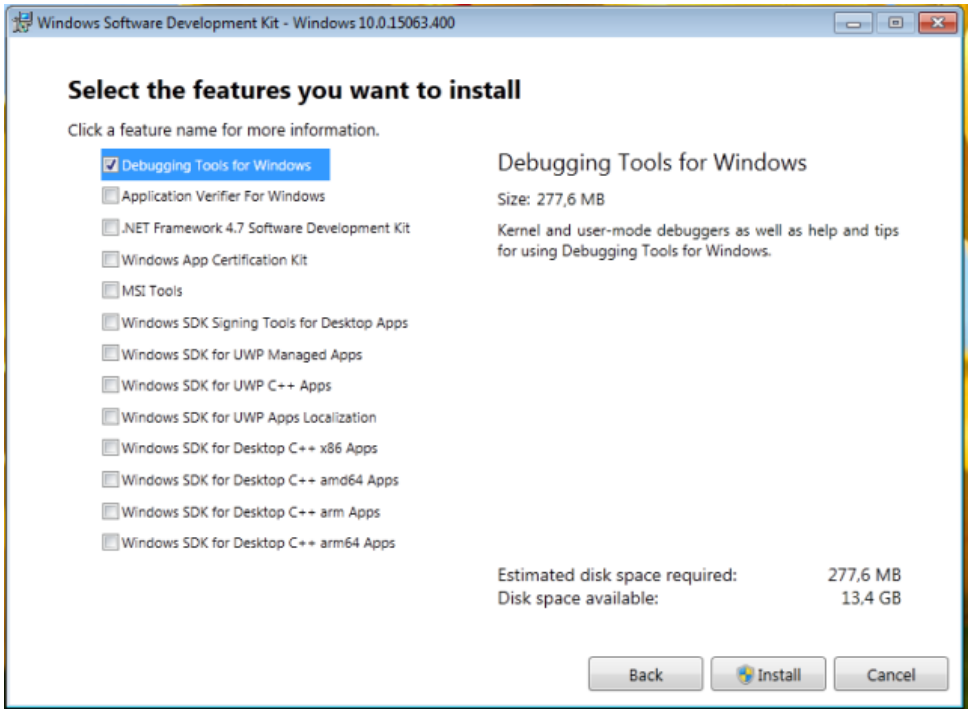
When we do userland debugging, we can have a debugger and a debuggee on the same machine. In case of kernel debugging it is no longer possible – we need a full control over the debugee operating system. Also, when we will interrupt the execution, full operating system will freeze. That's why we need two virtual machines with separate roles.

Setting up the Debugger

Debugger is the machine form where we will be watching the Debugee. That's why, we need to install WinDbg there, along with symbols, that will allow us to interpret system structures.

In order to install WinDbg we need to download Windows SDK (depending on the version of Windows, sometimes we will also need to install some required updates).

It is important to choose Debugging Tools from the installer options:



Once we have WinDbg installed. we should add Symbols. In order to do this, we just need to add an environment variable, to which WinDbg will automatically refer:

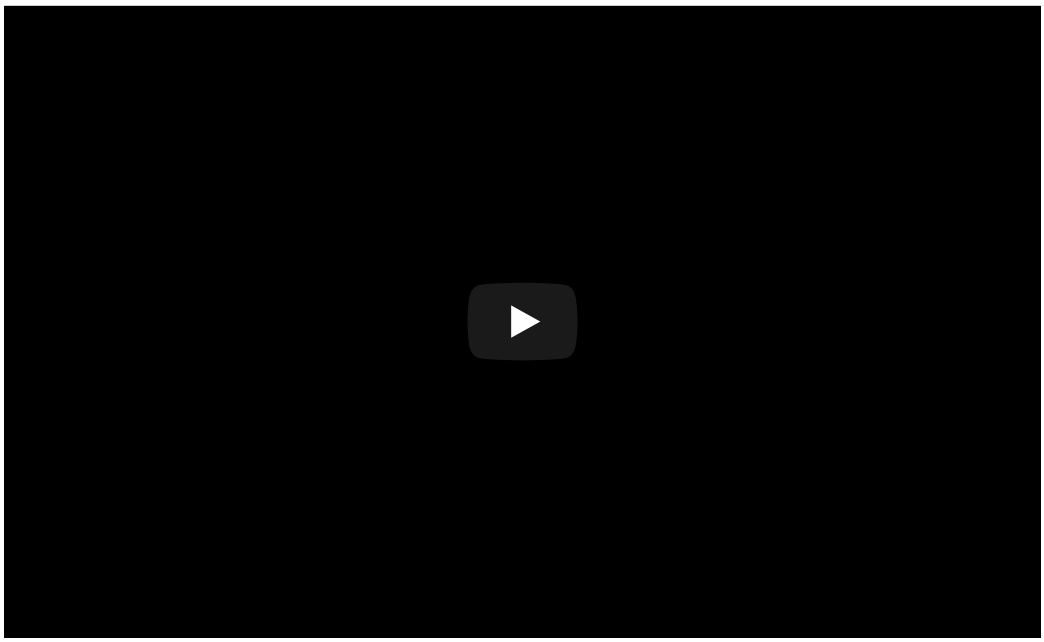
```
_NT_SYMBOL_PATH
```

… and fill it with the link from where it can download symbols.

```
https://msdl.microsoft.com/download/symbols
```

Full variable content may look like this (downloaded symbols will be stored in C:\Symbols):

```
SRV*C:\Symbols*https://msdl.microsoft.com/download/symbols
```



Setting up the Debugee

We need to enable Debugee to let it be controlled from outside. In order to do this, we are adding one more option in a boot menu – if we start the machine with this configuration, it is enabled for debugging.
We need to use a tool bcdedit. First we copy the current settings into a new entry, titled i.e. “Debug me”:

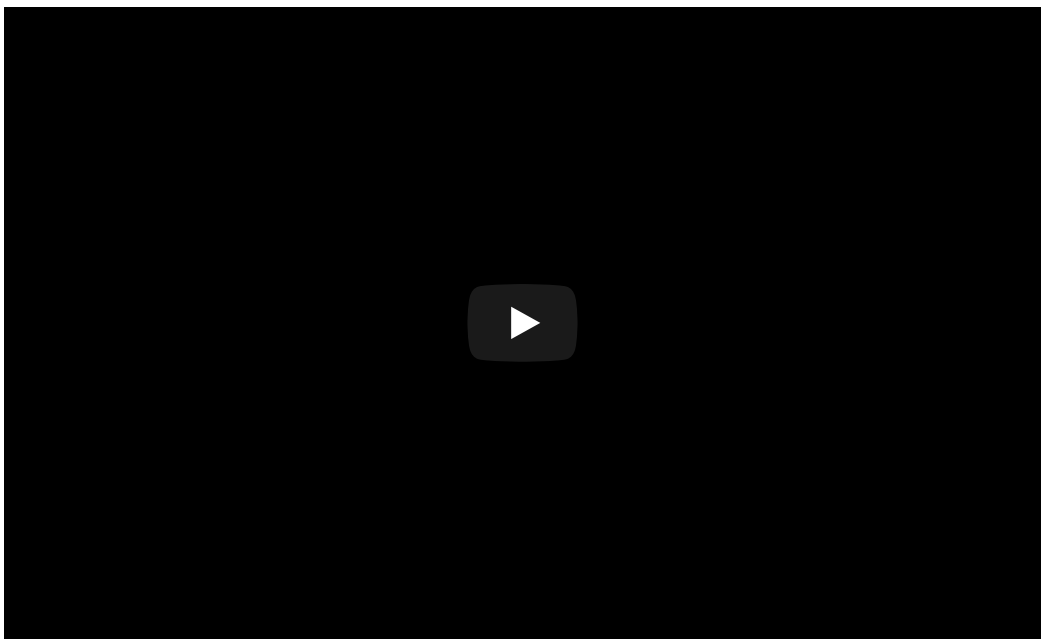
```
bcdedit /copy {current} /d "Debug me"
```

It gives us in return a GUID of the new entry. We need to copy it and use to enable debugging on this entry:

```
bcdedit /debug {MY_GUID} on
```

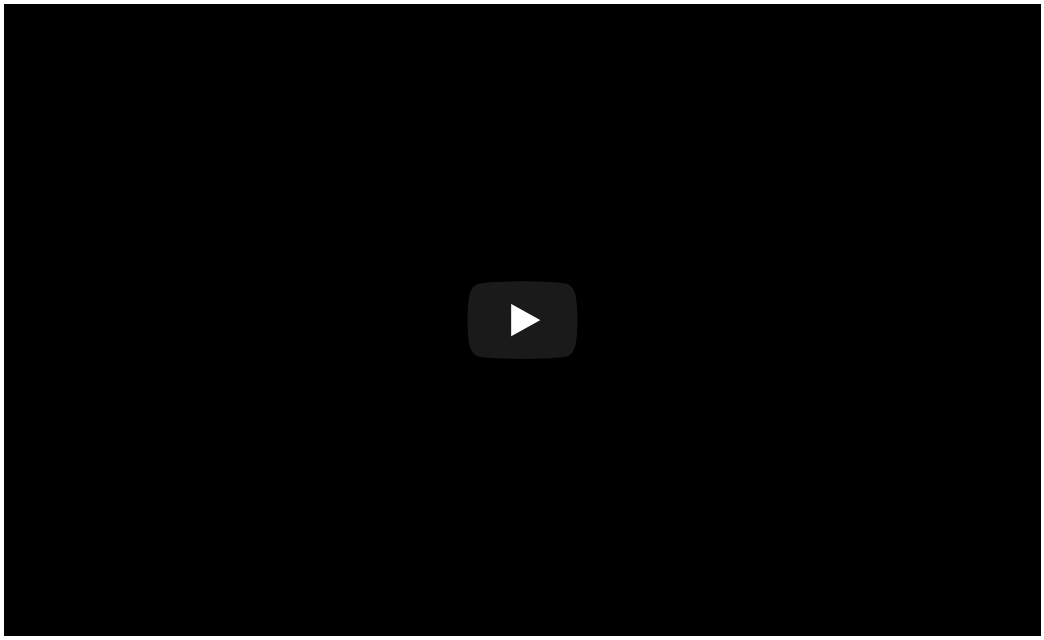
At the end we can see the settings where the debugging interface will be available:

```
bcdedit /dbgsettings
```



Setting up the connection between the Debugger and the Debuggee

Debugger and Debugge will be communicating via Serial Port COM1, that will be emulated in the host system by a Named Pipe. It is very simple to configure, we just have to make sure that the debugger and the debuggee have the same pipe name set. Debugger will be creating the pipe, while the Debuggee will be connecting to the existing one (that’s why we always have to run Debugger first):



I use Linux as my host system, so I chose as a pipe name:

```
/tmp/wke_pipe
```

Note that if you are using Windows as your host system, your pipe name will have to follow different convention. Example:

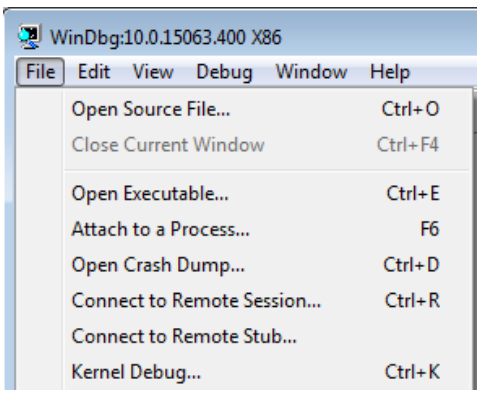
```
\\.\pipe\wke_pipe
```

Read more: https://en.wikipedia.org/wiki/Named_pipe

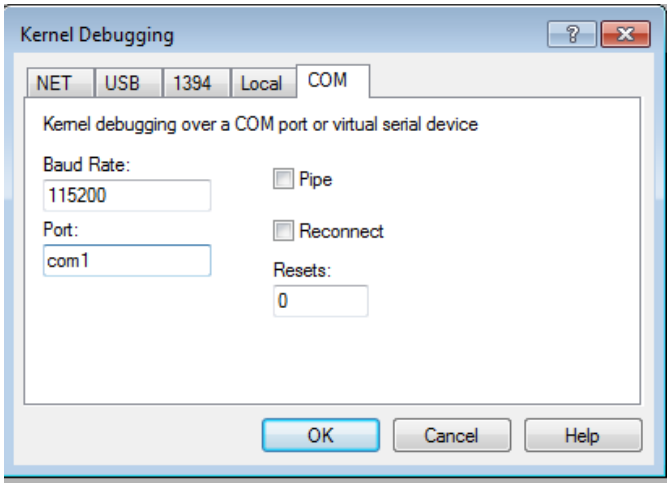
Testing the connection

We have everything set up, now we just need to test if it works correctly! Let's start the Debugger first, run WinDbg, and make it wait for the connection from the Debugee. Example:

File->Kernel Debug

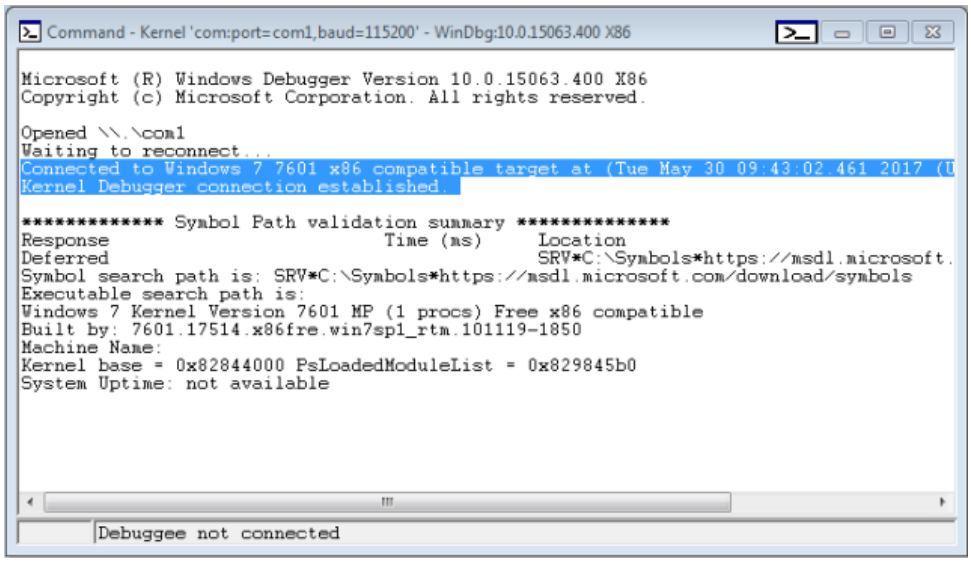


We are choosing COM as an interface:

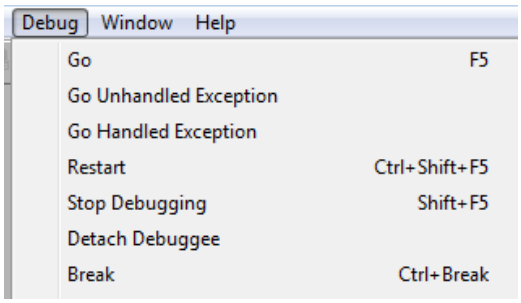


Then we will run the Debugee machine, and when we see that it got connected to the pipe, we will send it interrupt. Example:

The Debugee is connected to the pipe:



Now we can interrupt it, clicking Debug->Break:

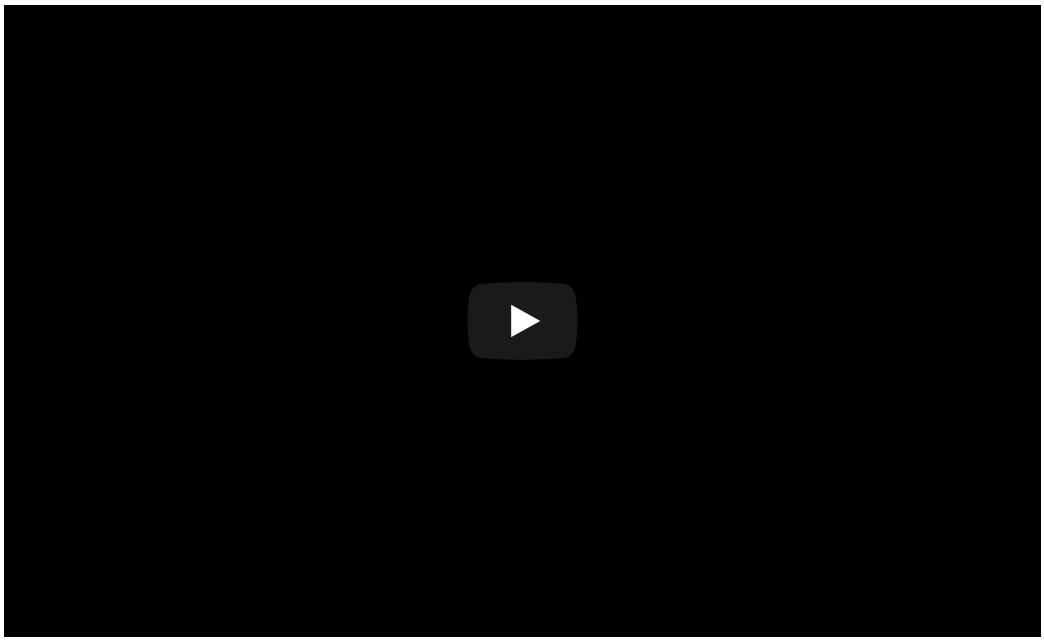


If we get the **kd** prompt, it means we are in control of the Debugee:

```
Break instruction exception - code 80000003 (first chance)
*******************************************************************************
*                                                                             *
*   You are seeing this message because you pressed either                    *
*       CTRL+C (if you run console kernel debugger) or,                       *
*       CTRL+BREAK (if you run GUI kernel debugger),                          *
*   on your debugger machine's keyboard.                                      *
*                                                                             *
*                   THIS IS NOT A BUG OR A SYSTEM CRASH                       *
*                                                                             *
* If you did not intend to break into the debugger, press the "g" key, then   *
* press the "Enter" key now.  This message might immediately reappear.  If it *
* does, press "g" and "Enter" again.                                          *
*                                                                             *
*******************************************************************************
nt!RtlpBreakWithStatusInstruction:
8289cd00 cc              int     3

kd>
```

See the full process on the video:



The Debugee frozen, waiting for the instructions form the Debugger. By a ‘g’ command we can release the Debugee and let it run further:

```
nt!RtlpBreakWithStatusInstruction:
8289cd00 cc              int     3
kd> g

*BUSY*  Debuggee is running...
```

Part 2:
https://hshrzd.wordpress.com/2017/06/05/starting-with-windows-kernel-exploitation-part-2/
