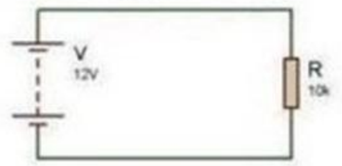
Advance Dynamic Analysis

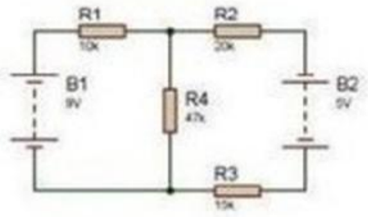
Mohd Zaki Mas’ud

Engineering

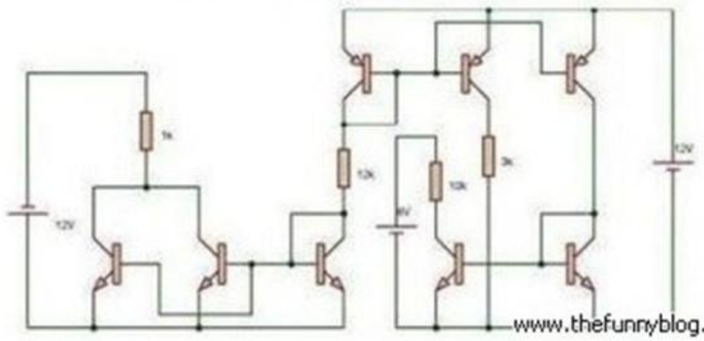
Class:



Homework:



Exam:



- Introduction
- Running On VM
- Network Simulation
- Tracing Log
- Debugger

Introduction

- Running malware deliberately, while monitoring the results
- Requires a **safe environment**
- Must prevent malware from spreading to production machines
- Real machines can be **airgapped** –no network connection to the Internet or to other machines

SIMULATION

Running on VM

- **Caution !!!!**
  - Make sure OS is on isolated environment
  - Network are configured to host only
  - Antivirus and Firewall of HOST OS must be updated and active
  - Take Guest OS snapshot before running the OS on Guest OS

Tool Needed

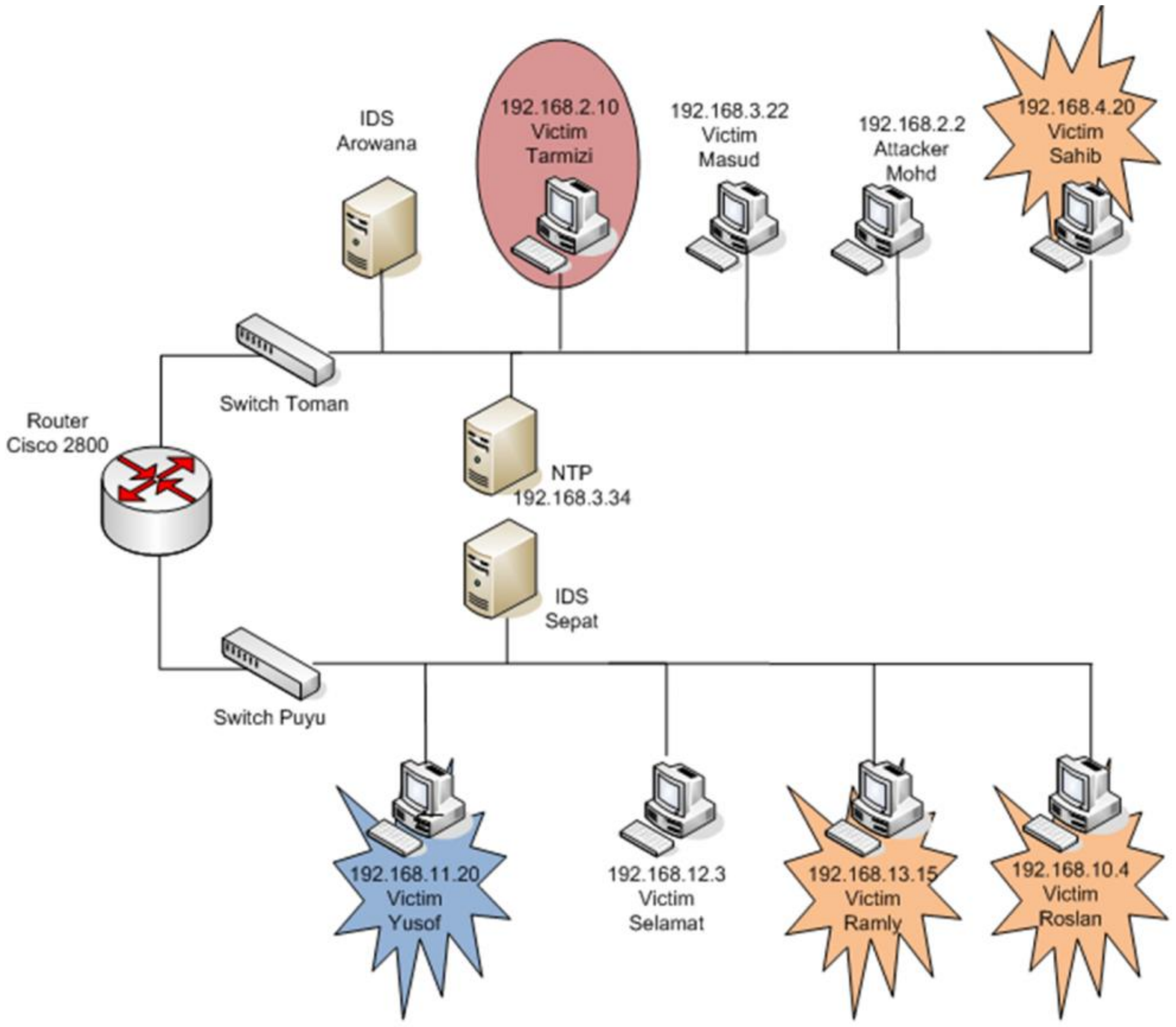
- Windows
  - Process explorer
  - Procmon
- Linux/Unix
  - Strace

Network Simulation

- Need to create a working Network environment that consist
  1. Working Domain
  2. DNS
  3. IDS/Firewall
  4. Webserver/Mail Server
  5. Few vulnerable Client
  6. NTP for Syncronizing time
- Tool Needed
  - TcpDump
  - Wireshark to analyze
  - Any other Log such as IDS Log, Firewall log

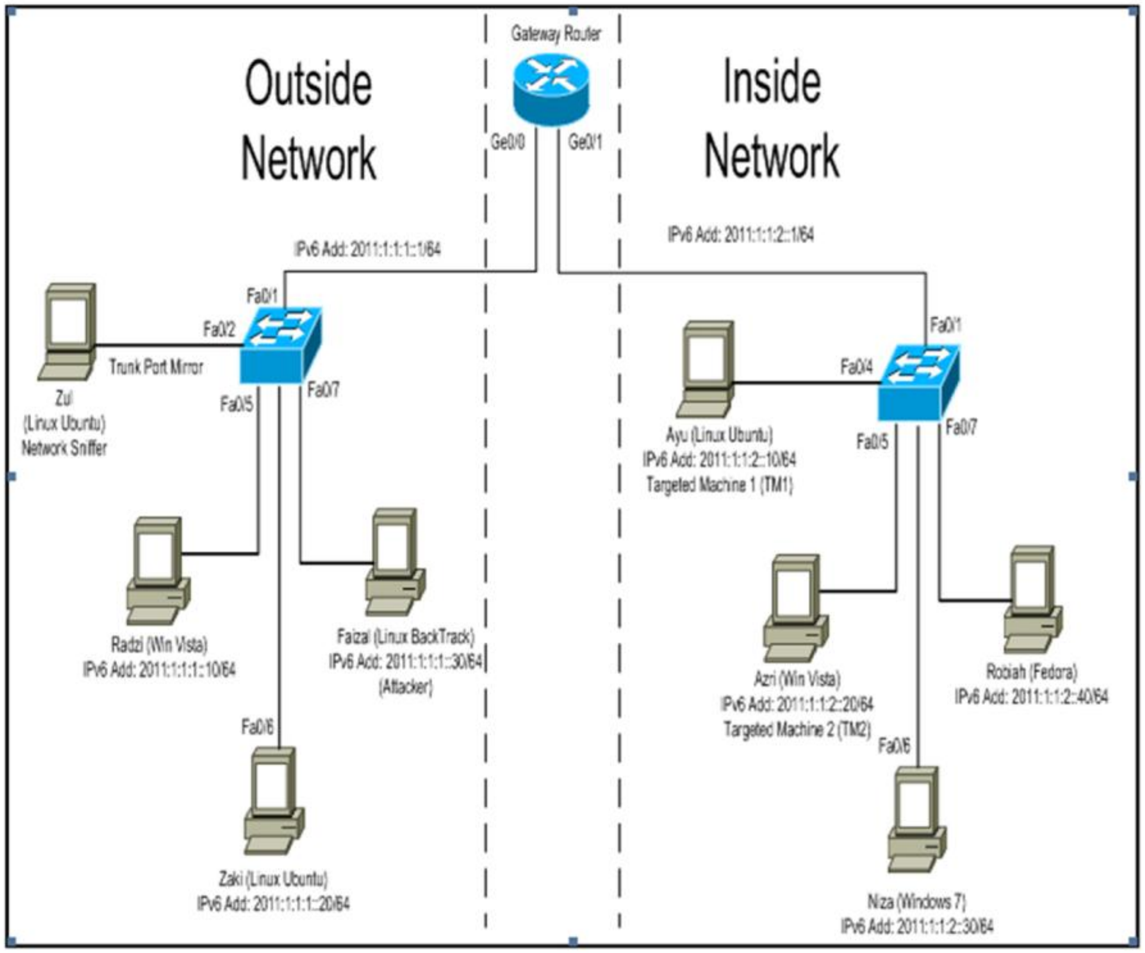
Among The Testbed Used for Capturing Malware/ Attack

Robiah And Rahayu
Simulating Worm Activity

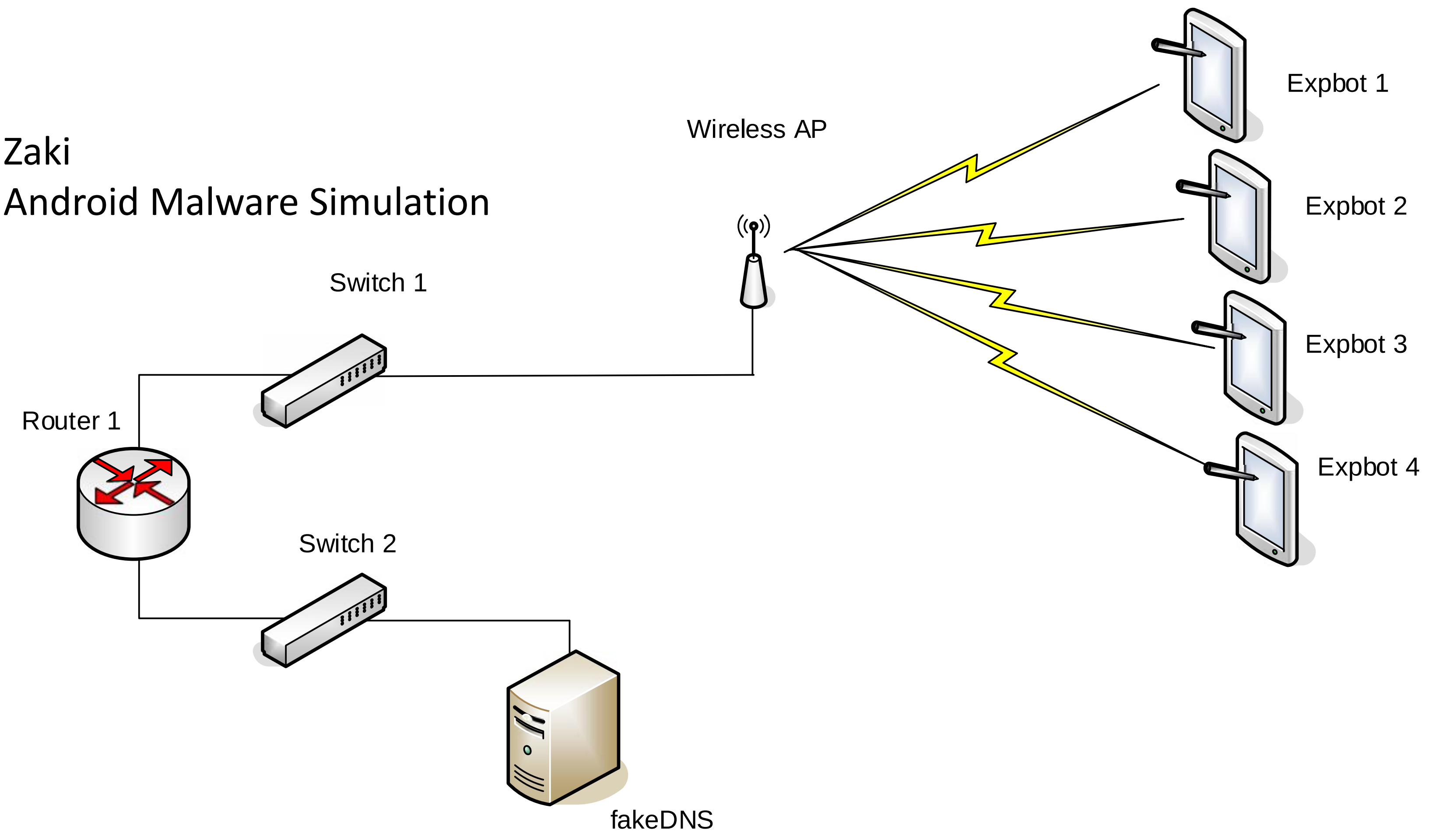


ENHANCED INTRUSION DETECTION SYSTEM THROUGH TIME-GENERATED FEATURE FOR IPV6 NETWORK

Zul Muslim
IpV6 Attack Simulation


Gateway Router
Outside Network
Inside Network
Ge0/0
Ge0/1
IPv6 Add: 2011:1:1:1::1/64
IPv6 Add: 2011:1:1:2::1/64
Fa0/1
Fa0/2
Trunk Port Mirror
Zul (Linux Ubuntu) Network Sniffer
Fa0/5
Fa0/7
Radzi (Win Vista) IPv6 Add: 2011:1:1:1::10/64
Faizal (Linux BackTrack) IPv6 Add: 2011:1:1:1::30/64 (Attacker)
Fa0/6
Zaki (Linux Ubuntu) IPv6 Add: 2011:1:1:1::20/64
Fa0/1
Fa0/4
Ayu (Linux Ubuntu) IPv6 Add: 2011:1:1:2::10/64 Targeted Machine 1 (TM1)
Fa0/5
Fa0/7
Azri (Win Vista) IPv6 Add: 2011:1:1:2::20/64 Targeted Machine 2 (TM2)
Robiah (Fedora) IPv6 Add: 2011:1:1:2::40/64
Fa0/6
Niza (Windows 7) IPv6 Add: 2011:1:1:2::30/64

Zaki
Android Malware Simulation



Analyzing Log

DEBUGGER

Disassemblers v. Debuggers

- A disassembler like IDA Pro shows the state of the program just before execution begins
- Debuggers show
  - Every memory location
  - Register
  - Argument to every function
- At any point during processing
  - And let you change them

Three Debuggers

- Immunity Debugger
- Ollydbg
  - Most popular for malware analysis
  - User-mode debugging only
  - IDA Pro has a built-in debugger, but it's not as easy to use or powerful as Ollydbg
- Windbg
  - Supports kernel-mode debugging

Source-Level v. Assembly-Level Debuggers

- Source-level debugger
  - Usually built into development platform
  - Can set breakpoints (which stop at lines of code)
  - Can step through program one line at a time
- Assembly-level debuggers (low-level)
  - Operate on assembly code rather than source code
  - Malware analysts are usually forced to use them, because they don't have source code

Kernel v. User-Mode Debugging

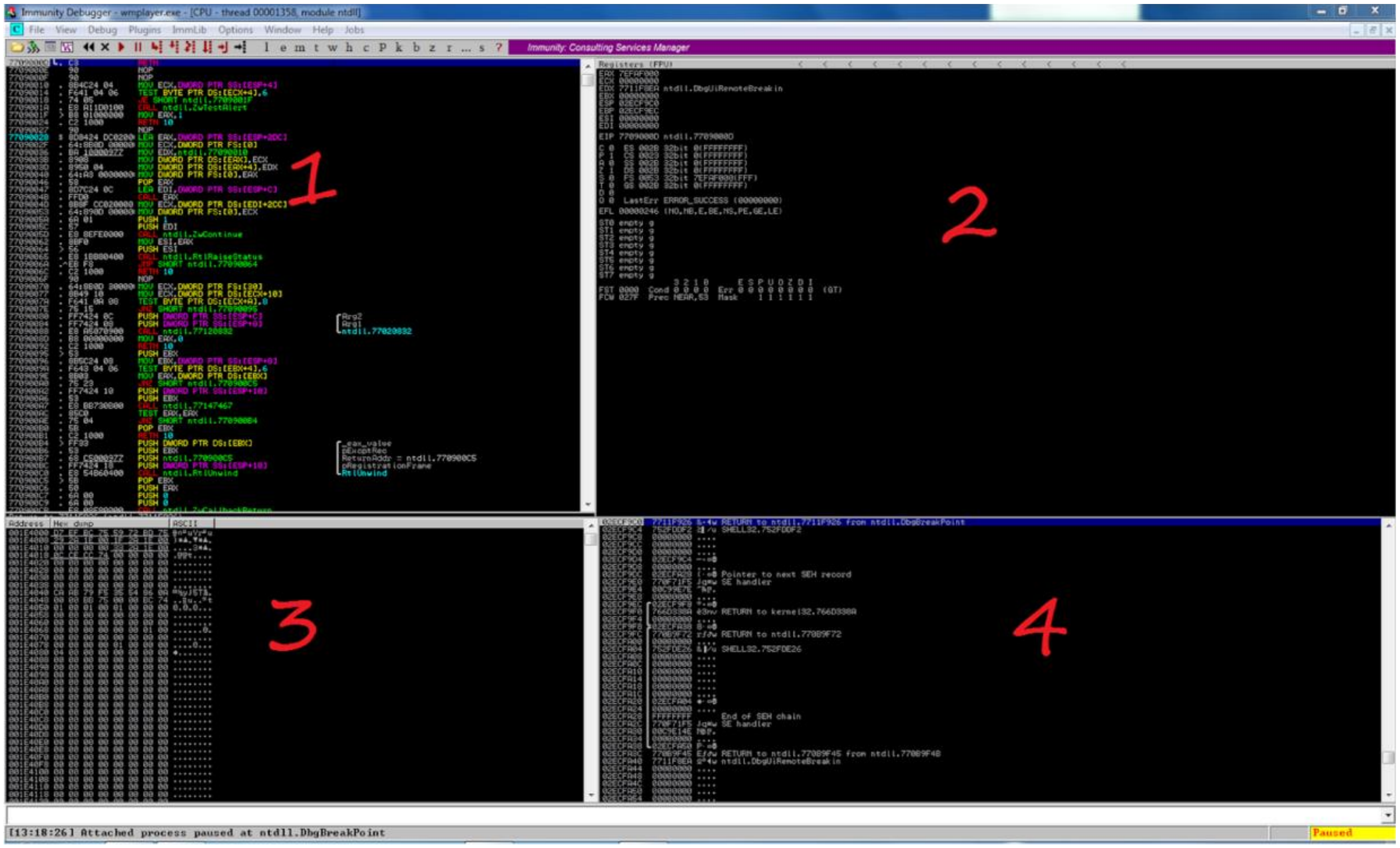
User Mode Debugging

- Debugger runs on the same system as the code being analyzed
- Debugging a single executable
- Separated from other executables by the OS

Kernel Mode Debugging

- Requires two computers, because there is only one kernel per computer
- If the kernel is at a breakpoint, the system stops
- One computer runs the code being debugged
- Other computer runs the debugger
- OS must be configured to allow kernel debugging
- Two machines must be connected

Immunity Debugger


Immunity Debugger - wmplayer.exe - [CPU - thread 00001358, module ntdll]
File View Debug Plugins ImmLib Options Window Help Jobs
l e m t w h c P k b z r ... s ?
Immunity: Consulting Services Manager
1
2
3
4
Paused

```
7709000D  L. C3            RETN
7709000E     90            NOP
7709000F     90            NOP
77090010   . 8B4C24 04     MOV ECX,DWORD PTR SS:[ESP+4]
77090014   . F641 04 06    TEST BYTE PTR DS:[ECX+4],6
77090018   . 74 05         JE SHORT ntdll.7709001F
7709001A   . E8 A11D0100   CALL ntdll.ZwTestAlert
7709001F   > B8 01000000   MOV EAX,1
77090024   . C2 1000       RETN 10
77090027     90            NOP
77090028   $ 8D8424 DC0200 LEA EAX,DWORD PTR SS:[ESP+2DC]
7709002F   . 64:8B0D 00000 MOV ECX,DWORD PTR FS:[0]
77090036   . BA 10000977   MOV EDX,ntdll.77090010
7709003B   . 8908          MOV DWORD PTR DS:[EAX],ECX
7709003D   . 8950 04       MOV DWORD PTR DS:[EAX+4],EDX
77090040   . 64:A3 0000000 MOV DWORD PTR FS:[0],EAX
77090046   . 58            POP EAX
77090047   . 8D7C24 0C     LEA EDI,DWORD PTR SS:[ESP+C]
7709004B   . FFD0          CALL EAX
7709004D   . 8B8F CC020000 MOV ECX,DWORD PTR DS:[EDI+2CC]
77090053   . 64:890D 00000 MOV DWORD PTR FS:[0],ECX
7709005A   . 6A 01         PUSH 1
7709005C   . 57            PUSH EDI
7709005D   . E8 8EFE0300   CALL ntdll.ZwContinue
77090062   . 8BF0          MOV ESI,EAX
77090064   > 56            PUSH ESI
77090065   . E8 1B880400   CALL ntdll.RtlRaiseStatus
7709006A   .^EB F8         JMP SHORT ntdll.77090064
7709006C   . C2 1000       RETN 10
7709006F     90            NOP
77090070   . 64:8B0D 30000 MOV ECX,DWORD PTR FS:[30]
77090077   . 8B49 10       MOV ECX,DWORD PTR DS:[ECX+10]
7709007A   . F641 0A 08    TEST BYTE PTR DS:[ECX+A],8
7709007E   . 75 15         JNZ SHORT ntdll.77090095
77090080   . FF7424 0C     PUSH DWORD PTR SS:[ESP+C]                  [Arg2
77090084   . FF7424 08     PUSH DWORD PTR SS:[ESP+8]                  |Arg1
77090088   . E8 A5070900   CALL ntdll.77120832                        Lntdll.77020832
7709008D   . B8 00000000   MOV EAX,0
77090092   . C2 1000       RETN 10
77090095   > 53            PUSH EBX
77090096   . 8B5C24 08     MOV EBX,DWORD PTR SS:[ESP+8]
7709009A   . F643 04 06    TEST BYTE PTR DS:[EBX+4],6
7709009E   . 8B03          MOV EAX,DWORD PTR DS:[EBX]
770900A0   . 75 23         JNZ SHORT ntdll.770900C5
770900A2   . FF7424 10     PUSH DWORD PTR SS:[ESP+10]
770900A6   . 53            PUSH EBX
770900A7   . E8 BB730800   CALL ntdll.77147467
770900AC   . 85C0          TEST EAX,EAX
770900AE   . 75 04         JNZ SHORT ntdll.770900B4
770900B0   . 5B            POP EBX
770900B1   . C2 1000       RETN 10
770900B4   > FF33          PUSH DWORD PTR DS:[EBX]                    [_eax_value
770900B6   . 53            PUSH EBX                                   |pExceptRec
770900B7   . 68 C5000977   PUSH ntdll.770900C5                        |ReturnAddr = ntdll.770900C5
770900BC   . FF7424 10     PUSH DWORD PTR SS:[ESP+10]                 |pRegistrationFrame
770900C0   . E8 54B60400   CALL ntdll.RtlUnwind                       LRtlUnwind
770900C5   > 5B            POP EBX
770900C6   . 50            PUSH EAX
770900C7   . 6A 00         PUSH 0
770900C9   . 6A 00         PUSH 0
```

```
Registers (FPU)
EAX 7EFAF000
ECX 00000000
EDX 7711F8EA ntdll.DbgUiRemoteBreakin
EBX 00000000
ESP 02ECF9C0
EBP 02ECF9EC
ESI 00000000
EDI 00000000

EIP 7709000D ntdll.7709000D

C 0  ES 002B 32bit 0(FFFFFFFF)
P 1  CS 0023 32bit 0(FFFFFFFF)
A 0  SS 002B 32bit 0(FFFFFFFF)
Z 1  DS 002B 32bit 0(FFFFFFFF)
S 0  FS 0053 32bit 7EFAF000(FFF)
T 0  GS 002B 32bit 0(FFFFFFFF)
D 0
O 0  LastErr ERROR_SUCCESS (00000000)

EFL 00000246 (NO,NB,E,BE,NS,PE,GE,LE)

ST0 empty g
ST1 empty g
ST2 empty g
ST3 empty g
ST4 empty g
ST5 empty g
ST6 empty g
ST7 empty g
               3 2 1 0      E S P U O Z D I
FST 0000  Cond 0 0 0 0  Err 0 0 0 0 0 0 0 0  (GT)
FCW 027F  Prec NEAR,53  Mask    1 1 1 1 1 1
```

```
Address   Hex dump                 ASCII
001E4000  D7 EF BC 75 59 72 BD 75  ╫n╝uYr╜u
001E4008  29 29 1E 00 1F 29 1E 00  ))▲.▼)▲.
001E4010  00 00 00 00 38 29 1E 00  ....8)▲.
001E4018  BC CE CC 74 00 00 00 00  .╬╠t....
001E4020  00 00 00 00 00 00 00 00  ........
001E4028  00 00 00 00 00 00 00 00  ........
001E4030  00 00 00 00 00 00 00 00  ........
001E4038  00 00 00 00 00 00 00 00  ........
001E4040  CA AB 79 F5 35 54 95 0A  ╩½yJ5Tò.
001E4048  00 00 00 75 00 00 BC 74  ...u..╝t
001E4050  01 00 01 00 01 00 00 00  0.0.0...
001E4058  00 00 00 00 00 00 00 00  ........
001E4060  00 00 00 00 00 00 00 00  ........
001E4068  00 00 00 00 00 00 01 00  ......0.
001E4070  00 00 00 00 00 00 00 00  ........
001E4078  00 00 00 00 01 00 00 00  ....0...
001E4080  04 00 00 00 00 00 00 00  ♦.......
001E4088  00 00 00 00 00 00 00 00  ........
001E4090  00 00 00 00 00 00 00 00  ........
001E4098  00 00 00 00 00 00 00 00  ........
001E40A0  00 00 00 00 00 00 00 00  ........
001E40A8  00 00 00 00 00 00 00 00  ........
001E40B0  00 00 00 00 00 00 00 00  ........
001E40B8  00 00 00 00 00 00 00 00  ........
001E40C0  00 00 00 00 00 00 00 00  ........
001E40C8  00 00 00 00 00 00 00 00  ........
001E40D0  00 00 00 00 00 00 00 00  ........
001E40D8  00 00 00 00 00 00 00 00  ........
001E40E0  00 00 00 00 00 00 00 00  ........
001E40E8  00 00 00 00 00 00 00 00  ........
001E40F0  00 00 00 00 00 00 00 00  ........
001E40F8  00 00 00 00 00 00 00 00  ........
001E4100  00 00 00 00 00 00 00 00  ........
001E4108  00 00 00 00 00 00 00 00  ........
001E4110  00 00 00 00 00 00 00 00  ........
001E4118  00 00 00 00 00 00 00 00  ........
```

```
02ECF9C0  7711F926  &∙◄w RETURN to ntdll.7711F926 from ntdll.DbgBreakPoint
02ECF9C4  752FDDF2  ≥▌/u SHELL32.752FDDF2
02ECF9C8  00000000  ....
02ECF9CC  00000000  ....
02ECF9D0  00000000  ....
02ECF9D4  02ECF9C4  ─∙∞☻
02ECF9D8  00000000  ....
02ECF9DC  02ECFA28  (·∞☻ Pointer to next SEH record
02ECF9E0  770F71F5  Jq☼w SE handler
02ECF9E4  00C99E7E  ~₧╔.
02ECF9E8  00000000  ....
02ECF9EC  02ECF9F8  °∙∞☻
02ECF9F0  766D338A  è3mv RETURN to kernel32.766D338A
02ECF9F4  00000000  ....
02ECF9F8  02ECFA38  8·∞☻
02ECF9FC  77069F72  rƒ♠w RETURN to ntdll.77069F72
02ECFA00  00000000  ....
02ECFA04  752FDE26  &▐/u SHELL32.752FDE26
02ECFA08  00000000  ....
02ECFA0C  00000000  ....
02ECFA10  00000000  ....
02ECFA14  00000000  ....
02ECFA18  00000000  ....
02ECFA1C  00000000  ....
02ECFA20  02ECFA04  ♦·∞☻
02ECFA24  00000000  ....
02ECFA28  FFFFFFFF       End of SEH chain
02ECFA2C  770F71F5  Jq☼w SE handler
02ECFA30  00C9E14E  Nß╔.
02ECFA34  00000000  ....
02ECFA38  02ECFA50  P·∞☻
02ECFA3C  77069F45  Eƒ♠w RETURN to ntdll.77069F45 from ntdll.77069F48
02ECFA40  7711F8EA  ê°◄w ntdll.DbgUiRemoteBreakin
02ECFA44  00000000  ....
02ECFA48  00000000  ....
02ECFA4C  00000000  ....
02ECFA50  00000000  ....
02ECFA54  00000000  ....
```

```
[13:18:26] Attached process paused at ntdll.DbgBreakPoint
```

- The CPU Instructions – displays the memory address, opcode and assembly instructions, additional comments, function names and other information related to the CPU instructions
- The Registers – displays the contents of the general purpose registers, instruction pointer, and flags associated with the current state of the application..
- The Stack – shows the contents of the current stack
- The Memory Dump – shows the contents of the application's memory

- **Disassembly**
- Disassembly part is divided into four columns. In the first column we can see memory address. Second column shows instruction operation code (hex view of instruction) located at that address. Machine language is made up from these operation codes, and that is what CPU is executing in reality. Third column is assembly code. Since immunity is dynamic debugger, you can double click on any assembly instruction and change it. Change will be visible immediately and you can see how it affects the program. And forth column contains comments. Immunity debugger tries to guess some details about instructions and if its successful it will place details in the comments. If you are not satisfied with debugger guess you can delete it and write your comments by double clicking on it.
- **Registers**
- Here you can see all the registers of you CPU and their values. Top selection makes general purpose registers, which contain temporarily values, and registers which are used for controlling program flow. Middle selection contains flag registers, which CPU changes when something of importance has happened in the program (like an overflow). The bottom selection contains registers which are used while executing floating point operations. Registers will change color from black to red when changed, which makes it easy to watch for the changes. Same as with assembly code, you can double click on any register and change its value. You can also follow value stored in the register if it is a valid memory address by right clicking on it and selecting *Follow in dump.*

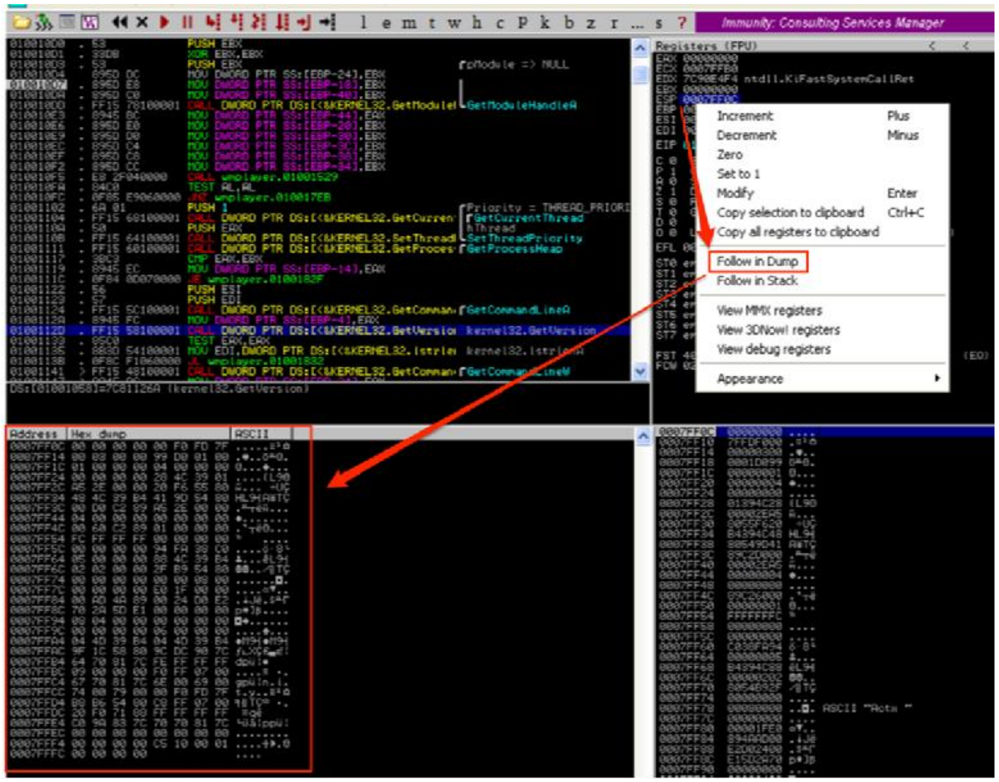
- **Dump**
- *Dump* window shows you the hex view of entire program. It is divided into three columns. First column shows the address. Second column show hex characters located at that address. In the third column we can see ASCII representation of hex data. You can search *Dump* values by right clicking on it and selecting *Search for -> Binary string*.
- **Stack**
- Memory location at which points ESP (stack pointer register) is shown at the top of the stack window. It is divided into three columns. First column shows the address. Second shows data located at that address. And the third contains comments. You can change data at the stack by double clicking on it.

Register

- EAX
- EBX
- ECX
- EDX
- ESI
- EDI
- EBP
- ESP
- **The Instruction Pointer (EIP)**
  - Not a general purpose register, but fitting to cover here, EIP points to the memory address of the next instruction to be executed by the CPU. As you'll see in the coming tutorials, control the value of EIP and you can control the execution flow of the application (to execute code of your choosing).
- **EFLAGS**
  - register is comprised of a series of flags that represent Boolean values resulting from calculations and comparisons and can be used to determine when/if to take conditional jumps

Memory Dump

- Skipping to the Memory Dump pane of the CPU view, this is simply where you can view the contents of a memory location.
- For example, contents of memory at ESP, which in the following screenshot is pointing to 0007FF0C.
- Right-click on ESP, select “Follow in Dump” and the Memory Dump pane will display that location.



For Detail Explanation Please Refer

http://www.securitysift.com/windows-exploit-development-part-1-basics/