Malware Reverse Engineering Basic Knowledge

Andrea Mambretti (mambro007@gmail.com)

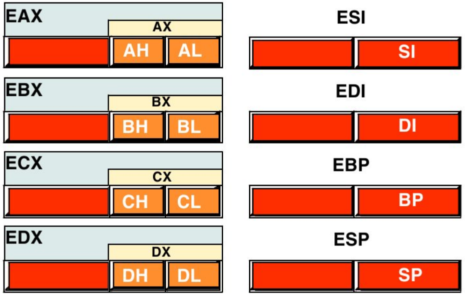
Politecnico di Milano

September 19, 2012

(1) How is the IA made?

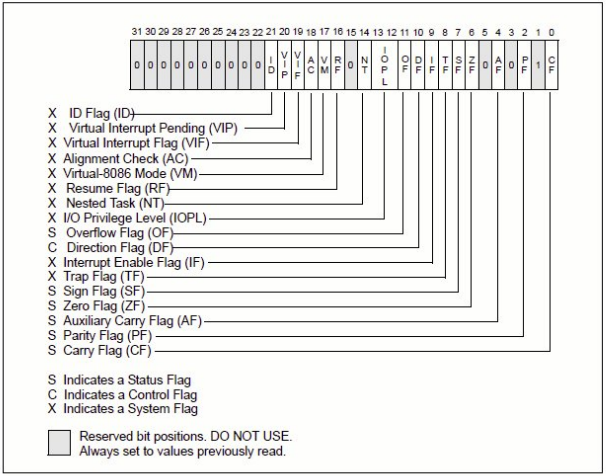
- The processor has 32 bits internal registers to manage and to execute operations on data
  They are EAX,EBX,ECX,EDX,ESI,EDI,EBP,EIP and ESP
- Among them we identify EAX,EBX,ECX,EDX that are used by the operations
- EBP (BP = base pointer) and ESP (SP = stack pointer) are the stack bounds
- EDI and ESI are extraregisters
- EIP is the register that contains the next instruction has to be execute

(2) How is the IA made?



- NB: The name is a porting from 16 bits Intel Architecture where the registers were called AX,BX and so on.
  We can use it also in 32 bits assembly language calling them without the "E"
  There's also the possibility to use AX,BX,CX,DX such as 8 bits registers for example we can use AX as AH and AL that mean higher and lower 8 bits of AX

(1)What about EFLAGS?


31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 ID VIP VIF AC VM RF 0 NT IOPL OF DF IF TF SF ZF 0 AF 0 PF 1 CF
X ID Flag (ID)
X Virtual Interrupt Pending (VIP)
X Virtual Interrupt Flag (VIF)
X Alignment Check (AC)
X Virtual-8086 Mode (VM)
X Resume Flag (RF)
X Nested Task (NT)
X I/O Privilege Level (IOPL)
S Overflow Flag (OF)
C Direction Flag (DF)
X Interrupt Enable Flag (IF)
X Trap Flag (TF)
S Sign Flag (SF)
S Zero Flag (ZF)
S Auxiliary Carry Flag (AF)
S Parity Flag (PF)
S Carry Flag (CF)
S Indicates a Status Flag
C Indicates a Control Flag
X Indicates a System Flag
Reserved bit positions. DO NOT USE. Always set to values previously read.

(2)What about EFLAGS?

- It's another 32 bit register
- Only 8 bits on 32 are of interest to us. The others are either for the kernel mode function or are of little interest for programmer
- The 8 bits are called flags. We consider them singularly. They are boolean (true/false)
- The meaning of each bit is different. They represent Overflow, Direction, interrupt disable, sign, zero, auxiliary carry, parity and carry flags
- For each instruction executed by the processor they change and represent the information about the last instruction. They are VERY important for the control flow of the program

Syntax

- In the assembly world we can find two main syntax the AT&T and the Intel
- AT&T is used by all the UNIX program (gdb)
- Intel syntax is used by Microsoft program (IDApro and others)

(1)Differences in the notation

- Consider the following operation:
  "move the value 0 to EAX"
- AT&T:
  mov $0x0,%eax
- Intel:
  mov eax, 0h
- Comments:
  - As you can see in AT&T syntax the destination is the second operand instead of the first as Intel syntax is
  - In the AT&T the register are denoted with % and the immediate/costant with $. In the Intel syntax they are not used.

(2)Differences in the notation

- Consider this other operation:

  "move the value 0 to the address contained in EBX+4"
- AT&T:

```
mov $0x0,0x4(%ebx)
```
- Intel:

```
mov [ebx+4h],0h
```
- Comments:
  - This case wants figure out how is done the deferentiation in assembly
  - In AT&T we use the parenthesis. In the Intel syntax we have to use square brackets
  - The way to manage the offset is another syntax difference in the first case we have to put it out of the parenthesis in the second one inside the square brackets

(1)Basic Instructions overview

- Every processor has a very huge number of instructions to execute (see Intel Manual[1])
- A subset of the whole instructions set is usually dipendent by the considered processor
- We will focus on the other subset of instructions that is common among all the processos
- We will use the Intel syntax to see them because is the same syntax used in IDApro by default

[1] `http://www.intel.com/content/www/us/en/processors/architectures-software-developer-manuals.html`

(2)Basic Instruction MOV

- With this instruction, as said in the example above, we move a value from a source place to a destination. There are a ton of different versions. They change in function of the operands ex 32 bits operands, 16 bits operands, immediate to reg, immediate to memory
- General: MOV **destination**, **source**
  source can be an immediate, a register, a memory location
  destination can be either a register or a memory location
  NB: Every combination is possible except memloc to memloc!!!
- Examples

- MOV eax, ebx
- MOV eax, FFFFFFFFh
- MOV ax, bx
- MOV [eax], ecx
- MOV [eax],[ecx] NO!!!
- MOV al, FFh

(3)Basic Instruction ADD

- With this instruction we can add a value from **source** to the destination operand and put the new value inside one of the two operands
- General: ADD **destination**, **source**
  source

What is Basic Static Analysis?

- B.S.A. consists in a very simple set of operation that can allow a malware analyst to get usefull information about a certain binary file
- B.S.A. tools can give us infomation about:
  - **Antivirus Scanning**: tell us if it is an already known malware or not and if yes which kind (ex www.virustotal.com)
  - **Hashing**: tell if a certain file has been corrupted or not
  - **Strings**: give us all the strings defined as costant in the program that can be usefull to undestand what that file does
  - **Recognizers Packing and Obfuscation**: if the file uses some type of packing (ex: upx) or obfuscation to avoid recognition
  - **Header Analysis**: Looking inside the header give us information about import and export function, when it's compiled, if it's packet, if there's some extra segment and other stuff

When is it necessary?

- If someone is really paranoic the answer is every time before launch an application
- for all the others when something of suspicious is detected during the previous execution
- after this phase a human analyzer knows if it is needed to procede with other analysis such as dynamic and advaced static analysis

What is Dynamic Analysis?

- D.A. consists in looking the behaviour of a malware running it and loggin every change and action done in the system
- of course is not always possible use this technique because a specific malware can damage the system and makes the information unreachable
- so what can we do to avoid it?

Possible Solution: Virtual Machine

- Using Virtual Machine we can set a perfect enviroment for our malware (fake a whole network of hosts,logger etc.)
- We can check out of box what happens inside (system call, network operation etc.).
  if something goes wrong we can, using snapshots, rollback the machine state to a sane position
- Possible sequence of operation:
  - Start with a clean snapshot with no malware running on it
  - Trasfer the malware to the virtual machine
  - Conduct your analysis on the virtual machine
  - Take your notes, screenshots, and data from the virtual machine and transfer it to the physical machine
  - Revert the virtual machine to the clean snapshot

Existing VM

VM

Someone of them are already prepared to Malware Analysis

- VMware
- VirtualBox
- Qemu
- Cuckoo
- Anubis
- Andrubis
- A ton of others

Problems and Solution

Problems

- It's prooved Today's Malware can avoid the dynamic analysis. They recognize all the well-know VB and they change their behavior when are runned in
- The worst scenario happens when specific malware are done to exploit vulnerabilities inside VM to own the whole machine where the VB is running

Solution

- One of the possible solution to understand if the malware has behaved not in his standard way and which are his possibility is to see the code more deeply with the Advanced Static Analysis (aka Reverse Engineering)

What is Reverse Engineering in General?

Definition

- Reverse engineering is taking apart an object to see how it works in order to duplicate or enhance the object. The practice, taken from older industries, is now frequently used on computer hardware and software.
- Reverse engineering is usually conducted to obtain missin knowledge, ideas, adn design philosophy when such information is unavailable

What is Software reverse engineering?

Definition

- Software reverse engineering involves reversing a program's machine code (the string of 0s and 1s that are sent to the logic processor) back into the assembly language (x86, x86-64, ARM so on and so forth)
- Software reverse engineering requires a combination of skills and a thorough undestanding of computers and software development, but like most worthwhile subjects, the only real prerequisite is a strong curiosity and desire to learn. Software reverse engineering integrates several arts: code breaking, puzzle solving, programming, and logical analysis.

When do we use it?

- Reversing Application
- Security-Related Reversing
  - Malicious Software
  - Reversing Cryptographic Algorithms
  - Digital Rights Management
  - Auditing Program Binaries
- Reversing Software Development
  - Achieving Interoperability with Proprietary Software
  - Developing Competing Software
  - Evaluating Software Quality and Robustness

Background of a good Reverser

- Assembly Language
- Compilers
- Virtual Machine and Bytecodes (ex Java)
- Operative Systems

Tools of a good Reverser

- System-Monitoring Tools
- Disassemblers
- Debuggers
- Decompilers

Some Common Reversing Tools

- IDA Pro
- OllyDbg
- WinDbg
- PEBrowse Professional Interactive
- SoftICE

Is it Legal?

- The Legal debate around reverse engineering has been going on for years
- 1998 - Digital Millenium Copyright Act
  - Circumvention of copyright protection systems
  - The development of circumvention technologies
- Luckily, DMCA makes several exceptions
  - Interoperability
  - Encryption research
  - Security testing
  - Educational institutions and public libraries
  - Government investigation
  - Regulation
  - Protection of privacy

Reverse on Malware and Antireversing Techniques

Today's malware and commercial program use techniques against Reversing

- Anti-Disassembly (Linear and Flow Oriented Disassembly)
  - Jump instructions with the same target
  - Jump instruction with a Costant Condition
  - Impossible disassembly
- Anti-Debugging
  - They understand that are executed in a debugger and change their behaviour either crashing itself, the debugger or totally doing other stuff
- Anti-Virtual Machine Techniques
- Packers and Unpacking

Conclusion

- Software Reverse Engineering is a very powerfull instrument but it requires a lot of lowlevel-knowledge
- Appling this technique on malware analysis is not optional if we want undestand how the malware works
- Can be very time consuming and if all the tools used for the analysis are not setted correctly there's no way to reverse the malware

End

- Thanks Folk...Questions?
- So now one practical example