Practical Reversing (I)

Harsimran Walia

www.SecurityXploded.com

Disclaimer

The Content, Demonstration, Source Code and Programs presented here is "AS IS" without any warranty or conditions of any kind. Also the views/ideas/knowledge expressed here are solely of the trainer's only and nothing to do with the company or the organization in which the trainer is currently working.

However in no circumstances neither the trainer nor SecurityXploded is responsible for any damage or loss caused due to use or misuse of the information presented here.

Acknowledgement

- Special thanks to **null** & **Garage4Hackers** community for their extended support and cooperation.
- Thanks to all the **Trainers** who have devoted their precious time and countless hours to make it happen.

Reversing & Malware Analysis Training

This presentation is part of our **Reverse Engineering & Malware Analysis** Training program. Currently it is delivered only during our local meet for FREE of cost.

For complete details of this course, visit our Security Training page.

$ whoami

harsimranwalia.info

- **b44nz0r**
- Research Scientist @ McAfee Labs
- Mechanical Engineer @IIT Delhi
- Independent Security Researcher
- RE, Exploit Analysis/Development, Malware Analysis

Twitter : b44nz0r

Email : walia.harsimran@gmail.com

Outline

- Break Point
- Debug Registers
- Flags
- API Help

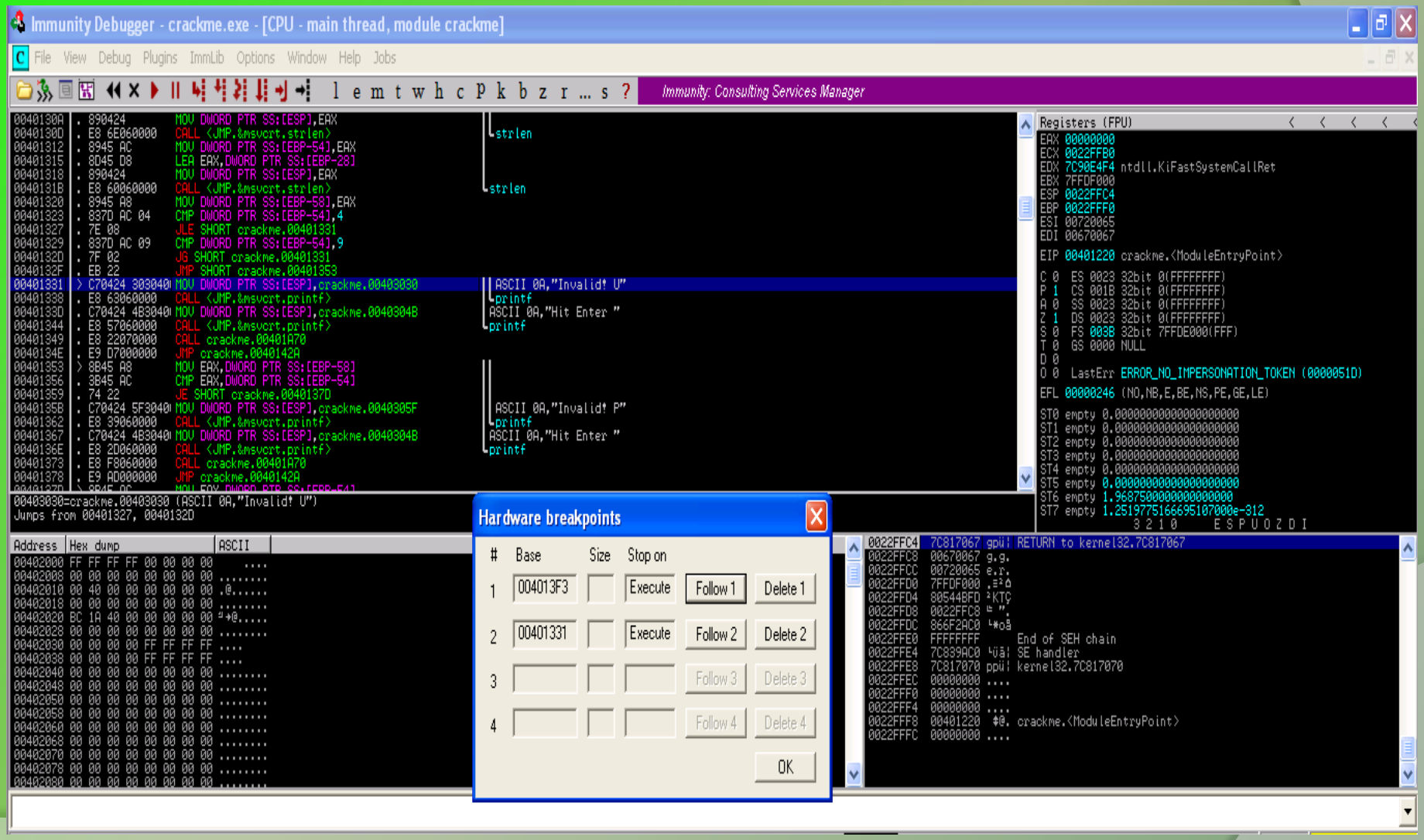
Types of Breakpoints

- Software
- Hardware
- Memory

Breakpoint

- Software breakpoints are set by replacing the instruction at the target address with 0xCC (INT3/ Breakpoint interrupt)
- Hardware breakpoints are set via debug registers. Only 4 hardware breakpoints can be set
- Debug registers:
  - 8 debug registers present
  - DR0 – DR3 : Address of breakpoint
  - DR6 : Debug Status – To determine which breakpoint is active
  - DR7 : Debug Control – Flags to control the breakpoints such as break on read or on-write
- Debug registers are not accessible in Ring 3

Hardware Breakpoints



Memory

- To access memory, need of permissions
- Lots of permissions
  - PAGE_GUARD
  - PAGE_READWRITE
  - PAGE_EXECUTE
  - PAGE_EXECUTE_READ
- To set memory breakpoint,
  - the permissions of that memory region is set to PAGE_GUARD
  - whenever an access is made to that memory STATUS_GUARD_PAGE_VIOLATION exception is raised
  - On getting the exception the debugger changes the permission back to the original
  - Notifies the user of the breakpoint

Breakpoints

DEMO

Flags (Eflags Register)

- 1 register – 32 bits
- Each bit signifies a flag
- Few important ones are:

Bit #	Abbreviation	Description
0	CF	Carry flag
2	PF	Parity flag
4	AF	Adjust flag
6	ZF	Zero flag
7	SF	Sign flag
8	TF	Trap flag (single step)
9	IF	Interrupt enable flag
11	OF	Overflow flag

Flags Demystified

- **Carry flag** is used to indicate when an arithmetic carry or borrow has been generated out of the most significant ALU bit position
- **Parity flag** indicates if the number of set bits is odd or even in the binary representation of the result of the last operation
- **Adjust flag** is used to indicate when an arithmetic carry or borrow has been generated out of the 4 least significant bits
- **Zero Flag** is used to check the result of an arithmetic operation, including bitwise logical instructions. It is set if an arithmetic result is zero, and reset otherwise
- **Sign flag** is used to indicate whether the result of last mathematic operation resulted in a value whose most significant bit was set
- A **trap flag** permits operation of a processor in single-step mode
- **Overflow flag** is used to indicate when an arithmetic overflow has occurred in an operation, indicating that the signed two's-complement result would not fit in the number of bits used for the operation

Basic Reversing Techniques

- Check for readable strings
- Import table (IAT) for imported Windows API
- Setting breakpoint on interesting API
- Single stepping

Variables

- Found under **Names** tab
  - L - library function
  - F - regular function
  - C - instruction
  - A - ascii string
  - D - data
  - I - imported name

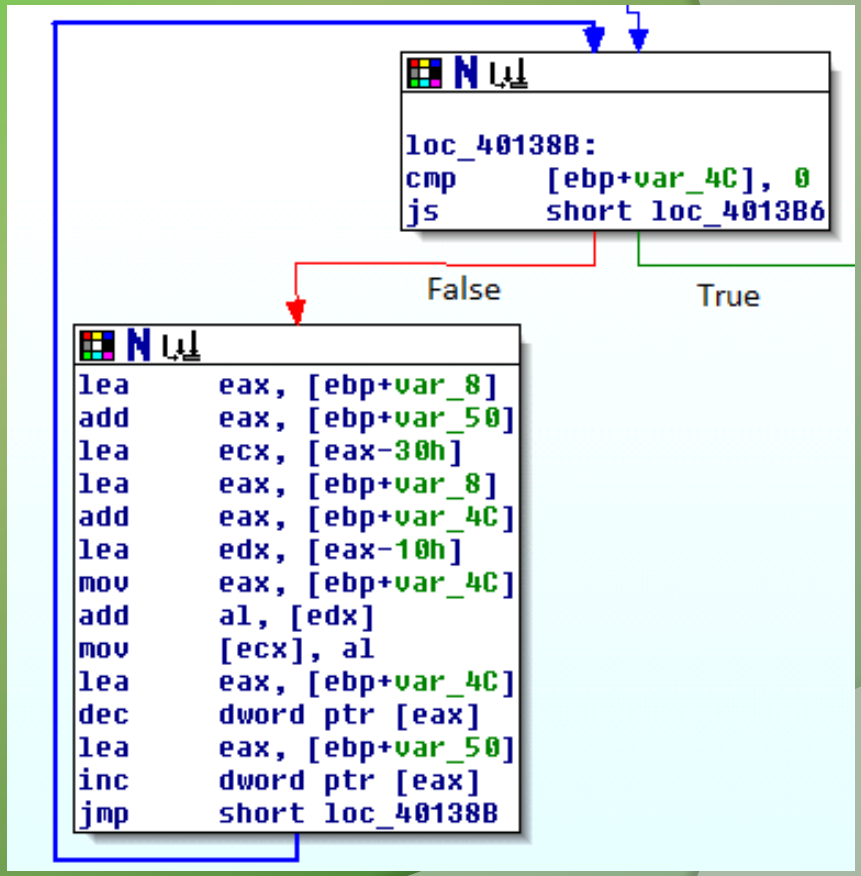
Contd..

```
shr     eax, 4
shl     eax, 4
mov     [ebp+var_6C], eax
mov     eax, [ebp+var_6C]
call    ___chkstk
call    ___main
add     dword_402000, 5
mov     dword ptr [esp], offset aCrackme ; "#Crackme\n\n"
call    printf
```

- Global variables are generally dword_<address>
  - dword_402000 – as shown in image
- Local variables are of the form var_<offset>
  - var_6C – as shown in image

Loop in IDA

- Red Line
  - If condition is false
  - (zero flag = 0)
- Green Line
  - If condition is true
  - (zero flag = 1)



Reversing a Simple Crackme

DEMO

Crackme Code

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>

int main()
{
   char a[10],b[10],c[10],d[10];
   int i,j,k,l,r,s;
   printf("#Crackme\n\n");
   printf("enter username: ");
   scanf("%s",a);
   printf("enter password: ");
   scanf("%s",b);
   k = strlen(a);
   l = strlen(b);
   if (k <5 || k >=10){
       printf("\nInvalid! Username Length\n");
       printf("\nHit Enter to Exit\n");
       getchar();
   } else {
   if (l != k){
       printf("\nInvalid! Password Length\n");
       printf("\nHit Enter to Exit\n");
      getchar();
   } else {
      i = k-1;
      j = 0;
      while (i >= 0){
           c[j] = a[i]+i;
           i--;
           j++;
      }
      c[j] = 0;
      r = strlen(c);
      if (r == l){
           i = strcmp(c,b);
           if (i == 0){
             printf("\nCongratulations! You did it..\n");
             printf("\nHit Enter to Exit\n");
           } else {
             printf("\nAccess Denied! Wrong Password\n");
```

References

- Complete Reference Guide for Reversing & Malware Analysis Training

Thank You !

www.SecurityXploded.com