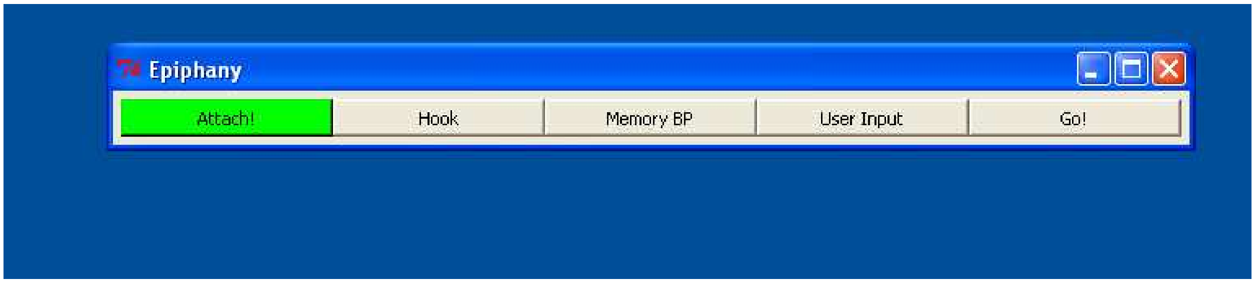
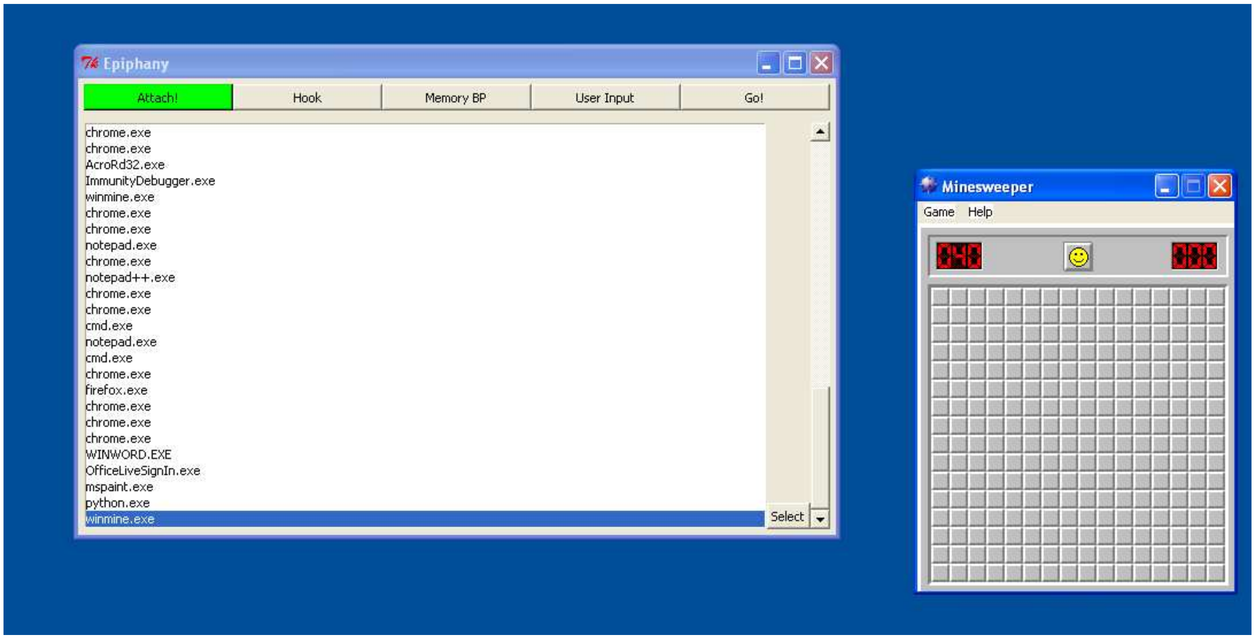
I will show the usage of the program through two examples.

Minesweeper:

1. Start the program. A frame with 5 button comes up:



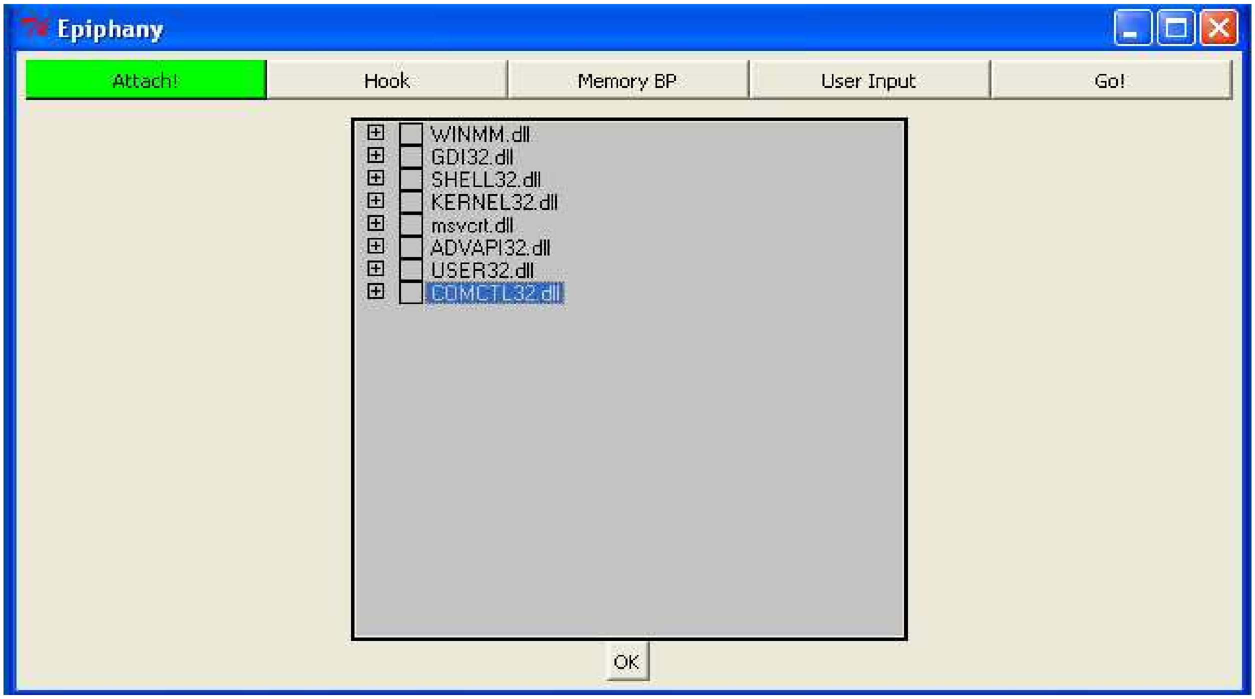
2. Start minesweeper and click on “Attach”



```
chrome.exe
chrome.exe
AcroRd32.exe
ImmunityDebugger.exe
winmine.exe
chrome.exe
chrome.exe
notepad.exe
chrome.exe
notepad++.exe
chrome.exe
chrome.exe
cmd.exe
notepad.exe
cmd.exe
chrome.exe
firefox.exe
chrome.exe
chrome.exe
chrome.exe
WINWORD.EXE
OfficeLiveSignIn.exe
mspaint.exe
python.exe
winmine.exe
```

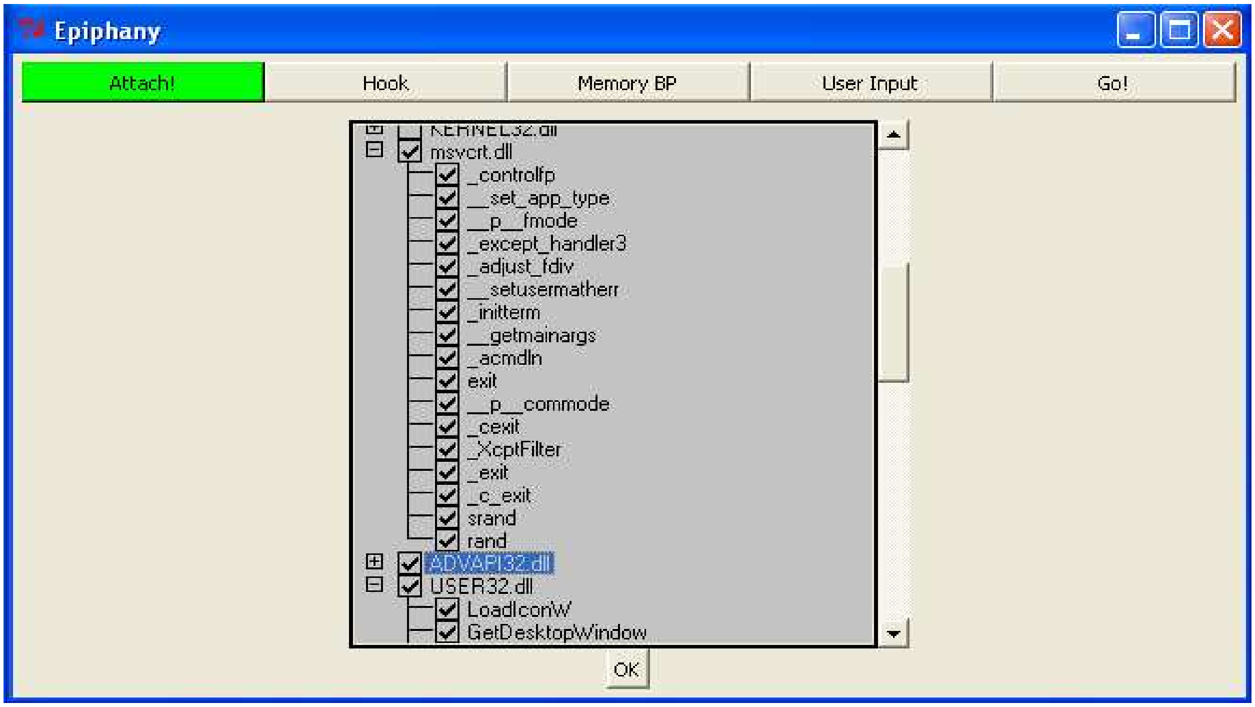
On clicking “Attach” button a list of currently running processes comes up. Select winmine.exe and click on Submit button.

3. Now, press the “Hook” button



- ⊞ □ WINMM.dll
- ⊞ □ GDI32.dll
- ⊞ □ SHELL32.dll
- ⊞ □ KERNEL32.dll
- ⊞ □ msvcrt.dll
- ⊞ □ ADVAPI32.dll
- ⊞ □ USER32.dll
- ⊞ □ COMCTL32.dll

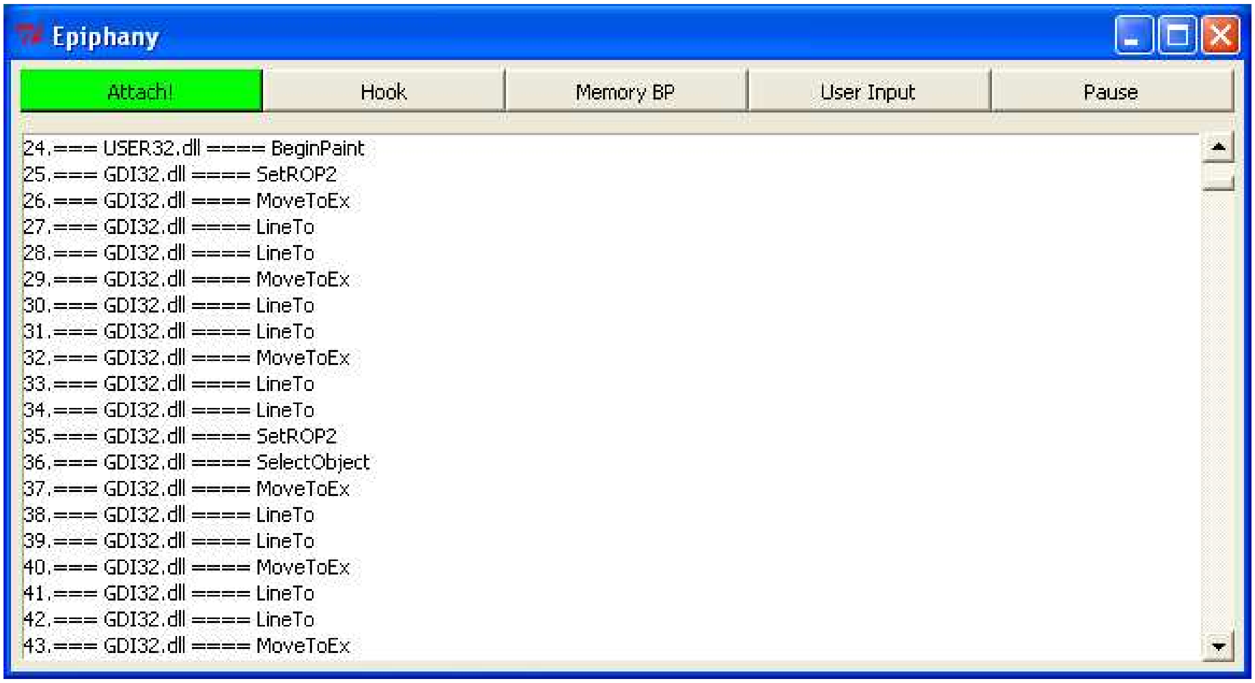
In all 8 modules are loaded when winmine.exe runs. Except for some functions in KERNEL32.dll and GDI32.dll select all functions:



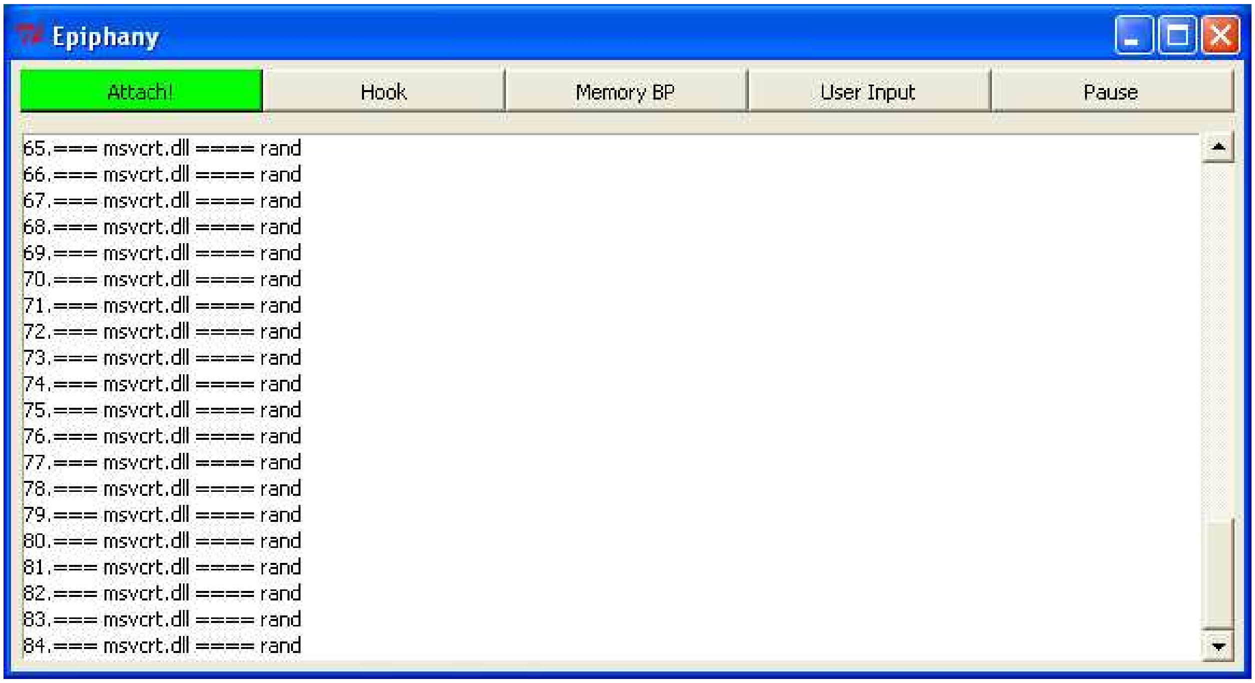
- ⊞ □ KERNEL32.dll
- ⊟ ☑ msvcrt.dll
  - ☑ _controlfp
  - ☑ __set_app_type
  - ☑ __p__fmode
  - ☑ _except_handler3
  - ☑ _adjust_fdiv
  - ☑ __setusermatherr
  - ☑ _initterm
  - ☑ __getmainargs
  - ☑ _acmdln
  - ☑ exit
  - ☑ __p__commode
  - ☑ _cexit
  - ☑ _XcptFilter
  - ☑ _exit
  - ☑ _c_exit
  - ☑ srand
  - ☑ rand
- ⊞ ☑ ADVAPI32.dll
- ⊟ ☑ USER32.dll
  - ☑ LoadIconW
  - ☑ GetDesktopWindow

Press OK

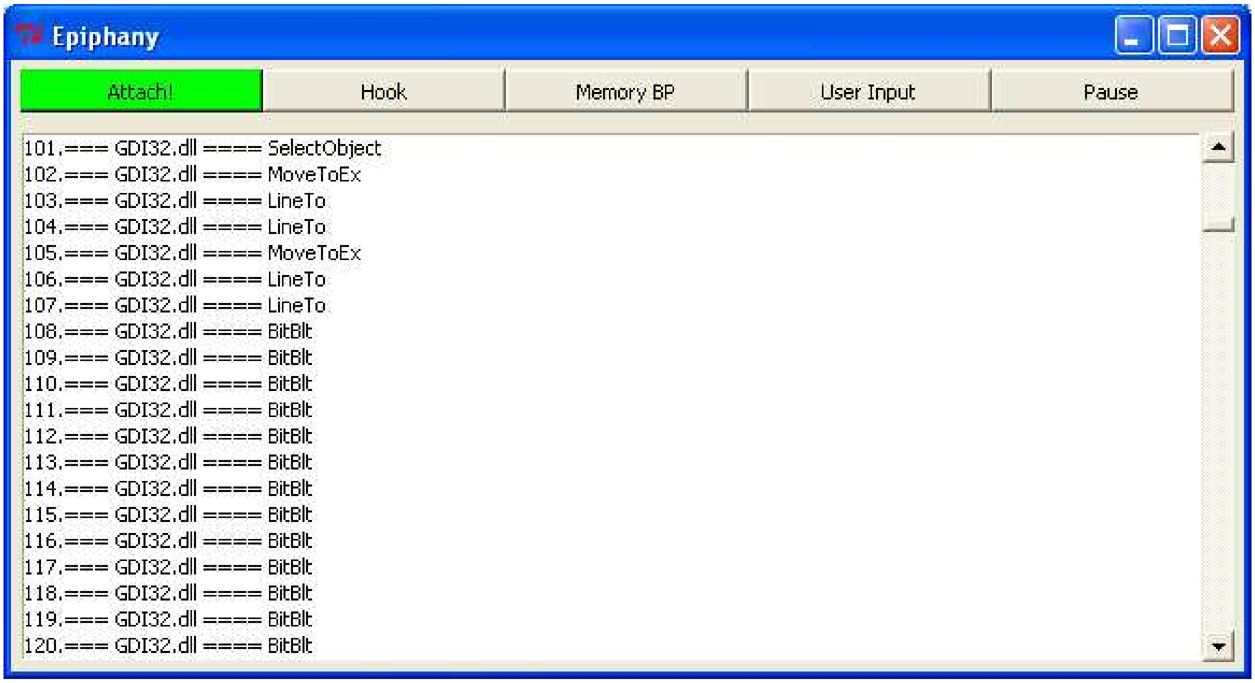
4. Now Click on Go Button. And press restart game button on Minesweeper. You will observe all the function calls that happen within individual modules in the main screen of the program:



```
24.=== USER32.dll ==== BeginPaint
25.=== GDI32.dll ==== SetROP2
26.=== GDI32.dll ==== MoveToEx
27.=== GDI32.dll ==== LineTo
28.=== GDI32.dll ==== LineTo
29.=== GDI32.dll ==== MoveToEx
30.=== GDI32.dll ==== LineTo
31.=== GDI32.dll ==== LineTo
32.=== GDI32.dll ==== MoveToEx
33.=== GDI32.dll ==== LineTo
34.=== GDI32.dll ==== LineTo
35.=== GDI32.dll ==== SetROP2
36.=== GDI32.dll ==== SelectObject
37.=== GDI32.dll ==== MoveToEx
38.=== GDI32.dll ==== LineTo
39.=== GDI32.dll ==== LineTo
40.=== GDI32.dll ==== MoveToEx
41.=== GDI32.dll ==== LineTo
42.=== GDI32.dll ==== LineTo
43.=== GDI32.dll ==== MoveToEx
```



```
65.=== msvcrt.dll ==== rand
66.=== msvcrt.dll ==== rand
67.=== msvcrt.dll ==== rand
68.=== msvcrt.dll ==== rand
69.=== msvcrt.dll ==== rand
70.=== msvcrt.dll ==== rand
71.=== msvcrt.dll ==== rand
72.=== msvcrt.dll ==== rand
73.=== msvcrt.dll ==== rand
74.=== msvcrt.dll ==== rand
75.=== msvcrt.dll ==== rand
76.=== msvcrt.dll ==== rand
77.=== msvcrt.dll ==== rand
78.=== msvcrt.dll ==== rand
79.=== msvcrt.dll ==== rand
80.=== msvcrt.dll ==== rand
81.=== msvcrt.dll ==== rand
82.=== msvcrt.dll ==== rand
83.=== msvcrt.dll ==== rand
84.=== msvcrt.dll ==== rand
```



```
101.=== GDI32.dll ==== SelectObject
102.=== GDI32.dll ==== MoveToEx
103.=== GDI32.dll ==== LineTo
104.=== GDI32.dll ==== LineTo
105.=== GDI32.dll ==== MoveToEx
106.=== GDI32.dll ==== LineTo
107.=== GDI32.dll ==== LineTo
108.=== GDI32.dll ==== BitBlt
109.=== GDI32.dll ==== BitBlt
110.=== GDI32.dll ==== BitBlt
111.=== GDI32.dll ==== BitBlt
112.=== GDI32.dll ==== BitBlt
113.=== GDI32.dll ==== BitBlt
114.=== GDI32.dll ==== BitBlt
115.=== GDI32.dll ==== BitBlt
116.=== GDI32.dll ==== BitBlt
117.=== GDI32.dll ==== BitBlt
118.=== GDI32.dll ==== BitBlt
119.=== GDI32.dll ==== BitBlt
120.=== GDI32.dll ==== BitBlt
```

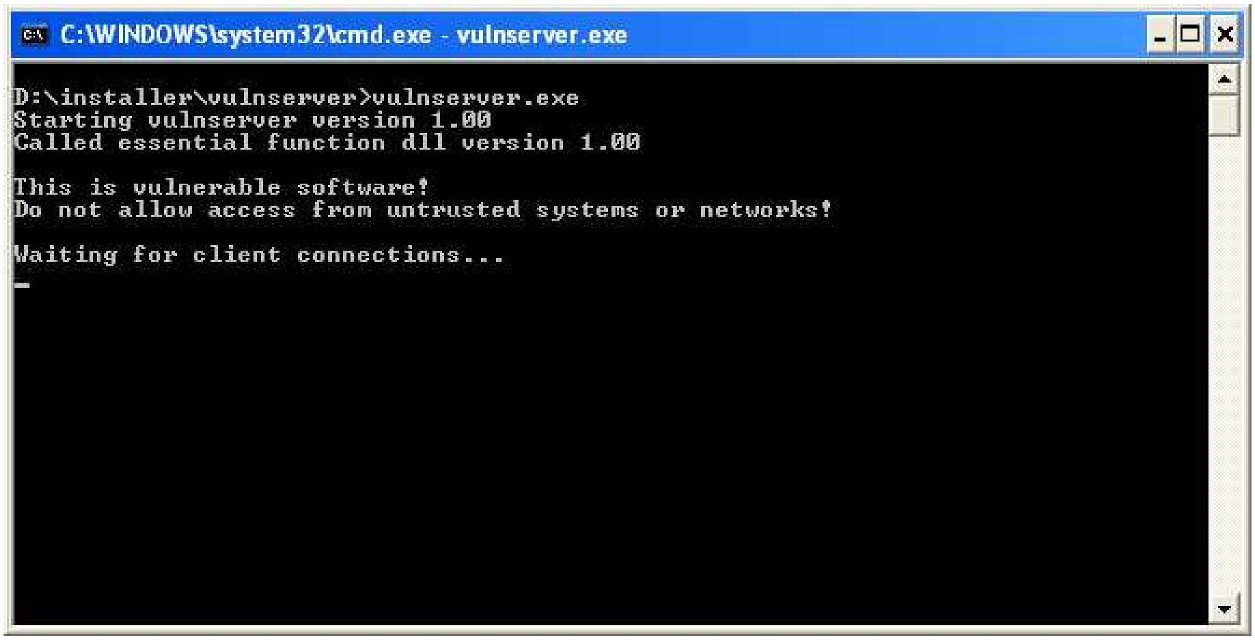
So via the few screenshots above, we saw what specific functions are being called at runtime in minesweeper. I haven't delved deep into it, but I think the game logic can be subverted using the same technique.

One thing that can be done probably is changing the return value of rand function to 0, or some fixed number. I observed that the number of calls to the rand() function is approximately twice the number of mines in a given level. And also the rand() function calls happen before the small rectangles are created and displayed via GDI32.BitBlt function. So, if the return value of rand() function is set to 0, or any other fixed value, one possibility that may happen is that no mines will be set or all mines may be clustered together. This is just a conjecture, I haven't tested this yet. I stated this just to emphasize the advantage that runtime behavior analysis can provide.

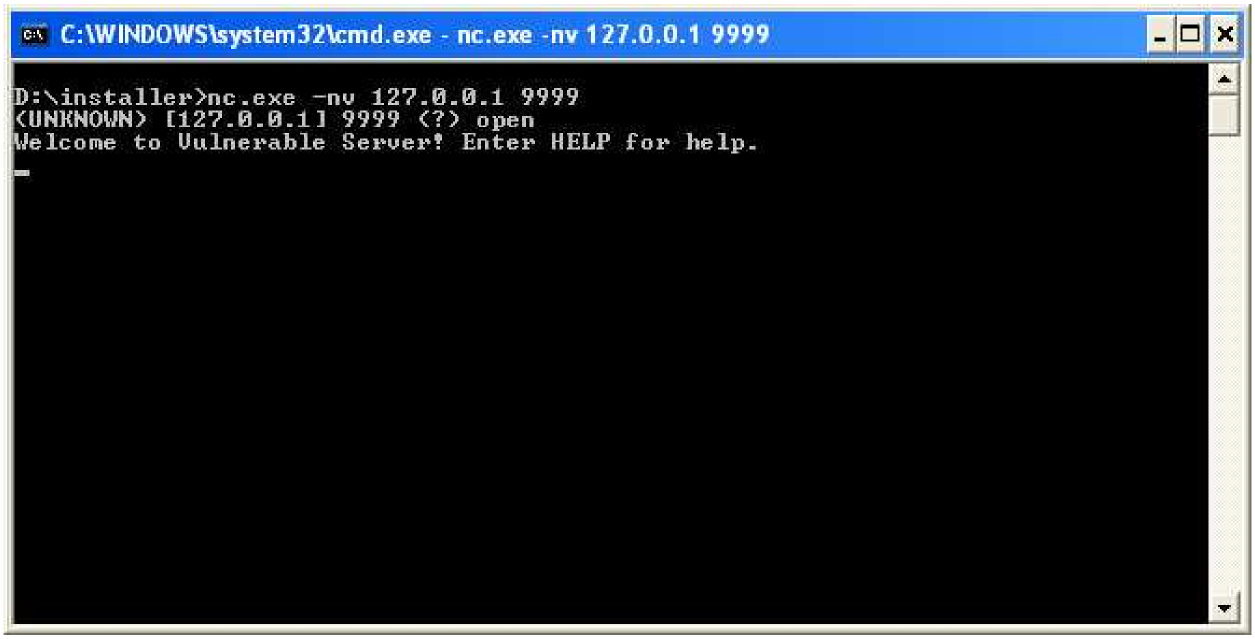
The second case would be more illustrative of the usage of runtime function analysis, as this example shows tracing of user input as well

Case 2: Network Program

I used a demo network program in the second case. This program listens on port 9999. A client can connect to this program via telnet or netcat. The program's name is vulnserver.exe.

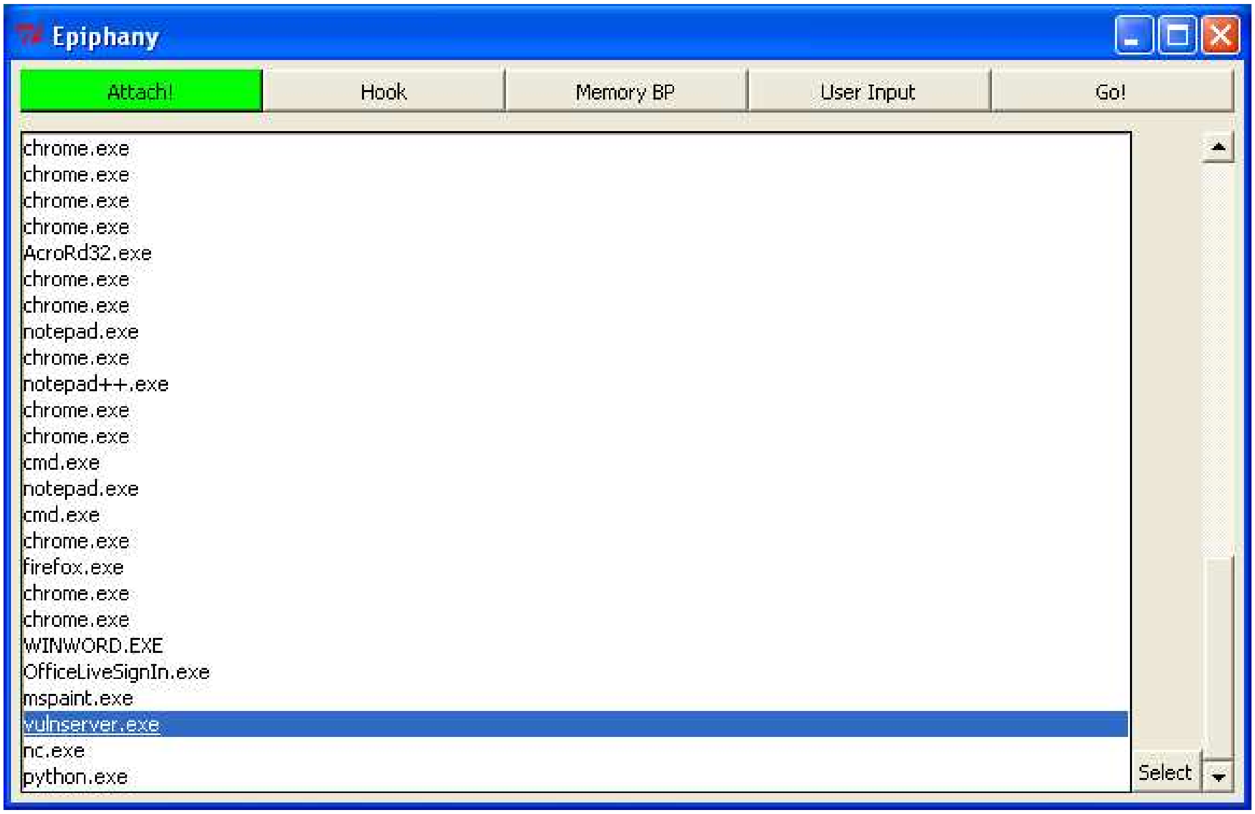


```
D:\installer\vulnserver>vulnserver.exe
Starting vulnserver version 1.00
Called essential function dll version 1.00

This is vulnerable software!
Do not allow access from untrusted systems or networks!

Waiting for client connections...
_
```

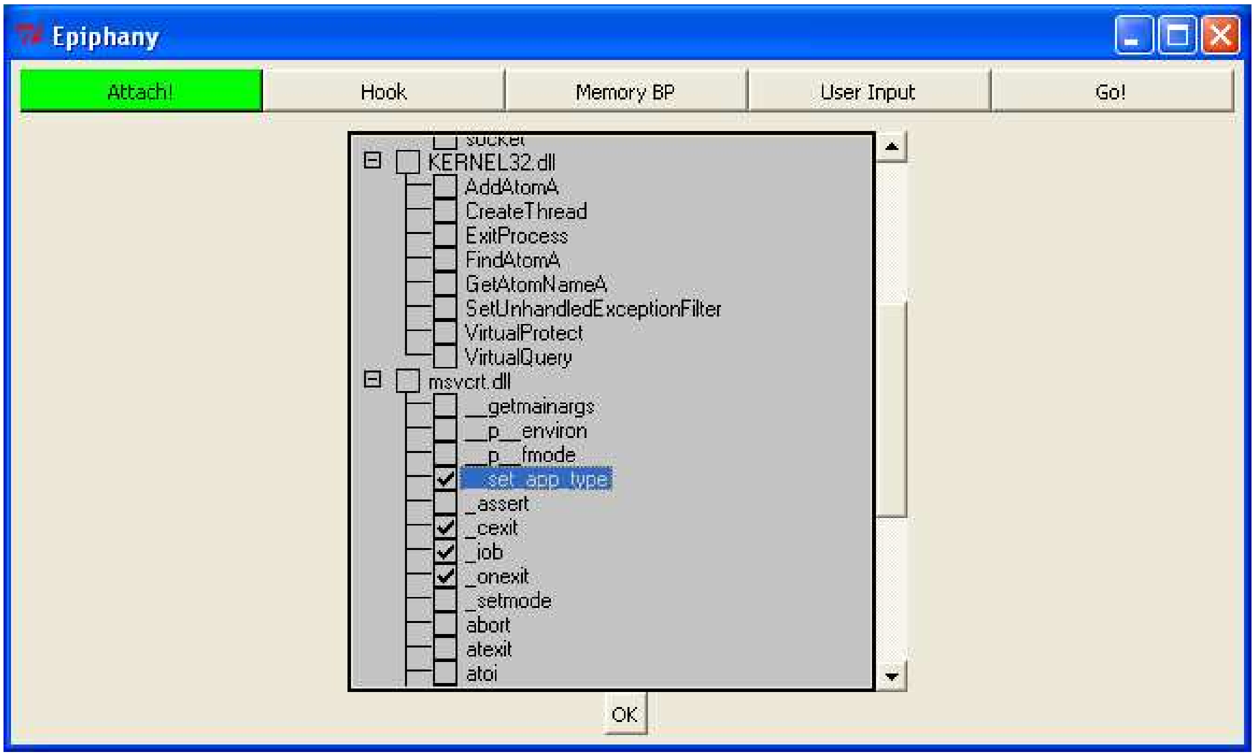


```
D:\installer>nc.exe -nv 127.0.0.1 9999
(UNKNOWN) [127.0.0.1] 9999 (?) open
Welcome to Vulnerable Server! Enter HELP for help.
_
```

Attach the debugging program to vulnserver.exe, and set hooks on desired functions.

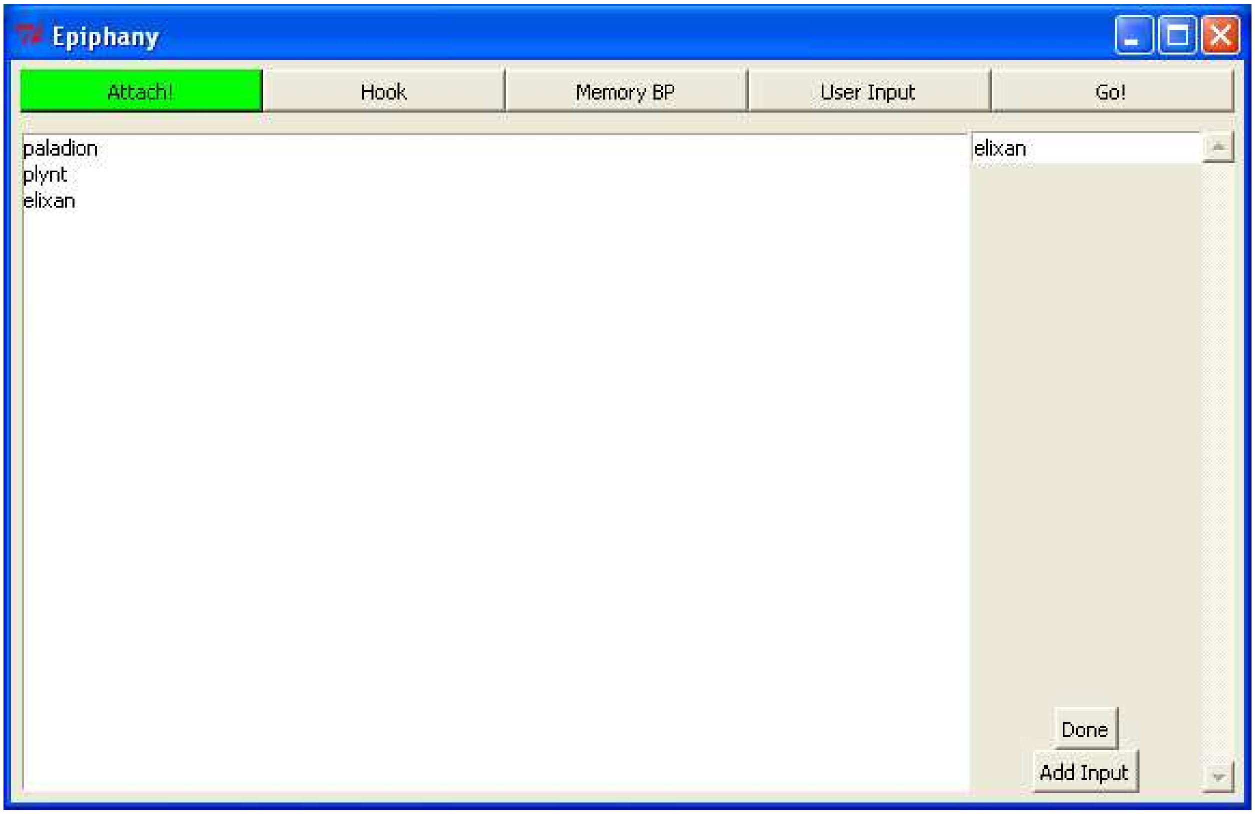


```
chrome.exe
chrome.exe
chrome.exe
chrome.exe
AcroRd32.exe
chrome.exe
chrome.exe
notepad.exe
chrome.exe
notepad++.exe
chrome.exe
chrome.exe
cmd.exe
notepad.exe
cmd.exe
chrome.exe
firefox.exe
chrome.exe
chrome.exe
WINWORD.EXE
OfficeLiveSignIn.exe
mspaint.exe
vulnserver.exe
nc.exe
python.exe
```

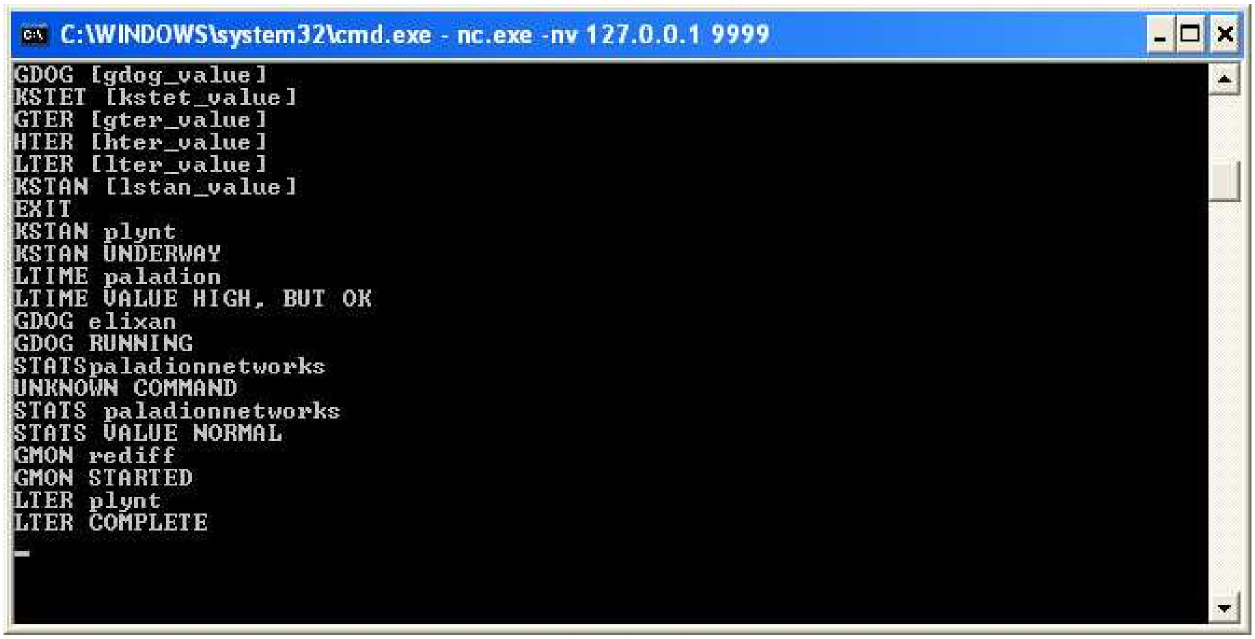


```
    ☐ socket
⊟ ☐ KERNEL32.dll
    ☐ AddAtomA
    ☐ CreateThread
    ☐ ExitProcess
    ☐ FindAtomA
    ☐ GetAtomNameA
    ☐ SetUnhandledExceptionFilter
    ☐ VirtualProtect
    ☐ VirtualQuery
⊟ ☐ msvcrt.dll
    ☐ __getmainargs
    ☐ __p__environ
    ☐ __p__fmode
    ☑ __set_app_type
    ☐ _assert
    ☑ _cexit
    ☑ _iob
    ☑ _onexit
    ☐ _setmode
    ☐ abort
    ☐ atexit
    ☐ atoi
```

Click on “User Input “ button and provide the strings you want to trace

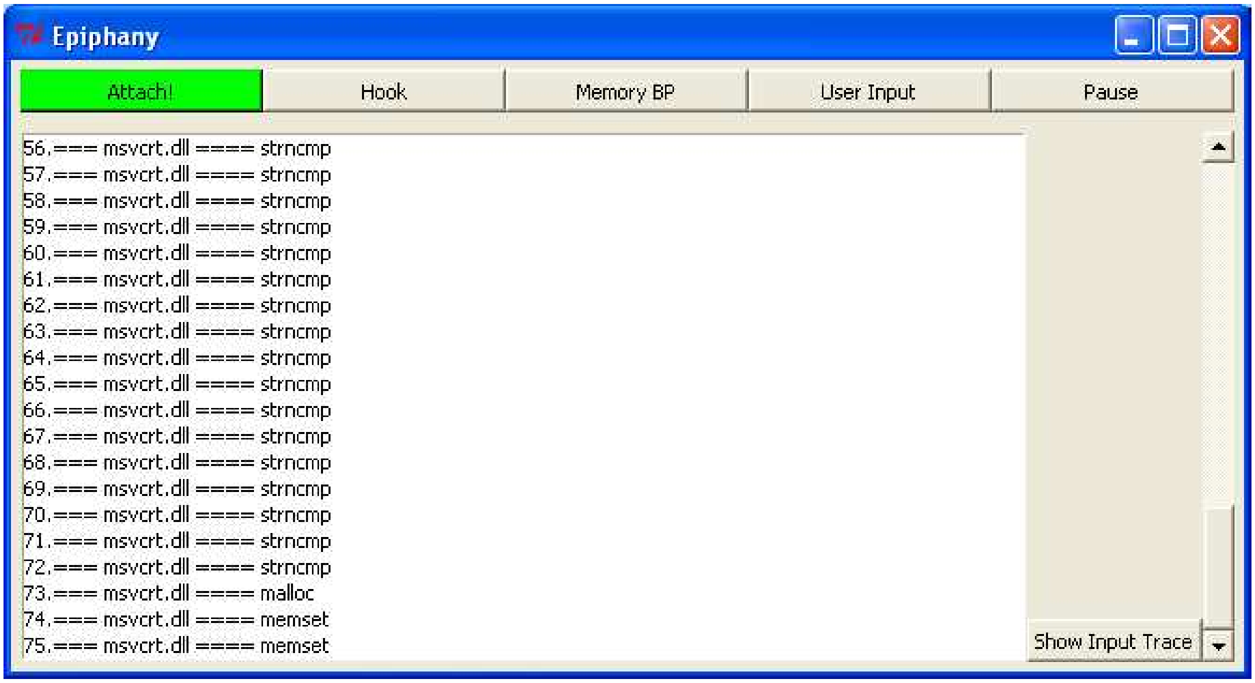


Once strings are provided click on Go! Button. In the client window, type in one of the commands with one of the provided strings as parameter:



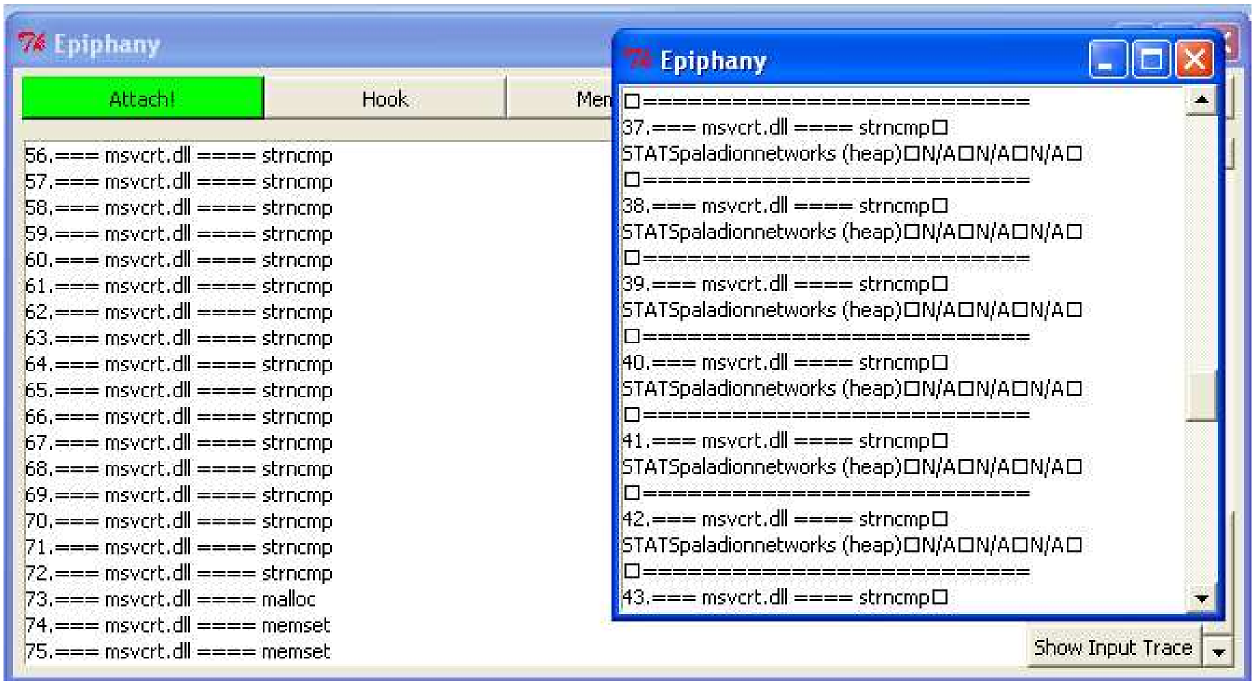
```
GDOG [gdog_value]
KSTET [kstet_value]
GTER [gter_value]
HTER [hter_value]
LTER [lter_value]
KSTAN [lstan_value]
EXIT
KSTAN plynt
KSTAN UNDERWAY
LTIME paladion
LTIME VALUE HIGH, BUT OK
GDOG elixan
GDOG RUNNING
STATSpaladionnetworks
UNKNOWN COMMAND
STATS paladionnetworks
STATS VALUE NORMAL
GMON rediff
GMON STARTED
LTER plynt
LTER COMPLETE
_
```

In the function call window, one can see various hooked functions, which were invoked upon running the command provided by the client.



```
56.=== msvcrt.dll ==== strncmp
57.=== msvcrt.dll ==== strncmp
58.=== msvcrt.dll ==== strncmp
59.=== msvcrt.dll ==== strncmp
60.=== msvcrt.dll ==== strncmp
61.=== msvcrt.dll ==== strncmp
62.=== msvcrt.dll ==== strncmp
63.=== msvcrt.dll ==== strncmp
64.=== msvcrt.dll ==== strncmp
65.=== msvcrt.dll ==== strncmp
66.=== msvcrt.dll ==== strncmp
67.=== msvcrt.dll ==== strncmp
68.=== msvcrt.dll ==== strncmp
69.=== msvcrt.dll ==== strncmp
70.=== msvcrt.dll ==== strncmp
71.=== msvcrt.dll ==== strncmp
72.=== msvcrt.dll ==== strncmp
73.=== msvcrt.dll ==== malloc
74.=== msvcrt.dll ==== memset
75.=== msvcrt.dll ==== memset
```

On clicking "Show Input Trace" button, one can see the user inputs that form one of the arguments to the hooked function.



```
56.=== msvcrt.dll ==== strncmp
57.=== msvcrt.dll ==== strncmp
58.=== msvcrt.dll ==== strncmp
59.=== msvcrt.dll ==== strncmp
60.=== msvcrt.dll ==== strncmp
61.=== msvcrt.dll ==== strncmp
62.=== msvcrt.dll ==== strncmp
63.=== msvcrt.dll ==== strncmp
64.=== msvcrt.dll ==== strncmp
65.=== msvcrt.dll ==== strncmp
66.=== msvcrt.dll ==== strncmp
67.=== msvcrt.dll ==== strncmp
68.=== msvcrt.dll ==== strncmp
69.=== msvcrt.dll ==== strncmp
70.=== msvcrt.dll ==== strncmp
71.=== msvcrt.dll ==== strncmp
72.=== msvcrt.dll ==== strncmp
73.=== msvcrt.dll ==== malloc
74.=== msvcrt.dll ==== memset
75.=== msvcrt.dll ==== memset
```

```
□==========================
37.=== msvcrt.dll ==== strncmp□
STATSpaladionnetworks (heap)□N/A□N/A□N/A□
□==========================
38.=== msvcrt.dll ==== strncmp□
STATSpaladionnetworks (heap)□N/A□N/A□N/A□
□==========================
39.=== msvcrt.dll ==== strncmp□
STATSpaladionnetworks (heap)□N/A□N/A□N/A□
□==========================
40.=== msvcrt.dll ==== strncmp□
STATSpaladionnetworks (heap)□N/A□N/A□N/A□
□==========================
41.=== msvcrt.dll ==== strncmp□
STATSpaladionnetworks (heap)□N/A□N/A□N/A□
□==========================
42.=== msvcrt.dll ==== strncmp□
STATSpaladionnetworks (heap)□N/A□N/A□N/A□
□==========================
43.=== msvcrt.dll ==== strncmp□
```

One can see, the user input are mostly created on heap and not stack.