

- **Summary:**
 - This is just a [x64dbg](#) script system support.
 - System export 2 functions: Call & Logs.
Call FunctionsName:
 - Used to call the functions you already written in GUI.
 - Call can be standalone and run script from GUI or command line.
 - logs input
logs input,outputpath
 - System use Fast Colored TextBox for GUI Syntax Highlighting and AutoComplete, Thanks for ([Pavel Torgashov](#)).
 - It supports most functions in x64dbg.
 - It's just to make your life easy that's all.

Notes:	
Main Function is Call	<div>Call FunctionName Sample: Call Test Use Copy button to copy this line . <div>Test ADD Test Copy</div> You can use this command from command line or on BP dialog box </div>
All number in the script window are in hex mode.	0x0000000000000030
' is the quoted symbol to use for quoted-string	<div>Sample: 'this is test' use only ' " this will not replaced ' , so use ' for the strings " it will be like any other character</div>
You can use + to joint strings to gather.	<div>'this value:' + var + 'same value as:' + 10F U can add 2 string to gather Sample: Str x x='this is test'+': its work' don't left spaces between ' and + logs x ret Result: this is test: its work But if u make spaces like this: x='this is test' + ': its work' Result: this is test : its work</div>
case sensitive	<div>RAX is not same rax Please take care to write the words with case sensitive str not STR int not Int and so on</div>
You can shift between types by assign variable to other	<div>int x,10 hex y,25 x=y // u will get the integer value from the hex same for string if it's valid integer. Logs x Result : x=37</div>
F5 bottom	<div>Used to run the script from the GUI same <div>Run</div></div>
Ctrl + s	<div>To save the script on GUI same <div>SAVE</div></div>
ret	<div>Each function should end with ret Script will stop at any line where ret exist ret can be at any line</div>
Redefine Variable	<div>U can redefine any variable in any step of the script, and u can change its type too.</div>
Brackets	<div>Any function begins at the beginning of the line not use bracket Other functions inside the command should use brackets Except python functions</div>

Functions Defineder Coded By AhmadMansoor 2022

Test ADD Test Copy Delete SAVE Cancel Run ☐ Run On Each Instruction

1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24

Add: add new Function, you will write the function at the textbox near ADD button.

Copy: to copy "Call FunName" to use it on commands and BP dialog box.

Delete: to delete the function.

Save: to save the function
each functions will be saved to the database with the name of the exe file in temp folder at x64dbg folder.
ShortCut : Ctrl+s

Cancel: to abrot saving.

Run: to run script from GUI
ShotrCut: F5

Run On Each Instruction: it will triger the script on each step of debug,it will be avaliavle soon

- **Define variables:**

This system supports this type of variables: int , intarr, str , strarr , hex , hexarr.

cmd	description	structure	sample	notes
int	normal integer number	int varname int varname,varvalue	int var int var,10F	In varvalue u can use functions or other variable value too like: int x,20 logs x hex y,x+10 logs y str z,y+x logs z ret Result: x=32 as integer 32 y=0x0000000000000030 as hex 0x0000000000000030 z=0x0000000000000050 as string value:0x0000000000000050
intarr	Array of normal integer number	intarr varname[size] intarr varname[size],first item value	intarr var[10] intarr var[10],10F	
str	string	str varname str varname,varvalue	str var str var,'this is sample'	
strarr	array of string	strarr varname[size] strarr varname[size], first item value	strarr var[10] strarr var[10],10F	
hex	integer number in hex mode	hex varname hex varname,varvalue	hex var hex var,10F	
hexarr	array of hex numbers	hexarr varname[size] hexarr varname[size], first item value	hexarr var[10] hexarr var[10],10F	

Sample:

```
1 int x, 0000000077325701
2 str jj, 'this is test'+$tid()+10
3 logs jj
4 jj=10+'Tet'+'heheh'+RAX
5 logs jj
6
```

- **Fast Access:**

System is support fast access to registers and memory direct commands

- Register:

Assign value to register: REG=any hex value

```
1 hex x, 25f
2 hexarr y[10], 30a
3 RAX=RAX+1 // change RAX value by add 1
4 EBX=y[0]+x-AX // assign EBX value usin x variable and y array and AX register
5 ret
```

Read register value: variable=REG

```
1 hex x, 25f
2 hexarr y[10], 30a
3 y[20]=RAX+1 // assign record 20 of Array y
4 x=y[0]+x-AX // assign x value usin y array and AX register
5 ret
```

```
1 hexarr x[25], 10
2 strarr API[3], 1
3 x[10]=$ArrayLen(x) + $ArrayLen(x) + 20
4 logs x[10]
5 API[2]=$ReadStr($mod.base(RIP)+3000+398)+$ReadStr($mod.base(RIP) +3000+ 398)
6 logs API[2]
7 RAX=qword ptr ds:[RAX+word ptr ds:[AX+10]]
8 x[20]=dword ptr ds:[RAX +10 + x[API[0]+10+x[0]]]
9 logs 'x[20]=' + x[20]
10 API[2]=10 + x[API[0]+10+x[0]]
11 logs API[2]
12 x[2]=x[0]+25
13 logs x[2]
14 ret
```

- Memory:

Assign value to memory directly or read memory to variable:

memory = hex value

var=memory

```
1 hex x
2 strarr y[10], 30
3 y[2]='C868E476000000002869E47600000000'
4 // read memory to variable
5 x=byte ptr ds:[RAX+1]
6 x=word ptr ds:[RAX - RBX]-byte ptr ds:[RDI]
7 x=dword ptr ds:[R8]
8 x=qword ptr ds:[RAX + byte ptr ds:[RAX]]
9 // write to memory
10 byte ptr ds:[RAX]=25
11 word ptr ds:[RAX]=RAX-1
12 dword ptr ds:[RBX]=y[0]+byte ptr ds:[RSI]
13 qword ptr ds:[R8+x]=byte ptr ds:[RAX]+9A10
14 // write n hex value to memory
15 [RAX]='C868E476000000002869E47600000000'
16 //or
17 [RAX]=y[2]
18 ret
```

- **Condition command:**

If command, syntax

If.cmdNumber condition	If.cmdNumber condition	If.cmdNumber condition
.....	Else.cmdNumber	repeat.cmdNumber
End.cmdNumber End.cmdNumber	End.cmdNumber

Sample:

```

1  str x, 'This is Test'
2  hex address, 0000000076DF1000
3  int i, 0 // counter
4  strarr y[10]
5  // if end
6  if.1 RAX >= 0
7  logs RAX
8  End.1
9
10 // if else end
11 if.2 x == $ReadStr(address)
12 logs 'This is Test'
13 Else.2
14 logs 'Not equal'
15 End.2
16
17 if.3 CIP != RAX
18 logs 'CIP=' + CIP
19 Else.3
20 logs 'RAX=' + RAX
21 End.3
22
23 // if repeat end, can be used as loop like to fill array items
24 if.4 i < $ArrayLen(y)
25 y[i] = 'RAX=' + RAX
26 i = i + 1
27 repeat.4
28 End.4
29 ret

```

- **Commands Supports:**

```

static array <String^>^ x64dbgFunList = gcnew array <String^> { "run ", "erun ", "serun ", "pause ", "DebugContinue ",
"StepInto ", "sti ", "eStepInto ", "esti ", "seStepInto ", "sesti ", "StepOver ", "eStepOver ",
, "seStepOver ", "StepOut ", "eStepOut ", "skip ", "InstrUndo ",
"bp ", "bph ", "bpm ", "bpdll ", "bpdll ", "bpdll ", "SetExceptionBPX ", "bpgoto ",
"bpgoto ", "bpc ", "bpe ", "bpd ", "bpc ", "bpe ", "bpd ", "bphc ", "bphe ", "bphd ",
"bpmc ", "bpme ", "bpmd ", "bcdll ", "bpedll ",
"bpdll ", "DeleteExceptionBPX ", "EnableExceptionBPX ", "DisableExceptionBPX ", "SetBPXOptions ",
"SetBreakpointCondition ", "SetBreakpointLog ", "SetBreakpointLogCondition ", "SetBreakpointCommand ",
"SetBreakpointCommandCondition ", "SetBreakpointFastResume ", "SetBreakpointSingleshoot ", "SetBreakpointSilent ",
"ResetBreakpointHitCount ", "SetHardwareBreakpointCondition ", "SetHardwareBreakpointLog ",
"SetHardwareBreakpointLogCondition ", "SetHardwareBreakpointCommand ", "SetHardwareBreakpointCommandCondition ",
"SetHardwareBreakpointsilent ", "SetHardwareBreakpointSingleshoot ", "SetHardwareBreakpointFastResume ",
"SetMemoryBreakpointLog ", "SetMemoryBreakpointCondition ", "ResetHardwareBreakpointHitCount ",
"SetMemoryBreakpointCommandCondition ", "SetMemoryBreakpointCommand ", "SetMemoryBreakpointLogCondition ",
"SetMemoryBreakpointsilent ", "SetMemoryBreakpointSingleshoot ", "SetMemoryBreakpointFastResume ",
"SetLibrarianBreakpointLog ", "SetLibrarianBreakpointCondition ",
"ResetMemoryBreakpointHitCount ", "SetLibrarianBreakpointCommandCondition ", "SetLibrarianBreakpointCommand ",
"SetLibrarianBreakpointLogCondition ", "SetLibrarianBreakpointsilent ", "SetLibrarianBreakpointSingleshoot ",
"SetLibrarianBreakpointFastResume ", "SetExceptionBreakpointLog ", "SetExceptionBreakpointCondition ",
"ResetLibrarianBreakpointHitCount ", "SetExceptionBreakpointCommandCondition ", "SetExceptionBreakpointCommand ",
"SetExceptionBreakpointLogCondition ", "SetExceptionBreakpointsilent ", "SetExceptionBreakpointSingleshoot ",
"SetExceptionBreakpointFastResume ", "ResetExceptionBreakpointHitCount ", "createthread ", "suspendthread ",
"switchthread ", "resumethread ", "suspendallthreads ", "killthread ", "setthreadname ", "setthreadpriority ",
"resumeallthreads ", "alloc", "memset ", "memcpy ", "free ", "setpagerights ", "savedata ", "DisablePrivilege ",
"EnablePrivilege ", "handleclose ", "dbsave ", "dbload ", "dbclean ", "commentset ", "commentdel ", "commentclear ",
"labelset ", "labeldel ", "labelclear ", "bookmarkset ", "bookmarkdel ", "bookmarkclear ", "functionadd ",
"functiondel ", "functionclear ", "argumentadd ", "argumentdel ", "argumentclear ", "findall ",
"findallmem ", "findasm ", "findref ", "strref ", "modcallfind ", "setmaxfindresult ", "doSleep ",
"HideDebugger", "loadlib ", "asm ", "setcmdline ", "Fill ", "logs ", "pythonBase ", "py_define ", "$ResizeArray "
};

```

You can find reference for all Commands at x64dbg help site:

<https://help.x64dbg.com/en/latest/commands/index.html>

```

2  asm CIP, 'mov rcx, qword ptr gs:[0x000000000000000030]', 0
3  ret

```

```

1  strarr x[10], 'byte ptr ds:[' + RAX + ']'
2  hexarr y[3]
3  str fl
4  intarr v[16], 25
5  y[0] = RAX + AX + EBX - byte ptr ds:[RAX] + 25
6  bp RAX, "Test", ss
7  bph $mod.base(CIP) + 390E
8  fl = $dis.len(RIP)
9  logs 'First excption' + fl
10 ret

```

```

1  hexarr IATTable[1], RAX
2  IATTable[0] = IATTable[0] + 10
3  logs IATTable
4  //RAX = IATTable[0] + 10
5  logs IATTable[0]
6  Fill IATTable[0], 25
7  dword ptr ds:[IATTable[0]] = 105
8  ret

```

- **Extra Commands:**

1- Logs: log anything u want like variables or anything

Syntax :

logs anything

or

logs anything, path(to save to file) // this will log value to a file

```
8 logs RAX, 'C:\test.txt'
```

Sample:

```
strarr x[10], 'byte ptr ds:['+RAX+']'
hexarr y[3]
str fl
intarr v[16], 25
y[0]=RAX + AX+EBX - byte ptr ds:[RAX]+25
bp RAX, "Test", ss
//bpc "Test"
fl=$dis.len(RIP)
logs 'First exception' + fl
ret
```

- 2- \$ commands: it's two parts x64dbg part and script part
- x64dbg parts:

```
static array <String>^ scListConst = { "$disasm.sel()", "$dump.sel()", "$stack.sel()", "$peb()", "$teb()", "$tid()", "$mod.main()",
"$GetTickCount()", "$func.start()", "$func.end()", "$ref.count()", "$ex.firstchance()", "$ex.addr()",
"$ex.code()", "$ex.flags()", "$ex.infocount()", "$result"};

static array <String>^ scListpara = { "$src.disp()", "$src.line()", "$mod.party()", "$mod.base()", "$mod.size()",
"$mod.hash()", "$mod.entry()", "$mod.system()", "$mod.user()", "$mod.rva()", "$mod.offset()", "$mod.isexport()",
"$bswap()", "$utf8()", "$utf16()", "$mem.valid()", "$mem.base()", "$mem.size()", "$mem.iscode()", "$dis.len()",
"$dis.iscond()", "$dis.isbranch()", "$dis.isret()", "$dis.iscall()", "$dis.ismem()", "$dis.isnop()", "$dis.isunusual()",
"$dis.branchdest()", "$dis.branchexec()", "$dis.imm()", "$dis.brtrue()", "$dis.brfalse()", "$dis.next()", "$dis.prev()",
"$dis.iscallsystem()" };
```

- scListConst : commands not take argument's.
- scListpara : commands take one argument.

We got them by typing \$ and the Auto List will show them.

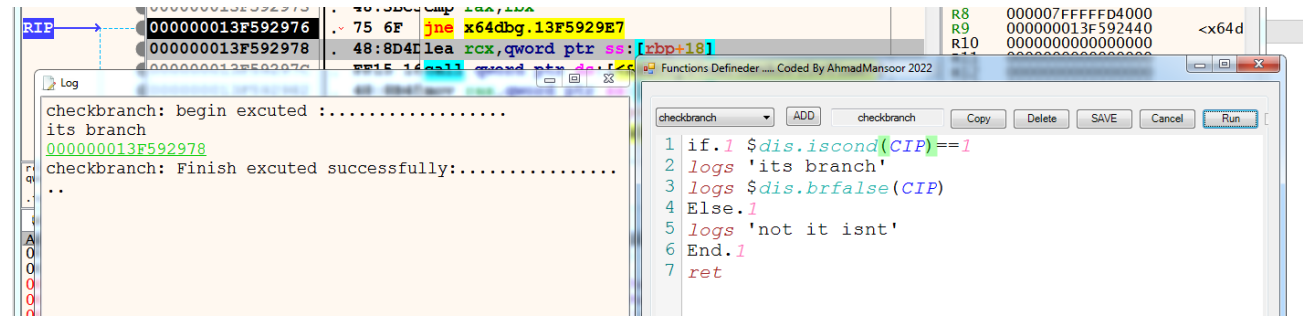
reference:

<https://help.x64dbg.com/en/latest/introduction/Expression-functions.html?highlight=mod.entry#modules>

sample:

```
1 hex ss, 15
2 str Val, 10
3 strarr gg[2], 15
4 str Val, 30
5 if.1 gg[0]==15
6 ss=$peb()
7 ss=$mod.base($mod.entry(CIP))
8 logs ss
9 logs 'before else'
10 Else.1
11 ss=$dump.sel()
12 logs ss
13 ss=$peb()
14 logs ss
15 logs 'after else'
16 End.1
17 ss=$teb() + $dump.sel()
18 //ss=qword ptr ds:[RAX]
19 logs ss
20 ret
```

We can use the condition commands too like check if branch or not.



- script part: this extra function it could modify later according to the needed of the users:

```
static array <String>^ scListextra = { "$ReadStr()", "$ArrayLen()" };
// ReadStr(Address)      ResizeArray(VarArray)
```

\$ReadStr: will enable u to read string at address if valid.

```
1 str GetData
2 GetData=$ReadStr($mod.base(RIP)+3000+398)+$ReadStr($mod.base(RIP) +3000+ 398)
3 str subStr
4 pythonBase 'c:\python27-x64\python.exe'
5 py_define createfile, 'D:\test.py'
6 py_define printarg, 'D:\test.py'
7 py_define Substring, 'D:\test.py'
8 logs GetData
9 GetData='This is Test'
10 logs GetData
11 logs '/////////'
12 subStr=py.Substring(py.Substring(GetData, 0, 11), 0, 5)+py.Substring(GetData + 10, 0, 5)
13 logs subStr
14 logs '/////////'
15 py.createfile()
16 py.printarg('This is From python')
17 ret
18
```

\$ArrayLen: function to resize the array size u can increase or decrease.

Sample:

```
ArrayLen(arrayVar,1) //increase arrayVar by 1
ArrayLen(arrayVar, FFFFFFFFFFFFFFFF) // decrease arrayVar by 1
```

FFFFFFFFFFFFFFFF=-1 in x64 system

FFFFFF=-1 in x32 system so take care when u write sign values.

```
24 if.4 i< $ArrayLen(y)
25 y[i]='RAX'+RAX
26 i=i+1
27 repeat.4
28 End.4
29 ret
```

- **Python:**

Now you can define python commands and use them in the script direct, you can call function with many arguments.

Limitation: you can get one value as return value.

It will support array return later.

How it works:

- pythonBase : it used to define the path of the python u used , it should defined at the top before any call.

pythonBase pythonPath

```
4 pythonBase 'c:\python27-x64\python.exe'
```

- py_define: it will define the function u want to call and script path.

Syntax : py_define FunctionName,ScriptPath

```
5 py_define createfile, 'D:\test.py'
```

- py : it used to call the function that all ready define it in py_define.

U can call functions anywhere in the command or as standalone function.

Sample:

```
1 str GetData
2 GetData=$ReadStr($mod.base(RIP)+3000+398)+$ReadStr($mod.base(RIP) +3000+ 398)
3 str subStr
4 pythonBase 'c:\python27-x64\python.exe'
5 py_define createfile, 'D:\test.py'
6 py_define printarg, 'D:\test.py'
7 py_define Substring, 'D:\test.py'
8 logs GetData
9 GetData='This is Test'
10 logs GetData
11 logs '////////////////'
12 subStr=py.Substring(py.Substring(GetData, 0, 11), 0, 5)+py.Substring(GetData + 10, 0, 18)
13 logs subStr
14 logs '////////////////'
15 py.createfile()
16 py.printarg('This is From python')
17 ret
```

Python script structure should be similar like this:

<pre>1 import sys 2 def pythonMessage(name): 3 print ("printarg: " + name) 4 def createfile(): 5 f = open("D:\\test.txt", "w+") 6 f.write("test") 7 f.close() 8 def printme(): 9 pythonMessage(sys.argv[1]) 10 def printarg(arg): 11 print ("printarg: " + arg) 12 f = open("D:\\test1.txt", "w+") 13 f.write(arg) 14 f.close() 15 def Substring(): 16 start=int(sys.argv[3]) 17 end=int(sys.argv[4]) 18 print (sys.argv[2][start:end]) 19 if sys.argv[1] == "printme" : 20 printme() 21 if sys.argv[1] == "printarg" : 22 printarg(sys.argv[2]) 23 if sys.argv[1] == "createfile" : 24 createfile() 25 if sys.argv[1] == "Substring" : 26 Substring() 27</pre>	<p>argv[0] : script path argv[1] : function name argv[.....] : arguments of the functions</p>
-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------	-------------------------------------------------------------------------------------------------------

- **Search Functions:**

findall
findallmem
findasm
findref
strref
modcallfind

This Should be done like this, Define any Array with any size no issue then
Assign this command to the array direct and the function will automatically will fill the array.
Syntax:

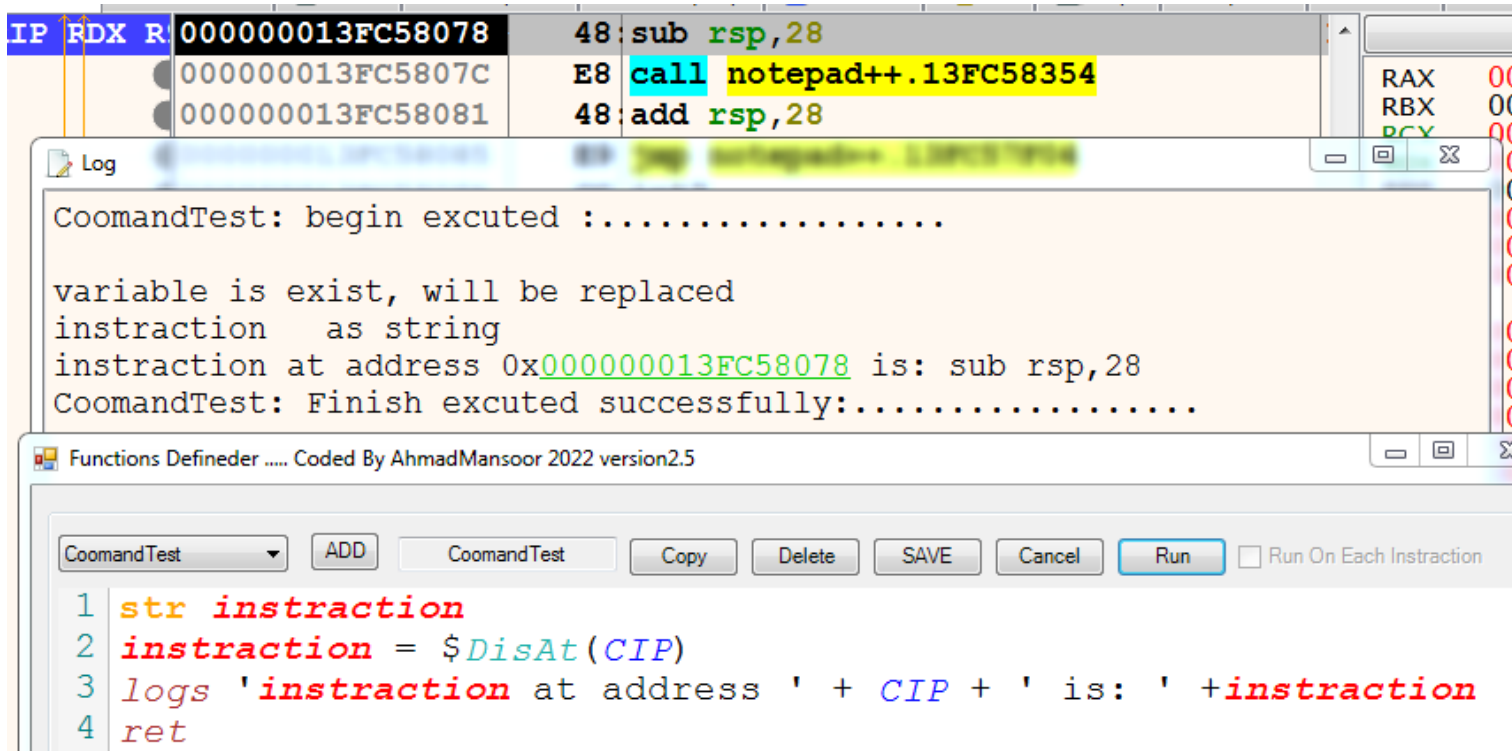
Array Var = function

```
1 hexarr y[1]
2 y=findref 000000013F5B228C
3 y=findall 000000013F331000, '4833C44889842490040000488BD9488D542470'
4 y=findasm 'mov rax, rsp'
5 k=findasm 'mov qword ptr ds:[rax+0x8]'
6 ret
7
```


X64dbgScript plugin v2.5

What's new in version 2.5:

- Fix bug update array length from search functions.
- Add new Functions : \$exepath() , \$exename() , \$exebase() , \$Sectionbase(addr) , \$ModuleBase(ModName) , \$SectionSize(addr) , \$DisAt(addr) .
- Support sign values.
- **\$DisAt(addr)** : Disassemble instruction at address .



The screenshot displays the X64dbgScript plugin v2.5 interface. At the top, a table shows memory addresses and instructions:

IP	RDX	R
000000013FC58078		48: sub rsp,28
000000013FC5807C		E8: call notepad++.13FC58354
000000013FC58081		48: add rsp,28

Below this, a 'Log' window shows the execution of the 'CoomandTest' function:

```
CoomandTest: begin excuted :.....  
variable is exist, will be replaced  
instruction as string  
instruction at address 0x000000013FC58078 is: sub rsp,28  
CoomandTest: Finish excuted successfully:.....
```

At the bottom, the 'Functions Definer' window shows the code for 'CoomandTest':

```
1 str instruction  
2 instruction = $DisAt(CIP)  
3 logs 'instruction at address ' + CIP + ' is: ' + instruction  
4 ret
```

- **\$Exepath()** : get exe path without exe name , just the path.
- **\$Sectionbase(addr)** : get the base of section by memory address inside it.
- **\$ModuleBase(ModName)** : get module base by it's name.
- **\$SectionSize(addr)** : get section size by memory address inside it.

Address	Size	Info	Log
0000000077340000	0000000000001000	user32.dll	Test: begin excuted :..... variable is exist, will be replaced x=C:\Program Files\Notepad+\ as string C:\Program Files\Notepad+\
0000000077341000	00000000000081000	".text"	
00000000773C2000	00000000000010000	".rdata"	
00000000773D2000	00000000000002000	".data"	
00000000773D4000	000000000000A000	".pdata"	
00000000773DE000	0000000000005B000	".rsrc"	
0000000077439000	00000000000001000	".reloc"	
0000000077440000	0000000000001000	kernel32.dll	
0000000077441000	0000000000009B000	".text"	
00000000774DC000	0000000000006E000	".rdata"	
000000007754A000	00000000000002000	".data"	variable is exist, will be replaced y=notepad+.exe as string notepad+.exe C:\Program Files\Notepad\notepad.exe 0x000000013F980000 Sectionbase: 0x000000013F981000 000000013F980000 module base by name: 0x000007FEDF750000 Section Size: 0x00000000033E000 Test: Finish excuted successfully:.....
000000007754C000	000000000000A000	".pdata"	
0000000077556000	0000000000001000	".rsrc"	
0000000077557000	0000000000008000	".reloc"	
0000000077560000	0000000000001000	ntdll.dll	
0000000077561000	0000000000123000	".text"	
0000000077684000	0000000000001000	".RT"	
0000000077685000	000000000000F000	".data"	
0000000077694000	000000000000E000	".pdata"	
00000000776A2000	0000000000005B000	".rsrc"	
00000000776FD000	00000000000002000	".reloc"	Reserved (000000000000FB000 Reserved KUSER_SHARED_D Reserved (000000000000F000 notepad++.exe
0000000077710000	0000000000001000	normaliz.dll	
0000000077711000	0000000000001000	".text"	
0000000077712000	0000000000001000	".rsrc"	
000000007EFE0000	0000000000005000	Reserved (000000000000FB000 Reserved KUSER_SHARED_D Reserved (000000000000F000 notepad++.exe	
000000007EFE5000	000000000000FB000	Reserved (000000000000FB000 Reserved KUSER_SHARED_D Reserved (000000000000F000 notepad++.exe	
000000007F0E0000	000000000000F0000	Reserved (000000000000FB000 Reserved KUSER_SHARED_D Reserved (000000000000F000 notepad++.exe	
000000007FFE0000	0000000000001000	Reserved (000000000000FB000 Reserved KUSER_SHARED_D Reserved (000000000000F000 notepad++.exe	
000000007FFE1000	000000000000F000	Reserved (000000000000FB000 Reserved KUSER_SHARED_D Reserved (000000000000F000 notepad++.exe	
000000013F980000	0000000000001000	notepad++.exe	
000000013F981000	00000000000033E000	".text"	Reserved (000000000000FB000 Reserved KUSER_SHARED_D Reserved (000000000000F000 notepad++.exe
000000013FC8F000	000000000000FC000	".rdata"	
000000013FDB8000	00000000000026000	".data"	
000000013FDE1000	00000000000018000	".pdata"	
000000013FDFC000	000000000000164000	".rsrc"	
000000013FF60000	0000000000006000	".reloc"	
000007FEDF750000	0000000000001000	sensapi.dll	
000007FEDF751000	00000000000002000	".text"	
000007FEDF752000	00000000000002000	".text"	
000007FEDF753000	00000000000002000	".text"	

Functions Defined Coded By AhmadMansoor 2022 version2.5

Test ADD Test Copy Delete SAVE Cancel Run ☐ Run On Each Instruction

```

1 str x,$exepath()
2 logs x
3 str y,$exename()
4 logs y
5 logs x+y
6 logs $exebase()
7 logs 'Sectionbase: ' + $Sectionbase(000000013FC5808A)
8 logs $mod.base(000000013FC5808A)
9 logs 'module base by name: ' + $ModuleBase('sensapi.dll')
10 logs 'Section Size: ' + $SectionSize(000000013FC5808A)
11 ret

```

X64dbgScript plugin v2.0

What’s new in version 2.0:

- Add commentlist , labellist , bookmarklist , functionlist , argumentlist , ref.addr() commands
- ReadFile Function:
Syntax :
varArray = ReadFile filepath
varArray any array variable with any size and prefer to be =1 as the function will auto fill the array with new items u can get the new size of the array later by \$ArrayLen() function.
If the file have integer or hex value u can define int or hex array , ReadFile will check the value before it assign it .
If u don’t know the type of data u can just define str array.
Sample:

```
1 strarr re[1]
2 re=ReadFile 'D:\Func.txt'
3 ret
.
```

- Add Shell function it’s same Shell command to run files or open folder
- Syntax :
Shell (file/folder)Path,wait(true,false)
Wait is bool value

True: it will keep the script wait till shelled application closed .

False : this is default options it will just shell the file

```
1 Shell 'C:\windows\system32\calc.exe' // it will not wait calc to exit
2 ret
3
4 Shell 'C:\windows\system32\calc.exe',true // it will wait calc to exit
5 ret
6
7 Shell 'C:\windows\system32\calc.exe',1 // it will wait calc to exit
8 ret
```

Hotkeys

The control supports following hotkeys:

- Left, Right, Up, Down, Home, End, PageUp, PageDown - moves caret
- Shift+(Left, Right, Up, Down, Home, End, PageUp, PageDown) - moves caret with selection
- Ctrl+F, Ctrl+H - shows Find and Replace dialogs
- F3 - find next
- Ctrl+G - shows GoTo dialog
- Ctrl+(C, V, X) - standard clipboard operations
- Ctrl+A - selects all text
- Ctrl+Z, Alt+Backspace, Ctrl+R - Undo/Redo operations
- Tab, Shift+Tab - increase/decrease left indent of selected range
- Ctrl+Home, Ctrl+End - go to first/last char of the text
- Shift+Ctrl+Home, Shift+Ctrl+End - go to first/last char of the text with selection
- Ctrl+Left, Ctrl+Right - go word left/right
- Shift+Ctrl+Left, Shift+Ctrl+Right - go word left/right with selection
- Ctrl+-, Shift+Ctrl+- - backward/forward navigation
- Ctrl+U, Shift+Ctrl+U - converts selected text to upper/lower case
- Ctrl+Shift+C - inserts/removes comment prefix in selected lines
- Ins - switches between Insert Mode and Overwrite Mode
- Ctrl+Backspace, Ctrl+Del - remove word left/right
- Alt+Mouse, Alt+Shift+(Up, Down, Right, Left) - enables column selection mode
- Alt+Up, Alt+Down - moves selected lines up/down
- Shift+Del - removes current line
- Ctrl+B, Ctrl+Shift-B, Ctrl+N, Ctrl+Shift+N - add, removes and navigates to bookmark
- Esc - closes all opened tooltips, menus and hints
- Ctrl+Wheel - zooming
- Ctrl+M, Ctrl+E - start/stop macro recording, executing of macro
- Alt+F [char] - finds nearest [char]
- Ctrl+(Up, Down) - scrolls Up/Down
- Ctrl+(NumpadPlus, NumpadMinus, 0) - zoom in, zoom out, no zoom
- Ctrl+I - forced AutoIndentChars of current line