ODBJScript

Ollydbg is now javascriptable!

1. OVERVIEW:

1. ODBJScript is a 'yet another scripting language' plugin for ollydbg 1.10 that lets you script ollydbg. It brings to the reverse engineering world n other hopefully usefull, more common and widely spread language: jacascript!: hence, Objects, Arrays, Conditional statments, Math functions all this is at your hands your hands now. (THE POWER OF JAVASCRIPT)
2. ODBJScript uses spidermonkey JavaScript engine.
3. It supports almost all ODBGScript command (with JavaScript standards of course). Thus, assembly lovers won't be depersonalized, you will be able to use the $RESULT, $RESULT_1, $VERSION, bp(0x401000) as you love :-).

2. INSTALLATION:

1. Copy plugin to plugins path
2. From plugin's menu, select ODBJScript ->Settings and set absolute path to ollydbg.js 'header'.
3. Write your script
4. Run it from main menu of from ACPUDASM windows (right click)

3. KNOWN ISSUES:

1. **It's known that:** these first alpha versions will ***only implement ODBGScript like commands***. For ollydbg API, wait for release candidate versions.
2. **It's known that:** ***jsmain()*** is the mandatory function name, sort of main in c/c++. (.i.e every script must have a jsmain function that will be executed as entry.
3. **It's known that:** JavaScript doesn't use DWORD, char* …, (without ctype support, witch's the case here). So if you see these types in declarations, this refers to their JavaScript homologous:
   - char* → String
   - DWORD/ulong/int/double → numeric variable (the conversion is maid internally, or in some case, must be done explicitly).
   - Bool → Boolean
4. **It's known that:** because of the effort to keep syntax as similar as possible to ODBGScript's one, the documentation of every command is nearly the same one in ODBGScript help file. ***Exceptions*** will be included to ***ODBJScript chm file***.
5. **It's known that:** although the last semicolon isn't necessary in JavaScript, i keep writing it, a good (bad if you like) c/c++ habit.

6. **Due to** JavaScript engine standards and with the aim to keep things as simple and as optimized/fast as possible, I had to choose a way to represent standard registers and flag registers. So I had two choices:
   - Either to keep them as is (.i.e eax stands for eax) and keep updating them in callback odbg_pluginmainloop() → too much time consuming.
   - Or the second (the one I've chosen), witch's to represent them as functions with a private variable 'v' (.i.e value) with a getter/setter, so that they are updated/got only when needed, of course the bad side is an extra 2 characters that will be needed every time (I think that's somehow fair).
   - ***THUS***, every r32, r16, r8 or flag must be wrote like this examples:
     - eip.v += 4;
     - var tmp = eax.v;
     - cx.v++;
     - zf.v = true …

   Two exceptions till now are **xchg** and **log** functions witch are written like this:

     - xchg(eax, ebx);
     - xchg(al,cl);
     - log(ebx);

   getting/setting is done ***internally***;

7. **Due to** JavaScript engine standards, it's not possible to use 'bp' for both breakpoint instruction and r16 register bp. and because of the frequent use of it as breakpoint instruction; I decided to keep it for this last use, thus, if you want to refer to bp r16 register. ***THUS*** you've to use bp16 instead of bp. ex: bp16.v = 0xdead; or var v = bp16.v;
8. **It's known that:**
   - **due to** the frequent use of: (in the first hand)
     - tmp = dd(addr)
     - tmp = dw(addr) and
     - tmp = db(addr)

     The analogues of 'in assembly language':

     - mov tmp, dword ptr [addr]
     - movzx tmp, word ptr [addr] and
     - movzx tmp, byte ptr [addr] respectively

     The analogues of 'in ODBGScript language':

     - mov tmp, [addr], 4
     - mov tmp, [addr], 2 and
     - mov tmp, [addr], 1 respectively
   - **due to** the possible use of $RESULT variable after using them (in the other hand),

   I decided ***not to let them set $RESULT*** (although they should).

Here is an example of the usefulness of this

```
while (find(addr, "68????????81??24")) {
   tmp = dd($RESULT + 1);

   if(db($RESULT + 6) == 4) {
      // letting db() or dd() set $RESULT will bias next operations
      mov($RESULT + 1, tmp);
      mov($RESULT + 5, "90909090909090");
   }
   addr += 12;
}
```

9. **It's known that:** this is a beta version, perhaps with some of inconsistencies, bugs ..., please, don't run the plugin after serious work or reversing session.
10. **It's known that:**
    - **Due to** the use of multi-threading in this plugin to work,
    - **Due to** the non 'multi-threaded' structure of ollydbg,
    - **Due to** my limited knowledge in multi-threading coding,
    - **Due to** all of that, some functions (especially those calling Disasm: opcode, gci... will report errors or fail particularly when called in loops, I'm working on the possibility to entirely use a foreign library to replace the use of Disasm (***PEOPLE WITH IDEAS ABOUT SOLVING THIS ISSUE ARE REALLY WANTED HERE*** :))
11. **It's known that:** this plugin requires the use of the C/C++ like syntax to represent digits: ***90 will be interpreted as decimal 90***., if you want to refer to nop, you shall ***use 0x90*** and so on. (I'LL FORCE THE JAVASCRIPT TO USE HEXADECIMAL AS DEFAULT IN FUTURE RELEASES IN CHALLAH).
12. **It's known that:** these alpha/beta versions aren't 'ctype enabled' and only returned values from plugin's functions are tested to be 32bits, therefore, if you do some arithmetic on addresses/values, don't be surprised that code like the one below will never log 'success':

```
var val = DD(eip.v+1);    // val = 0xFFFFFFFF or -1
val += 1;                 // val now = 0x100000000 as JavaScript uses
                          //   64bits integers as built-in types
if(val == 0)
        LOG("success");
else
        LOG("error");

// To fix this for now, you have to keep 'anding' result after arithmetic
operations with 0xFFFFFFFF

var val = DD(eip.v+1);
val += 1;
val &= 0xFFFFFFFF;        // the fix
if(val == 0)
        LOG("success");
else
        LOG("error");
```

13. **ODBJScript** will support plugins (i.e. you'll be able to write ollydbg's plugins in JavaScript in challah in future.
14. **It's known that:** only ***all-uppercase*** or ***all-lowercase*** is allowed with functions names.

15. **It's known that:** the global variables are till now only possible through function. Or by using predefined ones: tmp0 to tmp4 and TMP0 to TMP4.
16. **It's known that:** in next versions, you will be able to choose between two forms of internal global variables names, with/without $.

4. COMMING SOON: (ANY SUGGESTION IS WELLCOME)

1. Support of remaining ODBGScript like commands.
2. Localization of ODBJScript

5. COMMING LATER: (ANY SUGGESTION WILL BE IGNORED)

1. JavaScript plugins support: Supporting ollydbg SDK APIs.
2. Support of many other tools (especially library ones: titanengine, imperec ...).
3. Support of stdio/stdlib/… so you can manipulate files, run system commands…

6. GREETING:

To every user of this plugin.

7. TO TESTERS:

For old guys, thank you for all you efforts.

For newer, please, if you want to really help pushing the project forward, every time you find a bug, try to locate the exact command that causes the crash for example (and perhaps the variant that doesn't), write it down to a txt file with your JavaScript comment style above it and with the error message from ollydbg log windows to make it easy for me to dig quickly, and rename the txt file to: user_date_nameofcommand.txt then SEND IT TO MY MAIL anderzool@hotmail.fr. ***BIG THANKS TO ALL ALPHA/BETA TESTERS.***

8. HISTORY:

27/01/2012 : first public release (version 1.1.956 Beta)

AT4RE

27/01/2012