

Proview 1.7a



PVDasm Information:

PVDasm, what is it?

What's New in PVDasm 1.7a

Legal Information

Features

FAQ

Plugin System:

Plugin Messages

Tutorials

1. Masm Source Code Generation Using PVDasm (Updated)
2. Creating Detailed Disassembly using MAP Files (NEW)

PI_GETASM

PI_GETASM

An Plugin sends an PI_GETASM message to disassemble a vector of opcodes and place result at DISASSEMBLY struct pointed by IParam

```
//  
// DEFINED AS:  
//  
#define PI_GETASM          WM_USER+PI_BASE_MSG  
  
PI_GETASM  
wParam = (WPARAM) (LPCH)      lpOpVector; // address of Opcode Vector to disassemble  
IParam = (LPARAM) (DISASSEMBLY*) lpDisasm;  // address of Disassembly struct to fill
```

Parameters:

lpOpVector

Value of wParam. Points to the Opcode vector to disassemble by the disassembler engine

lpDisasm

Value of IParam. Pointer to a DISASSEMBLY structure that receives the disassembled data sent by lpOpVector.

Return Values:

(None)

when lpDisasm is filled, message is over and control back to the Plugin.

DISASSEMBLY

DISASSEMBLY

The DISASSEMBLY struct contains information about a decoded Instruction.

typedef struct Decoded

```
DWORD   Address;    // Current address of decoded instruction
CODE_FLOW CodeFlow; // Instructions: Jump or Call
BYTE    OpcodeSize; // Opcode Size
BYTE    PrefixSize; // Size of all prefixes used
char     Assembly[128]; // Menemonics
char     Remarks[256]; // Menemonic addons
char     Opcode[25];  // Opcode Byte forms
```

DISASSEMBLY ;

Members:

Address

Specifies the Address of Current decoded Instruction.
This member does not get Flushed by PI_FLUSHDISASM Message.

CodeFlow

<CODE_FLOW>

OpcodeSize

Size of Decoded Opcode (without PreFix Size).

PrefixSize

Size of Prefix (if used on the instruction).

Assembly

The Decoded Opcode's Assembly Syntax.

Remarks

Remarks Inserted by the disasm engine.

Opcode

String representation of the Opcode we decoded.

CODE_FLOW

CODE_FLOW

The Code_Flow struct contains information about branch Instruction such as: Call/Jxx/Ret and thier Size: SHORT / NEAR

The Code_Flow struct is a part of the [DISASSEMBLY](#) struct.

typedef struct Code_Flow

```
bool Jump;    // Instruction is a Jcc / jmp
bool Call;    // Instruction is a Call
bool BranchSize; // Short / Near
bool Ret;     // Instruction is Ret
```

CODE_FLOW;

Members:

Jump

If Instruction is a Jump Instruction family (JNZ,JMP,JZ..etc), this member is TRUE, otherwise FALSE.

Call

If Instruction is a CALL Instruction, this member is TRUE, otherwise FALSE.

BranchSize

Specifies the Size of Jump:

```
#define NEAR_JUMP 0
#define SHORT_JUMP 1
```

Ret

If Instruction is a Return Instruction, this member is TRUE, otherwise FALSE.

PI_FLUSHDISASM

PI_FLUSHDISASM

An Plugin sends an PI_FLUSHDISASM Message in order to clear the DISASSEMBLY struct Member

```
//  
// DEFINED AS:  
//  
#define PI_FLUSHDISASM WM_USER+PI_BASE_MSG+2  
  
PI_FLUSHDISASM  
wParam = None //Not in use  
lParam = (LPARAM) (DISASSEMBLY*) lpDisasm; // address of Disassembly struct to flush
```

Parameters:

lpDisasm

Return Values:

None
(None) of lParam. Pointer to a DISASSEMBLY structure that receives the disassembled data sent by lpOpVector.
when lpDisasm is filled, message is over and control is back to the Plugin

PI_GETASMFROMINDEX

PI_GETASMFROMINDEX

An Plugin sends an PI_GETASMFROMINDEX message to retrieve the disassembled data from an Index specifies the current selected line in disassembly view.

```
//  
// DEFINED AS:  
//  
#define PI_GETASMFROMINDEX WM_USER+PI_BASE_MSG+1  
  
PI_GETASMFROMINDEX  
wParam = (WPARAM) (int) iIndex;  
lParam = (LPARAM) (DISASSEMBLY *) lpDisasm; // address of Disassembly struct to fill
```

Parameters:

ilIndex

Index of the item to get disassembled information from

lpDisasm

Pointer to a DISASSEMBLY structure that will be filled according to ilIndex

Return Values:

(None)

when lpDisasm is filled, message is over and control is back to the Plugin

PI_GETFUNCTIONEPFROMINDEX

PI_GETFUNCTIONEPFROMINDEX

An Plugin sends an PI_GETFUNCTIONEPFROMINDEX message to retrieve the Current Function Boundaries, According to the Current Address at Disassembly Window.

```
//  
// DEFINED AS:  
//  
#define PI_GETFUNCTIONEPFROMINDEX WM_USER+PI_BASE_MSG+3
```

PI_GETFUNCTIONEPFROMINDEX

wParam = Selected Index in Disassembly Window / &StartAddr // Pointer
lParam = &EndAddr // Pointer

Parameters:

StartAddr

Fill StartAddr with the current selected Index in Disassembly Window, and pass StartAddr by Address.

EndAddr

Fill with NULL and Pass EndAddr by Address

Return Values:

(None)

StartAddr / EndAddr gets filled with the function Boundaries.

Example:

```
// Inside the Plugin's Code
```

```
StartAddr = GetDisasmIndex(...); // retrieve Current Selected Index (Bogus Function)
```

```
SendMessage(  
    *PlgData.Parent_hWnd,  
    PI_GETFUNCTIONEPFROMINDEX,  
    (WPARAM)&StartAddr,  
    (LPARAM)&EndAddr  
);
```

```
// After the Message:
```

```
// StartAddr Holds Function's Start Address
```

```
// EndAddr Holds Function's End Address
```

What's New In Version 1.7a

-> 20.02.2010:

- * Fixed a buffer access violation bug in the PE Imports scanner, thanks SecMAM for the bug report.□□
- * Increased buffers size for extra caution.

-> 28.03.2009:

- * Fixed a bug that caused PVDasm to crash when, You disassemble the same file (After it was already disassembled). Sometimes Re-Disassembly is needed (in case of defining Function entries / Data entries), and the Database need to match the view.

-> 20.03.2009:

- * Fixed the corrupted resources (I.e: Menu bar icons), for some reason, last versions of VS (or my PC) corrupted them.

-> 2008-2009:

- * Gui changes / addons / regrouped
- * 64Bit of PVDasm has been compiled and tested under WinXP/Win7 64bit,
- * No support for disassembling PE64 Image files, however PE+ Header (64Bit) is supported.

-> 08.04.2005:

- * Some bugs fixed in the function EntryPoint editor, Which allowed address with addresses like '4010' to be inserted/updated in a function. PVDasm will now accept only Dword length strings, i.e: '00401000'

-> 24.03.2005:

- * PVDasm Long Name Debug Vulnerability, yes that is correct, the problem was at 2 places where i did not used the MAX_PATH in 3 buffers, which caused pvdasm to crash. this version should fix this problem.

-> 12.02.2005:

- * Added File Map Export (File->Produce->Map File).
- * Updating a Function's Name, will now, Be changed in PVDasm without ReDisassembly.
- * updated **IDA2PV.idc**.

-> 01.02.2005:

- * updated IDA2PV.idc file to export the function's Names. MAX importing now supports reading the Function name from the .map file. With the proper functions names.

-> 29.01.2005:

- * Updated The MASM Wizard tool, Lines which points to a string data, Defined in the .DATA Section by PVDasm, Will now be fixed from: e.g: 'PUSH 12345678' TO: 'PUSH OFFSET <MY_STR_NAME>'
- * Added some Exception Handlers for various code places.
- * Added a missing code in the MASM Wizard.
- * Added Duplicate .DATA Lines Remover, created by the MASM Wizard.
- * Added more default Libs/Incs created by the MASM Wizard.
- * Fixed a bug when Wizard sees 0x0A, the output is a new line.
- * Fixed a bug when loading a Project and, Saving the project back again.

-> 25.01.2005:

- * Added A fix for latest reported Buffer Overflow, when reading the Imports.
- * Added more Plugin Messages (By Special Request)
- * Save/Load project now support function names.

-> 22.01.2005:

- * Fixed more general Bugs
- * Functions can now be renamed.
- * Masm Wizard now supports Functions with names.
This means, the source create will now have:
Call "MyFunc" insted of Call XXXXXXXX.
Note: function will be shown only if a function, has a name, else it will remain Call XXXXXXXX

-> 13.01.2005:

- * Fixed Reported Disasm bugs by 'mcoder'
To the Disasm Engine Core - thanks mcoder.
Thanks for the bugs.
- * Missing not handled Opcodes, in the Disasm engine core added.

-> 27.11.2004:

2 major bugs has been discovered,
and i had to release this as a new version,
to keep confuse our of users.

- * Fixed a nasty error,
an invalid entrypoint pointer was defined,
and caused a nasty crash.
Note: it has nothing related with the 1.5e new code,
it's regarding the FirstPass anaylzer.
- * nasty freez has been fixed,
it has been told to me by various members [thanks elfz!],
it seems that when the mouse is leaving the disasm client,
window and using the drag bar for the listview,
pvdasm freezes and die.
this is because the tooltip code is based on the,
mouse_movie event, and leaving the client still,
trigger the event and the calculation just freezes PVDasm.
so i set a restrictive border between the bar and the mouse.
It should work fine now.

-> 23.11.2004:

- * Added 2 more Plugin Messages.
- * **GUI Fixes** for User's comfrot (As reported by 'elfZ'):
 1. "Decode new file dialog's tab order is quite messed up" - *Corrected*
 2. "Ctrl-Space (references) opens a window,
but I can't navigate to any of address using keyboard,
(dblclick is req-d). Also, the "esc" doesn't close xref window" - *Added*
 3. "Ctrl-C as a shortcut to 'go to start of code' isn't really the best choice,
I'd expect it to be 'copy selected text to clipboard'. - *Corrected + More Fixes*
 4. Tab Order fixes in all dialogs.
 5. More ShurtCuts has been added to support the keyboard users.
Note: the 'Plugin' menu is dynamicly created by pvdasm,
so it can't be associated with 'Alt+P' shurtcut.
 6. Fixed some typos, added 'Add Coment' to menu and shortcut key.

7. Added a 'Show String' for instructions like: LEA ESI,DWORD PTR [00403012]
It will be shows as, e.g: LEA ESI,DWORD PTR [00403012] ASCIIZ "My String",0

* **ToolTip Helper:** When Hovering over address of instructions like: (CALL XXXXXXXX / Jxx XXXXXXXX),
a window will pop up and will show the 'target' jump code. (tooltip alike help).

Note: 'ESC' key to close the window.

* **MAP Reader :**

PVDasm can load a **Specific** MAP file generated by IDA from the script: "Map/pvmap.idc",

This will create a much more analyzed disassembly code that is correct and readable.

Note: See tutorial Section on how to work with MAP files.

-> **24.08.2004:**

* Added 6 more Plugin Messages.

* End of a function is now marked in the last
Instruction of the function boundry as a part of a remark.

Example: RET ; proc_XXXXXXXX endp

* Loaded File can be released via Menu ('Close File')

* Added some bug fixes (after file is closed).

* Fixed decoding of the Opcode sequence: '8C3F',
"Old Display: MOV WORD PTR DS:[EDI],+", // pointer passed the segment array onto the next one
"New Display: MOV WORD PTR DS:[EDI],SEG?",

-> **25.07.2004:**

* Fixed is a nasty out of range pointer,
Exception in the latest built (1.5b) on big files,
Regarding the visual xreferences.
In the latest build the disasm will be corrupted
When loading big files (>300k+-) and trying,
To access the data which is invalid.
Visual xreferences will now be added,
After the disasm process, so while disasm will,
Be visible, xref will be added afterward.
There will be times where there wont be any xref
Due to an internal exception caught by pvdasm,
Mostly accur on big file with bad disassembly and
Mis-aligned code.

Note: the addresses references to the address will not
Be shown since 'Ctrl+Space' will display them all,
Or via the tool-bar button.

* Disassembly Bug fixed, an register was added where it shouldn't.
Corrected "[ESP+ESP+0Xh]" -> "[ESP+0Xh]",
Thanks to 'CoDe_InSide' for the bug report.

Optimized code: **MMX** version of StringHex->Hex conversion algorithm.

Optimized code: **MMX** version of strlen(); function

Note: if there are users with non **MMX** CPU enabled (support),
Please tell me so i will release pvdasm,
Version with a backward compability.

* 2 Bug fixes in the MASM Wizard, when deleting,
An Data string member, the string pointer got mixed up.
It will now show the good pointers so after deleting,
Everything will be shown properly.
And another bug when creating a new,
Function entry point than the data pointer got lost,

For each new 'disassembly' listing the,
Data pointer will be recalculated.

- * Export / Import Function Database added to the subroutine,
Maker dialog in the Masm Wizard.
This is because for any disassembly action,
The database is deleted.
Exporting the database could save a lot of hand redefining.

Note: if database exported and afterward,
A new function is defined within pvdasm, it does not exist,
In the exported database, there for either add the new addition,
To the database or redefine all from beginning.
Exported information is in this format:

```
[Export File Begin .xpr]
; each separated line must contain ""
number of functions
function_1 proc...
function_1 proto...
function_1 endp
function_1 Start address
function_1 End address
function_1 index ; this means the index in the comboBox (starting from 0)
[EOF]
```

- * Added a 'Auto Search' button to the MASM code builder,
If the selected function is an entrypoint and the caller,
Is a 'CALL' instruction, then the auto search will find the number,
Of parameters sent to the function and will automatically insert the,
PROC / PROTO lines in the edit boxes.

Note: it does not 'create' the function, only defines it,
User will have to press the 'Create' button,
To enable it in the source output.

- * Crash fixed on disassembly view save operation when,
User canceled the operation.

-> 02.07.2004:

* Masm Wizard tool Functionality Enhanced!!

Here's what new:

1. It will now clean: **LEAVE** instructions in procs.
2. it will fix **RET XXXX** -> **RET** instructions in procs.
3. Jumps and Calls are now automatically refers to their labels!

e.g:

```
'jnz 12345678' -> 'jnz ref_12345678'
..code here..
ref_12345678:
'call 87654321' -> 'call proc_87654321'
```

Note: Call to an api imports stays the same! i.e: 'Call MessageBoxA'

4. Procedure StackFrame clean-up, wizard will remove,
The preceding "PUSH EBP" / "MOV EBP,ESP" signatures from a proc.
e.g:
before:

```
proc_12345678 proc ...
push ebp ; setup,
mov ebp,esp ; procedure stack frame,
add esp,xxh ; for local variables.
; unaffected code like more 'push ebp'
; or 'mov ebp,esp' repeated instructions.
```

ret

after:

proc_12345678 proc ...

add esp,xxh ; leave room for local variables.

ret

This gives us a compilable source code (in most cases),
but not executable image! since we need to make hand modifications,
for stuff like 'global' vars..etc.

* Added a preceding zero (0) for a decoded instruction,

e.g: 'add esp,FFFFFFFFh' -> 'add esp,0FFFFFFFFh'

note: i haven't covered all the possibilities, so it will take a while.

-> 01.07.2004:

- * Added a small bug fix in the Disassembly engine.
- * Save/Load project supports 'Data In Code' & 'Functions EntryPoint' information, next time loading a project it will automatically relocate your analyzed work.
- * Bug Fixes in the Masm wizard.
- * Conditional or Unconditional jumps as references are, shown in a new column in the disassembly view.

-> 08.05.2004:

- * Corrected an Disassembly bug (hopefully, Won't damage other decodings), this is the change:
'PUSH DWORD PTR SS:[EBP+00]' -> 'PUSH DWORD PTR SS:[EBP+EAX+00]'
Thanks to _Seven_ for the bug report.
- * Added a MASM Source Code Wizard Creator (win32 asm exe's only), Check Tutorial Section in order to see how to use it.
- * Added a Produce asm option, usually to dump the disasm window.

-> 14.02.2004:

- * While DbIClick on an CALL<API> Instruction, PVDasm will jump to it's "JMP<API>" jump table.
- * Api Recognition (engine) Added.
PVDasm will attempt to add API Parameters,
In order to make disassembly more easy to read.
Recognition DataBase is saved in the .sig file.
It can be altered anytime for your own need anytime.
the file can be renamed to any name.
only 1 (main) sig file is supported at this moment.
- * Few add-ons here and there.
- * You can now transform/Define address range to Data/Function EntryPoint by
Selecting the addresses (in the disassembly view) and
press either:
Ctrl+Insert: #Define Data blocks.
Alt+Insert: #Define a Function(s) EntryPoint(s).
or using the contex menu for both missions.
- * Data/Function EntryPoint Managers can be accesses using,
Contex Menu (right click).

-> 10.02.2004:

- * Added a New Plugin Message for Developers.
- * Updated Help File.
- * Added an Option to Edit/Add/Remove Function's
Start/End Addresses Analyzed by the FirstPass.
This Gives Flexibility for the users to modify
The Bad Analisis created by the Incomplete FirstPass.

-> 26.01.2004:

- * Api Calls/Jmps , Jmps,Calls Highlighting Added,

- Use the Appearance Dialog to select your colors.
- * Right Click On Disasm window pops up a Menu. (For faster access)
- * Copy to File/Clipboard added, found in the right click popup menu.
- * Some Visual Fixes here and there (e.g: Imports/exports has new icons).
- * Added a new Plugin to the Pvdasm site (Chip8 Emulator)

-> 15.12.2003:

- * 2 new SDK Messages has been added, see the pvdasm.hlp file at the PluginSDK.zip to review them.
- * FAQ added.
- * Some Bug Fixes.

-> 13.12.2003:

- * Simple! FirstPass analyzer has been added. if program uses FirstPass, you will be able to modify/add/delete Bad/Good data addresses found, by the analyzer when perssing 'data segments' for the next disasm process. No Herueistics in the analyzer yet.

-> 12.12.2003:

- * Chip8 CPU Added to the CPU Category. You can disassemble with added/New CPUs only if they are not Win32 Images (MZ/PE). if Image Loaded is Binary, Select The CPU and press on 'Set', finally press ok. No need to set CPU for loaded Win32 images, they are automaticly slects x86 CPU.

-> 02.12.2003:

- * CodePatcher bug currected, where instructions, With Prefixes were missing the last bytes (prefix size was not added).
- * Added ability to create a custom data sections (segments), if you want to treat code as data, add rva start and rva end.

-> 11.11.2003:

- * Load/Save Project Options Added. There are 7 files for each saved project. (yeah i know kidna allot, but it's easier to handle rather than 1 bif file.)
- * Plugin SDK Added!: you can code your own plugins for PVDasm now. Put your plugin at the dir and ur done. note: plugin will be executed only if a file has loaded into Pvdasm, (no disassemble process require to execute the plugin).

PVDasm for now ships with this dll(s) list:

1. "Command Line Disassembler"

CLD.Dll - Plugin coded by me, the plugin gives you option to preview disassemble Opcode Vector from a command line alike mini-tool. Source for plugin included.

-> 01.11.2003:

- * Debug Window is now Dock-able (Via ToolBar or Menu).
- * Searching Withing the Disassembled code is no availble (be sure to check 'Match Case').
- * XReferences is not supported, if an address is being References from another, ToolBar Control will be availble, or a message will be writetn in the DebugWindow. Press Ctrl+Space, ToolBar Control or double click the Address to view XReferences to selected line, The Window of xreferences will be opened accouring to your Mouse Pointer Position!,

idea came from the intellicase window, I kinda like it ;).

- * HexEditor - AddIn Created for RadASM (By KetilO) Has Been 'Converted' By Me to, be Used Inside VisualC++ (For VC Example Check:

<http://radasm.visualassembler.com/projects/CustDemo.zip>).

If AddIn DLL is not found in the AddInsDirectory, you will not be able to access it via Preview (Run-Time).

- * String References & Import References Dialogs has been Changed, now you can perform, Better Search Within Them, and view 'more' Information rather than using a simple ListBox ;).
- * Disasm Bugs Fixes (Those who has been reported.)
- * Added Few More Seh Frames to avoid Crashes.
- * CodePatcher - Added Inline code patcher with Assembly Preview (After Patch) in same window. After Patch has been complete you can or not ReDisassemble your Project in order to see, Changes, iv done it because i want to avoid MisData information when patching new bytes, So better keep stuff linear insted of curved ;) (PV is pretty fast to do ReDisassemble anyway hehe). Access it by Double Click on Opcodes Culumn, ToolBar or Menu.
- * Gui Fixes/Edits (also fixed the bug in the disassembly appearance for the background color)

-> 12.10.2003:

- * Branch Tracing/Back Tracing has been added Using <- or -> Arrow Keys, opr trace in by, Double Click Jxx instruction in disassembly Widnow (Column). You can trace jxx/calls and return from them the same Way u traced them (No Metter how deep u traced!). Fast Tracing / Ret from tracing with left/right arrow keys, tool bar or from the menu.
- * Tool Bar Has been upgraded, more option added

-> 01.10.2003:

- * 100% Disassembly Speed!, you will notice a *huge!*, Speed difference from the last build. I think PVDasm can compete with the big boys now ;) (disasm speed) Although there might be still false disasm (still in testing mode)
- * Added Colors Schmes (softice/ida/ollydbg/w32asm/custom) to the disasm window.
- * Goto.. (address/entrypoint/code start) options added.
- * x8 Speed optimizations to the disasm engine core.
- * Known Bug: if file is over 5mb PV might be not responding for a while Because PV uses memory to store the information insted of a temp File (e.g: w32dasm). but it doesn't mean it doesn't work, after a while u will get result, so if you have more than 256mb mem, u will be fine during big files :) .
- * WinXP Theme (Manifest) Added to PV'S Gui (when XP theme shell is being used only).
- * Added String References with search dialog.
- * Added a custom dialog About => just to play with skinnble dialogs.

-> 27.08.2003:

- * Import resolving added.
- * Imports dialog with searching added.
- * PV Now Uses Virtual ListView to hold big amount of data.
- * PV Will Auto Allocate memory based on the disassembled code data.
- * Double Click a disassembled line will allow to Add/Edit comments.

-> 15.08.2003:

- * Disasm Engine Complete!! (except bugs i will find later :)) it isn't the fastest engine, but it suites me fine now (as a student :))
- * Here we go.. Full support for 0F Set. Meaning: MMX / 3DNow! / SSE / SSE2 instructions + prefixes support
- * Bug Fixes in disasm engine.
- * Disasm from EP option added.

-> 10.08.2003:

- * added Options to the disassembler menu
- * progress bar /percent added
- * force disasm's bytes from ep is now user defined (0-50 bytes). Note: Smaller number of byte can cause few instructions to be not well decoded.

-> 09.08.2003:

- * added 1/4 support for 0F Instruction set (JXX & 1 byte opcodes set)
- * added another opcode support (forgot to add it).

-> 08.08.2003:

- * Fixed some problems, add a forgotten opcode :D
- * Added option to force disassembly before EP (length not yet user defined)
- * Auto jump to EP, code start, address added.
- * added option to restart disassembly

-> 07.08.2003:

- * Opcode 0x0F remains to complete the Disasm engine.
So you will get from time to time some gaps if your
Exe is using its Set of instructions.
- * Process Viewer/Dumper supports 9x/2k/XP

-> 18.05.2003: * Disassembler implemented.

-> 10.02.2003: * Added Expots Viewer

-> 05.02.2003:

- * Added Import viewer at the
Pe editor/Directory viewer

-> 03.02.2003:

- * Added a pe rebuilder
fix the alignment and headers size,
as well as the sections [vsize=rsz / vaddr=raddr]
i cannot say it will rebuild it successfully,
notepad did work though =)
and i am opened to suggestions.
 - * Added a partial process dumper
you can choose how many bytes to dump
and what address to start from.
- notes: full/partial dump does not work under

PVDasm.exe (v1.7 32Bit) MD5 Hash: [65c0530b6285b097eec80749a632be26](#)

PVDasm.exe (v1.7 64Bit) MD5 Hash: [4de8a4af2fe75692c8c138fb5c5f5f16](#)

If This is not the Hash you get, the exe has been altered!

What Is Proview? (A.K.A: PVDasm)

PVDasm is a fully written from scratch, the disassembly engine has been coded by me and its free for useage.

Proview is my attempt to make my own disassembler as a part for school final project and for basic knowledge.

Pv is coded fully in C (VC.6), a bit of C++ , and some STL Templates for memory managment.

It would be nice if someone will try it out and give some response.

Proview also includes a simplified version of a pe editor and a process manager (if you wish to dump from mem).

i hope to add a basic gui debugger in the future as well - hopefully :)

Requirements:

Works under Windows 9x, NT/2000, XP,XP 64, Vista, Vista 64, Win7, Win7 64. PVDasm require over 256mb of memory when Disassembling Large binaries, Take it for your attention.

Supported processors:

1. Intel (C) 80x86 CPU,MMX, 3DNow!,SSEx Instructions.

2. Chip8 CPU.

Planned CPUs For Future: z80,Morotola 68k and some more.

Analysis. Analyzer recognizes procedures, strings embedded in code, calls to API functions, and The PE Format.

Plugins. You can add features to PVDasm by writing your own plugins (Check the SDK Functions).

thank you all for your support.

Legal Information

TM

PVDasm is owned by Bengaly (A.K.A as Ben), and it is a copyrighted (c) 2003-2010 software.
You can use this software freely without any form of registration, Time Limit and other restricting rules.
This software is intended for educational use only.

License Agreement

You are not allowed to modify, decompile, disassemble or reverse engineer the Software except and only to the extent that such activity is expressly permitted by applicable law. You are not allowed to distribute or use any parts of the software separately. You may make and distribute copies of this Software provided that a) the copy contains all files from the original distribution and these files remain unchanged; b) if you distribute any other files (for example, plugins) together with the software, they must be clearly marked as such and the conditions of their use cannot be more restrictive than conditions of this Agreement; and c) you collect no fee (except for transport media, like CD or diskette), even if your distribution contains additional files.

You are allowed to develop and distribute your own plugins for pvdasm which are Dynamic Link Libraries (DLL) that connect to the Software and make use of the functions implemented in the Software, free of charge provided that a) your plugins contain no features that persuade or force user to register them, or limit functionality of unregistered plugins; b) you allow free distribution of your plugins on the conditions similar to that of the Software, If you want to develop commercial plugin, please contact Author for a special Agreement.
The distribution includes files PSAPI.DLL which is a Microsoft (R) Redistributable file.

Some Words:

Pvdasm is allowed to be used on legal binaries, hence your own source code for binary decompilation and source reconstruction.

Please do not use this software (PVDasm) over signed, copyright or shareware binaries for your own illegal source retrieving and code reversing .

- Bengaly/2010

Plugin Messages

PVDasm Plugin Messages:

Each Plugin is derived from a Base plugin and WM_USER:

```
// PLUGIN BASE MESSAGES  
#define PI_BASE_MSG 200
```

1. PI_GETASM
2. PI_FLUSHDISASM
3. PI_GETASMFROMINDEX
4. PI_GETFUNCTIONEPPFROMINDEX
5. PI_GETENTRYPOINT
6. PI_PRINTDBGTEXT
7. PI_GETBYTEFROMADDRESS
8. PI_RVATOFFSET
9. PI_GETNUMOFSTRINGREF
10. PI_GETSTRINGREFERENCE
11. PI_SETCOMMENT
12. PI_ADDCOMMENT
13. PI_ADDFUNCTIONNAME
14. PI_GETFUNCTIONNAME
15. PI_GETCODESEGMENTSTARTEND

Related Structs:

1. DISASSEMBLY
2. CODE_FLOW
3. FUNCTION_INFORMATION

PVDasm's Features

PVDasm's Features:

- Multi CPU Disassembler. (x86 , Chip8)
- PE Editor.
- Hex Editor (Addin Coded by KetilO, Part of the RadASM Project, Custom Control).
- Process Manager/Dumper.

Inside Features:

- * Reads & Edits the PE (32Bit) / PE+ (64Bit) Image files.
- * Integrated Hex Editor.
- * Integrated Process Manager and Memory Dumper.
- * Source Code Generator (Currently only for MASM Compiler).
- * Plugin SDK Architecture.
- * Coloring Themes/Custom Themes for disassembly coloring.
- * Function Parameters Recognition.
- * Data/Function Entries Manger (Define your own data/code sections).
- * Produce PVDasm MAP and Support for IDA MAP Files (using ida2pv IDC script) for better analysis.
- * First Pass analyzer (Simple Analyzer).
- * Easy GUI Interface.
- * Code Patcher (Edit and apply Executable changes on the fly).
- * View/Search Function References and String References.
- * View Call/Jxx functions without the need of manual tracing.
- * Save and load projects.
- * Create and execute Scripts using PVScript Engine.

FAQ

PVDasm Official FAQ (01.07.2009)

=====

Quick Question Navigation:

1. When & Why PVDasm has been created?
2. Can PVDasm compete/replace the well known w32dasm?
3. How about IDA (pro) disassembler?
4. How PVDasm works internally?
5. Why PVDasm acts slow when disassembling big files (3MB+) ?
6. This is weird, I disassemble a file, but where is the [Program Entry Point]?
7. So PVDasm support FirstPass?
8. How does FirstPass works in PVDasm?
9. Does Proview supports Plugin SDK?
10. What Programming language Pvdasm is programed in:
11. Can I save/load disassembled file as project?
12. What is the 'Data Segments' button at the disassembly main Dialog?
13. Is PVDasm Multi-CPU Disassembler?
14. Weird, I can't see all of the CPUs when loading some images, why?
15. What else PVDasm supports:
16. What more features we will see implemented?
17. Why when i press on CodePatcher at some address inOrder to patch the bytes i get diff bytes?
18. Can PVDasm create source code from the disassembled executable?
19. The source that has created is not complete (or not compilable), where are all the stuff?

Q. When & Why PVDasm has been created?

A. it was create at the beginning of 2003,
The actual disasm engine has been finished around September-October/03.
and the rest are still w.i.p (work in progress) untill now (December/2003)
looks like allot of time indeed, but i was also in school, so i had other projects.
Why it was created, mostly for School Project and some knowledge & fun.

Q. Can PVDasm compete/replace the well known w32dasm?

A. hehe, well the answer is no!, at least not for now.
the problem with creating a disassembler is not only by
coding an disasm engine, but it is a disassembler who can
able to determine & analyze between Code & Data.
PVDasm is in its early stages of it's FirstPass analyze.

Q. How about IDA?

A. IDA is a professional tool, way more advanced & supported, also has a great team behind it.

Q. How PVDasm works internally?

A. the process it self is pretty straight forward:

1. Determine if file is PE (Valid Win Image executable) and Load the CPU processor's module.
If not, you can choose other CPUs.
2. Run FirstPass analyzer and build Data segments.
(You can build Functions EntryPoints & Xref, but its not very accurate,
How About Anti-Disasm? :))
3. Decode Code section and use the data Segments to differ between Code&Data
Visually, decode Xref,Imports on the way.

Q. Why PVDasm acts slow when Disassembling Big Files (3MB+)

A. Ok, this is how Proview works differnt than Other Disassemblers,
For the well known Win32dasm you notice it disassemble very slow (load big file)
This is because W32dasm creates a 'tmp' file at the 'windows' directory
with the size of, e.g: 70mb.
This gives the impression that w32dasm uses low memory usage.
On the other hand,
Proview uses memory only access, this gives the effect of fast decode & access.

There for, if you have 256MB of mem (default today) it will use the swap file when memory is needed by PVDasm, e.g: used more than 200mb of mem.

Why O Why you say?

think of that, future Computers will have huge amount of memory (e.g: 1GB-10GB)

there for 256 is very low and PVDasm will work flawlessly ;)

Thus HDD speed will increase too, but you can't even compare MEM access to HDD access.

i don't really sure about IDA or ollydbg, could be they uses 'tmp' files as well.

Q. This is weird, I disassemble a file, where is the [Program Entry Point]?

A. aha, you stumbled upon the hardest things a disassembler is facing at.
the problem is that when doing a linear disassembly, you can't recognize between 'Data' and 'Code'.

if for example you disassemble with FirstPass analyzer, you get:

```
00401000 B8 00000000  MOV EAX,0
00401005 33D2          XOR EDX,EDX
00401007 EB 03        JMP SHORT test.0040100C ; good jump
00401009 68 69 00        ASCII "hi",0           ; here is the data.
0040100C 6A 00          PUSH 0              ; perfect align
```

How would linear disassembly will do? ... very bad.

it will look like this:

```
00401000 B8 00000000  MOV EAX,0
00401005 33D2          XOR EDX,EDX
00401007 EB 03        JMP SHORT test.0040100C ; jump to where??
00401009 68 69006A00    PUSH 006A0069          ; our address align has been destroyed
0040100E 68 00304000    PUSH 00403000          ; ...
```

hmm..now you understand where the EntryPoint went to? :P ..

Yeah, so the problem is to find what is 'Code' and what is 'Data'

in order to know how to 'align' your addresses.

i hope you understand why sometimes we get bad disassembly.

Q. So PVDasm support FirstPass?

A. well yes, it has a very basic FirstPass analyzer, which does not contain
Any heuristic code to separate data from bad opcodes..etc.

Q. How The FirstPass works in PVDasm?

A. ok, well, the basic idea for FirstPass is and will remain:

"Code Flow Simulation" (CFS) - hence 'FirstPass'.

When a disassembler supports FirstPass it is called SecondPass disassembler.

ok so how it works in pvdasm:

Always Start in EntryPoint, Decode any Instruction,

Upon Jump Instruction (Jxx) or Call Instruction, we follow to that Address.

if call/jmp redirect to an Import Entry (i.e: Call MessageBox) ignore this branch.

The Simulation works just like the CPU:

Upon Call: Follow Call and Save Return Address (Next Instruction's Address)

in your simulated call stack.

Upon Direct Jump (JMP XXXXXXXX) follow immediately (here i saved the next instruction's

Address 'assuming' it is data).

Upon Conditional or UnConditional (JNZ,JZ,JG,JGE..etc) we MUST somehow take all paths.

You can take any path but you need to return to it in the end.

Upon Ret/Ret XXXX instruction we mark the end of a 'function/procedure' and we return to the Caller (CALL)

we also mark good code rangers e.g: 00401000-00401051

this way we can check if we jump/call to an address we always marked.

by this simulation we mark start-end of functions, xref,and data locations.

Q. Does Proview supports Plugin SDK?

A. yes, you can create Plugins to Proview.

PVDasm ships with a command line Disassembler plugin as an example + source code.

there is UnComplete messages which the plugins can send to proview.

i will add more in time.

Plugin will refuse to load if file is not loaded first.

Q. What Programming language Pvdasm is programed in:

A C & Win32API,C++ & STL and ASM (some algorithms)

Q. Can i save/load disassembled file as project?

A yes, it will create around 7-8 files per project (so be aware)

Q. what is the 'Data Segments' button at the disassembly main Dialog?

A even if FirstPass has been analyzed, it is at best to allow the user to define His OWN Data Segments to make disassembly align better.

Q. Is PVDasm Multi-CPU Disassembler?

A Yes, Proview supports for now 2 CPUs:

INTEL 80x86 CPU.

Chip-8 CPU (games running under this are: SpaceInvaders, Pong).

Q. weird, i can't see all of the CPUs when loading some images, why?

A the other CPUs will be enabled for non-pe Image.

for example, if you load an GameBoy rom, or any other Non-PE.

80x86 will be disabled and will show all current supported CPUs.

Q. What else does PVDasm supports?

A it has an integrated Hex-Editor coded by KetilO of the RadASM project which has been ported to PVDasm, a PE Editor,Process Manager.

It also supports for xRef, comments, CodePatcher,Call/Jxx Tracing & BackTracing

Import Anlyzer, String References for disassembly, masm source code generating wizard.

Q. What more features we will see implemented?

A well, it is upto the people who uses it.

They can ask for any kind of suggestions to be added

but since pvdasm supports Plugin SDK, they can code thier own plugins.

thus, if a plugin requires special Message to be handled by PVDasm,

they need to contact me and i will add this message.

it could be i will add 'Flirt' alike system to make disassembly more easier.

Q. Why when I press on CodePatcher at some address in Order to patch the bytes I get diff bytes?

A Yes i know, this problem depends on disasm align problem as explained above.

Q. Can PVDasm create source code from the disassembled executable?

A Yes it can! but only for win32 assembly executables, and the target for now is for Masm only!

Q. The source created is not complete (not compilable), where are all the stuff?

A Well, as you can see, we must 'program' PVDasm that it must gather all the needed infromation in order to create the most of the source code we can get.

we must specify a correct Function Entrypoints addresses, as well as aligning, the disassembly by configuring data in code segments if available.

once this has been done, we use the Masm wizard and build the source code,

With the right parameters.

* see tutorials in this help file.

A Word from the author:

=====

First of all thank you all for supporting PVDasm.

I want you all to know that PVDasm is still w.i.p and it is at best to combine other tools to gain maximum disassembly out of pvdasm.

as you can see, you can 'teach' pvdasm what to do (e.g: data segments), in order to give more effective disassembly rather to provide an automatic assumption.

It is better to work with several tools than depending on a single tool.

you can discuss about pvdasm at: <http://board.anticrack.de/viewforum.php?f=50>

and download it from: <http://pvdasm.reverse-engineering.net>

-Bengaly 2003-2010

Masm Code Generation Tutorial

Source Code Rebuilding Using PVDasm

Note: Creating Masm Source Code (*.asm) from Win32ASM Executables only!

Example: COMCTL.EXE \masm32\ICZTUTES\TUTE18\

See Compilble Result

load comctl.exe into pvdasm, and be sure the default options are set and disassemble the file.
we should see something like that:

```
=====
=====
...Snip...
; =====[ Program Entry Point ]=====
00401000  6A00          PUSH 00H ; lpModuleName
00401002  E8 E3020000   CALL KERNEL32!GetModuleHandleA
00401007  A3 4C304000   MOV DWORD PTR DS:[0040304CH], EAX
0040100C  6A0A          PUSH 0AH
0040100E  6A00          PUSH 00H
00401010  6A00          PUSH 00H
00401012  FF354C304000  PUSH DWORD PTR DS:[0040304CH]
00401018  E8 0B000000   CALL 00401028
0040101D  50            PUSH EAX ; uExitCode
0040101E  E8 C1020000   CALL KERNEL32!ExitProcess
00401023  E8 68020000   CALL COMCTL32!Ordinal: 17
; =====[ Subroutine ]=====
00401028  55            PUSH EBP
00401029  8BEC          MOV EBP, ESP
0040102B  83C4B0        ADD ESP, -50H
0040102E  C745D0 30000000 MOV DWORD PTR SS:[EBP-30H],00000030H
00401035  C745D4 03000000 MOV DWORD PTR SS:[EBP-2CH],00000003H
0040103C  C745D8 FD104000 MOV DWORD PTR SS:[EBP-28H],004010FDH
00401043  C745DC 00000000 MOV DWORD PTR SS:[EBP-24H],00000000H
0040104A  C745E0 00000000 MOV DWORD PTR SS:[EBP-20H],00000000H
00401051  FF7508        PUSH DWORD PTR SS:[EBP+08H]
00401054  8F45E4        POP DWORD PTR SS:[EBP-1CH]
00401057  C745F0 0C000000 MOV DWORD PTR SS:[EBP-10H],0000000CH
0040105E  C745F4 00000000 MOV DWORD PTR SS:[EBP-0CH],00000000H
00401065  C745F8 00304000 MOV DWORD PTR SS:[EBP-08H],00403000H ;ASCIIZ:
"CommonControlWinClass",0
0040106C  68 007F0000   PUSH 00007F00H ; lpIconName
00401071  6A00          PUSH 00H ; hInstance
00401073  E8 42020000   CALL USER32!LoadIconA

..Snip..
=====
=====
```

nothing fancy i know .. :)

let see what Functions EntryPoint the Simple FirstPass analyzer has Marked:

press Ctrl+Alt+E to show the FEP Manager.

we see 2 functions Marked, although if we browes the file, we can cleary see that there is another function/subroutine unmarked.

lets select the range of addresses to convert to FEP (Function EntryPoint), or from the FEP manager to enter the start/end address of the function, i'll use the marking approach,

so lets mark address:

004010FE 8BEC MOV EBP, ESP

and scroll till (while selecting lines):

00401286 C2 1000 RET 0010H

and press Alt+Insert .

output window will show us this message: **Address Range 004010FE : 00401286 Transformed to Function EntryPoint.**

you can see it added in the FEP Manager [Alt+Ctrl+E]

now, to make some visual changes, lets ReDisassemble: Alt+R
and there is it:

... Snip ...

```
; =====[ Subroutine ]=====
004010FE  8BEC      MOV EBP, ESP
00401100  837D0C01  CMP DWORD PTR SS:[EBP+0CH],01H
00401104  0F85 95000000  JNZ 0040119F
0040110A  6A00      PUSH 00H ; lpParam
0040110C  FF354C304000  PUSH DWORD PTR DS:[0040304CH] ; hInstance
... Snip ...
```

ok, there are only 3 functions.. no more, lets now build the Source.

PVDasm comes with a small and easy to use Source Builder Wizard for MASM Output.

Creating Source File:

=====

File->Produce->Masm Project...

1. Setting up Path(s)

=====

we need here to setup the output filename (*.asm)
and the MASM patch, (e.g: c:)
when done, clicking Next for next Window.

2. Setting up Intialized Variables

=====

Mostly here the intialized strings will be created,
no other globals will be defined by pvdasm.

in this dialog we are presented with the name of the string, and the
string preview it self.

i have restricted the var name to 25 chars, and includes only A-Z/a-z
characters as a filter.

1. to Redefine an var, simply dbl-click it's name and it will be loaded to the,
EditBox.

e.g: msctls_progress____

changing it to: msctls_progress
and click the Redefine button.

2. we can also Delete duplicated strings, select your string and hit Delete Button.

click Next

3. Setting up the Functions:

=====

here on the 'Select Range' combobox we are presented with the
Function ranges that we and the FirstPass analyzer has marked.
usually i think hand-marking is better than automatic feature.

to define the functions, we must move from the top->down as we see
from the dropdown comboBox, creating in that order will give us the same

order on the output asm!!.

[The EntryPoint Subroutine]

selecting range:00401000 :: 0040101E

Corresponding Assembly Function Block will be shown alongside the remarks.

if this function is your entry point, Delete the 2 auto defined proto/proc text and create an empty skeleton.

or press the Empty Button to automatically delete the proc info.

messgaebox will tell us the function has an empty skeleton .. OK.

[The Subroutines]

now lets select the next Range: 00401028 :: 004010FA

the code will be shown to us.

notice the the main pvdasm is still selectable, so we can scroll and still use the disassembler in the background.

since this range is not an entrypoint startup code, we need or not, to create parameters for the function.

1.lets goto address 00401028 in the pvdasm (using goto option, or just scroll), selecting the address will tell us that the address has an XReference to it. right-click -> XReferences -> dbl-click the adress and we jump back to the caller.

we will see this:

```
=====
0040100C  6A0A      PUSH 0AH
0040100E  6A00      PUSH 00H
00401010  6A00      PUSH 00H
00401012  FF354C304000  PUSH DWORD PTR DS:[0040304CH]
00401018  E8 0B000000  CALL 00401028
=====
```

this function requires 4 parameters to be sent on the stack.

so, select the params you want to be declare from the drop down combobox.

usually i use DWORD, but its ur call to change what ever you want, or use a custom param and add.

2. we can use the Auto Search feature here to search for us the number of parameters, that he function needs, once clicked and found, it will automatically define the subroutine for us. default param type is DWORD.

all we need to do is to press Create button to define the Function.

note: no remove button for now, manual edit your editbox and click Create to make changes.

selecting the last Function range from the list: 004010FD :: 00401286

trying to find an xref for 004010FD won't work, since there is no call to this address using any branch instruction located in the code section, rather, this function is being reffered by a memory access.

see this code snippet:

```
=====
... Snip ...
00401028  55          PUSH EBP
00401029  8BEC        MOV EBP, ESP
0040102B  83C4B0      ADD ESP, -50H
0040102E  C745D0 30000000  MOV DWORD PTR SS:[EBP-30H],00000030H
00401035  C745D4 03000000  MOV DWORD PTR SS:[EBP-2CH],00000003H
0040103C  C745D8 FD104000  MOV DWORD PTR SS:[EBP-28H],004010FDH; <== LOOK HERE
00401043  C745DC 00000000  MOV DWORD PTR SS:[EBP-24H],00000000H
0040104A  C745E0 00000000  MOV DWORD PTR SS:[EBP-20H],00000000H
... Snip ...
=====
```

well, if we check several Win32ASM sources, we might see that this code reffers to

the WNDCLASSEXA struct using the RegisterClass api, so this function is the actual Event Handler Function, you may know it as the 'WndProc' function where we handle the 'WM_INITDIALOG / WM_PAINT' messages...etc ok, we got it..

this function also require 4 params (hwnd/msg/wparam/lparam).

same thing as above, i added 4 DWORDS to the skeleton of the function and defined it.

Click Next to continue.

The MASM Compiler Options:

=====

this is a nice little dialog where we can change the most used options for the compiler. (MASM32)
such as the machine type, entrypoint name, calling convention type, adding directives..etc

click next and see final result of the assembly file.
you can always go back and change your settings.

now since PVDasm created for us most of the stuff, we need to make hand modifications.
untill i will add some stuff to ease this process.
what do we need to change:

1. add the defined strings to our code insted of push xxxxxxxx change into push offset_myString
2. when need, add 'offset proc_xxxxxxxx' into some code parts [example: WndProc caller].

lets see created code by PVDasm:

~~~~~

```
; #####
; # This File is generated by Proview Disassembler (PVDasm)  #
; # Copyright (c) 2004 by Bengaly, <pvdasm.anticrack.de>    #
; #####
.386      ; create 32 bit code
.model flat, stdcall ; 32 bit memory model
option casemap:none  ; case sensitive

include C:.inc
include C:.inc
include C:.inc
include C:.inc
includelib C:.lib
includelib C:.lib
includelib C:.lib
; #####

proc_00401028 proto :DWORD,:DWORD,:DWORD,:DWORD
proc_004010FD proto :DWORD,:DWORD,:DWORD,:DWORD

.data
CommonControlWinClass db "CommonControlWinClass",0
Common_Control_Demo_ db "Common Control Demo",0
msctls_progress db "msctls_progress32",0
Finished db "Finished!",0

.data?

.code
start:
PUSH 00H ; lpModuleName
CALL GetModuleHandleA
MOV DWORD PTR DS:[0040304CH], EAX
PUSH 0AH
PUSH 00H
PUSH 00H
PUSH DWORD PTR DS:[0040304CH]
CALL proc_00401028
PUSH EAX ; uExitCode
CALL ExitProcess
```

```

proc_00401028 proc Var1:DWORD,Var2:DWORD,Var3:DWORD,Var4:DWORD
ADD ESP, -50H
MOV DWORD PTR SS:[EBP-30H],00000030H
MOV DWORD PTR SS:[EBP-2CH],00000003H
MOV DWORD PTR SS:[EBP-28H],004010FDH
MOV DWORD PTR SS:[EBP-24H],00000000H
MOV DWORD PTR SS:[EBP-20H],00000000H
PUSH DWORD PTR SS:[EBP+08H]
POP DWORD PTR SS:[EBP-1CH]
MOV DWORD PTR SS:[EBP-10H],0000000CH
MOV DWORD PTR SS:[EBP-0CH],00000000H
MOV DWORD PTR SS:[EBP-08H],00403000H ;ASCIIZ: "CommonControlWinClass",0
PUSH 00007F00H ; lpIconName
PUSH 00H ; hInstance
CALL LoadIconA
MOV DWORD PTR SS:[EBP-18H],EAX
MOV DWORD PTR SS:[EBP-04H],EAX
PUSH 00007F00H ; lpCursorName
PUSH 00H ; hInstance
CALL LoadCursorA
MOV DWORD PTR SS:[EBP-14H],EAX
LEA EAX,DWORD PTR SS:[EBP-30H]
PUSH EAX ; pcWndClassEx
CALL RegisterClassExA
PUSH 00H ; lpParam
PUSH DWORD PTR SS:[EBP+08H] ; hInstance
PUSH 00H ; hMenu
PUSH 00H ; hWndParent
PUSH 80000000H ; nHeight
PUSH 80000000H ; nWidth
PUSH 80000000H ; y
PUSH 80000000H ; x
PUSH 10CB0000H ; dwStyle
PUSH 00403016H ; lpWindowName [ ;ASCIIZ: "Common Control Demo",0 ]
PUSH 00403000H ; lpClassName [ ;ASCIIZ: "CommonControlWinClass",0 ]
PUSH 00000200H ; dwExStyle
CALL CreateWindowExA
MOV DWORD PTR SS:[EBP-50H],EAX
ref_004010CF:
PUSH 00H
PUSH 00H ; wParamFilterMax
PUSH 00H ; wParamFilterMin
LEA EAX,DWORD PTR SS:[EBP-4CH] ; hwnd
PUSH EAX ; lParam
CALL GetMessageA
OR EAX, EAX
JZ ref_004010F6
LEA EAX,DWORD PTR SS:[EBP-4CH]
PUSH EAX ; lParam
CALL TranslateMessage
LEA EAX,DWORD PTR SS:[EBP-4CH]
PUSH EAX ; lParam
CALL DispatchMessageA
JMP ref_004010CF
ref_004010F6:
MOV EAX,DWORD PTR SS:[EBP-44H]
RET ; proc_00401028 endp
proc_00401028 endp

```

```

proc_004010FD proc Var1:DWORD,Var2:DWORD,Var3:DWORD,Var4:DWORD
CMP DWORD PTR SS:[EBP+0CH],01H
JNZ ref_0040119F
PUSH 00H ; lpParam

```



```

PUSH DWORD PTR DS:[0040304CH] ; hInstance
PUSH 01H ; hMenu
PUSH DWORD PTR SS:[EBP+08H] ; hWndParent
PUSH 14H ; nHeight
PUSH 0000012CH ; nWidth
PUSH 000000C8H ; y
PUSH 64H ; x
PUSH 50000000H ; dwStyle
PUSH 00H ; lpWindowName
PUSH 0040302AH ; lpClassName [ ;ASCIIZ: "msctls_progress32",0 ]
PUSH 00H ; dwExStyle
CALL CreateWindowExA
MOV DWORD PTR DS:[00403050H], EAX
MOV EAX, 000003E8H
MOV DWORD PTR DS:[00403058H], EAX
SHL EAX, 10H
PUSH EAX ; lParam
PUSH 00H ; wParam
PUSH 00000401H ; wMsg
PUSH DWORD PTR DS:[00403050H] ; hwnd
CALL SendMessageA
PUSH 00H ; lParam
PUSH 0AH ; wParam
PUSH 00000404H ; wMsg
PUSH DWORD PTR DS:[00403050H] ; hwnd
CALL SendMessageA
PUSH 02H
PUSH DWORD PTR SS:[EBP+08H]
PUSH 00H
PUSH 50000000H
CALL Ordinal: 6
MOV DWORD PTR DS:[00403054H], EAX
PUSH 00H ; lpTimerFunc
PUSH 64H ; uElapse
PUSH 03H ; nIDEvent
PUSH DWORD PTR SS:[EBP+08H] ; hWnd
CALL SetTimer
MOV DWORD PTR DS:[00403046H], EAX
JMP ref_00401283
ref_0040119F:
CMP DWORD PTR SS:[EBP+0CH], 02H
JNZ ref_004011CC
PUSH 00H ; nExitCode
CALL PostQuitMessage
CMP DWORD PTR DS:[00403046H], 00H
JZ ref_00401283
PUSH DWORD PTR DS:[00403046H] ; nIDEvent
PUSH DWORD PTR SS:[EBP+08H] ; hwnd
CALL KillTimer
JMP ref_00401283
ref_004011CC:
CMP DWORD PTR SS:[EBP+0CH], 00000113H
JNZ ref_0040126E
PUSH 00H ; lParam
PUSH 00H ; wParam
PUSH 00000405H ; wMsg
PUSH DWORD PTR DS:[00403050H] ; hwnd
CALL SendMessageA
SUB DWORD PTR DS:[00403058H], 0AH
CMP DWORD PTR DS:[00403058H], 00H
JNZ ref_00401283
PUSH DWORD PTR DS:[00403046H] ; nIDEvent
PUSH DWORD PTR SS:[EBP+08H] ; hwnd
CALL KillTimer

```

```

MOV DWORD PTR DS:[00403046H],00000000H
PUSH 0040303CH ; IParam [ ;ASCIIZ: "Finished!",0 ]
PUSH 00H ; wParam
PUSH 00000401H ; wParam
PUSH DWORD PTR DS:[00403054H] ; hwnd
CALL SendMessageA
PUSH 40H ; wType
PUSH 00403016H ; lpCaption [ ;ASCIIZ: "Common Control Demo",0 ]
PUSH 0040303CH ; lpText [ ;ASCIIZ: "Finished!",0 ]
PUSH DWORD PTR SS:[EBP+08H] ; hwnd
CALL MessageBoxA
PUSH 00H ; IParam
PUSH 00H ; wParam
PUSH 00000401H ; wParam
PUSH DWORD PTR DS:[00403054H] ; hwnd
CALL SendMessageA
PUSH 00H ; IParam
PUSH 00H ; wParam
PUSH 00000402H ; wParam
PUSH DWORD PTR DS:[00403050H] ; hwnd
CALL SendMessageA
JMP ref_00401283
ref_0040126E:
PUSH DWORD PTR SS:[EBP+14H] ; IParam
PUSH DWORD PTR SS:[EBP+10H] ; wParam
PUSH DWORD PTR SS:[EBP+0CH] ; wParam
PUSH DWORD PTR SS:[EBP+08H] ; hwnd
CALL DefWindowProcA
RET
ref_00401283:
XOR EAX, EAX
RET ; proc_004010FD endp
proc_004010FD endp

```

end start

~~~~~

Hand Modifications - full compilble Source:

```

=====
; #####
; # This File is generated by Proview Disassembler (PVDasm) #
; # Copyright (c) 2004 by Bengaly, <pvdasm.anticrack.de> #
; #####
.386 ; create 32 bit code
.model flat, stdcall ; 32 bit memory model
option casemap:none ; case sensitive

include C:\masm32\include\windows.inc
include C:\masm32\include\kernel32.inc
include C:\masm32\include\user32.inc
include C:\masm32\include\comctl32.inc

includelib C:\masm32\lib\user32.lib
includelib C:\masm32\lib\kernel32.lib
includelib C:\masm32\lib\comctl32.lib
; #####

proc_004010FD proto :DWORD,:DWORD,:DWORD,:DWORD
proc_00401028 proto :DWORD,:DWORD,:DWORD,:DWORD

.data
CommonControlWinClass db "CommonControlWinClass",0

```

Common_Control_Demo db "Common Control Demo",0
msctls_progress db "msctls_progress32",0
Finished db "Finished!",0

.data?

hInstance dd ?
g_hwnd dd ?
g_param dd ?
g_param2 dd ?
g_param3 dd ?
g_param4 dd ?

.code

start:

PUSH 00H ; lpModuleName
CALL GetModuleHandleA
MOV hInstance , EAX ; changed
PUSH 0AH
PUSH 00H
PUSH 00H
PUSH hInstance ; changed
CALL proc_00401028 ; changed
PUSH EAX ; uExitCode
CALL ExitProcess

proc_00401028 proc Var1:DWORD,Var2:DWORD,Var3:DWORD,Var4:DWORD

PUSH EBP
MOV EBP, ESP
ADD ESP, -50H
MOV DWORD PTR SS:[EBP-30H],00000030H
MOV DWORD PTR SS:[EBP-2CH],00000003H
MOV DWORD PTR SS:[EBP-28H],offset proc_004010FD ; changed
MOV DWORD PTR SS:[EBP-24H],00000000H
MOV DWORD PTR SS:[EBP-20H],00000000H
PUSH DWORD PTR SS:[EBP+08H]
POP DWORD PTR SS:[EBP-1CH]
MOV DWORD PTR SS:[EBP-10H],0000000CH
MOV DWORD PTR SS:[EBP-0CH],00000000H
MOV DWORD PTR SS:[EBP-08H],offset CommonControlWinClass ;ASCIIZ: "CommonControlWinClass",0 -
changed
PUSH 00007F00H ; lpIconName
PUSH 00H ; hInstance
CALL LoadIconA
MOV DWORD PTR SS:[EBP-18H],EAX
MOV DWORD PTR SS:[EBP-04H],EAX
PUSH 00007F00H ; lpCursorName
PUSH 00H ; hInstance
CALL LoadCursorA
MOV DWORD PTR SS:[EBP-14H],EAX
LEA EAX,DWORD PTR SS:[EBP-30H]
PUSH EAX ; pcWndClassEx
CALL RegisterClassExA
PUSH 00H ; lpParam
PUSH DWORD PTR SS:[EBP+08H] ; hInstance
PUSH 00H ; hMenu
PUSH 00H ; hWndParent
PUSH 80000000H ; nHeight
PUSH 80000000H ; nWidth
PUSH 80000000H ; y
PUSH 80000000H ; x
PUSH 10CB0000H ; dwStyle
PUSH offset Common_Control_Demo ; lpWindowName [;ASCIIZ: "Common Control Demo",0] - changed

```

PUSH offset CommonControlWinClass ; lpClassName [ ;ASCIIZ: "CommonControlWinClass",0 ] - changed
PUSH 00000200H ; dwExStyle
CALL CreateWindowExA
MOV DWORD PTR SS:[EBP-50H],EAX
ref_004010CF:
PUSH 00H
PUSH 00H ; wParamFilterMax
PUSH 00H ; wParamFilterMin
LEA EAX,DWORD PTR SS:[EBP-4CH] ; hwnd
PUSH EAX ; lParam
CALL GetMessageA
OR EAX, EAX
JZ ref_004010F6 ; changed
LEA EAX,DWORD PTR SS:[EBP-4CH]
PUSH EAX ; lParam
CALL TranslateMessage
LEA EAX,DWORD PTR SS:[EBP-4CH]
PUSH EAX ; lParam
CALL DispatchMessageA
JMP ref_004010CF ; changed
ref_004010F6:
MOV EAX,DWORD PTR SS:[EBP-44H]
;LEAVE ;Releases the local variables - ; changed
RET ; changed
proc_00401028 endp

```

```

; this is our WndProc
proc_004010FD proc Var1:DWORD,Var2:DWORD,Var3:DWORD,Var4:DWORD
;PUSH EBP ; masm creates stack frame
;MOV EBP, ESP ; for us, so no need to make twice
CMP DWORD PTR SS:[EBP+0CH],01H
JNZ ref_0040119F ; changed
PUSH 00H ; lParam
PUSH
hInstance ; hInstance - changed
PUSH 01H ; hMenu
PUSH DWORD PTR SS:[EBP+08H] ; hWndParent
PUSH 14H ; nHeight
PUSH 0000012CH ; nWidth
PUSH 000000C8H ; y
PUSH 64H ; x
PUSH 50000000H ; dwStyle
PUSH 00H ; lpWindowName
PUSH offset msctls_progress ; lpClassName [ ;ASCIIZ: "msctls_progress32",0 ] - changed
PUSH 00H ; dwExStyle
CALL CreateWindowExA
MOV g_hwnd, EAX ; changed
MOV EAX, 000003E8H
MOV
g_param, EAX ; changed
SHL EAX, 10H
PUSH EAX ; lParam
PUSH 00H ; wParam
PUSH 00000401H ; wParam
PUSH g_hwnd ; hwnd - changed
CALL SendMessageA
PUSH 00H ; lParam
PUSH 0AH ; wParam
PUSH 00000404H ; wParam
PUSH g_hwnd ; hwnd - changed
CALL SendMessageA
PUSH 02H
PUSH DWORD PTR SS:[EBP+08H]
PUSH 00H

```

```

PUSH 50000000H
CALL CreateStatusWindow ; Added
MOV g_param4, EAX
PUSH 00H ; lpTimerFunc
PUSH 64H ; uElapse
PUSH 03H ; nIDEvent
PUSH DWORD PTR SS:[EBP+08H] ; hWnd
CALL SetTimer
MOV g_param3, EAX
JMP ref_00401283 ; changed
ref_0040119F:
CMP DWORD PTR SS:[EBP+0CH],02H
JNZ ref_004011CC
PUSH 00H ; nExitCode
CALL PostQuitMessage
CMP g_param3,00H
JZ ref_00401283 ; changed
PUSH g_param3 ; nIDEvent
PUSH DWORD PTR SS:[EBP+08H] ; hWnd
CALL KillTimer
JMP ref_00401283 ; changed
ref_004011CC:
CMP DWORD PTR SS:[EBP+0CH],00000113H
JNZ ref_0040126E ; changed
PUSH 00H ; lParam
PUSH 00H ; wParam
PUSH 00000405H ; wParam
PUSH g_hwnd ; hWnd
CALL SendMessageA
SUB g_param,0AH
CMP g_param,00H
JNZ ref_00401283 ; changed
PUSH g_param3 ; nIDEvent
PUSH DWORD PTR SS:[EBP+08H] ; hWnd
CALL KillTimer
MOV g_param3,00000000H
PUSH offset Finished ; lParam [ ;ASCIIZ: "Finished!",0 ]
PUSH 00H ; wParam
PUSH 00000401H ; wParam
PUSH g_param4 ; hWnd
CALL SendMessageA
PUSH 40H ; wType
PUSH offset Common_Control_Demo ; lpCaption [ ;ASCIIZ: "Common Control Demo",0 ]
PUSH offset Finished ; lpText [ ;ASCIIZ: "Finished!",0 ]
PUSH DWORD PTR SS:[EBP+08H] ; hWnd
CALL MessageBoxA
PUSH 00H ; lParam
PUSH 00H ; wParam
PUSH 00000401H ; wParam
PUSH g_param4 ; hWnd
CALL SendMessageA
PUSH 00H ; lParam
PUSH 00H ; wParam
PUSH 00000402H ; wParam
PUSH g_hwnd ; hWnd
CALL SendMessageA
JMP ref_00401283 ; changed
ref_0040126E:
PUSH DWORD PTR SS:[EBP+14H] ; lParam
PUSH DWORD PTR SS:[EBP+10H] ; wParam
PUSH DWORD PTR SS:[EBP+0CH] ; wParam
PUSH DWORD PTR SS:[EBP+08H] ; hWnd
CALL DefWindowProcA
;LEAVE ;Releases the local variables

```

```
RET ; changed
ref_00401283:
XOR EAX, EAX
;LEAVE ;Releases the local variables
RET ; changed
proc_004010FD endp
```

```
end start
```

PI_GETENTRYPOINT

PI_GETENTRYPOINT

A Plugin sends PI_GETENTRYPOINT message to retrieve the EntryPoint Address of the current disassembled

```
//  
// DEFINED AS:  
//  
#define PI_GETENTRYPOINT WM_USER+PI_BASE_MSG+4 // 23.8.04  
  
PI_GETENTRYPOINT  
wParam=(WPARAM) (DWORD*)lpEntryPoint; // Pointer to member to fill with EntryPoint  
lParam=NULL
```

Parameters:

lpEntryPoint

value of wParam. Points to DWORD variable to be filled.

Return Values:

(None)

Address of EntryPoint filled in the member lpEntryPoint.

Source Code:

```
//  
// Source Code Example ( as a part of PVDasm SDK Plugin code )  
//  
  
DWORD EntryPoint;  
  
// Get Entry Point  
SendMessage(*PlgData.Parent_hWnd,PI_GETENTRYPOINT,(WPARAM)&EntryPoint,NULL);
```

PI_PRINTDBGTEXT

PI_PRINTDBGTEXT

A Plugin sends PI_PRINTDBGTEXT message to output (send) a string onto the Proview Output Debug Window.

```
//  
// DEFINED AS:  
//  
#define PI_PRINTDBGTEXT WM_USER+PI_BASE_MSG+5 // 23.8.04  
  
PI_PRINTDBGTEXT  
wParam=(WPARAM) (char*)lpString; // String pointer to be output in dbg window  
lParam=NULL
```

Parameters:

lpString

value of wParam. Points to String text to be sent to the output debug window.

Return Values:

(None)

Source Code:

```
//  
// Source Code Example ( as a part of PVDasm SDK Plugin code )  
//  
// print console message ni pvdasm  
SendMessage(*PlgData.Parent_hWnd,PI_PRINTDBGTEXT,(WPARAM)"Example String",NULL);
```


PI_GETBYTEFROMADDRESS

PI_GETBYTEFROMADDRESS

A Plugin sends PI_GETENTRYPOINT message in order to retrieve a byte from a specific Address.

```
//  
// DEFINED AS:  
//  
#define PI_GETBYTEFROMADDRESS WM_USER+PI_BASE_MSG+6 // 23.8.04
```

PI_GETBYTEFROMADDRESS

```
wParam=(WPARAM) (DWORD)dwAddress; // Value of Address to get byte from  
lParam=(LPARAM) (BYTE*)lpByte; // Points to a member to fill with byte
```

Parameters:

dwAddress

value of wParam. holds an address of the byte to retrieve.

lpByte

value of lParam. member to be filled with a byte. (as specified by address member - dwAddress)

Return Values:

(None)

Source Code:

```
//  
// Source Code Example ( as a part of PVDasm SDK Plugin code )  
//  
BYTE data;  
// get byte from address  
SendMessage(*PlgData.Parent_hWnd,PI_GETBYTEFROMADDRESS,(WPARAM)Address,(LPARAM)&data);
```

PI_RVATOFFSET

PI_RVATOFFSET

A Plugin sends PI_RVATOFFSET message in order to retrieve the acculated Offset from a specific Address (RVA).

```
//  
// DEFINED AS:  
//  
#define PI_RVATOFFSET WM_USER+PI_BASE_MSG+7 // 26.8.04  
  
PI_RVATOFFSET  
wParam=(WPARAM) (DWORD)dwRVA; // Value of Address to get the offset of  
lParam=(LPARAM) (DWORD*)lpOffset; // Points to a member to fill with calculated offset
```

Parameters:

dwRVA

value of wParam. holds an address to calculated offset from.

lpOffset

value of lParam. member to be filled with the calculated offset.

Return Values:

(None)

Source Code:

```
//  
// Source Code Example ( as a part of PVDasm SDK Plugin code )  
//  
DWORD Offset;  
DWORD EntryPoint=00401000;  
// get byte from address  
SendMessage(*PlgData.Parent_hWnd,PI_RVATOFFSET,(WPARAM)EntryPoint,(LPARAM)&Offset);
```

PI_GETNUMOFSTRINGREF

PI_GETNUMOFSTRINGREF

A Plugin sends PI_GETNUMOFSTRINGREF message in order to retrieve the number of analyzed string references.

```
//  
// DEFINED AS:  
//  
#define PI_GETNUMOFSTRINGREF WM_USER+PI_BASE_MSG+8 // 26.8.04  
  
PI_GETNUMOFSTRINGREF  
wParam=(WPARAM) (DWORD*)lpNumberOfStrRef; // Pointer to fill with number of string refs  
lParam=NULL
```

Parameters:

lpNumberOfStrRef

value of wParam. Pointer to fill with number of string refs

Return Values:

(None)

Source Code:

```
//  
// Source Code Example ( as a part of PVDasm SDK Plugin code )  
//  
DWORD NumOfStrRef=0;  
SendMessage(*PlgData.Parent_hWnd,PI_GETNUMOFSTRINGREF,(WPARAM)&NumOfStrRef,(LPARAM)NULL);
```

PI_GETSTRINGREFERENCE

PI_GETSTRINGREFERENCE

A Plugin sends PI_GETSTRINGREFERENCE message in order to retrieve a analyzed String reference.

```
//  
// DEFINED AS:  
//  
#define PI_GETSTRINGREFERENCE WM_USER+PI_BASE_MSG+9 // 26.8.04  
  
PI_GETSTRINGREFERENCE  
wParam=(WPARAM) (DWORD)dwIndex; // Index of String Reference  
lParam=(LPARAM) (CHAR*)lpStrRef; // Points to string member to fill with the reference
```

Parameters:

dwIndex

value of wParam. holds the index for the string reference.

lpStrRef

value of lParam. member to be filled with the returned string reference.

Return Values:

(None)

Source Code:

```
//  
// Source Code Example ( as a part of PVDasm SDK Plugin code )  
//  
DWORD NumOfStrRef=0;  
char str[128];  
SendMessage(*PlgData.Parent_hWnd,PI_GETNUMOFSTRINGREF,(WPARAM)&NumOfStrRef,(LPARAM)NULL);  
  
for(int i=0;i<NumOfStrRef;i++){  
    SendMessage(*PlgData.Parent_hWnd,PI_GETSTRINGREFERENCE,(WPARAM)i,(LPARAM)str);  
    SendMessage(*PlgData.Parent_hWnd,PI_PRINTDBGTEXT,(WPARAM)str,NULL);  
}
```

PI_SETCOMMENT

PI_SETCOMMENT

A Plugin sends PI_SETCOMMENT message in order to **Set** a new comment for a specific line in PVDasm.

```
//  
// DEFINED AS:  
//  
#define PI_SETCOMMENT WM_USER+PI_BASE_MSG+10 // NEW 16.9.04  
  
PI_SETCOMMENT  
wParam=(WPARAM) (DWORD)dwIndex; // Index of Item to set comment to  
lParam=(LPARAM) (CHAR*)lpComment; // Points to string comment to fill to the index
```

Parameters:

dwIndex

value of wParam. holds the index to the item to set comment into.

lpComment

value of lParam. NULL Terminated String member to fill with the index supplied by dwIndex.

Return Values:

(None)

Source Code:

```
//  
// Source Code Example ( as a part of PVDasm SDK Plugin code )  
//  
SendMessage(*PlgData.Parent_hWnd,PI_SETCOMMENT,(WPARAM)2,(LPARAM)"Test Comment...");
```

PI_ADDCOMMENT

PI_ADDCOMMENT

A Plugin sends PI_ADDCOMMENT message in order to **Add** a comment for a specific line in PVDasm.

```
//  
// DEFINED AS:  
//  
#define PI_ADDCOMMENT WM_USER+PI_BASE_MSG+11 // NEW 16.9.04  
  
PI_ADDCOMMENT  
wParam=(WPARAM) (DWORD)dwIndex; // Index of Item to set comment to  
lParam=(LPARAM) (CHAR*)lpComment; // Points to string comment to fill to the index
```

Parameters:

dwIndex

value of wParam. holds the index to the item to set comment into.

lpComment

value of lParam. NULL Terminated String member to fill with the index supplied by dwIndex.

Return Values:

(None)

Source Code:

```
//  
// Source Code Example ( as a part of PVDasm SDK Plugin code )  
//  
SendMessage(*PlgData.Parent_hWnd,PI_ADDCOMMENT,(WPARAM)2,(LPARAM)"Test Comment...");
```

MAP File

Creating Detailed Disassembly using a MAP file

PVDasm can use a specific MAP file, not the generic one that IDA Exports to us, this is because PVDasm is designed to keep functions/data separated internally. the default exported map does not bring pvdasm enough info to make it separate those data. there for pvdasm comes with a small script to make a new specific MAP file.

IDA:

1. Load target in IDA
2. let IDA Analyze the file completely.
3. Run the Script: "/Map/pvmap.idc" (may be slow for big files! with allot of info)
4. Upon finishing, a new fille will be saved in c:/pvmap.map

PVDasm:

1. Load target in PVDasm
2. upon "Decode New File" dialog screen, Click "Import MAP File" Button.
3. Select c:/pvmap.map - IDA will load the information into it.
4. Disable/Uncheck "First Pass Analyzer" - IDA already did that for us!
5. Press ok to disassemble and view the output.

Note: upon successfull MAP importing, we can see all the data imported via, Function Editor and Data Editor (Alt+Ctrl+D / Alt+Ctrl+E)