determina

esreveR gnireenignE tfosorciM seiraniB

Alexander Sotirov
asotirov@determina.com

CanSecWest / core06

determina

Reverse Engineering Microsoft Binaries

Alexander Sotirov
asotirov@determina.com

CanSecWest / core06

Overview

determina™

In the next one hour, we will cover:

- Setting up a scalable reverse engineering environment
  - getting binaries and symbols
  - reverse engineering tools
- Common features of Microsoft binaries
  - compiler optimizations
  - hotpatching
  - exception handling
- Improving your tools
  - IDA autoanalysis problems
  - loading debugging symbols
  - improving the analysis with IDA plugins

Why Reverse Engineering?

determina™

Reverse engineering plays an important role in security research. It is most often used for the following:

- Binary patch analysis
- Binary code auditing
- Exploit development
- Interoperability

Recent developments:

- In the last five years reverse engineering tools have matured
- New tools designed specifically for the security community have been developed (SABRE BinDiff)
- More complicated exploitation techniques (heap overflows, uninitialized variables) require the use of reverse engineering

Why Microsoft Binaries?

determina™

Most reverse engineering talks focus on reversing malware, but Microsoft binaries present a very different challenge:

- Bigger than most malware
- No code obfuscation (with the exception of licensing, DRM and PatchGuard code)
- Debugging symbols are usually available
- Compiled with the Microsoft Visual C++ compiler
- Most of the code is written in object oriented C++
- Heavy use of COM and RPC

Why Me?

determina

The security research team at Determina reverse engineers most Microsoft patches, especially when vulnerability details are not publicly available. I have been reversing patches every month for the last 9 months, including a lot of patches for older vulnerabilities.

The main challenge we've encountered is how to deal with the large volume of the patches and produce results quickly with limited resources.

determina™

Part I

Scalable Reverse Engineering

Patch Statistics

determina

Microsoft has released almost 400 security bulletins since 1998, often addressing multiple vulnerabilities with a single bulletin. The record for most CVE numbers is 14, in MS04-011.

Even bulletins with one CVE number often contain fixes for more than one bug. MS05-053 describes a single WMF vulnerability, but the patch fixes more than 30 different integer overflows. In addition to that, Microsoft often silently fixes undisclosed vulnerabilities and adds additional security enhancements not directly related to a specific vulnerability.

Microsoft's advance notification gives out the number of bulletins they are going to release, but this number in no way corresponds to the actual number of vulnerabilities fixed each month, or their complexity.

Scalable Reverse Engineering

determina™

Reverse engineering patches at Determina turned out to be very different than my previous reversing experience. In the past I could easily spend more than a month reversing a single interesting vulnerability to write an exploit for it. Now we have to analyze a lot more vulnerabilities in a shorter amount of time, while still producing accurate results. This is an interesting challenge.

We have developed internal processes and tools with the following goals in mind:

- Automate as much as possible
- Get accurate results with minimal human intervention
- Scale the reversing process to deal with the volume of Microsoft patches

Getting the Binaries

determina

Most Microsoft software, including older versions and service packs, is available for download from MSDN. We download these manually.

To download security updates automatically, we use the information in mssecure.xml, which is automatically downloaded by MBSA 1.2.1. The XML file contains a list of security bulletins for Windows, Office, Exchange, SQL Server and other products. It also provides direct download links to the patch files.

Unfortunately MBSA 1.2.1 was retired at the end of March 2006. The XML schema used by MBSA 2.0 is different, and our scripts don't support it yet.

Once you have all old security updates, downloading the new ones every month can be done manually.

Extracting the Binaries

determina

- CAB and most EXE files can be unpacked with `cabextract`
- MSI and MSP files are difficult to unpack. Usually they contain CAB archives that can be extracted, but the files in them have mangled names. Still working on a solution.
- An administrative installation is our temporary solution for dealing with Microsoft Office.
- Some IIS files are renamed during the installation. For example smtpsvc.dll is distributed as smtp_smtpsvc.dll in IMS.CAB on the Windows 2000 installation CD.
- Recent patches use intra-package delta compression (US patent application 20050022175). Unpacking them with cabextract gives you files with names like `_sfx_0000._p`. To unpack these patches, you have to run them with the /x command line option.

Binary Database

determina

We have an internal database of binaries indexed by the name and SHA1 hash of the file. We store the following file metadata:

- name — *ntdll.dll*
- size — *654336 bytes*
- modification date — *May 01, 2003, 3:56:12 PM*
- SHA1 hash — *9c3102ea1d30c8533dbf5d9da2a47…*
- DBG and PDB path — *Sym/ntdll.pdb/3E7B64D65/ntdll.pdb*
- source of the file
  - product — *Windows XP*
  - version — *SP1*
  - security update — *MS03-007*
  - build — *qfe*
  - comment

Binary Database

determina™

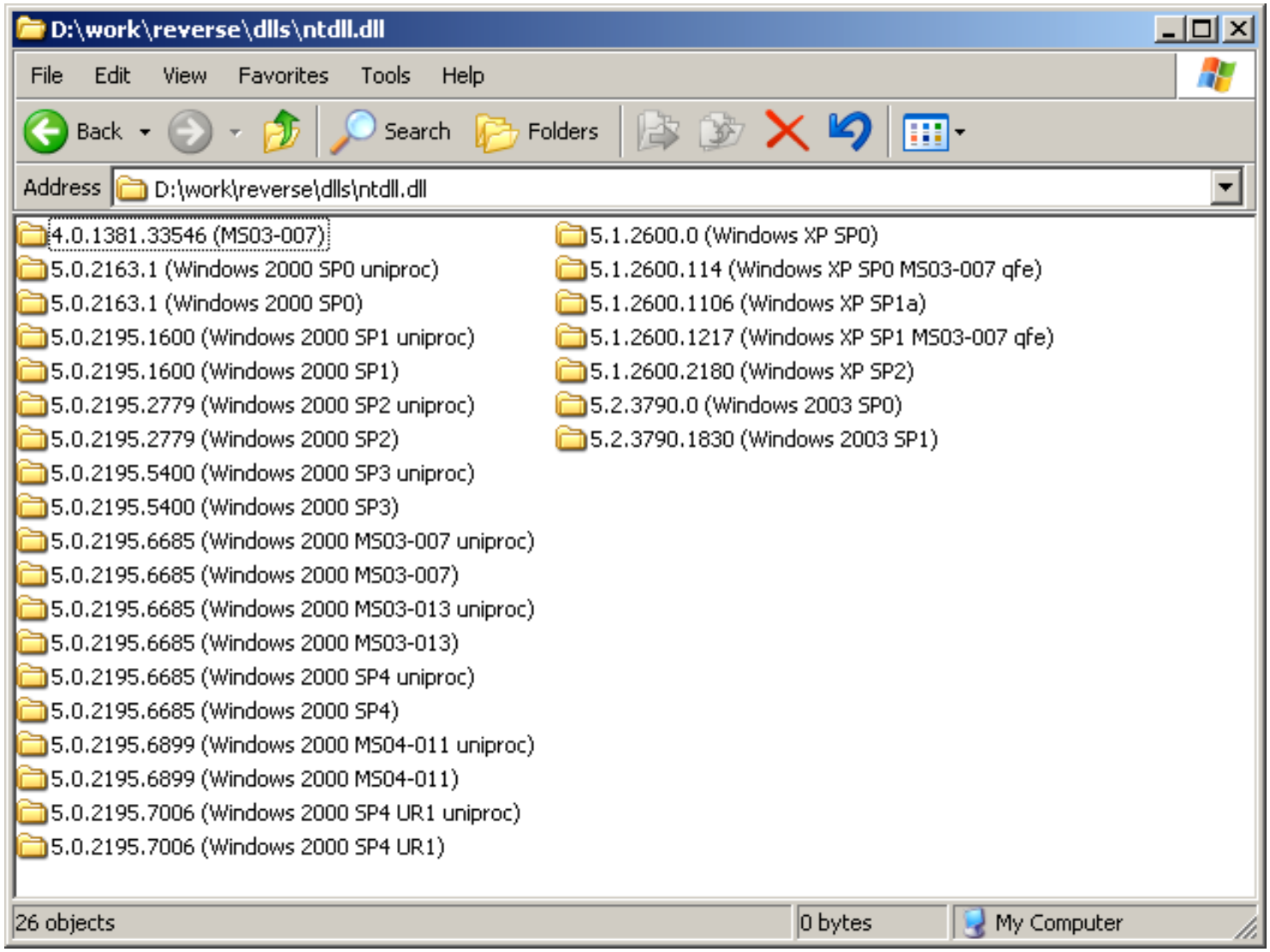
Current size of our database, including all service packs and security updates for Windows 2000, XP and 2003:

- 30GB of files
- 7GB of symbols
- 7500 different file names
- 28800 files total

and growing…

Binary Database

determina



Getting Symbols

determina™

Microsoft provides symbols for most Windows binaries. They can be downloaded from their public symbol server by including it in your symbol path. See the Debugging Tools for Windows help for more information.

Use symchk.exe to download symbols for a binary and store them in a local symbol store.

We have scripts that automatically run symchk.exe for all new files that are added to the binary database.

Most Windows binaries have symbols, with the exception of some older Windows patches. In this case BinDiff can be used to compare the binaries and port the function names from another version that has symbols. Unfortunately symbols are not available for Office and most versions of Exchange.

Any Vendors Reading This?

determina

This is my advice to vendors who want to help 3-rd party security researchers:

- provide researchers with an accurate list of all security updates, hotfixes and other patches (preferably in XML format)
- make older releases of your software available
- don't combine security patches with other updates
- have a consistent naming and versioning system
- do not update software without updating its version or build number
- provide debugging symbols for all binaries

Microsoft does almost all of the above.

Reverse Engineering Tools

determina

- IDA Pro
  - its plugin API is turning IDA into a reverse engineering platform that other tools depend on
- BinDiff
  - invaluable for binary patch analysis
- WinDbg
  - good support for debugging symbols, command line interface, frequent updates
- SoftICE
  - great for debugging code between userspace and the kernel
- VMWare
  - Workstation 5 supports multiple snapshots
  - GSX and Server provide a scripting API
  - Workstation 5.5 can be controlled with a command line tool

Ideas

determina™

I would like to automate the following tasks:

- Disassemble and run BinDiff on all files updated in a patch
- Compare patches for the same vulnerability in multiple versions of Windows: XP and 2003, IE 5 and IE 6, etc.
- Follow the code changes in a single function by diffing all versions of the file

determina™

BinDiff Demo

MS05-039

Plug and Play Buffer Overflow

determina™

Part II

Common Features of Microsoft Binaries

Common Features of Microsoft Binaries

determina™

- Visual C++ compiler optimizations
  - function chunking
  - reuse of stack frame slots
  - sbb comparison optimization
  - switch optimization
- Hotpatching
- Exception handling

Function Chunking

determina™

Function chunking is a compiler optimization for improving code locality. Profiling information is used to move rarely executed code outside of the main function body. This allows pages with rarely executed code to be swapped out.

It completely breaks tools that assume that a function is a contiguous block of code. IDA has supported chunked functions since version 4.7, but its function detection algorithm still has problems in some cases.

This optimization leaks profiling information into the binary. We know that the code in the main function body is executed more often than the function chunks. For code auditing purposes, we can focus on the function chunks, since they are more likely to contain rarely executed and insufficiently tested code.

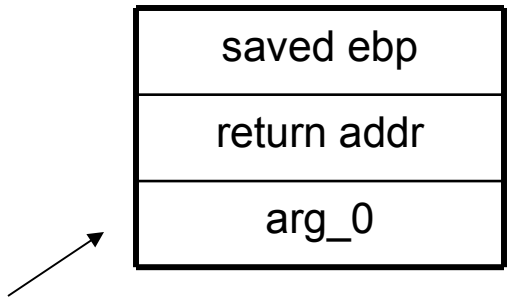
Reuse of Stack Frame Slots

determina

In non-optimized code, there is a one-to-one correspondence between local variables and the stack slots where they are stored. In optimized code, the stack slots are reused if there are multiple variables with non-overlapping live ranges.

For example:

```
int foo(Object* obj)
{
        int a = obj->bar();
        return a;
}
```



used for both `obj` and `a`

The live ranges of `obj` and `a` don't overlap, so they can be stored in same slot on the stack. The argument slot for `obj` is used for storing both variables.

SBB Comparison Optimization

determina™

The SBB instruction adds the second operand and the carry flag, and subtracts the result from the first operand.

- `sbb eax, ebx`

  eax = eax - (ebx + CF)

- `sbb eax, eax`

  eax = eax - (eax + CF)

  eax = - CF

SBB Comparison Optimization

The SBB instruction can be used to avoid branching in an `if` statement.

in assembly:

	ebx < ecx	ebx >= ecx
`cmp ebx, ecx`	CF = 1	CF = 0
`sbb eax, eax`	eax = -1	eax = 0
`inc eax`	eax = 0	eax = 1

in C:

```
if (ebx >= ecx)
        eax = 1;
else
        eax = 0;
```

SBB Comparison Optimization

determina

An example:

```
77D74796 loc_77D74796:
77D74796                mov eax, [esi+file_mapping.ptr]
77D74799                cmp [esi+file_mapping.end], eax
77D7479C                sbb eax, eax
77D7479E                inc eax
77D7479F
77D7479F return:
77D7479F                pop edi
77D747A0                pop esi
77D747A1                retn 0Ch
77D747A1 _ReadChunk@12 endp
```

in C:

```
// have we reached the end of the file?
return (file_mapping.ptr <= file_mapping.end);
```

Switch Optimization

determina

Non-optimized code :

```
switch (arg_0)
{
        case 1:         ...
        case 2:         ...
        case 3:         ...
        case 8001:      ...
        case 8002:      ...
}
```

```
00401030                cmp [ebp+arg_0], 1
00401034                jz short case_1
00401036                cmp [ebp+arg_0], 2
0040103A                jz short case_2
0040103C                cmp [ebp+arg_0], 3
00401040                jz short case_3
```

Switch Optimization

determina

Optimized code:

```
767AFDA1 _GetResDesSize@4 proc near
767AFDA1
767AFDA1        arg_0   = dword ptr 4
767AFDA1
767AFDA1                mov eax, [esp+arg_0]
767AFDA5                mov ecx, 8001h
767AFDAA                cmp eax, ecx
767AFDAC                ja short greater_than_8001
767AFDAE                jz short case_8001
767AFDB0                dec eax
767AFDB1                jz short case_1         ; after 1 dec
767AFDB3                dec eax
767AFDB4                jz short case_2         ; after 2 decs
767AFDB6                dec eax
767AFDB7                jz short case_3         ; after 3 decs
```

Microsoft Hotpatching

determina

The Microsoft hotpatching implementation is described in US patent application 20040107416. It is currently supported only on Windows 2003 SP1, but we'll probably see more of it in Vista.

The hotpatches are generated by an automated tool that compares the original and patched binaries. The functions that have changed are included in a file with a .hp.dll extension. When the hotpatch DLL is loaded in a running process, the first instruction of the vulnerable function is replaced with a jump to the hotpatch.

The /hotpatch compiler option ensures that the first instruction of every function is a `mov edi, edi` instruction that can be safely overwritten by the hotpatch. Older versions of Windows are not compiled with this option and cannot be hotpatched.

Exception Handling

determina™

This is better than anything I could have said about it:

Reversing Microsoft Visual C++ Part I: Exception Handling
by Igor Skochinsky:

http://www.openrce.org/articles/full_view/21

determina™

Part III

Improving Your Tools

Improving Your Tools

determina™

- IDA autoanalysis
  - Overview of the autoanalysis algorithm
  - Problems with the disassembly
- Improving the analysis with IDA plugins
  - Symbol loading plugin

Autoanalysis Algorithm

determina™

The autoanalysis algorithm is not documented, but it can be roughly described as follows:

1. Load the file in the database and create segments
2. Add the entry point and all DLL exports to the analysis queue
3. Find all typical code sequences and mark them as code. Add their addresses to the analysis queue
4. Get an address from the queue and disassemble the code at that address, adding all code references to the queue
5. If the queue is not empty, go to 4
6. Make a final analysis pass, converting all unexplored bytes in the text section to code or data

Autoanalysis Problems

determina™

There are a number of situations where the autoanalysis heuristics lead to incorrect disassembly. Some of these problems create serious difficulties for automatic analysis tools like BinDiff. The three main problems that I've seen are:

- Data disassembled as code
- Function detection problems
- Incorrectly identified local variables due to stack pointer tracking problems

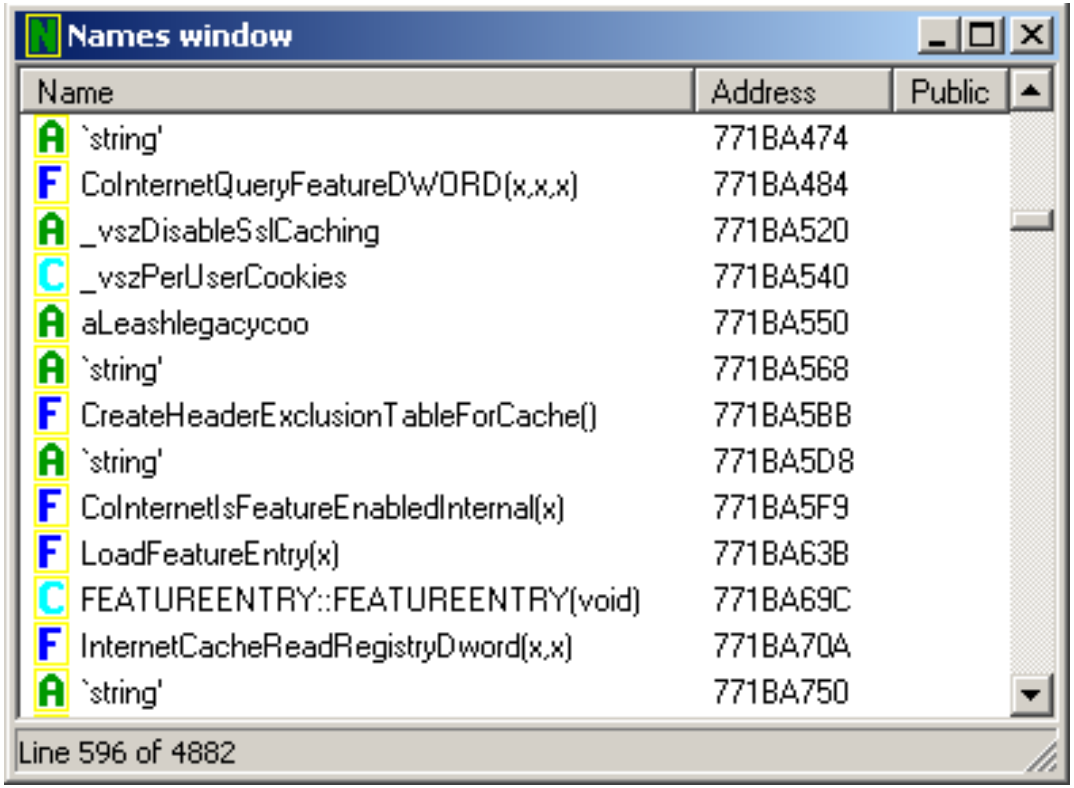
Autoanalysis Problems

determina

Code outside of a function is an indication of incorrectly disassembled data or a function detection problem:

should be a string →

should be a function →



Data Disassembled as Code

determina

```
771B7650 ; const CHAR _vszSyncMode
771B7650 _vszSyncMode:
771B7650                push ebx
771B7651                jns short near ptr loc_771B76BF+2
771B7653                arpl [ebp+6Fh], cx
771B7656
771B7656 loc_771B7656:
771B7656                db 64h, 65h
771B7656                xor eax, 48000000h
771B765D                imul esi, [ebx+74h], 2E79726Fh
771B7664                dec ecx
771B7665                inc ebp
```

Instead of:

```
771B7650 ; const CHAR _vszSyncMode
771B7650 _vszSyncMode db 'SyncMode5',0
```

Data Disassembled as Code

determina

Solutions:

- use PDB plugin from IDA 4.9 SP or later
- disable "Make final analysis pass"

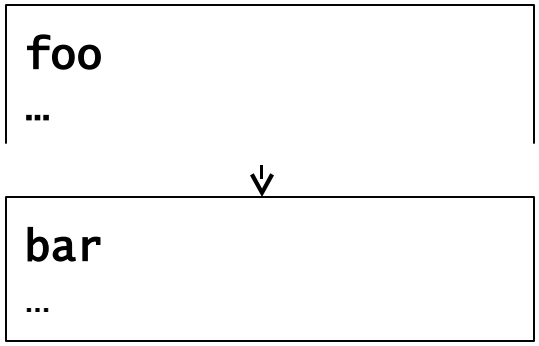
The PDB plugin in IDA 4.9 SP automatically creates strings for string variables, which partially solves this problem. This only works for binaries with debugging symbols.

The final analysis pass runs after the initial autoanalysis is complete and converts all unexplored bytes in the text segment to data or code. It is often too aggressive and disassembles data as code. Disabling the option ensures that only real code is disassembled, but might leave some functions unexplored. If it is disabled, only the first element in a vtable is analyzed, leaving the rest of the member functions unexplored.

Function Detection Problems

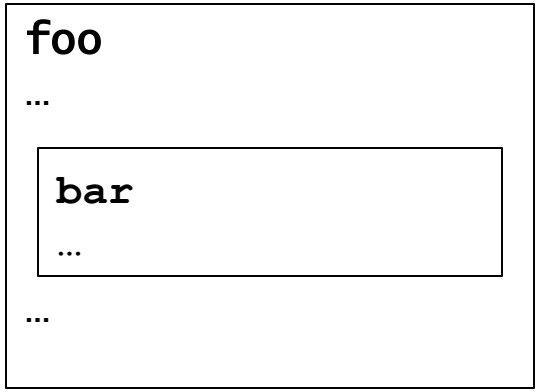
determina

- If foo is a wrapper around bar, the compiler can put the two functions next to each other and let foo fall through to bar.




```
void foo(a) {
        if (a == 0)
                return;
        else
                bar(a);
}
```

- Functions chunks inside another function



If the function foo is analyzed first, it will include the bar chunk. If bar is analyzed first, foo will be split in two chunks.

Stack Pointer Tracking Problems

determina

IDA uses a stack pointer propagation algorithm to convert esp relative data references to local variable references. If a function is called using the stdcall calling convention, but IDA assumes cdecl, the stack pointer after the call will be incorrect.

```
00405169 004          mov eax, [esp+4+var_4]
0040516D 004          push eax
0040516E 008          call func
00405174 008          mov [esp+8+var_8], eax
```

The second variable reference is actually to var_4:

```
00405169 004          mov eax, [esp+4+var_4]
0040516D 004          push eax
0040516E 008          call func
00405174 004          mov [esp+4+var_4], eax
```

Improving the Analysis with IDA Plugins

determina™

Ideas:

- Show code outside of functions
- Detect incorrect stack pointer values at branch merge points or at the end of the function
- Improve the symbol loading plugin with name and type based heuristics

IDA PDB Plugin

determina

- Source code included in the IDA SDK
- Uses the DbgHelp API
- Supports DBG and PDB symbols through dbghelp.dll
- Algorithm:
  - create names for all symbols
  - if the symbol name is `` `string' ``, create a C or UNICODE string
  - if the demangled name is not of type MANGLED_DATA, create a function at that address

Determina PDB Plugin

determina™

- Uses FPO records to detect functions
- Does not create functions for demangled names of an unknown type
  - reduces the instances of data disassembled as code
- Special handling for imports, floats, doubles and strings
- Creates vtables as arrays of function pointers
- The symbols are applied to the database starting from the end of the file and going up
  - significantly improves function chunking
- Much better GUI

Available under a BSD license from:

http://www.determina.com/security.research/

determina™

Determina PDB Plugin Demo

determina™

Questions?

asotirov@determina.com