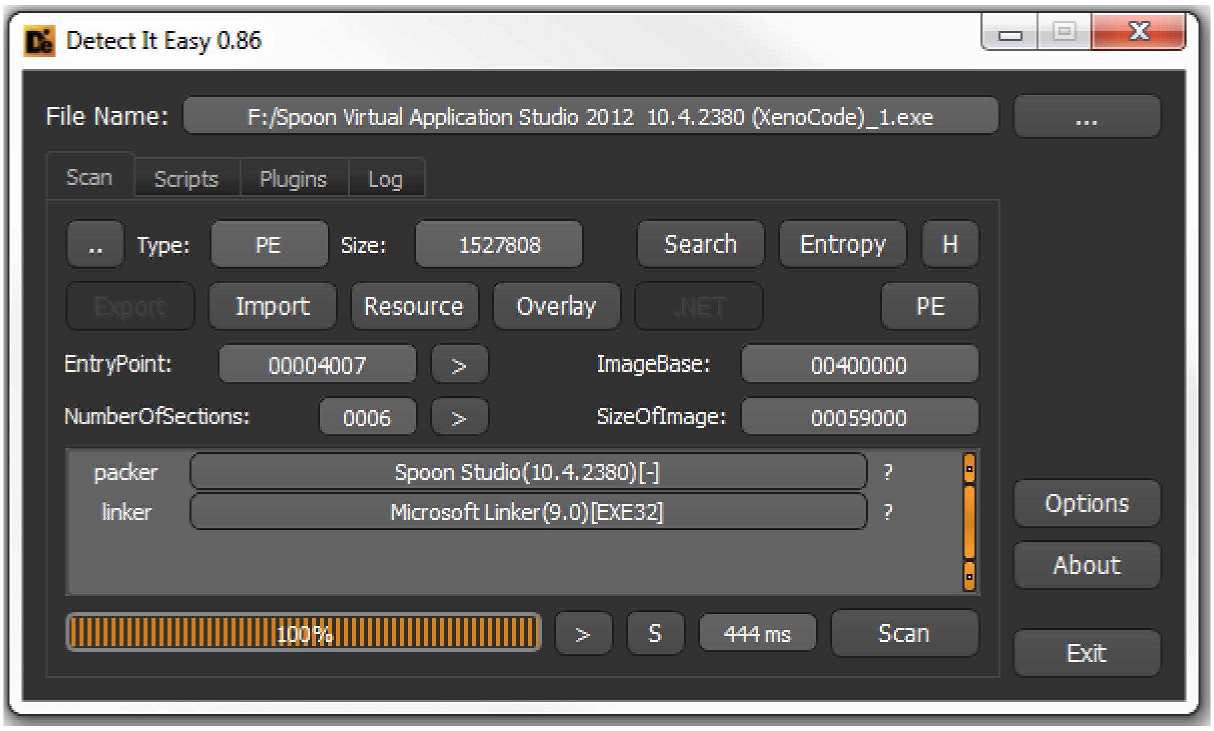
In this article it will be described in detail how to create signatures for the program “Detect It Easy”. Detect It Easy, or abbreviated “DIE” is a program for determining types of files.

“DIE” is a cross-platform application, apart from Windows version there are also available versions for Linux and Mac OS.



```
File Name: F:/Spoon Virtual Application Studio 2012  10.4.2380 (XenoCode)_1.exe
Type: PE    Size: 1527808
EntryPoint: 00004007          ImageBase: 00400000
NumberOfSections: 0006        SizeOfImage: 00059000
packer  Spoon Studio(10.4.2380)[-]        ?
linker  Microsoft Linker(9.0)[EXE32]      ?
100%    444 ms
```

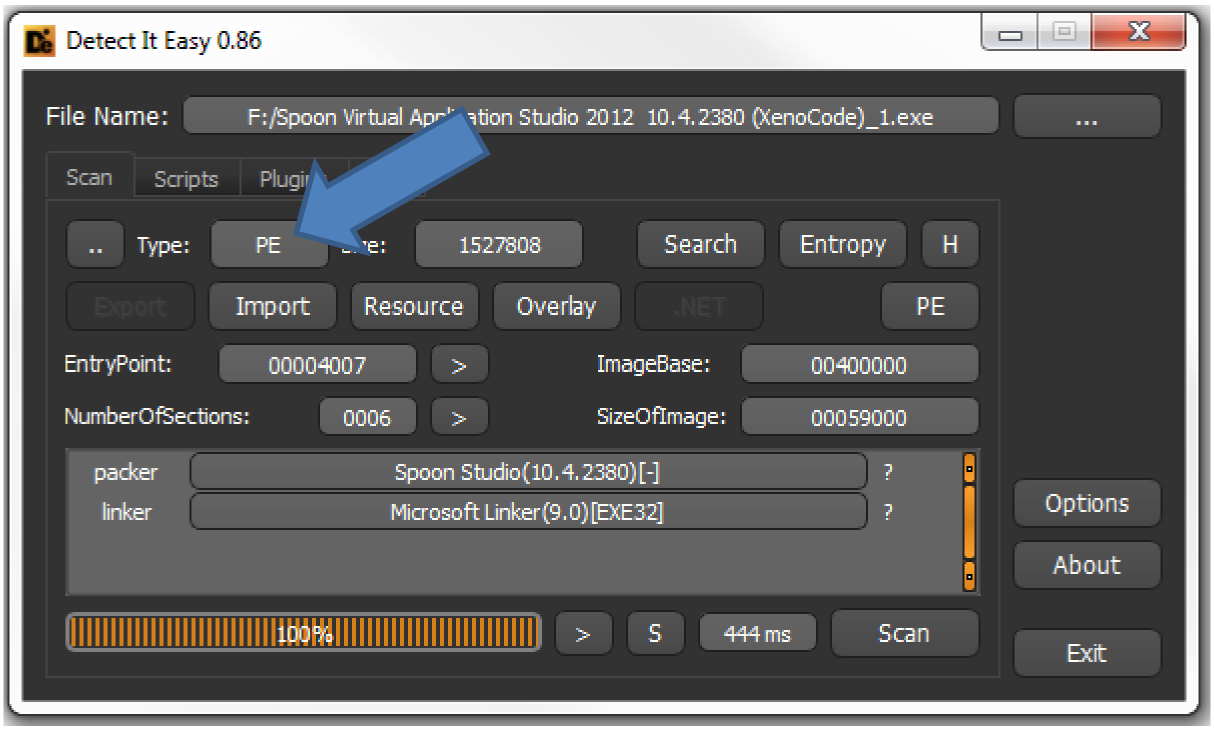
Many programs of the kind (PEID, PE tools) allow to use third-party signatures. Unfortunately, those signatures scan only bytes by the pre-set mask, and it is not possible to specify additional parameters. As the result, false triggering often occur. More complicated algorithms are usually strictly set in the program itself. Hence, to add a new complex detect one needs to recompile the entire project. No one, except the authors themselves, can change the algorithm of a detect. As time passes, such programs lose relevance without the constant support.

Detect It Easy has totally open architecture of signatures. You can easily add your own algorithms of detects or modify those that already exist. This is achieved by using scripts. The script language is very similar to JavaScript and any person, who understands the basics of programming, will understand easily how it works. Possibly, someone may decide the scripts are working very slow. Indeed, scripts run slower than compiled code, but, thanks to the good optimization of Script Engine, this doesn't cause any special inconvenience. The possibilities of open architecture compensate these limitations.

DIE exists in three versions. Basic version (“DIE”), Lite version (“DIEL”) and console version (“DIEC”). All the three use the same signatures, which are located in the folder “db”. If you open this folder, nested sub-folders will be found (“Binary”, “PE” and others). The names of sub-folders correspond to the types of files. First, DIE determines the type of file, and then sequentially loads all the signatures, which lie in the corresponding folder. Currently the program defines the following types:

- **MSDOS** executable files MS-DOS
- **PE** executable files Windows
- **ELF** executable files Linux
- **Text** plain text files.
- **Binary** all other files

By opening a file in DIE you can learn the file type:



File Name: F:/Spoon Virtual App[...]ation Studio 2012 10.4.2380 (XenoCode)_1.exe

Type: PE Size: 1527808

EntryPoint: 00004007 ImageBase: 00400000

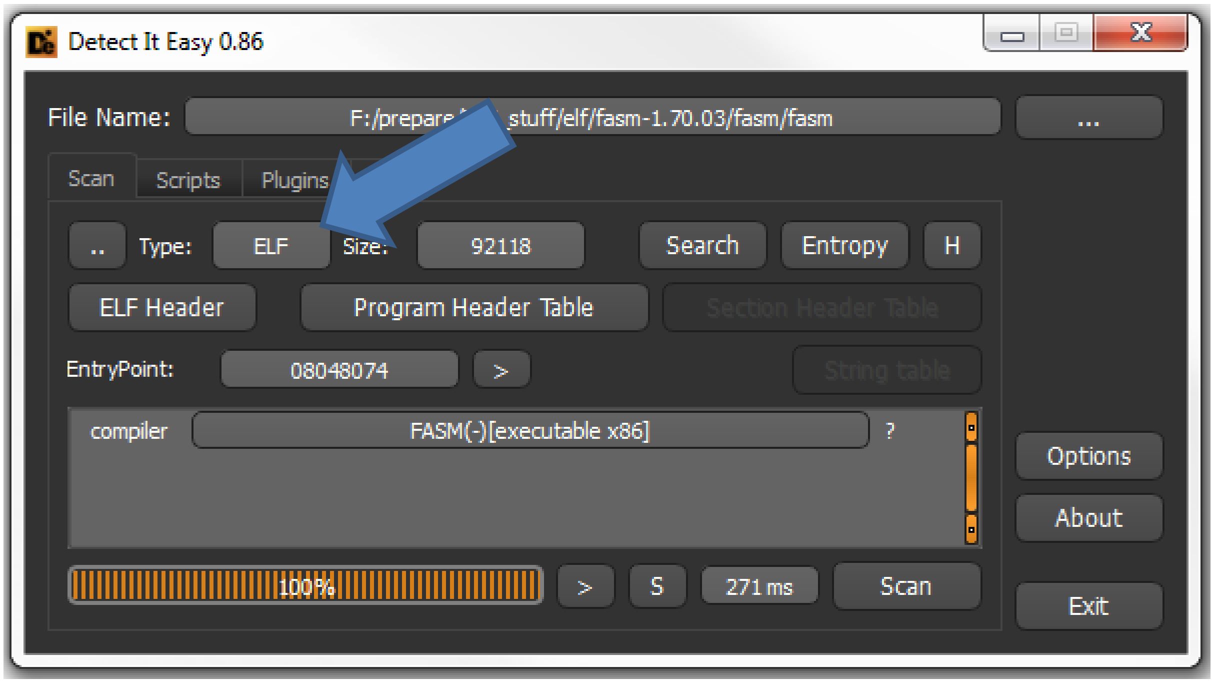
NumberOfSections: 0006 SizeOfImage: 00059000

packer Spoon Studio(10.4.2380)[-] ?

linker Microsoft Linker(9.0)[EXE32] ?

100% 444 ms

The central part of the window also varies depending on the type:



The signatures are ordinary text files, only with the extension (*.sg). These can be edited in any text editor. Each signature must have in its body the function “detect”, which starts when you boot. The prototype of this function:

function detect(bShowType,bShowVersion,bShowOptions)

where **bShowType, bShowVersion** and **bShowOptions** are flags, which should be set 0 or 1. If signature is successfully deployed, the “detect” function should return a string in accordance with the set flags.

Let's illustrate this with an example.

If all three flags are set, the UPX packer will return such a string:

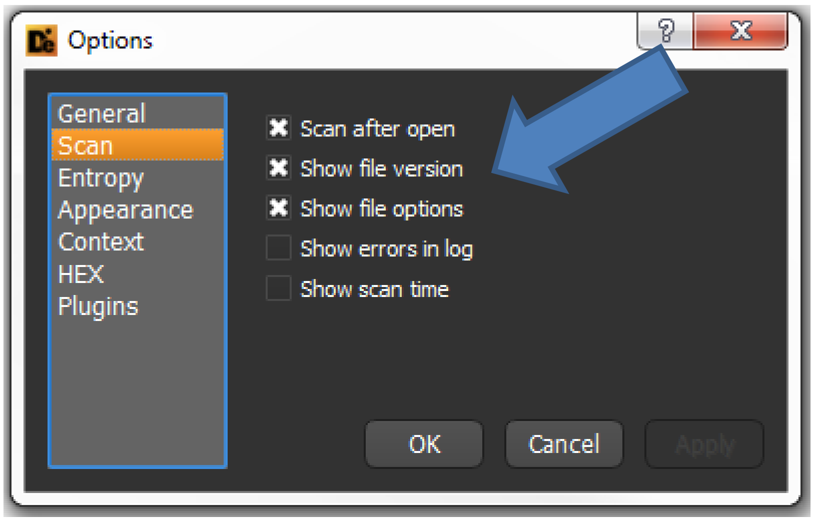
packer: UPX(3.09)[DLL32]

- packer—type of signature (Type). Type of signatures shall not be confused with file types. For example, for ASPack http://www.aspack.com/aspack.html the file type is PE, and the type of signature is “packer”.
- **UPX**—Name
- **3.09**—Version
- **DLL32**—Options

You can ignore this recommendation and simply return the string of arbitrary format. But then a basic version (DIE) might not be able to properly display the result (for Lite and console versions this is not important). So that DIE was able to correctly show the result, the string must have at least type and name, separated by “:”.

Type: Name

In the console and basic versions (DIE and DIEL) you can use flags **bShowVersion** and **bShowOptions**. In the basic version this can be done in the settings.



☒ Scan after open
☒ Show file version
☒ Show file options
☐ Show errors in log
☐ Show scan time

In the console version this can be done with the keys -showoptions and -showversion (by default both flag are set 1).

If you use ready signature template, it is not necessary to think how to properly form a returned string. In the template this is all already implemented.

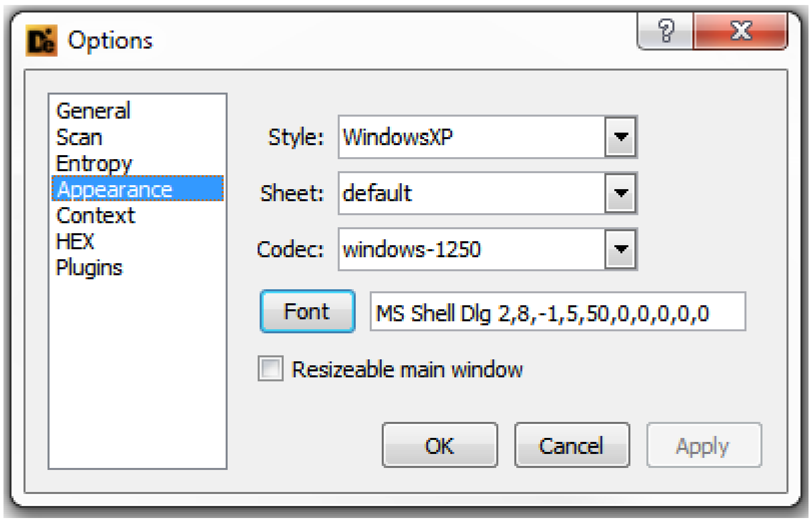
The presence of “detect” function signature is optional, but in addition to the functions user functions can also be located in the file. For example:

```
function sum(arg1,arg2)
{
    return arg1+arg2;
}
function detect(bShowType,bShowVersion,bShowOptions)
{
    var  k=1;
    var  l=3;
    var  s=sum(k,l);
```

As we already mentioned, you can use any text editor for editing and creation of new signatures. Also this is possible by the program itself. To do so, let's launch a basic version DIE and open any file.

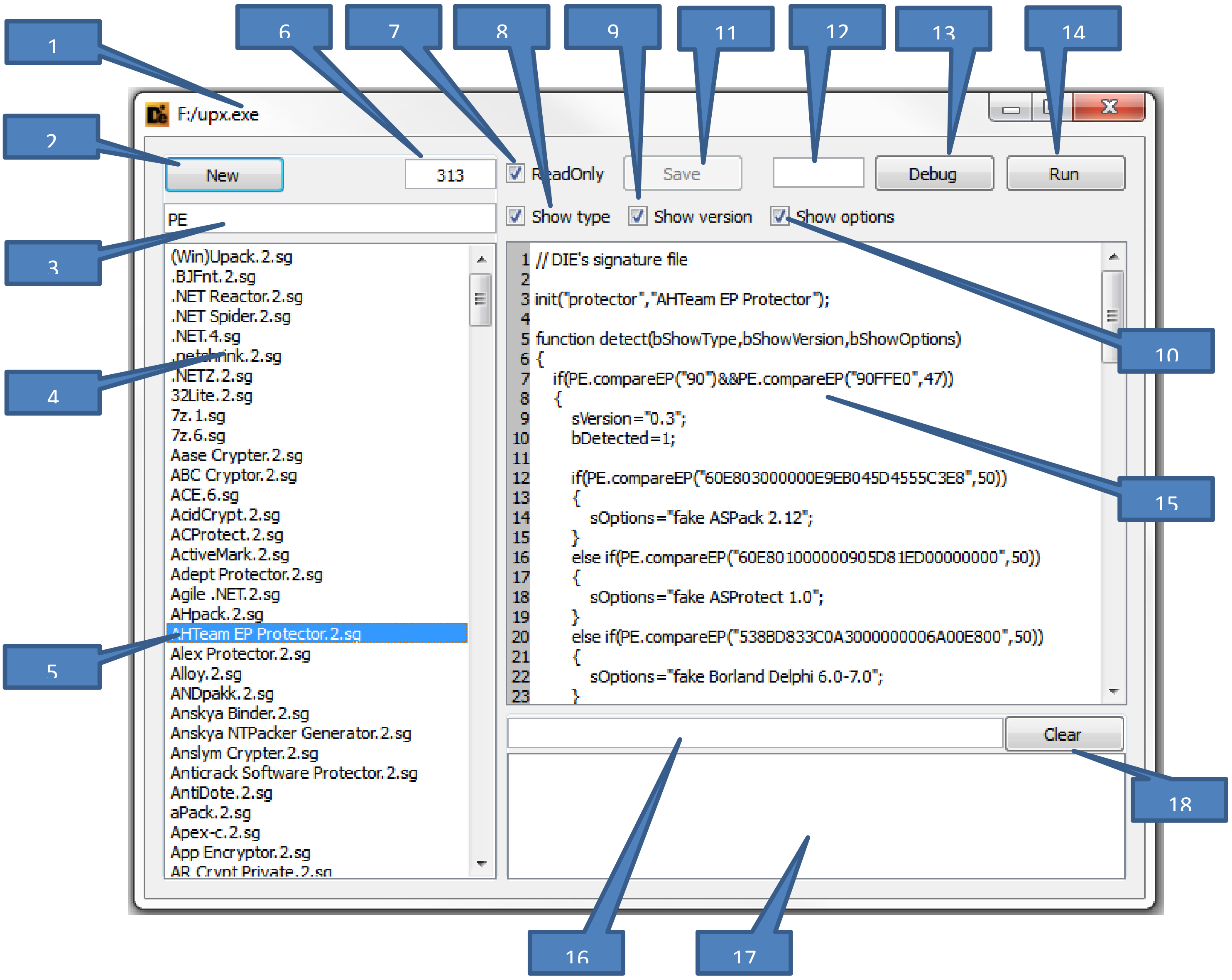
A small comment: for comfortable work with signatures (especially with a debugger) you'd better disable all custom styles of appearance. To do so, we switch to the tab "Appearance" in the menu settings, and set the styles to default:

Style	WindowsXP
Style sheet	default


Style: WindowsXP
Sheet: default
Codec: windows-1250
Font: MS Shell Dlg 2,8,-1,5,50,0,0,0,0,0
☐ Resizeable main window

Then click the button with the caption "S" at the bottom of the window.

Signature Editor opens:



(Win)Upack.2.sg
.BJFnt.2.sg
.NET Reactor.2.sg
.NET Spider.2.sg
.NET.4.sg
.netshrink.2.sg
.NETZ.2.sg
32Lite.2.sg
7z.1.sg
7z.6.sg
Aase Crypter.2.sg
ABC Cryptor.2.sg
ACE.6.sg
AcidCrypt.2.sg
ACProtect.2.sg
ActiveMark.2.sg
Adept Protector.2.sg
Agile .NET.2.sg
AHpack.2.sg
AHTeam EP Protector.2.sg
Alex Protector.2.sg
Alloy.2.sg
ANDpakk.2.sg
Anskya Binder.2.sg
Anskya NTPacker Generator.2.sg
Anslym Crypter.2.sg
Anticrack Software Protector.2.sg
AntiDote.2.sg
aPack.2.sg
Apex-c.2.sg
App Encryptor.2.sg
AR Crypt Private.2.sg

```
// DIE's signature file

init("protector","AHTeam EP Protector");

function detect(bShowType,bShowVersion,bShowOptions)
{
    if(PE.compareEP("90")&&PE.compareEP("90FFE0",47))
    {
        sVersion="0.3";
        bDetected=1;

        if(PE.compareEP("60E803000000E9EB045D4555C3E8",50))
        {
            sOptions="fake ASPack 2.12";
        }
        else if(PE.compareEP("60E801000000905D81ED00000000",50))
        {
            sOptions="fake ASProtect 1.0";
        }
        else if(PE.compareEP("538BD833C0A3000000006A00E800",50))
        {
            sOptions="fake Borland Delphi 6.0-7.0";
        }
```

1. The title of the window with the name of the opened file.
2. The button for creation of a new signature.
3. The type of the opened file.
4. The list of the signatures, available for this type.
5. The selected signature.
6. The number of available signatures.
7. Turn on edit mode.
8. Setting bShowType flag.
9. Setting bShowVersion flag.
10. Setting bShowOptions flag.

11. Save the signature.
12. Execution Time of the signature.
13. Start execution of the code of signature in the can.
14. Start execution of the code of signature.
15. Code of the selected signature.
16. String result of execution of the signature.
17. Log window.
18. Clean the result line and log window.

Each type of file has its own built-in functions. They are requested as

<file type>.<function name>

For example, Binary.readDword or PE.compareEP.

The list of built-in functions is constantly expanding and if there are any ideas which features you want to add, please write to horsicq@gmail.com

A complete description of all the features can be found in SDK/signatures/index.html

Besides, there are the so-called "global functions" that can be accessed from anywhere.

The most important of these features is "includeScript". By using this function, you can include other scripts in a script. „$DIEAPP/db" is the default folder for including scripts.

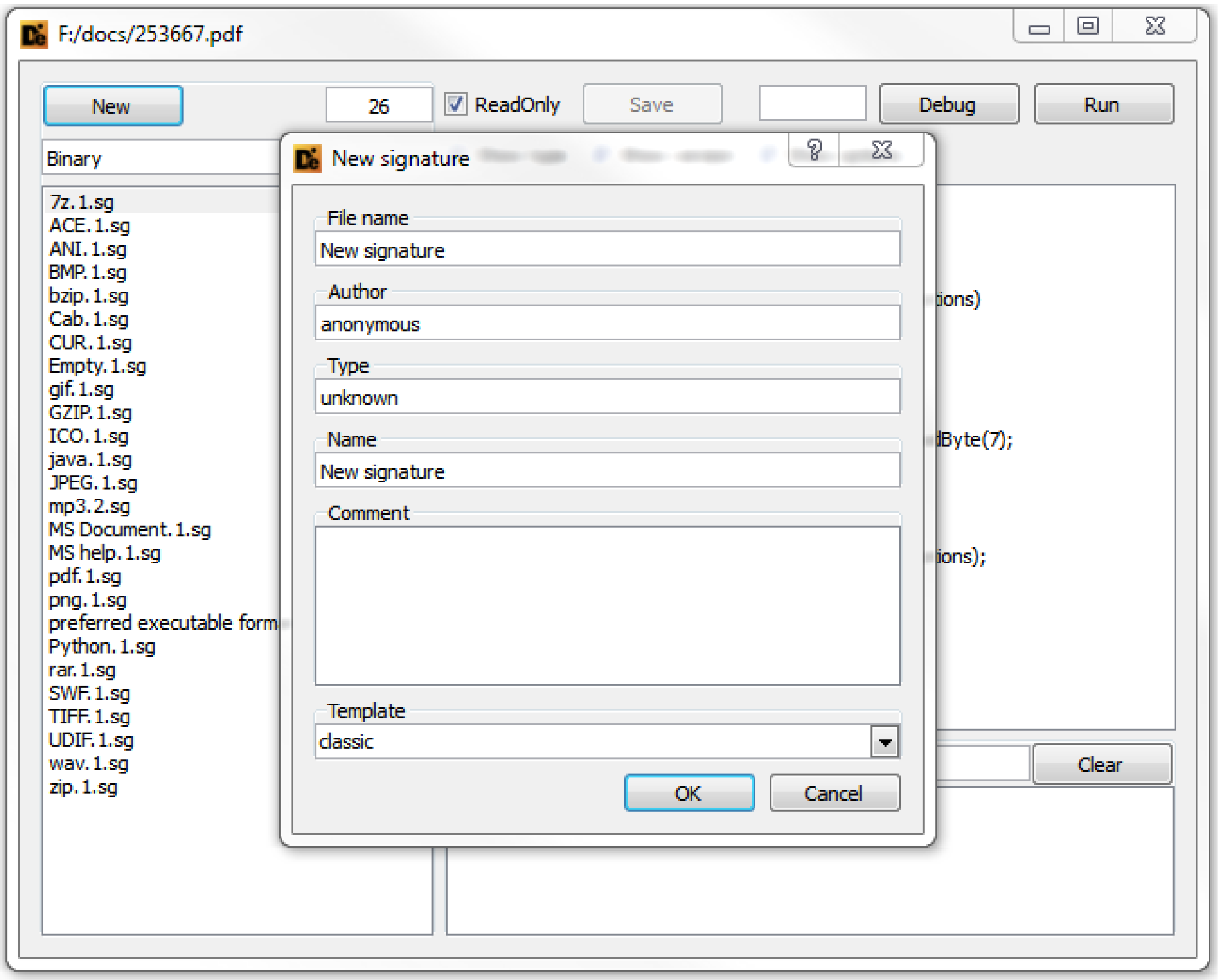
```
includeScript("result"); // adds a script $DIEAPP/db/result
```

It is also convenient to use the function «_log» to display debug messages.

```
_log("Hello world!"); // displays the string.
_log(123); // displays the number.
```

There is a special «_init» file in a folder of each type. For example, "$DIEAPP/db/PE/_init". This file is loaded by default before running all scripts and you can put there functions and global variables common to this type of files (in this case "PE").

Now, for example, let's try to create a signature for PDF. To do so, we will open any file of this type. Then we proceed to the signature editor and click "New". A window for creation of a new signature will appear.


F:/docs/253667.pdf

New | 26 | ☑ ReadOnly | Save | Debug | Run

Binary

7z.1.sg
ACE.1.sg
ANI.1.sg
BMP.1.sg
bzip.1.sg
Cab.1.sg
CUR.1.sg
Empty.1.sg
gif.1.sg
GZIP.1.sg
ICO.1.sg
java.1.sg
JPEG.1.sg
mp3.2.sg
MS Document.1.sg
MS help.1.sg
pdf.1.sg
png.1.sg
preferred executable form
Python.1.sg
rar.1.sg
SWF.1.sg
TIFF.1.sg
UDIF.1.sg
wav.1.sg
zip.1.sg

New signature

File name: New signature

Author: anonymous

Type: unknown

Name: New signature

Comment:

Template: classic

OK | Cancel

Any string can be set as file name, but it will be easier to deal with signatures if the file name matches the name of the defined signature. That is, if it is created for PDF, the file is best named as “PDF”, not, for instance, “new3”. If a file with the same name already exists, you can use an underscore or numbers (_PDF, PDF2). Of course you can create a signature that would define several types of files (for instance, PDF and JPG), but it is better to do individual signatures.

You can also set priority of scanning. Although this is not required. To do so the name of a file must look in this way:

The name.[priority].sg

Priority can have values from 0 to 9. 0 is the highest, 9—the lowest.

For example, “MicroJoiner.1.sg” or “Microsoft Visual Studio.3.sg”. Sometimes it is useful to ensure that the result of the scanning is shown in a certain sequence. For example, to show the information on a compiler before information on a linker. To do so, one can use priority 3 for compilers and priority 5 for linkers.

Insert the following data in the dialog box:

File Name	PDF	Since a file with the name PDF already exists.
Author	Me	You can leave this field blank.
Type	Format	You can insert any string, it's just better not be left blank.
Name	PDF	The name of the defined signature.
Comment	11.06.2014	You can leave this field blank.

Also one can choose a template to use. All templates are located in the folder „$DIEAPP/editor/templates”, they can be edited and new ones can be added.

If you select a “classic” template(“classic”), file “PDF.sg” appears looking like that:

```
// DIE's signature file
// Author: Me
/*
11.06.2014
*/
function detect(bShowType,bShowVersion,bShowOptions)
{
    var sType="Format";
    var sName="PDF";
    var sVersion="-";
    var sOptions="-";
    var sResult="";
    var nDetected=0;
    // Start of user's code
    // End of user's code
    if(nDetected)
    {
        if(bShowType)
        {
            sResult+=sType+": ";
        }
        sResult+=sName;
        if(bShowVersion)
        {
            sResult+="("+sVersion+")";
        }
        if(bShowOptions)
        {
            sResult+="["+sOptions+"]";
        }
    }
    return sResult;
}
```

We search—for instance, in Google,—information on the structure of PDF files. Links like http://resources.infosecinstitute.com/pdf-file-format-basic-structure/ are easily found.

So, in the beginning of every PDF file there is the double-word 0x46445025, and by the offset 5 we can find a string representing the version number with length of 3 characters.

Write the following code:

```
// Start of user's code
if(Binary.getSize()>=8) // If the file length is less than 8 bytes, obviously
it is not PDF.
{
   if(Binary.readDword(0)==0x46445025) // Read the first double word and
compare.
   {
         sVersion=Binary.getString(5,3); // Define version.
         nDetected=1; // Set as 1, if checks are successfull.
    }
}
// End of user's code
```

To verify signature we will click the button “Run”.

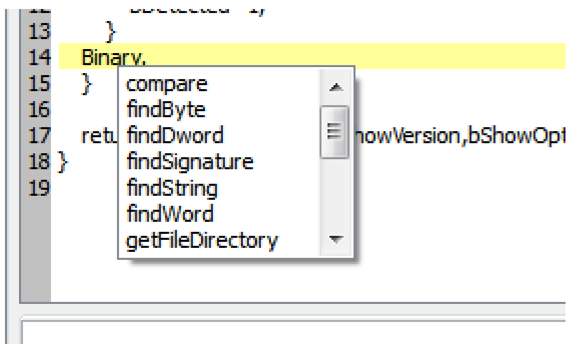
A line will appear in the result box:

Format: PDF(1.2)[-]

Since the “options” will always be “-”, you can delete the following lines of code:

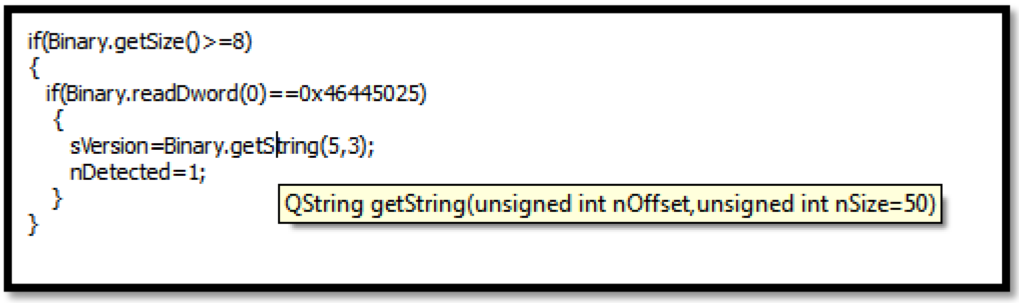
```
var sOptions="-";
if(bShowOptions)
{
     sResult+="["+sOptions+"]";
}
```

If we insert “<type>.” in the editor window and press the keyboard shortcut “Ctrl+Space”, it will show all built-in functions for this type (in this case, “Binary”):



```
   }
   Binary.
   }
   retu                                   howVersion,bShowOpt
 }
```

If you hover the text cursor over the built-in functions and press the keyboard shortcut “Alt”, a prototype of function will appear:

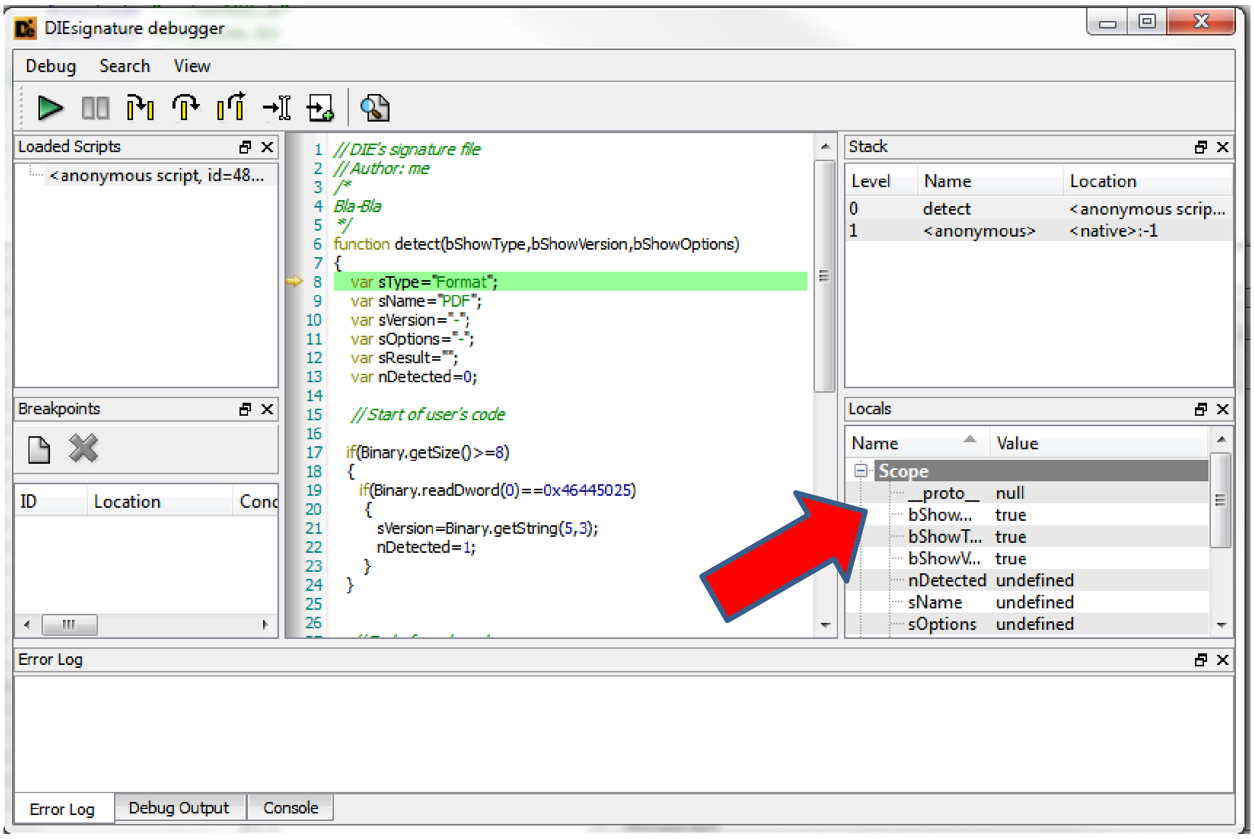


```
if(Binary.getSize()>=8)
{
  if(Binary.readDword(0)==0x46445025)
  {
    sVersion=Binary.getString(5,3);
    nDetected=1;
  }
}
```

QString getString(unsigned int nOffset,unsigned int nSize=50)

Signatures also can be debugged. For this you have to click the button “Debug”.

A window appears:



```
//DIE's signature file
//Author: me
/*
Bla-Bla
*/
function detect(bShowType,bShowVersion,bShowOptions)
{
  var sType="Format";
  var sName="PDF";
  var sVersion="-";
  var sOptions="-";
  var sResult="";
  var nDetected=0;

  //Start of user's code

  if(Binary.getSize()>=8)
  {
    if(Binary.readDword(0)==0x46445025)
    {
      sVersion=Binary.getString(5,3);
      nDetected=1;
    }
  }
```

Level	Name	Location
0	detect	<anonymous scrip...
1	<anonymous>	<native>:-1

Name	Value
Scope	
__proto__	null
bShow...	true
bShowT...	true
bShowV...	true
nDetected	undefined
sName	undefined
sOptions	undefined

Debugger has standard management. All used local variables are located in the “Scope”.

Signatures for PEID can be used in the DIE also.

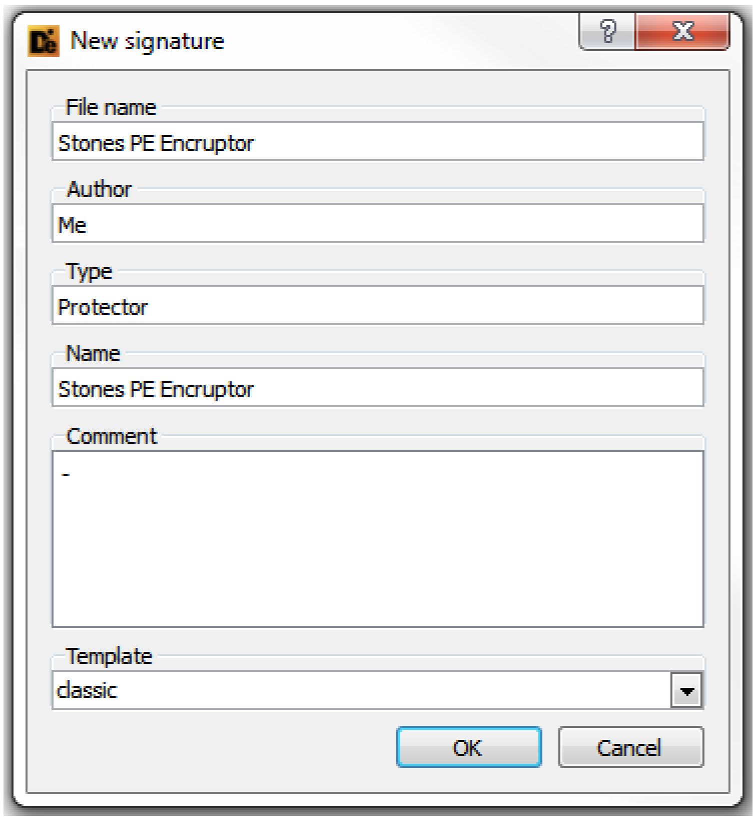
For example, there is such a PEID-signature:

```
[Stones PE Encryptor v1.13]
signature = 55 57 56 52 51 53 E8 ?? ?? ?? ?? 5D 8B D5 81
ep_only = true
```

Let's launch the signature editor for PE files. To do so, you need to open any PE-file and click the button “New”

Then we enter the following data:

File Name	Stones PE Encruptor	Name
Author	Me	You can leave this field blank.
Type	Protector	Type
Name	Stones PE Encruptor	The name of the defined signature.
Comment	-	You can leave this field blank.


New signature

File name
Stones PE Encruptor

Author
Me

Type
Protector

Name
Stones PE Encruptor

Comment
-

Template
classic

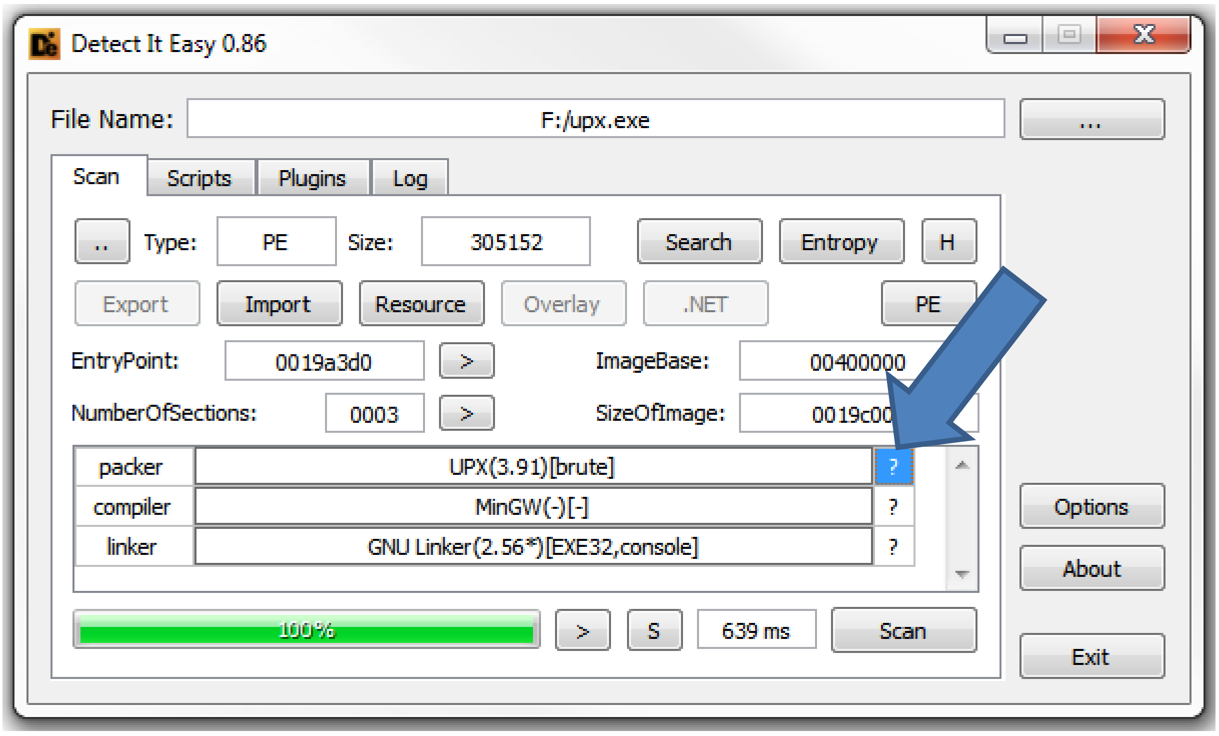
The template appears:

```
// DIE's signature file
// Author: Me
/*
-
*/
function detect(bShowType,bShowVersion,bShowOptions)
{
    var sType="Protector";
    var sName="Stones PE Encryptor";
    var sVersion="-";
    var sOptions="-";
    var sResult="";
    var nDetected=0;
    // Start of user's code

    // End of user's code
    if(nDetected)
    {
        if(bShowType)
        {
            sResult+=sType+": ";
        }
        sResult+=sName;
        if(bShowVersion)
        {
            sResult+="("+sVersion+")";
        }
        if(bShowOptions)
        {
            sResult+="["+sOptions+"]";
        }
    }
    return sResult;
}
```

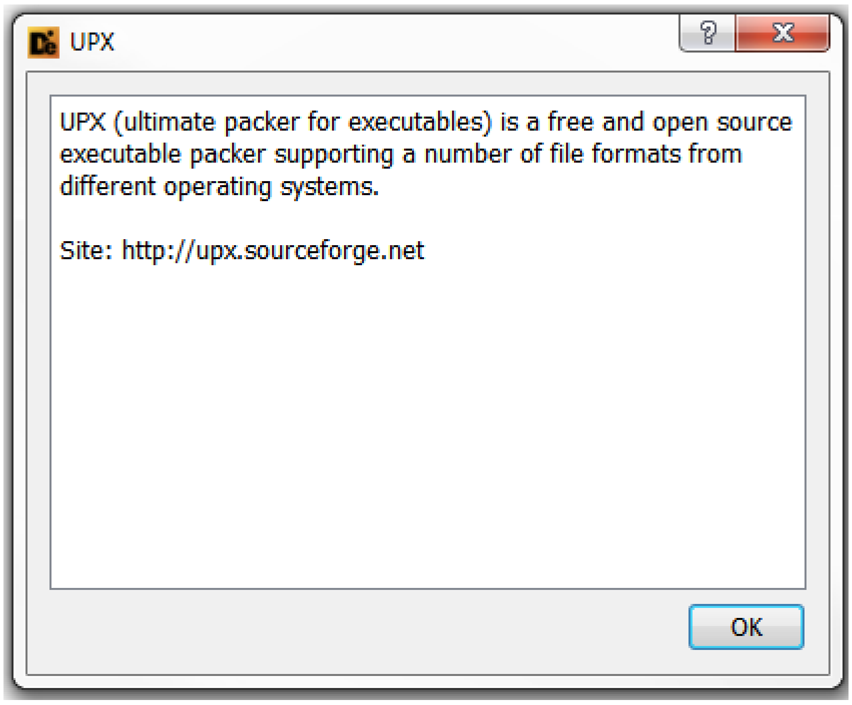
We write the following code:

```
// Start of user's code
if(PE.comparePE("55 57 56 52 51 53 E8 ?? ?? ?? ?? 5D 8B D5 81")) // Our PEID
signature   {
         sVersion=" 1.13"; // Version
         sOptions="-"; // Unknown options
         nDetected=1;  // Set as 1, if checks are successful

   }
// End of user's code
```

If there is a little more information, you can put it in the .html file in the “info” folder of the program. If you click on the button with a question mark in a basic version, a window appears with this additional information. For sure, this happens only if the corresponding file exists.



packer	UPX(3.91)[brute]	?
compiler	MinGW(-)[-]	?
linker	GNU Linker(2.56*)[EXE32,console]	?

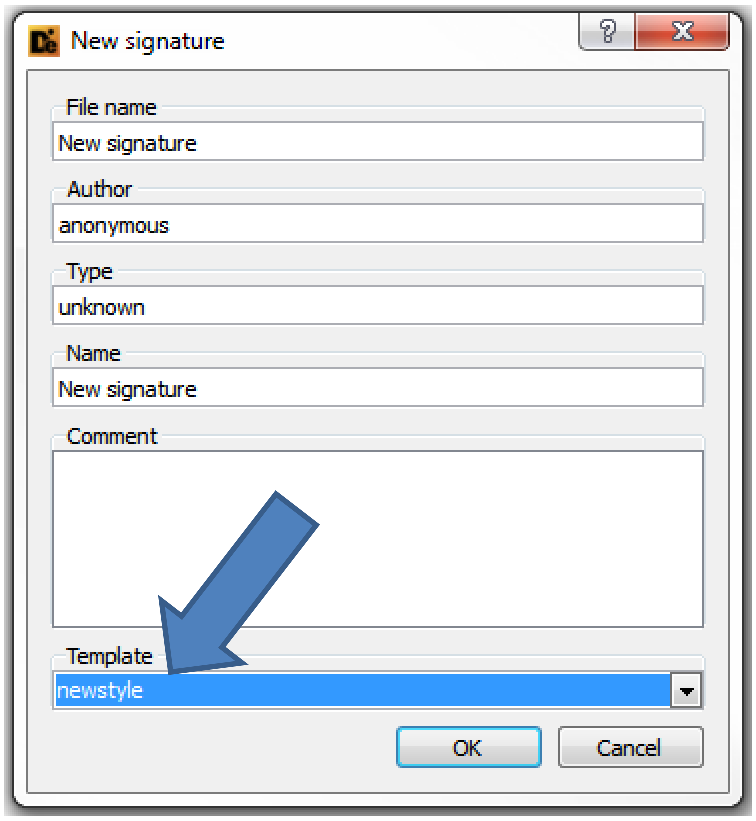


UPX (ultimate packer for executables) is a free and open source executable packer supporting a number of file formats from different operating systems.

Site: http://upx.sourceforge.net

It was already mentioned you can use your own templates for scripts.

Jason Hood (http://adoxa.altervista.org) has written a new system of templates (“newstyle”):


New signature

File name: New signature

Author: anonymous

Type: unknown

Name: New signature

Comment:

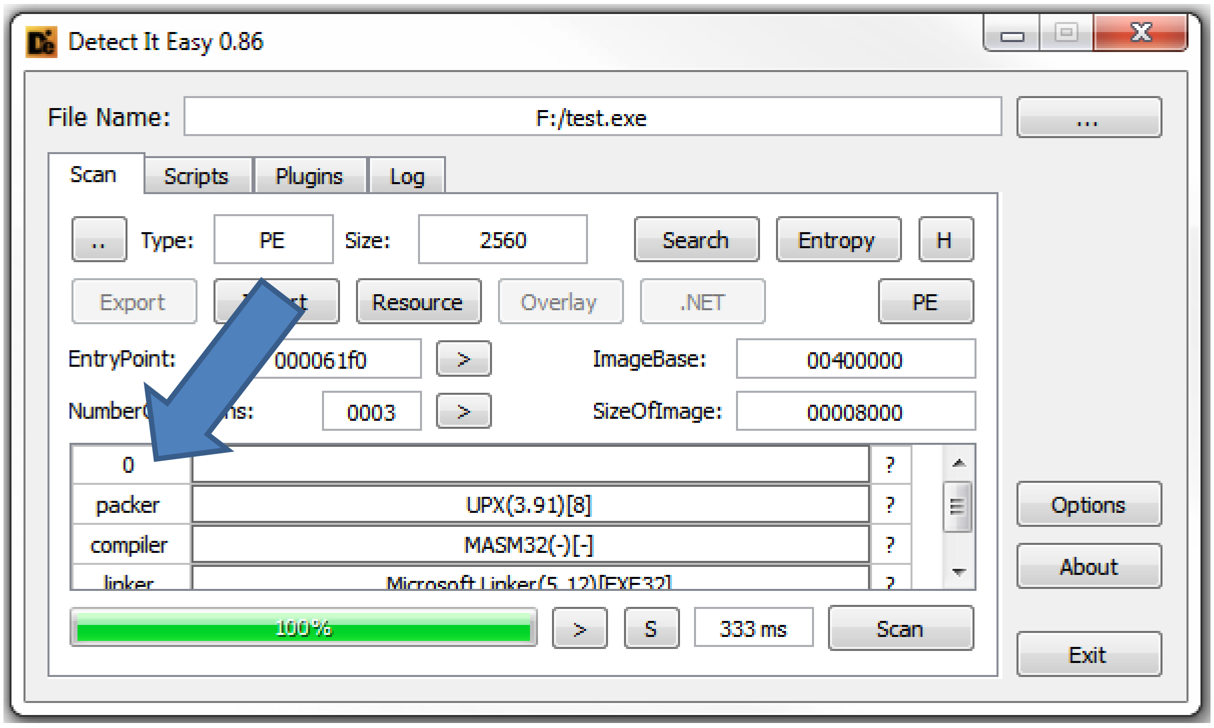
Template: newstyle

In this system of templates our example can be written as:

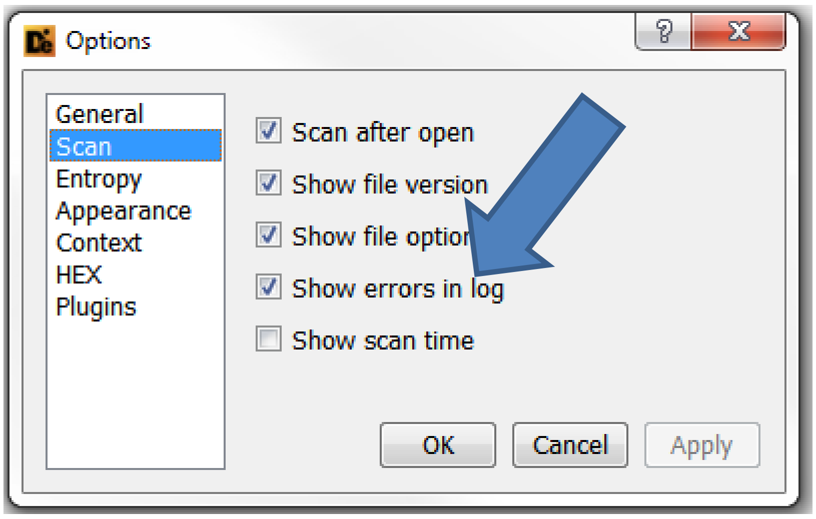
```
// DIE's signature file
// Author: Me
/*
-
*/
init("Protector "," Stones PE Encryptor ");

function detect(bShowType,bShowVersion,bShowOptions)
{
    // Start of user's code
    if(PE.comparePE("55 57 56 52 51 53 E8 ?? ?? ?? ?? 5D 8B D5 81")) // Our
PEID signature
        {
            sVersion=" 1.13"; // Version
            sOptions="-"; // Unknown options
            bDetected =1;  // Set as 1, if checks are successful
        }
    // End of user's code
    return result(bShowType,bShowVersion,bShowOptions);
}
```

Let's expand a little bit on errors in a script.

If the following result has appeared during scanning a file:

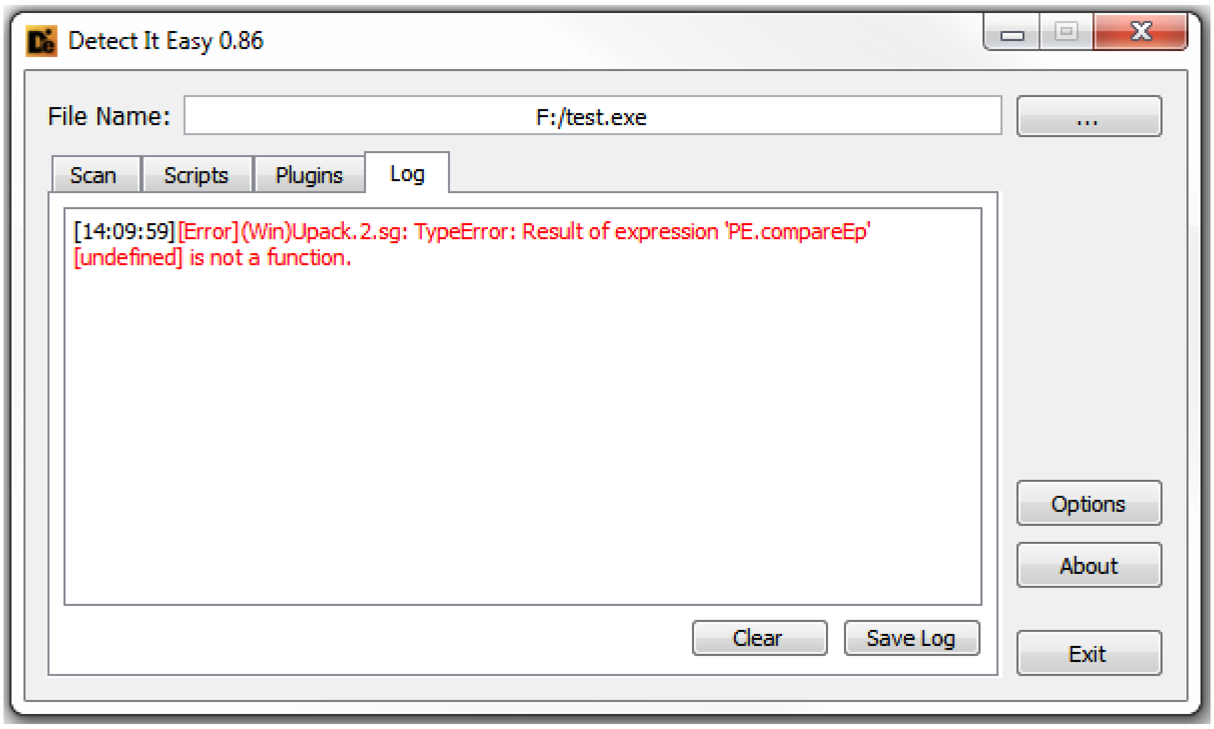


It means an error has occurred somewhere. To find an exact place we should enter the settings and turn on displaying errors:



- [✓] Scan after open
- [✓] Show file version
- [✓] Show file options
- [✓] Show errors in log
- [ ] Show scan time

Then we launch scanning files once again, and then turn to the log window:

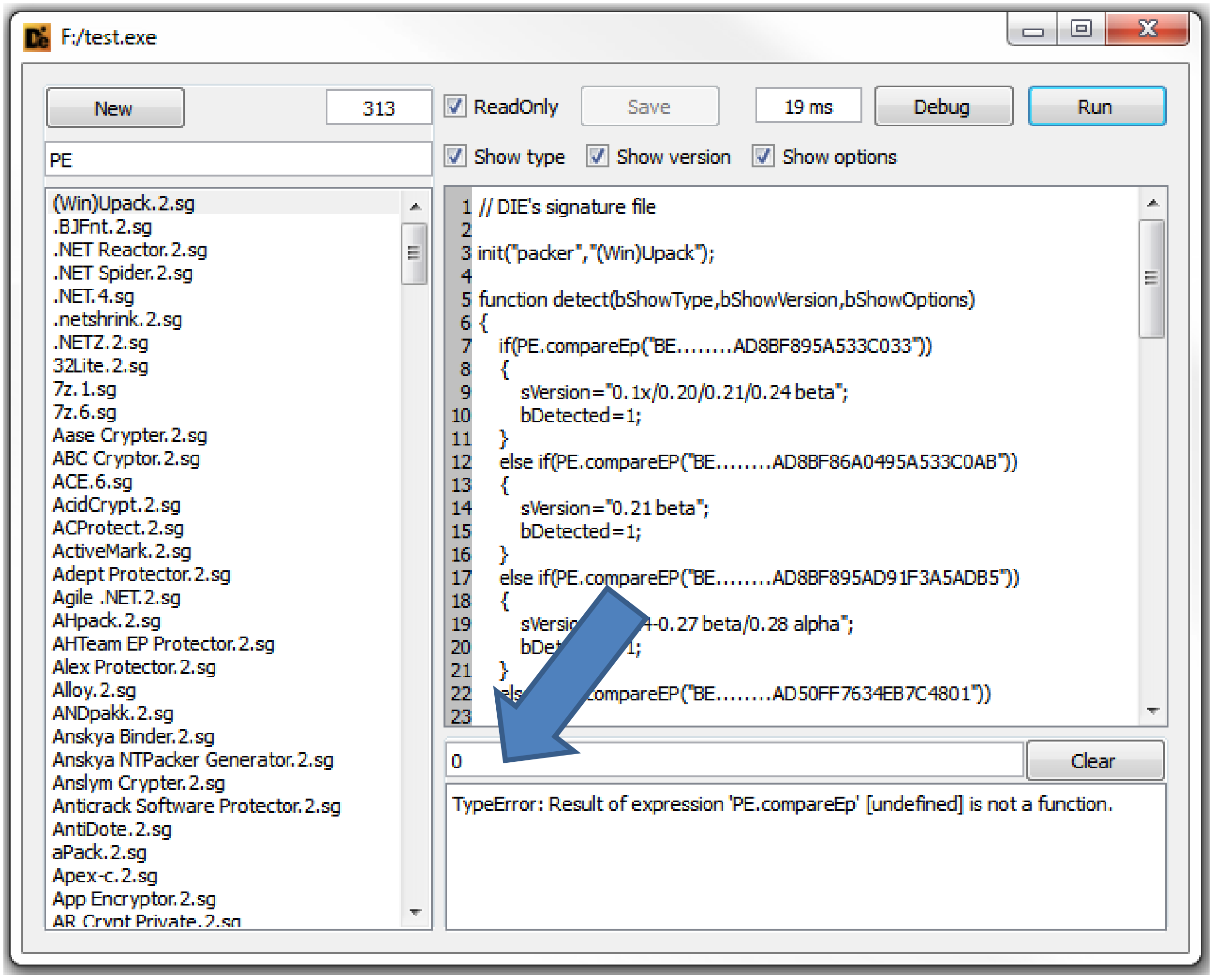


F:/test.exe

```
[14:09:59][Error](Win)Upack.2.sg: TypeError: Result of expression 'PE.compareEp'
[undefined] is not a function.
```

This means an error occurred somewhere in the file “(Win)Upack.2.sg”.

We open the file in the signature editor and click “Run” button.



☑ ReadOnly ☑ Show type ☑ Show version ☑ Show options

PE

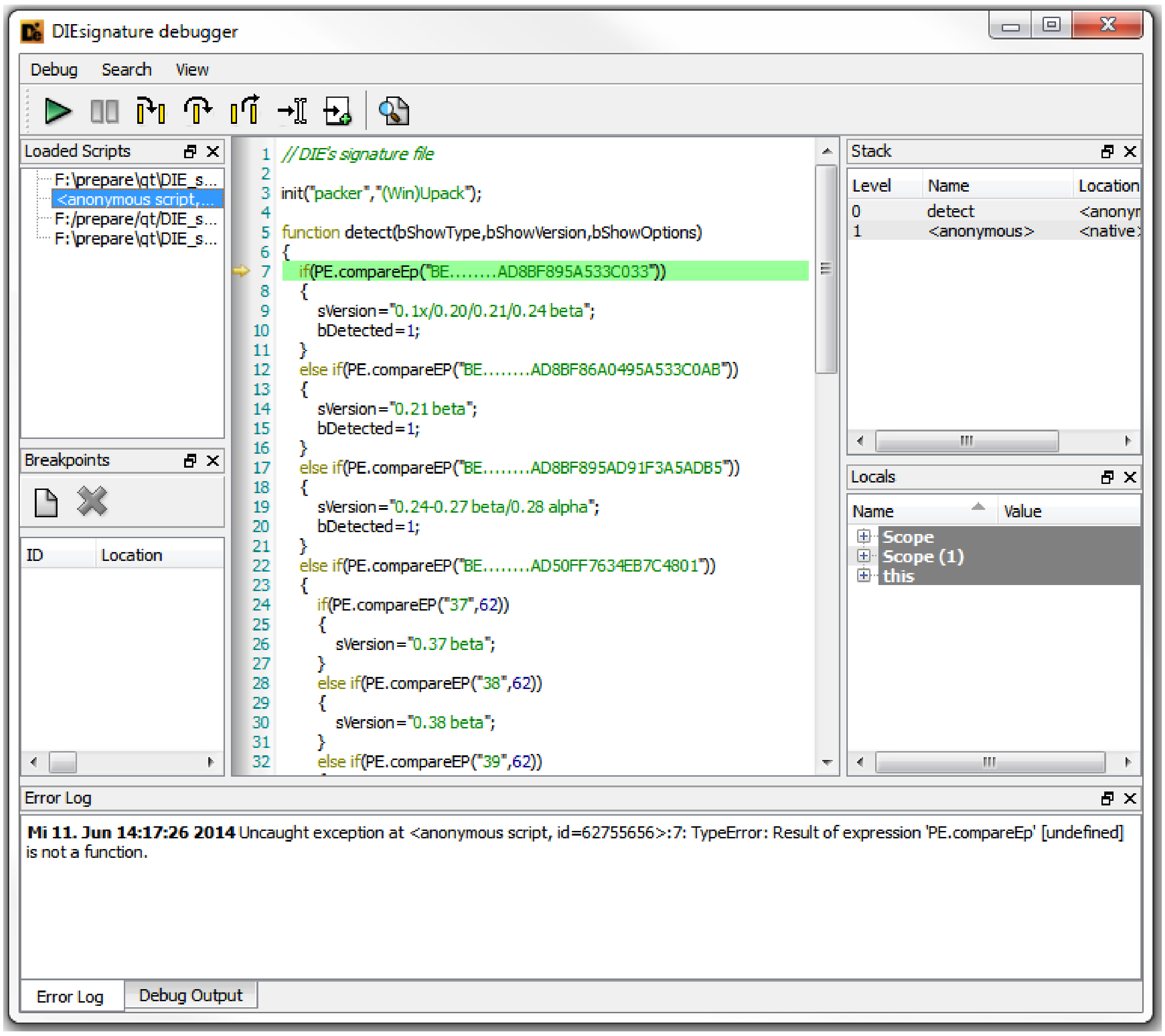
```
(Win)Upack.2.sg
.BJFnt.2.sg
.NET Reactor.2.sg
.NET Spider.2.sg
.NET.4.sg
.netshrink.2.sg
.NETZ.2.sg
32Lite.2.sg
7z.1.sg
7z.6.sg
Aase Crypter.2.sg
ABC Cryptor.2.sg
ACE.6.sg
AcidCrypt.2.sg
ACProtect.2.sg
ActiveMark.2.sg
Adept Protector.2.sg
Agile .NET.2.sg
AHpack.2.sg
AHTeam EP Protector.2.sg
Alex Protector.2.sg
Alloy.2.sg
ANDpakk.2.sg
Anskya Binder.2.sg
Anskya NTPacker Generator.2.sg
Anslym Crypter.2.sg
Anticrack Software Protector.2.sg
AntiDote.2.sg
aPack.2.sg
Apex-c.2.sg
App Encryptor.2.sg
AR Crypt Private.2.sg
```

```
// DIE's signature file

init("packer","(Win)Upack");

function detect(bShowType,bShowVersion,bShowOptions)
{
  if(PE.compareEp("BE........AD8BF895A533C033"))
  {
    sVersion="0.1x/0.20/0.21/0.24 beta";
    bDetected=1;
  }
  else if(PE.compareEP("BE........AD8BF86A0495A533C0AB"))
  {
    sVersion="0.21 beta";
    bDetected=1;
  }
  else if(PE.compareEP("BE........AD8BF895AD91F3A5ADB5"))
  {
    sVersio[illegible]4-0.27 beta/0.28 alpha";
    bDe[illegible]1;
  }
  els[illegible]compareEP("BE........AD50FF7634EB7C4801"))
```

```
0
```

```
TypeError: Result of expression 'PE.compareEp' [undefined] is not a function.
```

It appears that the problem in the script. In order to accurately locate the line of the error, click "Debug" button—it opens the debugger. Then press "F5" on the keyboard: debugger starts and we will stop at the line with the error:



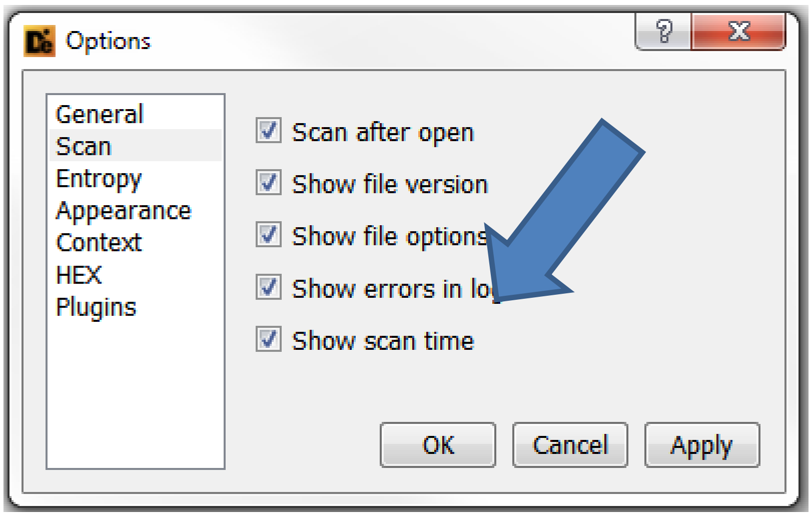
```
 1  // DIE's signature file
 2
 3  init("packer","(Win)Upack");
 4
 5  function detect(bShowType,bShowVersion,bShowOptions)
 6  {
 7    if(PE.compareEp("BE........AD8BF895A533C033"))
 8    {
 9      sVersion="0.1x/0.20/0.21/0.24 beta";
10      bDetected=1;
11    }
12    else if(PE.compareEP("BE........AD8BF86A0495A533C0AB"))
13    {
14      sVersion="0.21 beta";
15      bDetected=1;
16    }
17    else if(PE.compareEP("BE........AD8BF895AD91F3A5ADB5"))
18    {
19      sVersion="0.24-0.27 beta/0.28 alpha";
20      bDetected=1;
21    }
22    else if(PE.compareEP("BE........AD50FF7634EB7C4801"))
23    {
24      if(PE.compareEP("37",62))
25      {
26        sVersion="0.37 beta";
27      }
28      else if(PE.compareEP("38",62))
29      {
30        sVersion="0.38 beta";
31      }
32      else if(PE.compareEP("39",62))
```

```
Mi 11. Jun 14:17:26 2014 Uncaught exception at <anonymous script, id=62755656>:7: TypeError: Result of expression 'PE.compareEp' [undefined] is not a function.
```

The error appears to be in line 7. We change compareEp to compareEP and the error message disappears.

DIE script language is based on Java Script and therefore names of variables and functions are sensitive to case: compareEp differs from compareEP. This is different i.e. from Pascal which is not case-sensitive.

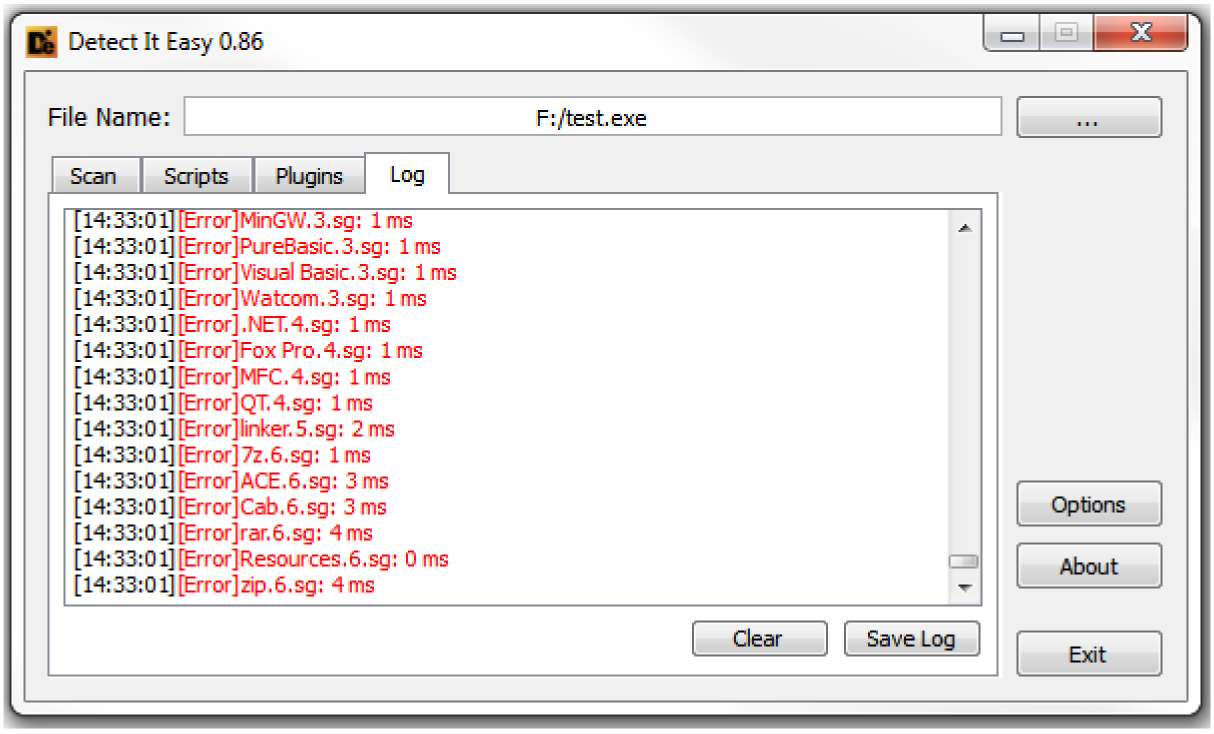
In some cases the time of scanning file is important, so it might be useful to optimize code of signatures for best performance. To learn the speed of running individual signatures for a specific file, you should adjust the settings to display scanning time and errors:



- [✓] Scan after open
- [✓] Show file version
- [✓] Show file options
- [✓] Show errors in log
- [✓] Show scan time

From now on scanning time for each signatures will be recorded i the log separately.

DIE at the current moment has not reached final version (1.0) yet, therefore it is actively improving. If you would like to share new ideas on improving functionality, please write to horsicq@gmail.com



File Name: F:/test.exe

```
[14:33:01][Error]MinGW.3.sg: 1 ms
[14:33:01][Error]PureBasic.3.sg: 1 ms
[14:33:01][Error]Visual Basic.3.sg: 1 ms
[14:33:01][Error]Watcom.3.sg: 1 ms
[14:33:01][Error].NET.4.sg: 1 ms
[14:33:01][Error]Fox Pro.4.sg: 1 ms
[14:33:01][Error]MFC.4.sg: 1 ms
[14:33:01][Error]QT.4.sg: 1 ms
[14:33:01][Error]linker.5.sg: 2 ms
[14:33:01][Error]7z.6.sg: 1 ms
[14:33:01][Error]ACE.6.sg: 3 ms
[14:33:01][Error]Cab.6.sg: 3 ms
[14:33:01][Error]rar.6.sg: 4 ms
[14:33:01][Error]Resources.6.sg: 0 ms
[14:33:01][Error]zip.6.sg: 4 ms
```

DIE at the current moment has not reached final version (1.0) yet, therefore it is actively improving. If you would like to share new ideas on improving functionality, please write to horsicq@gmail.com.