Maldive

Dynamic Malware Analysis Tool

Design Manual

AUTHOR
Shane Doherty

Mon Apr 17 2023

Contents

Maldive i
Dynamic Malware Analysis Tool i
AUTHOR i
Shane Doherty i
Mon Apr 17 2023 i
Dynamic Malware Analysis Tool 6
South East Technological University 6
Name: Shane Doherty 6
Student Number: C00249279 6
Supervisor: Joseph Kehoe 6
Hierarchical Index 7
Class Hierarchy 7
Class Index 8
Class List 8
File Index 9
File List 9
Class Documentation 11
dynamicAnalysis.ActiveConnection Class Reference 11
Public Member Functions 11
Detailed Description 11
Constructor & Destructor Documentation 11
Member Function Documentation 11
dynamicAnalysis.CandidateWindow Class Reference 14
Public Member Functions 14
Static Public Member Functions 14
Protected Member Functions 14
Protected Attributes 14
Detailed Description 14
Member Function Documentation 14
Member Data Documentation 14
CapstoneTest Class Reference 16
Public Member Functions 16
Detailed Description 16
Member Function Documentation 16
dynamicAnalysis.CodeExtract Class Reference 17
Public Member Functions 17
Detailed Description 17
Constructor & Destructor Documentation 17
Member Function Documentation 17
dynamicAnalysis.CommandLine Class Reference 20
Public Member Functions 20
Detailed Description 20
Constructor & Destructor Documentation 20
Member Function Documentation 20
dynamicAnalysis.DataDirectory Class Reference 22
Public Member Functions 22
Detailed Description 22
Constructor & Destructor Documentation 22
Member Function Documentation 22
dynamicAnalysis.Details Class Reference 24
Public Member Functions 24
Protected Member Functions 24
Detailed Description 24
Constructor & Destructor Documentation 24
Member Function Documentation 24
dynamicAnalysis.DllFile Class Reference 26

Public Member Functions 26
Detailed Description 26
Constructor & Destructor Documentation 26
Member Function Documentation 26
dynamicAnalysis.ExecuteCode Class Reference 28
Public Member Functions 28
Detailed Description 28
Constructor & Destructor Documentation 28
Member Function Documentation 28
dynamicAnalysis.FilesComposite Class Reference 31
Public Member Functions 31
Protected Member Functions 31
Detailed Description 31
Constructor & Destructor Documentation 31
Member Function Documentation 31
dynamicAnalysis.InstructionsComposite Class Reference 33
Public Member Functions 33
Protected Member Functions 33
Detailed Description 33
Constructor & Destructor Documentation 33
Member Function Documentation 33
dynamicAnalysis.LegacyWindow Class Reference 35
Public Member Functions 35
Static Public Member Functions 35
Static Public Attributes 35
Protected Member Functions 35
Protected Attributes 35
Detailed Description 35
Member Function Documentation 35
Member Data Documentation 36
dynamicAnalysis.MemoryComposite Class Reference 37
Public Member Functions 37
Protected Member Functions 37
Detailed Description 37
Constructor & Destructor Documentation 37
Member Function Documentation 37
dynamicAnalysis.MemoryWindow Class Reference 39
Public Member Functions 39
Protected Member Functions 39
Protected Attributes 39
Detailed Description 39
Constructor & Destructor Documentation 39
Member Function Documentation 39
Member Data Documentation 41
dynamicAnalysis.Mnem Enum Reference 42
Public Member Functions 42
Public Attributes 42
Detailed Description 42
Member Function Documentation 42
Member Data Documentation 42
dynamicAnalysis.NetworkComposite Class Reference 44
Public Member Functions 44
Protected Member Functions 44
Detailed Description 44
Constructor & Destructor Documentation 44
Member Function Documentation 44
dynamicAnalysis.NetworkStats Class Reference 45
Public Member Functions 45
Detailed Description 45
Constructor & Destructor Documentation 45

Member Function Documentation 45
dynamicAnalysis.NetworkTraffic Class Reference 46
Static Public Member Functions 46
Detailed Description 46
Member Function Documentation 46
dynamicAnalysis.PacketTrace Class Reference 47
Public Member Functions 47
Detailed Description 47
Constructor & Destructor Documentation 47
Member Function Documentation 47
dynamicAnalysis.PEFile Class Reference 49
Public Member Functions 49
Detailed Description 49
Constructor & Destructor Documentation 49
Member Function Documentation 49
dynamicAnalysis.ProcessManager Class Reference 52
Public Member Functions 52
Detailed Description 52
Constructor & Destructor Documentation 52
Member Function Documentation 52
dynamicAnalysis.ReadWrite Class Reference 55
Static Public Member Functions 55
Detailed Description 55
Member Function Documentation 55
dynamicAnalysis.SelectFile Class Reference 58
Public Member Functions 58
Protected Member Functions 58
Protected Attributes 58
Detailed Description 58
Constructor & Destructor Documentation 58
Member Function Documentation 58
Member Data Documentation 60
dynamicAnalysis.SelectProcess Class Reference 61
Public Member Functions 61
Protected Member Functions 61
Protected Attributes 61
Detailed Description 61
Constructor & Destructor Documentation 61
Member Function Documentation 61
Member Data Documentation 63
org.eclipse.wb.swt.SWTResourceManager Class Reference 64
Static Public Member Functions 64
Static Public Attributes 64
Static Protected Member Functions 64
Static Protected Attributes 64
Detailed Description 64
Member Function Documentation 64
Member Data Documentation 68
dynamicAnalysis.test Class Reference 69
Public Member Functions 69
Static Public Member Functions 69
Protected Member Functions 69
Protected Attributes 69
Detailed Description 69
Member Function Documentation 69
Member Data Documentation 69
dynamicAnalysis.Version Enum Reference 71
Public Member Functions 71
Public Attributes 71
Detailed Description 71

Member Function Documentation 71
Member Data Documentation 71
dynamicAnalysis.VirtualMemory Class Reference 72
Public Member Functions 72
Detailed Description 72
Constructor & Destructor Documentation 72
Member Function Documentation 72
dynamicAnalysis.Window Class Reference 74
Public Member Functions 74
Static Public Member Functions 74
Static Public Attributes 74
Protected Member Functions 74
Protected Attributes 74
Detailed Description 74
Member Function Documentation 74
Member Data Documentation 74
File Documentation 76
dynamicAnalysis_ExecuteCode.h 76
dynamicAnalysis_ExecuteCode.h 77
dynamicAnalysis_ExecuteCode.h 78
dynamicAnalysis_VirtualMemory.h 79
dynamicAnalysis_VirtualMemory.h 80
dynamicAnalysis_VirtualMemory.h 81
ExecuteImpl.c 82
ExecuteImpl.c 83
ExecuteImpl.c 84
ReadProcess.cpp 85
ReadProcess.cpp 86
ReadProcess.cpp 87
VirtualMemory.cpp 88
VirtualMemory.cpp 90
VirtualMemory.cpp 92
ActiveConnection.java 94
CandidateWindow.java 96
CapstoneTest.java 100
CodeExtract.java 102
CommandLine.java 105
DataDirectory.java 107
Details.java 108
DllFile.java 110
ExecuteCode.java 111
FilesComposite.java 113
InstructionsComposite.java 115
LegacyWindow.java 121
MemoryComposite.java 126
MemoryWindow.java 132
Mnem.java 138
NetworkComposite.java 139
NetworkStats.java 145
NetworkTraffic.java 147
PacketTrace.java 149
PEFile.java 151
ProcessManager.java 154
ReadWrite.java 157
SelectFile.java 162
SelectProcess.java 166
test.java 171
Version.java 173
VirtualMemory.java 174
Window.java 175

SWTResourceManager.java 182
Index ..... **Error! Bookmark not defined.**

Dynamic Malware Analysis Tool

South East Technological University

Name: Shane Doherty

Student Number: C00249279

Supervisor: Joseph Kehoe

Hierarchical Index

Class Hierarchy

This inheritance list is sorted roughly, but not completely, alphabetically:

- dynamicAnalysis.ActiveConnection 11
- dynamicAnalysis.CandidateWindow 14
- dynamicAnalysis.CodeExtract 17
- dynamicAnalysis.CommandLine 20
- dynamicAnalysis.DataDirectory 22
- dynamicAnalysis.DllFile 26
- dynamicAnalysis.ExecuteCode 28
- dynamicAnalysis.LegacyWindow 35
- dynamicAnalysis.MemoryWindow 39
- dynamicAnalysis.Mnem 42
- dynamicAnalysis.NetworkStats 45
- dynamicAnalysis.NetworkTraffic 46
- dynamicAnalysis.PacketTrace 47
- dynamicAnalysis.PEFile 49
- dynamicAnalysis.ProcessManager 52
- dynamicAnalysis.ReadWrite 55
- dynamicAnalysis.SelectFile 58
- dynamicAnalysis.SelectProcess 61
- org.eclipse.wb.swt.SWTResourceManager 64
- dynamicAnalysis.test 69
- dynamicAnalysis.Version 71
- dynamicAnalysis.VirtualMemory 72
- dynamicAnalysis.Window 74
- Composite
  - dynamicAnalysis.Details 24
  - dynamicAnalysis.FilesComposite 31
  - dynamicAnalysis.InstructionsComposite 33
  - dynamicAnalysis.MemoryComposite 37
  - dynamicAnalysis.NetworkComposite 44
- ProxySelector
  - CapstoneTest 16

Class Index

Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

dynamicAnalysis.ActiveConnection 11
dynamicAnalysis.CandidateWindow 14
CapstoneTest 16
dynamicAnalysis.CodeExtract 17
dynamicAnalysis.CommandLine 20
dynamicAnalysis.DataDirectory 22
dynamicAnalysis.Details 24
dynamicAnalysis.DllFile 26
dynamicAnalysis.ExecuteCode 28
dynamicAnalysis.FilesComposite 31
dynamicAnalysis.InstructionsComposite 33
dynamicAnalysis.LegacyWindow 35
dynamicAnalysis.MemoryComposite 37
dynamicAnalysis.MemoryWindow 39
dynamicAnalysis.Mnem 42
dynamicAnalysis.NetworkComposite 44
dynamicAnalysis.NetworkStats 45
dynamicAnalysis.NetworkTraffic 46
dynamicAnalysis.PacketTrace 47
dynamicAnalysis.PEFile 49
dynamicAnalysis.ProcessManager 52
dynamicAnalysis.ReadWrite 55
dynamicAnalysis.SelectFile 58
dynamicAnalysis.SelectProcess 61
org.eclipse.wb.swt.SWTResourceManager 64
dynamicAnalysis.test 69
dynamicAnalysis.Version 71
dynamicAnalysis.VirtualMemory 72
dynamicAnalysis.Window 74

File Index

File List

Here is a list of all documented files with brief descriptions:

bin/dynamicAnalysis/dynamicAnalysis_ExecuteCode.h 76
bin/dynamicAnalysis/dynamicAnalysis_VirtualMemory.h 79
bin/dynamicAnalysis/ExecuteImpl.c 82
bin/dynamicAnalysis/ReadProcess.cpp 85
bin/dynamicAnalysis/VirtualMemory.cpp 88
src/dynamicAnalysis/ActiveConnection.java 94
src/dynamicAnalysis/CandidateWindow.java 96
src/dynamicAnalysis/CapstoneTest.java 100
src/dynamicAnalysis/CodeExtract.java 102
src/dynamicAnalysis/CommandLine.java 105
src/dynamicAnalysis/DataDirectory.java 107
src/dynamicAnalysis/Details.java 108
src/dynamicAnalysis/DllFile.java 110
src/dynamicAnalysis/dynamicAnalysis_ExecuteCode.h 77
src/dynamicAnalysis/dynamicAnalysis_VirtualMemory.h 80
src/dynamicAnalysis/ExecuteCode.java 111
src/dynamicAnalysis/ExecuteImpl.c 83
src/dynamicAnalysis/FilesComposite.java 113
src/dynamicAnalysis/InstructionsComposite.java 115
src/dynamicAnalysis/LegacyWindow.java 121
src/dynamicAnalysis/MemoryComposite.java 126
src/dynamicAnalysis/MemoryWindow.java 132
src/dynamicAnalysis/Mnem.java 138
src/dynamicAnalysis/NetworkComposite.java 139
src/dynamicAnalysis/NetworkStats.java 145
src/dynamicAnalysis/NetworkTraffic.java 147
src/dynamicAnalysis/PacketTrace.java 149
src/dynamicAnalysis/PEFile.java 151
src/dynamicAnalysis/ProcessManager.java 154
src/dynamicAnalysis/ReadProcess.cpp 86
src/dynamicAnalysis/ReadWrite.java 157
src/dynamicAnalysis/SelectFile.java 162
src/dynamicAnalysis/SelectProcess.java 166
src/dynamicAnalysis/test.java 171
src/dynamicAnalysis/Version.java 173
src/dynamicAnalysis/VirtualMemory.cpp 90
src/dynamicAnalysis/VirtualMemory.java 174
src/dynamicAnalysis/Window.java 175
src/org/eclipse/wb/swt/SWTResourceManager.java 182
target/classes/dynamicAnalysis/dynamicAnalysis_ExecuteCode.h 78
target/classes/dynamicAnalysis/dynamicAnalysis_VirtualMemory.h 81
target/classes/dynamicAnalysis/ExecuteImpl.c 84
target/classes/dynamicAnalysis/ReadProcess.cpp 87

target/classes/dynamicAnalysis/VirtualMemory.cpp 92

Class Documentation

dynamicAnalysis.ActiveConnection Class Reference

Public Member Functions

ActiveConnection (String protocol, String localAddress, String foreignAddress, String state, long pid)
String **getProtocol** ()
void **setProtocol** (String protocol)
String **getLocalAddress** ()
void **setLocalAddress** (String localAddress)
String **getForeignAddress** ()
void **setForeignAddress** (String foreignAddress)
String **getState** ()
void **setState** (String state)
long **getPid** ()
void **setPid** (long pid)
String **toString** ()

Detailed Description

Information relating to the sent packet. Contains information that can be used to determine what the packet contains.

Definition at line **6** of file **ActiveConnection.java**.

Constructor & Destructor Documentation

dynamicAnalysis.ActiveConnection.ActiveConnection (String *protocol*, String *localAddress*, String *foreignAddress*, String *state*, long *pid*)

Instantiates a new active connection.

Parameters

protocol	the protocol of the packet
localAddress	the local address of the packet
foreignAddress	the foreign address of the packet
state	the state of the packet
pid	the pid that the packet was sent from

Definition at line **33** of file **ActiveConnection.java**.

Member Function Documentation

String dynamicAnalysis.ActiveConnection.getForeignAddress ()

Gets the foreign address.

Returns

the foreign address

Definition at line **87** of file **ActiveConnection.java**.

String dynamicAnalysis.ActiveConnection.getLocalAddress ()

Gets the local address.

Returns

the local address

Definition at line **67** of file **ActiveConnection.java**.

long dynamicAnalysis.ActiveConnection.getPid ()

Gets the pid.

Returns

the pid

Definition at line **127** of file **ActiveConnection.java**.

String dynamicAnalysis.ActiveConnection.getProtocol ()

Gets the protocol.

Returns

the protocol

Definition at line **47** of file **ActiveConnection.java**.

String dynamicAnalysis.ActiveConnection.getState ()

Gets the state.

Returns

the state

Definition at line **107** of file **ActiveConnection.java**.

void dynamicAnalysis.ActiveConnection.setForeignAddress (String *foreignAddress*)

Sets the foreign address.

Parameters

foreignAddress	the new foreign address

Definition at line **97** of file **ActiveConnection.java**.

void dynamicAnalysis.ActiveConnection.setLocalAddress (String *localAddress*)

Sets the local address.

Parameters

localAddress	the new local address

Definition at line **77** of file **ActiveConnection.java**.

void dynamicAnalysis.ActiveConnection.setPid (long *pid*)

Sets the pid.

Parameters

pid	the new pid

Definition at line **137** of file **ActiveConnection.java**.

void dynamicAnalysis.ActiveConnection.setProtocol (String *protocol*)

Sets the protocol.

Parameters

protocol	the new protocol

Definition at line **57** of file **ActiveConnection.java**.

void dynamicAnalysis.ActiveConnection.setState (String *state*)

Sets the state.

Parameters

state	the new state

Definition at line **117** of file **ActiveConnection.java**.

String dynamicAnalysis.ActiveConnection.toString ()

To string.

Returns

the full information of the packet

Definition at line **148** of file **ActiveConnection.java**.

The documentation for this class was generated from the following file:

src/dynamicAnalysis/ActiveConnection.java

dynamicAnalysis.CandidateWindow Class Reference

Public Member Functions

void **open** ()

Static Public Member Functions

static void **main** (String[] args)

Protected Member Functions

void **createContents** ()

Protected Attributes

Shell **shell**

Detailed Description

The Class **CandidateWindow**.

Definition at line **35** of file **CandidateWindow.java**.

Member Function Documentation

void dynamicAnalysis.CandidateWindow.createContents () `[protected]`

Create contents of the window.

Definition at line **85** of file **CandidateWindow.java**.

static void dynamicAnalysis.CandidateWindow.main (String[] *args*) `[static]`

Launch the application.

Parameters

args	the arguments

Definition at line **52** of file **CandidateWindow.java**.

void dynamicAnalysis.CandidateWindow.open ()

Open the window.

Definition at line **67** of file **CandidateWindow.java**.

Member Data Documentation

Shell dynamicAnalysis.CandidateWindow.shell `[protected]`

The shell.

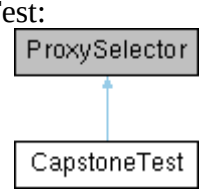
Definition at line **39** of file **CandidateWindow.java**.

The documentation for this class was generated from the following file:

src/dynamicAnalysis/CandidateWindow.java

CapstoneTest Class Reference

Inheritance diagram for CapstoneTest:



Public Member Functions

void **PrivateDataProxy** ()
List< Proxy > **select** (URI uri)
void **connectFailed** (URI arg0, SocketAddress arg1, IOException arg2)

Detailed Description

The Class **CapstoneTest**.

Definition at line **23** of file **CapstoneTest.java**.

Member Function Documentation

void CapstoneTest.connectFailed (URI *arg0*, SocketAddress *arg1*, IOException *arg2*)

Connect failed.

Parameters

arg0	the arg 0
arg1	the arg 1
arg2	the arg 2

Definition at line **91** of file **CapstoneTest.java**.

void CapstoneTest.PrivateDataProxy ()

Private data proxy.

Definition at line **39** of file **CapstoneTest.java**.

List< Proxy > CapstoneTest.select (URI *uri*)

Select.

Parameters

uri	the uri

Returns

the list

Definition at line **69** of file **CapstoneTest.java**.

The documentation for this class was generated from the following file:

src/dynamicAnalysis/CapstoneTest.java

dynamicAnalysis.CodeExtract Class Reference

Public Member Functions

CodeExtract (File file)
byte[] **loadInstructions** ()
File **getFile** ()
void **setFile** (File file)
byte[] **getInstructions** ()
String **getCode** ()
String[] **getCodeArr** ()
PEFile getPeFile ()
void **setPeFile** (**PEFile** peFile)
int **getPointer** ()
Capstone.CsInsn[] **getAllInsn** ()
void **setAllInsn** (Capstone.CsInsn[] allInsn)
byte[] **getBytes** () throws IOException
Version getVersion ()
String **toString** ()

Detailed Description

Top level extraction of code from a PE file. Uses Capstone to translate a byte stream to x86 instructions.

Definition at line **15** of file **CodeExtract.java**.

Constructor & Destructor Documentation

dynamicAnalysis.CodeExtract.CodeExtract (File *file*)

Instantiates the **CodeExtract** class.

Parameters

file	the file to extract the x86 instructions from

Definition at line **40** of file **CodeExtract.java**.

Member Function Documentation

Capstone.CsInsn[] dynamicAnalysis.CodeExtract.getAllInsn ()

Gets the Capstone instruction set.

Returns

the Capstone instruction set

Definition at line **221** of file **CodeExtract.java**.

byte[] dynamicAnalysis.CodeExtract.getBytes () throws IOException

Gets the raw instruction bytes.

Returns

the raw instruction bytes

Exceptions

IOException	Signals that an I/O exception has occurred.

Definition at line **242** of file **CodeExtract.java**.

String dynamicAnalysis.CodeExtract.getCode ()

Gets the code.

Returns

the code

Definition at line **173** of file **CodeExtract.java**.

String[] dynamicAnalysis.CodeExtract.getCodeArr ()

Gets the code arr.

Returns

the code arr

Definition at line **182** of file **CodeExtract.java**.

File dynamicAnalysis.CodeExtract.getFile ()

Gets the file.

Returns

the file

Definition at line **137** of file **CodeExtract.java**.

byte[] dynamicAnalysis.CodeExtract.getInstructions ()

Gets the instructions.

Returns

the instructions

Definition at line **164** of file **CodeExtract.java**.

PEFile dynamicAnalysis.CodeExtract.getPeFile ()

Gets the pe file.

Returns

the pe file

Definition at line **191** of file **CodeExtract.java**.

int dynamicAnalysis.CodeExtract.getPointer ()

Gets the pointer.

Returns

the pointer

Definition at line **211** of file **CodeExtract.java**.

Version dynamicAnalysis.CodeExtract.getVersion ()

Gets the PE version, either 32 bit or 64 bit.

Returns

the PE version

Definition at line **252** of file **CodeExtract.java**.

byte[] dynamicAnalysis.CodeExtract.loadInstructions ()

Load x86 instructions.

Returns

byte array containing raw instruction bytes

Definition at line **62** of file **CodeExtract.java**.

void dynamicAnalysis.CodeExtract.setAllInsn (Capstone.CsInsn[] *allInsn*)

Sets the Capstone instruction set.

Parameters

allInsn	the new Capstone instruction set.

Definition at line **231** of file **CodeExtract.java**.

void dynamicAnalysis.CodeExtract.setFile (File *file*)

Sets the file.

Parameters

file	the new file

Definition at line **146** of file **CodeExtract.java**.

void dynamicAnalysis.CodeExtract.setPeFile (PEFile *peFile*)

Sets the pe file.

Parameters

peFile	the new pe file

Definition at line **201** of file **CodeExtract.java**.

String dynamicAnalysis.CodeExtract.toString ()

To string.

Returns

the string

Definition at line **263** of file **CodeExtract.java**.

The documentation for this class was generated from the following file:

src/dynamicAnalysis/CodeExtract.java

dynamicAnalysis.CommandLine Class Reference

Public Member Functions

CommandLine (long pid)
CommandLine ()
long **getPid** ()
void **setPid** (long pid)
String **runName** ()
String **runDLLs** ()
String **runFiles** ()
String **getAll** ()
String **getNetstat** ()
String **toString** ()

Detailed Description

The Class **CommandLine**. Used to communicate with the command line whenever it is required.

Definition at line **13** of file **CommandLine.java**.

Constructor & Destructor Documentation

dynamicAnalysis.CommandLine.CommandLine (long *pid*)

Instantiates a new command line.

Parameters

pid	the process ID

Definition at line **24** of file **CommandLine.java**.

dynamicAnalysis.CommandLine.CommandLine ()

Instantiates a new command line.

Definition at line **32** of file **CommandLine.java**.

Member Function Documentation

String dynamicAnalysis.CommandLine.getAll ()

Gets all processes currently running.

Returns

the cmd result from running the command

Definition at line **129** of file **CommandLine.java**.

String dynamicAnalysis.CommandLine.getNetstat ()

Gets the network information from all processes.

Returns

the cmd result from running the command

Definition at line **139** of file **CommandLine.java**.

long dynamicAnalysis.CommandLine.getPid ()

Gets the pid.

Returns

the pid

Definition at line **42** of file **CommandLine.java**.

String dynamicAnalysis.CommandLine.runDLLs ()

Lists DLLS used by the process. Credit to Mark Russinovich of Microsoft for the utility.

Returns

the cmd result from running the command

Definition at line **109** of file **CommandLine.java**.

String dynamicAnalysis.CommandLine.runFiles ()

Run files.

Returns

the cmd result from running the command

Definition at line **119** of file **CommandLine.java**.

String dynamicAnalysis.CommandLine.runName ()

Uses tasklist to get relevant process information from the process ID.

Returns

the cmd result from running the command

Definition at line **99** of file **CommandLine.java**.

void dynamicAnalysis.CommandLine.setPid (long *pid*)

Sets the pid.

Parameters

pid	the new pid

Definition at line **52** of file **CommandLine.java**.

String dynamicAnalysis.CommandLine.toString ()

To string.

Returns

the string

Definition at line **150** of file **CommandLine.java**.

The documentation for this class was generated from the following file:

src/dynamicAnalysis/CommandLine.java

dynamicAnalysis.DataDirectory Class Reference

Public Member Functions

DataDirectory (byte[] bytes, int virtualAddress, int size)
byte[] **getBytes** ()
void **setBytes** (byte[] bytes)
int **getVirtualAddress** ()
void **setVirtualAddress** (int virtualAddress)
int **getSize** ()
void **setSize** (int size)

Detailed Description

The Class **DataDirectory**.

Definition at line **9** of file **DataDirectory.java**.

Constructor & Destructor Documentation

dynamicAnalysis.DataDirectory.DataDirectory (byte[] *bytes*, int *virtualAddress*, int *size*)

Instantiates a new data directory.

Parameters

bytes	the bytes
virtualAddress	the virtual address
size	the size

Definition at line **28** of file **DataDirectory.java**.

Member Function Documentation

byte[] dynamicAnalysis.DataDirectory.getBytes ()

Gets the bytes.

Returns

the bytes

Definition at line **40** of file **DataDirectory.java**.

int dynamicAnalysis.DataDirectory.getSize ()

Gets the size.

Returns

the size

Definition at line **80** of file **DataDirectory.java**.

int dynamicAnalysis.DataDirectory.getVirtualAddress ()

Gets the virtual address.

Returns

the virtual address

Definition at line **60** of file **DataDirectory.java**.

void dynamicAnalysis.DataDirectory.setBytes (byte[] *bytes*)

Sets the bytes.

Parameters

bytes	the new bytes

Definition at line **50** of file **DataDirectory.java**.

void dynamicAnalysis.DataDirectory.setSize (int *size*)

Sets the size.

Parameters

size	the new size

Definition at line **90** of file **DataDirectory.java**.

void dynamicAnalysis.DataDirectory.setVirtualAddress (int *virtualAddress*)

Sets the virtual address.

Parameters

virtualAddress	the new virtual address

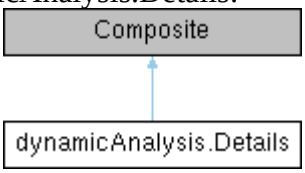
Definition at line **70** of file **DataDirectory.java**.

The documentation for this class was generated from the following file:

src/dynamicAnalysis/DataDirectory.java

dynamicAnalysis.Details Class Reference

Inheritance diagram for dynamicAnalysis.Details:



Public Member Functions

Details (Composite parent, int style, boolean selection)
void **clearData** ()
boolean **isSelection** ()
void **setSelection** (boolean selection)

Protected Member Functions

void **checkSubclass** ()

Detailed Description

Unused implementation of a composite to show the details of a process. This is included as a demonstration only.

Definition at line **20** of file **Details.java**.

Constructor & Destructor Documentation

dynamicAnalysis.Details.Details (Composite *parent*, int *style*, boolean *selection*)

Create the details composite.

Parameters

parent	the main window that acts as the parent
style	the SWT style applied to the composite
selection	the selection

Definition at line **48** of file **Details.java**.

Member Function Documentation

void dynamicAnalysis.Details.checkSubclass () `[protected]`

Check subclass.

Definition at line **158** of file **Details.java**.

void dynamicAnalysis.Details.clearData ()

Clear data from the GUI table.

Definition at line **121** of file **Details.java**.

boolean dynamicAnalysis.Details.isSelection ()

Checks if the selection is toggled.

Returns

the selection value

Definition at line **139** of file **Details.java**.

void dynamicAnalysis.Details.setSelection (boolean *selection*)

Sets the selection toggle.

Parameters

selection	the new selection value

Definition at line **149** of file **Details.java**.

The documentation for this class was generated from the following file:

src/dynamicAnalysis/Details.java

dynamicAnalysis.DllFile Class Reference

Public Member Functions

DllFile (String path)
String **getPath** ()
void **setPath** (String path)
File **getFile** ()
String **toString** ()

Detailed Description

Class used to identify DLL files.

Definition at line **11** of file **DllFile.java**.

Constructor & Destructor Documentation

dynamicAnalysis.DllFile.DllFile (String *path*)

Instantiates a new dll file.

Parameters

path	the path of the DLL file

Definition at line **25** of file **DllFile.java**.

Member Function Documentation

File dynamicAnalysis.DllFile.getFile ()

Gets the file.

Returns

the file

Definition at line **56** of file **DllFile.java**.

String dynamicAnalysis.DllFile.getPath ()

Gets the path of the DLL file.

Returns

the path of the DLL file

Definition at line **36** of file **DllFile.java**.

void dynamicAnalysis.DllFile.setPath (String *path*)

Sets the path.

Parameters

path	the new path

Definition at line **46** of file **DllFile.java**.

String dynamicAnalysis.DllFile.toString ()

To string.

Returns

the string

Definition at line **77** of file **DllFile.java**.

The documentation for this class was generated from the following file:

src/dynamicAnalysis/DllFile.java

dynamicAnalysis.ExecuteCode Class Reference

Public Member Functions

ExecuteCode (byte[] codes, File file)
ExecuteCode (byte code, File file)
File **getFile** ()
void **setFile** (File file)
byte[] **getCodes** ()
void **setCodes** (byte[] codes)
byte **getCode** ()
void **setCode** (byte code)
native String **test** ()
int[] **read** ()
String **toString** ()

Detailed Description

Unused implementation that would have allowed for registry view. Included for demonstration purposes only

Definition at line **12** of file **ExecuteCode.java**.

Constructor & Destructor Documentation

dynamicAnalysis.ExecuteCode.ExecuteCode (byte[] *codes*, File *file*)

Instantiates the **ExecuteCode** method with an array

Parameters

codes	the byte array of registry values
file	the file to be accessed

Definition at line **48** of file **ExecuteCode.java**.

dynamicAnalysis.ExecuteCode.ExecuteCode (byte *code*, File *file*)

Instantiates the **ExecuteCode** method with single instruction

Parameters

codes	the byte containing a registry value
file	the file to be accessed

Definition at line **60** of file **ExecuteCode.java**.

Member Function Documentation

byte dynamicAnalysis.ExecuteCode.getCode ()

Gets a single registry code.

Returns

the registry code

Definition at line **109** of file **ExecuteCode.java**.

byte[] dynamicAnalysis.ExecuteCode.getCodes ()

Gets the registry codes.

Returns

the registry codes

Definition at line **91** of file **ExecuteCode.java**.

File dynamicAnalysis.ExecuteCode.getFile ()

Gets the file.

Returns

the file

Definition at line **71** of file **ExecuteCode.java**.

int[] dynamicAnalysis.ExecuteCode.read ()

Read the current registry values.

Returns

the array containing the four registry values

Definition at line **151** of file **ExecuteCode.java**.

void dynamicAnalysis.ExecuteCode.setCode (byte *code*)

Sets the registry code.

Parameters

code	the new registry code

Definition at line **118** of file **ExecuteCode.java**.

void dynamicAnalysis.ExecuteCode.setCodes (byte[] *codes*)

Sets the registry codes.

Parameters

codes	the new registry codes

Definition at line **100** of file **ExecuteCode.java**.

void dynamicAnalysis.ExecuteCode.setFile (File *file*)

Sets the file.

Parameters

file	the new file

Definition at line **81** of file **ExecuteCode.java**.

native String dynamicAnalysis.ExecuteCode.test ()

Test.

Returns

the string

String dynamicAnalysis.ExecuteCode.toString ()

To string.

Returns

the string describing values being used

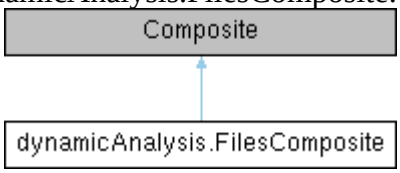
Definition at line **165** of file **ExecuteCode.java**.

The documentation for this class was generated from the following file:

src/dynamicAnalysis/ExecuteCode.java

dynamicAnalysis.FilesComposite Class Reference

Inheritance diagram for dynamicAnalysis.FilesComposite:



Public Member Functions

FilesComposite (Composite parent, int style)
int **getProcessId** ()
void **setProcessId** (int processId)

Protected Member Functions

void **checkSubclass** ()

Detailed Description

Composite to display the files used by the process

Definition at line **17** of file **FilesComposite.java**.

Constructor & Destructor Documentation

dynamicAnalysis.FilesComposite.FilesComposite (Composite *parent*, int *style*)

Create the composite.

Parameters

parent	the parent that contains this composite
style	the SWT style

Definition at line **35** of file **FilesComposite.java**.

Member Function Documentation

void dynamicAnalysis.FilesComposite.checkSubclass () `[protected]`

Check subclass.

Definition at line **97** of file **FilesComposite.java**.

int dynamicAnalysis.FilesComposite.getProcessId ()

Gets the process id.

Returns

the process id

Definition at line **78** of file **FilesComposite.java**.

void dynamicAnalysis.FilesComposite.setProcessId (int *processId*)

Sets the process id.

Parameters

processId	the new process id

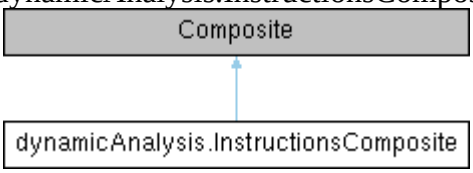
Definition at line **88** of file **FilesComposite.java**.

The documentation for this class was generated from the following file:

src/dynamicAnalysis/FilesComposite.java

dynamicAnalysis.InstructionsComposite Class Reference

Inheritance diagram for dynamicAnalysis.InstructionsComposite:



Public Member Functions

InstructionsComposite (Composite parent, int style, File file)
void **layout** ()
Capstone.CsInsn[] **getAllInsn** ()
void **setAllInsn** (Capstone.CsInsn[] allInsn)

Protected Member Functions

void **checkSubclass** ()

Detailed Description

Composite to display the x86 instructions of a PE file

Definition at line **38** of file **InstructionsComposite.java**.

Constructor & Destructor Documentation

dynamicAnalysis.InstructionsComposite.InstructionsComposite (Composite *parent*, int *style*, File *file*)

Create the instructions composite.

Parameters

parent	the parent that contains the composite
style	the SWT style
file	the file that contains the x86 instructions

Definition at line **75** of file **InstructionsComposite.java**.

Member Function Documentation

void dynamicAnalysis.InstructionsComposite.checkSubclass () `[protected]`

Check subclass.

Definition at line **422** of file **InstructionsComposite.java**.

Capstone.CsInsn[] dynamicAnalysis.InstructionsComposite.getAllInsn ()

Gets the Capstone instruction array.

Returns

the Capstone instruction array

Definition at line **403** of file **InstructionsComposite.java**.

void dynamicAnalysis.InstructionsComposite.layout ()

Layout.

Definition at line **392** of file **InstructionsComposite.java**.

void dynamicAnalysis.InstructionsComposite.setAllInsn (Capstone.CsInsn[] *allInsn*)

Sets the Capstone instruction array.

Parameters

allInsn	the Capstone instruction array

Definition at line **413** of file **InstructionsComposite.java**.

The documentation for this class was generated from the following file:

src/dynamicAnalysis/InstructionsComposite.java

dynamicAnalysis.LegacyWindow Class Reference

Public Member Functions

void **open** ()
void **tone** (int hz, int msecs) throws LineUnavailableException

Static Public Member Functions

static void **main** (String[] args)
static void **tone** (int hz, int msecs, double vol) throws LineUnavailableException

Static Public Attributes

static float **SAMPLE_RATE** = 8000f

Protected Member Functions

void **createContents** ()

Protected Attributes

Shell **shell**

Detailed Description

The initial window used by the program. Now unused, included for demonstration purposes only.

Definition at line **36** of file **LegacyWindow.java**.

Member Function Documentation

void dynamicAnalysis.LegacyWindow.createContents () `[protected]`

Create contents of the window.

Definition at line **136** of file **LegacyWindow.java**.

static void dynamicAnalysis.LegacyWindow.main (String[] *args*) `[static]`

The main method.

Parameters

args	the arguments

Definition at line **61** of file **LegacyWindow.java**.

void dynamicAnalysis.LegacyWindow.open ()

Open.

Definition at line **74** of file **LegacyWindow.java**.

void dynamicAnalysis.LegacyWindow.tone (int *hz*, int *msecs*) throws LineUnavailableException

Tone.

Parameters

hz	the hz

msecs	the msecs

Exceptions

LineUnavailableException	the line unavailable exception

Definition at line **96** of file **LegacyWindow.java**.

static void dynamicAnalysis.LegacyWindow.tone (int *hz*, int *msecs*, double *vol*) throws LineUnavailableException `[static]`

Tone.

Parameters

hz	the hz
msecs	the msecs
vol	the vol

Exceptions

LineUnavailableException	the line unavailable exception

Definition at line **109** of file **LegacyWindow.java**.

Member Data Documentation

float dynamicAnalysis.LegacyWindow.SAMPLE_RATE = 8000f `[static]`

The sample rate.

Definition at line **87** of file **LegacyWindow.java**.

Shell dynamicAnalysis.LegacyWindow.shell `[protected]`

The shell.

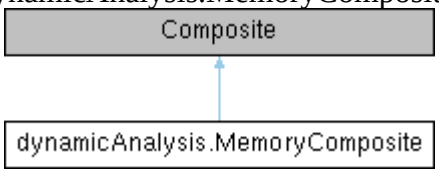
Definition at line **39** of file **LegacyWindow.java**.

The documentation for this class was generated from the following file:

src/dynamicAnalysis/LegacyWindow.java

dynamicAnalysis.MemoryComposite Class Reference

Inheritance diagram for dynamicAnalysis.MemoryComposite:



Public Member Functions

MemoryComposite (Composite parent, int style, Color green)
int **getProcessId** ()
void **setProcessId** (int processId)
byte[] **getBytes** ()
void **setBytes** (byte[] bytes)
Color **getRed** ()
void **setRed** (Color red)

Protected Member Functions

void **checkSubclass** ()

Detailed Description

The composite to display the virtual memory in the GUI

Definition at line **42** of file **MemoryComposite.java**.

Constructor & Destructor Documentation

dynamicAnalysis.MemoryComposite.MemoryComposite (Composite *parent*, int *style*, Color *green*)

Create the memory composite.

Parameters

parent	the main window that acts as the parent
style	the SWT style applied to the composite
green	the SWT color green

Definition at line **85** of file **MemoryComposite.java**.

Member Function Documentation

void dynamicAnalysis.MemoryComposite.checkSubclass () `[protected]`

Check subclass.

Definition at line **448** of file **MemoryComposite.java**.

byte[] dynamicAnalysis.MemoryComposite.getBytes ()

Gets the virtual memory space.

Returns

the virtual memory space byte array

Definition at line **229** of file **MemoryComposite.java**.

int dynamicAnalysis.MemoryComposite.getProcessId ()

Gets the process id.

Returns

the process id

Definition at line **197** of file **MemoryComposite.java**.

Color dynamicAnalysis.MemoryComposite.getRed ()

Gets the SWT color red.

Returns

the SWT color red

Definition at line **249** of file **MemoryComposite.java**.

void dynamicAnalysis.MemoryComposite.setBytes (byte[] *bytes*)

Sets the virtual memory space.

Parameters

bytes	the new virtual memory space.

Definition at line **239** of file **MemoryComposite.java**.

void dynamicAnalysis.MemoryComposite.setProcessId (int *processId*)

Sets the process id.

Parameters

processId	the new process id

Definition at line **207** of file **MemoryComposite.java**.

void dynamicAnalysis.MemoryComposite.setRed (Color *red*)

Sets the SWT color red.

Parameters

red	the new SWT color red

Definition at line **258** of file **MemoryComposite.java**.

The documentation for this class was generated from the following file:

src/dynamicAnalysis/MemoryComposite.java

dynamicAnalysis.MemoryWindow Class Reference

Public Member Functions

MemoryWindow (int processId, int x, int y)
int **getProcessId** ()
void **setProcessId** (int processId)
int **getX** ()
void **setX** (int x)
int **getY** ()
void **setY** (int y)
void **open** ()
byte[] **getBytes** ()
void **setBytes** (byte[] bytes)

Protected Member Functions

void **createBaseContents** ()
synchronized void **createContents** ()

Protected Attributes

Shell **shell**

Detailed Description

The Class **MemoryWindow**.

Definition at line **29** of file **MemoryWindow.java**.

Constructor & Destructor Documentation

dynamicAnalysis.MemoryWindow.MemoryWindow (int *processId*, int *x*, int *y*)

Instantiates a new memory window.

Parameters

processId	the process id
x	the x
y	the y

Definition at line **94** of file **MemoryWindow.java**.

Member Function Documentation

void dynamicAnalysis.MemoryWindow.createBaseContents () `[protected]`

Creates the base contents.

Definition at line **382** of file **MemoryWindow.java**.

synchronized void dynamicAnalysis.MemoryWindow.createContents () `[protected]`

Creates the contents.

Definition at line **454** of file **MemoryWindow.java**.

byte[] dynamicAnalysis.MemoryWindow.getBytes ()

Gets the bytes.

Returns

the bytes

Definition at line **205** of file **MemoryWindow.java**.

int dynamicAnalysis.MemoryWindow.getProcessId ()

Gets the process id.

Returns

the process id

Definition at line **113** of file **MemoryWindow.java**.

int dynamicAnalysis.MemoryWindow.getX ()

Gets the x.

Returns

the x

Definition at line **133** of file **MemoryWindow.java**.

int dynamicAnalysis.MemoryWindow.getY ()

Gets the y.

Returns

the y

Definition at line **153** of file **MemoryWindow.java**.

void dynamicAnalysis.MemoryWindow.open ()

Open.

Definition at line **172** of file **MemoryWindow.java**.

void dynamicAnalysis.MemoryWindow.setBytes (byte[] *bytes*)

Sets the bytes.

Parameters

bytes	the new bytes

Definition at line **216** of file **MemoryWindow.java**.

void dynamicAnalysis.MemoryWindow.setProcessId (int *processId*)

Sets the process id.

Parameters

processId	the new process id

Definition at line **123** of file **MemoryWindow.java**.

void dynamicAnalysis.MemoryWindow.setX (int *x*)

Sets the x.

Parameters

x	the new x

Definition at line **143** of file **MemoryWindow.java**.

void dynamicAnalysis.MemoryWindow.setY (int *y*)

Sets the y.

Parameters

y	the new y

Definition at line **164** of file **MemoryWindow.java**.

Member Data Documentation

Shell dynamicAnalysis.MemoryWindow.shell `[protected]`

The shell.

Definition at line **33** of file **MemoryWindow.java**.

The documentation for this class was generated from the following file:

src/dynamicAnalysis/MemoryWindow.java

dynamicAnalysis.Mnem Enum Reference

Public Member Functions

byte **getByte** ()

Public Attributes

JNE
JMP
JL
JBE
JE
JAE
JB
JA
JLE

Detailed Description

The Enum for mnemonics used by the x86 instruction set.

Definition at line **9** of file **Mnem.java**.

Member Function Documentation

byte dynamicAnalysis.Mnem.getByte ()

Gets the byte value of the jump instruction.

Returns

the byte value of the jump instruction

Definition at line **35** of file **Mnem.java**.

Member Data Documentation

dynamicAnalysis.Mnem.JA

Jump Above.

Definition at line **26** of file **Mnem.java**.

dynamicAnalysis.Mnem.JAE

The jae.

Definition at line **22** of file **Mnem.java**.

dynamicAnalysis.Mnem.JB

Jump Below.

Definition at line **24** of file **Mnem.java**.

dynamicAnalysis.Mnem.JBE

Jump Before or Equal.

Definition at line **18** of file **Mnem.java**.

dynamicAnalysis.Mnem.JE

Jump if Equal.

Definition at line **20** of file **Mnem.java**.

dynamicAnalysis.Mnem.JL

Jump Less than.

Definition at line **16** of file **Mnem.java**.

dynamicAnalysis.Mnem.JLE

Jump if Less or Equal.

Definition at line **28** of file **Mnem.java**.

dynamicAnalysis.Mnem.JMP

Jump.

Definition at line **14** of file **Mnem.java**.

dynamicAnalysis.Mnem.JNE

Jump Not Equal.

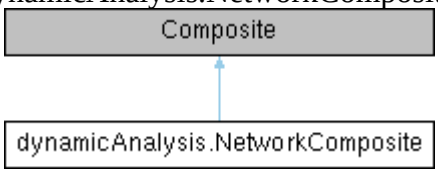
Definition at line **12** of file **Mnem.java**.

The documentation for this enum was generated from the following file:

src/dynamicAnalysis/Mnem.java

dynamicAnalysis.NetworkComposite Class Reference

Inheritance diagram for dynamicAnalysis.NetworkComposite:



Public Member Functions

NetworkComposite (Composite parent, int style, long pid) throws PcapNativeException

Protected Member Functions

void **checkSubclass** ()

Detailed Description

The composite to monitor network traffic

Definition at line **48** of file **NetworkComposite.java**.

Constructor & Destructor Documentation

dynamicAnalysis.NetworkComposite.NetworkComposite (Composite *parent*, int *style*, long *pid*) throws PcapNativeException

Instantiates the network composite.

Parameters

parent	the main window that acts as the parent
style	the SWT style applied to the network composite
pid	the unique process identifier

Exceptions

PcapNativeException	the pcap libraries' native exception

Definition at line **92** of file **NetworkComposite.java**.

Member Function Documentation

void dynamicAnalysis.NetworkComposite.checkSubclass () `[protected]`

Check subclass.

Definition at line **387** of file **NetworkComposite.java**.

The documentation for this class was generated from the following file:

src/dynamicAnalysis/NetworkComposite.java

dynamicAnalysis.NetworkStats Class Reference

Public Member Functions

NetworkStats ()
ActiveConnection[] **getActiveConnections** ()
void **setActiveConnections** (**ActiveConnection**[] activeConnections)

Detailed Description

Retrieves statistics from the network

Definition at line **11** of file **NetworkStats.java**.

Constructor & Destructor Documentation

dynamicAnalysis.NetworkStats.NetworkStats ()

Instantiates a new network stats method.

Definition at line **20** of file **NetworkStats.java**.

Member Function Documentation

ActiveConnection[] dynamicAnalysis.NetworkStats.getActiveConnections ()

Gets the active connections.

Returns

the active connections

Definition at line **86** of file **NetworkStats.java**.

void dynamicAnalysis.NetworkStats.setActiveConnections (ActiveConnection[] *activeConnections*)

Sets the active connections.

Parameters

activeConnections	the new active connections

Definition at line **96** of file **NetworkStats.java**.

The documentation for this class was generated from the following file:

src/dynamicAnalysis/NetworkStats.java

dynamicAnalysis.NetworkTraffic Class Reference

Static Public Member Functions

static void **main** (String[] args)

Detailed Description

The class used for testing purposes relating to network traffic. Unused, included for demonstration purposes only.

Definition at line **26** of file **NetworkTraffic.java**.

Member Function Documentation

static void dynamicAnalysis.NetworkTraffic.main (String[] *args*) `[static]`

The main method.

Parameters

args	the arguments

Definition at line **33** of file **NetworkTraffic.java**.

The documentation for this class was generated from the following file:

src/dynamicAnalysis/NetworkTraffic.java

dynamicAnalysis.PacketTrace Class Reference

Public Member Functions

PacketTrace () throws PcapNativeException
HashMap< String, String > **getDevices** ()
String[] **getAddresses** (String deviceName)
ArrayList< Multimap< String, IpPacket > > **getPackets** ()
ArrayList< IpPacket > **getPackets** (String address, int attempts)

Detailed Description

The Class **PacketTrace**. Uses several maps to tie packets to addresses, and addresses to network interfaces.

Definition at line **30** of file **PacketTrace.java**.

Constructor & Destructor Documentation

dynamicAnalysis.PacketTrace.PacketTrace () throws PcapNativeException

Instantiates a new packet trace class.

Exceptions

PcapNativeException	the Pcap native exception

Definition at line **56** of file **PacketTrace.java**.

Member Function Documentation

String[] dynamicAnalysis.PacketTrace.getAddresses (String *deviceName*)

Gets the string array containing all addresses from a network device.

Parameters

deviceName	the name of the network device

Returns

the addresses from the device

Definition at line **119** of file **PacketTrace.java**.

HashMap< String, String > dynamicAnalysis.PacketTrace.getDevices ()

Gets the hashmap of active devices. Will only contain devices that have packet activity.

Returns

the devices

Definition at line **100** of file **PacketTrace.java**.

ArrayList< Multimap< String, IpPacket > > dynamicAnalysis.PacketTrace.getPackets ()

Gets all packets from all addresses.

Returns

the full list of packets

Definition at line **129** of file **PacketTrace.java**.

ArrayList< IpPacket > dynamicAnalysis.PacketTrace.getPackets (String *address*, int *attempts*)

Gets the packets from a specific address. Has a timeout counter of 10 as it will be recursively called if there is a problem with concurrency.

Parameters

address	the address to retrieve packets from
attempts	the incrementing number of attempts to be recursively incremented

Returns

the packets from the chosen address

Definition at line **141** of file **PacketTrace.java**.

The documentation for this class was generated from the following file:

src/dynamicAnalysis/PacketTrace.java

dynamicAnalysis.PEFile Class Reference

Public Member Functions

PEFile (File file)
File **getFile** ()
void **setFile** (File file)
int **getOffset** ()
Version getVersion ()
byte[] **getInstructions** ()
int **getPointer** ()
byte[] **getBytes** () throws IOException
void **setBytes** (byte[] bytes)
void **readFile** ()
String **toString** ()

Detailed Description

The Class **PEFile**. Retrieves and stores information on a Portable Executable relating to the file format

Definition at line **19** of file **PEFile.java**.

Constructor & Destructor Documentation

dynamicAnalysis.PEFile.PEFile (File *file*)

Instantiates a new PE file.

Parameters

file	the PE file

Definition at line **44** of file **PEFile.java**.

Member Function Documentation

byte[] dynamicAnalysis.PEFile.getBytes () throws IOException

Gets all bytes contained in the PE file.

Returns

the bytes contained in the PE file.

Exceptions

IOException	Signals that the file cannot be read or is not found

Definition at line **142** of file **PEFile.java**.

File dynamicAnalysis.PEFile.getFile ()

Gets the PE file.

Returns

the PE file

Definition at line **54** of file **PEFile.java**.

byte[] dynamicAnalysis.PEFile.getInstructions ()

Gets the raw x86 instruction bytes.

Returns

the raw x86 instruction bytes

Definition at line **94** of file **PEFile.java**.

int dynamicAnalysis.PEFile.getOffset ()

Gets the offset.

Returns

the offset

Definition at line **74** of file **PEFile.java**.

int dynamicAnalysis.PEFile.getPointer ()

Gets the pointer.

Returns

the pointer

Definition at line **121** of file **PEFile.java**.

Version dynamicAnalysis.PEFile.getVersion ()

Gets the PE file's version.

Returns

the version

Definition at line **84** of file **PEFile.java**.

void dynamicAnalysis.PEFile.readFile ()

Read the PE file, populating fields that are stored

Definition at line **160** of file **PEFile.java**.

void dynamicAnalysis.PEFile.setBytes (byte[] *bytes*)

Sets the bytes.

Parameters

bytes	the new bytes

Definition at line **152** of file **PEFile.java**.

void dynamicAnalysis.PEFile.setFile (File *file*)

Sets the PE file.

Parameters

file	the new PE file

Definition at line **64** of file **PEFile.java**.

String dynamicAnalysis.PEFile.toString ()

To string.

Returns

the string containing information on the PE file

Definition at line **199** of file **PEFile.java**.

The documentation for this class was generated from the following file:

src/dynamicAnalysis/PEFile.java

dynamicAnalysis.ProcessManager Class Reference

Public Member Functions

ProcessManager (File file)
ProcessManager (int pid)
File **getFile** ()
void **setFile** (File file)
Process **createProcess** ()
long **getPid** ()
String **getPidAsString** ()
String **getName** ()
String[] **getDLLs** ()
DllFile[] **getDllFiles** ()
String[] **getFiles** ()
Process **getProcess** ()
void **setProcess** (Process process)
boolean **destroyProcess** ()
String **toString** ()

Detailed Description

The Class **ProcessManager**. Manages the created and accessed processes.

Definition at line **13** of file **ProcessManager.java**.

Constructor & Destructor Documentation

dynamicAnalysis.ProcessManager.ProcessManager (File *file*)

Instantiates a new manager for processes when using a file.

Parameters

file	the file to be accessed

Definition at line **42** of file **ProcessManager.java**.

dynamicAnalysis.ProcessManager.ProcessManager (int *pid*)

Instantiates a new manager for processes when using an already running process.

Parameters

pid	the unique identifier for the process

Definition at line **56** of file **ProcessManager.java**.

Member Function Documentation

Process dynamicAnalysis.ProcessManager.createProcess ()

Creates a new process.

Returns

the process

Definition at line **89** of file **ProcessManager.java**.

boolean dynamicAnalysis.ProcessManager.destroyProcess ()

Forcibly destroy process.

Returns

true, if successfully destroyed

Definition at line **285** of file **ProcessManager.java**.

DllFile[] dynamicAnalysis.ProcessManager.getDllFiles ()

Gets the DLL files in use by the process.

Returns

the DLL files in use by the process

Definition at line **208** of file **ProcessManager.java**.

String[] dynamicAnalysis.ProcessManager.getDLLs ()

Gets the string array representation of DLLs in use by the process.

Returns

the string array representation of DLLs in use by the process

Definition at line **198** of file **ProcessManager.java**.

File dynamicAnalysis.ProcessManager.getFile ()

Gets the file being accessed.

Returns

the file being accessed.

Definition at line **69** of file **ProcessManager.java**.

String[] dynamicAnalysis.ProcessManager.getFiles ()

Gets the files in use by the process.

Returns

the files in use by the process

Definition at line **218** of file **ProcessManager.java**.

String dynamicAnalysis.ProcessManager.getName ()

Gets the name of the process.

Returns

the name of the process

Definition at line **188** of file **ProcessManager.java**.

long dynamicAnalysis.ProcessManager.getPid ()

Gets the unique identifier of the process in use.

Returns

the unique identifier of the process in use

Definition at line **111** of file **ProcessManager.java**.

String dynamicAnalysis.ProcessManager.getPidAsString ()

Gets the unique identifier of the process in use as a string.

Returns

the unique identifier of the process in use as a string

Definition at line **121** of file **ProcessManager.java**.

Process dynamicAnalysis.ProcessManager.getProcess ()

Gets the process.

Returns

the process

Definition at line **265** of file **ProcessManager.java**.

void dynamicAnalysis.ProcessManager.setFile (File *file*)

Sets the file being accessed.

Parameters

file	the new file

Definition at line **79** of file **ProcessManager.java**.

void dynamicAnalysis.ProcessManager.setProcess (Process *process*)

Sets the process.

Parameters

process	the new process

Definition at line **275** of file **ProcessManager.java**.

String dynamicAnalysis.ProcessManager.toString ()

To string.

Returns

the string containing values pertaining to the process manager

Definition at line **300** of file **ProcessManager.java**.

The documentation for this class was generated from the following file:

src/dynamicAnalysis/ProcessManager.java

dynamicAnalysis.ReadWrite Class Reference

Static Public Member Functions

static void **write** (String word, String file)
static void **writeLine** (String word, String file)
static void **delete** (String file)
static int **getLength** (String file)
static String **getLine** (int number, String file)
static int **indexOf** (String word, String file)
static void **replace** (String oldWord, String newWord, String file)
static void **replace** (int index, String newWord, String file)
static boolean **isReady** (String file)
static void **deleteLine** (int index, String file)
static String **toString** (String file)

Detailed Description

Helper class for reading and writing to a text file.

Definition at line **10** of file **ReadWrite.java**.

Member Function Documentation

static void dynamicAnalysis.ReadWrite.delete (String *file*) [static]

Delete.

Parameters

file	the text file to write to

Definition at line **65** of file **ReadWrite.java**.

static void dynamicAnalysis.ReadWrite.deleteLine (int *index*, String *file*) [static]

Delete line at the specified index.

Parameters

index	the index to delete the line at
file	the text file

Definition at line **316** of file **ReadWrite.java**.

static int dynamicAnalysis.ReadWrite.getLength (String *file*) [static]

Gets the length.

Parameters

file	the text file to write to

Returns

the number of lines in the file

Definition at line **76** of file **ReadWrite.java**.

static String dynamicAnalysis.ReadWrite.getLine (int *number*, String *file*) [static]

Gets the line at the specified index.

Parameters

number	the index to retrieve
file	the file

Returns

the line at the specified index

Definition at line **113** of file **ReadWrite.java**.

static int dynamicAnalysis.ReadWrite.indexOf (String *word*, String *file*) [static]

First index that the specified string is found at. Returns -1 if it is not found.

Parameters

word	the string to be searched in the text file
file	the text file to be searched

Returns

the index that the string is located in the text file, -1 if it is not found

Definition at line **150** of file **ReadWrite.java**.

static boolean dynamicAnalysis.ReadWrite.isReady (String *file*) [static]

Checks if the text file is ready to be written and read from.

Parameters

file	the text file

Returns

true, if is ready to be read and wrote to

Definition at line **293** of file **ReadWrite.java**.

static void dynamicAnalysis.ReadWrite.replace (int *index*, String *newWord*, String *file*) [static]

Replace the line at the specified index with a new line.

Parameters

index	the index to be replaces
newWord	the new word to replace at the index
file	the text file

Definition at line **240** of file **ReadWrite.java**.

static void dynamicAnalysis.ReadWrite.replace (String *oldWord*, String *newWord*, String *file*) [static]

Replace the first occurence of a line with another.

Parameters

oldWord	the line to be replaces
newWord	the new line to replace the first occurence of oldWord
file	the text file

Definition at line **189** of file **ReadWrite.java**.

static String dynamicAnalysis.ReadWrite.toString (String *file*) [static]

Reads the entire text file and returns as a string

Parameters

file	the text file

Returns

the entire contents of the text file

Definition at line **366** of file **ReadWrite.java**.

static void dynamicAnalysis.ReadWrite.write (String *word*, String *file*) [static]

Write to a file. Clears existing lines in the file.

Parameters

word	the string to be written to the text file
file	the text file to write to

Definition at line **19** of file **ReadWrite.java**.

static void dynamicAnalysis.ReadWrite.writeLine (String *word*, String *file*) [static]

Write to a file. Keeps existing lines in the file.

Parameters

word	the string to be written to the text file
file	the text file to write to

Definition at line **46** of file **ReadWrite.java**.

The documentation for this class was generated from the following file:

src/dynamicAnalysis/ReadWrite.java

dynamicAnalysis.SelectFile Class Reference

Public Member Functions

SelectFile (int x, int y, boolean pidMode)
void **open** ()
int **getX** ()
void **setX** (int x)
int **getY** ()
void **setY** (int y)
boolean **isPidMode** ()
void **setPidMode** (boolean pidMode)
String **getText** ()
int **getPid** ()
void **setPid** (int pid)
boolean **isDisposed** ()
void **focus** ()

Protected Member Functions

void **createContents** ()

Protected Attributes

Shell **shell**

Detailed Description

The Class **SelectFile**. Displays a window to choose a file from. Can also select a process.

Definition at line **24** of file **SelectFile.java**.

Constructor & Destructor Documentation

dynamicAnalysis.SelectFile.SelectFile (int *x*, int *y*, boolean *pidMode*)

Launch the file selection window.

Parameters

x	the X value to position at
y	the Y value to position at
pidMode	whether to read a file or a process

Definition at line **55** of file **SelectFile.java**.

Member Function Documentation

void dynamicAnalysis.SelectFile.createContents () `[protected]`

Create contents of the window.

Definition at line **88** of file **SelectFile.java**.

void dynamicAnalysis.SelectFile.focus ()

Force focus on the file selection window.

Definition at line **345** of file **SelectFile.java**.

int dynamicAnalysis.SelectFile.getPid ()

Gets the PID.

Returns

the PID

Definition at line **317** of file **SelectFile.java**.

String dynamicAnalysis.SelectFile.getText ()

Gets the file path.

Returns

the file path

Definition at line **307** of file **SelectFile.java**.

int dynamicAnalysis.SelectFile.getX ()

Gets the X value.

Returns

the X value

Definition at line **247** of file **SelectFile.java**.

int dynamicAnalysis.SelectFile.getY ()

Gets the Y value.

Returns

the Y value

Definition at line **267** of file **SelectFile.java**.

boolean dynamicAnalysis.SelectFile.isDisposed ()

Checks if the window is disposed.

Returns

true, if window is disposed

Definition at line **337** of file **SelectFile.java**.

boolean dynamicAnalysis.SelectFile.isPidMode ()

Checks if the window should read a process or a file.

Returns

true, if is in pid mode

Definition at line **287** of file **SelectFile.java**.

void dynamicAnalysis.SelectFile.open ()

Open the window.

Definition at line **66** of file **SelectFile.java**.

void dynamicAnalysis.SelectFile.setPid (int *pid*)

Sets the PID.

Parameters

pid	the new PID

Definition at line **327** of file **SelectFile.java**.

void dynamicAnalysis.SelectFile.setPidMode (boolean *pidMode*)

Sets the pid mode.

Parameters

pidMode	the new pid mode

Definition at line **297** of file **SelectFile.java**.

void dynamicAnalysis.SelectFile.setX (int *x*)

Sets the X value.

Parameters

x	the new X value

Definition at line **257** of file **SelectFile.java**.

void dynamicAnalysis.SelectFile.setY (int *y*)

Sets the Y value.

Parameters

y	the new Y value

Definition at line **277** of file **SelectFile.java**.

Member Data Documentation

Shell dynamicAnalysis.SelectFile.shell `[protected]`

The SWT shell for the window.

Definition at line **28** of file **SelectFile.java**.

The documentation for this class was generated from the following file:

src/dynamicAnalysis/SelectFile.java

dynamicAnalysis.SelectProcess Class Reference

Public Member Functions

SelectProcess (String[] names, int[] pids, String[] memory, int x, int y)
void **open** ()
String[] **getNames** ()
void **setNames** (String[] names)
int[] **getPids** ()
void **setPids** (int[] pids)
int **getPid** ()
String[] **getMemory** ()
void **setMemory** (String[] memory)
int **getX** ()
void **setX** (int x)
int **getY** ()
void **setY** (int y)

Protected Member Functions

void **createContents** ()

Protected Attributes

Shell **shell**

Detailed Description

The window to choose a currently running process to hook into. Displays a list of all running processes

Definition at line **33** of file **SelectProcess.java**.

Constructor & Destructor Documentation

dynamicAnalysis.SelectProcess.SelectProcess (String[] *names*, int[] *pids*, String[] *memory*, int *x*, int *y*)

Launch the file selection window.

Parameters

names	the names of the processes
pids	the PIDs of the processes
memory	the memory that the processes use
x	the X value for positioning
y	the Y value for positioning

Definition at line **76** of file **SelectProcess.java**.

Member Function Documentation

void dynamicAnalysis.SelectProcess.createContents () `[protected]`

Create contents of the window.

Definition at line **296** of file **SelectProcess.java**.

String[] dynamicAnalysis.SelectProcess.getMemory ()

Gets the memory used by the processes.

Returns

the memory used by the processes

Definition at line **169** of file **SelectProcess.java**.

String[] dynamicAnalysis.SelectProcess.getNames ()

Gets the names of the processes.

Returns

the names of the processes

Definition at line **109** of file **SelectProcess.java**.

int dynamicAnalysis.SelectProcess.getPid ()

Gets the unique identifier of a process.

Returns

the unique identifier of a process

Definition at line **149** of file **SelectProcess.java**.

int[] dynamicAnalysis.SelectProcess.getPids ()

Gets the unique identifiers of the processes.

Returns

the unique identifiers of the processes

Definition at line **129** of file **SelectProcess.java**.

int dynamicAnalysis.SelectProcess.getX ()

Gets the X value.

Returns

the X value

Definition at line **189** of file **SelectProcess.java**.

int dynamicAnalysis.SelectProcess.getY ()

Gets the Y value.

Returns

the Y value

Definition at line **209** of file **SelectProcess.java**.

void dynamicAnalysis.SelectProcess.open ()

Open the window.

Definition at line **88** of file **SelectProcess.java**.

void dynamicAnalysis.SelectProcess.setMemory (String[] *memory*)

Sets the memory used by the processes.

Parameters

memory	the new memory used by the processes

Definition at line **179** of file **SelectProcess.java**.

void dynamicAnalysis.SelectProcess.setNames (String[] *names*)

Sets the names of the processes.

Parameters

names	the new names of the processes

Definition at line **119** of file **SelectProcess.java**.

void dynamicAnalysis.SelectProcess.setPids (int[] *pids*)

Sets the unique identifiers of the processes.

Parameters

pids	the new unique identifiers of the processes

Definition at line **139** of file **SelectProcess.java**.

void dynamicAnalysis.SelectProcess.setX (int *x*)

Sets the X value.

Parameters

x	the new X value

Definition at line **199** of file **SelectProcess.java**.

void dynamicAnalysis.SelectProcess.setY (int *y*)

Sets the Y value.

Parameters

y	the new Y value

Definition at line **219** of file **SelectProcess.java**.

Member Data Documentation

Shell dynamicAnalysis.SelectProcess.shell `[protected]`

The shell for the process selection window.

Definition at line **37** of file **SelectProcess.java**.

The documentation for this class was generated from the following file:

src/dynamicAnalysis/SelectProcess.java

org.eclipse.wb.swt.SWTResourceManager Class Reference

Static Public Member Functions

static Color **getColor** (int systemColorID)
static Color **getColor** (int r, int g, int b)
static Color **getColor** (RGB rgb)
static void **disposeColors** ()
static Image **getImage** (String path)
static Image **getImage** (Class<?> clazz, String path)
static Image **decorateImage** (Image baseImage, Image decorator)
static Image **decorateImage** (final Image baseImage, final Image decorator, final int corner)
static void **disposeImages** ()
static Font **getFont** (String name, int height, int style)
static Font **getFont** (String name, int size, int style, boolean strikeout, boolean underline)
static Font **getBoldFont** (Font baseFont)
static void **disposeFonts** ()
static Cursor **getCursor** (int id)
static void **disposeCursors** ()
static void **dispose** ()

Static Public Attributes

static final int **TOP_LEFT** = 1
static final int **TOP_RIGHT** = 2
static final int **BOTTOM_LEFT** = 3
static final int **BOTTOM_RIGHT** = 4

Static Protected Member Functions

static Image **getImage** (InputStream stream) throws IOException

Static Protected Attributes

static final int **LAST_CORNER_KEY** = 5

Detailed Description

Utility class for managing OS resources associated with SWT controls such as colors, fonts, images, etc.

!!! IMPORTANT !!! Application code must explicitly invoke the **`dispose()`** method to release the operating system resources managed by cached objects when those objects and OS resources are no longer needed (e.g. on application shutdown)

This class may be freely distributed as part of any application or plugin.

Author

scheglov_ke

Dan Rubel

Definition at line **43** of file **SWTResourceManager.java**.

Member Function Documentation

static Image org.eclipse.wb.swt.SWTResourceManager.decorateImage (final Image *baseImage*, final Image *decorator*, final int *corner*) `[static]`

Returns an **`Image`** composed of a base image decorated by another image.

Parameters

baseImage	the base **`Image`** that should be decorated
decorator	the **`Image`** to decorate the base image
corner	the corner to place decorator image

Returns

the resulting decorated **`Image`**

Definition at line **233** of file **SWTResourceManager.java**.

static Image org.eclipse.wb.swt.SWTResourceManager.decorateImage (Image *baseImage*, Image *decorator*) `[static]`

Returns an **`Image`** composed of a base image decorated by another image.

Parameters

baseImage	the base **`Image`** that should be decorated
decorator	the **`Image`** to decorate the base image

Returns

`Image` The resulting decorated image

Definition at line **219** of file **SWTResourceManager.java**.

static void org.eclipse.wb.swt.SWTResourceManager.dispose () `[static]`

Dispose of cached objects and their underlying OS resources. This should only be called when the cached objects are no longer needed (e.g. on application shutdown).

Definition at line **441** of file **SWTResourceManager.java**.

static void org.eclipse.wb.swt.SWTResourceManager.disposeColors () `[static]`

Dispose of all the cached **`Color`** 's.

Definition at line **94** of file **SWTResourceManager.java**.

static void org.eclipse.wb.swt.SWTResourceManager.disposeCursors () `[static]`

Dispose all of the cached cursors.

Definition at line **426** of file **SWTResourceManager.java**.

static void org.eclipse.wb.swt.SWTResourceManager.disposeFonts () `[static]`

Dispose all of the cached **`Font`** 's.

Definition at line **386** of file **SWTResourceManager.java**.

static void org.eclipse.wb.swt.SWTResourceManager.disposeImages () `[static]`

Dispose all of the cached **`Image`** 's.

Definition at line **275** of file **SWTResourceManager.java**.

static Font org.eclipse.wb.swt.SWTResourceManager.getBoldFont (Font *baseFont*) `[static]`

Returns a bold version of the given **`Font`** .

Parameters

baseFont	the **`Font`** for which a bold version is desired

Returns

the bold version of the given **`Font`**

Definition at line **373** of file **SWTResourceManager.java**.

static Color org.eclipse.wb.swt.SWTResourceManager.getColor (int *r*, int *g*, int *b*) [static]

Returns a **Color** given its red, green and blue component values.

Parameters

r	the red component of the color
g	the green component of the color
b	the blue component of the color

Returns

the **Color** matching the given red, green and blue component values

Definition at line **72** of file **SWTResourceManager.java**.

static Color org.eclipse.wb.swt.SWTResourceManager.getColor (int *systemColorID*) [static]

Returns the system **Color** matching the specific ID.

Parameters

systemColorID	the ID value for the color

Returns

the system **Color** matching the specific ID

Definition at line **57** of file **SWTResourceManager.java**.

static Color org.eclipse.wb.swt.SWTResourceManager.getColor (RGB *rgb*) [static]

Returns a **Color** given its RGB value.

Parameters

rgb	the **RGB** value of the color

Returns

the **Color** matching the RGB value

Definition at line **82** of file **SWTResourceManager.java**.

static Cursor org.eclipse.wb.swt.SWTResourceManager.getCursor (int *id*) [static]

Returns the system cursor matching the specific ID.

Parameters

id	int The ID value for the cursor

Returns

Cursor The system cursor matching the specific ID

Definition at line **414** of file **SWTResourceManager.java**.

static Font org.eclipse.wb.swt.SWTResourceManager.getFont (String *name*, int *height*, int *style*) [static]

Returns a **Font** based on its name, height and style.

Parameters

name	the name of the font
height	the height of the font
style	the style of the font

Returns

`Font` The font matching the name, height and style

Definition at line **321** of file **SWTResourceManager.java**.

static Font org.eclipse.wb.swt.SWTResourceManager.getFont (String *name*, int *size*, int *style*, boolean *strikeout*, boolean *underline*) `[static]`

Returns a `Font` based on its name, height and style. Windows-specific strikeout and underline flags are also supported.

Parameters

name	the name of the font
size	the size of the font
style	the style of the font
strikeout	the strikeout flag (warning: Windows only)
underline	the underline flag (warning: Windows only)

Returns

`Font` The font matching the name, height, style, strikeout and underline

Definition at line **340** of file **SWTResourceManager.java**.

static Image org.eclipse.wb.swt.SWTResourceManager.getImage (Class<?> *clazz*, String *path*) `[static]`

Returns an `Image` stored in the file at the specified path relative to the specified class.

Parameters

clazz	the `Class` relative to which to find the image
path	the path to the image file, if starts with `'/'`

Returns

the `Image` stored in the file at the specified path

Definition at line **157** of file **SWTResourceManager.java**.

static Image org.eclipse.wb.swt.SWTResourceManager.getImage (InputStream *stream*) throws IOException `[static]`, `[protected]`

Returns an `Image` encoded by the specified `InputStream`.

Parameters

stream	the `InputStream` encoding the image data

Returns

the `Image` encoded by the specified input stream

Definition at line **116** of file **SWTResourceManager.java**.

static Image org.eclipse.wb.swt.SWTResourceManager.getImage (String *path*) `[static]`

Returns an `Image` stored in the file at the specified path.

Parameters

path	the path to the image file

Returns

the `Image` stored in the file at the specified path

Definition at line **135** of file **SWTResourceManager.java**.

Member Data Documentation

final int org.eclipse.wb.swt.SWTResourceManager.BOTTOM_LEFT = 3 `[static]`

Style constant for placing decorator image in bottom left corner of base image.

Definition at line **196** of file **SWTResourceManager.java**.

final int org.eclipse.wb.swt.SWTResourceManager.BOTTOM_RIGHT = 4 `[static]`

Style constant for placing decorator image in bottom right corner of base image.

Definition at line **200** of file **SWTResourceManager.java**.

final int org.eclipse.wb.swt.SWTResourceManager.LAST_CORNER_KEY = 5 `[static]`, `[protected]`

Internal value.

Definition at line **204** of file **SWTResourceManager.java**.

final int org.eclipse.wb.swt.SWTResourceManager.TOP_LEFT = 1 `[static]`

Style constant for placing decorator image in top left corner of base image.

Definition at line **188** of file **SWTResourceManager.java**.

final int org.eclipse.wb.swt.SWTResourceManager.TOP_RIGHT = 2 `[static]`

Style constant for placing decorator image in top right corner of base image.

Definition at line **192** of file **SWTResourceManager.java**.

The documentation for this class was generated from the following file:

src/org/eclipse/wb/swt/SWTResourceManager.java

dynamicAnalysis.test Class Reference

Public Member Functions

void **open** ()

Static Public Member Functions

static void **main** (String[] args)

Protected Member Functions

void **createContents** ()

Protected Attributes

Shell **shell**

Detailed Description

The Class test.

Definition at line **21** of file **test.java**.

Member Function Documentation

void dynamicAnalysis.test.createContents () `[protected]`

Create contents of the window.

Definition at line **74** of file **test.java**.

static void dynamicAnalysis.test.main (String[] *args*) `[static]`

Launch the application.

Parameters

args	the arguments

Definition at line **41** of file **test.java**.

void dynamicAnalysis.test.open ()

Open the window.

Definition at line **56** of file **test.java**.

Member Data Documentation

Shell dynamicAnalysis.test.shell `[protected]`

The shell.

Definition at line **25** of file **test.java**.

The documentation for this class was generated from the following file:

src/dynamicAnalysis/test.java

dynamicAnalysis.Version Enum Reference

Public Member Functions

boolean **getValue** ()

Public Attributes

x32 =(false)
x64 =(true)

Detailed Description

The **Version** enum. Translates a boolean value into either x32 or x64 for a PE file

Definition at line **9** of file **Version.java**.

Member Function Documentation

boolean dynamicAnalysis.Version.getValue ()

Gets the boolean value from the PE version.

Returns

the boolean value

Definition at line **34** of file **Version.java**.

Member Data Documentation

dynamicAnalysis.Version.x32 =(false)

The x32 value, set to false.

Definition at line **12** of file **Version.java**.

dynamicAnalysis.Version.x64 =(true)

The x64 value, set to true.

Definition at line **15** of file **Version.java**.

The documentation for this enum was generated from the following file:

src/dynamicAnalysis/Version.java

dynamicAnalysis.VirtualMemory Class Reference

Public Member Functions

VirtualMemory (int processId)
int **getProcessId** ()
void **setProcessId** (int processId)
byte[] **readMemory** ()

Detailed Description

Top level loader to call VirtualMemory.cpp through Java Native Interface. Retrieves the virtual memory from a given process ID.

Definition at line **11** of file **VirtualMemory.java**.

Constructor & Destructor Documentation

dynamicAnalysis.VirtualMemory.VirtualMemory (int *processId*)

Instantiates a new virtual memory with the process ID.

Parameters

processId	the process ID to retrieve the virtual memory space from

Definition at line **32** of file **VirtualMemory.java**.

Member Function Documentation

int dynamicAnalysis.VirtualMemory.getProcessId ()

Gets the unique identifier from the selected process.

Returns

the unique identifier of the process

Definition at line **42** of file **VirtualMemory.java**.

byte[] dynamicAnalysis.VirtualMemory.readMemory ()

Calls the C++ function to read the virtual memory space, given a process ID.

Returns

the full virtual memory space of the process

Definition at line **75** of file **VirtualMemory.java**.

void dynamicAnalysis.VirtualMemory.setProcessId (int *processId*)

Sets the unique identifier for the process.

Parameters

processId	the new unique identifier for the selected process.

Definition at line **52** of file **VirtualMemory.java**.

The documentation for this class was generated from the following file:

src/dynamicAnalysis/VirtualMemory.java

dynamicAnalysis.Window Class Reference

Public Member Functions

void **open** ()

Static Public Member Functions

static void **main** (String[] args)

Static Public Attributes

static int **processId**

Protected Member Functions

void **createContents** ()

Protected Attributes

Shell **shell**

Detailed Description

The main window that is loaded when the program is first run. Contains most functionality.

Definition at line **49** of file **Window.java**.

Member Function Documentation

void dynamicAnalysis.Window.createContents () `[protected]`

Create contents of the window.

Definition at line **130** of file **Window.java**.

static void dynamicAnalysis.Window.main (String[] *args*) `[static]`

Launch the application.

Parameters

args	the arguments

Definition at line **81** of file **Window.java**.

void dynamicAnalysis.Window.open ()

Open the window.

Definition at line **98** of file **Window.java**.

Member Data Documentation

int dynamicAnalysis.Window.processId `[static]`

The process id.

Definition at line **68** of file **Window.java**.

Shell dynamicAnalysis.Window.shell `[protected]`

The window's SWT shell.

Definition at line **53** of file **Window.java**.

The documentation for this class was generated from the following file:

src/dynamicAnalysis/Window.java

File Documentation

dynamicAnalysis_ExecuteCode.h

```
00001 /* DO NOT EDIT THIS FILE - it is machine generated */
00002 #include <jni.h>
00003 /* Header for class dynamicAnalysis_ExecuteCode */
00004
00005 #ifndef _Included_dynamicAnalysis_ExecuteCode
00006 #define _Included_dynamicAnalysis_ExecuteCode
00007 #ifdef __cplusplus
00008 extern "C" {
00009 #endif
00010 /*
00011  * Class:     dynamicAnalysis_ExecuteCode
00012  * Method:    executeInstruction
00013  * Signature: (B)V
00014  */
00015 JNIEXPORT void JNICALL Java_dynamicAnalysis_ExecuteCode_executeInstruction
00016   (JNIEnv *, jobject, jbyte);
00017 /*
00018  * Class:     dynamicAnalysis_ExecuteCode
00019  * Method:    read
00020  * Signature: (V)[B
00021  */
00022 JNIEXPORT jbyteArray JNICALL Java_dynamicAnalysis_ExecuteCode_read
00023   (JNIEnv *, jobject);
00024
00025 #ifdef __cplusplus
00026 }
00027 #endif
00028 #endif
```

dynamicAnalysis_ExecuteCode.h

```
00001 /* DO NOT EDIT THIS FILE - it is machine generated */
00002 #include <jni.h>
00003 /* Header for class dynamicAnalysis_ExecuteCode */
00004
00005 #ifndef  Included dynamicAnalysis ExecuteCode
00006 #define _Included_dynamicAnalysis_ExecuteCode
00007 #ifdef __cplusplus
00008 extern "C" {
00009 #endif
00010 /*
00011  * Class:     dynamicAnalysis ExecuteCode
00012  * Method:    executeInstruction
00013  * Signature: (B)V
00014  */
00015 JNIEXPORT void JNICALL Java_dynamicAnalysis_ExecuteCode_executeInstruction
00016   (JNIEnv *, jobject, jbyte);
00017 /*
00018  * Class:     dynamicAnalysis_ExecuteCode
00019  * Method:    read
00020  * Signature: (V)[B
00021  */
00022 JNIEXPORT jbyteArray JNICALL Java_dynamicAnalysis_ExecuteCode_read
00023   (JNIEnv *, jobject);
00024
00025 #ifdef __cplusplus
00026 }
00027 #endif
00028 #endif
```

dynamicAnalysis_ExecuteCode.h

```
00001 /* DO NOT EDIT THIS FILE - it is machine generated */
00002 #include <jni.h>
00003 /* Header for class dynamicAnalysis_ExecuteCode */
00004
00005 #ifndef  Included dynamicAnalysis ExecuteCode
00006 #define _Included_dynamicAnalysis_ExecuteCode
00007 #ifdef __cplusplus
00008 extern "C" {
00009 #endif
00010 /*
00011  * Class:     dynamicAnalysis ExecuteCode
00012  * Method:    executeInstruction
00013  * Signature: (B)V
00014  */
00015 JNIEXPORT void JNICALL Java_dynamicAnalysis_ExecuteCode_executeInstruction
00016   (JNIEnv *, jobject, jbyte);
00017 /*
00018  * Class:     dynamicAnalysis_ExecuteCode
00019  * Method:    read
00020  * Signature: (V)[B
00021  */
00022 JNIEXPORT jbyteArray JNICALL Java_dynamicAnalysis_ExecuteCode_read
00023   (JNIEnv *, jobject);
00024
00025 #ifdef __cplusplus
00026 }
00027 #endif
00028 #endif
```

dynamicAnalysis_VirtualMemory.h

```
00001 /* DO NOT EDIT THIS FILE - it is machine generated */
00002 #include <jni.h>
00003 /* Header for class dynamicAnalysis_VirtualMemory */
00004
00005 #ifndef  Included dynamicAnalysis VirtualMemory
00006 #define _Included_dynamicAnalysis_VirtualMemory
00007 #ifdef __cplusplus
00008 extern "C" {
00009 #endif
00010 /*
00011  * Class:     dynamicAnalysis VirtualMemory
00012  * Method:    scanProcess
00013  * Signature: (I)[B
00014  */
00015 JNIEXPORT jbyteArray JNICALL Java_dynamicAnalysis_VirtualMemory_scanProcess
00016   (JNIEnv *, jobject, jint);
00017
00018 #ifdef __cplusplus
00019 }
00020 #endif
00021 #endif
```

dynamicAnalysis_VirtualMemory.h

```
00001 /* DO NOT EDIT THIS FILE - it is machine generated */
00002 #include <jni.h>
00003 /* Header for class dynamicAnalysis_VirtualMemory */
00004
00005 #ifndef  Included dynamicAnalysis VirtualMemory
00006 #define _Included_dynamicAnalysis_VirtualMemory
00007 #ifdef __cplusplus
00008 extern "C" {
00009 #endif
00010 /*
00011  * Class:     dynamicAnalysis VirtualMemory
00012  * Method:    scanProcess
00013  * Signature: (I)[B
00014  */
00015 JNIEXPORT jbyteArray JNICALL Java_dynamicAnalysis_VirtualMemory_scanProcess
00016   (JNIEnv *, jobject, jint);
00017
00018 #ifdef __cplusplus
00019 }
00020 #endif
00021 #endif
```

dynamicAnalysis_VirtualMemory.h

```
00001 /* DO NOT EDIT THIS FILE - it is machine generated */
00002 #include <jni.h>
00003 /* Header for class dynamicAnalysis_VirtualMemory */
00004
00005 #ifndef  Included dynamicAnalysis VirtualMemory
00006 #define _Included_dynamicAnalysis_VirtualMemory
00007 #ifdef __cplusplus
00008 extern "C" {
00009 #endif
00010 /*
00011  * Class:     dynamicAnalysis VirtualMemory
00012  * Method:    scanProcess
00013  * Signature: (I)[B
00014  */
00015 JNIEXPORT jbyteArray JNICALL Java_dynamicAnalysis_VirtualMemory_scanProcess
00016   (JNIEnv *, jobject, jint);
00017
00018 #ifdef __cplusplus
00019 }
00020 #endif
00021 #endif
```

ExecuteImpl.c

```
00001 #include <jni.h>        // JNI header provided by JDK
00002 #include <stdio.h>      // C Standard IO Header
00003 #include "dynamicAnalysis_ExecuteCode.h"   // Generated
00004
00005 // Implementation of the native method sayHello()
00006 JNIEXPORT void JNICALL JNICALL
Java_dynamicAnalysis_ExecuteCode_executeInstruction
00007   (JNIEnv *, jobject, jbyte) {
00008    printf("Hello World!\n");
00009    return;
00010 }
```

ExecuteImpl.c

```
00001 #include <jni.h>        // JNI header provided by JDK
00002 #include <stdio.h>      // C Standard IO Header
00003 #include "dynamicAnalysis_ExecuteCode.h"   // Generated
00004
00005 // Implementation of the native method sayHello()
00006 JNIEXPORT void JNICALL JNICALL
Java_dynamicAnalysis_ExecuteCode_executeInstruction
00007   (JNIEnv *, jobject, jbyte) {
00008    printf("Hello World!\n");
00009    return;
00010 }
```

ExecuteImpl.c

```
00001 #include <jni.h>        // JNI header provided by JDK
00002 #include <stdio.h>      // C Standard IO Header
00003 #include "dynamicAnalysis_ExecuteCode.h"   // Generated
00004
00005 // Implementation of the native method sayHello()
00006 JNIEXPORT void JNICALL JNICALL
Java_dynamicAnalysis_ExecuteCode_executeInstruction
00007   (JNIEnv *, jobject, jbyte) {
00008    printf("Hello World!\n");
00009    return;
00010 }
```

ReadProcess.cpp

```
00001 #include <iostream>
00002 #include <windows.h>
00003 #include <memoryapi.h>
00004
00005 using namespace std;
00006
00007 void EnableDebugPriv()
00008 {
00009     HANDLE hToken;
00010     LUID luid;
00011     TOKEN PRIVILEGES tkp;
00012
00013     OpenProcessToken(GetCurrentProcess(), TOKEN_ADJUST_PRIVILEGES | TOKEN_QUERY,
&hToken);
00014
00015     LookupPrivilegeValue(NULL, SE_DEBUG_NAME, &luid);
00016
00017     tkp.PrivilegeCount = 1;
00018     tkp.Privileges[0].Luid = luid;
00019     tkp.Privileges[0].Attributes = SE_PRIVILEGE_ENABLED;
00020
00021     AdjustTokenPrivileges(hToken, false, &tkp, sizeof(tkp), NULL, NULL);
00022
00023     CloseHandle(hToken);
00024 }
00025
00026 int main()
00027 {
00028     EnableDebugPriv();
00029     MEMORY_BASIC_INFORMATION mbi; //mbi used as register for assigning in query
00030     HANDLE hProcess = OpenProcess(PROCESS_ALL_ACCESS, FALSE, 16824); //process
using pid
00031     DWORD oldprotect; //given the value of old protect
00032     BYTE* buffer; //string for buffer
00033     int addr; //need to find what this gets assigned to
00034     VirtualQueryEx(hProcess, NULL, (PMEMORY_BASIC_INFORMATION)&mbi,
sizeof(mbi)); //early assign of mbi, equals 28 (size)
00035     addr = (int)mbi.BaseAddress; //BaseAddress gives 0, shouldnt be correct
00036     int check = VirtualProtectEx(hProcess, 0, mbi.RegionSize,
PAGE_EXECUTE_READWRITE, &oldprotect);
00037     int temp;
00038     SIZE_T size;
00039     while (temp = VirtualQueryEx(hProcess, (LPCVOID)addr,
(PMEMORY_BASIC_INFORMATION)&mbi, mbi.RegionSize)) //will not work for 64 bit programs
(Invalid access to memory location.)
00040     {
00041         int check = VirtualProtectEx(hProcess, (LPVOID)addr, mbi.RegionSize,
PAGE_EXECUTE_READWRITE, &oldprotect); //issue here, error 487: ERROR_INVALID_ADDRESS,
address incrementing seems valid. 100% relation to whether memory gets read
00042         if (check != 0)
00043         {
00044             buffer = new BYTE[mbi.RegionSize];
00045             ReadProcessMemory(hProcess, (LPVOID)addr, buffer, mbi.RegionSize,
&size); //not filling entire buffer
00046             VirtualProtectEx(hProcess, (LPVOID)addr, mbi.RegionSize, oldprotect,
&oldprotect);
00047             cout << buffer;
00048         }
00049         addr += mbi.RegionSize;
00050     }
00051     CloseHandle(hProcess);
00052
00053 }
```

ReadProcess.cpp

```
00001 #include <iostream>
00002 #include <windows.h>
00003 #include <memoryapi.h>
00004
00005 using namespace std;
00006
00007 void EnableDebugPriv()
00008 {
00009     HANDLE hToken;
00010     LUID luid;
00011     TOKEN PRIVILEGES tkp;
00012
00013     OpenProcessToken(GetCurrentProcess(), TOKEN_ADJUST_PRIVILEGES | TOKEN_QUERY,
&hToken);
00014
00015     LookupPrivilegeValue(NULL, SE_DEBUG_NAME, &luid);
00016
00017     tkp.PrivilegeCount = 1;
00018     tkp.Privileges[0].Luid = luid;
00019     tkp.Privileges[0].Attributes = SE_PRIVILEGE_ENABLED;
00020
00021     AdjustTokenPrivileges(hToken, false, &tkp, sizeof(tkp), NULL, NULL);
00022
00023     CloseHandle(hToken);
00024 }
00025
00026 int main()
00027 {
00028     EnableDebugPriv();
00029     MEMORY_BASIC_INFORMATION mbi; //mbi used as register for assigning in query
00030     HANDLE hProcess = OpenProcess(PROCESS_ALL_ACCESS, FALSE, 16824); //process
using pid
00031     DWORD oldprotect; //given the value of old protect
00032     BYTE* buffer; //string for buffer
00033     int addr; //need to find what this gets assigned to
00034     VirtualQueryEx(hProcess, NULL, (PMEMORY_BASIC_INFORMATION)&mbi,
sizeof(mbi)); //early assign of mbi, equals 28 (size)
00035     addr = (int)mbi.BaseAddress; //BaseAddress gives 0, shouldnt be correct
00036     int check = VirtualProtectEx(hProcess, 0, mbi.RegionSize,
PAGE_EXECUTE_READWRITE, &oldprotect);
00037     int temp;
00038     SIZE_T size;
00039     while (temp = VirtualQueryEx(hProcess, (LPCVOID)addr,
(PMEMORY_BASIC_INFORMATION)&mbi, mbi.RegionSize)) //will not work for 64 bit programs
(Invalid access to memory location.)
00040     {
00041         int check = VirtualProtectEx(hProcess, (LPVOID)addr, mbi.RegionSize,
PAGE_EXECUTE_READWRITE, &oldprotect); //issue here, error 487: ERROR_INVALID_ADDRESS,
address incrementing seems valid. 100% relation to whether memory gets read
00042         if (check != 0)
00043         {
00044             buffer = new BYTE[mbi.RegionSize];
00045             ReadProcessMemory(hProcess, (LPVOID)addr, buffer, mbi.RegionSize,
&size); //not filling entire buffer
00046             VirtualProtectEx(hProcess, (LPVOID)addr, mbi.RegionSize, oldprotect,
&oldprotect);
00047             cout << buffer;
00048         }
00049         addr += mbi.RegionSize;
00050     }
00051     CloseHandle(hProcess);
00052
00053 }
```

ReadProcess.cpp

```
00001 #include <iostream>
00002 #include <windows.h>
00003 #include <memoryapi.h>
00004
00005 using namespace std;
00006
00007 void EnableDebugPriv()
00008 {
00009     HANDLE hToken;
00010     LUID luid;
00011     TOKEN PRIVILEGES tkp;
00012
00013     OpenProcessToken(GetCurrentProcess(), TOKEN_ADJUST_PRIVILEGES | TOKEN_QUERY,
&hToken);
00014
00015     LookupPrivilegeValue(NULL, SE_DEBUG_NAME, &luid);
00016
00017     tkp.PrivilegeCount = 1;
00018     tkp.Privileges[0].Luid = luid;
00019     tkp.Privileges[0].Attributes = SE_PRIVILEGE_ENABLED;
00020
00021     AdjustTokenPrivileges(hToken, false, &tkp, sizeof(tkp), NULL, NULL);
00022
00023     CloseHandle(hToken);
00024 }
00025
00026 int main()
00027 {
00028     EnableDebugPriv();
00029     MEMORY_BASIC_INFORMATION mbi; //mbi used as register for assigning in query
00030     HANDLE hProcess = OpenProcess(PROCESS_ALL_ACCESS, FALSE, 16824); //process
using pid
00031     DWORD oldprotect; //given the value of old protect
00032     BYTE* buffer; //string for buffer
00033     int addr; //need to find what this gets assigned to
00034     VirtualQueryEx(hProcess, NULL, (PMEMORY_BASIC_INFORMATION)&mbi,
sizeof(mbi)); //early assign of mbi, equals 28 (size)
00035     addr = (int)mbi.BaseAddress; //BaseAddress gives 0, shouldnt be correct
00036     int check = VirtualProtectEx(hProcess, 0, mbi.RegionSize,
PAGE_EXECUTE_READWRITE, &oldprotect);
00037     int temp;
00038     SIZE_T size;
00039     while (temp = VirtualQueryEx(hProcess, (LPCVOID)addr,
(PMEMORY_BASIC_INFORMATION)&mbi, mbi.RegionSize)) //will not work for 64 bit programs
(Invalid access to memory location.)
00040     {
00041         int check = VirtualProtectEx(hProcess, (LPVOID)addr, mbi.RegionSize,
PAGE_EXECUTE_READWRITE, &oldprotect); //issue here, error 487: ERROR_INVALID_ADDRESS,
address incrementing seems valid. 100% relation to whether memory gets read
00042         if (check != 0)
00043         {
00044             buffer = new BYTE[mbi.RegionSize];
00045             ReadProcessMemory(hProcess, (LPVOID)addr, buffer, mbi.RegionSize,
&size); //not filling entire buffer
00046             VirtualProtectEx(hProcess, (LPVOID)addr, mbi.RegionSize, oldprotect,
&oldprotect);
00047             cout << buffer;
00048         }
00049         addr += mbi.RegionSize;
00050     }
00051     CloseHandle(hProcess);
00052
00053 }
```

VirtualMemory.cpp

```
00001 #include <jni.h>        // JNI header provided by JDK
00002 #include <stdio.h>      // C Standard IO Header
00003 #include "dynamicAnalysis_VirtualMemory.h"   // Generated
00004 #include <windows.h>
00005 #include <memoryapi.h>
00006 #include <vector>
00007
00008 using namespace std;
00009
00010 void EnableDebugPriv()
00011 {
00012     HANDLE hToken;
00013     LUID luid;
00014     TOKEN_PRIVILEGES tkp;
00015
00016     OpenProcessToken(GetCurrentProcess(), TOKEN_ADJUST_PRIVILEGES | TOKEN_QUERY,
&hToken);
00017
00018     LookupPrivilegeValue(NULL, SE_DEBUG_NAME, &luid);
00019
00020     tkp.PrivilegeCount = 1;
00021     tkp.Privileges[0].Luid = luid;
00022     tkp.Privileges[0].Attributes = SE_PRIVILEGE_ENABLED;
00023
00024     AdjustTokenPrivileges(hToken, false, &tkp, sizeof(tkp), NULL, NULL);
00025
00026     CloseHandle(hToken);
00027 }
00028
00029 JNIEXPORT jbyteArray JNICALL Java_dynamicAnalysis_VirtualMemory_scanProcess
(JNIEnv *env, jobject, jint processId)
00030 {
00031     EnableDebugPriv();
00032     MEMORY_BASIC_INFORMATION64 mbi;
00033     HANDLE hProcess = OpenProcess(PROCESS_ALL_ACCESS, FALSE, (int)processId);
00034     DWORD oldprotect;
00035     unsigned char* buffer = new unsigned char[1];
00036     __int64 addr = 0;
00037     VirtualQueryEx(hProcess, 0, (PMEMORY_BASIC_INFORMATION)&mbi, sizeof(mbi));
00038     addr = mbi.BaseAddress;
00039     int check = VirtualProtectEx(hProcess, (LPVOID)0, mbi.RegionSize,
PAGE_EXECUTE_READWRITE, &oldprotect);
00040     SIZE_T size;
00041     int signedSize = 0;
00042     vector<unsigned char> regionData;
00043     while (VirtualQueryEx(hProcess, (LPVOID)addr,
(PMEMORY_BASIC_INFORMATION)&mbi, sizeof(mbi)))
00044     {
00045         signedSize = static_cast<int>(mbi.RegionSize);
00046         int check = VirtualProtectEx(hProcess, (LPVOID)addr, mbi.RegionSize,
PAGE_EXECUTE_READWRITE, &oldprotect);
00047         if (check)
00048         {
00049             try
00050             {
00051                 buffer = new unsigned char[signedSize];
00052                 ReadProcessMemory(hProcess, (LPCVOID)addr, buffer,
mbi.RegionSize, &size);
00053                 for (int index = 0; index < mbi.RegionSize; index++)
00054                 {
00055                     regionData.push_back(buffer[index]);
00056                 }
00057             }
00058             catch (bad_alloc e)
00059             {
00060                 break;
00061             }
00062         }
00063         addr = (addr + mbi.RegionSize);
00064     }
00065     unsigned char* total = new unsigned char[regionData.size()];
00066     int totalIndex = 0;
00067     try
```

```
00068     {
00069         for (auto i = regionData.begin(); i < regionData.end(); ++i)
00070         {
00071             total[totalIndex] = *i;
00072             totalIndex++;
00073         }
00074     }
00075     catch (bad_alloc e) {}
00076     CloseHandle(hProcess);
00077     jbyteArray result = env->NewByteArray(totalIndex);
00078         env->SetByteArrayRegion( result, 0, totalIndex, (const jbyte*)total );
00079     return result;
00080 }
```

VirtualMemory.cpp

```
00001 #include <jni.h>        // JNI header provided by JDK
00002 #include <stdio.h>      // C Standard IO Header
00003 #include "dynamicAnalysis_VirtualMemory.h"   // Generated
00004 #include <windows.h>
00005 #include <memoryapi.h>
00006 #include <vector>
00007
00008 using namespace std;
00009
00010 void EnableDebugPriv()
00011 {
00012     HANDLE hToken;
00013     LUID luid;
00014     TOKEN_PRIVILEGES tkp;
00015
00016     OpenProcessToken(GetCurrentProcess(), TOKEN_ADJUST_PRIVILEGES | TOKEN_QUERY,
&hToken);
00017
00018     LookupPrivilegeValue(NULL, SE_DEBUG_NAME, &luid);
00019
00020     tkp.PrivilegeCount = 1;
00021     tkp.Privileges[0].Luid = luid;
00022     tkp.Privileges[0].Attributes = SE_PRIVILEGE_ENABLED;
00023
00024     AdjustTokenPrivileges(hToken, false, &tkp, sizeof(tkp), NULL, NULL);
00025
00026     CloseHandle(hToken);
00027 }
00028
00029 JNIEXPORT jbyteArray JNICALL Java_dynamicAnalysis_VirtualMemory_scanProcess
(JNIEnv *env, jobject, jint processId)
00030 {
00031     EnableDebugPriv();
00032     MEMORY_BASIC_INFORMATION64 mbi;
00033     HANDLE hProcess = OpenProcess(PROCESS_ALL_ACCESS, FALSE, (int)processId);
00034     DWORD oldprotect;
00035     unsigned char* buffer = new unsigned char[1];
00036     __int64 addr = 0;
00037     VirtualQueryEx(hProcess, 0, (PMEMORY_BASIC_INFORMATION)&mbi, sizeof(mbi));
00038     addr = mbi.BaseAddress;
00039     int check = VirtualProtectEx(hProcess, (LPVOID)0, mbi.RegionSize,
PAGE_EXECUTE_READWRITE, &oldprotect);
00040     SIZE_T size;
00041     int signedSize = 0;
00042     vector<unsigned char> regionData;
00043     while (VirtualQueryEx(hProcess, (LPVOID)addr,
(PMEMORY_BASIC_INFORMATION)&mbi, sizeof(mbi)))
00044     {
00045         signedSize = static_cast<int>(mbi.RegionSize);
00046         int check = VirtualProtectEx(hProcess, (LPVOID)addr, mbi.RegionSize,
PAGE_EXECUTE_READWRITE, &oldprotect);
00047         if (check)
00048         {
00049             try
00050             {
00051                 buffer = new unsigned char[signedSize];
00052                 ReadProcessMemory(hProcess, (LPCVOID)addr, buffer,
mbi.RegionSize, &size);
00053                 for (int index = 0; index < mbi.RegionSize; index++)
00054                 {
00055                     regionData.push_back(buffer[index]);
00056                 }
00057             }
00058             catch (bad_alloc e)
00059             {
00060                 break;
00061             }
00062         }
00063         addr = (addr + mbi.RegionSize);
00064     }
00065     unsigned char* total = new unsigned char[regionData.size()];
00066     int totalIndex = 0;
00067     try
```

```
00068     {
00069         for (auto i = regionData.begin(); i < regionData.end(); ++i)
00070         {
00071             total[totalIndex] = *i;
00072             totalIndex++;
00073         }
00074     }
00075     catch (bad_alloc e) {}
00076     CloseHandle(hProcess);
00077     jbyteArray result = env->NewByteArray(totalIndex);
00078         env->SetByteArrayRegion( result, 0, totalIndex, (const jbyte*)total );
00079     return result;
00080 }
```

VirtualMemory.cpp

```
00001 #include <jni.h>        // JNI header provided by JDK
00002 #include <stdio.h>      // C Standard IO Header
00003 #include "dynamicAnalysis_VirtualMemory.h"   // Generated
00004 #include <windows.h>
00005 #include <memoryapi.h>
00006 #include <vector>
00007
00008 using namespace std;
00009
00010 void EnableDebugPriv()
00011 {
00012     HANDLE hToken;
00013     LUID luid;
00014     TOKEN_PRIVILEGES tkp;
00015
00016     OpenProcessToken(GetCurrentProcess(), TOKEN_ADJUST_PRIVILEGES | TOKEN_QUERY,
&hToken);
00017
00018     LookupPrivilegeValue(NULL, SE_DEBUG_NAME, &luid);
00019
00020     tkp.PrivilegeCount = 1;
00021     tkp.Privileges[0].Luid = luid;
00022     tkp.Privileges[0].Attributes = SE_PRIVILEGE_ENABLED;
00023
00024     AdjustTokenPrivileges(hToken, false, &tkp, sizeof(tkp), NULL, NULL);
00025
00026     CloseHandle(hToken);
00027 }
00028
00029 JNIEXPORT jbyteArray JNICALL Java_dynamicAnalysis_VirtualMemory_scanProcess
(JNIEnv *env, jobject, jint processId)
00030 {
00031     EnableDebugPriv();
00032     MEMORY_BASIC_INFORMATION64 mbi;
00033     HANDLE hProcess = OpenProcess(PROCESS_ALL_ACCESS, FALSE, (int)processId);
00034     DWORD oldprotect;
00035     unsigned char* buffer = new unsigned char[1];
00036     __int64 addr = 0;
00037     VirtualQueryEx(hProcess, 0, (PMEMORY_BASIC_INFORMATION)&mbi, sizeof(mbi));
00038     addr = mbi.BaseAddress;
00039     int check = VirtualProtectEx(hProcess, (LPVOID)0, mbi.RegionSize,
PAGE_EXECUTE_READWRITE, &oldprotect);
00040     SIZE_T size;
00041     int signedSize = 0;
00042     vector<unsigned char> regionData;
00043     while (VirtualQueryEx(hProcess, (LPVOID)addr,
(PMEMORY_BASIC_INFORMATION)&mbi, sizeof(mbi)))
00044     {
00045         signedSize = static_cast<int>(mbi.RegionSize);
00046         int check = VirtualProtectEx(hProcess, (LPVOID)addr, mbi.RegionSize,
PAGE_EXECUTE_READWRITE, &oldprotect);
00047         if (check)
00048         {
00049             try
00050             {
00051                 buffer = new unsigned char[signedSize];
00052                 ReadProcessMemory(hProcess, (LPCVOID)addr, buffer,
mbi.RegionSize, &size);
00053                 for (int index = 0; index < mbi.RegionSize; index++)
00054                 {
00055                     regionData.push_back(buffer[index]);
00056                 }
00057             }
00058             catch (bad_alloc e)
00059             {
00060                 break;
00061             }
00062         }
00063         addr = (addr + mbi.RegionSize);
00064     }
00065     unsigned char* total = new unsigned char[regionData.size()];
00066     int totalIndex = 0;
00067     try
```

```
00068     {
00069         for (auto i = regionData.begin(); i < regionData.end(); ++i)
00070         {
00071             total[totalIndex] = *i;
00072             totalIndex++;
00073         }
00074     }
00075     catch (bad_alloc e) {}
00076     CloseHandle(hProcess);
00077     jbyteArray result = env->NewByteArray(totalIndex);
00078         env->SetByteArrayRegion( result, 0, totalIndex, (const jbyte*)total );
00079     return result;
00080 }
```

ActiveConnection.java

```
00001 package dynamicAnalysis;
00002
00006 public class ActiveConnection
00007 {
00008
00010     private String protocol;
00011
00013     private String localAddress;
00014
00016     private String foreignAddress;
00017
00019     private String state;
00020
00022     private long pid;
00023
00033     public ActiveConnection(String protocol, String localAddress, String
foreignAddress, String state, long pid)
00034     {
00035         setProtocol(protocol);
00036         setLocalAddress(localAddress);
00037         setForeignAddress(foreignAddress);
00038         setState(state);
00039         setPid(pid);
00040     }
00041
00047     public String getProtocol()
00048     {
00049         return protocol;
00050     }
00051
00057     public void setProtocol(String protocol)
00058     {
00059         this.protocol = protocol;
00060     }
00061
00067     public String getLocalAddress()
00068     {
00069         return localAddress;
00070     }
00071
00077     public void setLocalAddress(String localAddress)
00078     {
00079         this.localAddress = localAddress;
00080     }
00081
00087     public String getForeignAddress()
00088     {
00089         return foreignAddress;
00090     }
00091
00097     public void setForeignAddress(String foreignAddress)
00098     {
00099         this.foreignAddress = foreignAddress;
00100     }
00101
00107     public String getState()
00108     {
00109         return state;
00110     }
00111
00117     public void setState(String state)
00118     {
00119         this.state = state;
00120     }
00121
00127     public long getPid()
00128     {
00129         return pid;
00130     }
00131
00137     public void setPid(long pid)
00138     {
00139         this.pid = pid;
```

```
00140     }
00141
00147     @Override
00148     public String toString()
00149     {
00150         return "ActiveConnection [protocol=" + protocol + ", localAddress=" +
localAddress + ", foreignAddress="
00151                 + foreignAddress + ", state=" + state + ", pid=" + pid + "]";
00152     }
00153
00154 }
```

CandidateWindow.java

```
00001 /*
00002  *
00003  */
00004 package dynamicAnalysis;
00005
00006 import org.eclipse.swt.widgets.Display;
00007 import org.eclipse.swt.widgets.Shell;
00008 import org.eclipse.swt.layout.GridLayout;
00009 import org.eclipse.swt.widgets.Text;
00010
00011 import java.io.File;
00012
00013 import org.eclipse.swt.SWT;
00014 import org.eclipse.swt.widgets.Label;
00015 import org.eclipse.swt.layout.GridData;
00016 import org.eclipse.swt.widgets.Button;
00017 import org.eclipse.swt.widgets.Table;
00018 import org.eclipse.swt.widgets.Composite;
00019 import org.eclipse.swt.layout.FormLayout;
00020 import org.eclipse.swt.layout.FormData;
00021 import org.eclipse.swt.layout.FormAttachment;
00022 import org.eclipse.swt.events.ModifyEvent;
00023 import org.eclipse.swt.events.ModifyListener;
00024 import org.eclipse.swt.events.SelectionAdapter;
00025 import org.eclipse.swt.events.SelectionEvent;
00026 import org.eclipse.swt.custom.StyledText;
00027 import org.eclipse.swt.widgets.TableColumn;
00028 import org.eclipse.swt.widgets.TableItem;
00029 import org.eclipse.swt.widgets.Menu;
00030 import org.eclipse.swt.widgets.MenuItem;
00031
00035 public class CandidateWindow
00036 {
00037
00039     protected Shell shell;
00040
00042     private Label text;
00043
00045     private String filePath;
00046
00052     public static void main(String[] args)
00053     {
00054         try
00055         {
00056             CandidateWindow window = new CandidateWindow();
00057             window.open();
00058         } catch (Exception e)
00059         {
00060             e.printStackTrace();
00061         }
00062     }
00063
00067     public void open()
00068     {
00069         Display display = Display.getDefault();
00070         createContents();
00071         shell.open();
00072         shell.layout();
00073         while (!shell.isDisposed())
00074         {
00075             if (!display.readAndDispatch())
00076             {
00077                 display.sleep();
00078             }
00079         }
00080     }
00081
00085     protected void createContents()
00086     {
00087         shell = new Shell();
00088         shell.setSize(324, 249);
00089         shell.setText("SWT Application");
00090         shell.setLayout(new FormLayout());
```

```
00091
00092         Button btnProcess = new Button(shell, SWT.CHECK);
00093
00094         text = new Label(shell, SWT.BORDER);
00095
00096         Button btnLaunch = new Button(shell, SWT.NONE);
00097
00098         FormData fd_text = new FormData();
00099         fd_text.bottom = new FormAttachment(0, 28);
00100         text.setLayoutData(fd_text);
00101
00102
00103         Button btnInstructions = new Button(shell, SWT.NONE);
00104         btnInstructions.setEnabled(false);
00105         btnInstructions.addSelectionListener(new SelectionAdapter() {
00106             @Override
00107             public void widgetSelected(SelectionEvent e) {
00108             }
00109         });
00110         FormData fd_btnInstructions = new FormData();
00111         fd_btnInstructions.left = new FormAttachment(text, 120, SWT.LEFT);
00112         fd_btnInstructions.right = new FormAttachment(text, 0, SWT.RIGHT);
00113         btnInstructions.setLayoutData(fd_btnInstructions);
00114         btnInstructions.setText("x86 Instructions");
00115
00116         Button btnMemory = new Button(shell, SWT.NONE);
00117         fd_btnInstructions.top = new FormAttachment(0, 117);
00118         btnMemory.setEnabled(false);
00119         btnMemory.addSelectionListener(new SelectionAdapter() {
00120             @Override
00121             public void widgetSelected(SelectionEvent e) {
00122             }
00123         });
00124         btnMemory.setText("Virtual Memory");
00125         FormData fd_btnMemory = new FormData();
00126         fd_btnMemory.bottom = new FormAttachment(btnInstructions, -6);
00127         fd_btnMemory.right = new FormAttachment(text, 0, SWT.RIGHT);
00128         fd_btnMemory.left = new FormAttachment(0, 203);
00129         btnMemory.setLayoutData(fd_btnMemory);
00130
00131         Button btnAdvanced = new Button(shell, SWT.NONE);
00132         fd_btnInstructions.bottom = new FormAttachment(btnAdvanced, -6);
00133         btnAdvanced.setEnabled(false);
00134         btnAdvanced.addSelectionListener(new SelectionAdapter() {
00135             @Override
00136             public void widgetSelected(SelectionEvent e) {
00137             }
00138         });
00139         btnAdvanced.setText("Advanced");
00140         FormData fd_btnAdvanced = new FormData();
00141         fd_btnAdvanced.right = new FormAttachment(0, 298);
00142         fd_btnAdvanced.top = new FormAttachment(0, 148);
00143         fd_btnAdvanced.left = new FormAttachment(0, 203);
00144         btnAdvanced.setLayoutData(fd_btnAdvanced);
00145
00146         Button btnSelect = new Button(shell, SWT.NONE);
00147         btnSelect.addSelectionListener(new SelectionAdapter() {
00148             @Override
00149             public void widgetSelected(SelectionEvent e) {
00150                 System.out.println("selection: "+btnProcess.getSelection());
00151                 SelectFile selectFile = new SelectFile(shell.getLocation().x,
shell.getLocation().y, btnProcess.getSelection());
00152                 filePath = selectFile.getText();
00153                 btnLaunch.setEnabled(true);
00154                 try
00155                 {
00156                     text.setText(filePath);
00157                 }
00158                 catch(IllegalArgumentException e1) {}
00159             }
00160         });
00161         btnSelect.setText("Select File");
00162         FormData fd_btnSelect = new FormData();
00163         fd_btnSelect.top = new FormAttachment(text, 6);
00164         fd_btnSelect.right = new FormAttachment(btnMemory, 0, SWT.RIGHT);
00165         fd_btnSelect.left = new FormAttachment(btnMemory, 0, SWT.LEFT);
00166         btnSelect.setLayoutData(fd_btnSelect);
```

```
00167
00168         Label lblFilePath = new Label(shell, SWT.NONE);
00169         fd_text.top = new FormAttachment(lblFilePath, -1, SWT.TOP);
00170         fd_text.right = new FormAttachment(lblFilePath, 221, SWT.RIGHT);
00171         fd_text.left = new FormAttachment(lblFilePath, 6);
00172         lblFilePath.setAlignment(SWT.RIGHT);
00173         FormData fd_lblFilePath = new FormData();
00174         fd_lblFilePath.top = new FormAttachment(0, 12);
00175         fd_lblFilePath.left = new FormAttachment(0, 10);
00176         fd_lblFilePath.right = new FormAttachment(0, 77);
00177         lblFilePath.setLayoutData(fd_lblFilePath);
00178         lblFilePath.setText("File Path");
00179
00180         /*TableItem tableItems[] = new TableItem[4];
00181
00182         tableItems[0].setText(0, "Directory");
00183         tableItems[1].setText(0, "Version");
00184         tableItems[2].setText(0, "Name");
00185         tableItems[3].setText(0, "PID");*/
00186         Menu menu = new Menu(shell, SWT.BAR);
00187         shell.setMenuBar(menu);
00188
00189         MenuItem mntmFile = new MenuItem(menu, SWT.CASCADE);
00190         mntmFile.setText("File");
00191
00192         Menu menu_1 = new Menu(mntmFile);
00193         mntmFile.setMenu(menu_1);
00194
00195         MenuItem mntmOpen = new MenuItem(menu_1, SWT.NONE);
00196         mntmOpen.setText("Open");
00197
00198         MenuItem mntmProcess = new MenuItem(menu, SWT.CASCADE);
00199         mntmProcess.setText("Process");
00200
00201         Menu menu_2 = new Menu(mntmProcess);
00202         mntmProcess.setMenu(menu_2);
00203
00204         MenuItem mntmSelectProcess = new MenuItem(menu_2, SWT.NONE);
00205         mntmSelectProcess.setText("Select Process");
00206         btnProcess.addSelectionListener(new SelectionAdapter() {
00207             @Override
00208             public void widgetSelected(SelectionEvent e) {
00209                 if (btnProcess.getSelection())
00210                 {
00211                     lblFilePath.setText("Process ID");
00212                     btnSelect.setText("Select Process");
00213                 }
00214                 else
00215                 {
00216                     lblFilePath.setText("File Path");
00217                     btnSelect.setText("Select File");
00218                 }
00219             }
00220         });
00221         FormData fd_btnProcess = new FormData();
00222         fd_btnProcess.top = new FormAttachment(btnSelect, 4, SWT.TOP);
00223         fd_btnProcess.right = new FormAttachment(btnSelect, -53);
00224         fd_btnProcess.left = new FormAttachment(text, 0, SWT.LEFT);
00225         btnProcess.setLayoutData(fd_btnProcess);
00226         btnProcess.setText("Process");
00227         btnLaunch.addSelectionListener(new SelectionAdapter() {
00228             @Override
00229             public void widgetSelected(SelectionEvent e) {
00230                 btnMemory.setEnabled(true);
00231                 btnAdvanced.setEnabled(true);
00232                 if(!btnProcess.getSelection())
00233                 {
00234                     btnInstructions.setEnabled(true);
00235                     String filePath = text.getText();
00236                     ProcessManager process = new ProcessManager(new
File(filePath));
00237                     CodeExtract codeExtract = new CodeExtract(new
File(filePath));
00238                     //tableItems[0].setText(1, filePath);
00239                     if(codeExtract.getPeFile().isX32())
00240                     {
00241                         //tableItems[1].setText(1, "32-bit");
```

```
00242                     }
00243                     else
00244                     {
00245                         //tableItems[1].setText(1, "64-bit");
00246                     }
00247                     //tableItems[2].setText(1, process.getName());
00248                     //tableItems[3].setText(1, process.getPidAsString());
00249                 }
00250                 else
00251                 {
00252                     btnInstructions.setEnabled(false);
00253                 }
00254             }
00255         });
00256         FormData fd_btnLaunch = new FormData();
00257         fd_btnLaunch.right = new FormAttachment(100, -345);
00258         fd_btnLaunch.left = new FormAttachment(0, 135);
00259         fd_btnLaunch.top = new FormAttachment(0, 34);
00260         fd_btnLaunch.bottom = new FormAttachment(100, -245);
00261         btnLaunch.setLayoutData(fd_btnLaunch);
00262         btnLaunch.setText("Launch");
00263 
00264     }
00265 }
```

CapstoneTest.java

```
00001 // Java Program to illustrate ProxySelector Class
00002 // of java.net package
00003 // only creating methods here
00004
00005 // Importing standard input output classes
00006 import java.io.IOException;
00007 // Importing classes from java.net package
00008 import java.net.InetSocketAddress;
00009 import java.net.Proxy;
00010 import java.net.ProxySelector;
00011 import java.net.SocketAddress;
00012 import java.net.URI;
00013 // Importing List and ArrayList as utility classes from
00014 // java.util package
00015 import java.util.ArrayList;
00016 import java.util.List;
00017
00018 // Class 1
00022 // Helper class extending ProxySelector class
00023 public class CapstoneTest extends ProxySelector {
00024
00025     // According to API we need to return List<Proxy>
00026     // even if we return only one element, so
00027
00029     // Creating List class object of Proxy type
00030     private final List<Proxy> noProxy = new ArrayList<>();
00031
00033     private final List<Proxy> proxies = new ArrayList<>();
00034
00038     // Constructor of this class
00039     public void  PrivateDataProxy()
00040     {
00041
00042         // If no proxy required to access resource
00043         // use Proxy.NO_PROXY
00044         noProxy.add(Proxy.NO_PROXY);
00045
00046         // Creating  InetSocketAddress, and
00047         // secure.connection.com doesn't exist 443 is an
00048         // https port
00049         InetSocketAddress inetSocketAddress
00050             = new InetSocketAddress("secure.connection.com",
00051                                     443);
00052
00053         // Now creating http proxy
00054         Proxy proxy
00055             = new Proxy(Proxy.Type.HTTP, inetSocketAddress);
00056
00057         // Finally adding proxy into proxy list
00058         proxies.add(proxy);
00059     }
00060
00061     // Method 1 of this class
00068     //@Override
00069     public List<Proxy> select(URI uri)
00070     {
00071         if (uri.getPath().startsWith("/confidential")) {
00072             // If URI path starts with '/confidential' then
00073             // use proxy server
00074             return proxies;
00075         }
00076
00077         // If url don't start with '/confidential' then
00078         //  no need in proxy
00079         return noProxy;
00080     }
00081
00082     // Method 2 of this class
00090     // @Override
00091     public void connectFailed(URI arg0, SocketAddress arg1,
00092                               IOException arg2)
00093     {
00094         // Properly handle connection failing
```

```
00095     }
00096 }
```

CodeExtract.java

```
00001 package dynamicAnalysis;
00002
00003 import java.io.File;
00004 import java.io.IOException;
00005 import java.io.RandomAccessFile;
00006 import java.nio.file.Files;
00007 import java.nio.file.Paths;
00008 import java.util.Arrays;
00009 import java.util.List;
00010 import capstone.Capstone;
00011
00015 public class CodeExtract {
00016
00018     private File file;
00019
00021     private byte[] instructions;
00022
00024     private String code;
00025
00027     private String[] codeArr;
00028
00030     private PEFile peFile;
00031
00033     private Capstone.CsInsn[] allInsn;
00034
00040     public CodeExtract(File file) {
00041         setFile(file);
00042         loadPE(getFile());
00043     }
00044
00050     private void loadPE(File file)
00051     {
00052         peFile = new PEFile(file);
00053         peFile.readFile();
00054
00055     }
00056
00062     public byte[] loadInstructions()
00063     {
00064         long start = System.currentTimeMillis();
00065         byte[] bytes = peFile.getInstructions();
00066         System.out.println("time for loadInstruction:
"+(System.currentTimeMillis()-start));
00067         setInstructions(bytes);
00068         try
00069         {
00070             codeArr = extractArr(getInstructions());
00071         }
00072         catch(RuntimeException e)
00073         {
00074             e.printStackTrace();
00075         }
00076         return bytes;
00077     }
00078
00085     private Capstone.CsInsn[] loadCapstone(byte[] instructions)
00086     {
00087         file = new File(file.getAbsolutePath());
00088         @SuppressWarnings("unused")
00089         byte[] bytes = null;
00090         try {
00091             bytes = Files.readAllBytes(Paths.get(file.toString()));
00092         } catch (IOException | NullPointerException e) {
00093             e.printStackTrace();
00094         }
00095         Capstone cs = null;
00096         if(peFile.getVersion() == Version.x32)
00097         {
00098             cs = new Capstone(Capstone.CS_ARCH_X86, Capstone.CS_MODE_32);
00099             System.out.println("Running x32 exe");
00100         }
00101         else
00102         {
```

```
00103             cs = new Capstone(Capstone.CS_ARCH_X86, Capstone.CS_MODE_64);
00104             System.out.println("Running x64 exe");
00105         }
00106         cs.setDetail(1);
00107         long start = System.currentTimeMillis();
00108         Capstone.CsInsn[] allInsn = cs.disasm(instructions,
peFile.getPointer());
00109         System.out.println("dissasemble time: "+(System.currentTimeMillis()-
start));
00110         setAllInsn(allInsn);
00111         return allInsn;
00112     }
00113
00120     private String[] extractArr(byte[] instructions)
00121     {
00122         Capstone.CsInsn[] allInsn = loadCapstone(instructions);
00123         String[] code = new String[allInsn.length];
00124         for (int i=0; i<allInsn.length; i++)
00125         {
00126             code[i] = String.format("0x%x: %s %s\n", allInsn[i].address,
00127                   allInsn[i].mnemonic, allInsn[i].opStr);
00128         }
00129        return code;
00130     }
00131
00137     public File getFile() {
00138         return file;
00139     }
00140
00146     public void setFile(File file) {
00147         this.file = file;
00148     }
00149
00155     private void setInstructions(byte[] instructions) {
00156         this.instructions = instructions;
00157     }
00158
00164     public byte[] getInstructions() {
00165         return instructions;
00166     }
00167
00173     public String getCode() {
00174         return code;
00175     }
00176
00182     public String[] getCodeArr() {
00183         return codeArr;
00184     }
00185
00191     public PEFile getPeFile()
00192     {
00193         return peFile;
00194     }
00195
00201     public void setPeFile(PEFile peFile)
00202     {
00203         this.peFile = peFile;
00204     }
00205
00211     public int getPointer()
00212     {
00213         return peFile.getPointer();
00214     }
00215
00221     public Capstone.CsInsn[] getAllInsn()
00222     {
00223         return allInsn;
00224     }
00225
00231     public void setAllInsn(Capstone.CsInsn[] allInsn)
00232     {
00233         this.allInsn = allInsn;
00234     }
00235
00242     public byte[] getBytes() throws IOException
00243     {
00244         return peFile.getBytes();
```

```
00245     }
00246
00252     public Version getVersion()
00253     {
00254         return peFile.getVersion();
00255     }
00256
00262     @Override
00263     public String toString() {
00264         return "CodeExtract [file=" + file + ", instructions=" +
Arrays.toString(instructions) + ", code=" + code + "]";
00265     }
00266 }
```

CommandLine.java

```
00001 /*
00002  *
00003  */
00004 package dynamicAnalysis;
00005
00006 import java.io.BufferedReader;
00007 import java.io.IOException;
00008 import java.io.InputStreamReader;
00009
00013 public class CommandLine
00014 {
00015
00017     private long pid;
00018
00024     public CommandLine(long pid)
00025     {
00026         setPid(pid);
00027     }
00028
00032     public CommandLine()
00033     {
00034
00035     }
00036
00042     public long getPid()
00043     {
00044         return pid;
00045     }
00046
00052     public void setPid(long pid)
00053     {
00054         this.pid = pid;
00055     }
00056
00064     private String run(String command, boolean cmdCheck)
00065     {
00066         String response = "";
00067         ProcessBuilder builder;
00068         if(cmdCheck) builder = new ProcessBuilder("cmd.exe", "/c", command);
00069         else builder = new ProcessBuilder("cmd.exe", "/c", "cd /d
\""+System.getProperty("user.dir")+"\\lib\" && "+command);
00070         builder.redirectErrorStream(true);
00071         Process p = null;
00072         try {
00073             p = builder.start();
00074         } catch (IOException e) {
00075             e.printStackTrace();
00076         }
00077         BufferedReader r = new BufferedReader(new
InputStreamReader(p.getInputStream()));
00078         String line = "";
00079         while (true) {
00080             try {
00081                 line = r.readLine();
00082             } catch (IOException e) {
00083                 e.printStackTrace();
00084             }
00085             if (line == null) {
00086                 break;
00087             }
00088             response += line + "\n";
00089         }
00090         return response;
00091     }
00092
00098     //replace with listdlls
00099     public String runName()
00100     {
00101         return run("tasklist /fi \"pid eq "+getPid()+"\" /fo csv", true);
00102     }
00103
00109     public String runDLLs()
00110     {
```

```
00111         return run("listdlls "+getPid(), false);
00112     }
00113 
00119     public String runFiles()
00120     {
00121         return run("handle -p "+getPid()+" -v", false);
00122     }
00123 
00129     public String getAll()
00130     {
00131         return run("tasklist /fo csv", true);
00132     }
00133 
00139     public String getNetstat()
00140     {
00141         return run("netstat -ano", true);
00142     }
00143 
00149     @Override
00150     public String toString()
00151     {
00152         return "CommandLine [pid=" + pid + "]";
00153     }
00154 
00155 }
```

DataDirectory.java

```
00001 /*
00002  *
00003  */
00004 package dynamicAnalysis;
00005
00009 public class DataDirectory
00010 {
00011
00013     private byte[] bytes;
00014
00016     private int virtualAddress;
00017
00019     private int size;
00020
00028     public DataDirectory(byte[] bytes, int virtualAddress, int size)
00029     {
00030         setBytes(bytes);
00031         setVirtualAddress(virtualAddress);
00032         setSize(size);
00033     }
00034
00040     public byte[] getBytes()
00041     {
00042         return bytes;
00043     }
00044
00050     public void setBytes(byte[] bytes)
00051     {
00052         this.bytes = bytes;
00053     }
00054
00060     public int getVirtualAddress()
00061     {
00062         return virtualAddress;
00063     }
00064
00070     public void setVirtualAddress(int virtualAddress)
00071     {
00072         this.virtualAddress = virtualAddress;
00073     }
00074
00080     public int getSize()
00081     {
00082         return size;
00083     }
00084
00090     public void setSize(int size)
00091     {
00092         this.size = size;
00093     }
00094
00095
00096 }
```

Details.java

```
00001 /*
00002  *
00003  */
00004 package dynamicAnalysis;
00005
00006 import org.eclipse.swt.widgets.Composite;
00007 import org.eclipse.swt.widgets.Table;
00008 import org.eclipse.swt.SWT;
00009 import org.eclipse.swt.widgets.Button;
00010 import org.eclipse.swt.layout.FormLayout;
00011 import org.eclipse.swt.layout.FormData;
00012 import org.eclipse.swt.layout.FormAttachment;
00013 import org.eclipse.swt.widgets.TableItem;
00014 import org.eclipse.swt.widgets.TableColumn;
00015 import org.eclipse.wb.swt.SWTResourceManager;
00016
00020 public class Details extends Composite
00021 {
00022
00024     private Table table;
00025
00027     private TableItem tableItems[] = new TableItem[4];
00028
00030     private boolean selection;
00031
00033     private Button btnInstructions = new Button(this, SWT.NONE);
00034
00036     private Button btnMemory = new Button(this, SWT.NONE);
00037
00039     private Button btnAdvanced = new Button(this, SWT.NONE);
00040
00048     public Details(Composite parent, int style, boolean selection)
00049     {
00050         super(parent, style);
00051         setLayout(new FormLayout());
00052
00053
00054         FormData fd_btnInstructions = new FormData();
00055         fd_btnInstructions.right = new FormAttachment(100, -27);
00056         fd_btnInstructions.bottom = new FormAttachment(table, 25, SWT.TOP);
00057         fd_btnInstructions.top = new FormAttachment(table, 0, SWT.TOP);
00058         btnInstructions.setLayoutData(fd_btnInstructions);
00059
btnInstructions.setForeground(SWTResourceManager.getColor(SWT.COLOR_WIDGET_LIGHT_SHADO
W));
00060         btnInstructions.setText("x86 Instructions");
00061
00062
00063         FormData fd_btnMemory = new FormData();
00064         fd_btnMemory.right = new FormAttachment(100, -27);
00065         fd_btnMemory.left = new FormAttachment(0, 191);
00066         fd_btnMemory.top = new FormAttachment(btnInstructions, 10);
00067
00068         btnMemory.setLayoutData(fd_btnMemory);
00069         btnMemory.setText("Virtual Memory");
00070
00071
00072         FormData fd_btnAdvanced = new FormData();
00073
00074         fd_btnAdvanced.left = new FormAttachment(0, 191);
00075         fd_btnAdvanced.top = new FormAttachment(table, -25, SWT.BOTTOM);
00076         fd_btnAdvanced.bottom = new FormAttachment(table, 0, SWT.BOTTOM);
00077         btnAdvanced.setLayoutData(fd_btnAdvanced);
00078         btnAdvanced.setText("Advanced");
00079
00080         table = new Table(this, SWT.FULL_SELECTION | SWT.NO_SCROLL);
00081         fd_btnAdvanced.bottom = new FormAttachment(table, 0, SWT.BOTTOM);
00082         fd_btnAdvanced.top = new FormAttachment(btnMemory, 10, SWT.BOTTOM);
00083         FormData fd_table = new FormData();
00084         fd_table.left = new FormAttachment(btnInstructions, -191, SWT.LEFT);
00085         fd_table.right = new FormAttachment(50, 20);
00086         fd_table.top = new FormAttachment(0);
00087         fd_table.bottom = new FormAttachment(100, -20);
```

```
00088         table.setLayoutData(fd_table);
00089         table.setLinesVisible(true);
00090         table.setHeaderVisible(false);
00091 
00092         fd_btnAdvanced.top = new FormAttachment(table, -25, SWT.BOTTOM);
00093         fd_btnAdvanced.right = new FormAttachment(btnInstructions, 0,
SWT.RIGHT);
00094         fd_btnMemory.right = new FormAttachment(btnInstructions, 0, SWT.RIGHT);
00095         fd_btnInstructions.right = new FormAttachment(table, 100, SWT.RIGHT);
00096         fd_btnInstructions.left = new FormAttachment(table, 5, SWT.RIGHT);
00097 
00098         TableColumn labels = new TableColumn(table, SWT.NONE);
00099         labels.setWidth(100);
00100 
00101         TableColumn values = new TableColumn(table, SWT.CENTER | SWT.V_SCROLL);
00102         values.setWidth(150);
00103 
00104 
00105 
00106         for(int index = 0;index<tableItems.length;index++)
00107         {
00108             tableItems[index] = new TableItem(table, SWT.NONE);
00109         }
00110 
00111         tableItems[0].setText(0, "Directory");
00112         tableItems[1].setText(0, "Version");
00113         tableItems[2].setText(0, "Name");
00114         tableItems[3].setText(0, "PID");
00115 
00116     }
00117 
00121     public void clearData()
00122     {
00123         tableItems[0].setText(1, "");
00124         tableItems[1].setText(1, "");
00125         tableItems[2].setText(1, "");
00126         tableItems[3].setText(1, "");
00127         btnMemory.setEnabled(true);
00128         btnAdvanced.setEnabled(true);
00129         tableItems[0].setGrayed(isSelection());
00130         tableItems[1].setGrayed(isSelection());
00131         btnInstructions.setEnabled(!isSelection());
00132     }
00133 
00139     public boolean isSelection()
00140     {
00141         return selection;
00142     }
00143 
00149     public void setSelection(boolean selection)
00150     {
00151         this.selection = selection;
00152     }
00153 
00157     @Override
00158     protected void checkSubclass()
00159     {
00160         // Disable the check that prevents subclassing of SWT components
00161     }
00162 }
```

DllFile.java

```
00001 /*
00002  *
00003  */
00004 package dynamicAnalysis;
00005
00006 import java.io.File;
00007
00011 public class DllFile
00012 {
00013
00015     private String path;
00016
00018     private File file;
00019
00025     public DllFile(String path)
00026     {
00027         setPath(path);
00028         file = createFile();
00029     }
00030
00036     public String getPath()
00037     {
00038         return path;
00039     }
00040
00046     public void setPath(String path)
00047     {
00048         this.path = path;
00049     }
00050
00056     public File getFile()
00057     {
00058         return file;
00059     }
00060
00066     private File createFile()
00067     {
00068         return new File(getPath());
00069     }
00070
00076     @Override
00077     public String toString()
00078     {
00079         return "DllFile [path=" + path + ", file=" + file + "]";
00080     }
00081
00082 }
```

ExecuteCode.java

```
00001 /*
00002  *
00003  */
00004 package dynamicAnalysis;
00005
00006 import java.io.File;
00007 import java.util.Arrays;
00008
00012 public class ExecuteCode {
00013
00015     private byte[] codes;
00016
00018     private byte code;
00019
00021     private boolean isArr;
00022
00024     private File file;
00025
00031     private native void executeInstruction(byte code);
00032
00038     private native int[] readRegisters();
00039
00040     static {System.load(getFile("execute.dll"));}
00041
00048     public ExecuteCode(byte[] codes, File file) {
00049         setCodes(codes);
00050         setFile(file);
00051         isArr=true;
00052     }
00053
00060     public ExecuteCode(byte code, File file) {
00061         setCode(code);
00062         setFile(file);
00063         isArr=false;
00064     }
00065
00071     public File getFile()
00072     {
00073         return file;
00074     }
00075
00081     public void setFile(File file)
00082     {
00083         this.file = file;
00084     }
00085
00091     public byte[] getCodes() {
00092         return codes;
00093     }
00094
00100     public void setCodes(byte[] codes) {
00101         this.codes = codes;
00102     }
00103
00109     public byte getCode() {
00110         return code;
00111     }
00112
00118     public void setCode(byte code) {
00119         this.code = code;
00120     }
00121
00128     private static String getFile(String fileName)
00129     {
00130         return
System.getProperty("user.dir")+"\\src\\dynamicAnalysis\\"+fileName;
00131     }
00132
00138     public native String test();
00139
00140
00141     /*public ProcessManager execute()
00142     {
```

```
00143         ExecuteCode executeCode = new ExecuteCode(getCode(), getFile());
00144         executeCode.executeInstruction(getCode());
00145     }*/
00151     public int[] read()
00152     {
00153         int[] registers = new int[4];
00154         ExecuteCode executeCode = new ExecuteCode(getCode(), getFile());
00155         registers = executeCode.readRegisters();
00156         return registers;
00157     }
00158 
00164     @Override
00165     public String toString()
00166     {
00167         return "ExecuteCode [codes=" + Arrays.toString(codes) + ", code=" + code
+ ", isArr=" + isArr + ", file=" + file
00168                 + "]";
00169     }
00170 
00171 }
00172  //gcc ExecuteImpl.c -I I:/jre/jre6/include -I I:/jre/jre6/include/win32
00173 //x86_64-w64-mingw32-gcc -I I:/jre/jre6/include -I I:/jre/jre6/include/win32 -
shared -o execute.dll ExecuteImpl.c
```

FilesComposite.java

```
00001 package dynamicAnalysis;
00002
00003 import org.eclipse.swt.widgets.Composite;
00004 import org.eclipse.swt.layout.GridLayout;
00005 import org.eclipse.swt.widgets.Table;
00006 import org.eclipse.swt.widgets.TableItem;
00007 import org.eclipse.swt.SWT;
00008 import org.eclipse.swt.widgets.Label;
00009 import org.eclipse.swt.layout.GridData;
00010 import org.eclipse.swt.layout.FormLayout;
00011 import org.eclipse.swt.layout.FormData;
00012 import org.eclipse.swt.layout.FormAttachment;
00013
00017 public class FilesComposite extends Composite
00018 {
00019
00021     private Table dllsTable;
00022
00024     private Table filesTable;
00025
00027     private int processId;
00028
00035     public FilesComposite(Composite parent, int style)
00036     {
00037         super(parent, style);
00038         setProcessId(Window.processId);
00039         setLayout(new GridLayout(2, false));
00040
00041         dllsTable = new Table(this, SWT.BORDER | SWT.FULL_SELECTION);
00042         GridData gd_dllsTable = new GridData(SWT.FILL, SWT.FILL, false, true, 1,
1);
00043         gd_dllsTable.widthHint = 222;
00044         dllsTable.setLayoutData(gd_dllsTable);
00045         dllsTable.setHeaderVisible(true);
00046         dllsTable.setLinesVisible(true);
00047
00048         filesTable = new Table(this, SWT.BORDER | SWT.FULL_SELECTION);
00049         filesTable.setLayoutData(new GridData(SWT.FILL, SWT.FILL, true, true, 1,
1));
00050         filesTable.setHeaderVisible(true);
00051         filesTable.setLinesVisible(true);
00052
00053         ProcessManager processManager = new ProcessManager(getProcessId());
00054         String[] DLLs = processManager.getDLLs();
00055         for(int index=0;index<DLLs.length;index++)
00056         {
00057             TableItem tableItem = new TableItem(dllsTable, SWT.NULL);
00058             tableItem.setText(DLLs[index]);
00059         }
00060         String files[] = processManager.getFiles();
00061         System.out.println("files: "+files.length);
00062         for(int index=0;index<files.length;index++)
00063         {
00064             TableItem tableItem = new TableItem(filesTable, SWT.NULL);
00065             System.out.println("files: "+files[index]);
00066             tableItem.setText(files[index]);
00067         }
00068
00069     }
00070
00071
00072
00078     public int getProcessId()
00079     {
00080         return processId;
00081     }
00082
00088     public void setProcessId(int processId)
00089     {
00090         this.processId = processId;
00091     }
00092
00096     @Override
```

```
00097     protected void checkSubclass()
00098     {
00099         // Disable the check that prevents subclassing of SWT components
00100     }
00101 }
```

InstructionsComposite.java

```
00001 /*
00002  *
00003  */
00004 package dynamicAnalysis;
00005
00006 import org.eclipse.swt.widgets.Composite;
00007 import org.eclipse.swt.widgets.Control;
00008 import org.eclipse.swt.widgets.Display;
00009 import org.eclipse.swt.widgets.Event;
00010 import org.eclipse.swt.widgets.FileDialog;
00011 import org.eclipse.swt.widgets.Listener;
00012 import org.eclipse.swt.layout.FormLayout;
00013 import org.eclipse.swt.widgets.Table;
00014
00015 import java.io.File;
00016 import java.io.FileNotFoundException;
00017 import java.io.FileOutputStream;
00018 import java.io.IOException;
00019 import java.util.ArrayList;
00020
00021 import org.eclipse.swt.SWT;
00022 import org.eclipse.swt.layout.FormData;
00023 import org.eclipse.swt.layout.FormAttachment;
00024 import org.eclipse.swt.widgets.TableColumn;
00025 import org.eclipse.swt.widgets.TableItem;
00026
00027 import capstone.Capstone;
00028 import org.eclipse.swt.widgets.Button;
00029 import org.eclipse.swt.events.SelectionAdapter;
00030 import org.eclipse.swt.events.SelectionEvent;
00031 import org.eclipse.swt.graphics.Color;
00032 import org.eclipse.swt.custom.CCombo;
00033 import org.eclipse.swt.widgets.Text;
00034
00038 public class InstructionsComposite extends Composite
00039 {
00040
00042     private Table tableInstructions;
00043
00045     private Capstone.CsInsn[] allInsn;
00046
00048     private Text textOpcode;
00049
00051     private TableColumn tblclmnAddress;
00052
00054     private TableColumn tblclmnMnemonic;
00055
00057     private TableColumn tblclmnOpcode;
00058
00060     private byte[] updatedInstructions;
00061
00063     private Composite parent;
00064
00066     private TableItem[] tableItems = null;
00067
00075     public InstructionsComposite(Composite parent, int style, File file)
00076     {
00077         super(parent, style);
00078         this.parent = parent;
00079         setLayout(new FormLayout());
00080
00081         CodeExtract codeExtract = new CodeExtract(file);
00082         codeExtract.loadInstructions();
00083         allInsn = codeExtract.getAllInsn();
00084         setAllInsn(allInsn);
00085
00086         tableInstructions = new Table(this, SWT.BORDER | SWT.FULL_SELECTION |
SWT.VIRTUAL);
00087         FormData fd_tableInstructions = new FormData();
00088         fd_tableInstructions.top = new FormAttachment(0, 10);
00089         fd_tableInstructions.left = new FormAttachment(0, 10);
00090         fd_tableInstructions.right = new FormAttachment(0, 353);
00091         tableInstructions.setLayoutData(fd_tableInstructions);
```

```
00092         tableInstructions.setHeaderVisible(true);
00093         tableInstructions.setLinesVisible(true);
00094         tableInstructions.setItemCount(getAllInsn().length);
00095 
00096         Text comboMnemonic = new Text(this, SWT.BORDER);
00097 
00098         tableInstructions.addListener(SWT.Selection, new Listener() {
00099               public void handleEvent(Event e) {
00100                 Capstone.CsInsn instruction =
getAllInsn()[tableInstructions.getSelectionIndex()];
00101                 if(tableItems!=null) clearColors(tableItems);
00102                 tableItems = null;
00103                 if(instruction.groups.length == 2)
00104                 {
00105                     if(instruction.groups[0] == 7 && instruction.groups[1] == 1)
//group for jump instructions, jmp works differently ?
00106                     {
00107                         long entryIndex = tableInstructions.getSelectionIndex();
00108                         long operandIndex = instruction.bytes[1] + entryIndex;
00109                         boolean lowerOperand = instruction.bytes[1] < 0;
00110                         System.out.println("lower operand: "+lowerOperand);
00111                         System.out.println("entry index" + entryIndex);
00112                         System.out.println("operand index " + operandIndex );
00113                         if(lowerOperand) entryIndex--;
00114                         else entryIndex++;
00115                         System.out.println("instruction addr:
"+Long.decode(instruction.opStr));
00116                         tableItems = new
TableItem[Math.abs(instruction.bytes[1])];
00117                         int tableIndex = 0;
00118                         while(Long.decode(instruction.opStr) !=
((Capstone.CsInsn)(tableInstructions.getItem((int) entryIndex).getData())).address)
00119                         {
00120                             System.out.println(entryIndex);
00121                             try
00122                             {
00123                                 tableItems =
setGray(tableInstructions.getItem((int) entryIndex), tableItems, tableIndex);
00124                                 tableIndex++;
00125                             }
00126                             catch(ArrayIndexOutOfBoundsException e1)
00127                             {
00128                                 e1.printStackTrace();
00129                                 clearColors(tableItems);
00130                             }
00131                             if(lowerOperand) entryIndex--;
00132                             else entryIndex++;
00133                         }
00134                         tableItems = setGreen(tableInstructions.getItem((int)
entryIndex), tableItems, tableIndex);
00135                     }
00136                 }
00137                 TableItem item = tableInstructions.getSelection()[0];
00138                 System.out.println(item.getText());
00139                 //comboMnemonic.setText(Integer.toString(instruction.bytes[0] &
0xff));
00140                 comboMnemonic.setText(Byte.toString(instruction.bytes[0]));
00141                 String opcode = "";
00142                 for(int index = 1;index<instruction.bytes.length;index++)
00143                 {
00144                     //opcode+=Integer.toString(instruction.bytes[index] &
0xff)+" ";
00145                     opcode+=Byte.toString(instruction.bytes[index])+" ";
00146                 }
00147                 if(opcode.length()!=0) textOpcode.setText(opcode.substring(0,
opcode.length()-1));
00148                 else textOpcode.setText(opcode);
00149 
00150               }
00151         });
00152 
00153         tblclmnAddress = new TableColumn(tableInstructions, SWT.NONE);
00154         tblclmnAddress.setWidth(100);
00155         tblclmnAddress.setText("Address");
00156 
00157         tblclmnMnemonic = new TableColumn(tableInstructions, SWT.NONE);
00158         tblclmnMnemonic.setWidth(100);
```

```
00159         tblclmnMnemonic.setText("Mnemonic");
00160
00161         tblclmnOpcode = new TableColumn(tableInstructions, SWT.NONE);
00162         tblclmnOpcode.setWidth(5000);
00163         tblclmnOpcode.setText("Opcode");
00164
00165         Button btnSave = new Button(this, SWT.NONE);
00166         btnSave.addSelectionListener(new SelectionAdapter() {
00167             @Override
00168             public void widgetSelected(SelectionEvent e) {
00169                 byte[] bytes = null;
00170                 try
00171                 {
00172                     bytes = codeExtract.getBytes();
00173                 } catch (IOException e2)
00174                 {
00175                     e2.printStackTrace();
00176                 }
00177                 int bytesIndex = codeExtract.getPointer();
00178                 for(int index=0;index<tableInstructions.getItemCount();index++)
00179                 {
00180                     Capstone.CsInsn instruction = (Capstone.CsInsn)
tableInstructions.getItem(index).getData();
00181                     for(int j = 0; j < instruction.bytes.length; j++)
00182                     {
00183                         bytes[bytesIndex] = instruction.bytes[j];
00184                         bytesIndex++;
00185                     }
00186                 }
00187                 FileDialog fileDialog = new FileDialog(parent.getShell(),
SWT.MULTI);
00188                 String[] files = {
00189                         "*.exe",
00190                     };
00191                     fileDialog.setFilterExtensions(files);
00192                     fileDialog.setFilterPath(file.getPath());
00193
fileDialog.setFileName(codeExtract.getFile().getName().substring(0,
codeExtract.getFile().getName().length()-4)+"_1.exe");
00194                 try (FileOutputStream fos = new
FileOutputStream(fileDialog.open())) {
00195                        fos.write(bytes);
00196                     } catch (FileNotFoundException e1)
00197                     {
00198                         e1.printStackTrace();
00199                     } catch (IOException e1)
00200                     {
00201                         e1.printStackTrace();
00202                     }
00203             }
00204         });
00205         fd_tableInstructions.bottom = new FormAttachment(100, -41);
00206         fd_tableInstructions.right = new FormAttachment(btnSave, -20);
00207         FormData fd_btnSave = new FormData();
00208         fd_btnSave.right = new FormAttachment(100, -10);
00209         fd_btnSave.top = new FormAttachment(0, 10);
00210         btnSave.setLayoutData(fd_btnSave);
00211         btnSave.setText("Save");
00212
00213         fillTable();
00214         tableInstructions.pack();
00215
00216
00217         FormData fd_comboMnemonic = new FormData();
00218         fd_comboMnemonic.top = new FormAttachment(tableInstructions, 6);
00219         fd_comboMnemonic.left = new FormAttachment(0, 109);
00220         fd_comboMnemonic.right = new FormAttachment(0, 207);
00221         comboMnemonic.setLayoutData(fd_comboMnemonic);
00222
00223         textOpcode = new Text(this, SWT.BORDER);
00224         FormData fd_textOpcode = new FormData();
00225         fd_textOpcode.left = new FormAttachment(comboMnemonic, 6);
00226         fd_textOpcode.top = new FormAttachment(tableInstructions, 6);
00227         textOpcode.setLayoutData(fd_textOpcode);
00228
00229         Button btnNewButton = new Button(this, SWT.NONE);
00230         btnNewButton.addSelectionListener(new SelectionAdapter() {
```

```
00231             @Override
00232             public void widgetSelected(SelectionEvent e) {
00233                 String opcodeInstructions = textOpcode.getText();
00234                 int count = 2;
00235                 for(int index = 0;index<opcodeInstructions.length();index++)
00236                 {
00237                     if(opcodeInstructions.charAt(index)==' ')
00238                     {
00239                         count++;
00240                     }
00241                 }
00242                 if(opcodeInstructions.equals("")) count=1;
00243                 byte[] updatedInstruction = new byte[count];
00244                 updatedInstruction[0]=Byte.parseByte(comboMnemonic.getText());
00245                 Capstone cs;
00246                 if(codeExtract.getVersion() == Version.x32)
00247                 {
00248                     cs = new Capstone(Capstone.CS_ARCH_X86,
Capstone.CS_MODE_32);
00249                 }
00250                 else
00251                 {
00252                     cs = new Capstone(Capstone.CS_ARCH_X86,
Capstone.CS_MODE_64);
00253                 }
00254                 if(!opcodeInstructions.equals(""))
00255                 {
00256                     int instructionIndex = 1;
00257                     while(opcodeInstructions.contains(" "))
00258                     {
00259
updatedInstruction[instructionIndex]=Byte.parseByte(opcodeInstructions.substring(0,
opcodeInstructions.indexOf(' ')));
00260                         opcodeInstructions =
opcodeInstructions.substring(opcodeInstructions.indexOf(' ')+1);
00261                         instructionIndex++;
00262                     }
00263
updatedInstruction[instructionIndex]=Byte.parseByte(opcodeInstructions);
00264                 }
00265                 Capstone.CsInsn[] instruction = cs.disasm(updatedInstruction,
((Capstone.CsInsn)tableInstructions.getSelection()[0].getData()).address);
00266                 tableInstructions.getSelection()[0].setText(0,
"0x"+Long.toHexString(instruction[0].address));
00267                 tableInstructions.getSelection()[0].setText(1,
instruction[0].mnemonic);
00268                 tableInstructions.getSelection()[0].setText(2,
instruction[0].opStr);
00269                 tableInstructions.getSelection()[0].setData(instruction[0]);
00270
System.out.println(((Capstone.CsInsn)tableInstructions.getSelection()[0].getData()).op
Str);
00271             }
00272         });
00273         fd_textOpcode.right = new FormAttachment(100, -143);
00274         FormData fd_btnNewButton = new FormData();
00275         fd_btnNewButton.bottom = new FormAttachment(comboMnemonic, 0,
SWT.BOTTOM);
00276         fd_btnNewButton.top = new FormAttachment(tableInstructions, 6);
00277         fd_btnNewButton.right = new FormAttachment(tableInstructions, 0,
SWT.RIGHT);
00278         fd_btnNewButton.left = new FormAttachment(textOpcode, 28);
00279         btnNewButton.setLayoutData(fd_btnNewButton);
00280         btnNewButton.setText("Replace");
00281
00282     }
00283
00287     private void fillTable()
00288     {
00289         new Thread() {
00290             public void run() {
00291                 long start = System.currentTimeMillis();
00292                 Capstone.CsInsn[] allInsn = getAllInsn();
00293                 System.out.println("time for capstone load:
"+(System.currentTimeMillis()-start));
00294                 TableItem[] tableItems = new TableItem[allInsn.length];
00295                 Display.getDefault().asyncExec(new Runnable() {
```

```
00296                     @Override
00297                     public void run() {
00298
00299                         for(int index=0;index<allInsn.length;index++)
00300                         {
00301                             tableInstructions.getItem(index).setText(0,
"0x"+Long.toHexString(allInsn[index].address));
00302                             tableInstructions.getItem(index).setText(1,
allInsn[index].mnemonic);
00303                             tableInstructions.getItem(index).setText(2,
allInsn[index].opStr);
00304
tableInstructions.getItem(index).setData(allInsn[index]);
00305                         }
00306                     }
00307                 });
00308             }
00309         }.start();
00310
00311
00312     }
00313
00323     private TableItem[] setColor(TableItem item, Color color, TableItem[] items,
int index)
00324     {
00325         items[index] = item;
00326         item.setBackground(color);
00327         return items;
00328     }
00329
00336     private void setColor(TableItem item, Color color)
00337     {
00338         item.setBackground(color);
00339     }
00340
00349     private TableItem[] setGray(TableItem item, TableItem[] items, int index)
00350     {
00351         return setColor(item, new Color(parent.getDisplay(), 200, 200, 200),
items, index);
00352     }
00353
00362     private TableItem[] setGreen(TableItem item, TableItem[] items, int index)
00363     {
00364         return setColor(item, new Color(parent.getDisplay(), 0, 255, 0), items,
index);
00365     }
00366
00372     private void clearColors(TableItem[] items)
00373     {
00374         for(int index = 0;index < items.length;index++)
00375         {
00376             try
00377             {
00378                 System.out.println("data: "+items[index]);
00379                 setColor(items[index], new Color(parent.getDisplay(), 255, 255,
255));
00380             }
00381             catch(NullPointerException e)
00382             {
00383                 break;
00384             }
00385         }
00386     }
00387
00391     @Override
00392     public void layout()
00393     {
00394         System.out.println("printing layout");
00395     }
00396
00397
00403     public Capstone.CsInsn[] getAllInsn()
00404     {
00405         return allInsn;
00406     }
00407
00413     public void setAllInsn(Capstone.CsInsn[] allInsn)
```

```
00414     {
00415         this.allInsn = allInsn;
00416     }
00417 
00421     @Override
00422     protected void checkSubclass()
00423     {
00424         // Disable the check that prevents subclassing of SWT components
00425     }
00426 }
```

LegacyWindow.java

```
00001 /*
00002  *
00003  */
00004 package dynamicAnalysis;
00005
00006 import org.eclipse.swt.widgets.Display;
00007 import org.eclipse.swt.widgets.Event;
00008 import org.eclipse.swt.widgets.Shell;
00009 import org.eclipse.swt.widgets.Menu;
00010 import org.eclipse.swt.SWT;
00011 import org.eclipse.swt.widgets.MenuItem;
00012 import org.eclipse.swt.widgets.Label;
00013 import org.eclipse.swt.widgets.Listener;
00014 import org.eclipse.wb.swt.SWTResourceManager;
00015
00016
00017 import org.eclipse.swt.widgets.Table;
00018 import org.eclipse.swt.widgets.TableItem;
00019
00020 import java.awt.Dimension;
00021 import java.awt.Toolkit;
00022 import java.io.File;
00023
00024 import org.eclipse.jface.viewers.TableViewer;
00025 import org.eclipse.swt.layout.GridLayout;
00026 import org.eclipse.swt.layout.GridData;
00027 import org.eclipse.swt.widgets.TableColumn;
00028 import org.eclipse.swt.events.SelectionAdapter;
00029 import org.eclipse.swt.events.SelectionEvent;
00030
00031 import javax.sound.sampled.*;
00032
00036 public class LegacyWindow {
00037
00039     protected Shell shell;
00040
00042     private Table code;
00043
00045     private Table details;
00046
00048     private Table dllImports;
00049
00051     private String filePath;
00052
00054     static TableItem tableItems[] = new TableItem[4];
00055
00061     public static void main(String[] args) {
00062         try {
00063             LegacyWindow window = new LegacyWindow();
00064             window.open();
00065         } catch (Exception e) {
00066             e.printStackTrace();
00067         }
00068
00069     }
00070
00074     public void open() {
00075         Display display = Display.getDefault();
00076         createContents();
00077         shell.open();
00078         shell.layout();
00079         while (!shell.isDisposed()) {
00080             if (!display.readAndDispatch()) {
00081                 display.sleep();
00082             }
00083         }
00084     }
00085
00087     public static float SAMPLE_RATE = 8000f;
00088
00096     public void tone(int hz, int msecs) throws LineUnavailableException
00097       {
00098          tone(hz, msecs, 1.0);
```

```
00099       }
00100
00109     public static void tone(int hz, int msecs, double vol)
00110           throws LineUnavailableException
00111       {
00112         byte[] buf = new byte[1];
00113         AudioFormat af =
00114             new AudioFormat(
00115                 SAMPLE_RATE, // sampleRate
00116                 8,           // sampleSizeInBits
00117                 1,           // channels
00118                 true,        // signed
00119                 false);      // bigEndian
00120         SourceDataLine sdl = AudioSystem.getSourceDataLine(af);
00121         sdl.open(af);
00122         sdl.start();
00123         for (int i=0; i < msecs*8; i++) {
00124           double angle = i / (SAMPLE_RATE / hz) * 2.0 * Math.PI;
00125           buf[0] = (byte)(Math.sin(angle) * 127.0 * vol);
00126           sdl.write(buf,0,1);
00127         }
00128         sdl.drain();
00129         sdl.stop();
00130         sdl.close();
00131       }
00132
00136     protected void createContents() {
00137         shell = new Shell();
00138         shell.setBackground(SWTResourceManager.getColor(192, 192, 192));
00139         shell.setSize(881, 520);
00140         shell.setText("Dynamic Malware Analysis");
00141
00142         Dimension dim = Toolkit.getDefaultToolkit().getScreenSize();
00143         shell.setLocation((dim.width/2)-400,(dim.height/2)-200);
00144
00145         GridLayout gl_shell = new GridLayout(9, false);
00146         gl_shell.marginBottom = 15;
00147         shell.setLayout(gl_shell);
00148
00149         Menu menu = new Menu(shell, SWT.BAR);
00150         shell.setMenuBar(menu);
00151
00152         MenuItem mntmFile_1 = new MenuItem(menu, SWT.CASCADE);
00153         mntmFile_1.setText("File");
00154
00155         Menu menu_1 = new Menu(mntmFile_1);
00156         mntmFile_1.setMenu(menu_1);
00157
00158         MenuItem mntmEdit = new MenuItem(menu, SWT.NONE);
00159         mntmEdit.setText("Edit");
00160
00161         MenuItem mntmView = new MenuItem(menu, SWT.CASCADE);
00162         mntmView.setText("View");
00163
00164         Menu menu_2 = new Menu(mntmView);
00165         mntmView.setMenu(menu_2);
00166
00167         MenuItem mntmVirtualMemory = new MenuItem(menu_2, SWT.NONE);
00168         mntmVirtualMemory.setText("Virtual Memory");
00169
00170         mntmVirtualMemory.addListener(SWT.Selection, new Listener() {
00171             public void handleEvent(Event e) {
00172                 System.out.println(tableItems[3].getText(1));
00173                 try
00174                 {
00175                     MemoryWindow memoryWindow = new
MemoryWindow(Integer.parseInt(tableItems[3].getText(1)), shell.getLocation().x,
shell.getLocation().y);
00176                     memoryWindow.open();
00177                 }
00178                 catch(NumberFormatException e1)
00179                 {
00180
00181                 }
00182             }
00183         });
00184
```

```
00185         MenuItem mntmHelp = new MenuItem(menu, SWT.NONE);
00186         mntmHelp.setText("Help");
00187         new Label(shell, SWT.NONE);
00188         new Label(shell, SWT.NONE);
00189 
00190         MenuItem mntmOpen = new MenuItem(menu_1, SWT.NONE);
00191         mntmOpen.setText("Open");
00192 
00193         MenuItem mntmRun = new MenuItem(menu, SWT.NONE);
00194         mntmRun.setText("Run");
00195         mntmRun.addSelectionListener(new SelectionAdapter() {
00196             @Override
00197             public void widgetSelected(SelectionEvent e) {
00198                 try
00199                 {
00200                     ProcessManager process = new ProcessManager(new
File(filePath));
00201                     tableItems[2].setText(1, process.getName());
00202                     tableItems[3].setText(1, process.getPidAsString());
00203                     String[] DLLs = process.getDLLs();
00204                     dllImports.clearAll();
00205                     dllImports.setItemCount(0);
00206                     for(int index=0;index<DLLs.length;index++)
00207                     {
00208                         TableItem tableItem = new TableItem(dllImports,
SWT.NULL);
00209                         tableItem.setText(DLLs[index]);
00210                     }
00211                 }
00212                 catch(NullPointerException e1)
00213                 {
00214                     System.out.println("No file selected");
00215                 }
00216             }
00217         });
00218         new Label(shell, SWT.NONE);
00219         new Label(shell, SWT.NONE);
00220         new Label(shell, SWT.NONE);
00221         new Label(shell, SWT.NONE);
00222         new Label(shell, SWT.NONE);
00223         new Label(shell, SWT.NONE);
00224         new Label(shell, SWT.NONE);
00225         new Label(shell, SWT.NONE);
00226 
00227         Label lblCode = new Label(shell, SWT.NONE);
00228         lblCode.setBackground(SWTResourceManager.getColor(192, 192, 192));
00229         lblCode.setLayoutData(new GridData(SWT.CENTER, SWT.CENTER, true, false,
1, 1));
00230         lblCode.setAlignment(SWT.RIGHT);
00231         lblCode.setFont(SWTResourceManager.getFont("Segoe UI", 11, SWT.BOLD));
00232         lblCode.setText("Code");
00233         new Label(shell, SWT.NONE);
00234         new Label(shell, SWT.NONE);
00235         new Label(shell, SWT.NONE);
00236 
00237         Label lblDetails = new Label(shell, SWT.NONE);
00238         lblDetails.setBackground(SWTResourceManager.getColor(192, 192, 192));
00239         GridData gd_lblDetails = new GridData(SWT.CENTER, SWT.CENTER, true,
false, 1, 1);
00240         gd_lblDetails.widthHint = 82;
00241         lblDetails.setLayoutData(gd_lblDetails);
00242         lblDetails.setText("Details");
00243         lblDetails.setFont(SWTResourceManager.getFont("Segoe UI", 11,
SWT.BOLD));
00244         lblDetails.setAlignment(SWT.CENTER);
00245         new Label(shell, SWT.NONE);
00246 
00247         Label lblDllImports = new Label(shell, SWT.NONE);
00248         lblDllImports.setBackground(SWTResourceManager.getColor(192, 192, 192));
00249         lblDllImports.setLayoutData(new GridData(SWT.CENTER, SWT.CENTER, true,
false, 1, 1));
00250         lblDllImports.setText("DLL Imports");
00251         lblDllImports.setFont(SWTResourceManager.getFont("Segoe UI", 11,
SWT.BOLD));
00252         lblDllImports.setAlignment(SWT.CENTER);
00253         new Label(shell, SWT.NONE);
00254         new Label(shell, SWT.NONE);
```

```
00255
00256         TableViewer tableViewer = new TableViewer(shell, SWT.BORDER |
SWT.FULL_SELECTION | SWT.V_SCROLL);
00257         code = tableViewer.getTable();
00258         code.setToolTipText("x86 instructions");
00259         code.setBackground(SWTResourceManager.getColor(192, 192, 192));
00260         code.setHeaderBackground(SWTResourceManager.getColor(192, 192, 192));
00261         GridData gd_code = new GridData(SWT.FILL, SWT.FILL, true, true, 1, 1);
00262         gd_code.widthHint = 121;
00263         gd_code.heightHint = 313;
00264         code.setLayoutData(gd_code);
00265         new Label(shell, SWT.NONE);
00266         new Label(shell, SWT.NONE);
00267         new Label(shell, SWT.NONE);
00268
00269         details = new Table(shell, SWT.BORDER | SWT.FULL_SELECTION);
00270         details.setToolTipText("File information");
00271         details.setHeaderBackground(SWTResourceManager.getColor(192, 192, 192));
00272         details.setBackground(SWTResourceManager.getColor(192, 192, 192));
00273         GridData gd_details = new GridData(SWT.FILL, SWT.FILL, false, false, 1,
1);
00274         gd_details.widthHint = 316;
00275         details.setLayoutData(gd_details);
00276         details.setHeaderVisible(true);
00277         details.pack();
00278
00279         TableColumn labels = new TableColumn(details, SWT.CENTER);
00280         labels.setWidth(160);
00281         labels.setText("Labels");
00282
00283         TableColumn values = new TableColumn(details, SWT.CENTER |
SWT.V_SCROLL);
00284         values.setWidth(170);
00285         values.setText("Values");
00286         new Label(shell, SWT.NONE);
00287
00288         dllImports = new Table(shell, SWT.BORDER | SWT.FULL_SELECTION |
SWT.V_SCROLL);
00289         dllImports.setToolTipText("DLL file paths");
00290         dllImports.setHeaderBackground(SWTResourceManager.getColor(192, 192,
192));
00291         dllImports.setBackground(SWTResourceManager.getColor(192, 192, 192));
00292         GridData gd_dllImports = new GridData(SWT.FILL, SWT.FILL, true, false,
1, 1);
00293         gd_dllImports.widthHint = 200;
00294         dllImports.setLayoutData(gd_dllImports);
00295         new Label(shell, SWT.NONE);
00296
00297         for(int index = 0;index<tableItems.length;index++)
00298         {
00299             tableItems[index] = new TableItem(details, SWT.NONE);
00300         }
00301         tableItems[0].setText(0, "Directory");
00302         tableItems[1].setText(0, "Version");
00303         tableItems[2].setText(0, "Name");
00304         tableItems[3].setText(0, "PID");
00305
00306
00307         mntmOpen.addListener(SWT.Selection, new Listener() {
00308             public void handleEvent(Event e) {
00309                 SelectFile selectFile = new SelectFile(shell.getLocation().x,
shell.getLocation().y, true);
00310                 code.clearAll();
00311                 if(!selectFile.isPidMode())
00312                 {
00313                     filePath = selectFile.getText();
00314                     try
00315                     {
00316                         tableItems[0].setText(1, filePath);
00317                         try
00318                         {
00319                             CodeExtract codeExtract = new CodeExtract(new
File(filePath));
00320                             String[] codeArr = codeExtract.getCodeArr();
00321                            // byte[] resources = codeExtract.getResources();
00322                             if(codeExtract.getPeFile().getVersion() ==
Version.x32)
```

```
00323                                {
00324                                    tableItems[1].setText(1, "32-bit");
00325                                }
00326                                else
00327                                {
00328                                    tableItems[1].setText(1, "64-bit");
00329                                }
00330                                code.setItemCount(0);
00331                                dllImports.clearAll();
00332                                for(int index=0;index<codeArr.length;index++)
00333                                {
00334                                    TableItem tableItem = new TableItem(code,
SWT.NULL);
00335                                    tableItem.setText(codeArr[index]);
00336                                }
00337                                tableItems[0].setGrayed(false);
00338                                tableItems[1].setGrayed(false);
00339                            }
00340                            catch (NullPointerException e1) {}
00341                        }
00342                        catch (IllegalArgumentException e1) {}
00343                    }
00344                    else
00345                    {
00346                        ProcessManager process = new
ProcessManager(selectFile.getPid());
00347                        tableItems[2].setText(1, process.getName());
00348                        tableItems[3].setText(1,
Integer.toString(selectFile.getPid()));
00349                        String[] DLLs = process.getDLLs();
00350                        dllImports.clearAll();
00351                        dllImports.setItemCount(0);
00352                        for(int index=0;index<DLLs.length;index++)
00353                        {
00354                            TableItem tableItem = new TableItem(dllImports,
SWT.NULL);
00355                            tableItem.setText(DLLs[index]);
00356                        }
00357                        tableItems[0].setGrayed(true);
00358                        tableItems[1].setGrayed(true);
00359                    }
00360                   /* }
00361                }).start();*/
00362                }
00363            });
00364        }
00365 }
```

MemoryComposite.java

```
00001 /*
00002  *
00003  */
00004 package dynamicAnalysis;
00005
00006 import org.eclipse.swt.widgets.Composite;
00007 import org.eclipse.swt.widgets.Display;
00008 import org.eclipse.swt.widgets.Event;
00009 import org.eclipse.swt.widgets.Button;
00010 import org.eclipse.swt.widgets.Listener;
00011
00012 import java.util.ArrayList;
00013
00014 import org.eclipse.jface.resource.JFaceResources;
00015 import org.eclipse.swt.SWT;
00016 import org.eclipse.swt.events.KeyAdapter;
00017 import org.eclipse.swt.events.KeyEvent;
00018 import org.eclipse.swt.events.SelectionAdapter;
00019 import org.eclipse.swt.events.SelectionEvent;
00020 import org.eclipse.swt.graphics.Color;
00021 import org.eclipse.swt.graphics.Font;
00022 import org.eclipse.swt.graphics.Point;
00023 import org.eclipse.swt.graphics.TextLayout;
00024 import org.eclipse.swt.graphics.TextStyle;
00025 import org.eclipse.swt.layout.GridLayout;
00026 import org.eclipse.swt.widgets.Label;
00027 import org.eclipse.swt.widgets.ProgressBar;
00028 import org.eclipse.wb.swt.SWTResourceManager;
00029 import org.eclipse.swt.widgets.Text;
00030 import org.eclipse.swt.widgets.Table;
00031 import org.eclipse.swt.widgets.TableItem;
00032 import org.eclipse.swt.layout.FormLayout;
00033 import org.eclipse.swt.layout.GridData;
00034 import org.eclipse.swt.layout.FormData;
00035 import org.eclipse.swt.layout.FormAttachment;
00036 import org.eclipse.swt.custom.StyleRange;
00037 import org.eclipse.swt.custom.StyledText;
00038
00042 public class MemoryComposite extends Composite
00043 {
00044
00046     private Text memoryField;
00047
00049     private int processId;
00050
00052     private byte[] bytes;
00053
00055     private Table asciiTable;
00056
00058     private Text searchText;
00059
00061     private TableItem[] tableStore = null;
00062
00064     private TableItem[] tableItems;
00065
00067     private String[] asciiSections;
00068
00070     private Color red;
00071
00073     private String memory;
00074
00076     private final int memoryCount = 10000;
00077
00085     public MemoryComposite(Composite parent, int style, Color green)
00086     {
00087
00088         super(parent, style);
00089         setProcessId(Window.processId);
00090         setRed(green);
00091         setLayout(new GridLayout(4, false));
00092         Button btnUpdate = new Button(this, SWT.NONE);
00093         btnUpdate.setFont(SWTResourceManager.getFont("Segoe UI", 9,
SWT.NORMAL));
```

```
00094         btnUpdate.setText("Update");
00095
00096         Label txtLength = new Label(this, SWT.BORDER);
00097         GridData gd_txtLength = new GridData(SWT.FILL, SWT.CENTER, true, false,
1, 1);
00098         gd_txtLength.widthHint = 109;
00099         txtLength.setLayoutData(gd_txtLength);
00100         txtLength.setText("Length: ");
00101
00102         searchText = new Text(this, SWT.BORDER);
00103         GridData gd_searchText = new GridData(SWT.FILL, SWT.CENTER, true, false,
1, 1);
00104         gd_searchText.widthHint = 157;
00105         searchText.setLayoutData(gd_searchText);
00106
00107         Button btnNewButton_1 = new Button(this, SWT.NONE);
00108         btnNewButton_1.addSelectionListener(new SelectionAdapter() {
00109             @Override
00110             public void widgetSelected(SelectionEvent e) {
00111                 searchEvent();
00112             }
00113         });
00114         btnNewButton_1.setText("Search");
00115
00116         memoryField = new Text(this, SWT.BORDER | SWT.WRAP | SWT.V_SCROLL);
00117         memoryField.setEditable(false);
00118         GridData gd_memoryField = new GridData(SWT.FILL, SWT.FILL, false, true,
2, 1);
00119         gd_memoryField.widthHint = 154;
00120         gd_memoryField.heightHint = 217;
00121         memoryField.setLayoutData(gd_memoryField);
00122
00123
00124         asciiTable = new Table(this, SWT.BORDER | SWT.FULL_SELECTION |
SWT.VIRTUAL);
00125
00126         GridData gd_asciiTable = new GridData(SWT.FILL, SWT.FILL, true, true, 2,
1);
00127         gd_asciiTable.heightHint = 225;
00128         gd_asciiTable.widthHint = 197;
00129         asciiTable.setLayoutData(gd_asciiTable);
00130         asciiTable.setHeaderVisible(true);
00131         asciiTable.setLinesVisible(true);
00132
00133         new Thread() {
00134             public void run() {
00135                 String memoryText = updateMemory();
00136                 asciiSections = findAsciiSections();
00137                 Display.getDefault().asyncExec(new Runnable() {
00138                     @Override
00139                     public void run() {
00140                         System.out.println(memoryText);
00141                         memoryField.setText(memoryText);
00142                         txtLength.setText("Length: "+getBytes().length);
00143                         tableItems = new TableItem[asciiSections.length];
00144                         asciiTable.setItemCount(asciiSections.length);
00145                         for(int index=0;index<asciiSections.length;index++)
00146                         {
00147
asciiTable.getItem(index).setText(asciiSections[index].substring(asciiSections[index].
indexOf('}')+1));
00148
asciiTable.getItem(index).setData(Integer.parseInt(asciiSections[index].substring(1,
asciiSections[index].indexOf('}'))));
00149                         }
00150                     }
00151                 });
00152             }
00153         }.start();
00154         btnUpdate.addSelectionListener(new SelectionAdapter() {
00155             @Override
00156             public void widgetSelected(SelectionEvent e) {
00157                 new Thread() {
00158                     public void run() {
00159                         updateMemory();
00160                         Display.getDefault().asyncExec(new Runnable() {
00161                             public void run() {
```

```
00162                                   txtLength.setText("Length: "+getBytes().length);
00163                               }
00164                           });
00165                       }
00166                   }.start();
00167               }
00168           });
00169
00170           searchText.addKeyListener(new KeyAdapter(){
00171               public void keyPressed(KeyEvent e){
00172                   if(e.keyCode == SWT.CR) {
00173                       searchEvent();
00174                   }
00175               }
00176           });
00177
00178           asciiTable.addSelectionListener(new SelectionAdapter() {
00179               @Override
00180               public void widgetDefaultSelected(SelectionEvent e) {
00181                   int index=asciiTable.getSelectionIndex();
00182
00183                   int startIndex=(int)asciiTable.getItem(index).getData();
00184                   System.out.println("index: "+startIndex);
00185                   String tableText = asciiTable.getItem(index).getText();
00186                   //System.out.println(tableText);
00187                   populateMemory(memoryField, startIndex, tableText);
00188               }
00189           });
00190       }
00191
00197       public int getProcessId()
00198       {
00199           return processId;
00200       }
00201
00207       public void setProcessId(int processId)
00208       {
00209           this.processId = processId;
00210       }
00211
00217       private byte[] readMemory()
00218       {
00219           VirtualMemory virtualMemory = new VirtualMemory(getProcessId());
00220           setBytes(virtualMemory.readMemory());
00221           return getBytes();
00222       }
00223
00229       public byte[] getBytes()
00230       {
00231           return bytes;
00232       }
00233
00239       public void setBytes(byte[] bytes)
00240       {
00241           this.bytes = bytes;
00242       }
00243
00249       public Color getRed() {
00250           return red;
00251       }
00252
00258       public void setRed(Color red) {
00259           this.red = red;
00260       }
00261
00267       private String updateMemory()
00268       {
00269           long start = System.currentTimeMillis();
00270           byte[] chars = readMemory();
00271           String output = "";
00272           String update = "";
00273           int sizeIndex = 0;
00274           int index = 0;
00275           while(sizeIndex!=memoryCount)
00276           {
00277               try
00278               {
```

```
00279                 if(chars[index]!=0)
00280                 {
00281                     output += (char)chars[index];
00282                     sizeIndex++;
00283                 }
00284                 index++;
00285             }
00286             catch(ArrayIndexOutOfBoundsException e)
00287             {
00288                 e.printStackTrace();
00289                 //MessageDialog.openError(shell, "Error", "Process was
closed.");
00290                 break;
00291             }
00292         }
00293         setBytes(chars);
00294         System.out.println("memory time: "+(System.currentTimeMillis()-start));
00295         return output;
00296     }
00297
00303     private String[] findAsciiSections()
00304     {
00305         long startTime = System.currentTimeMillis();
00306         int startIndex = 0;
00307         boolean start = true;
00308         ArrayList<String> asciiSections = new ArrayList<String>();
00309         String current="";
00310         for(int index=0;index<getBytes().length;index++)
00311         {
00312             while(isAscii((char)getBytes()[index]))
00313             {
00314                 if(start)
00315                 {
00316                     startIndex=index;
00317                     start=false;
00318                 }
00319                 current+=(char)getBytes()[index];
00320                 index++;
00321             }
00322             if(current.length()>8)
00323             {
00324                 asciiSections.add("{"+startIndex+"}"+current);
00325             }
00326             if(!start) start=true;
00327             current="";
00328         }
00329         return asciiSections.toArray(new String[0]);
00330     }
00331
00338     private boolean isAscii(char character)
00339     {
00340         return character>=32&&character<=126;
00341     }
00342
00350     private String[] search(String[] entries, String toSearch)
00351     {
00352         ArrayList<String> filtered = new ArrayList<String>();
00353         for(int index = 0; index < entries.length; index++)
00354         {
00355             if(toSearch.endsWith("*"))
00356             {
00357
if(entries[index].toLowerCase().substring(entries[index].indexOf('}')+1).startsWith(to
Search.toLowerCase().substring(0, toSearch.length()-1)))
00358                 {
00359                     filtered.add(entries[index]);
00360                 }
00361             }
00362             else if(toSearch.startsWith("*"))
00363             {
00364
if(entries[index].toLowerCase().substring(entries[index].indexOf('}')+1).endsWith(toSe
arch.toLowerCase().substring(1)))
00365                 {
00366                     filtered.add(entries[index]);
00367                 }
00368             }
```

```
00369             else
00370             {
00371
if(entries[index].toLowerCase().substring(entries[index].indexOf('}')+1).contains(toSe
arch.toLowerCase()))
00372                 {
00373                     filtered.add(entries[index]);
00374                 }
00375             }
00376
00377         }
00378         String[] filteredArray = filtered.toArray(new String[0]);
00379         return filteredArray;
00380     }
00381
00385     private void searchEvent()
00386     {
00387         if(searchText.getText()=="")
00388         {
00389             asciiTable.setItemCount(asciiSections.length);
00390             for(int index=0;index<asciiSections.length;index++)
00391             {
00392
asciiTable.getItem(index).setText(asciiSections[index].substring(asciiSections[index].
indexOf('}')+1));
00393
asciiTable.getItem(index).setData(Integer.parseInt(asciiSections[index].substring(1,
asciiSections[index].indexOf('}'))));
00394             }
00395         }
00396         else
00397         {
00398             if(tableStore==null)
00399             {
00400                 tableStore = tableItems;
00401             }
00402             String[] filtered = search(asciiSections, searchText.getText());
00403             asciiTable.setItemCount(filtered.length);
00404             for(int index=0;index<filtered.length;index++)
00405             {
00406
asciiTable.getItem(index).setText(filtered[index].substring(filtered[index].indexOf('}
')+1));
00407
asciiTable.getItem(index).setData(Integer.parseInt(filtered[index].substring(1,
filtered[index].indexOf('}'))));
00408             }
00409         }
00410     }
00411
00420     private String populateMemory(Text text, int memoryIndex, String tableText)
00421     {
00422         byte[] fullMemory = getBytes();
00423         memory = "";
00424         int startIndex;
00425         if(memoryIndex<memoryCount/2) startIndex = 0;
00426         else startIndex=memoryIndex-(memoryCount/2);
00427         for(int index = startIndex; index<startIndex+memoryCount; index++)
00428         {
00429             if(fullMemory[index]>0)
00430             {
00431                 memory+=(char)fullMemory[index];
00432             }
00433
00434         }
00435         text.setText(memory);
00436         System.out.println(memory);
00437         System.out.println("startindex: "+memory.indexOf(tableText));
00438         System.out.println("endindex:
"+(memory.indexOf(tableText)+tableText.length()));
00439         text.setFocus();
00440         text.setSelection( new Point( text.getText().indexOf(tableText),
text.getText().indexOf(tableText)+tableText.length() ) );
00441         return memory;
00442     }
00443
00447     @Override
```

```
00448     protected void checkSubclass()
00449     {
00450         // Disable the check that prevents subclassing of SWT components
00451     }
00452 }
```

MemoryWindow.java

```
00001 /*
00002  *
00003  */
00004 package dynamicAnalysis;
00005 import org.eclipse.swt.widgets.Display;
00006 import org.eclipse.swt.widgets.Shell;
00007 import org.eclipse.swt.layout.GridLayout;
00008
00009 import java.util.ArrayList;
00010
00011 import org.eclipse.jface.dialogs.MessageDialog;
00012 import org.eclipse.swt.SWT;
00013 import org.eclipse.swt.widgets.Label;
00014 import org.eclipse.swt.layout.GridData;
00015 import org.eclipse.wb.swt.SWTResourceManager;
00016 import org.eclipse.swt.widgets.Text;
00017 import org.eclipse.swt.widgets.Button;
00018 import org.eclipse.swt.events.KeyAdapter;
00019 import org.eclipse.swt.events.KeyEvent;
00020 import org.eclipse.swt.events.SelectionAdapter;
00021 import org.eclipse.swt.events.SelectionEvent;
00022 import org.eclipse.swt.widgets.Table;
00023 import org.eclipse.swt.widgets.TableItem;
00024 import org.eclipse.swt.widgets.ProgressBar;
00025
00029 public class MemoryWindow
00030 {
00031
00033     protected Shell shell;
00034
00036     private int processId;
00037
00039     private Text text;
00040
00042     private Label txtLength;
00043
00045     private byte[] bytes;
00046
00048     private int x;
00049
00051     private int y;
00052
00054     private Table asciiTable;
00055
00057     private Button btnNewButton;
00058
00060     private Text searchText;
00061
00063     private TableItem[] tableStore = null;
00064
00066     private TableItem[] tableItems;
00067
00069     private Button btnUpdate;
00070
00072     private GridData gd_asciiTable;
00073
00075     private ProgressBar progressBar;
00076
00078     private String[] asciiSections;
00079
00080     /*
00081     public static void main(String[] args)
00082     {
00083         MemoryWindow memoryWindow = new MemoryWindow();
00084         memoryWindow.open();
00085     }*/
00086
00094     public MemoryWindow(int processId, int x, int y)
00095     {
00096         try
00097         {
00098             setProcessId(processId);
00099             setX(x);
```

```
00100             setY(y);
00101         } catch (Exception e)
00102         {
00103             e.printStackTrace();
00104         }
00105     }
00106 
00107 
00113     public int getProcessId()
00114     {
00115         return processId;
00116     }
00117 
00123     public void setProcessId(int processId)
00124     {
00125         this.processId = processId;
00126     }
00127 
00133     public int getX()
00134     {
00135         return x;
00136     }
00137 
00143     public void setX(int x)
00144     {
00145         this.x = x;
00146     }
00147 
00153     public int getY()
00154     {
00155         return y;
00156     }
00157 
00158 
00164     public void setY(int y)
00165     {
00166         this.y = y;
00167     }
00168 
00172     public void open()
00173     {
00174         Display display = Display.getDefault();
00175         createBaseContents();
00176         shell.open();
00177         createContents();
00178         while (!shell.isDisposed())
00179         {
00180             if (!display.readAndDispatch())
00181             {
00182                 display.sleep();
00183             }
00184         }
00185     }
00186 
00192     private byte[] readMemory()
00193     {
00194         VirtualMemory virtualMemory = new VirtualMemory(getProcessId());
00195         return virtualMemory.readMemory();
00196     }
00197 
00198 
00199 
00205     public byte[] getBytes()
00206     {
00207         return bytes;
00208     }
00209 
00210 
00216     public void setBytes(byte[] bytes)
00217     {
00218         this.bytes = bytes;
00219     }
00220 
00221 
00228     private String updateMemory(Text text)
00229     {
00230         long start = System.currentTimeMillis();
```

```
00231         byte[] chars = readMemory();
00232         String output = "";
00233         String update = "";
00234         int sizeIndex = 0;
00235         int index = 0;
00236         while(sizeIndex!=10000)
00237         {
00238             try
00239             {
00240                 if(chars[index]!=0)
00241                 {
00242                     output += (char)chars[index];
00243                     sizeIndex++;
00244                 }
00245                 index++;
00246             }
00247             catch(ArrayIndexOutOfBoundsException e)
00248             {
00249                 //MessageDialog.openError(shell, "Error", "Process was
closed.");
00250                 break;
00251             }
00252         }
00253         setBytes(chars);
00254         System.out.println("memory time: "+(System.currentTimeMillis()-start));
00255         return output;
00256     }
00257
00263     private String[] findAsciiSections()
00264     {
00265         long startTime = System.currentTimeMillis();
00266         int startIndex = 0;
00267         boolean start = true;
00268         ArrayList<String> asciiSections = new ArrayList<String>();
00269         String current="";
00270         for(int index=0;index<getBytes().length;index++)
00271         {
00272             while(isAscii((char)getBytes()[index]))
00273             {
00274                 if(start)
00275                 {
00276                     startIndex=index;
00277                     start=false;
00278                 }
00279                 current+=(char)getBytes()[index];
00280                 index++;
00281             }
00282             if(current.length()>8)
00283             {
00284                 asciiSections.add("{"+startIndex+"}"+current);
00285             }
00286             if(!start) start=true;
00287             current="";
00288         }
00289         System.out.println("ascii time: "+(System.currentTimeMillis()-
startTime));
00290         return asciiSections.toArray(new String[0]);
00291     }
00292
00299     private boolean isAscii(char character)
00300     {
00301         return character>=32&&character<=126;
00302     }
00303
00311     private String[] search(String[] entries, String toSearch)
00312     {
00313         ArrayList<String> filtered = new ArrayList<String>();
00314         for(int index = 0; index < entries.length; index++)
00315         {
00316             if(toSearch.endsWith("*"))
00317             {
00318
if(entries[index].toLowerCase().startsWith(toSearch.toLowerCase().substring(0,
toSearch.length()-1)))
00319                 {
00320                     filtered.add(entries[index]);
00321                 }
```

```
00322             }
00323             else if(toSearch.startsWith("*"))
00324             {
00325 
if(entries[index].toLowerCase().endsWith(toSearch.toLowerCase().substring(1)))
00326                 {
00327                     filtered.add(entries[index]);
00328                 }
00329             }
00330             else
00331             {
00332 
if(entries[index].toLowerCase().contains(toSearch.toLowerCase()))
00333                 {
00334                     filtered.add(entries[index]);
00335                 }
00336             }
00337 
00338         }
00339         String[] filteredArray = filtered.toArray(new String[0]);
00340         return filteredArray;
00341     }
00342 
00346     private void searchEvent()
00347     {
00348         if(searchText.getText()=="")
00349         {
00350             asciiTable.setItemCount(0);
00351             for(int index=0;index<asciiSections.length;index++)
00352             {
00353                 if(index>6000)
00354                 {
00355                     break;
00356                 }
00357                 tableItems = new TableItem[asciiSections.length];
00358                 tableItems[index]=new TableItem(asciiTable, SWT.NULL);
00359                 tableItems[index].setText(asciiSections[index]);
00360             }
00361         }
00362         else
00363         {
00364             if(tableStore==null)
00365             {
00366                 tableStore = tableItems;
00367             }
00368             String[] filtered = search(asciiSections, searchText.getText());
00369             asciiTable.setItemCount(0);
00370             tableItems = new TableItem[filtered.length];
00371             for(int index=0;index<filtered.length;index++)
00372             {
00373                 tableItems[index]=new TableItem(asciiTable, SWT.NULL);
00374                 tableItems[index].setText(filtered[index]);
00375             }
00376         }
00377     }
00378 
00382     protected void createBaseContents()
00383     {
00384         shell = new Shell();
00385         shell.setSize(865, 432);
00386         shell.setLayout(new GridLayout(5, false));
00387         shell.setLocation(x+490/2, y+342/2);
00388 
00389         btnUpdate = new Button(shell, SWT.NONE);
00390         btnUpdate.addSelectionListener(new SelectionAdapter() {
00391             @Override
00392             public void widgetSelected(SelectionEvent e) {
00393                 updateMemory(text);
00394                 txtLength.setText("Length: "+getBytes().length);
00395             }
00396         });
00397         btnUpdate.setText("Update");
00398         txtLength = new Label(shell, SWT.BORDER);
00399         txtLength.setText("Length: ");
00400         txtLength.setLayoutData(new GridData(SWT.FILL, SWT.CENTER, false, false,
1, 1));
00401 
```

```
00402         progressBar = new ProgressBar(shell, SWT.NONE);
00403         new Label(shell, SWT.NONE);
00404         new Label(shell, SWT.NONE);
00405 
00406         new Label(shell, SWT.NONE);
00407 
00408         text = new Text(shell, SWT.BORDER | SWT.WRAP | SWT.V_SCROLL);
00409         text.setLayoutData(new GridData(SWT.FILL, SWT.CENTER, true, false, 1,
1));
00410         text.setFont(SWTResourceManager.getFont("Calibri", 9, SWT.NORMAL));
00411         GridData gd_text = new GridData(SWT.FILL, SWT.FILL, true, true, 1, 1);
00412         gd_text.widthHint = 399;
00413         text.setLayoutData(gd_text);
00414 
00415         asciiTable = new Table(shell, SWT.BORDER | SWT.FULL_SELECTION);
00416         gd_asciiTable = new GridData(SWT.FILL, SWT.FILL, true, true, 1, 1);
00417         gd_asciiTable.widthHint = 186;
00418         asciiTable.setLayoutData(gd_asciiTable);
00419         asciiTable.setHeaderVisible(true);
00420         asciiTable.setLinesVisible(true);
00421 
00422         searchText = new Text(shell, SWT.BORDER);
00423         GridData gd_searchText = new GridData(SWT.FILL, SWT.TOP, true, false, 1,
1);
00424         gd_searchText.widthHint = 70;
00425         searchText.setLayoutData(gd_searchText);
00426         searchText.addKeyListener(new KeyAdapter(){
00427             public void keyPressed(KeyEvent e){
00428                 if(e.keyCode == SWT.CR) {
00429                     searchEvent();
00430                 }
00431             }
00432         });
00433 
00434 
00435         btnNewButton = new Button(shell, SWT.NONE);
00436         GridData gd_btnNewButton = new GridData(SWT.LEFT, SWT.TOP, false, false,
1, 1);
00437         gd_btnNewButton.widthHint = 62;
00438         btnNewButton.setLayoutData(gd_btnNewButton);
00439         btnNewButton.setText("Search");
00440     }
00441     /*
00442     private void updateBar(ProgressBar progresBar, int selection)
00443     {
00444         progressBar.getDisplay().asyncExec(new Runnable() {
00445             @Override
00446             public void run() {
00447                 progressBar.setSelection(selection);
00448             }});
00449     }*/
00450 
00454     protected synchronized void createContents()
00455     {
00456         progressBar.setMaximum(100);
00457         progressBar.setVisible(true);
00458         long start = System.currentTimeMillis();
00459         text.setText(updateMemory(text));
00460         progressBar.setSelection(33);
00461         System.out.println("set memory text time: "+(System.currentTimeMillis()-
start));
00462         txtLength.setText("Length: "+getBytes().length);
00463         progressBar.setSelection(66);
00464         asciiSections = findAsciiSections();
00465         tableItems = new TableItem[asciiSections.length];
00466         start = System.currentTimeMillis();
00467         for(int index=0;index<asciiSections.length;index++)
00468         {
00469             if(index>10000)
00470             {
00471                 break;
00472             }
00473             tableItems[index] = new TableItem(asciiTable, SWT.NULL);
00474             tableItems[index].setText(asciiSections[index]);
00475         }
00476         System.out.println("set table time: "+(System.currentTimeMillis()-
start));
```

```
00477
00478         btnNewButton.addSelectionListener(new SelectionAdapter() {
00479             @Override
00480             public void widgetSelected(SelectionEvent e) {
00481                 searchEvent();
00482             }
00483         });
00484
00485
00486
00487         progressBar.setVisible(false);
00488     }
00489
00490 }
```

Mnem.java

```
00001 /*
00002  *
00003  */
00004 package dynamicAnalysis;
00005
00009 public enum Mnem {
00010
00012     JNE,
00014  JMP,
00016  JL,
00018  JBE,
00020  JE,
00022  JAE,
00024  JB,
00026  JA,
00028  JLE;
00029
00035     public byte getByte()
00036     {
00037         switch(this)
00038         {
00039         case JAE:
00040             return 115;
00041         case JBE:
00042             return 118;
00043         case JE:
00044             return 15;
00045         case JL:
00046             return 124;
00047         case JMP:
00048             return -21;
00049         case JNE:
00050             return 117;
00051         case JB:
00052             return 114;
00053         case JA:
00054             return 119;
00055         case JLE:
00056             return 126;
00057         default:
00058             return 0;
00059
00060         }
00061     }
00062 }
```

NetworkComposite.java

```
00001 /*
00002  *
00003  */
00004 package dynamicAnalysis;
00005
00006 import org.eclipse.swt.widgets.Composite;
00007 import org.eclipse.swt.widgets.Display;
00008 import org.eclipse.swt.widgets.Event;
00009 import org.eclipse.swt.widgets.ToolBar;
00010 import org.eclipse.swt.SWT;
00011 import org.eclipse.swt.widgets.ToolItem;
00012 import org.pcap4j.core.NotOpenException;
00013 import org.pcap4j.core.PcapNativeException;
00014 import org.pcap4j.packet.IpPacket;
00015 import org.pcap4j.packet.Packet;
00016 import org.eclipse.swt.events.SelectionAdapter;
00017 import org.eclipse.swt.events.SelectionEvent;
00018 import org.eclipse.swt.widgets.Combo;
00019 import org.eclipse.swt.widgets.List;
00020 import org.eclipse.swt.widgets.Listener;
00021 import org.eclipse.swt.widgets.MessageBox;
00022
00023 import java.util.ArrayList;
00024 import java.util.HashMap;
00025 import java.util.Map.Entry;
00026 import java.util.concurrent.TimeUnit;
00027
00028 import org.eclipse.jface.viewers.ListViewer;
00029 import org.eclipse.swt.layout.FormLayout;
00030 import org.eclipse.swt.layout.FormData;
00031 import org.eclipse.swt.layout.FormAttachment;
00032 import org.eclipse.swt.layout.GridLayout;
00033 import org.eclipse.swt.layout.GridData;
00034 import org.eclipse.swt.widgets.Label;
00035 import org.eclipse.ui.forms.widgets.FormToolkit;
00036 import org.eclipse.swt.custom.StyledText;
00037 import org.eclipse.jface.text.TextViewer;
00038 import org.eclipse.wb.swt.SWTResourceManager;
00039 import org.eclipse.swt.widgets.Tree;
00040 import org.eclipse.ui.forms.widgets.ExpandableComposite;
00041 import org.eclipse.ui.forms.widgets.ScrolledForm;
00042 import org.eclipse.ui.forms.widgets.FormText;
00043 import org.eclipse.swt.widgets.Button;
00044
00048 public class NetworkComposite extends Composite
00049 {
00050
00052     private PacketTrace packetTrace;
00053
00055     private List networkInterfaces;
00056
00058     private List addresses;
00059
00061     private List packets;
00062
00064     private HashMap<String, String> devices;
00065
00067     private final FormToolkit formToolkit = new
FormToolkit(Display.getDefault());
00068
00070     private boolean addressSelected = false;
00071
00073     private ActiveConnection[] activeConnections;
00074
00076     private StyledText packetInfo;
00077
00079     private StyledText hexPayload;
00080
00082     private StyledText asciiPayload;
00083
00092     public NetworkComposite(Composite parent, int style, long pid) throws
PcapNativeException
00093     {
```

```
00094         super(parent, style);
00095         setLayout(new FormLayout());
00096
00097         networkInterfaces = new List(this, SWT.BORDER | SWT.H_SCROLL |
SWT.V_SCROLL);
00098         FormData fd_networkInterfaces = new FormData();
00099         fd_networkInterfaces.top = new FormAttachment(0, 41);
00100         fd_networkInterfaces.right = new FormAttachment(0, 211);
00101         networkInterfaces.setLayoutData(fd_networkInterfaces);
00102
00103         packetInfo = new StyledText(this, SWT.BORDER | SWT.WRAP);
00104         packetInfo.setEditable(false);
00105         FormData fd_packetInfo = new FormData();
00106         fd_packetInfo.left = new FormAttachment(networkInterfaces, 5);
00107         fd_packetInfo.right = new FormAttachment(100, -4);
00108         fd_packetInfo.bottom = new FormAttachment(100, -172);
00109         fd_packetInfo.top = new FormAttachment(0, 41);
00110         packetInfo.setLayoutData(fd_packetInfo);
00111         packetInfo.setFont(SWTResourceManager.getFont("Segoe UI", 8,
SWT.NORMAL));
00112         formToolkit.adapt(packetInfo);
00113         formToolkit.paintBordersFor(packetInfo);
00114
00115         addresses = new List(this, SWT.BORDER | SWT.V_SCROLL);
00116         fd_networkInterfaces.left = new FormAttachment(addresses, 0, SWT.LEFT);
00117         fd_networkInterfaces.bottom = new FormAttachment(addresses, -6);
00118         FormData fd_addresses = new FormData();
00119         fd_addresses.bottom = new FormAttachment(0, 203);
00120         fd_addresses.right = new FormAttachment(0, 211);
00121         fd_addresses.top = new FormAttachment(0, 126);
00122         fd_addresses.left = new FormAttachment(0, 5);
00123         addresses.setLayoutData(fd_addresses);
00124
00125         packets = new List(this, SWT.BORDER | SWT.V_SCROLL);
00126         FormData fd_packets = new FormData();
00127         fd_packets.top = new FormAttachment(addresses, 5);
00128         fd_packets.bottom = new FormAttachment(100, -5);
00129         fd_packets.right = new FormAttachment(0, 211);
00130         fd_packets.left = new FormAttachment(0, 5);
00131         packets.setLayoutData(fd_packets);
00132         formToolkit.adapt(packets, true, true);
00133         Label label = new Label(this, SWT.NONE);
00134         FormData fd_label = new FormData();
00135         fd_label.top = new FormAttachment(0, 271);
00136         fd_label.left = new FormAttachment(0, 216);
00137         label.setLayoutData(fd_label);
00138         formToolkit.adapt(label, true, true);
00139
00140         hexPayload = new StyledText(this, SWT.BORDER | SWT.WRAP | SWT.V_SCROLL);
00141         hexPayload.setEditable(false);
00142         FormData fd_hexPayload = new FormData();
00143         fd_hexPayload.right = new FormAttachment(packetInfo, 0, SWT.RIGHT);
00144         fd_hexPayload.bottom = new FormAttachment(packetInfo, 85, SWT.BOTTOM);
00145         fd_hexPayload.top = new FormAttachment(packetInfo, 6);
00146         hexPayload.setLayoutData(fd_hexPayload);
00147         formToolkit.adapt(hexPayload);
00148         formToolkit.paintBordersFor(hexPayload);
00149
00150         asciiPayload = new StyledText(this, SWT.BORDER | SWT.WRAP |
SWT.V_SCROLL);
00151         asciiPayload.setEditable(false);
00152         fd_hexPayload.left = new FormAttachment(asciiPayload, 0, SWT.LEFT);
00153         FormData fd_asciiPayload = new FormData();
00154         fd_asciiPayload.right = new FormAttachment(100, -4);
00155         fd_asciiPayload.top = new FormAttachment(hexPayload, 6);
00156         fd_asciiPayload.bottom = new FormAttachment(100, -5);
00157         asciiPayload.setLayoutData(fd asciiPayload);
00158         formToolkit.adapt(asciiPayload);
00159         formToolkit.paintBordersFor(asciiPayload);
00160
00161         Label lblNewLabel = formToolkit.createLabel(this, "Hex Payload",
SWT.NONE);
00162         lblNewLabel.setBackground(SWTResourceManager.getColor(240, 240, 240));
00163         FormData fd_lblNewLabel = new FormData();
00164         fd_lblNewLabel.top = new FormAttachment(packetInfo, 6);
00165         fd_lblNewLabel.right = new FormAttachment(hexPayload, -6);
00166         lblNewLabel.setLayoutData(fd_lblNewLabel);
```

```
00167
00168         Label lblAsciiPayload = formToolkit.createLabel(this, "ASCII Payload",
SWT.NONE);
00169         lblAsciiPayload.setBackground(SWTResourceManager.getColor(240, 240,
240));
00170         fd_asciiPayload.left = new FormAttachment(0, 299);
00171         FormData fd_lblAsciiPayload = new FormData();
00172         fd_lblAsciiPayload.top = new FormAttachment(hexPayload, 6);
00173         fd_lblAsciiPayload.right = new FormAttachment(asciiPayload, -6);
00174         lblAsciiPayload.setLayoutData(fd_lblAsciiPayload);
00175
00176         Button btnFilterByProcess = new Button(this, SWT.CHECK);
00177         btnFilterByProcess.addSelectionListener(new SelectionAdapter() {
00178             @Override
00179             public void widgetSelected(SelectionEvent e) {
00180                 clearAll();
00181             }
00182         });
00183         btnFilterByProcess.setGrayed(true);
00184         btnFilterByProcess.setText("Filter by process");
00185         FormData fd_btnFilterByProcess = new FormData();
00186         fd_btnFilterByProcess.top = new FormAttachment(0, 11);
00187         fd_btnFilterByProcess.bottom = new FormAttachment(packetInfo, -6);
00188         fd_btnFilterByProcess.left = new FormAttachment(packetInfo, 0,
SWT.LEFT);
00189         btnFilterByProcess.setLayoutData(fd_btnFilterByProcess);
00190         formToolkit.adapt(btnFilterByProcess, true, true);
00191
00192         Label pidMatch = formToolkit.createLabel(this, "New Label", SWT.NONE);
00193         pidMatch.setBackground(SWTResourceManager.getColor(240, 240, 240));
00194         FormData fd_pidMatch = new FormData();
00195         fd_pidMatch.bottom = new FormAttachment(networkInterfaces, -15);
00196         fd_pidMatch.right = new FormAttachment(btnFilterByProcess, -6);
00197         fd_pidMatch.left = new FormAttachment(0, 5);
00198         fd_pidMatch.top = new FormAttachment(0, 11);
00199         pidMatch.setLayoutData(fd_pidMatch);
00200
00201         Button btnClear = formToolkit.createButton(this, "Clear", SWT.NONE);
00202         btnClear.addSelectionListener(new SelectionAdapter() {
00203             @Override
00204             public void widgetSelected(SelectionEvent e) {
00205                 clearAll();
00206             }
00207         });
00208         FormData fd_btnClear = new FormData();
00209         fd_btnClear.bottom = new FormAttachment(packetInfo, -6);
00210         fd_btnClear.right = new FormAttachment(100, -4);
00211         fd_btnClear.top = new FormAttachment(0, 10);
00212         btnClear.setLayoutData(fd_btnClear);
00213
00214         String address = "";
00215         System.out.println("pid: "+pid);
00216         if(pid == 0)
00217         {
00218             pidMatch.setText("No process selected");
00219         }
00220         else
00221         {
00222             NetworkStats networkStats = new NetworkStats();
00223             activeConnections = networkStats.getActiveConnections();
00224             int count = 1;
00225             boolean extraAddress = false;
00226             for(int index = 0; index < activeConnections.length; index++)
00227             {
00228                 if(activeConnections[index].getPid() == pid)
00229                 {
00230                     extraAddress = !address.equals("");
00231                     address = activeConnections[index].getLocalAddress();
00232                     pidMatch.setText("Process address: "+address);
00233                     if(extraAddress) pidMatch.setText(pidMatch.getText()+"
(+"+count+++")");
00234                 }
00235             }
00236         }
00237
00238         networkInterfaces.addListener(SWT.Selection, new Listener() {
00239               public void handleEvent(Event e) {
```

```
00240                 addresses.removeAll();
00241                 try
00242                 {
00243                     String deviceName = (String)
networkInterfaces.getData(networkInterfaces.getSelection()[0]);
00244                     String[] addressesArr =
packetTrace.getAddresses(deviceName);
00245                     System.out.println("addresses length:
"+addressesArr.length);
00246                     for(int index = 0; index < addressesArr.length; index++)
00247                     {
00248                         try
00249                         {
00250                             if(pid != 0 && btnFilterByProcess.getSelection())
00251                             {
00252                                 for(int j = 0; j < activeConnections.length;
j++)
00253                                 {
00254                                     if(activeConnections[j].getPid()==pid &&
activeConnections[j].getLocalAddress() == addressesArr[index])
00255                                     {
00256                                         addresses.add(addressesArr[index]);
00257                                     }
00258                                 }
00259                             }
00260                             else
00261                             {
00262                                 addresses.add(addressesArr[index]);
00263                             }
00264
00265                         }
00266                         catch(IllegalArgumentException e1)
00267                         {
00268                             e1.printStackTrace();
00269                             continue;
00270                         }
00271                     }
00272                 }
00273                 catch(ArrayIndexOutOfBoundsException e1) {}
00274             }
00275         });
00276         addresses.addListener(SWT.Selection, new Listener() {
00277               public void handleEvent(Event e) {
00278                   packets.removeAll();
00279                   addressSelected = true;
00280               }
00281           });
00282         packets.addListener(SWT.Selection, new Listener() {
00283               public void handleEvent(Event e) {
00284                   IpPacket packet =
((IpPacket)packets.getData(packets.getSelection()[0]));
00285                   packetInfo.setText(packet.getHeader().toString());
00286                   byte[] payloadBytes = packet.getPayload().getRawData();
00287                   String asciiPayloadStr = "";
00288                   String hexPayloadStr = "";
00289                   for(int index = 0; index < payloadBytes.length;index++)
00290                   {
00291                       asciiPayloadStr+=(char) payloadBytes[index];
00292                       hexPayloadStr+=String.format("%02X ",
payloadBytes[index]);
00293                   }
00294                   asciiPayload.setText(asciiPayloadStr);
00295                   hexPayload.setText(hexPayloadStr);
00296               }
00297           });
00298         packetTrace = new PacketTrace();
00299         updateNetwork();
00300     }
00301
00305     private synchronized void updateNetwork()
00306     {
00307         Thread updateThread = new Thread(() -> { // Set up thread for each
network interface
00308             while (true)
00309             {
00310                 HashMap<String, String> newDevices = packetTrace.getDevices();
00311
```

```
00312                     Display.getDefault().asyncExec(new Runnable() {
00313                         public void run() {
00314                             if(!newDevices.equals(devices))
00315                             {
00316                                 devices = newDevices;
00317                                 networkInterfaces.removeAll();
00318                                 for(Entry<String, String> device :
devices.entrySet())
00319                                 {
00320                                     networkInterfaces.add(device.getValue());
00321                                     networkInterfaces.setData(device.getValue(),
device.getKey());
00322                                 }
00323                             }
00324                             if(addressSelected)
00325                             {
00326                               try
00327                               {
00328                                   ArrayList<IpPacket> packetList =
packetTrace.getPackets(addresses.getSelection()[0], 0);
00329                                   if(packetList == null)
00330                                   {
00331                                       errorAlert("Something went wrong when
fetching packets.");
00332                                   }
00333                                   for(IpPacket packet : packetList)
00334                                   {
00335                                       String packetDescription =
packet.getHeader().getProtocol().name()+" -
"+packet.getPayload().getRawData().length+" bytes {"+packet.hashCode()+"}";
00336                                       packets.add(packetDescription);
00337                                       packets.setData(packetDescription,
packet);
00338                                   }
00339                               }
00340                               catch(ArrayIndexOutOfBoundsException e) {}
00341                             }
00342                         }
00343                     });
00344                 try
00345                 {
00346                     TimeUnit.SECONDS.sleep(1);
00347                 } catch (InterruptedException e)
00348                 {
00349                     e.printStackTrace();
00350                 }
00351             }
00352         });
00353         updateThread.start();
00354     }
00355
00361     private void errorAlert(String message)
00362     {
00363         MessageBox messageBox = new MessageBox(this.getShell(), SWT.ERROR);
00364         messageBox.setText("Error");
00365         messageBox.setMessage(message);
00366         messageBox.open();
00367     }
00368
00372     private void clearAll()
00373     {
00374         networkInterfaces.deselectAll();
00375         addresses.removeAll();
00376         addresses.deselectAll();
00377         packets.removeAll();
00378         packets.deselectAll();
00379         packetInfo.setText("");
00380         hexPayload.setText("");
00381     }
00382
00386     @Override
00387     protected void checkSubclass()
00388     {
00389         // Disable the check that prevents subclassing of SWT components
00390     }
00391 }
00392
```

```
00393 //list netstat info
00394 //dropdowns for network interface, address, list packet
00395 //show udp/tcp info
```

NetworkStats.java

```
00001 /*
00002  *
00003  */
00004 package dynamicAnalysis;
00005
00006 import java.util.ArrayList;
00007
00011 public class NetworkStats
00012 {
00013
00015     private ActiveConnection[] activeConnections;
00016
00020     public NetworkStats()
00021     {
00022         CommandLine commandLine = new CommandLine();
00023         String stats = commandLine.getNetstat();
00024         System.out.println(stats);
00025         stats=stats.substring(stats.indexOf("PID")+6);
00026         ArrayList<ActiveConnection> activeConnectionsList = new
ArrayList<ActiveConnection>();
00027         while(!stats.equals(""))
00028         {
00029             try
00030             {
00031                 String protocol=stats.substring(0, 3);
00032                 stats = spaceBreak(stats);
00033                 String localAddress = stats.substring(0, stats.indexOf(' '));
00034                 stats = spaceBreak(stats);
00035                 String foreignAddress = stats.substring(0, stats.indexOf(' '));
00036                 stats = spaceBreak(stats);
00037                 String state = stats.substring(0, stats.indexOf(' '));
00038                 System.out.println(localAddress);
00039                 long pid = 0;
00040                 try
00041                 {
00042                     pid = Long.parseLong(state.substring(0, state.length()-1));
00043                 }
00044                 catch(NumberFormatException e)
00045                 {
00046                     stats = spaceBreak(stats);
00047                     pid = Long.parseLong(stats.substring(0,
stats.indexOf('\n')));
00048                 }
00049
00050                 ActiveConnection activeConnection = new
ActiveConnection(protocol, localAddress, foreignAddress, state, pid);
00051                 activeConnectionsList.add(activeConnection);
00052                 stats = stats.substring(stats.indexOf('\n')+3);
00053             }
00054             catch(StringIndexOutOfBoundsException e)
00055             {
00056                 //e.printStackTrace();
00057                 break;
00058             }
00059         }
00060         ActiveConnection[] activeConnections = new
ActiveConnection[activeConnectionsList.size()];
00061         for(int index = 0;index<activeConnections.length;index++)
00062         {
00063             activeConnections[index]=activeConnectionsList.get(index);
00064         }
00065         setActiveConnections(activeConnections);
00066     }
00067
00074     private String spaceBreak(String stats)
00075     {
00076         int index=stats.indexOf(' ')+1;
00077         while(stats.charAt(index)==' ') index++;
00078         return stats.substring(index);
00079     }
00080
00086     public ActiveConnection[] getActiveConnections()
00087     {
```

```
00088         return activeConnections;
00089     }
00090 
00096     public void setActiveConnections(ActiveConnection[] activeConnections)
00097     {
00098         this.activeConnections = activeConnections;
00099     }
00100 
00101 }
```

NetworkTraffic.java

```
00001 /*
00002  *
00003  */
00004 package dynamicAnalysis;
00005
00006 import java.io.EOFException;
00007 import java.io.IOException;
00008 import java.net.InetAddress;
00009 import java.util.List;
00010 import java.util.concurrent.TimeoutException;
00011
00012 import org.pcap4j.core.*;
00013 import org.pcap4j.core.BpfProgram.BpfCompileMode;
00014 import org.pcap4j.packet.IpPacket;
00015 import org.pcap4j.packet.Packet;
00016 import org.pcap4j.packet.TcpPacket;
00017 import org.pcap4j.packet.namednumber.IpNumber;
00018 import org.pcap4j.packet.namednumber.TcpPort;
00019 import org.pcap4j.packet.namednumber.UdpPort;
00020 import org.pcap4j.packet.IpV4Packet;
00021
00022
00026 public class NetworkTraffic {
00027
00033     public static void main(String[] args) {
00034         NetworkStats netStats = new NetworkStats();
00035         System.out.println(netStats.getActiveConnections()[0]);
00036         try {
00037             // Get a list of all available network interfaces
00038             List<PcapNetworkInterface> devices = Pcaps.findAllDevs();
00039
00040             // Create a separate PcapHandle for each interface and start
capturing traffic
00041             for (PcapNetworkInterface device : devices) {
00042                 //System.out.println(device.getName());
00043                 PcapHandle handle = device.openLive(65536,
PcapNetworkInterface.PromiscuousMode.PROMISCUOUS, 10);
00044                 try {
00045                     handle.setFilter("ip", BpfProgram.BpfCompileMode.OPTIMIZE);
00046                 } catch (NotOpenException e1) {
00047                     // TODO Auto-generated catch block
00048                     e1.printStackTrace();
00049                 }
00050
00051                 Thread listenerThread = new Thread(() -> {
00052                     try {
00053                         while (true) {
00054                             Packet packet = handle.getNextPacket();
00055                             if(packet!=null)
00056                             {
00057                                 if (packet.contains(IpPacket.class)) {
00058                                     IpPacket ipPacket =
packet.get(IpPacket.class);
00059                                     //System.out.println(ipPacket);
00060                                     //System.out.println("Port: "+ port.);
00061                                     // Now you can access the IP packet fields
00062                                     String srcAddr =
ipPacket.getHeader().getSrcAddr().getHostAddress();
00063                                    /* byte[] data =
ipPacket.getPayload().getRawData();
00064                                     for(int index = 0;index<data.length;index++)
00065                                     {
00066                                         System.out.print((char)data[index]+"");
00067                                     }*/
00068
//System.out.println(ipPacket.getHeader().getProtocol().name()+" -
"+ipPacket.getPayload().getRawData().length+" bytes {"+ipPacket.hashCode()+"}");
00069                                     String dstAddr =
ipPacket.getHeader().getDstAddr().getHostAddress();
00070                                     int protocol =
ipPacket.getHeader().getProtocol().value();
00071                                 }
00072                             }
```

```
00073
00074                         }
00075                     } catch (NotOpenException e) {
00076                         e.printStackTrace();
00077                     }
00078                 });
00079                 listenerThread.start();
00080             }
00081
00082         } catch (PcapNativeException e1) {
00083             // TODO Auto-generated catch block
00084             e1.printStackTrace();
00085         }
00086     }
00087
00088 }
00089 // netstat -ano
00090 // netstat -ano -p tcp - gets local address and port
00091 // Destination port always 49389 (unknown)
```

PacketTrace.java

```
00001 /*
00002  *
00003  */
00004 package dynamicAnalysis;
00005
00006 import java.util.ArrayList;
00007 import java.util.Collection;
00008 import java.util.ConcurrentModificationException;
00009 import java.util.HashMap;
00010 import java.util.List;
00011 import java.util.Map;
00012
00013 import org.pcap4j.core.BpfProgram;
00014 import org.pcap4j.core.NotOpenException;
00015 import org.pcap4j.core.PcapHandle;
00016 import org.pcap4j.core.PcapNativeException;
00017 import org.pcap4j.core.PcapNetworkInterface;
00018 import org.pcap4j.core.Pcaps;
00019 import org.pcap4j.packet.IpPacket;
00020 import org.pcap4j.packet.Packet;
00021
00022 import com.google.common.collect.ArrayListMultimap;
00023 import com.google.common.collect.Multimap;
00024
00025 import scala.collection.mutable.MultiMap;
00026
00030 public class PacketTrace
00031 {
00032
00034     private List<PcapNetworkInterface> devices;
00035
00037     private ArrayList<Multimap<String, IpPacket>> packets = new
ArrayList<Multimap<String, IpPacket>>();
00038
00040     private Multimap<String, String> addressMap = ArrayListMultimap.create();
00041
00043     private HashMap<String, Integer> interfaceMap = new HashMap<String,
Integer>();
00044
00046     private HashMap<String, String> idMap = new HashMap<String, String>();
00047
00049     private Thread listenerThread;
00050
00056     public PacketTrace() throws PcapNativeException
00057     {
00058         devices = Pcaps.findAllDevs();
00059         for(int index = 0;index < devices.size();index++)
00060         {
00061             idMap.put(devices.get(index).getName(),
devices.get(index).getDescription());
00062             interfaceMap.put(devices.get(index).getName(), index); // Separating
from multithread to avoid synchronization issues
00063             packets.add(ArrayListMultimap.create());
00064         }
00065         for (PcapNetworkInterface device : devices)
00066         {
00067             PcapHandle handle = device.openLive(65536,
PcapNetworkInterface.PromiscuousMode.PROMISCUOUS, 10);
00068             try {
00069                 handle.setFilter("ip", BpfProgram.BpfCompileMode.OPTIMIZE);
00070             } catch (NotOpenException e1) {
00071                 e1.printStackTrace();
00072             }
00073             listenerThread = new Thread(() -> { // Set up thread for each
network interface
00074                 try {
00075                     while (true) {
00076                         Packet packet = handle.getNextPacket();
00077                         if(packet!=null)
00078                         {
00079                             if (packet.contains(IpPacket.class)) {
00080                                 IpPacket ipPacket = packet.get(IpPacket.class);
```

```
00081
if(!addressMap.containsKey(ipPacket.getHeader().getSrcAddr().getHostAddress()))
addressMap.put(device.getName(), ipPacket.getHeader().getSrcAddr().getHostAddress());
00082
packets.get(interfaceMap.get(device.getName())).put(ipPacket.getHeader().getSrcAddr().
getHostAddress(), ipPacket); //needs address
00083                                 }
00084                             }
00085
00086                         }
00087                     } catch (NotOpenException e) {
00088                         e.printStackTrace();
00089                     }
00090                 });
00091                 listenerThread.start();
00092             }
00093         }
00094
00100         public HashMap<String, String> getDevices()
00101         {
00102             HashMap<String, String> activeDevices = new HashMap<String, String>();
00103             for(PcapNetworkInterface device : devices)
00104             {
00105                 if(addressMap.containsKey(device.getName()))
00106                 {
00107                     activeDevices.put(device.getName(), device.getDescription());
00108                 }
00109             }
00110             return activeDevices;
00111         }
00112
00119         public String[] getAddresses(String deviceName)
00120         {
00121             return addressMap.get(deviceName).toArray(new
String[addressMap.size()]);
00122         }
00123
00129         public ArrayList<Multimap<String, IpPacket>> getPackets()
00130         {
00131             return packets;
00132         }
00133
00141         public ArrayList<IpPacket> getPackets(String address, int attempts)
00142         {
00143             final int TIMEOUTCOUNTER = 10;
00144             ArrayList<IpPacket> packetList = new ArrayList<IpPacket>();
00145             Collection<IpPacket> packetCollection;
00146             for(int index = 0;index < packets.size();index++)
00147             {
00148                 packetCollection = packets.get(index).get(address);
00149                 try
00150                 {
00151                     for(IpPacket packet : packetCollection)
00152                     {
00153                         packetList.add(packet);
00154                     }
00155                 } catch (ConcurrentModificationException e)
00156                 {
00157                     if(attempts > TIMEOUTCOUNTER) return null;
00158                     else return getPackets(address, attempts+1);
00159                 }
00160
00161             }
00162             return packetList;
00163         }
00164
00165 }
```

PEFile.java

```
00001 /*
00002  *
00003  */
00004 package dynamicAnalysis;
00005
00006 import java.io.File;
00007 import java.io.IOException;
00008 import java.nio.file.Files;
00009 import java.nio.file.Paths;
00010 import java.util.concurrent.Semaphore;
00011
00012 import org.eclipse.swt.SWT;
00013 import org.eclipse.swt.widgets.Display;
00014 import org.eclipse.swt.widgets.TableItem;
00015
00019 public class PEFile {
00020
00022     private File file;
00023
00025     private int offset;
00026
00028     private Version version;
00029
00031     private int pointer;
00032
00034     private byte[] bytes = null;
00035
00037     private int pointerToRawData;
00038
00044     public PEFile(File file)
00045     {
00046         setFile(file);
00047     }
00048
00054     public File getFile()
00055     {
00056         return file;
00057     }
00058
00064     public void setFile(File file)
00065     {
00066         this.file = file;
00067     }
00068
00074     public int getOffset()
00075     {
00076         return offset;
00077     }
00078
00084     public Version getVersion()
00085     {
00086         return version;
00087     }
00088
00094     public byte[] getInstructions()
00095     {
00096         int bytesIndex = offset;
00097         while(true)
00098         {
00099             if(bytes[bytesIndex]==0x2e && bytes[bytesIndex+1] == 0x74 &&
bytes[bytesIndex+2] == 0x65 && bytes[bytesIndex+3] == 0x78 && bytes[bytesIndex+4] ==
0x74) break;
00100             bytesIndex++;
00101         }
00102         bytesIndex+=16;
00103         int sizeOfRawData = ((bytes[bytesIndex] & 0xff) | (bytes[bytesIndex+1] &
0xff) << 8 | (bytes[bytesIndex+2] & 0xff) << 16 | (bytes[bytesIndex+3] & 0xff) << 24);
00104         bytesIndex+=4;
00105         pointerToRawData = ((bytes[bytesIndex] & 0xff) | (bytes[bytesIndex+1] &
0xff) << 8 | (bytes[bytesIndex+2] & 0xff) << 16 | (bytes[bytesIndex+3] & 0xff) << 24);
00106         setPointer(pointerToRawData);
00107         byte[] instructions = new byte[sizeOfRawData];
```

```
00108         for(int index = pointerToRawData;index < sizeOfRawData +
pointerToRawData;index++)
00109         {
00110             instructions[index-pointerToRawData] = (byte)(bytes[index] & 0xff);
00111             if(index-pointerToRawData <100) System.out.print(instructions[index-
pointerToRawData] + " ");
00112         }
00113         return instructions;
00114     }
00115
00121     public int getPointer()
00122     {
00123         return pointer;
00124     }
00125
00131     private void setPointer(int pointer)
00132     {
00133         this.pointer = pointer;
00134     }
00135
00142     public byte[] getBytes() throws IOException
00143     {
00144         return Files.readAllBytes(Paths.get(file.toString()));
00145     }
00146
00152     public void setBytes(byte[] bytes)
00153     {
00154         this.bytes = bytes;
00155     }
00156
00160     public void readFile()
00161     {
00162
00163         try
00164         {
00165             bytes = Files.readAllBytes(Paths.get(file.toString()));
00166             offset = (((bytes[0x3c] & 0xff) | (bytes[0x3d] & 0xff) << 8 |
(bytes[0x3e] & 0xff) << 16 | (bytes[0x3f] & 0xff) << 24)) + 24;
00167             if(bytes[offset+1]==0x01) version = Version.x32;
00168             else if (bytes[offset+1]==0x02) version = Version.x64;
00169             else System.out.println("ERROR finding version");
00170             int rvaOffset;
00171             if(version == Version.x32) rvaOffset = offset + 92;
00172             else rvaOffset = offset + 108;
00173             int numberOfRvaAndSizes = ((bytes[rvaOffset] & 0xff) |
(bytes[rvaOffset+1] & 0xff) << 8 | (bytes[rvaOffset+2] & 0xff) << 16 |
(bytes[rvaOffset+3] & 0xff) << 24);
00174             DataDirectory dataDirectories[] = new
DataDirectory[numberOfRvaAndSizes];
00175             int directoryOffset;
00176             if(version == Version.x32) directoryOffset = offset + 96;
00177             else directoryOffset = offset + 112;
00178             int virtualAddress;
00179             int size;
00180             for(int index = 0; index < numberOfRvaAndSizes; index++)
00181             {
00182                 virtualAddress = ((bytes[directoryOffset] & 0xff) |
(bytes[directoryOffset+1] & 0xff) << 8 | (bytes[directoryOffset+2] & 0xff) << 16 |
(bytes[directoryOffset+3] & 0xff) << 24);
00183                 size = ((bytes[directoryOffset+4] & 0xff) |
(bytes[directoryOffset+5] & 0xff) << 8 | (bytes[directoryOffset+6] & 0xff) << 16 |
(bytes[directoryOffset+7] & 0xff) << 24);
00184                 dataDirectories[index] = new DataDirectory(bytes,
virtualAddress, size);
00185             }
00186         }
00187         catch (IOException | NullPointerException e)
00188         {
00189             e.printStackTrace();
00190         }
00191     }
00192
00198     @Override
00199     public String toString()
00200     {
00201         return "PEFile [file=" + file + ", offset=" + offset + ", version=" +
version + "]";
```

```
00202     }
00203
00204 }
```

ProcessManager.java

```
00001 /*
00002  *
00003  */
00004 package dynamicAnalysis;
00005
00006 import java.io.File;
00007 import java.io.IOException;
00008 import java.util.Arrays;
00009
00013 public class ProcessManager
00014 {
00015
00017     private File file;
00018
00020     private Process process;
00021
00023     private CommandLine commandLine;
00024
00026     private String name;
00027
00029     private String[] dlls;
00030
00032     private DllFile[] dllFiles;
00033
00035     private String[] files;
00036
00042     public ProcessManager(File file)
00043     {
00044         setFile(file);
00045         createProcess();
00046         commandLine = new CommandLine(getPid());
00047         setName();
00048         setDLLs();
00049     }
00050
00056     public ProcessManager(int pid)
00057     {
00058         commandLine = new CommandLine(pid);
00059         setName();
00060         setDLLs();
00061         setFiles();
00062     }
00063
00069     public File getFile()
00070     {
00071         return file;
00072     }
00073
00079     public void setFile(File file)
00080     {
00081         this.file = file;
00082     }
00083
00089     public Process createProcess()
00090     {
00091         ProcessBuilder builder = new
ProcessBuilder(getFile().getAbsolutePath());
00092         process = null;
00093         try
00094         {
00095             process = builder.start();
00096         } catch (IOException e)
00097         {
00098             e.printStackTrace();
00099
00100         }
00101         System.out.println("children: "+process.children());
00102         System.out.println("info: " + process.info());
00103         return process;
00104     }
00105
00111     public long getPid()
00112     {
```

```
00113         return process.pid();
00114     }
00115
00121     public String getPidAsString()
00122     {
00123         return Long.toString(process.pid());
00124     }
00125
00129     private void setName()
00130     {
00131         String name = commandLine.runName();
00132         try
00133         {
00134             this.name = name.substring(name.lastIndexOf("\"",
name.indexOf(".exe"))+1, name.lastIndexOf("\"", name.indexOf(".exe")+4));
00135         }
00136         catch(StringIndexOutOfBoundsException e)
00137         {
00138             e.printStackTrace();
00139             this.name = "Error";
00140         }
00141     }
00142
00146     private void setDLLs()
00147     {
00148         String dllString = commandLine.runDLLs();
00149         int count=0;
00150         for(int index=dllString.indexOf('\n',
dllString.indexOf("Path")+5)+1;index<dllString.length();index++)
00151         {
00152             if(dllString.charAt(index)=='\n')
00153             {
00154                 count++;
00155             }
00156         }
00157         String DLLs[] = new String[count];
00158         int index=dllString.indexOf('\n', dllString.indexOf("Path")+5)+1;
00159         int dllIndex=0;
00160         while(count!=1)
00161         {
00162             try
00163             {
00164                 DLLs[dllIndex]=dllString.substring(dllString.lastIndexOf(' ',
dllString.indexOf('\n', index)), dllString.lastIndexOf('\n', dllString.indexOf('\n',
index)));
00165             } catch(StringIndexOutOfBoundsException | IllegalArgumentException
e)
00166             {
00167                 DLLs[dllIndex] = "";
00168                 e.printStackTrace();
00169             }
00170             index=dllString.indexOf('\n', index+1);
00171             dllIndex++;
00172             count--;
00173         }
00174         DLLs[dllIndex]=dllString.substring(dllString.lastIndexOf(' '));
00175         dllFiles = new DllFile[DLLs.length];
00176         for(int j = 0;j<DLLs.length;j++)
00177         {
00178             dllFiles[j] = new DllFile(DLLs[j]);
00179         }
00180         this.dlls = DLLs;
00181     }
00182
00188     public String getName()
00189     {
00190         return this.name;
00191     }
00192
00198     public String[] getDLLs()
00199     {
00200         return this.dlls;
00201     }
00202
00208     public DllFile[] getDllFiles()
00209     {
00210         return dllFiles;
```

```
00211     }
00212
00218     public String[] getFiles()
00219     {
00220         return files;
00221     }
00222
00226     private void setFiles()
00227     {
00228         String filesOutput = commandLine.runFiles();
00229         int count=0;
00230         int index = filesOutput.indexOf("File,");
00231         while (index != -1)
00232         {
00233             count++;
00234             index = filesOutput.indexOf("File,", index+1);
00235         }
00236         String filesArr[] = new String[count];
00237         String file="";
00238         index=filesOutput.indexOf("Name")+7;
00239         int filesIndex=0;
00240         while(count!=0)
00241         {
00242             try
00243             {
00244                 file=filesOutput.substring(filesOutput.indexOf("File,",index)+6,
filesOutput.indexOf(10, index+1));
00245             }
00246             catch(StringIndexOutOfBoundsException e)
00247             {
00248
file=filesOutput.substring(filesOutput.indexOf("File,",index)+6);
00249             }
00250             filesArr[filesIndex]=file;
00251             filesIndex++;
00252             index=filesOutput.indexOf(10, index+1);
00253             System.out.println("index: "+index);
00254             count--;
00255             System.out.println(count);
00256         }
00257         this.files = filesArr;
00258     }
00259
00265     public Process getProcess()
00266     {
00267         return process;
00268     }
00269
00275     public void setProcess(Process process)
00276     {
00277         this.process = process;
00278     }
00279
00285     public boolean destroyProcess()
00286     {
00287         Process process = getProcess();
00288         if(process == null) return false;
00289         process = process.destroyForcibly();
00290         setProcess(process);
00291         return true;
00292     }
00293
00299     @Override
00300     public String toString()
00301     {
00302         return "ProcessManager [file=" + file + ", process=" + process + ",
commandLine=" + commandLine + ", name="
00303                 + name + ", dlls=" + Arrays.toString(dlls) + ", dllFiles=" +
Arrays.toString(dllFiles) + "]";
00304     }
00305
00306 }
```

ReadWrite.java

```
00001 /*
00002  *
00003  */
00004 package dynamicAnalysis;
00005 import java.io.*;
00006
00010 public class ReadWrite
00011 {
00012
00019      public static void write(String word, String file) //Method for clearing a
text file and writing a line.
00020     {
00021         PrintWriter delete=null;
00022         try
00023             {
00024                 delete = new PrintWriter(file);
00025                 FileWriter writer = new FileWriter(file);
00026                 delete.print("");
00027                 writer.write(word);
00028                 writer.close();
00029             }
00030         catch (IOException e)
00031             {
00032                 e.printStackTrace();
00033             }
00034         finally
00035         {
00036             delete.close();
00037         }
00038     }
00039
00046      public static void writeLine(String word, String file) //Method for adding
a line to a text file.
00047     {
00048         try
00049             {
00050                 FileWriter writer = new FileWriter(file, true);
00051                 writer.write(word+"\n");
00052                 writer.close();
00053             }
00054         catch (IOException e)
00055             {
00056                 e.printStackTrace();
00057             }
00058     }
00059
00065      public static void delete(String file) //Method for clearing a text file.
00066     {
00067         write("", file);
00068     }
00069
00076      public static int getLength(String file) //Returns how many lines are in
the text file.
00077     {
00078         int count=1;
00079         @SuppressWarnings("unused")
00080         String dummy;
00081         BufferedReader checkLine=null;
00082         try
00083         {
00084         FileReader reader = new FileReader(file);
00085         checkLine=new BufferedReader(reader);
00086         while (true)
00087                 {
00088                     dummy=checkLine.readLine().toString();
00089                     if (!(checkLine.ready()))
00090                     {
00091                         break;
00092                     }
00093                     else
00094                     {
00095                         count++;
00096                     }
```

```
00097                 }
00098         }
00099         catch (IOException e)
00100             {
00101                 e.printStackTrace();
00102             }
00103         return count;
00104     }
00105 
00113      public static String getLine(int number, String file) //Method for
returning a line from a text file.
00114     {
00115         String word="";
00116         BufferedReader checkLine=null;
00117         FileReader reader=null;
00118         try
00119             {
00120             try
00121             {
00122                 reader = new FileReader(file);
00123             }
00124                 catch (FileNotFoundException e)
00125             {
00126                     return "";
00127             }
00128                 checkLine=new BufferedReader(reader);
00129                 for (int index=-1;index<number;index++)
00130                     {
00131                         word=checkLine.readLine().toString();
00132 
00133                     }
00134                 reader.close();
00135             }
00136         catch (IOException e)
00137             {
00138                 e.printStackTrace();
00139             }
00140         return word;
00141     }
00142 
00150      public static int indexOf(String word, String file) //Returns the index
that the word is at.
00151     {
00152         int index=-1;
00153         String temp;
00154         BufferedReader checkLine=null;
00155         try
00156             {
00157                 FileReader reader = new FileReader(file);
00158                 checkLine=new BufferedReader(reader);
00159                 while (true)
00160                 {
00161                     if (!(checkLine.ready()))
00162                     {
00163                         index=-1;
00164                         break;
00165                     }
00166                     index++;
00167                     temp=checkLine.readLine().toString();
00168                     if (temp.equals(word))
00169                     {
00170                         break;
00171                     }
00172 
00173                 }
00174             }
00175         catch (IOException e)
00176             {
00177                 e.printStackTrace();
00178             }
00179         return index;
00180     }
00181 
00189      public static void replace(String oldWord, String newWord, String file)
//Replaces the old word with the new word in a text file.
00190     {
00191         delete("temp.txt");
```

```
00192         String temp="";
00193         BufferedReader checkLine;
00194         try
00195             {
00196                 FileReader reader = new FileReader(file);
00197                 checkLine=new BufferedReader(reader);
00198                 while (true)
00199                 {
00200                     temp=checkLine.readLine().toString();
00201                     if (!(temp.equals(oldWord)))
00202                     {
00203                         writeLine(temp,"temp.txt");
00204                     }
00205                     else
00206                     {
00207                         writeLine(newWord,"temp.txt");
00208                     }
00209                     if (!(checkLine.ready()))
00210                     {
00211                         break;
00212                     }
00213                 }
00214                 delete(file);
00215                 reader = new FileReader("temp.txt");
00216                 checkLine=new BufferedReader(reader);
00217                 while (true)
00218                 {
00219                     writeLine(checkLine.readLine().toString(), file);
00220                     if (!(checkLine.ready()))
00221                     {
00222                         break;
00223                     }
00224                 }
00225 
00226             }
00227         catch (IOException e)
00228             {
00229                 e.printStackTrace();
00230             }
00231     }
00232 
00240      public static void replace(int index, String newWord, String file)
//Replaces the word at specified index with the new word.
00241     {
00242         delete("temp.txt");
00243         int increment=-1;
00244         String temp="";
00245         BufferedReader checkLine;
00246         try
00247             {
00248 
00249                 FileReader reader = new FileReader(file);
00250                 checkLine=new BufferedReader(reader);
00251                 while (true)
00252                 {
00253                     increment++;
00254                     temp=checkLine.readLine().toString();
00255                     if (increment!=index)
00256                     {
00257                         writeLine(temp,"temp.txt");
00258                     }
00259                     else
00260                     {
00261                         writeLine(newWord,"temp.txt");
00262                     }
00263                     if (!(checkLine.ready()))
00264                     {
00265                         break;
00266                     }
00267                 }
00268                 delete(file);
00269                 reader = new FileReader("temp.txt");
00270                 checkLine=new BufferedReader(reader);
00271                 while (true)
00272                 {
00273                     writeLine(checkLine.readLine().toString(), file);
00274                     if (!(checkLine.ready()))
```

```
00275                     {
00276                         break;
00277                     }
00278                 }
00279 
00280             }
00281         catch (IOException e)
00282             {
00283                 e.printStackTrace();
00284             }
00285     }
00286 
00293      public static boolean isReady(String file) //Check if text file is ready to
be written.
00294     {
00295         boolean check=true;
00296         BufferedReader checkLine;
00297         try
00298         {
00299             FileReader reader = new FileReader(file);
00300             checkLine=new BufferedReader(reader);
00301             check=checkLine.ready();
00302         }
00303         catch(IOException e)
00304         {
00305             e.printStackTrace();
00306         }
00307         return check;
00308     }
00309 
00316      public static void deleteLine(int index, String file) //Deletes the line at
a specified index.
00317     {
00318         delete("temp.txt");
00319         int increment=-1;
00320         String temp="";
00321         BufferedReader checkLine;
00322         try
00323             {
00324 
00325                 FileReader reader = new FileReader(file);
00326                 checkLine=new BufferedReader(reader);
00327                 while (true)
00328                 {
00329                     increment++;
00330                     temp=checkLine.readLine().toString();
00331                     if (increment!=index)
00332                     {
00333                         writeLine(temp,"temp.txt");
00334                     }
00335                     if (!(checkLine.ready()))
00336                     {
00337                         break;
00338                     }
00339                 }
00340                 delete(file);
00341                 reader = new FileReader("temp.txt");
00342                 checkLine=new BufferedReader(reader);
00343                 while (true)
00344                 {
00345                     writeLine(checkLine.readLine().toString(), file);
00346                     if (!(checkLine.ready()))
00347                     {
00348                         break;
00349                     }
00350                 }
00351 
00352             }
00353         catch (IOException e)
00354             {
00355                 e.printStackTrace();
00356             }
00357 
00358     }
00359 
00366      public static String toString(String file)
00367     {
```

```
00368         BufferedReader checkLine;
00369         String total="";
00370         try
00371         {
00372         FileReader reader = new FileReader(file);
00373         checkLine=new BufferedReader(reader);
00374         while (true)
00375         {
00376             total+=checkLine.readLine().toString()+"\n";
00377             if (!(checkLine.ready()))
00378             {
00379                 break;
00380             }
00381 
00382         }
00383         checkLine.close();
00384         }
00385         catch(IOException e)
00386         {
00387             e.printStackTrace();
00388         }
00389         return total;
00390     }
00391 }
```

SelectFile.java

```
00001 /*
00002  *
00003  */
00004 package dynamicAnalysis;
00005
00006 import org.eclipse.swt.widgets.Display;
00007 import org.eclipse.swt.widgets.FileDialog;
00008 import org.eclipse.swt.widgets.Shell;
00009 import org.eclipse.swt.widgets.Text;
00010
00011 import java.awt.Dimension;
00012 import java.awt.Toolkit;
00013
00014 import org.eclipse.swt.SWT;
00015 import org.eclipse.swt.widgets.Button;
00016 import org.eclipse.wb.swt.SWTResourceManager;
00017 import org.eclipse.swt.events.SelectionAdapter;
00018 import org.eclipse.swt.events.SelectionEvent;
00019 import org.eclipse.swt.widgets.Label;
00020
00024 public class SelectFile
00025 {
00026
00028     protected Shell shell;
00029
00031     private Text text;
00032
00034     private String filePath;
00035
00037     private int x;
00038
00040     private int y;
00041
00043     private boolean pidMode;
00044
00046     private int pid;
00047
00055     public SelectFile(int x, int y, boolean pidMode)
00056     {
00057         setX(x);
00058         setY(y);
00059         setPidMode(pidMode);
00060         open();
00061     }
00062
00066     public void open()
00067     {
00068         Display display = Display.getDefault();
00069         createContents();
00070         shell.open();
00071         shell.layout();
00072         while (!shell.isDisposed())
00073         {
00074             try
00075             {
00076                 if (!display.readAndDispatch())
00077                 {
00078                     display.sleep();
00079                 }
00080             }
00081             catch(IllegalArgumentException e) {}
00082         }
00083     }
00084
00088     protected void createContents()
00089     {
00090         System.out.println(getX());
00091         shell = new Shell();
00092         shell.setBackground(SWTResourceManager.getColor(192, 192, 192));
00093         shell.setSize(400, 180);
00094
00095         if(!isPidMode()) shell.setText("Choose a file");
00096         else shell.setText("Choose a Process");
```

```
00097
00098         shell.setLocation(getX()+200,getY()+90);
00099
00100         text = new Text(shell, SWT.BORDER);
00101         text.setBackground(SWTResourceManager.getColor(255, 255, 255));
00102         text.setBounds(69, 38, 250, 25);
00103
00104         Button btnOk = new Button(shell, SWT.NONE);
00105         btnOk.addSelectionListener(new SelectionAdapter() {
00106             @Override
00107             public void widgetSelected(SelectionEvent e) {
00108                 if(!isPidMode())
00109                 {
00110                     filePath = text.getText();
00111                     shell.dispose();
00112                 }
00113                 else
00114                 {
00115                     setPid(Integer.parseInt(text.getText()));
00116                     System.out.println("pid: "+getPid());
00117                     shell.dispose();
00118                 }
00119             }
00120         });
00121         btnOk.setBounds(315, 105, 46, 25);
00122         btnOk.setText("OK");
00123
00124         Button btnSelectFile = new Button(shell, SWT.NONE);
00125         btnSelectFile.addSelectionListener(new SelectionAdapter() {
00126             @Override
00127             public void widgetSelected(SelectionEvent e) {
00128                 if(!isPidMode())
00129                 {
00130                     FileDialog fileDialog = new FileDialog(shell, SWT.MULTI);
00131                     String[] files = {
00132                             "*.exe",
00133                         };
00134                         fileDialog.setFilterExtensions(files);
00135                         text.setText(fileDialog.open());
00136                 }
00137                 else
00138                 {
00139                     CommandLine commandLine = new CommandLine();
00140                     String output = commandLine.getAll();
00141                     int breakCount = 0;
00142
00143                     for(int index = 0;index<output.length();index++)
00144                     {
00145                         if(output.charAt(index)=='\n')
00146                         {
00147                             breakCount++;
00148                         }
00149                     }
00150
00151                     String[] names = new String[breakCount-2];
00152                     int[] pids = new int[breakCount-2];
00153                     String[] memory = new String[breakCount-2];
00154                     int outputIndex = output.indexOf('\n');
00155                     int arrayIndex = 0;
00156                     while(outputIndex <= output.length())
00157                     {
00158                         try
00159                         {
00160                             names[arrayIndex]=output.substring(outputIndex+2,
output.indexOf('\"', outputIndex+2));
00161
pids[arrayIndex]=Integer.parseInt(output.substring(output.indexOf(',', outputIndex)+2,
output.indexOf('\"', output.indexOf(',', outputIndex)+2))); //stuck here
00162
memory[arrayIndex]=output.substring(output.lastIndexOf("\"", output.indexOf("\n",
outputIndex+1)-2)+1, output.lastIndexOf("\"", output.indexOf("\n", outputIndex+1)));
00163                             outputIndex=output.indexOf('\n', outputIndex+1);
00164                         }
00165                         catch(ArrayIndexOutOfBoundsException e1)
00166                         {
00167                             break;
00168                         }
```

```
00169                         arrayIndex++;
00170                     }
00171                     SelectProcess selectProcess = new SelectProcess(names, pids,
memory, shell.getLocation().x, shell.getLocation().y);
00172                     selectProcess.open();
00173                     if(selectProcess.getPid() == 0) text.setText("");
00174                     else text.setText(Integer.toString(selectProcess.getPid()));
00175                 }
00176             }
00177         });
00178         btnSelectFile.setBounds(235, 105, 74, 25);
00179         btnSelectFile.setText("Select File");
00180 
00181         Label lblFileLocation = new Label(shell, SWT.NONE);
00182         lblFileLocation.setBackground(SWTResourceManager.getColor(192, 192,
192));
00183         lblFileLocation.setBounds(161, 17, 67, 15);
00184         lblFileLocation.setText("File Location");
00185 
00186         if(isPidMode())
00187         {
00188             setPidMode(true);
00189             lblFileLocation.setText("Process ID");
00190             //btnEnterPid.setText("Enter File Location");
00191             btnSelectFile.setText("Select Process");
00192             //btnEnterPid.setBounds(116, 105, 103, 25);
00193             btnSelectFile.setBounds(223, 105, 86, 25);
00194 
00195         }
00196         else
00197         {
00198             setPidMode(false);
00199             lblFileLocation.setText("File Location");
00200             //btnEnterPid.setText("Enter PID");
00201             btnSelectFile.setText("Select File");
00202             lblFileLocation.setBounds(161, 17, 67, 15);
00203             btnSelectFile.setBounds(235, 105, 74, 25);
00204             //btnEnterPid.setBounds(162, 105, 67, 25);
00205 
00206         }
00207 
00208         Button btnEnterPid = new Button(shell, SWT.NONE);
00209         btnEnterPid.setVisible(false);
00210         btnEnterPid.setSelection(true);
00211         btnEnterPid.addSelectionListener(new SelectionAdapter() {
00212             @Override
00213             public void widgetSelected(SelectionEvent e) {
00214                 if(!isPidMode())
00215                 {
00216                     setPidMode(true);
00217                     lblFileLocation.setText("Process ID");
00218                     btnEnterPid.setText("Enter File Location");
00219                     btnSelectFile.setText("Select Process");
00220                     btnEnterPid.setBounds(116, 105, 103, 25);
00221                     btnSelectFile.setBounds(223, 105, 86, 25);
00222 
00223                 }
00224                 else
00225                 {
00226                     setPidMode(false);
00227                     lblFileLocation.setText("File Location");
00228                     btnEnterPid.setText("Enter PID");
00229                     btnSelectFile.setText("Select File");
00230                     lblFileLocation.setBounds(161, 17, 67, 15);
00231                     btnSelectFile.setBounds(235, 105, 74, 25);
00232                     btnEnterPid.setBounds(162, 105, 67, 25);
00233 
00234                 }
00235             }
00236         });
00237         btnEnterPid.setBounds(162, 105, 67, 25);
00238         btnEnterPid.setText("Enter PID");
00239 
00240     }
00241 
00247     public int getX()
00248     {
```

```
00249         return x;
00250     }
00251
00257     public void setX(int x)
00258     {
00259         this.x = x;
00260     }
00261
00267     public int getY()
00268     {
00269         return y;
00270     }
00271
00277     public void setY(int y)
00278     {
00279         this.y = y;
00280     }
00281
00287     public boolean isPidMode()
00288     {
00289         return pidMode;
00290     }
00291
00297     public void setPidMode(boolean pidMode)
00298     {
00299         this.pidMode = pidMode;
00300     }
00301
00307     public String getText()
00308     {
00309         return filePath;
00310     }
00311
00317     public int getPid()
00318     {
00319         return pid;
00320     }
00321
00327     public void setPid(int pid)
00328     {
00329         this.pid = pid;
00330     }
00331
00337     public boolean isDisposed()
00338     {
00339         return shell.isDisposed();
00340     }
00341
00345     public void focus()
00346     {
00347         shell.forceFocus();
00348     }
00349 }
00350
00351
```

SelectProcess.java

```
00001 /*
00002  *
00003  */
00004 package dynamicAnalysis;
00005
00006 import java.util.ArrayList;
00007
00008 import org.eclipse.swt.SWT;
00009 import org.eclipse.swt.layout.GridData;
00010 import org.eclipse.swt.layout.GridLayout;
00011 import org.eclipse.swt.widgets.Button;
00012 import org.eclipse.swt.widgets.Display;
00013 import org.eclipse.swt.widgets.Event;
00014 import org.eclipse.swt.widgets.Shell;
00015 import org.eclipse.swt.widgets.Table;
00016 import org.eclipse.swt.widgets.TableColumn;
00017 import org.eclipse.swt.widgets.TableItem;
00018 import org.eclipse.swt.widgets.Label;
00019 import org.eclipse.swt.widgets.Listener;
00020 import org.eclipse.swt.events.KeyAdapter;
00021 import org.eclipse.swt.events.KeyEvent;
00022 import org.eclipse.swt.events.SelectionAdapter;
00023 import org.eclipse.swt.events.SelectionEvent;
00024 import org.eclipse.swt.layout.FormLayout;
00025 import org.eclipse.swt.layout.FormData;
00026 import org.eclipse.swt.layout.FormAttachment;
00027 import org.eclipse.swt.widgets.Text;
00028 import org.eclipse.wb.swt.SWTResourceManager;
00029
00033 public class SelectProcess
00034 {
00035
00037     protected Shell shell;
00038
00040     private Table table;
00041
00043     private String[] names;
00044
00046     private int[] pids;
00047
00049     private String[] memory;
00050
00052     private int pid;
00053
00055     private int x;
00056
00058     private int y;
00059
00061     private Text text;
00062
00064     private TableItem[] tableItems;
00065
00076     public SelectProcess(String[] names, int[] pids, String[] memory, int x, int
y)
00077     {
00078         setNames(names);
00079         setPids(pids);
00080         setMemory(memory);
00081         setX(x);
00082         setY(y);
00083     }
00084
00088     public void open()
00089     {
00090
00091         Display display = Display.getDefault();
00092         createContents();
00093         shell.open();
00094         shell.layout();
00095         while (!shell.isDisposed())
00096         {
00097             if (!display.readAndDispatch())
00098             {
```

```
00099                 display.sleep();
00100             }
00101         }
00102     }
00103 
00109     public String[] getNames()
00110     {
00111         return names;
00112     }
00113 
00119     public void setNames(String[] names)
00120     {
00121         this.names = names;
00122     }
00123 
00129     public int[] getPids()
00130     {
00131         return pids;
00132     }
00133 
00139     public void setPids(int[] pids)
00140     {
00141         this.pids = pids;
00142     }
00143 
00149     public int getPid()
00150     {
00151         return pid;
00152     }
00153 
00159     private void setPid(int pid)
00160     {
00161         this.pid = pid;
00162     }
00163 
00169     public String[] getMemory()
00170     {
00171         return memory;
00172     }
00173 
00179     public void setMemory(String[] memory)
00180     {
00181         this.memory = memory;
00182     }
00183 
00189     public int getX()
00190     {
00191         return x;
00192     }
00193 
00199     public void setX(int x)
00200     {
00201         this.x = x;
00202     }
00203 
00209     public int getY()
00210     {
00211         return y;
00212     }
00213 
00219     public void setY(int y)
00220     {
00221         this.y = y;
00222     }
00223 
00231     private int[] search(String[] names, String toSearch)
00232     {
00233         ArrayList<Integer> arrayList = new ArrayList<Integer>();
00234         for(int index = 0; index < names.length; index++)
00235         {
00236             if(names[index].toLowerCase().contains(toSearch.toLowerCase()))
00237             {
00238                 arrayList.add(index);
00239             }
00240         }
00241         return arrayList.stream().mapToInt(i -> i).toArray();
00242     }
```

```
00243
00249     private void fullPopulate(Table table)
00250     {
00251         table.setItemCount(0);
00252         tableItems = new TableItem[names.length];
00253         for(int index = 0;index < tableItems.length;index++)
00254         {
00255             try
00256             {
00257                 tableItems[index] = new TableItem(table, SWT.NULL);
00258                 tableItems[index].setText(0, getNames()[index]);
00259                 tableItems[index].setText(1,
Integer.toString(getPids()[index]));
00260                 tableItems[index].setText(2, getMemory()[index]);
00261             }
00262             catch(IllegalArgumentException e)
00263             {
00264                 e.printStackTrace();
00265                 continue;
00266             }
00267         }
00268     }
00269
00273     private void searchEvent()
00274     {
00275         if(text.getText()=="")
00276         {
00277             fullPopulate(table);
00278         }
00279         else
00280         {
00281             int[] indexes = search(getNames(), text.getText());
00282             table.setItemCount(0);
00283             for(int index=0;index<indexes.length;index++)
00284             {
00285                 tableItems[index] = new TableItem(table, SWT.NULL);
00286                 tableItems[index].setText(0, getNames()[indexes[index]]);
00287                 tableItems[index].setText(1,
Integer.toString(getPids()[indexes[index]]));
00288                 tableItems[index].setText(2, getMemory()[indexes[index]]);
00289             }
00290         }
00291     }
00292
00296     protected void createContents()
00297     {
00298         try
00299         {
00300
00301             shell = new Shell();
00302             shell.setSize(450, 300);
00303             shell.setText("Processes");
00304             shell.setLocation(getX(), getY());
00305             shell.setLayout(new FormLayout());
00306             Button btnNewButton = new Button(shell, SWT.NONE);
00307             FormData fd_btnNewButton = new FormData();
00308             fd_btnNewButton.top = new FormAttachment(0, 5);
00309             fd_btnNewButton.left = new FormAttachment(0, 386);
00310             btnNewButton.setLayoutData(fd_btnNewButton);
00311
00312             table = new Table(shell, SWT.BORDER | SWT.FULL SELECTION);
00313             FormData fd_table = new FormData();
00314             fd_table.bottom = new FormAttachment(0, 256);
00315             fd_table.right = new FormAttachment(0, 429);
00316             fd_table.top = new FormAttachment(0, 35);
00317             fd_table.left = new FormAttachment(0, 5);
00318             table.setLayoutData(fd table);
00319             table.setHeaderVisible(true);
00320             table.setLinesVisible(true);
00321             table.addListener(SWT.SELECTED, new Listener() {
00322                 public void handleEvent(Event e) {
00323                     btnNewButton.setEnabled(true);
00324                 }
00325             });
00326             table.addListener(SWT.DefaultSelection, new Listener() {
00327                 public void handleEvent(Event e) {
```

```
00328
setPid(Integer.parseInt(table.getSelection()[0].getText(1)));
00329                     shell.dispose();
00330                 }
00331             });
00332
00333             int width = 140;
00334
00335             TableColumn tableNames = new TableColumn(table, SWT.CENTER);
00336             tableNames.setWidth(width);
00337             tableNames.setText("Name");
00338
00339             TableColumn tablePids = new TableColumn(table, SWT.CENTER);
00340             tablePids.setWidth(width);
00341             tablePids.setText("Process ID");
00342
00343             TableColumn tableMemory = new TableColumn(table, SWT.CENTER);
00344             tableMemory.setWidth(width);
00345             tableMemory.setText("Memory");
00346
00347             fullPopulate(table);
00348
00349             System.out.println("table items: "+tableItems.length);
00350
00351             btnNewButton.addSelectionListener(new SelectionAdapter() {
00352                 @Override
00353                 public void widgetSelected(SelectionEvent e) {
00354
00355                 }
00356             });
00357             btnNewButton.setText("Select");
00358             btnNewButton.setEnabled(false);
00359
00360             Button btnSearch = new Button(shell, SWT.NONE);
00361             btnSearch.addSelectionListener(new SelectionAdapter() {
00362                 @Override
00363                 public void widgetSelected(SelectionEvent e) {
00364                     searchEvent();
00365                 }
00366             });
00367             FormData fd_btnSearch = new FormData();
00368             fd_btnSearch.top = new FormAttachment(btnNewButton, 0, SWT.TOP);
00369             fd_btnSearch.right = new FormAttachment(btnNewButton, -232);
00370             btnSearch.setLayoutData(fd_btnSearch);
00371             btnSearch.setText("Search");
00372
00373             Button button = new Button(shell, SWT.NONE);
00374             button.addSelectionListener(new SelectionAdapter() {
00375                 @Override
00376                 public void widgetSelected(SelectionEvent e) {
00377                     fullPopulate(table);
00378                     text.setText("");
00379                 }
00380             });
00381             fd_btnSearch.left = new FormAttachment(button, 6);
00382             button.setImage(SWTResourceManager.getImage(SelectProcess.class,
"/icons/full/message_error.png"));
00383             FormData fd_button = new FormData();
00384             button.setLayoutData(fd_button);
00385
00386             text = new Text(shell, SWT.BORDER);
00387             FormData fd_text = new FormData();
00388             fd_text.bottom = new FormAttachment(btnNewButton, 0, SWT.BOTTOM);
00389             fd_text.top = new FormAttachment(btnNewButton, 0, SWT.TOP);
00390             fd_text.right = new FormAttachment(btnNewButton, -281);
00391             fd_text.left = new FormAttachment(0, 5);
00392             text.setLayoutData(fd text);
00393             fd_button.right = new FormAttachment(text, 0, SWT.RIGHT);
00394             fd_button.bottom = new FormAttachment(text, 0, SWT.BOTTOM);
00395             fd_button.left = new FormAttachment(text, -20);
00396             text.addKeyListener(new KeyAdapter(){
00397                 public void keyPressed(KeyEvent e){
00398                     if(e.keyCode == SWT.CR) {
00399                         searchEvent();
00400                     }
00401                 }
00402             });
```

```
00403         }
00404         catch(Exception e)
00405         {
00406             e.printStackTrace();
00407         }
00408 
00409     }
00410 }
```

test.java

```
00001 /*
00002  *
00003  */
00004 package dynamicAnalysis;
00005
00006 import org.eclipse.swt.SWT;
00007 import org.eclipse.swt.events.SelectionAdapter;
00008 import org.eclipse.swt.events.SelectionEvent;
00009 import org.eclipse.swt.layout.GridData;
00010 import org.eclipse.swt.layout.GridLayout;
00011 import org.eclipse.swt.widgets.Button;
00012 import org.eclipse.swt.widgets.Display;
00013 import org.eclipse.swt.widgets.Label;
00014 import org.eclipse.swt.widgets.Shell;
00015 import org.eclipse.swt.widgets.Text;
00016 import org.eclipse.wb.swt.SWTResourceManager;
00017
00021 public class test
00022 {
00023
00025     protected Shell shell;
00026
00028     private int processId;
00029
00031     private Text text;
00032
00034     private Text txtLength;
00035
00041     public static void main(String[] args)
00042     {
00043         try
00044         {
00045             test window = new test();
00046             window.open();
00047         } catch (Exception e)
00048         {
00049             e.printStackTrace();
00050         }
00051     }
00052
00056     public void open()
00057     {
00058         Display display = Display.getDefault();
00059         createContents();
00060         shell.open();
00061         shell.layout();
00062         while (!shell.isDisposed())
00063         {
00064             if (!display.readAndDispatch())
00065             {
00066                 display.sleep();
00067             }
00068         }
00069     }
00070
00074     protected void createContents()
00075     {
00076         shell = new Shell();
00077         shell.setSize(490, 342);
00078         shell.setLayout(new GridLayout(3, false));
00079
00080         Button btnUpdate = new Button(shell, SWT.NONE);
00081         btnUpdate.setLayoutData(new GridData(SWT.LEFT, SWT.CENTER, true, false,
1, 1));
00082         btnUpdate.addSelectionListener(new SelectionAdapter() {
00083             @Override
00084             public void widgetSelected(SelectionEvent e) {
00085
00086             }
00087         });
00088         btnUpdate.setText("Update");
00089
00090         txtLength = new Text(shell, SWT.BORDER);
```

```
00091         txtLength.setText("Length: ");
00092         txtLength.setLayoutData(new GridData(SWT.FILL, SWT.CENTER, false, false,
1, 1));
00093         new Label(shell, SWT.NONE);
00094
00095         text = new Text(shell, SWT.BORDER | SWT.WRAP | SWT.V_SCROLL);
00096         text.setLayoutData(new GridData(SWT.FILL, SWT.CENTER, true, false, 1,
1));
00097         text.setFont(SWTResourceManager.getFont("Calibri", 9, SWT.NORMAL));
00098         text.setLayoutData(new GridData(SWT.FILL, SWT.FILL, true, true, 1, 1));
00099         new Label(shell, SWT.NONE);
00100
00101         new Label(shell, SWT.NONE);
00102
00103     }
00104
00105 }
```

Version.java

```
00001 /*
00002  *
00003  */
00004 package dynamicAnalysis;
00005
00009 public enum Version {
00010
00012     x32(false),
00013
00015     x64(true);
00016
00018     private final boolean value;
00019
00025     private Version(boolean value) {
00026         this.value = value;
00027     }
00028
00034     public boolean getValue() {
00035         switch(Boolean.hashCode(value)) {
00036         case 1: return true;
00037         case 0: return false;
00038         }
00039         return value;
00040     }
00041
00042 }
```

VirtualMemory.java

```
00001 /*
00002  *
00003  */
00004 package dynamicAnalysis;
00005
00006 import java.io.File;
00007
00011 public class VirtualMemory
00012 {
00013
00015     private int processId;
00016
00017     static {System.load(getFile("memory.dll"));}
00018
00025     private native byte[] scanProcess(int processId);
00026
00032     public VirtualMemory(int processId)
00033     {
00034         setProcessId(processId);
00035     }
00036
00042     public int getProcessId()
00043     {
00044         return processId;
00045     }
00046
00052     public void setProcessId(int processId)
00053     {
00054         this.processId = processId;
00055     }
00056
00063     private static String getFile(String fileName)
00064     {
00065         /*File file = new File(".\\"+fileName);
00066         return file.getAbsolutePath();*/ //JAR release
00067         return
System.getProperty("user.dir")+"\\src\\dynamicAnalysis\\"+fileName;
00068     }
00069
00075     public byte[] readMemory()
00076     {
00077         VirtualMemory virtualMemory = new VirtualMemory(getProcessId());
00078         return virtualMemory.scanProcess(getProcessId());
00079     }
00080 }
00081
00082 //javac -h . VirtualMemory.java
00083 //g++ -I I:/jre/jre6/include -I I:/jre/jre6/include/win32 -shared -o memory.dll
VirtualMemory.cpp
```

Window.java

```
00001 /*
00002  *
00003  */
00004 package dynamicAnalysis;
00005
00006 import org.eclipse.swt.widgets.Display;
00007 import org.eclipse.swt.widgets.Event;
00008 import org.eclipse.swt.widgets.Shell;
00009 import org.eclipse.swt.layout.GridLayout;
00010 import org.eclipse.swt.widgets.Text;
00011 import org.pcap4j.core.PcapNativeException;
00012
00013 import java.awt.Dimension;
00014 import java.awt.Toolkit;
00015 import java.io.File;
00016
00017 import org.eclipse.swt.SWT;
00018 import org.eclipse.swt.widgets.Label;
00019 import org.eclipse.swt.widgets.Listener;
00020 import org.eclipse.swt.layout.GridData;
00021 import org.eclipse.swt.widgets.Button;
00022 import org.eclipse.swt.widgets.Table;
00023 import org.eclipse.swt.widgets.Composite;
00024 import org.eclipse.swt.widgets.Control;
00025 import org.eclipse.swt.layout.FormLayout;
00026 import org.eclipse.swt.layout.FormData;
00027 import org.eclipse.swt.layout.FormAttachment;
00028 import org.eclipse.swt.events.ModifyEvent;
00029 import org.eclipse.swt.events.ModifyListener;
00030 import org.eclipse.swt.events.SelectionAdapter;
00031 import org.eclipse.swt.events.SelectionEvent;
00032 import org.eclipse.swt.graphics.Color;
00033 import org.eclipse.swt.graphics.Rectangle;
00034 import org.eclipse.swt.custom.StyledText;
00035 import org.eclipse.swt.widgets.TableColumn;
00036 import org.eclipse.swt.widgets.TableItem;
00037 import org.eclipse.swt.widgets.Menu;
00038 import org.eclipse.swt.widgets.MenuItem;
00039 import org.eclipse.swt.widgets.MessageBox;
00040 import org.eclipse.swt.widgets.Group;
00041 import org.eclipse.swt.widgets.TabFolder;
00042 import org.eclipse.swt.widgets.TabItem;
00043 import org.eclipse.swt.custom.CTabFolder;
00044 import org.eclipse.swt.custom.CTabItem;
00045
00049 public class Window
00050 {
00051
00053     protected Shell shell;
00054
00056     private Label text;
00057
00059     private Table table;
00060
00062     private String filePath;
00063
00065     private SelectFile selectFile;
00066
00068     public static int processId;
00069
00071     private Display display;
00072
00074     private ProcessManager process;
00075
00081     public static void main(String[] args)
00082     {
00083         Window window = null;
00084         try
00085         {
00086             window = new Window();
00087             window.open();
00088         } catch (Exception e)
00089         {
```

```
00090             e.printStackTrace();
00091         }
00092         if(window.process!=null) window.process.destroyProcess();
00093     }
00094
00098     public void open()
00099     {
00100         display = Display.getDefault();
00101         createContents();
00102         shell.open();
00103         shell.layout();
00104         while (!shell.isDisposed())
00105         {
00106             if (!display.readAndDispatch())
00107             {
00108                 display.sleep();
00109             }
00110         }
00111
00112     }
00113
00119     private void errorAlert(String message)
00120     {
00121         MessageBox messageBox = new MessageBox(shell, SWT.ERROR);
00122         messageBox.setText("Error");
00123         messageBox.setMessage(message);
00124         messageBox.open();
00125     }
00126
00130     protected void createContents()
00131     {
00132         shell = new Shell();
00133         shell.setSize(491, 573);
00134         shell.setText("Dynamic Malware Analyzer");
00135         shell.setLayout(new FormLayout());
00136         shell.setMinimumSize(491, 573);
00137
00138         Dimension dim = Toolkit.getDefaultToolkit().getScreenSize();
00139         shell.setLocation((dim.width/2)-200,(dim.height/2)-250);
00140
00141         Button btnProcess = new Button(shell, SWT.CHECK);
00142
00143         text = new Label(shell, SWT.BORDER);
00144
00145         Button btnLaunch = new Button(shell, SWT.NONE);
00146
00147         TableItem tableItems[] = new TableItem[4];
00148
00149         FormData fd_text = new FormData();
00150         fd_text.top = new FormAttachment(0, 11);
00151         text.setLayoutData(fd_text);
00152
00153
00154         Button btnInstructions = new Button(shell, SWT.NONE);
00155         btnInstructions.setEnabled(false);
00156
00157         FormData fd_btnInstructions = new FormData();
00158         btnInstructions.setLayoutData(fd_btnInstructions);
00159         btnInstructions.setText("x86 Instructions");
00160
00161         Button btnMemory = new Button(shell, SWT.NONE);
00162         fd_btnInstructions.bottom = new FormAttachment(btnMemory, -6);
00163         btnMemory.setEnabled(false);
00164
00165         btnMemory.setText("Virtual Memory");
00166         FormData fd_btnMemory = new FormData();
00167         fd btnInstructions.bottom = new FormAttachment(btnMemory, -6);
00168
00169         btnMemory.setLayoutData(fd_btnMemory);
00170
00171         Button btnFiles = new Button(shell, SWT.NONE);
00172         fd btnMemory.bottom = new FormAttachment(btnFiles, -6);
00173         btnFiles.setEnabled(false);
00174         btnFiles.setText("Files");
00175         FormData fd_btnFiles = new FormData();
00176         fd_btnFiles.bottom = new FormAttachment(0, 178);
00177         fd_btnFiles.top = new FormAttachment(0, 152);
```

```
00178
00179         btnFiles.setLayoutData(fd_btnFiles);
00180         table = new Table(shell, SWT.BORDER | SWT.FULL_SELECTION |
SWT.NO_SCROLL);
00181         fd_btnFiles.right = new FormAttachment(table, 101, SWT.RIGHT);
00182         fd_btnMemory.right = new FormAttachment(table, 101, SWT.RIGHT);
00183         fd_btnInstructions.right = new FormAttachment(table, 101, SWT.RIGHT);
00184         fd_btnInstructions.left = new FormAttachment(table, 6);
00185         fd_btnMemory.left = new FormAttachment(table, 6);
00186         fd_btnFiles.left = new FormAttachment(table, 6);
00187         fd_btnMemory.left = new FormAttachment(table, 6);
00188         FormData fd_table = new FormData();
00189         fd_table.bottom = new FormAttachment(0, 177);
00190         fd_table.top = new FormAttachment(0, 90);
00191         fd_table.right = new FormAttachment(0, 331);
00192         table.setLayoutData(fd_table);
00193         table.setHeaderVisible(false);
00194         table.setLinesVisible(true);
00195
00196         Button btnSelect = new Button(shell, SWT.NONE);
00197         btnSelect.addSelectionListener(new SelectionAdapter() {
00198             @Override
00199             public void widgetSelected(SelectionEvent e) {
00200                 System.out.println("selection: "+btnProcess.getSelection());
00201                 selectFile = new SelectFile(shell.getLocation().x,
shell.getLocation().y, btnProcess.getSelection());
00202                 filePath = selectFile.getText();
00203                 if(!selectFile.isPidMode())
00204                 {
00205                     try
00206                     {
00207                         text.setText(filePath);
00208                         btnLaunch.setEnabled(true);
00209                     }
00210                     catch(IllegalArgumentException e1)
00211                     {
00212                         btnLaunch.setEnabled(false);
00213                     }
00214                 }
00215                 else
00216                 {
00217                     try
00218                     {
00219                         text.setText(Integer.toString(selectFile.getPid()));
00220                         btnLaunch.setEnabled(true);
00221                     }
00222                     catch(IllegalArgumentException e1)
00223                     {
00224                         btnLaunch.setEnabled(false);
00225                     }
00226                 }
00227             }
00228         });
00229         btnSelect.setText("Select File");
00230         FormData fd_btnSelect = new FormData();
00231         fd_btnSelect.right = new FormAttachment(btnInstructions, 0, SWT.RIGHT);
00232         fd_btnSelect.bottom = new FormAttachment(btnInstructions, -58);
00233         fd_btnSelect.left = new FormAttachment(text, 5, SWT.RIGHT);
00234         btnSelect.setLayoutData(fd_btnSelect);
00235
00236         Label lblFilePath = new Label(shell, SWT.NONE);
00237         fd_text.right = new FormAttachment(table, 0, SWT.RIGHT);
00238         fd_text.left = new FormAttachment(lblFilePath, 13);
00239         lblFilePath.setAlignment(SWT.RIGHT);
00240         fd_table.left = new FormAttachment(0, 26);
00241         FormData fd_lblFilePath = new FormData();
00242         fd_lblFilePath.right = new FormAttachment(0, 79);
00243         fd_lblFilePath.left = new FormAttachment(0, 26);
00244         fd_lblFilePath.top = new FormAttachment(0, 11);
00245         lblFilePath.setLayoutData(fd_lblFilePath);
00246         lblFilePath.setText("File Path");
00247
00248         TableColumn labels = new TableColumn(table, SWT.NONE);
00249         labels.setWidth(100);
00250
00251         TableColumn values = new TableColumn(table, SWT.CENTER | SWT.V_SCROLL);
00252         values.setWidth(200);
```

```
00253
00254
00255         for(int index = 0;index<tableItems.length;index++)
00256         {
00257             tableItems[index] = new TableItem(table, SWT.NONE);
00258         }
00259
00260         tableItems[0].setText(0, "Directory");
00261         tableItems[1].setText(0, "Version");
00262         tableItems[2].setText(0, "Name");
00263         tableItems[3].setText(0, "PID");
00264         Menu menu = new Menu(shell, SWT.BAR);
00265         shell.setMenuBar(menu);
00266
00267         MenuItem mntmFile = new MenuItem(menu, SWT.CASCADE);
00268         mntmFile.setText("File");
00269
00270         Menu menu_1 = new Menu(mntmFile);
00271         mntmFile.setMenu(menu_1);
00272
00273         MenuItem mntmOpen = new MenuItem(menu_1, SWT.NONE);
00274         mntmOpen.setText("Open");
00275
00276         mntmOpen.addListener(SWT.Selection, new Listener() {
00277             public void handleEvent(Event e) {
00278                 selectFile = new SelectFile(shell.getLocation().x,
shell.getLocation().y, btnProcess.getSelection());
00279                 filePath = selectFile.getText();
00280                 try
00281                 {
00282                     text.setText(filePath);
00283                     btnLaunch.setEnabled(true);
00284                 }
00285                 catch(IllegalArgumentException e1)
00286                 {
00287                     btnLaunch.setEnabled(false);
00288                 }
00289             }
00290         });
00291
00292         MenuItem mntmProcess = new MenuItem(menu, SWT.CASCADE);
00293         mntmProcess.setText("Process");
00294
00295         Menu menu_2 = new Menu(mntmProcess);
00296         mntmProcess.setMenu(menu_2);
00297
00298         MenuItem mntmDestroyProcess = new MenuItem(menu_2, SWT.NONE);
00299         mntmDestroyProcess.setText("Destroy Process");
00300
00301         MenuItem mntmNetwork = new MenuItem(menu, SWT.NONE);
00302         mntmNetwork.setText("Network");
00303
00304         mntmDestroyProcess.addListener(SWT.Selection, new Listener() {
00305             public void handleEvent(Event e) {
00306                 if(process!=null) process.destroyProcess();
00307             }
00308         });
00309         btnProcess.addSelectionListener(new SelectionAdapter() {
00310             @Override
00311             public void widgetSelected(SelectionEvent e) {
00312                 text.setText("");
00313                 btnLaunch.setEnabled(false);
00314                 if (btnProcess.getSelection())
00315                 {
00316                     lblFilePath.setText("Process ID");
00317                     btnSelect.setText("Select Process");
00318                 }
00319                 else
00320                 {
00321                     lblFilePath.setText("File Path");
00322                     btnSelect.setText("Select File");
00323                 }
00324             }
00325         });
00326         FormData fd_btnProcess = new FormData();
00327         fd_btnProcess.right = new FormAttachment(btnLaunch, 130);
00328         fd_btnProcess.left = new FormAttachment(btnLaunch, 10);
```

```
00329         fd_btnProcess.top = new FormAttachment(btnSelect, 6);
00330         btnProcess.setLayoutData(fd_btnProcess);
00331         btnProcess.setText("Process");
00332         btnLaunch.addSelectionListener(new SelectionAdapter() {
00333             @Override
00334             public void widgetSelected(SelectionEvent e) {
00335                 tableItems[0].setText(1, "");
00336                 tableItems[1].setText(1, "");
00337                 tableItems[2].setText(1, "");
00338                 tableItems[3].setText(1, "");
00339                 btnMemory.setEnabled(true);
00340                 btnFiles.setEnabled(true);
00341                 tableItems[0].setGrayed(btnProcess.getSelection());
00342                 tableItems[1].setGrayed(btnProcess.getSelection());
00343                 btnInstructions.setEnabled(!btnProcess.getSelection());
00344                 if(!btnProcess.getSelection())
00345                 {
00346                     String filePath = text.getText();
00347                     try
00348                     {
00349                         process = new ProcessManager(new File(filePath));
00350                         CodeExtract codeExtract = new CodeExtract(new
File(filePath));
00351                         tableItems[0].setText(1, filePath);
00352                         if(codeExtract.getPeFile().getVersion() == Version.x32)
00353                         {
00354                             tableItems[1].setText(1, "32-bit");
00355                         }
00356                         else
00357                         {
00358                             tableItems[1].setText(1, "64-bit");
00359                         }
00360                         tableItems[2].setText(1, process.getName());
00361                         tableItems[3].setText(1, process.getPidAsString());
00362                         System.out.println("version:
"+codeExtract.getPeFile().getVersion());
00363                     }
00364                     catch(NullPointerException e1)
00365                     {
00366                         e1.printStackTrace();
00367                         errorAlert("Admin privileges are required to run the
process.");
00368                     }
00369                     catch(ArrayIndexOutOfBoundsException e1)
00370                     {
00371                         e1.printStackTrace();
00372                         errorAlert("Could not open the selected process.");
00373                     }
00374                 }
00375                 else
00376                 {
00377                     ProcessManager process = new
ProcessManager(Integer.parseInt(text.getText()));
00378                     tableItems[2].setText(1, process.getName());
00379                     tableItems[3].setText(1,
Integer.toString(selectFile.getPid()));
00380                 }
00381             }
00382         });
00383         FormData fd_btnLaunch = new FormData();
00384         fd btnLaunch.top = new FormAttachment(btnProcess, -4, SWT.TOP);
00385         fd_btnLaunch.right = new FormAttachment(lblFilePath, 66, SWT.RIGHT);
00386         fd_btnLaunch.left = new FormAttachment(text, 0, SWT.LEFT);
00387         btnLaunch.setLayoutData(fd_btnLaunch);
00388         btnLaunch.setText("Launch");
00389         btnLaunch.setEnabled(false);
00390
00391         CTabFolder tabFolder = new CTabFolder(shell, SWT.BORDER);
00392         FormData fd_tabFolder = new FormData();
00393         fd_tabFolder.right = new FormAttachment(100, -45);
00394         fd_tabFolder.top = new FormAttachment(table, 30);
00395         fd tabFolder.bottom = new FormAttachment(100, -23);
00396         fd_tabFolder.left = new FormAttachment(table, 0, SWT.LEFT);
00397         tabFolder.setLayoutData(fd_tabFolder);
00398
tabFolder.setSelectionBackground(Display.getCurrent().getSystemColor(SWT.COLOR_TITLE_I
NACTIVE_BACKGROUND_GRADIENT));
```

```
00399
00400
00401         btnMemory.addSelectionListener(new SelectionAdapter() {
00402             @Override
00403             public void widgetSelected(SelectionEvent e) {
00404                 CTabItem tbtmMemory = new CTabItem(tabFolder, SWT.CLOSE);
00405                 tbtmMemory.setText("Memory");
00406                 processId = Integer.parseInt(tableItems[3].getText(1));
00407                 Color red = display.getSystemColor(SWT.COLOR_RED);
00408                 MemoryComposite memoryComposite = new MemoryComposite(tabFolder,
SWT.NULL, red);
00409                 memoryComposite.layout();
00410                 memoryComposite.setFocus();
00411                 System.out.println(memoryComposite.getProcessId());
00412                 tbtmMemory.setControl(memoryComposite);
00413                 tabFolder.setSelection(tbtmMemory);
00414             }
00415         });
00416
00417         btnFiles.addSelectionListener(new SelectionAdapter() {
00418             @Override
00419             public void widgetSelected(SelectionEvent e) {
00420                 CTabItem tbtmAdvanced = new CTabItem(tabFolder, SWT.CLOSE);
00421                 tbtmAdvanced.setText("Advanced");
00422                 processId = Integer.parseInt(tableItems[3].getText(1));
00423                 FilesComposite filesComposite = new FilesComposite(tabFolder,
SWT.NULL);
00424                 filesComposite.layout();
00425                 tbtmAdvanced.setControl(filesComposite);
00426                 tabFolder.setSelection(tbtmAdvanced);
00427             }
00428         });
00429
00430         btnInstructions.addSelectionListener(new SelectionAdapter() {
00431             @Override
00432             public void widgetSelected(SelectionEvent e) {
00433                 CTabItem tbtmInstructions = new CTabItem(tabFolder, SWT.CLOSE);
00434                 tbtmInstructions.setText("Instructions");
00435                 InstructionsComposite instructionsComposite = new
InstructionsComposite(tabFolder, SWT.NULL, new File(filePath));
00436                 instructionsComposite.layout();
00437                 tbtmInstructions.setControl(instructionsComposite);
00438                 tabFolder.setSelection(tbtmInstructions);
00439             }
00440         });
00441
00442         mntmNetwork.addListener(SWT.Selection, new Listener() {
00443             public void handleEvent(Event e) {
00444                 CTabItem tbtmNetwork = new CTabItem(tabFolder, SWT.CLOSE);
00445                 try
00446                 {
00447                     processId = Integer.parseInt(tableItems[3].getText(1));
00448                 } catch(NumberFormatException e1)
00449                 {
00450                     processId = 0;
00451                 }
00452
00453                 tbtmNetwork.setText("Network");
00454                 try
00455                 {
00456                     NetworkComposite networkComposite = new
NetworkComposite(tabFolder, SWT.NULL, processId);
00457                     tbtmNetwork.setControl(networkComposite);
00458                     tabFolder.setSelection(tbtmNetwork);
00459                 } catch (PcapNativeException e1)
00460                 {
00461                     // TODO Auto-generated catch block
00462                     e1.printStackTrace();
00463                 }
00464             }
00465         });
00466
00467         shell.addListener (SWT.Resize,  new Listener () {
00468             public void handleEvent (Event e) {
00469                 Control[] composites = tabFolder.getChildren();
00470                 for(int index = 0; index < composites.length; index++)
00471                 {
```

```
00472                     composites[index].requestLayout();
00473                     System.out.println("modified "+index);
00474                 }
00475             }
00476           });
00477     }
00478 }
```

SWTResourceManager.java

```
00001 /*******************************************************************************
00002  * Copyright (c) 2011 Google, Inc.
00003  * All rights reserved. This program and the accompanying materials
00004  * are made available under the terms of the Eclipse Public License v1.0
00005  * which accompanies this distribution, and is available at
00006  * http://www.eclipse.org/legal/epl-v10.html
00007  *
00008  * Contributors:
00009  *    Google, Inc. - initial API and implementation
00010 
*******************************************************************************/
00011 package org.eclipse.wb.swt;
00012 
00013 import java.io.FileInputStream;
00014 import java.io.IOException;
00015 import java.io.InputStream;
00016 import java.util.HashMap;
00017 import java.util.Map;
00018 
00019 import org.eclipse.swt.SWT;
00020 import org.eclipse.swt.graphics.Color;
00021 import org.eclipse.swt.graphics.Cursor;
00022 import org.eclipse.swt.graphics.Font;
00023 import org.eclipse.swt.graphics.FontData;
00024 import org.eclipse.swt.graphics.GC;
00025 import org.eclipse.swt.graphics.Image;
00026 import org.eclipse.swt.graphics.ImageData;
00027 import org.eclipse.swt.graphics.RGB;
00028 import org.eclipse.swt.graphics.Rectangle;
00029 import org.eclipse.swt.widgets.Display;
00030 
00043 public class SWTResourceManager {
00045     //
00046     // Color
00047     //
00049     private static Map<RGB, Color> m_colorMap = new HashMap<RGB, Color>();
00057     public static Color getColor(int systemColorID) {
00058         Display display = Display.getCurrent();
00059         return display.getSystemColor(systemColorID);
00060     }
00072     public static Color getColor(int r, int g, int b) {
00073         return getColor(new RGB(r, g, b));
00074     }
00082     public static Color getColor(RGB rgb) {
00083         Color color = m_colorMap.get(rgb);
00084         if (color == null) {
00085             Display display = Display.getCurrent();
00086             color = new Color(display, rgb);
00087             m_colorMap.put(rgb, color);
00088         }
00089         return color;
00090     }
00094     public static void disposeColors() {
00095         for (Color color : m_colorMap.values()) {
00096             color.dispose();
00097         }
00098         m_colorMap.clear();
00099     }
00101     //
00102     // Image
00103     //
00105 
00108     private static Map<String, Image> m_imageMap = new HashMap<String, Image>();
00116     protected static Image getImage(InputStream stream) throws IOException {
00117         try {
00118             Display display = Display.getCurrent();
00119             ImageData data = new ImageData(stream);
00120             if (data.transparentPixel > 0) {
00121                 return new Image(display, data, data.getTransparencyMask());
00122             }
00123             return new Image(display, data);
00124         } finally {
00125             stream.close();
```

```
00126         }
00127     }
00135     public static Image getImage(String path) {
00136         Image image = m_imageMap.get(path);
00137         if (image == null) {
00138             try {
00139                 image = getImage(new FileInputStream(path));
00140                 m_imageMap.put(path, image);
00141             } catch (Exception e) {
00142                 image = getMissingImage();
00143                 m_imageMap.put(path, image);
00144             }
00145         }
00146         return image;
00147     }
00157     public static Image getImage(Class<?> clazz, String path) {
00158         String key = clazz.getName() + '|' + path;
00159         Image image = m_imageMap.get(key);
00160         if (image == null) {
00161             try {
00162                 image = getImage(clazz.getResourceAsStream(path));
00163                 m_imageMap.put(key, image);
00164             } catch (Exception e) {
00165                 image = getMissingImage();
00166                 m_imageMap.put(key, image);
00167             }
00168         }
00169         return image;
00170     }
00171     private static final int MISSING_IMAGE_SIZE = 10;
00175     private static Image getMissingImage() {
00176         Image image = new Image(Display.getCurrent(), MISSING_IMAGE_SIZE,
MISSING_IMAGE_SIZE);
00177         //
00178         GC gc = new GC(image);
00179         gc.setBackground(getColor(SWT.COLOR_RED));
00180         gc.fillRectangle(0, 0, MISSING_IMAGE_SIZE, MISSING_IMAGE_SIZE);
00181         gc.dispose();
00182         //
00183         return image;
00184     }
00188     public static final int TOP_LEFT = 1;
00192     public static final int TOP_RIGHT = 2;
00196     public static final int BOTTOM_LEFT = 3;
00200     public static final int BOTTOM_RIGHT = 4;
00204     protected static final int LAST_CORNER_KEY = 5;
00208     @SuppressWarnings("unchecked")
00209     private static Map<Image, Map<Image, Image>>[] m_decoratedImageMap = new
Map[LAST_CORNER_KEY];
00219     public static Image decorateImage(Image baseImage, Image decorator) {
00220         return decorateImage(baseImage, decorator, BOTTOM_RIGHT);
00221     }
00233     public static Image decorateImage(final Image baseImage, final Image
decorator, final int corner) {
00234         if (corner <= 0 || corner >= LAST_CORNER_KEY) {
00235             throw new IllegalArgumentException("Wrong decorate corner");
00236         }
00237         Map<Image, Map<Image, Image>> cornerDecoratedImageMap =
m_decoratedImageMap[corner];
00238         if (cornerDecoratedImageMap == null) {
00239             cornerDecoratedImageMap = new HashMap<Image, Map<Image, Image>>();
00240             m_decoratedImageMap[corner] = cornerDecoratedImageMap;
00241         }
00242         Map<Image, Image> decoratedMap = cornerDecoratedImageMap.get(baseImage);
00243         if (decoratedMap == null) {
00244             decoratedMap = new HashMap<Image, Image>();
00245             cornerDecoratedImageMap.put(baseImage, decoratedMap);
00246         }
00247         //
00248         Image result = decoratedMap.get(decorator);
00249         if (result == null) {
00250             Rectangle bib = baseImage.getBounds();
00251             Rectangle dib = decorator.getBounds();
00252             //
00253             result = new Image(Display.getCurrent(), bib.width, bib.height);
00254             //
00255             GC gc = new GC(result);
```

```
00256             gc.drawImage(baseImage, 0, 0);
00257             if (corner == TOP_LEFT) {
00258                 gc.drawImage(decorator, 0, 0);
00259             } else if (corner == TOP_RIGHT) {
00260                 gc.drawImage(decorator, bib.width - dib.width, 0);
00261             } else if (corner == BOTTOM_LEFT) {
00262                 gc.drawImage(decorator, 0, bib.height - dib.height);
00263             } else if (corner == BOTTOM_RIGHT) {
00264                 gc.drawImage(decorator, bib.width - dib.width, bib.height -
dib.height);
00265             }
00266             gc.dispose();
00267             //
00268             decoratedMap.put(decorator, result);
00269         }
00270         return result;
00271     }
00275     public static void disposeImages() {
00276         // dispose loaded images
00277         {
00278             for (Image image : m_imageMap.values()) {
00279                 image.dispose();
00280             }
00281             m_imageMap.clear();
00282         }
00283         // dispose decorated images
00284         for (int i = 0; i < m_decoratedImageMap.length; i++) {
00285             Map<Image, Map<Image, Image>> cornerDecoratedImageMap =
m_decoratedImageMap[i];
00286             if (cornerDecoratedImageMap != null) {
00287                 for (Map<Image, Image> decoratedMap :
cornerDecoratedImageMap.values()) {
00288                     for (Image image : decoratedMap.values()) {
00289                         image.dispose();
00290                     }
00291                     decoratedMap.clear();
00292                 }
00293                 cornerDecoratedImageMap.clear();
00294             }
00295         }
00296     }
00298     //
00299     // Font
00300     //
00302
00305     private static Map<String, Font> m_fontMap = new HashMap<String, Font>();
00309     private static Map<Font, Font> m_fontToBoldFontMap = new HashMap<Font,
Font>();
00321     public static Font getFont(String name, int height, int style) {
00322         return getFont(name, height, style, false, false);
00323     }
00340     public static Font getFont(String name, int size, int style, boolean
strikeout, boolean underline) {
00341         String fontName = name + '|' + size + '|' + style + '|' + strikeout +
'|' + underline;
00342         Font font = m_fontMap.get(fontName);
00343         if (font == null) {
00344             FontData fontData = new FontData(name, size, style);
00345             if (strikeout || underline) {
00346                 try {
00347                     Class<?> logFontClass =
Class.forName("org.eclipse.swt.internal.win32.LOGFONT"); //$NON-NLS-1$
00348                     Object logFont =
FontData.class.getField("data").get(fontData); //$NON-NLS-1$
00349                     if (logFont != null && logFontClass != null) {
00350                         if (strikeout) {
00351                             logFontClass.getField("lfStrikeOut").set(logFont,
Byte.valueOf((byte) 1)); //$NON-NLS-1$
00352                         }
00353                         if (underline) {
00354                             logFontClass.getField("lfUnderline").set(logFont,
Byte.valueOf((byte) 1)); //$NON-NLS-1$
00355                         }
00356                     }
00357                 } catch (Throwable e) {
00358                     System.err.println("Unable to set underline or strikeout" +
" (probably on a non-Windows platform). " + e); //$NON-NLS-1$ //$NON-NLS-2$
```

```
00359                 }
00360             }
00361             font = new Font(Display.getCurrent(), fontData);
00362             m_fontMap.put(fontName, font);
00363         }
00364         return font;
00365     }
00373     public static Font getBoldFont(Font baseFont) {
00374         Font font = m_fontToBoldFontMap.get(baseFont);
00375         if (font == null) {
00376             FontData fontDatas[] = baseFont.getFontData();
00377             FontData data = fontDatas[0];
00378             font = new Font(Display.getCurrent(), data.getName(),
data.getHeight(), SWT.BOLD);
00379             m_fontToBoldFontMap.put(baseFont, font);
00380         }
00381         return font;
00382     }
00386     public static void disposeFonts() {
00387         // clear fonts
00388         for (Font font : m_fontMap.values()) {
00389             font.dispose();
00390         }
00391         m_fontMap.clear();
00392         // clear bold fonts
00393         for (Font font : m_fontToBoldFontMap.values()) {
00394             font.dispose();
00395         }
00396         m_fontToBoldFontMap.clear();
00397     }
00399     //
00400     // Cursor
00401     //
00403
00406     private static Map<Integer, Cursor> m_idToCursorMap = new HashMap<Integer,
Cursor>();
00414     public static Cursor getCursor(int id) {
00415         Integer key = Integer.valueOf(id);
00416         Cursor cursor = m_idToCursorMap.get(key);
00417         if (cursor == null) {
00418             cursor = new Cursor(Display.getDefault(), id);
00419             m_idToCursorMap.put(key, cursor);
00420         }
00421         return cursor;
00422     }
00426     public static void disposeCursors() {
00427         for (Cursor cursor : m_idToCursorMap.values()) {
00428             cursor.dispose();
00429         }
00430         m_idToCursorMap.clear();
00431     }
00433     //
00434     // General
00435     //
00437
00441     public static void dispose() {
00442         disposeColors();
00443         disposeImages();
00444         disposeFonts();
00445         disposeCursors();
00446     }
00447
```