Taint-enabled Reverse Engineering Environment (TREE)

Abstract:

This document describes how to install, remove, and use TREE to generate, analyze, and visualize trace across various operating systems and architectures. This document may update frequently with new developments, check often for latest update at code.google.com/p/tree-cbass/.

For users who intend to learn the designs and internals behind the tool, check the "TREE Design Document".

TREE Development Team:

- Lixin(Nathan) Li [lil@battelle]
- Xing Li [lix@battelle.org]
- Loc Nguyen [nguyenl@battelle.org]

What is TREE?

TREE stands for Taint-enabled Reverse Engineering Environment.

Components

- /Tree Analyzer - Main component for the analyze/visualizer widgets
- /Tree Tracer - Main component for the tracer widget
- /dispatcher/* - Core component for TREE
- /documentation/*
- /TestSuites/*

Getting Started

Requirements

Host Platform:

Windows XP SP3 - Tested and Verified
Windows 7 64bit - Tested and Verified

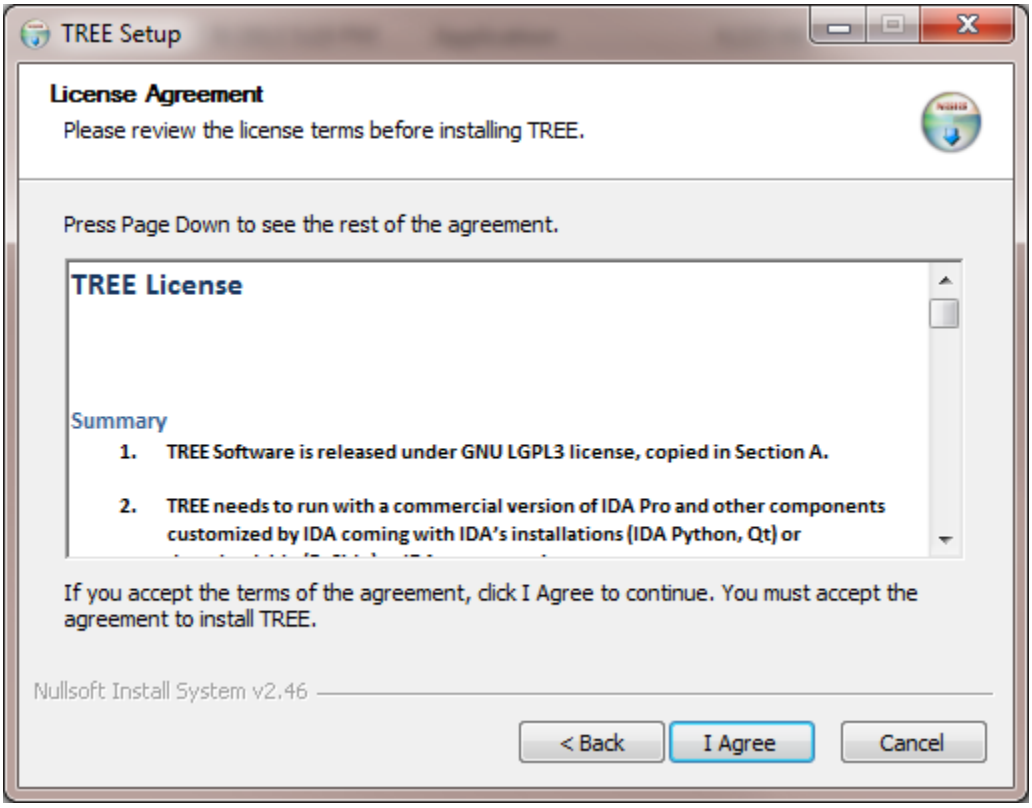
Third Party Tools:

IDA Pro 6.4.130306 or newer
Python 2.7

NetworkX - http://networkx.github.io/
PySide for IDA Pro - *https://www.hex-rays.com/products/ida/support/download.shtml*

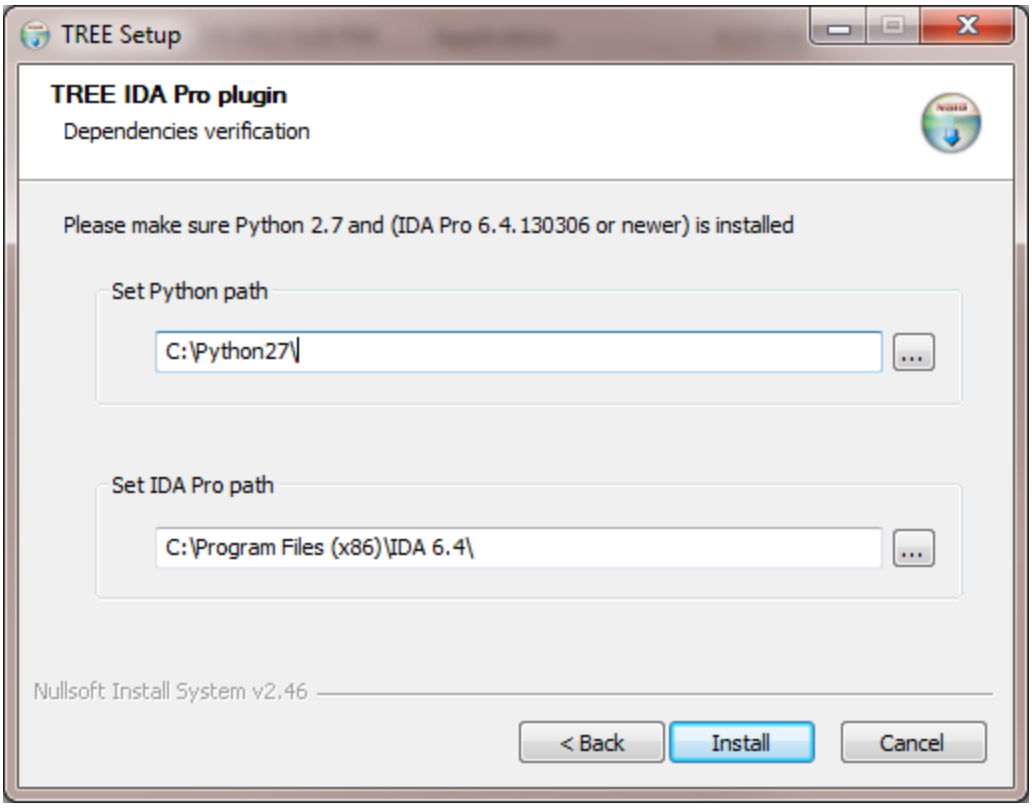
Installation

>Locate and run InstallTREE.exe:

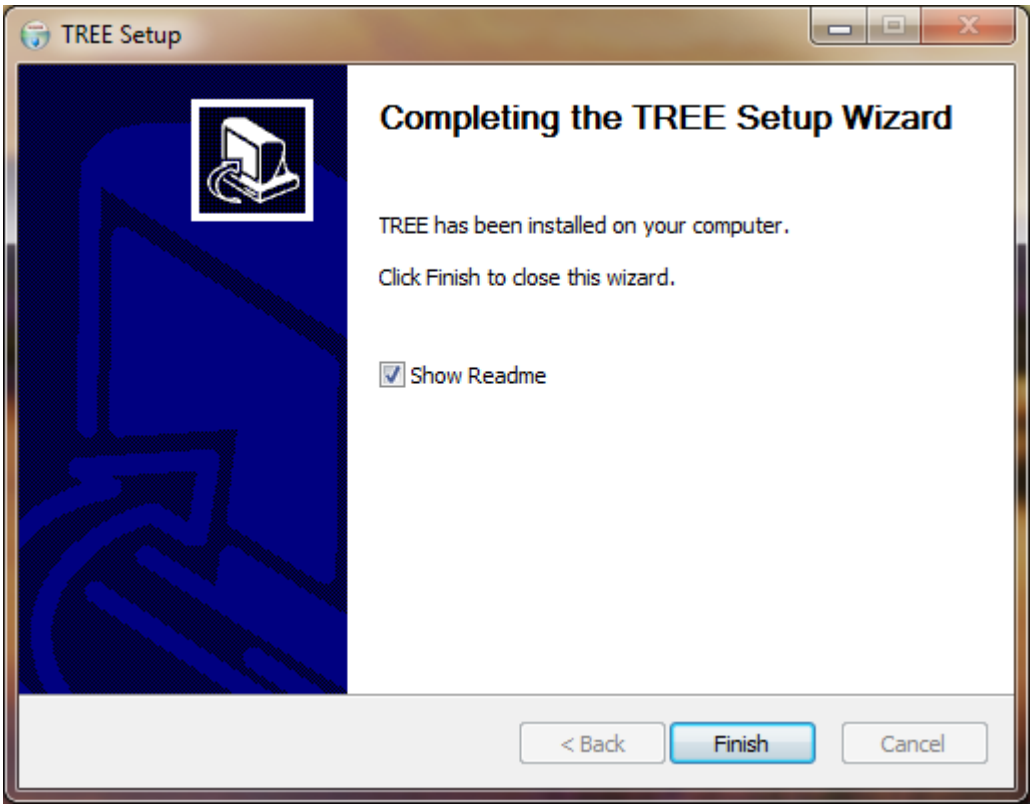


Accept TREE license, and move on.

>Verified the IDA Pro and Python installed path or browse to the correct path

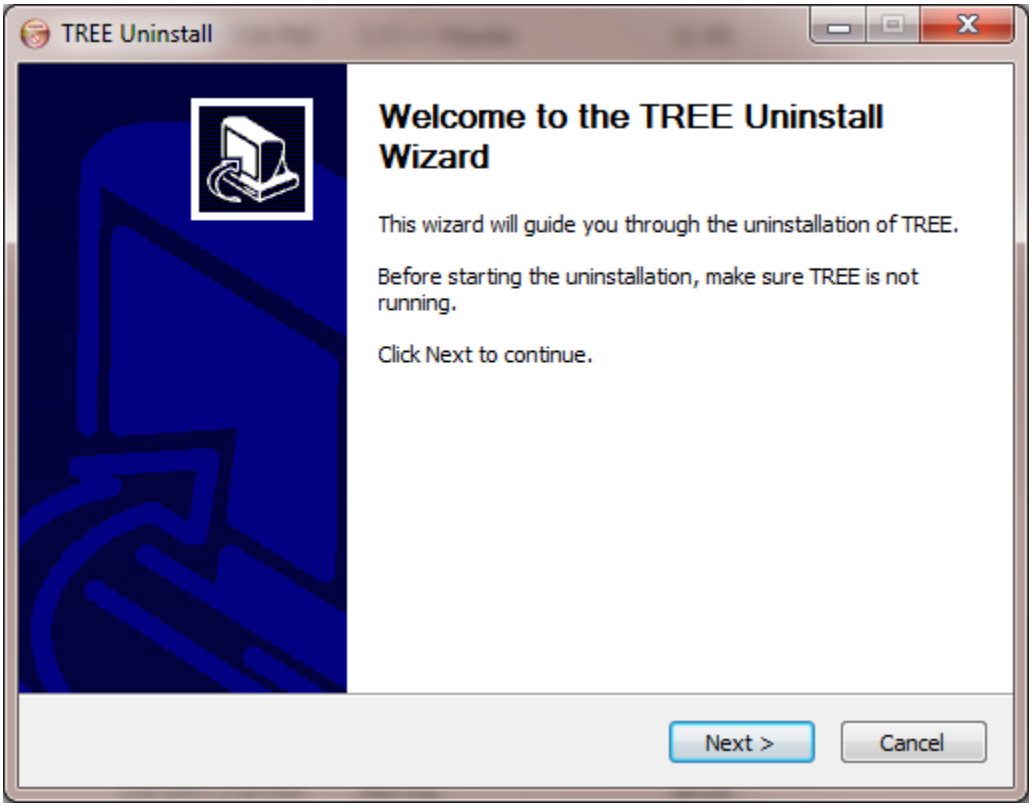


>Close the installer. The TREE plugin should be installed at this point.



Removal

>Locate and run uninstallTREE.exe *(This file is usually located in your IDA Pro plugins folder)*



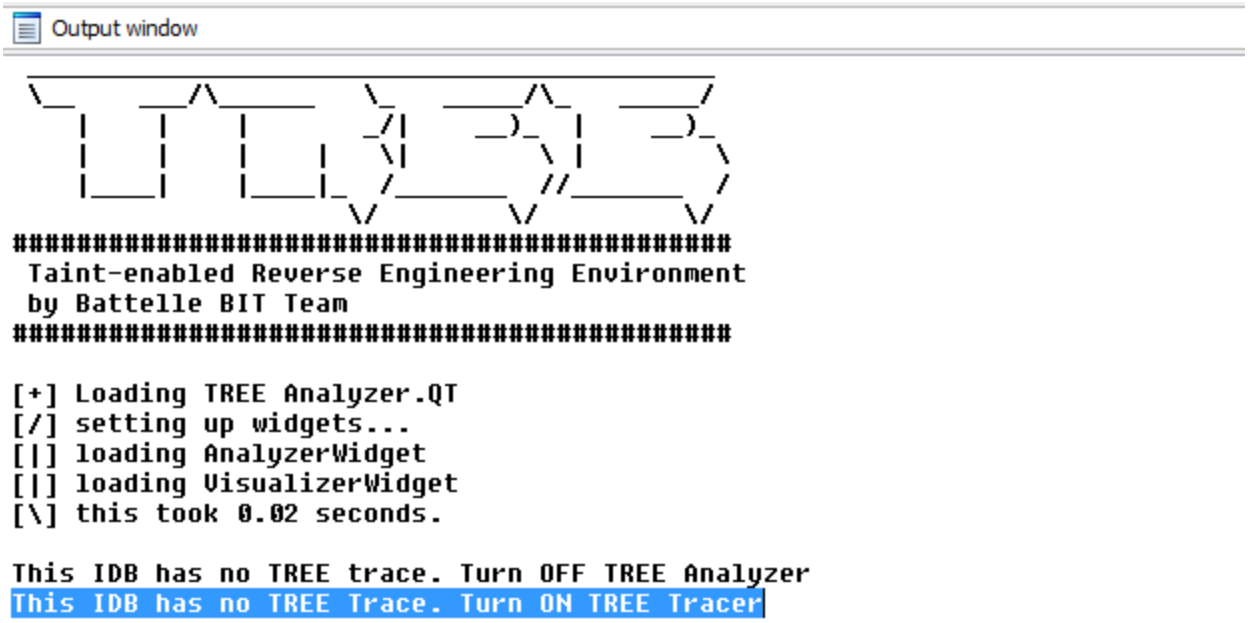
Usage

Initializing TREE

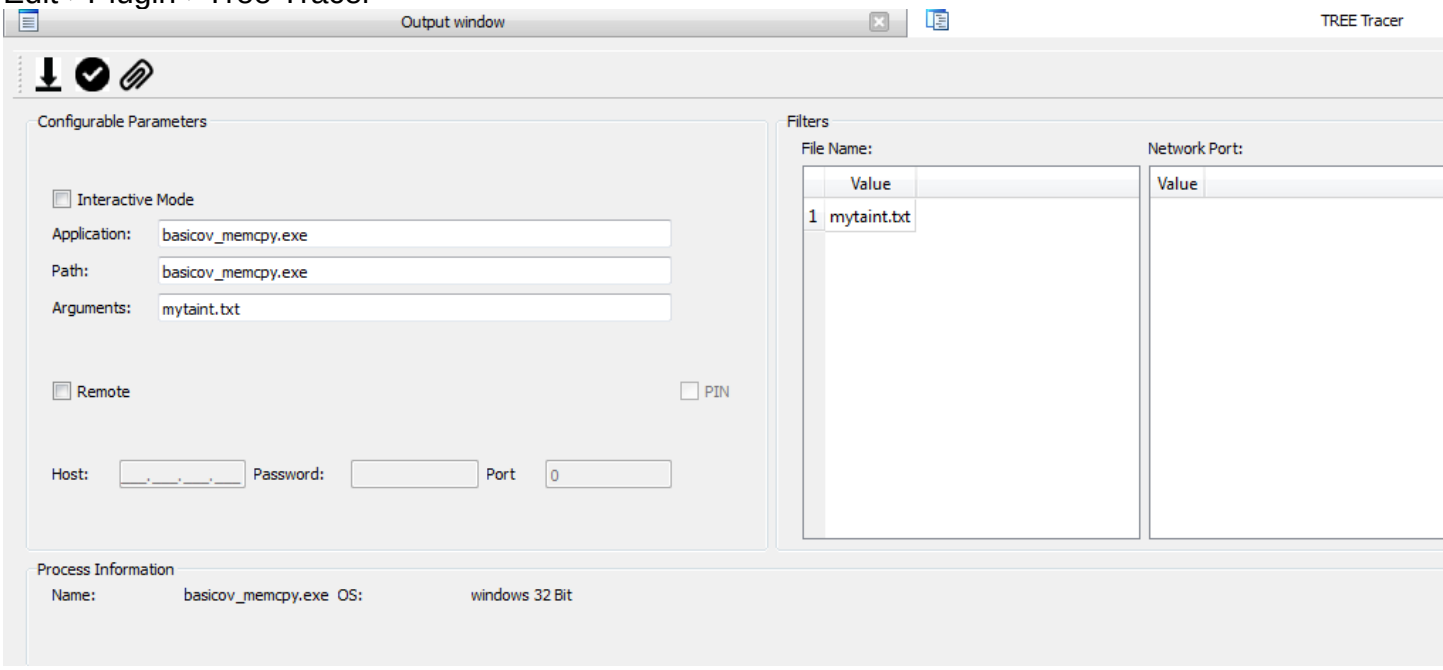
Tracer

- Launch IDA Pro to disassemble a new file
- ***On Windows 7***
  - Run IDA Pro as Admin for maximum privileges.

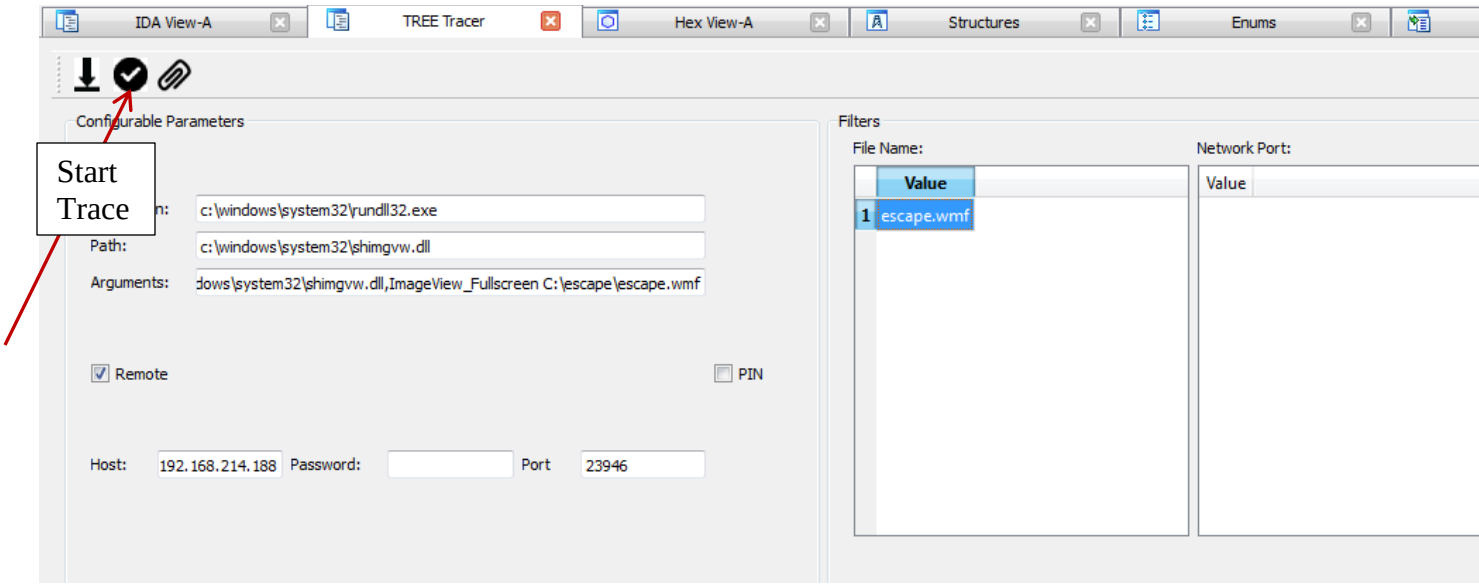
If this this IDB file has not run TREE trace yet, you will see the message in the output window as follows:



Edit->Plugin->Tree Tracer

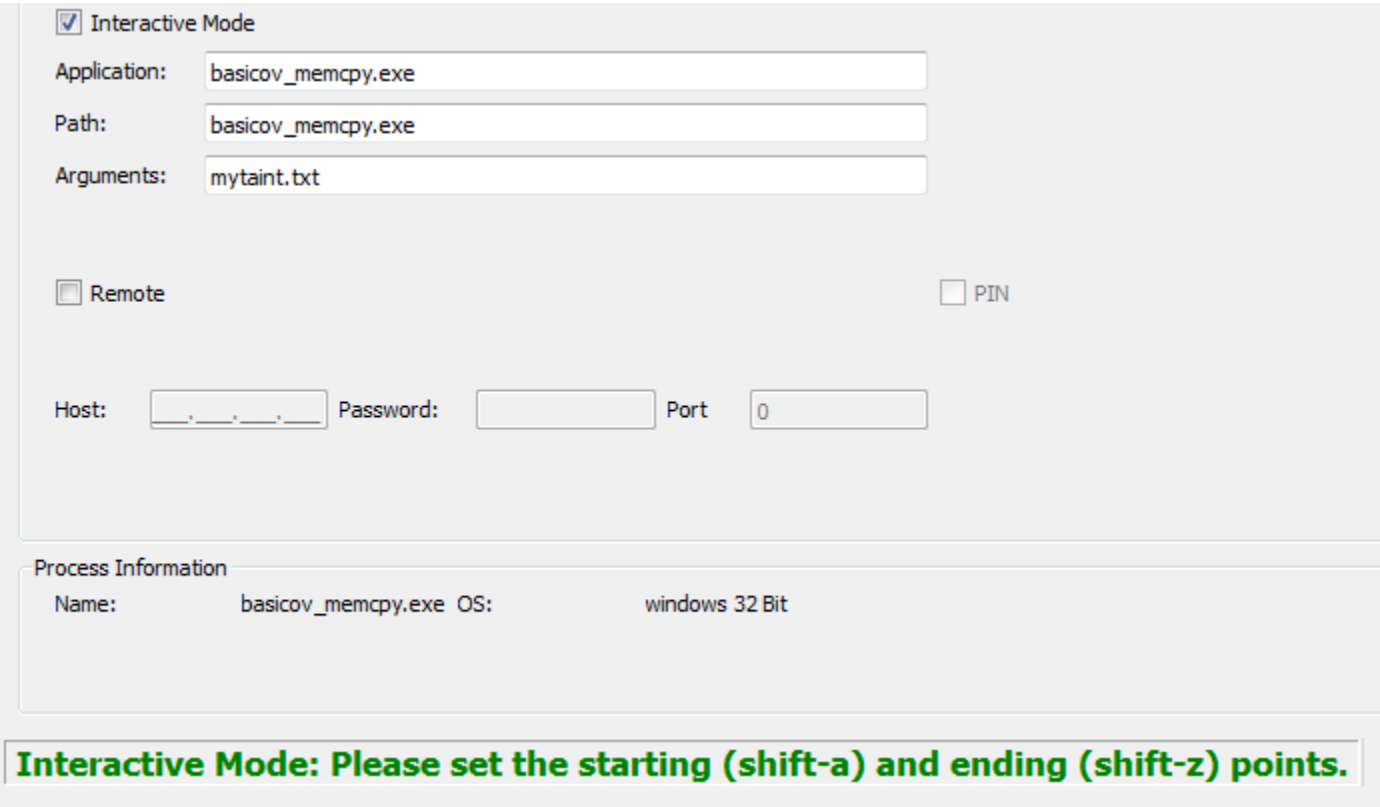


Typically, the *Application* and the *Path* will be the same for the target, like the example above. However, *Application* and *Path* will differ when the binary under IDA is a DLL. Arguments to the target should be provided as *Arguments*.



- For remote debugging, check the remote checkbox and input the network information to reach the machine.
- Apply filters for file names and network ports through the right-click menu (shown above) on each respective table, by right clicking on the entry.

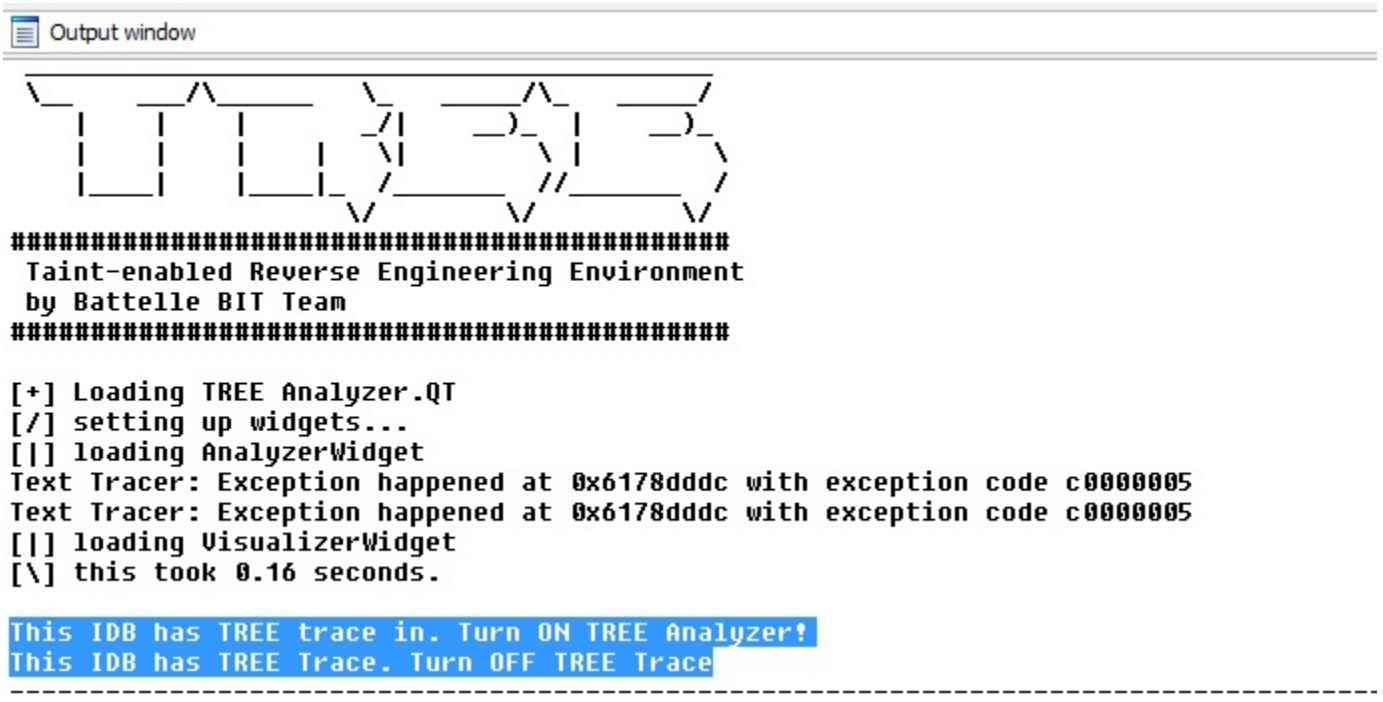
For interactive mode, user needs to provide the trace starting point and the ending point.

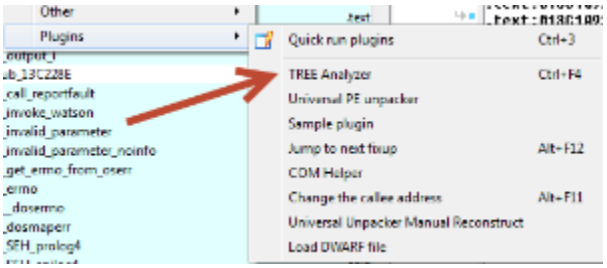


Click on the Trace button to start tracing. After tracing is done, the trace is saved into the IDB file, whose name starts with trace, along with a snapshot at the ending point. The traced IDB file is ready to load into Analyzer for analysis.

Analyzer

Edit->Plugin->Tree Analyzer





Use Basic Overflow Plus as a running example:

Basic Overflow Plus

Vulnerability Name: Basic Overflow Plus Example
Application Mode: User Mode
Target: Windows

Sample ran on:
Windows 7 32-bit for IDA 6.4 environment

Description

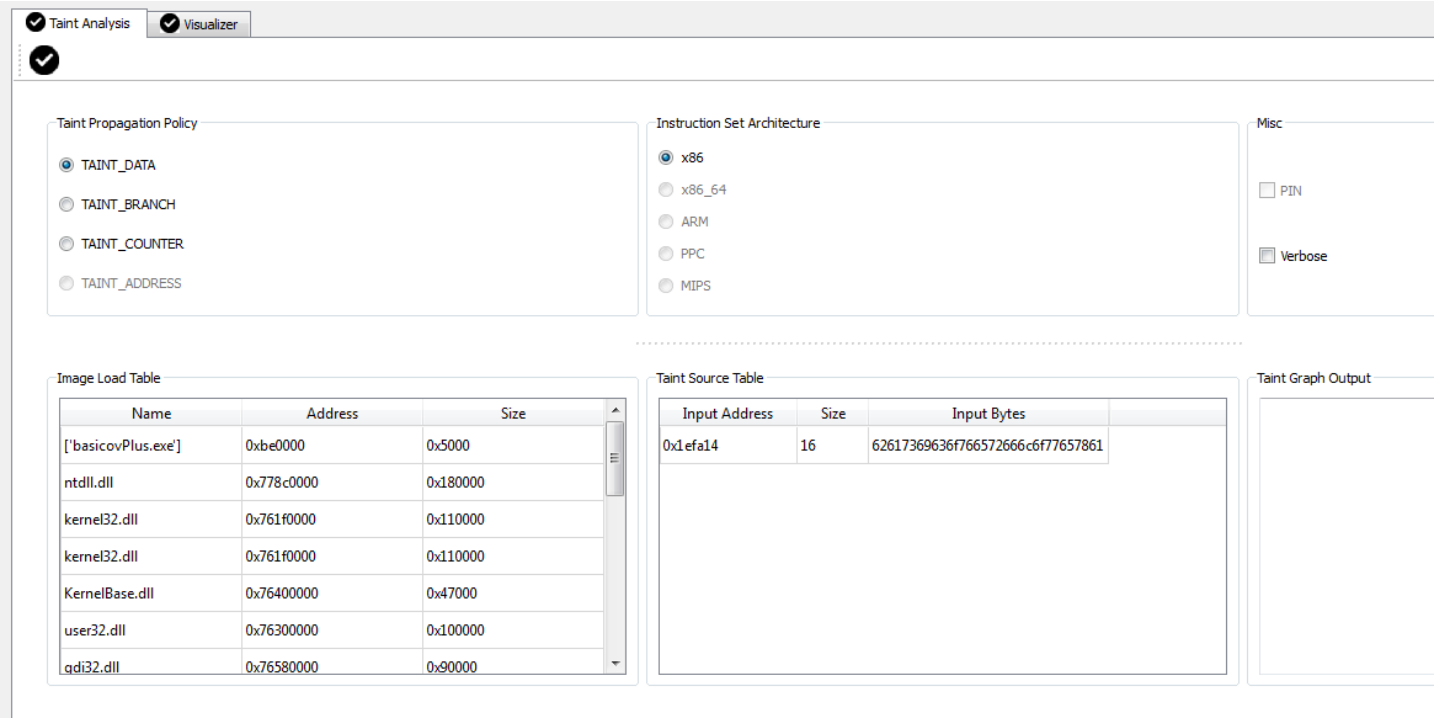
Basic buffer overflow. The overflow occurs when the input file "mytaint.txt" has more than 8 bytes.

BasicOVPlus.cpp

```
// filebufov.cpp : Defines the entry point for the console application.
//

#include "windows.h"
#include <stdio.h>

HANDLE hFile = NULL;

void StackOVflow(char * sBig,int num);

int main(int argc, char* argv[])
{
        char sBigBuf[16]={0};

        hFile = CreateFile("mytaint.txt",      // Open One.txt
                                        GENERIC_READ,           // Open for
reading
                                        0,                      // Do not share
                                        NULL,                   // No security
                                        OPEN_EXISTING,          // Existing file
only
                                        FILE_ATTRIBUTE_NORMAL,  // Normal file
                                        NULL);                  // No template
file

        if (hFile == INVALID_HANDLE_VALUE)
        {
                return 1;
        }

        DWORD dwBytesRead;
        ReadFile(hFile, sBigBuf, 16, &dwBytesRead, NULL);

        CloseHandle(hFile);

        for(int i=0; i< (dwBytesRead-2); i++)
                sBigBuf[i] +=sBigBuf[i+1];

        StackOVflow(sBigBuf,dwBytesRead);

        return 0;
}
```

```
void StackOVflow(char *sBig,int num)
{
        char sBuf[8]= {0};

        for(int i=0;i<num;i++)
        {
                sBuf[i] = sBig[i];
        }
        return;
}
```

- Select a taint propagation policy (default being TAINT_DATA):
- Select an instruction set architecture and optionally select verbose for extended output.
- Start the analyzer after inputting the appropriate options



The taint data is rendered into a table alongside a textbox containing the raw output of the results.

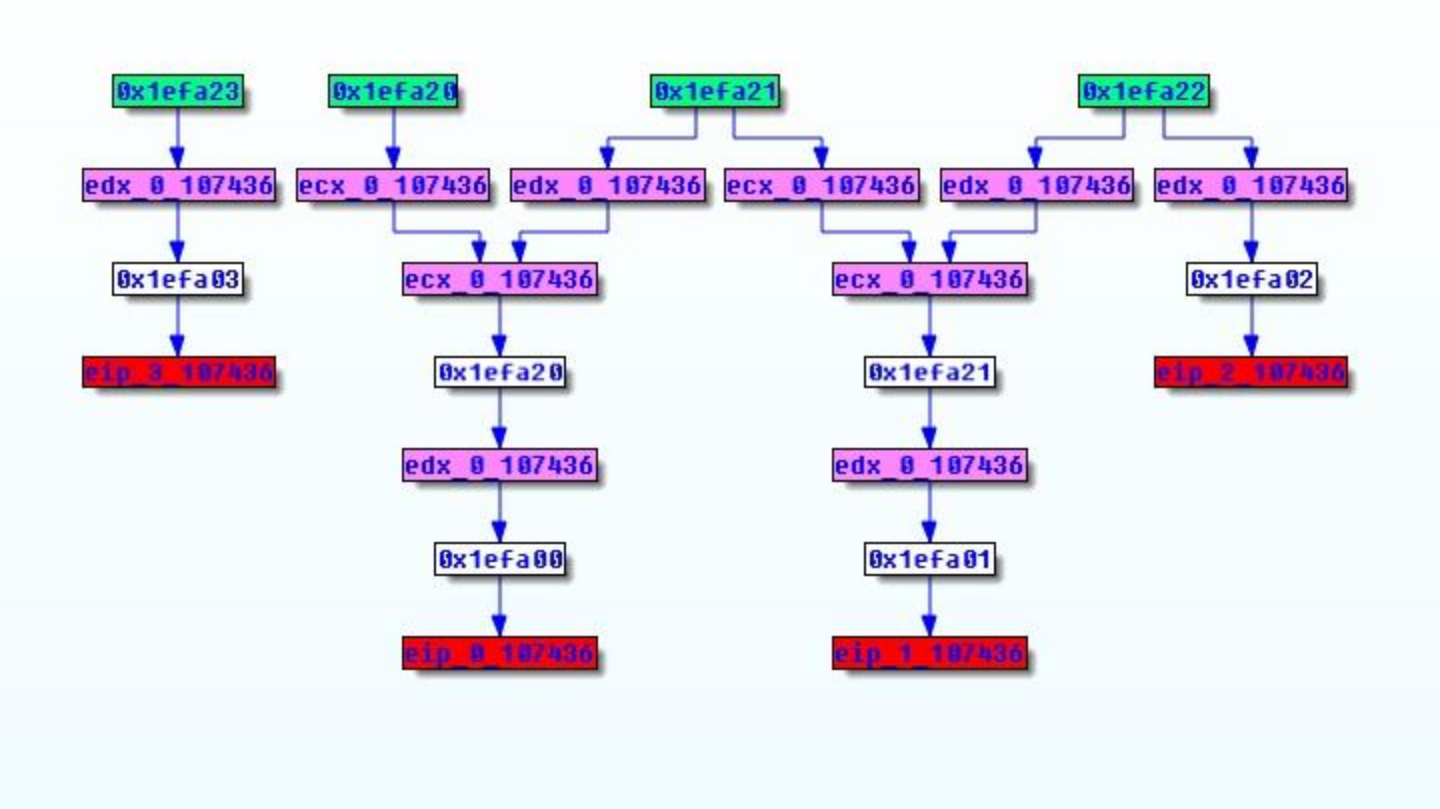
UUID	Type	Name	Start Sequence	End Sequence	Transformation Instruction	Child C	C
210	register	eip_3_107436	0x1da		retl		202
209	register	eip_2_107436	0x1da		retl		200
208	register	eip_1_107436	0x1da		retl		198
207	register	eip_0_107436	0x1da		retl		196
202	memory	0x1efa03	0x1d0		movb %dl, -0x8(%ebp,%ecx,1)		201
201	register	edx_0_107436	0x1cf	0x1d5	movb (%eax), %dl		16
200	memory	0x1efa02	0x1c4		movb %dl, -0x8(%ebp,%ecx,1)		199
199	register	edx_0_107436	0x1c3	0x1c9	movb (%eax), %dl		15
198	memory	0x1efa01	0x1b8		movb %dl, -0x8(%ebp,%ecx,1)		197
197	register	edx_0_107436	0x1b7	0x1bd	movb (%eax), %dl		170
196	memory	0x1efa00	0x1ac		movb %dl, -0x8(%ebp,%ecx,1)		195
195	register	edx_0_107436	0x1ab	0x1b1	movb (%eax), %dl		159
170	memory	0x1efa21	0xfd		movb %cl, -0x10(%ebp,%edx,1)		162
162	register	ecx_0_107436	0xfb	0x108	add %edx, %ecx		161 160
161	register	ecx_0_107436	0xfa		movsxb -0x10(%ebp,%eax,1), %ecx		14
160	register	edx_0_107436	0xf8	0xfc	movsxb -0xf(%ebp,%ecx,1), %edx		15
16	input	0x1efa23	0x0		0xbe1060		
159	memory	0x1efa20	0xee		movb %cl, -0x10(%ebp,%edx,1)		151
151	register	ecx_0_107436	0xec	0xf7	add %edx, %ecx		150 149
150	register	ecx_0_107436	0xeb		movsxb -0x10(%ebp,%eax,1), %ecx		13
15	input	0x1efa22	0x0		0xbe1060		
149	register	edx_0_107436	0xe9	0xed	movsxb -0xf(%ebp,%ecx,1), %edx		14
14	input	0x1efa21	0x0	0xfd	0xbe1060		
13	input	0x1efa20	0x0	0xee	0xbe1060		

Visualizer

Extended taint table view allows user easily track taint dependency. Select a child cell will highlight the taint flow (children rows).

Start the IDA grapher to display taint graph.

All the nodes are clickable; i.e. clicking on a taint node will jump to the instruction in IDA's disassembly view.



Limitation: IDA grapher only provides limited customizations. Therefore, we currently cannot add customized edges and annotations. Nevertheless, we are working on Qt solutions to solve this problem.

TREE Case Study – Sage

A complete example for both tracer and analyzer:

Sage

Application Mode: User Mode
Target: Windows

Sample ran on:
Windows 7 32-bit for IDA 6.4 environment

Description

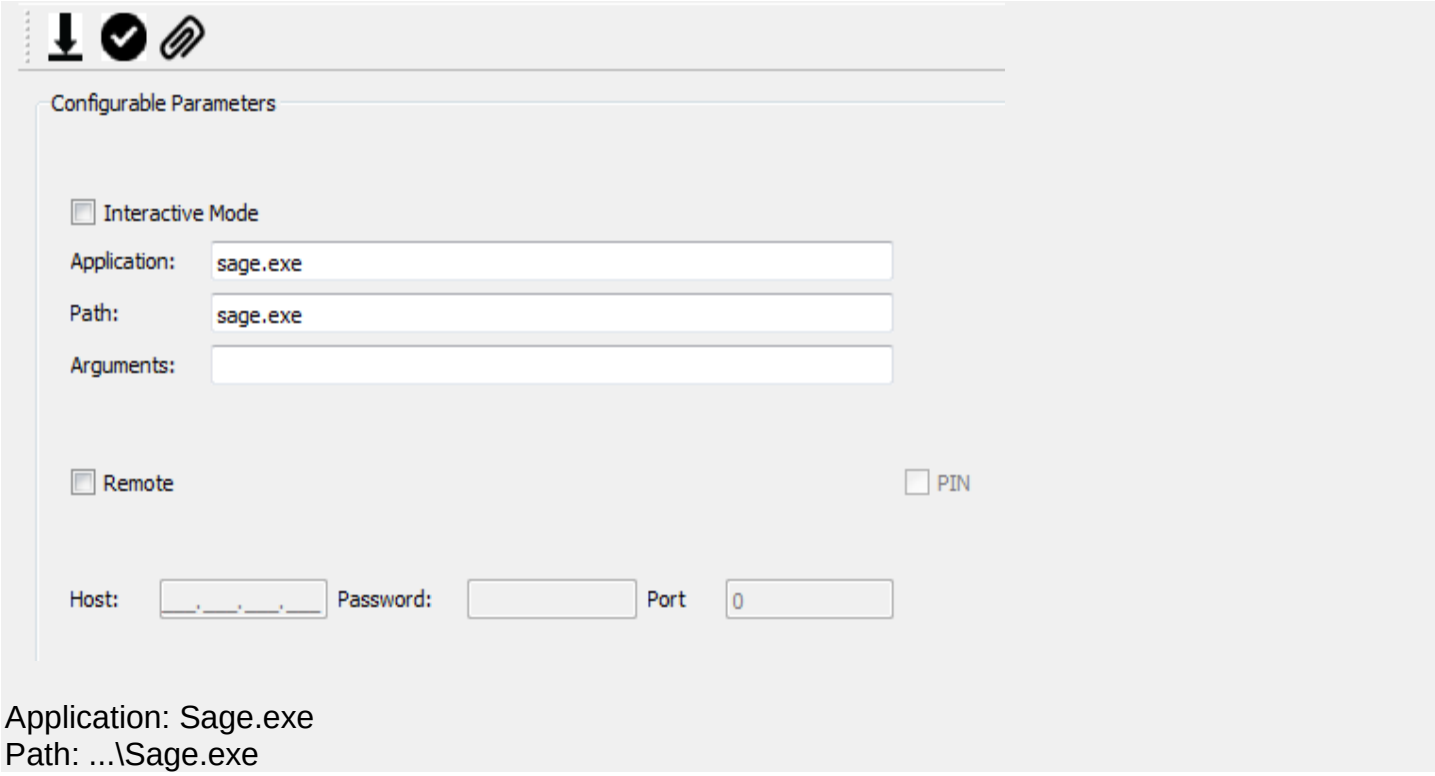
Sage example. Branch Condition propagation for input file.

Sage.c

```
#include "windows.h"
#include <stdio.h>

HANDLE hFile = NULL;

int main(int argc, char** argv)
{
	int  j = 0;
	DWORD dwBytesRead=0;
	char input[4];

	hFile = CreateFile("mytaint.txt",	// Open One.txt
				GENERIC_READ,		// Open for reading
				0,			// Do not share
				NULL,			// No security
				OPEN_EXISTING,		// Existing file only
				FILE_ATTRIBUTE_NORMAL,	// Normal file
				NULL);			// No template file

	if (hFile == INVALID_HANDLE_VALUE)
	{
		return 1;
	}

	ReadFile(hFile, input, 4, &dwBytesRead, NULL);

	CloseHandle(hFile);

	if (input[0] == 'b') j++;
	if (input[1] == 'a') j++;
	if (input[2] == 'd') j++;
	if (input[3] == '!') j++;
	if (j == 4) printf("bad!");
		else printf("ok!");
	return 0;
}
```

Configuring the Tracer

Indicate mytaint.txt file content

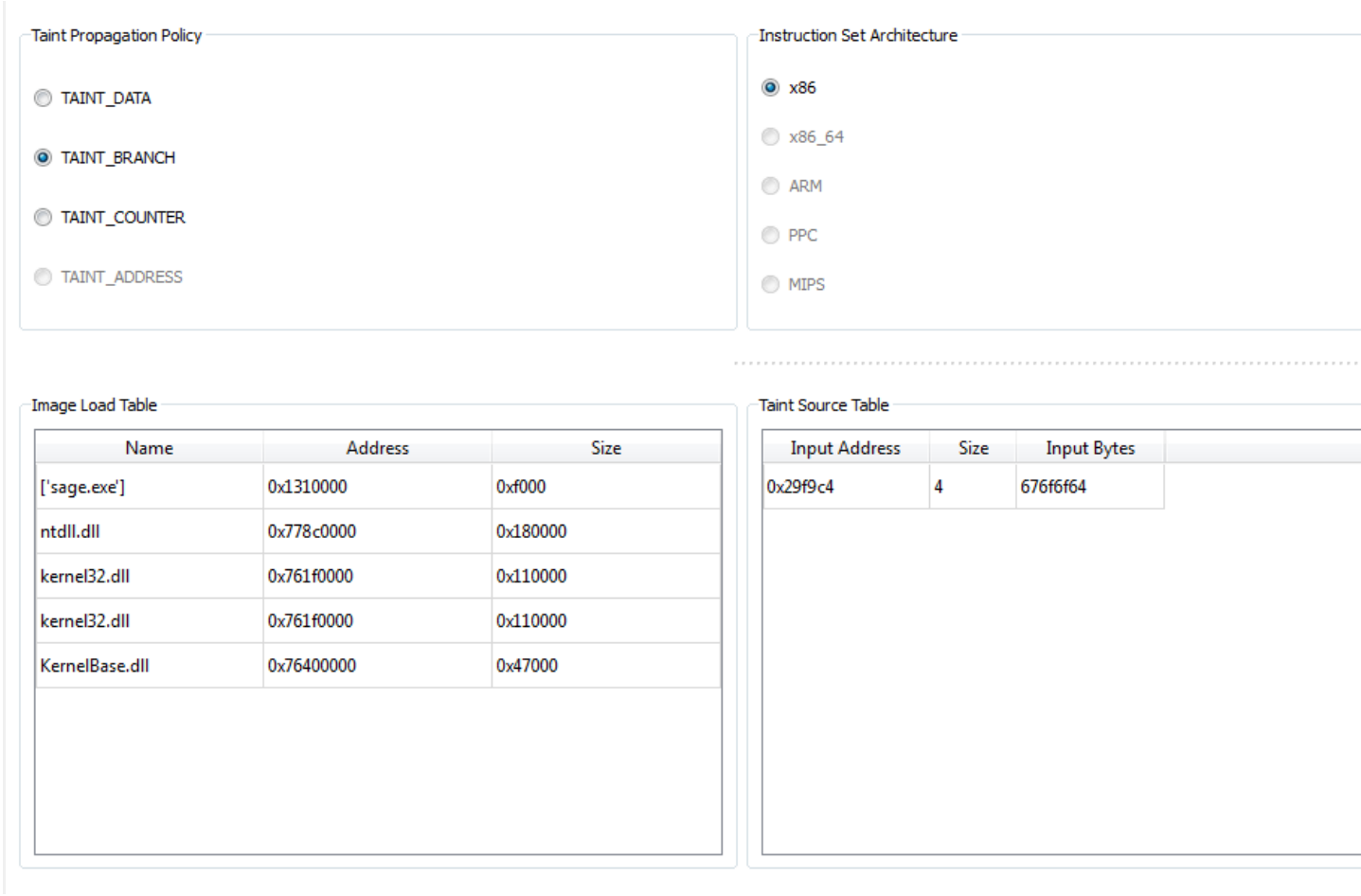


Application: Sage.exe
Path: ...\Sage.exe

Filters: mytaint.txt

Taint Propagation Policy - Taint Branch

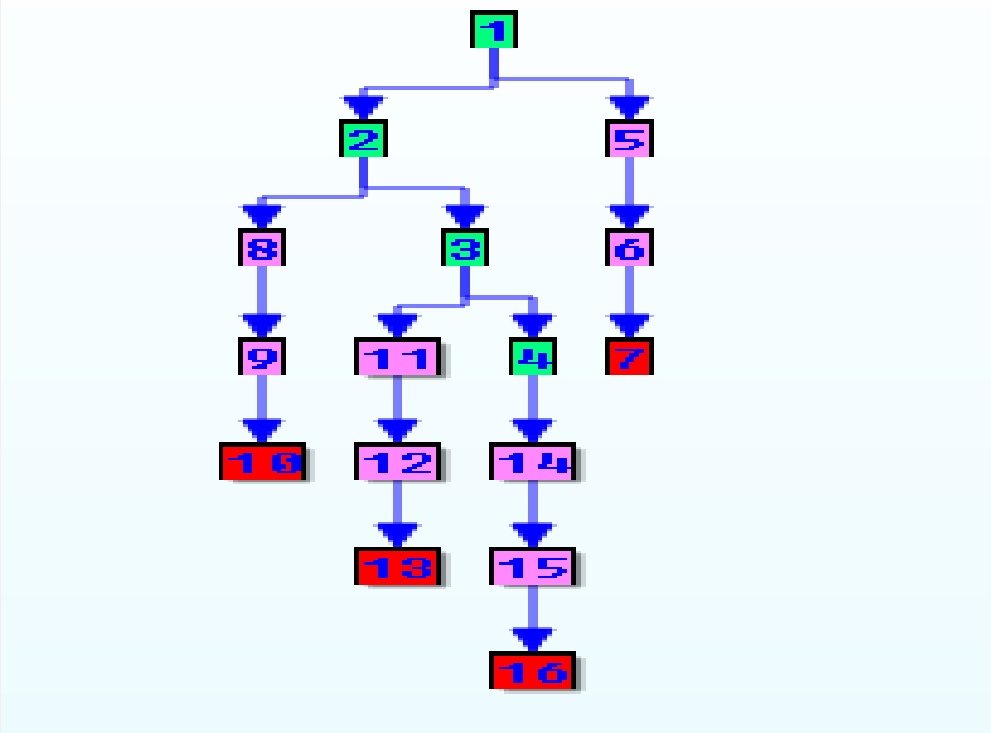
Taint source table indicates the source inputs and byte values.


Taint Propagation Policy
TAINT_DATA
TAINT_BRANCH
TAINT_COUNTER
TAINT_ADDRESS
Instruction Set Architecture
x86
x86_64
ARM
PPC
MIPS
Image Load Table
Name
Address
Size
['sage.exe']
0x1310000
0xf000
ntdll.dll
0x778c0000
0x180000
kernel32.dll
0x761f0000
0x110000
kernel32.dll
0x761f0000
0x110000
KernelBase.dll
0x76400000
0x47000
Taint Source Table
Input Address
Size
Input Bytes
0x29f9c4
4
676f6f64

Taint Table

UUID	Type	Name	Start Sequence	
6	register	eflags_100052	0x2f	0x32
16	branch	0x39	0x39	
4	input	0x29f9c7	0x0	
2	input	0x29f9c5	0x0	
9	register	eflags_100052	0x32	0x35
1	input	0x29f9c4	0x0	
7	branch	0x30	0x30	
11	register	edx_0_100052	0x34	0x109
12	register	eflags_100052	0x35	0x38
3	input	0x29f9c6	0x0	
10	branch	0x33	0x33	
14	register	ecx_0_100052	0x37	0x84
13	branch	0x36	0x36	
15	register	eflags_100052	0x38	
5	register	ecx_0_100052	0x2e	0x37
8	register	eax_0_100052	0x31	0x43

Taint Graph



(Note, node numbers may differ between traces and may not necessarily fully reflect the example)

Green Nodes indicate input nodes

Red Nodes indicate the sink and branch nodes