Microsoft

.NET malware dynamic instrumentation

Hexiang Hu, Microsoft
Steven Zhou, Microsoft
Geoff McDonald, Microsoft

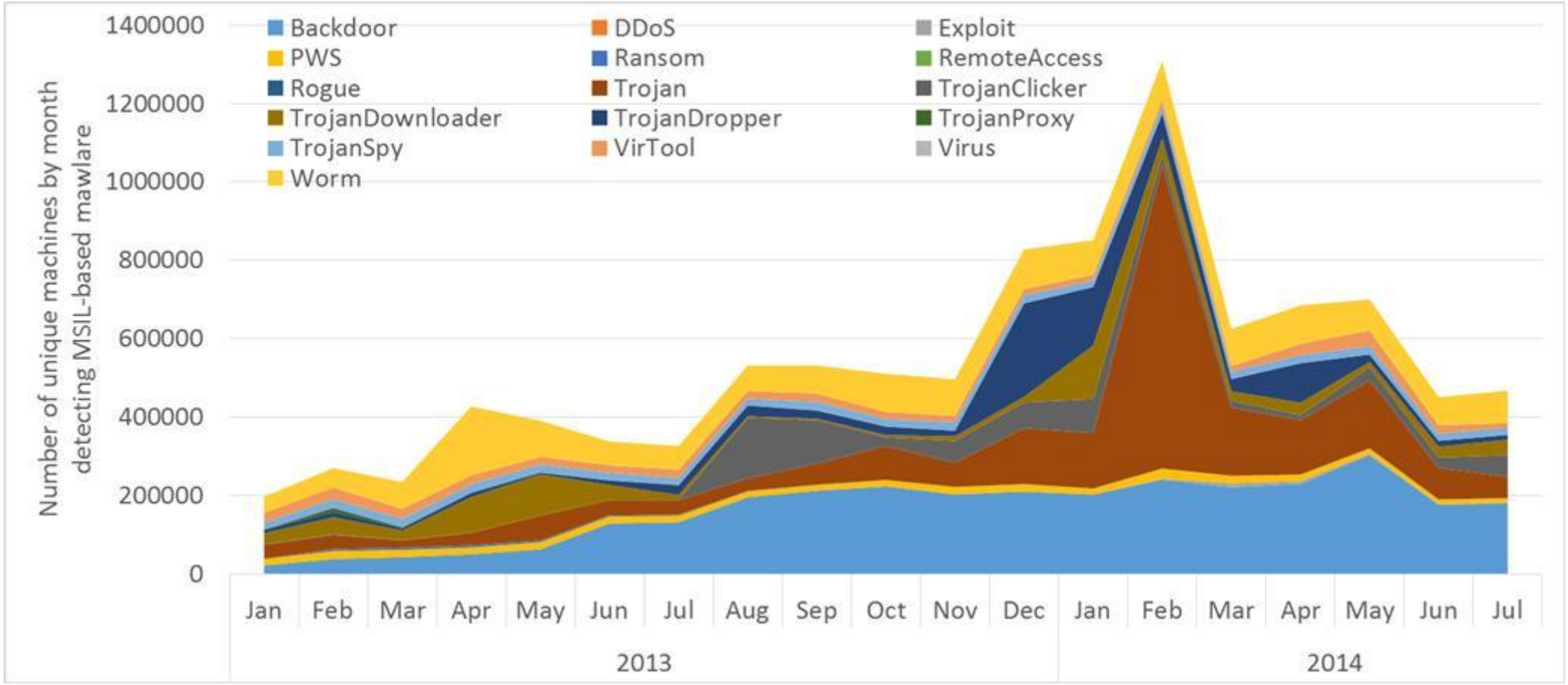
Introduction

Common Intermediate Language

Common Intermediate Language (CIL also called MSIL) is the lowest-level human-readable programming language defined by the Common Language Infrastructure (CLI) specification

It is used by .NET Framework

.NET malware



MSIL malware is a growing problem

.NET packer

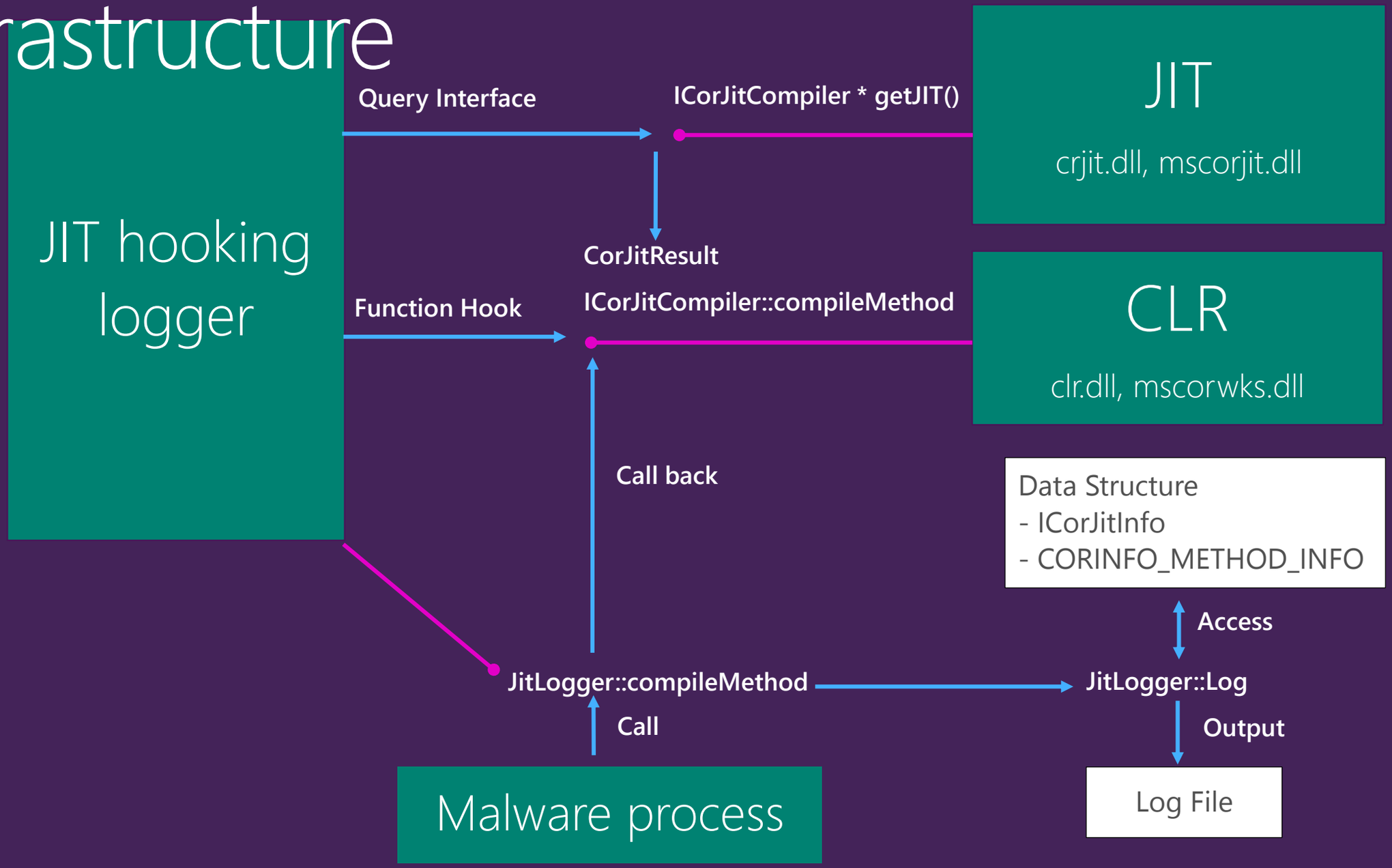
Large number of .NET malware families are using various custom and commercial .NET packers to obfuscate and pack their code

The packers can:

- Destroy the original data structure of the malware
- Hide or encrypt important string/function name inside
- Change the original flow of the program

JIT compile hooking method

Infrastructure



Problem with JIT hooking

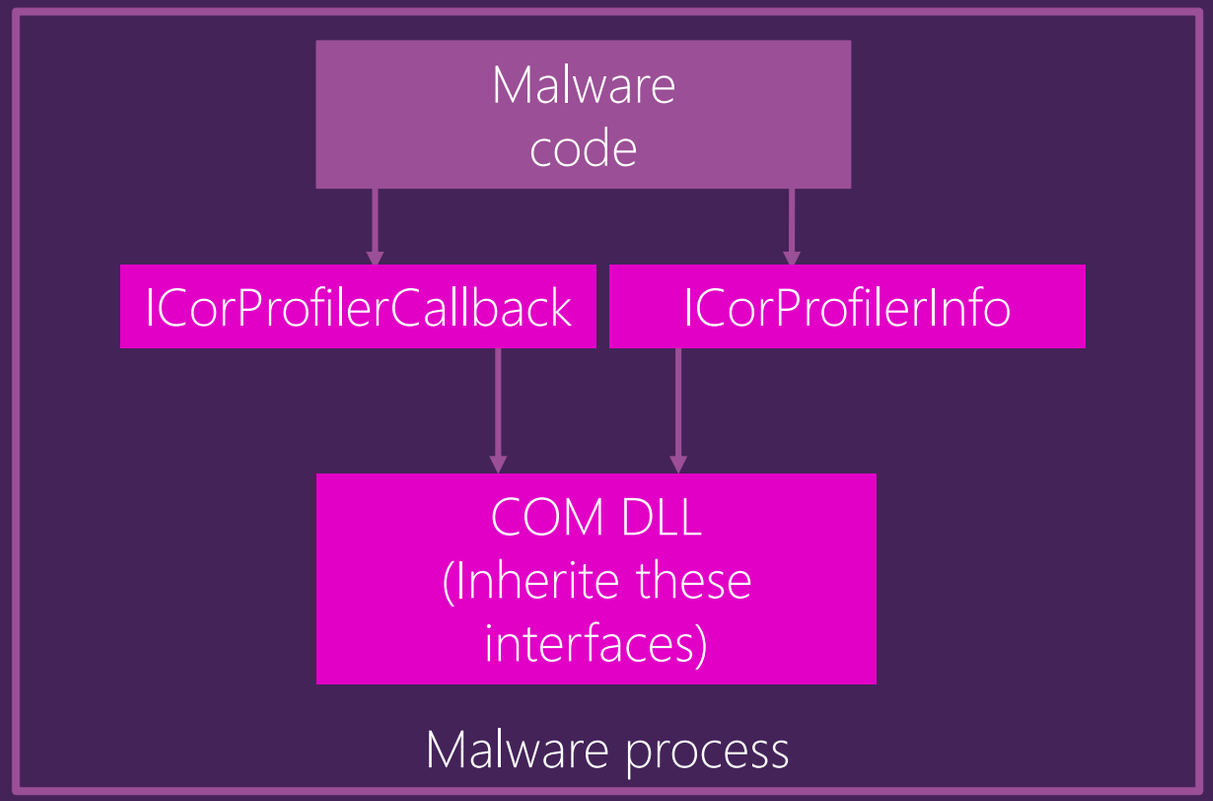
Since the .NET framework changes, it is time-consuming to support all versions

The compileMethod() hooking only enables logging info at compilation time, as opposed to every time it is called

If the method is pre-JITed, the tool will be unable to record any information

.NET instrumentation framework

Core technique

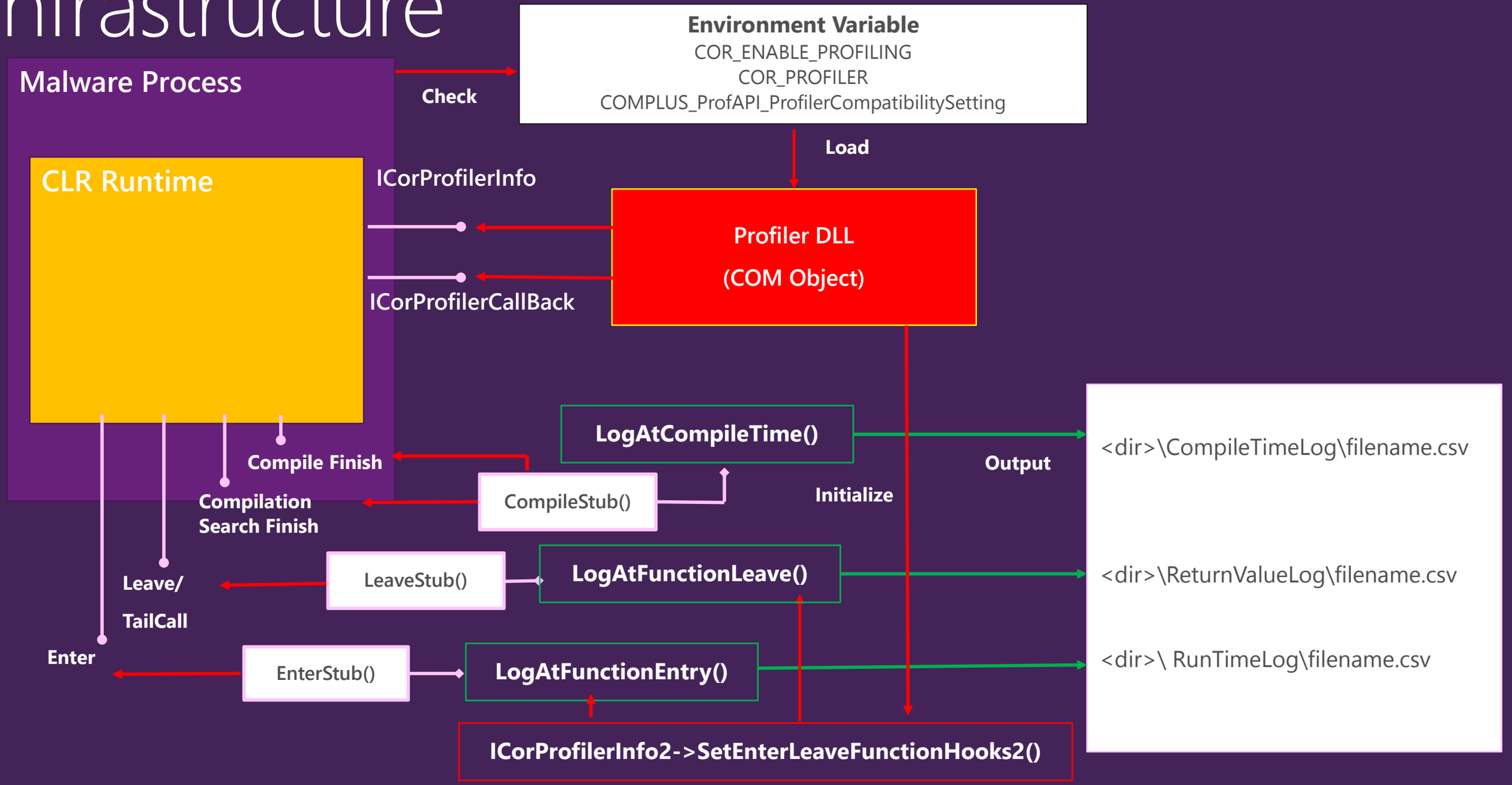


Profiling API

ICorProfilerCallback
Based on CLR internal event, run event's callback routine

ICorProfilerInfo
Retrieve CLR internal structure value

Infrastructure



Challenges

Parsing function parameter value

1. Undocumented .NET internal variable type parsing
2. Program runtime parameter value retrieval
3. Anti-profiling technique

Problem with generic information representation

1. Representation of function
2. Representation of arguments regarding to a function

.NET internal type parsing

Refer to Microsoft-released source code of CLR profiler, most .NET internal types can be correctly parsed

But it doesn't provide enough support for generic types

CorElementType	Description
ELEMENT_TYPE_GENERICINST	A type modifier for generic types

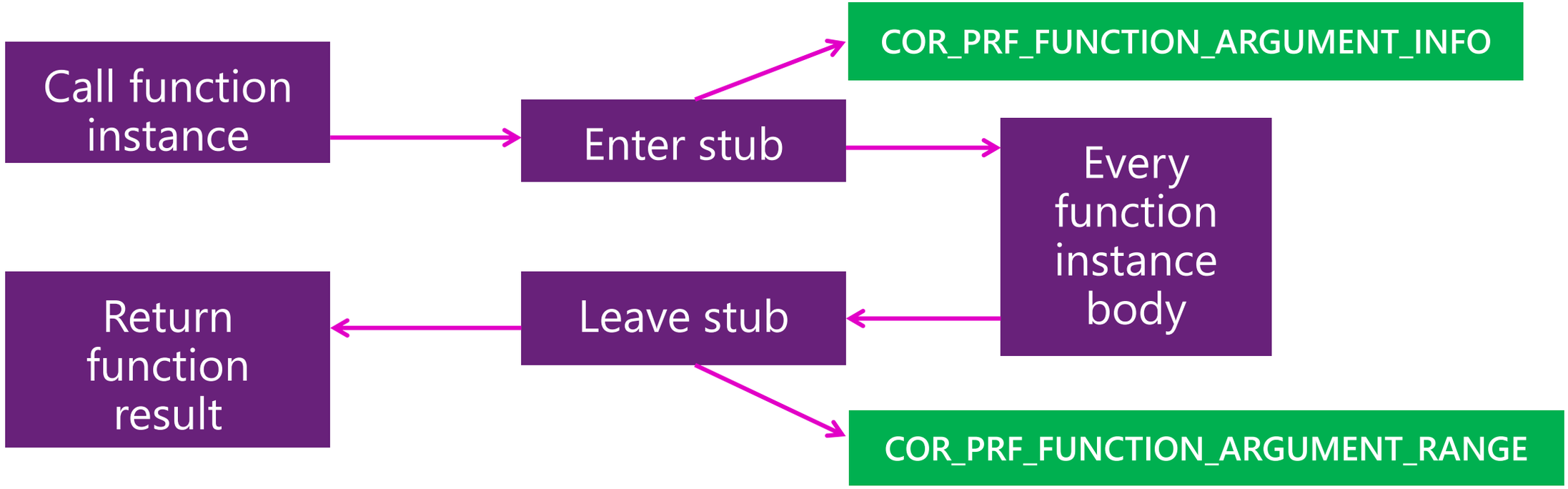
Its signature blob contains following structure:

```
sigblob[0]      = ELEMENT_TYPE_GENERICINST;
sigblob[1]      = N;                 //number that indicates the dimension of generic type
sigblob[2]      = TypeToken;         // token that indicates type of first dimension
……
sigblob[N+1]    = TypeToken;         // token that indicates type of last dimension
```

Runtime parameter value retrieval

Shadow Stack

Using the profiling interface FunctionEnter2, FunctionLeave2, FunctionTailcall2 to query stack information for all positions



Anti-profiling

It is popular for current commercial obfuscators to avoid profiling with checking environment

Solution

Identify the function used for checking environment variable and modify the return value since we can access managed function stack

For more advanced anti-profiling technique

Hook that detection function and modify its return value once we find it out

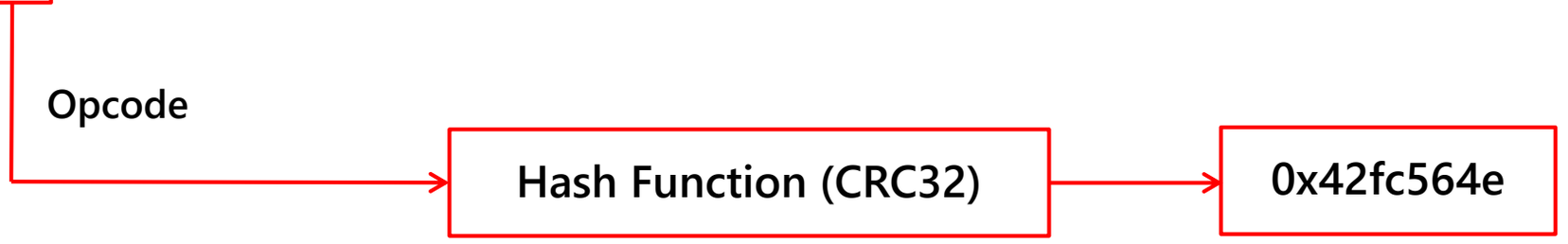
Generic hash

```
IL_0000:  00
IL_0001:  1F 63          ldc.i4.s 99
IL_0003:  6A             conv.i8
IL_0004:  13 05          stloc.s V_5
IL_0006:  1F 4B          ldc.i4.s 75
IL_0008:  6A             conv.i8
IL_0009:  13 06          stloc.s V_6
IL_000b:  20 8D030000    ldc.i4 0x38d
IL_0010:  6A             conv.i8
IL_0011:  13 07          stloc.s V_7
IL_0013:  28 (06)000002  call int64 Project1.Program::IsDebuggerPresent()
IL_0018:  26             pop
IL_0019:  11 06          ldloc.s V_6
IL_001b:  2B 20          br.s
```

Generic Function Hash:

1. Use IL Opcode hash to increase generic impact
2. Exclude “nop” so as to avoid a simple case of IL obfuscation

Generic Argument Value Hash = Hash(IL Opcode + argument value in string format)



Data output

Compilation time function signature

Function ID	Function signature	IL generic checksum	IL full checksum	IL assembly
17478236	Static void j.Ok::ins()	F0537728	45641a07	20e80300002851000000a7e0b000004390b0100007e0e0000047e0500000428 3a00000a6f4400000a72790200707e04000004e…

Runtime function parameter value

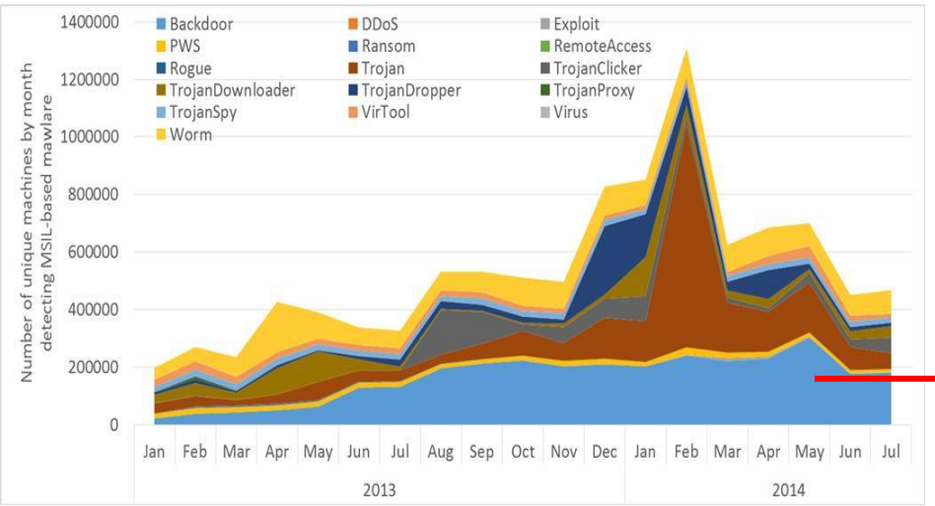
Thread ID	Function ID	Function signature	Argument information
5305416	11800204	static System.Diagnostics.Process System.Diagnostics.Process::Start(String)	[{"Address":"3203572","Type":"String","Value":"C:\Users\User00001\AppData\Roaming\WindowsLogs.exe","CRC32":"73216f87"}]

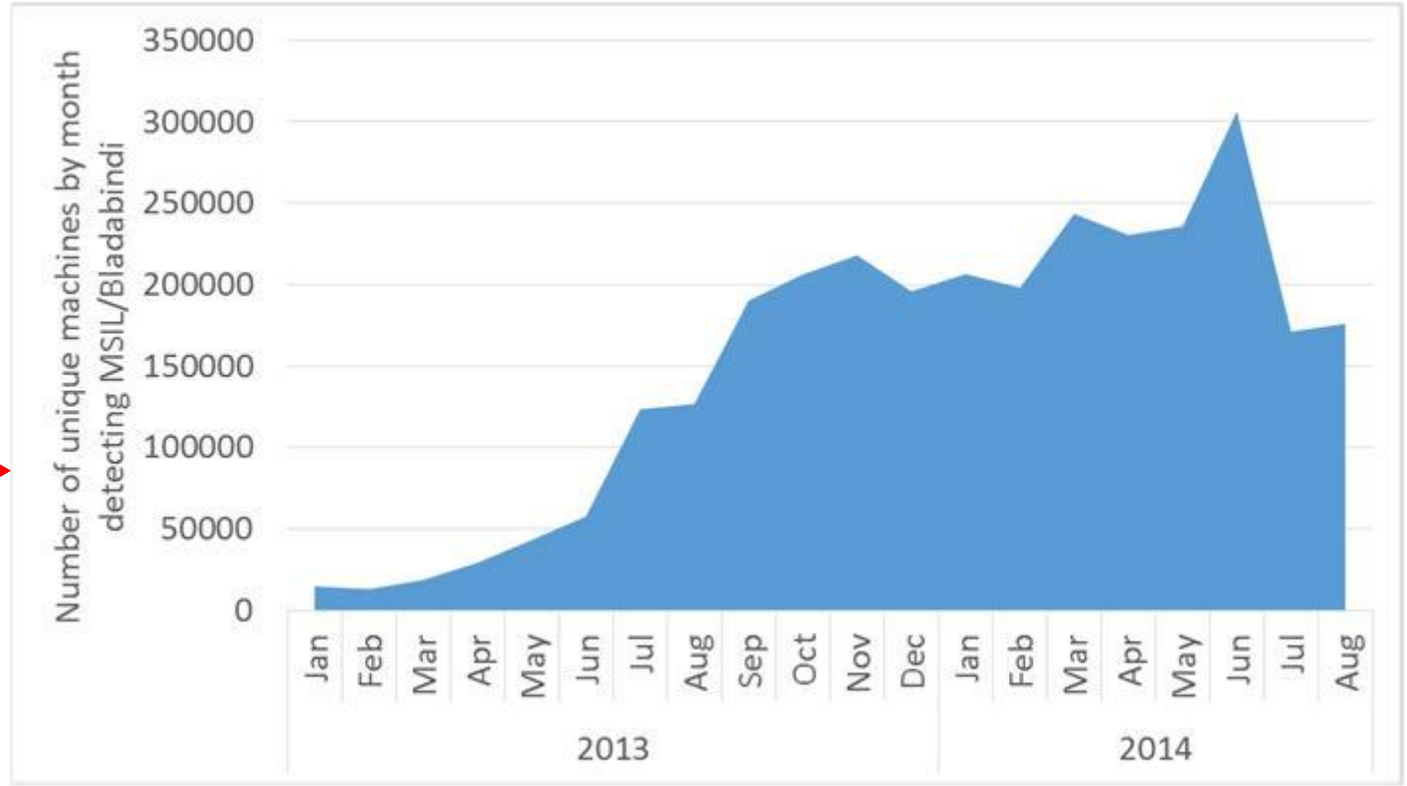
Runtime function return value

Thread ID	Function ID	Function signature	Argument information
2572952	17477820	static bool j.OK::CompDir(System.IO.FileInfo,System.IO.FileInfo)	{"Address":"1960772","Type":"bool","Value":"True"}

Bladabindi case study

Backdoor: MSIL/Bladabindi





Bladabindi is a particular problem

Backdoor: MSIL/Bladabindi

.NET framework based backdoor Trojan

keylogger, remote access, usb spread

Uses various commercial obfuscators

.NET Reactor, DeepSea, SmartAssembly, Crypto, Confuser, etc.

Behavior validation

Behavior	Profiled Log
Drops file "e6lo5xeg22fb3xp0tnod.exe“ **[Drop virus file in temp directory]**	System.IO.File::WriteAllBytes(String:"C:\Users\User00099\AppData\Roaming\E6lO5xEG22fB3Xp0tnod.exe",unsigned int8[]:"")
Drops itself as "internet.exe" **[Drop file that will be run as backdoor server]**	System.IO.FileStream::.ctor(this:"26198672",String:"C:\Users\User00099\AppData\Roaming\internet.exe",System.IO.FileMode:"")
Launches the file “netsh firewall add allowedprogram "C:\Documents and Settings\Administrator\Application Data\internet.exe" "internet.exe" ENABLE” **[Modify system firewall setting]**	static int32 Microsoft.VisualBasic.Interaction::Shell(String:"netsh firewall add allowedprogram "C:\Documents and Settings\Administrator\Application Data\internet.exe" "internet.exe" ENABLE",Microsoft.VisualBasic.AppWinStyle:"",bool:"True",int32:"5000")
Attempts connecting to nj7-mikey.no-ip.org at TCP **[Attempt remote connection]**	void System.Net.Sockets.TcpClient::.ctor(this:"25970848",System.Net.Sockets.AddressFamily:"")

Backdoor: MSIL/Bladabindi - a954c1e3104e119ff683bd2fc549dba4cd1568ab

Generic detection impact evaluation

Function Info	Checksum
static void j.A::main()	95b269a2
static void j.OK::.cctor()	cdf1d762
static void j.OK::ko()	587a285b
static void j.OK::INS()	f0537728
static bool j.OK::CompDir(System.IO.FileInfo:"27332448",System.IO.FileInfo:"27344224")	d62ad292
static void j.OK::RC()	9016078b
static bool j.OK::connect()	601a0096
void j.kl::.ctor(this:"23206744")	42fc564e
void j.kl::WRK(this:"23206744")	ff1f9519
static Object j.OK::GTV(String:"[kl]",Object:"")	f8be4f26
static void j.OK::pr(int32:"1")	cfd01a73
static String j.OK::ACT()	d957e1f2
static bool j.OK::Send(String:"act|'|'|MGU4YjEzYzliMDdkOGFiZTZiM2YzZWY0ZmQ1NWYyNGZhMzQwMzk5NSAtIEZhdGFsIGVycm9yAA==")	45206646
static bool j.OK::Sendb(unsigned int8[]:"")	a6846c26
static String j.OK::ACT()	d957e1f2

0e2ef3bd304ee78ed9cae5d2d6d309920b3a0aaa

Obfuscated by Yano

Function Info	Checksum
static void USG_STUB.Module1::.cctor()	cf105e84
static void USG_STUB.Module1::Main()	2773c8ec
static void j.A::main()	95b269a2
static void j.OK::.cctor()	cdf1d762
static void j.OK::ko()	587a285b
static void j.OK::INS()	f0537728
static bool j.OK::CompDir(System.IO.FileInfo:"29913544",System.IO.FileInfo:"29924312")	d62ad292
void j.kl::.ctor(this:"8704112")	42fc564e
void j.kl::WRK(this:"8704112")	ff1f9519
static Object j.OK::GTV(String:"[kl]",Object:"")	f8be4f26
static void j.OK::RC()	9016078b
static bool j.OK::connect()	601a0096
static void j.OK::pr(int32:"1")	cfd01a73
static String j.OK::ACT()	d957e1f2
static bool j.OK::Send(String:"act|'|'|QzpcUHJvZ3JhbSBGaWxlc1xNYWx3YXJlXFJ1bm5lci5leGUA")	45206646
static bool j.OK::Sendb(unsigned int8[]:"")	74a9b20b

0e7935efe3218f4bde2ddb26a016964e21dec517

Base64 Encoded by Customized obfuscator

All Functions are detected Bladabindi internal function

Statistics

Bladabindi Case:

Lower bound = detected sha1/ total sha1
Upper bound = detected sha1/good sha1

Total samples processed: 585 random samples out of 4500+ samples

Successfully profiled samples: 585/585 random samples

Successfully behavior detected: 369/585 random samples

Function sig	IL generic CRC32	Detected SHA1 #	Good & working SHA1#	Total SHA1#	Detection ratio
A::INS()	f0537728	333	369	585	0.570 ~ 0.902
A::CompDir()	d62ad292	318	369	585	0.544 ~ 0.862
A::RC()	9016078b	236	369	585	0.403 ~ 0.640
A::GTV()	f8be4f26	237	369	585	0.403 ~ 0.641
A.KL::WRK()	ff1f9519	237	369	585	0.403 ~ 0.641

Analysis

Obfuscator cases

The following obfuscators get good generic detection

SmartAssembly, Crypto Obfuscator, Babel, DeepSea, Confuser, Yano

Problem

Problem with samples not running correctly

1. Anti-VM techniques
2. Anti-profiling techniques
3. User privilege restriction

Problem with generic IL checksum

IL obfuscation

Future machine learning

More info from argument value

Recap with argument hash

For each argument in a function instance:

Argument hash = hash(generic IL Opcode + argument value)

Functionality

Binds argument value with function IL opcode, which we can use to identify specific behavior

e.g. CRC32[(static int32 Microsoft.VisualBasic.Interaction::Shell) + (“netsh firewall add allowedprogram /“C:\Users\av\AppData\Roaming\svchost.exe/“ /“svchost.exe/“ ENABLE”)] = 7ef45f07

This feature enables machine to learn about program’s behavior

Potential application

Auto classification

Determine whether a .NET program is malware or not by learning massive instrumented program data with current label

Auto clustering

Cluster .NET programs that have either similar behaviors (inspect through argument value hash) or similar functions (inspect through generic IL hash) into a category

References

[1] CLR Profilier, Microsoft, 14 Nov 2012, http://clrprofiler.codeplex.com/

[2] Shared Source Common Language Infrastructure 2.0

http://www.microsoft.com/en-ca/download/details.aspx?id=4917

[3] NET Internals and Code Injection, Daniel Pistelli, 14 May 2008, http://www.codeproject.com/Articles/26060/NET-Internals-and-Code-Injection

[4] DisasMSIL - MSIL disasm engine, Daniel Pistelli, 14 May 2008, http://www.ntcore.com/files/disasmsil.htm

[5] Profiling (Unmanaged API reference), Microsoft, 2014 http://msdn.microsoft.com/en-us/library/ms404386(v=vs.110).aspx