WHY ATTACKERS SHOULD AVOID C#

Alex Davies - @pwndizzle

26 November 2019

I WANT TO RUN SOME CODE

1. Run compiled binary:

cmd /c malware.exe
rundll32 malware.dll

2. Run a script:

wscript malware.vbs

3. Execute directly in memory:

VirtualAlloc -> WriteProcessMemory -> CreateRemoteThread

F-Secure

BUT WHICH LANGUAGE?

Language	Dev Overhead	Native Support	Native Logging
Assembly	Insane	Yes	No
C / C++	High	Yes	No
C#	**Medium**	**Yes**	**YES**
Go	Medium	Yes	No
Java	Medium	No	No
Perl	Low	No	No
Powershell	Low	Yes	Yes
Python	Low	No	No
Ruby	Low	No	No
Rust	Medium	Yes	No
VBScript	Low	Yes	AMSI

F-Secure

.NET IS THE NEW PS RIGHT?

Ways to execute code:

1. Compile code then deploy binary
e.g. <.NET PATH>\csc.exe payload.cs

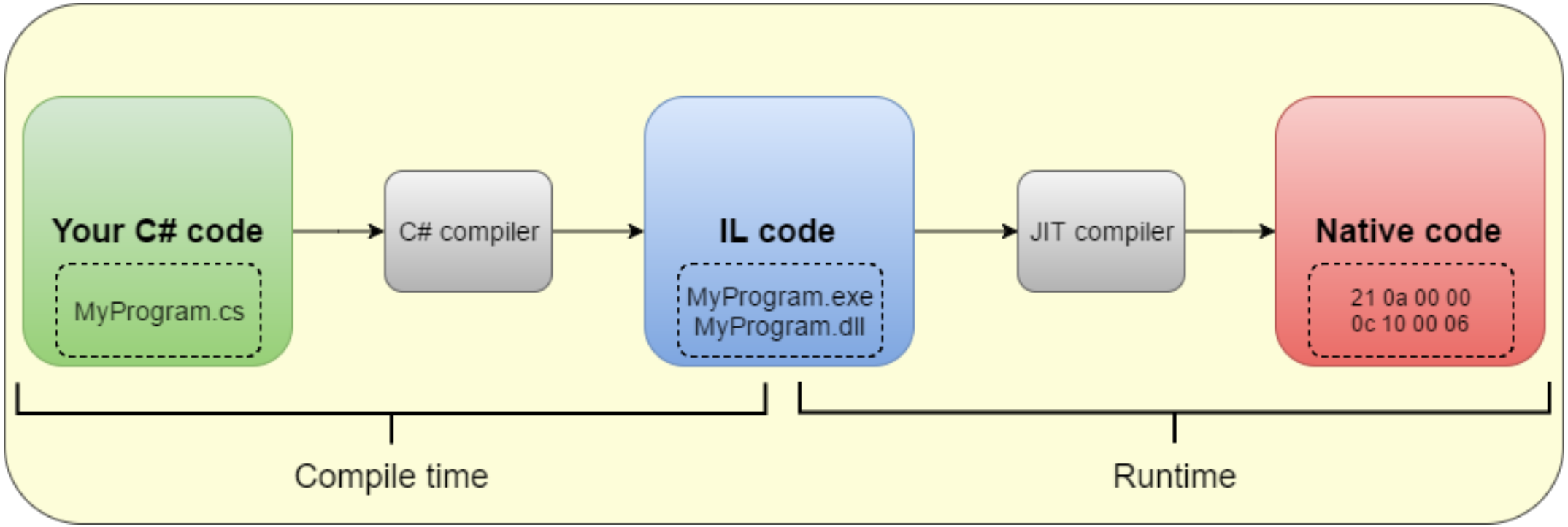
2. Using .NET Framework tools + Build Tasks
e.g. <.NET PATH>\msbuild.exe pshell.xml

3. In-memory with third party tools like Cobaltstrike's "execute assembly"

TLDR, C# is a good choice for attackers

F-Secure

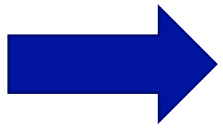
HOW DOES C# EXECUTE?



https://codeasy.net/lesson/c_sharp_compilation_process

F-Secure

ENTER .NET ETW PT1



Event Logs/Sysmon

Pro – Free and good visibility
Con – Pain to deploy/setup, no .NET

EDR (Driver)

Pro – Deep visibility + other goodies
Con – High cost + typically no .NET

ETW

Pro – Decent visibility including .NET!
Con – Performance overheads

F-Secure

ENTER .NET ETW PT2

Microsoft-Windows-DotNETRuntime
{E13C0D23-CCBC-4E12-931B-D9CC2EEE27E4}

- AssemblyLoad_V1
- ModuleLoad_V2
- DomainModuleLoad_V1
- MethodName
- MethodJittingStarted
- MethodJitInliningSucceeded
- MethodJitInliningFailed
- ILStubGenerated

Microsoft-Windows-DotNETRuntimeRundown
{A669021C-C450-4609-A035-5AF59AF4DF18}

- AssemblyDCStart_V1
- ModuleDCStart_V2
- DomainModuleDCStart_V1
- DCStartComplete_V1

https://gist.github.com/countercept/7765ba05ad00255bcf6a4a26d7647f6e

NOTE: .NET 3.5 only supports Method info

F-Secure

GHOSTPACK SAFETYKATZ

MethodName

```
MethodName, SafetyKatz.Constants, .cctor
MethodName, SafetyKatz.NativeDeclarations, .cctor
MethodName, SafetyKatz.Program, Main
MethodName, SafetyKatz.Program, IsHighIntegrity
MethodName, SafetyKatz.Program, Minidump
MethodName, dynamicClass, IL_STUB
MethodName, SafetyKatz.PELoader, .ctor
MethodName, SafetyKatz.PELoader, FromBinaryReader
MethodName, SafetyKatz.PELoader, FromBinaryReader
MethodName, SafetyKatz.PELoader, get_Is32BitHeader
MethodName, SafetyKatz.PELoader, get_FileHeader
MethodName, SafetyKatz.PELoader, FromBinaryReader
MethodName, SafetyKatz.PELoader, FromBinaryReader
MethodName, SafetyKatz.PELoader, get_OptionalHeader64
MethodName, dynamicClass, IL_STUB
MethodName, SafetyKatz.PELoader, get_ImageSectionHeaders
MethodName, SafetyKatz.PELoader, get_RawBytes
```

MethodJitInliningFailed

```
SafetyKatz.Program, Main, SafetyKatz.NativeDeclarations, VirtualAlloc
SafetyKatz.Program, Main, SafetyKatz.NativeDeclarations, VirtualAlloc
SafetyKatz.Program, Main, System.Runtime.InteropServices.Marshal, PtrToStructur
SafetyKatz.Program, Main, System.Runtime.InteropServices.Marshal, SizeOf
SafetyKatz.Program, Main, System.Runtime.InteropServices.Marshal, PtrToStructur
SafetyKatz.Program, Main, System.Runtime.InteropServices.Marshal, ReadInt16
SafetyKatz.Program, Main, System.Runtime.InteropServices.Marshal, PtrToStringAr
SafetyKatz.Program, Main, SafetyKatz.NativeDeclarations, LoadLibrary
SafetyKatz.Program, Main, System.Runtime.InteropServices.Marshal, PtrToStringAr
SafetyKatz.Program, Main, SafetyKatz.NativeDeclarations, GetProcAddress
SafetyKatz.Program, Main, System.Console, WriteLine
SafetyKatz.Program, Main, SafetyKatz.NativeDeclarations, CreateThread
SafetyKatz.Program, Main, SafetyKatz.NativeDeclarations, WaitForSingleObject
```

Lots of obviously malicious Methods and APIs!

ILStub

```
10624, ILStubGenerated, None, SafetyKatz.Program, MiniDumpWriteDump
10624, ILStubGenerated, None, SafetyKatz.NativeDeclarations, VirtualAlloc
10624, ILStubGenerated, None, SafetyKatz.NativeDeclarations, LoadLibrary
10624, ILStubGenerated, None, SafetyKatz.NativeDeclarations, GetProcAddress
10624, ILStubGenerated, None, SafetyKatz.NativeDeclarations, CreateThread
10624, ILStubGenerated, None, SafetyKatz.NativeDeclarations, WaitForSingleObject
```

F-Secure

GHOSTPACK SEATBELT

MethodName

```
MethodName, Seatbelt.Program, ListAuditSettings
MethodName, Seatbelt.Program, ListWEFSettings
MethodName, Seatbelt.Program, ListLSASettings
MethodName, Seatbelt.Program, ListUserEnvVariables
MethodName, Seatbelt.Program, ListSystemEnvVariables
MethodName, Seatbelt.Program, ListUserFolders
MethodName, Seatbelt.Program, ListNonstandardServices
MethodName, Seatbelt.Program, ListInternetSettings
```

ILStub

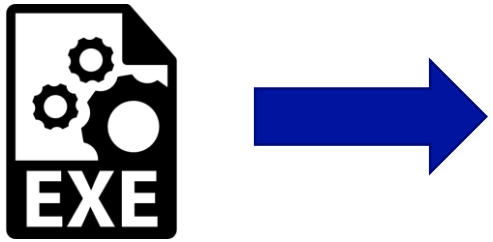
```
Seatbelt.Program, LsaRegisterLogonProcess
Seatbelt.Program, OpenProcessToken
Seatbelt.Program, DuplicateToken
Seatbelt.Program, ImpersonateLoggedOnUser
Seatbelt.Program, CloseHandle
Seatbelt.Program, RevertToSelf
Seatbelt.Program, LsaEnumerateLogonSessions
Seatbelt.Program, LsaGetLogonSessionData
Seatbelt.Program, LsaLookupAuthenticationPackage
Seatbelt.Program, LsaCallAuthenticationPackage
Seatbelt.Program, LsaFreeReturnBuffer
Seatbelt.Program, LsaDeregisterLogonProcess
```

Again lots of obvious functions and API's to detect!

```
ILStubGenerated, None, Seatbelt.VaultCli, VaultEnumerateVaults
ILStubGenerated, None, Seatbelt.VaultCli, VaultOpenVault
ILStubGenerated, None, Seatbelt.VaultCli, VaultEnumerateItems
ILStubGenerated, None, Seatbelt.VaultCli, VaultGetItem_WIN8
```

F-Secure

COBALTSTRIKE EXECUTE ASSEMBLY




```
beacon> execute-assembly C:\Users\1\Downloads\gp\Rubeus-v40.exe triage
[*] Tasked beacon to run .NET program: Rubeus-v40.exe triage
[+] host called home, sent: 315447 bytes
[+] received output:
```

- No obvious MethodNames or APIs from a .NET telemetry perspective, but….
- Uses process injection:

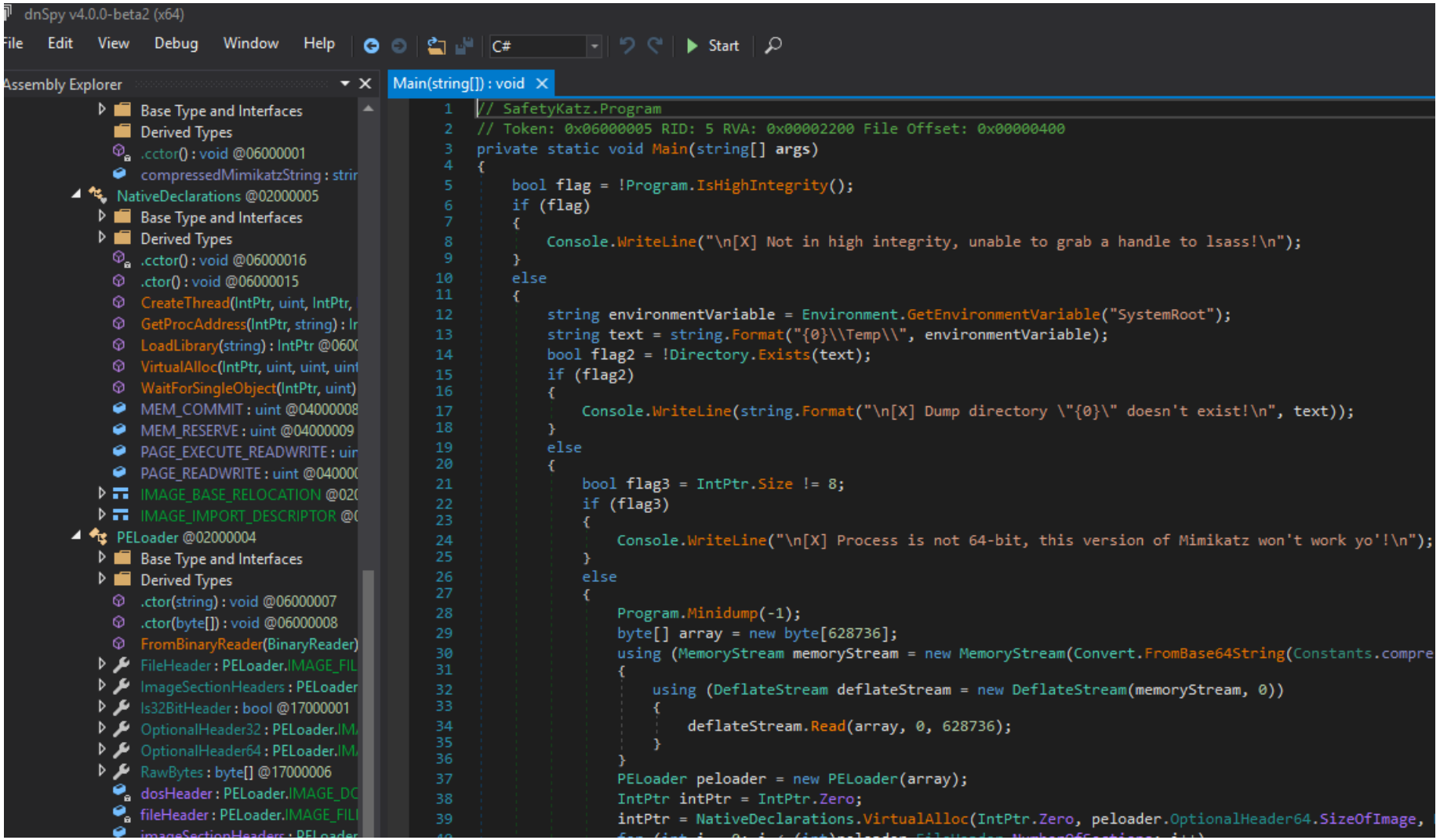
ScanResultType	ExecutableInfo.Name	TargetExecutableInfo.Name
ThreadInjectionEtw	artifact.exe	rundll32.exe

- Loads a module from memory:

Execution Technique	Event Type	ModuleILPath
Run binary from cmd	ModuleLoad_V2	c:\Users\1\Downloads\gp\Rubeus-v40.exe
Execute-Assembly	ModuleLoad_V2	Rubeus (Really anomalous!)

F-Secure

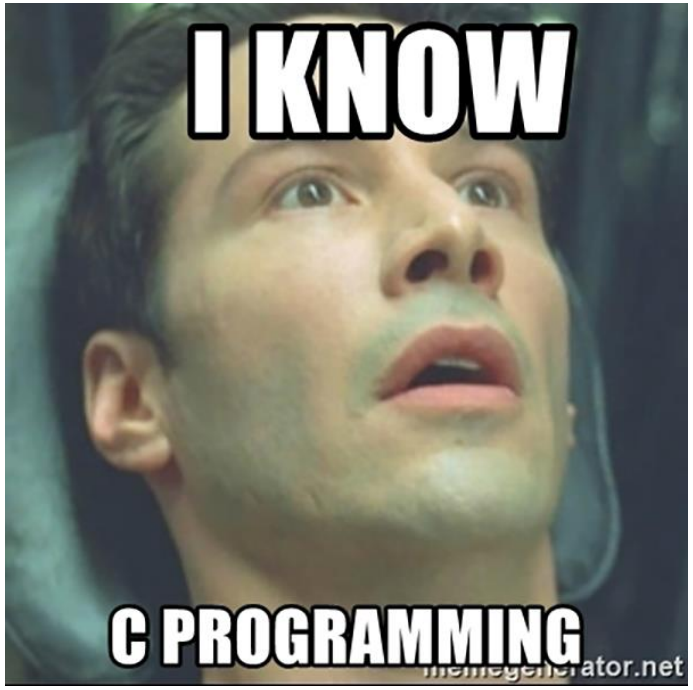
AND REVERSING IS EASY (BY DEFAULT)



F-Secure

WHAT SHOULD ATTACKERS USE?

If it ain't broke, don't fix it...
... use C or shellcode....or VB/PS



If you wanna be hipster - Go, Rust
(easy to dev, harder to reverse)

F-Secure

GENERAL DETECTION ADVICE

- The EDR broken record, install EDR it works!
- .NET telemetry is worth collecting, get an EDR that gives you this
- ILStub and MethodName events are magic
- .NET IL code is reverse-able! DnSpy is your friend

QUESTIONS?

F-Secure