Detecting Fileless Malicious Behaviour of .NET C2 Agents using ETW

Supervisors:  
Leandro Velasco  
Joao de Novais Marques

Course:  
Research Project 1

Authors:  
Alexander Bode  
Niels Warnars
Event Tracing for Windows

Enables logging kernel or application data, since Windows 2000

**Components of ETW**
- Providers
- Controllers
- Consumers

**Source:** Microsoft Docs, 2020
Fileless Malicious Behaviour of .NET C2 Agents

.NET assemblies can be dynamically loaded and executed into memory

- Using methods from the System.Reflection namespace
- Allowing remote execution of malicious code
Example

.NET code / executables are uploaded to bots and executed through the server by the botnet administrator

Source: Paisan Homhuan/123RF.com
Research Questions

Main Research Question

How can ETW be leveraged to detect fileless malicious behaviour of .NET agents used by popular C2 frameworks?

Sub Questions

What language-specific features can be used by .NET C2 agents for fileless attacks?

Which event types are relevant for detecting malicious .NET behaviour?
Importance

- Attackers shifting away from PowerShell to malicious .NET
- Logging and tracing support since Windows 2000
- Complexity and volume of data produced by ETW

Research Goals

- Find ways to detect .NET agents used by popular C2 frameworks using ETW
- Reduce false-positives and data volume
- Identify limitations of proposed detection methods
Current Research

Detection using ETW
- .NET code injection (F-Secure)
- Ransomware (CyberPoint)

Bypassing ETW
- For specific events, e.g., Asynchronous Procedure Calls (Tsukerman)
- Disable or delete ETW components (Palentir)
- ETW logs being renamed in the wild (Kaspersky)
Shortcomings

Detection using ETW

- Methods for detecting .NET code injection using ETW (F-Secure)
  - Inefficient research POC which uses the PyWintrace library
  - Relies on high-risk built-in function names
Methodology
Lab Setup

- Virtual Machine 1:
  - OS: Linux
  - Function: Command and Control server

- Virtual Machine 2:
  - OS: Windows 10
  - Function: Logging ETW events during code execution / loading agents
Investigated C2 frameworks

Tested four popular C2 frameworks documented by C2 Matrix project

- Generate .NET agents
- Load .NET assemblies into memory
Assembly loading in C2 frameworks

- Executing built-in assembly in Covenant C2

GruntTasking: 357c3e6d4d

<table>
<thead>
<tr>
<th>Name</th>
<th>Grunt</th>
<th>Task</th>
</tr>
</thead>
<tbody>
<tr>
<td>357c3e6d4d</td>
<td>1b0accf185</td>
<td>ProcessList</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Status</th>
<th>CommandTime</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Completed</td>
<td>2/1/20 4:56:42 PM</td>
<td>Assembly</td>
</tr>
</tbody>
</table>
Log Creation and Analysis

1. Determine relevant ETW providers and event names

2. Generate ETW logs:
   a. Malicious .NET agents
   b. Assembly loading POCs
   c. Benign .NET software

3. Compare event logs side-by-side
SilkETW

- Developed by Ruben Boonen of FireEye
- Logging utility for ETW
- Abstracts complexities
- Entries written to
  - JSON file
  - Windows Event logs
  - Logstash

Methodology
SilkETW is installed on hosts to control ETW sessions and providers.
SilkETW

SilkETW is installed on hosts to control ETW sessions and providers.

Methodology

Hello World / Create File

Assembly.Load*

POC's

Data □ JSON log file
SilkETW is installed on hosts to control ETW sessions and providers.
Example ETW Event (Simplified)

```json
{
    "ProviderName": "Microsoft-Windows-DotNETRuntime",
    "EventName": "Loader/AssemblyLoad",
    "TimeStamp": "2020-01-17T07:34:18.0794758-08:00",
    "ProcessName": "N/A",
    ...
    "XmlEventData": {
        "AssemblyFlags": "DomainNeutral|Native",
        "FullyQualifiedAssemblyName": "mscorlib, Version=4.0.0.0, Culture=neutral, PublicKeyToken=...",
        "EventName": "Loader/AssemblyLoad"
        ...
    }
}
```
Results
Assembly.Load

```csharp
var payload = Convert.FromBase64String("[PAYLOAD REMOVED]");  
var asm = Assembly.Load(payload);
asm.EntryPoint.Invoke(0, new object[] { new string[] { } });
```
ETW Filtering Steps

Start: Assembly loading POC + logging all .NET-runtime events

99.937 events
26 types of events

Manually clear away irrelevant and verbose event types (Unload, GC, Method/Load, etc.)

End result: Only subscribe to Loader events

9 events
3 types of events
Assembly loading seen from ETW (.NET 4.x)

1. Loader/AssemblyLoad
2. Loader/ModuleLoad
3. Loader/DomainModuleLoad

(* Optional if a module is loaded into an existing assembly)
Assembly loading seen from ETW (.NET 3.5)

1. CLRLoader/ModuleLoad
2. Loader/ModuleLoad

(* Both events contain same information)
Assembly loading seen from ETW

Assembly: Any executable or module, including:
- .NET application itself
- .NET libraries and dependencies
- Dynamically loaded components
## AssemblyLoad Event (.NET 4.x)

<table>
<thead>
<tr>
<th>Legit Module</th>
<th>Assembly name</th>
<th>AssemblyFlags</th>
<th>PublicKeyToken</th>
</tr>
</thead>
<tbody>
<tr>
<td>mscorlib.dll (as observed in Assembly.Load POC)</td>
<td>mscorlib</td>
<td>&quot;DomainNeutral</td>
<td>Native&quot;</td>
</tr>
<tr>
<td>mscorlib.dll (as observed in Covenant agent)</td>
<td>mscorlib</td>
<td>&quot;DomainNeutral&quot;</td>
<td>b77a5c561934e089</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>C2 framework</th>
<th>Assembly name</th>
<th>AssemblyFlags</th>
<th>PublicKeyToken</th>
</tr>
</thead>
<tbody>
<tr>
<td>Covenant</td>
<td>&quot;jhyfwkp2.hwm&quot;</td>
<td>&quot;0&quot;</td>
<td>null</td>
</tr>
<tr>
<td>PoshC2</td>
<td>&quot;Core&quot;</td>
<td>&quot;0&quot;</td>
<td>null</td>
</tr>
<tr>
<td>FactionC2</td>
<td>&quot;stdlib&quot;</td>
<td>&quot;0&quot;</td>
<td>null</td>
</tr>
<tr>
<td>SilentTrinity</td>
<td>&quot;Stage&quot;</td>
<td>&quot;Dynamic&quot;</td>
<td>null</td>
</tr>
</tbody>
</table>
## ModuleLoad Event (.NET 4.x)

<table>
<thead>
<tr>
<th>Legit Module</th>
<th>ModuleILPath</th>
<th>ModuleNativePath</th>
<th>ModuleFlags</th>
</tr>
</thead>
<tbody>
<tr>
<td>mscorlib.dll (as observed in Assembly.Load POC)</td>
<td>&quot;C:[...]\mscorlib.dll&quot;</td>
<td>&quot;C:[...]\mscorlib.ni.dll&quot;</td>
<td>&quot;DomainNeutral</td>
</tr>
<tr>
<td>mscorlib.dll (as observed in Covenant agent)</td>
<td>&quot;C:[...]\mscorlib.dll&quot;</td>
<td>&quot;&quot;</td>
<td>&quot;DomainNeutral</td>
</tr>
<tr>
<td>C2 framework</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Covenant</td>
<td>&quot;jhyfwkp2.hwm&quot;</td>
<td>&quot;&quot;</td>
<td>&quot;Manifest&quot;</td>
</tr>
<tr>
<td>PoshC2</td>
<td>&quot;Core&quot;</td>
<td>&quot;&quot;</td>
<td>&quot;Manifest&quot;</td>
</tr>
<tr>
<td>FactionC2</td>
<td>&quot;stdlib&quot;</td>
<td>&quot;&quot;</td>
<td>&quot;Manifest&quot;</td>
</tr>
<tr>
<td>SilentTrinity</td>
<td>&quot;Stage.exe&quot;</td>
<td>&quot;&quot;</td>
<td>&quot;Dynamic&quot;</td>
</tr>
</tbody>
</table>

**Results**
# ModuleLoad Event (.NET 3.5)

## Results

<table>
<thead>
<tr>
<th>Legit Module</th>
<th>ModuleILPath</th>
<th>ModuleNativePath</th>
<th>ModuleFlags</th>
</tr>
</thead>
<tbody>
<tr>
<td>mscorlib.dll (as observed in Assembly.Load POC)</td>
<td>&quot;C:[...]\mscorlib.dll&quot;</td>
<td>&quot;C:[...]\mscorlib.ni.dll&quot;</td>
<td>&quot;3&quot; (DomainNeutral</td>
</tr>
<tr>
<td>mscorlib.dll (as observed in Covenant agent)</td>
<td>&quot;C:[...]\mscorlib.dll&quot;</td>
<td>&quot;&quot;</td>
<td>&quot;1&quot; (DomainNeutral)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>C2 framework</th>
<th>ModuleILPath</th>
<th>ModuleNativePath</th>
<th>ModuleFlags</th>
</tr>
</thead>
<tbody>
<tr>
<td>Covenant</td>
<td>&quot;&quot;</td>
<td>&quot;&quot;</td>
<td>&quot;0&quot;</td>
</tr>
<tr>
<td>FactionC2</td>
<td>&quot;&quot;</td>
<td>&quot;&quot;</td>
<td>&quot;0&quot;</td>
</tr>
</tbody>
</table>
## ModuleLoad Signature

<table>
<thead>
<tr>
<th>Field</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>ModuleILPath</td>
<td>No absolute path (i.e. exclude slashes)</td>
</tr>
<tr>
<td>ModuleNativePath</td>
<td>Empty string</td>
</tr>
<tr>
<td>ModuleFlags (if present)</td>
<td>&quot;0&quot;, &quot;Dynamic&quot; or &quot;Manifest&quot;</td>
</tr>
</tbody>
</table>
ModuleLoad Signature - FP Testing

Tested against numerous .NET applications:

- Paint.NET
- KeePass
- Visual Studio

No false positives
Discussion
Limitations - General Considerations

- Assembly loading may occur for legitimate reasons
- Only performed limited false-positive testing
- Different .NET versions result in different event output
Conclusion
Conclusion

How can ETW be leveraged to detect fileless malicious behaviour of .NET agents used by C2 frameworks?

- Agents of multiple C2 frameworks dynamically load assemblies
- Detection possible based on ModuleLoad event
Future Work

- Investigate other use cases of ETW for endpoint monitoring
- Investigate real-world implementation of detection
Questions?
Backup slides
Limitations - ModuleLoad signature

- ModuleLoad signature relies on absence of full path

- Loading assembly file from disk results in absolute path logged in ModuleILPath
  - `Assembly.LoadFile(string path)`
  - `Assembly.LoadFrom(string assemblyName)`
Limitations - ModuleLoad signature

- ModuleLoad signature relies on absence of full path
- For dynamically loaded assembly, `ModuleILPath = assembly name`
- Bypass: Patch assembly name with fake path to get fake absolute path logged in ModuleILPath
## Documentation

<table>
<thead>
<tr>
<th>Event</th>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AssemblyLoad</td>
<td>AssemblyFlags</td>
<td>Type of assembly</td>
</tr>
<tr>
<td></td>
<td>PublicKeyToken</td>
<td>&quot;Last 8 bytes of the SHA-1 hash of the public key under which the application is signed.&quot;</td>
</tr>
<tr>
<td>ModuleLoad</td>
<td>ModuleILPath</td>
<td>&quot;Path of the Microsoft intermediate language (MSIL) image for the module, or dynamic module name if it is a dynamic assembly.&quot;</td>
</tr>
<tr>
<td></td>
<td>ModuleNativePath</td>
<td>&quot;Path of the module native image, if present&quot;</td>
</tr>
<tr>
<td></td>
<td>ModuleFlags</td>
<td>Type of module</td>
</tr>
</tbody>
</table>

Sources:
- [https://docs.microsoft.com/en-us/dotnet/api/system.applicationid.publickeytoken](https://docs.microsoft.com/en-us/dotnet/api/system.applicationid.publickeytoken)
Assembly.Load Variants

Assemblies can be loaded using:

- Assembly.Load
- Assembly.loadFile
- Assembly.LoadFrom
- Assembly.LoadModule
- Assembly.LoadWithPartialName
- Assembly.UnsafeloadFrom