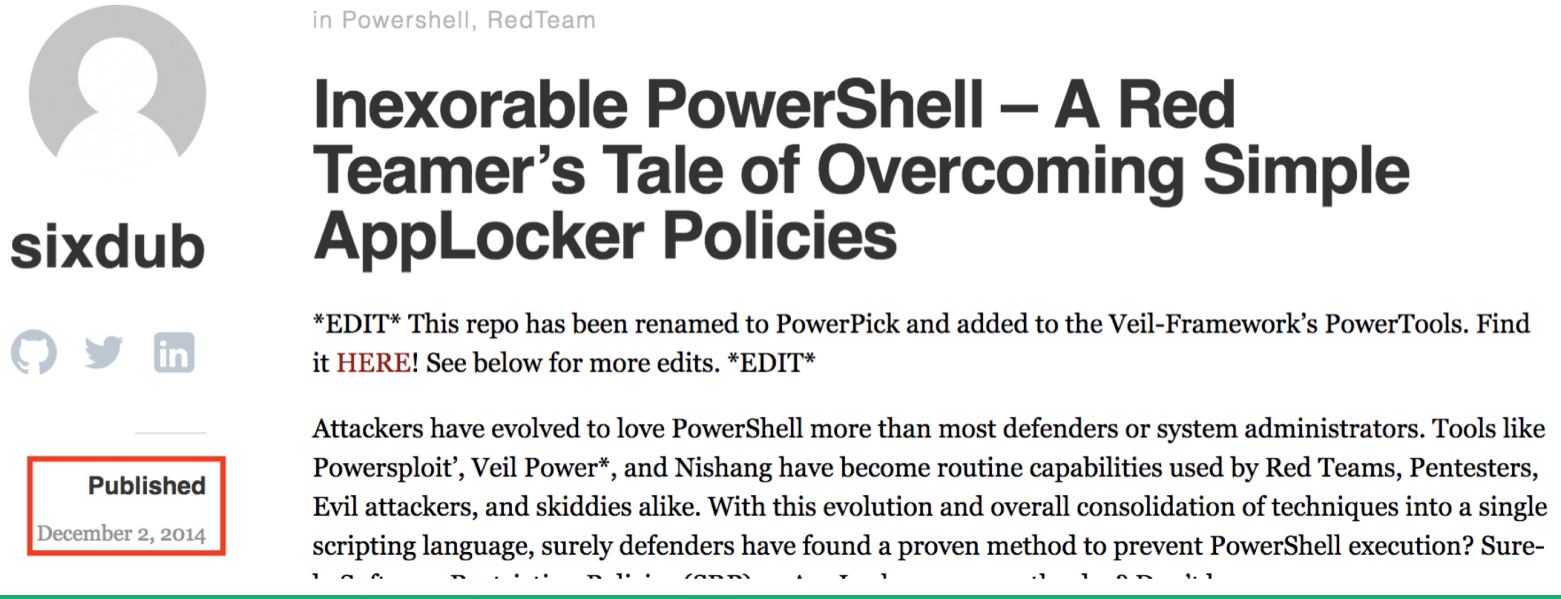
PowerShell Without powershell.exe

Who needs powershell(.exe) anyway?

Our Genesis

- Back in 2014, we realized that eventually powershell.exe would begin to be signatured, and we began investigating alternative ways to invoke our PowerShell code



sixdub

Published
December 2, 2014

in Powershell, RedTeam

Inexorable PowerShell – A Red Teamer's Tale of Overcoming Simple AppLocker Policies

EDIT This repo has been renamed to PowerPick and added to the Veil-Framework's PowerTools. Find it HERE! See below for more edits. *EDIT*

Attackers have evolved to love PowerShell more than most defenders or system administrators. Tools like Powersploit', Veil Power*, and Nishang have become routine capabilities used by Red Teams, Pentesters, Evil attackers, and skiddies alike. With this evolution and overall consolidation of techniques into a single scripting language, surely defenders have found a proven method to prevent PowerShell execution? Sure-

The Real Genesis?

PowerShell Pipeline Runners

- ***This is not a new idea!***

- Remember that PowerShell != powershell.exe
  - PowerShell == System.Management.Automation.(ni.)dll
- Following SharpPick/PowerPick, other offensive projects followed:
  - @jaredhaight's PSAttack project
  - @Cneelis's p0wnedShell
  - @ben0xa's NPS project

- Conceptually these utilize the same basic mechanism for PowerShell script invocation through C#

PowerShell Pipeline Runners

```
RunspaceConfiguration rsconfig = RunspaceConfiguration.Create();
Runspace runspace = RunspaceFactory.CreateRunspace(rsconfig);
runspace.Open();
RunspaceInvoke scriptInvoker = new RunspaceInvoke(runspace);
Pipeline pipeline = runspace.CreatePipeline();
String cmd = "Start-Process calc.exe";

pipeline.Commands.AddScript(cmd);
pipeline.Commands.Add("Out-String");
Collection<PSObject> results = pipeline.Invoke();
runspace.Close();
```

UnmanagedPowerShell

UnmanagedPowerShell

- **@tifkin_**'s response to the "can PowerShell run without powershell.exe" problem

- "UnmanagedPowerShell" (https://github.com/leechristensen/UnmanagedPowerShell) provides the ability to run PowerShell code in an unmanaged (C/C++/non-.NET) process
  - This is a different problem than running PowerShell in managed (.NET) code!

UnmanagedPowerShell: Process

1. Loads up the .NET Common Language Runtime (CLR) in the current process (needs code injection for a foreign process):
   a. .NET 4+ : CLRCreateInstance() to create a CLR instance, gets the runtime interface with .GetRuntime()/.GetInterface()
   b. .NET 2/3: CorBindToRuntime() (depreciated in 4)
2. Grabs a pointer to the CLR AppDomain with **.GetDefaultDomain()** and **.QueryInterface()**
3. Then loads up a custom C# assembly using **appDomain->Load_3()**
   a. This custom assembly is essentially just a “PowerShell runner”
4. The desired command or scriptblock is copied into the assembly and the execution method is called in the assembly

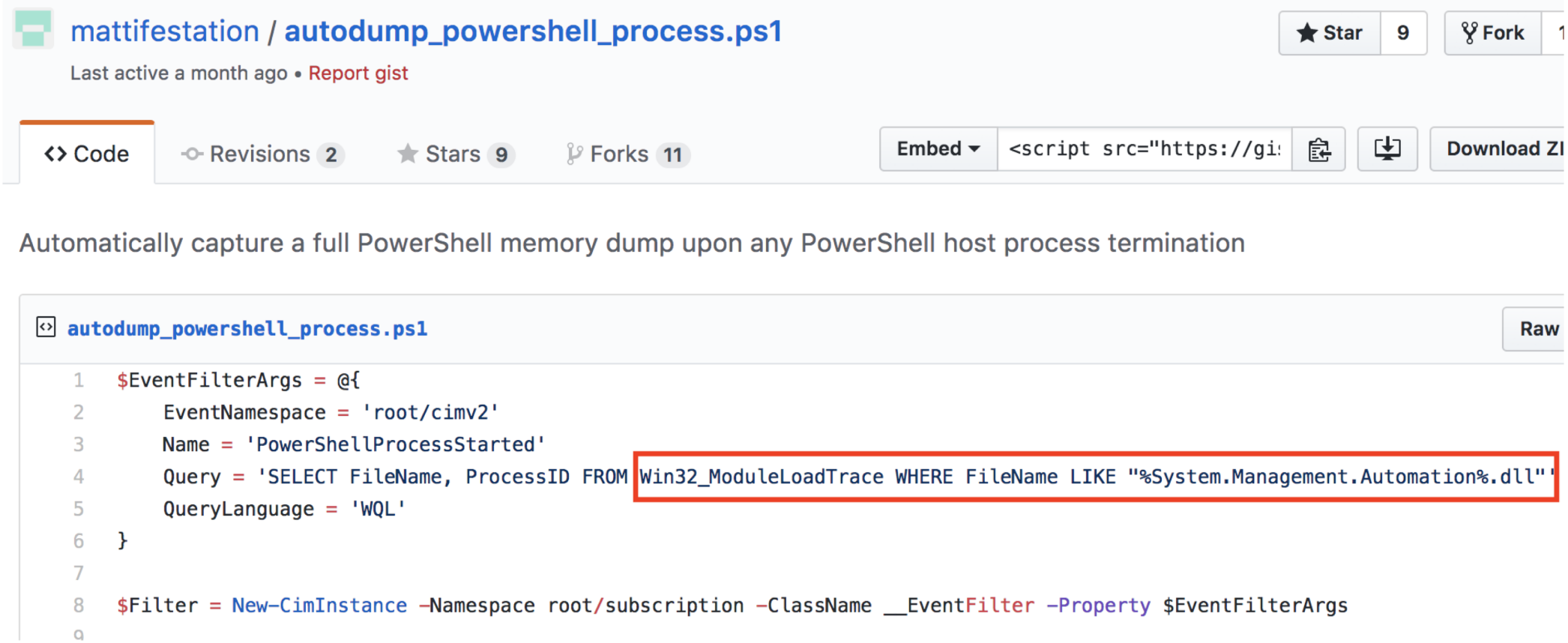
UnmanagedPowerShell: Weaponization

- **UnmanagedPowerShell** was what allowed us to build process injection for PowerShell Empire
  - @sixdub then took Lee's work and wrapped it up with Stephen Fewer's ReflectiveDLLInjection code
  - It has since been incorporated into Meterpreter and Cobalt Strike

- **ReflectivePick-** Reflective DLL that instantiates a PowerShell runspace (can be injected into another process)

- **PSInjector-** Script that uses ReflectivePick and automates injection

UnmanagedPowerShell: Defense



This would be a good time to

take a break and attempt

Lab: Building Your Own SharpPick