

Magdeburger Journal zur Sicherheitsforschung

Gegründet 2011 | ISSN: 2192-4260
Herausgegeben von Stefan Schumacher
Erschienen im Magdeburger Institut für Sicherheitsforschung

This article appears in the special edition »In Depth Security – Proceedings of the DeepSec Conferences«.
Edited by Stefan Schumacher and René Pfeiffer

Bypassing McAfee's Application Whitelisting for Critical Infrastructure Systems

René Freingruber

This paper describes the results of the research conducted by SEC Consult Vulnerability Lab on the security of McAfee Application Control. This product is an example of an application whitelisting solution which can be used to further harden critical systems such as server systems in SCADA environments or client systems with high security requirements like administrative workstations. Application whitelisting is a concept which works by whitelisting all installed software on a system and after that prevent the execution of not whitelisted software. This should prevent the execution of malware and therefore protect against advanced persistent threat (APT) attacks. McAfee Application Control is an example of such a software. It can be installed on any system, however, the main field of application is the protection of highly critical infrastructures. While the core feature of the product is application whitelisting, it also supports additional security features including write- and read-protection as well as different memory corruption protections.

Citation: Freingruber, R. (2016). Bypassing McAfee's Application Whitelisting for Critical Infrastructure Systems. *Magdeburger Journal zur Sicherheitsforschung*, *12*, 705–714. Retrieved September 6, 2016, from http://www.sicherheitsforschung-magdeburg.de/publikationen/journal.html

During the research the Windows version (version 6.1.3.353) of the product was checked against weaknesses and flaws in the design and implementation. Several methods were identified which can be used to bypass the main feature of McAfee Application Control to start execution of not whitelisted and therefore unauthorized code. During the audit different methods were developed for the most common attack vectors nowadays. In most cases the initial attack was prevented by the application, however, by only applying minimal changes it was possible to bypass the protections and infect the system. These scenarios consisted of different social engineering attacks and memory corruption exploitation. McAfee Application Control claims to implement protections against memory corruption attacks (e.g. buffer overflows). In fact, these protections only correspond to the typical operating system protections such as ASLR and DEP. Therefore exploits developed for newer systems run without any modification because they already include a DEP and ASLR bypass.

Additional design flaws and weaknesses were identified which can be used to bypass the read and write protection. Moreover several vulnerabilities exist in the kernel driver which can be abused to crash the system. Bearing in mind that the main field of application is the security of critical infrastructures (for example servers which regularly inspect the temperature of reactors from power plants) such an attack on the reliability can cause serious problems. On a final note, McAfee Application Control ships with very outdated components from 1999 that can be exploited as well.

1 Introduction

»McAfee Application Control software provides an effective way to block unauthorized applications and code on servers, corporate desktops, and fixed-function devices. This centrally managed whitelisting solution uses a dynamic trust model and innovative security features that thwart advanced persistent threats — without requiring signature updates or labor-intensive list management.« (1)

McAfee Application Control is a software which can be used to further harden operating systems by whitelisting applications. This is especially useful to protect critical infrastructures. Infrastructures were updates may not be installed because of certain reliability and availability requirements are another field of application. Examples for such requirements can often be found in SCADA environments were updates are not applied to avoid the risk of a flaw from an update-package. In theory the application should block not whitelisted executables and therefore prevent the execution of attacker supplied code. The following cite can be found on the product homepage:

»Minimize patching while protecting memory — Allows you to delay patch deployment until your regular patch cycle. In addition, it prevents whitelisted applications from being exploited via memory buffer overflow attacks on Windows 32- and 64-bit systems.« (1)

The aim of this paper is to describe the results of the research conducted to verify if the protections provided by McAfee Application Control stop or prevent attacks and how hard it is for an attacker to bypass them.

Section 2 describes various ways to bypass the whitelisting protection to achieve arbitrary code execution. It is split into three parts whereby the first describes techniques to retrieve so called »basic code execution« which means a basic form of code execution without having the ability to execute arbitrary code. The second part discusses how such a basic code execution can be turned into full code execution to accomplish the goal of a complete whitelisting bypass. This also includes a discussion on the security of the memory corruption protections provided by McAfee Application Control and how these can be bypassed by attackers. The third part explains concepts to bypass the UAC (user account control) feature of Microsoft and how it can be bypassed on systems running with McAfee Application Control.

Section 3 deals with the concept of write and read protection and how these protections can easily be bypassed as soon as code execution is achieved. Section 4 describes the identified kernel driver vulnerabilities as well as the impact of them. In section 5 some general minor weaknesses in McAfee Application Control are discussed. The last chapters gives a conclusion of the research.

2 Bypassing code execution protection

The main security feature of McAfee Application Control is the prevention of the execution of unauthorized code. Therefore, the first step is to demonstrate how this feature can be bypassed. In the following discussion the overall goal of getting full code execution is split into three parts.

The first step is to achieve basic code execution which means some sort of a very basic command execution. An example of basic code execution is the ability to start a whitelisted application with specific arguments.

The second step is to turn the basic code execution to full code execution which means that arbitrary code (shellcode) can be executed. At this stage code execution protections from McAfee Application Control are fully bypassed.

A final but not necessary step is to bypass user account control (UAC) to achieve code execution in the context of an administrative user. This is only possible if the attacked account owns administrative privileges, however, the other two steps can be done with a standard account.

2.1 Basic code execution

2.1.1 Abuse of unchecked file types: HTA and JS

McAfee Application Control prevents the execution of not whitelisted scripting-files with extensions such as .bat, .com, .vbs, and so on. However, a blacklist approach is used and it's very common that in such a blacklist approach file extensions or types are forgotten and therefore not verified. Exactly this was observed in McAfee Application Control.

In this special case HTML applications (HTA files) are not checked and can be started without restrictions. Using the run method from the Wscript.Shell object it's possible to start other whitelisted applications on the system which is required in a later stage of the attack. The malicious HTA file can be delivered to the victim for example via E-Mail or USB stick (simple social engineering attacks). If the victim opens the file the code starts to execute and can further infect the system.

Another unchecked file type was identified during the analysis. The execution of untrusted JScript-files is also not verified and can therefore be abused by attackers. To start another application the same code as discussed above can be used. In the most common attack scenario the ActiveXObject `Scripting.FileSystemObject` is used instead to write an executable (the virus) to the hard drive and after that Wscript.shell is used to start the executable. However, because of the application whitelisting this executable would not be authorized to run and therefore be blocked. The second part of this chapter discusses techniques which can be used to achieve full code execution. These techniques require the ability to start a whitelisted application with specific arguments. Exactly this can be done by using JS or HTA files. Moreover, it's possible to implement the complete malware inside the JS/HTA file.

It's very likely that other such unchecked file extensions exist which also allow a basic code execution.

2.1.2 File Shortcuts

McAfee Application Control does not prevent the execution of not whitelisted file shortcuts. Therefore an attacker can create a shortcut to a pre-installed and thus whitelisted executable including arguments and send the shortcut to the victim. If the victim opens the shortcut the whitelisted application starts and the specified arguments are passed to the executable. By specifying malicious arguments it's possible to abuse the whitelisted application to achieve code execution.

2.1.3 Malicious USB stick

If one of the above techniques is used the malicious file must somehow be sent to the victim. This can be done via different channels. For example via E-Mail, a network share or a USB stick. In the case of a USB stick a better approach exists. The USB protocol also supports other devices such as USB keyboards, USB mice or USB hubs. A malicious USB stick can be created which pretends itself as being a USB hub at which a USB keyboard and USB stick are connected. Then it can use the keyboard to send very fast specific keystrokes. For example the keystrokes for the Windows button and R key can be sent to open the run-dialog. After that an executable from the USB stick can be started. With this attack the required user interaction (starting a file) can be reduced and the victim only has to plug in the manipulated USB stick. The prevention of such an attack is not directly the task of McAfee Application Control, however, it is described here because it can be used as the first step to bypass application whitelisting.

2.1.4 Pass the hash

Another very common attack vector is that the attacker has already compromised systems within the internal network and then uses a pass-the-hash attack to further compromise other components. If for example all servers share the same administrator password their hashes are also the same and therefore the extracted password from one attacked system can be used to authenticate on another system. After that commands can be sent which are executed on the remote system. Metasploit includes a module for this attack, however, it's not working against systems with McAfee Application Control installed. The reason behind this is that the module first writes all commands into a bat file and after that executes it. Because the execution of not whitelisted .bat files is forbidden, this attack is successfully blocked. Though, this does not protect against the actual attack vector. By just modifying one line of code the module can be fixed to directly invoke the supplied command. Examples for malicious commands which can be used to achieve full code execution are presented in the next chapter.

2.2 Full code execution

2.2.1 Abuse of whitelisted applications: PowerShell

The typical use case is to solidify the system and therefore whitelist all existing applications to ensure the correct operability of the system. The problem with this approach is that applications which can possibly be abused by attackers are also whitelisted.

One example of such an application is PowerShell and its executable `powershell.exe` which is installed on all newer systems. As soon as the system is solidified this application is also whitelisted per default and therefore can be executed by an attacker. McAfee Application Control checks if an attacker tries to start a PowerShell script (a file with .ps1 extension) and only allows the execution of whitelisted scripts. However, it's still possible to abuse `powershell.exe` by specifying the complete script inside the arguments. For

example, it's possible to start calc.exe by using the following command:

```
powershell.exe -nop -windows hidden -noni \n
               -command calc.exe
```

Another method is to use the encodedCommand argument. Using this argument complex scripts can be started. The following PowerShell script encodes a command:

```
$cmd = 'calc.exe'
$bytes = [System.Text.Encoding]:: \n
           Unicode.GetBytes($cmd)
[Convert]::ToBase64String($bytes)
```

The base64 encoded output can be used to invoke calc.exe from PowerShell. The following command is an example of an encoded command which starts calc.exe:

```
powershell.exe -enc YwBhAGwAYwAuAGUAeABlAA==
```

Starting only already whitelisted applications via PowerShell would not be very powerful, however, PowerShell can do more. It's possible to access and invoke any function exposed by a Windows library like CreateThread() or VirtualAlloc(). Combining both functions it's possible to allocate additional memory to store shellcode in the current process and start a new thread to execute it.

To start PowerShell with the malicious arguments one of the above mentioned techniques (see the »basic code execution« chapter) can be used. The selection of an applicable technique depends on the specific attack vector, system configuration and actual scenario.

Here PowerShell was used as an example of a default whitelisted application which can be abused by an attacker. It's very likely that other whitelisted programs exist which can be abused the same way.

Examples are debuggers because they can be used to write shellcode into the process space of a whitelisted application to start the code in the context of that process. Other examples are script-interpreters like Python or Perl because they can be used to start the execution of shellcode. In both cases it's possible to specify the complete script inside the arguments as done with PowerShell. Python is using the –c argument and Perl the –e argument to execute a script specified in the arguments.

Please note that it's very likely that more such whitelisted applications exist which can be abused by an attacker.

2.2.2 Java applets

One of the most common attack vectors nowadays is to embed a malicious Java applet on a website and trick te victim to start it. Tests showed that McAfee Application Control does not prevent the execution of not whitelisted Java applets. Therefore it's possible to achieve code execution by abusing Java applets if the Java runtime is installed and whitelisted on the target system. However, it's important to note that McAfee Application Control protects against many malicious applets in the wild because of their attack nature. Typically the attack code just extracts a file from the resources to the file system and after that executes it. Because of the application whitelisting the execution of such a newly created executable is prevented. Multiple approaches exist to circumvent this. For example it's possible to start PowerShell and pass the shellcode as argument as already described. Even if PowerShell would be removed from the list of default whitelisted applications an attacker can execute the shellcode directly inside the Java applet. More details on such an approach can be found at (13) and (14).

2.2.3 Office macros

Office macros are an old technique used in social engineering attacks to infect a system. If the victim enables macros in a malicious word or excel file VBA code starts to execute. McAfee Application Control does not protect against this, however, typically the code just drops a binary to the file system and then starts it. Msfencode is an example for this where just a binary is dropped to the file system and started afterwards. Because the binary was not whitelisted its execution would be blocked. To overcome this, one of the whitelisted applications, e.g. PowerShell, can be started instead. Moreover it's possible to directly execute the shellcode inside the Office application. Source code for such an attack can be found at (15), however this code only works on 32-bit systems. On a 64-bit Windows systems the declaration of the functions must be adapted by using the PtrSafe attribute and all occurrences of the type Long must be replaced with the type LongPtr; see (16) for more information.

2.2.4 Memory corruption exploitation

Even if all the above issues would not exist an attacker can still exploit a memory corruption vulnerability in one of the whitelisted applications (e.g. in the browser, one of the office applications, PDF reader, and so on). In such a case shellcode can be executed on behalf of the attacked process.

McAfee Application Control tries to prevent against such attacks by implementing buffer-overflow protections. The following cites are taken from the product homepage:

»In addition, it prevents whitelisted applications from being exploited via memory buffer overflow attacks on Windows 32- and 64-bit systems.« (1)

»Key Advantages: Protect against zero-day and APTs without signature updates.« (2)

»Whitelisted programs that might contain some inherent vulnerabilities cannot be exploited through a buffer overflow.« (3)

Exploitation on Windows XP systems

To verify the above statements multiple exploits were tested on different Windows systems running McAfee Application Control. After solidifying and rebooting

the system the features `mp` (memory protection) and `mp-casp` are enabled per default, `mp-vasr` is disabled on Windows XP, on Windows 7 it's enabled but more on this later. Even though Windows XP is out of support and it should not be used anymore the additional security provided by McAfee Application Control was checked. This was done because of two reasons. First, Windows XP does not include memory protections which are nowadays shipped per default with newer operating systems and therefore the protections from McAfee Application Control can better be studied. The second reason was that Windows XP systems are nowadays still in use in SCADA environments and therefore the security of such systems is still important.

Out of 30 tested vulnerabilities from different applications McAfee Application Control blocked 28 of them. Only a Firefox exploit (.reduceRight() vulnerability – CVE-2011-2371) and a VLC exploit (.s3m vulnerability – CVE-2011-1574) were able to bypass the protections without any modifications. These two exploits were developed to target newer systems and therefore contained a ROP (return oriented programming) chain to disable DEP (data execution prevention) whereas the others didn't contain a ROP chain.

The memory protections from McAfee Application Control work by injection a DLL into all running processes. As soon as a process is debugged with McAfee Application Control enabled several access violations occur inside a debugger. This is due to the fact that McAfee Application Control modified the memory protection options from the kernel32.dll DOS/PE header to be not readable. As soon as an instruction attempts to read from the PE header an access violation is triggered and code allocated by scinject.dll (on behalf of ntdll.dll) handles the access violation.

It first loads the address of various functions and then calls ZwQueryVirtualMemory to check if the triggering instruction belongs to a page marked as executable. If the check passes it calls scinject.casp_inject_save_addr which copies the triggering instruction to a new allocated memory, then calls ZwContinue to continue execution of the instruction and after that jumps back into scinject.dll to 0x66441120. Before executing the instruction the function located at 0x7c9B0060 is used to make the PE header readable and the function located at 0x664410f0 removes the read protection afterwards

With this approach McAfee Application Control implements a kind of »software DEP«. Since the OS/architecture doesn't support DEP it has to implement the checks inside the software. This is achieved by forcing an exception during shellcode execution. Typically shellcode has to parse the PE header of kernel32.dll to retrieve the address of different functions and therefore this method can be used to pause execution of the shellcode to apply additional checks. In this case only a simple check is done by verifying if the associated page was marked as executable.

To circumvent the protection two different methods can be used. Either the PE header can again be marked as readable or the shellcode can be marked as executable. In both situations a function such as VirtualProtect() or VirtualAlloc() must be called. Since these functions belong to kernel32.dll their addresses can't be obtained without triggering the checks. To deal with this problem the deeper function ZwProtectVirtualMemory from ntdll.dll can be used instead.

In the above setup three important observations can be made:

1. The memory allocated by scinject.dll is marked as RWX and because ASLR is not supported by Windows XP the memory is always located at 0x7C9B0000. This means that the shellcode can copy itself to this location to bypass the protection (e.g. 0x7C9B0750 is not used by the code and therefore a good target).
2. Inside the allocated location a pointer to ZwProtectVirtualMemory is stored at 0x7C9B0695. Shellcode can read the dword stored at that location to obtain a pointer to ZwProtectVirtualMemory and call it to mark its own location as executable.
3. The functions stored at 0x66441120 and 0x7c9B0060 can be called to set protection options for any page. Therefore, they can be called to mark the shellcode as executable or the PE header as readable.

However, all these methods presuppose that McAfee Application Control is installed on the target system. A better approach is to obtain the address of ZwProtectVirtualMemory by searching inside the list of loaded modules and parsing the export table to find it. This also works reliably on systems which don't have McAfee Application Control installed and therefore ensures maximum exploitation reliability. Shellcode for that method was developed and by using it, it was possible to bypass the memory protections from McAfee Application control in all 30 exploits.

Exploitation on Windows 7 systems

Since Windows 7 the »mp-vasr« and »mp-vasr-forced-relocation« features are in addition enabled per default. These mitigation techniques correspond to the mitigation techniques ASLR and forced ASLR from the operating system. Therefore the overall security is not increased because up-to-date systems already include both protections (forced ASLR since update KB 2639308 on Windows 7, see (5)).

The injected scinject.dll library now supports common protections such as ASLR, DEP and SafeSEH. However, when scinejct.dll allocates memory for additional code it marks these locations as read-, write- and executable as already mentioned in the Windows XP part.

These write- and executable sections can be abused by attackers in various ways to bypass DEP, however, in most cases it doesn't make a big difference. This is due to the fact that the base location of these regions

is randomized by ASLR. An attacker first has to somehow bypass ASLR and as soon as ASLR is bypassed it doesn't matter if a ROP chain must be developed to disable DEP (e.g. a call to VirtualProtect(), VirtualAlloc(), and so on) or to write the shellcode to one of the RWX locations. Only in some special situations this section can play into the hands of an attacker. One such example would be if additional security applications are in place such as EMET. EMET protects calls to VirtualProtect() or VirtualAlloc() to detect ongoing attacks. In such a case an attacker can bypass all protections of EMET by abusing the RWX locations instead of calling one of that protected functions.

Exploitation on Windows 8.1 x64 systems

On Windows 8.1 x64 systems the memory protection features disappeared in McAfee Application Control. This is most likely because the operating system itself already implements all these protections and therefore it's not required anymore to inject a library into all processes.

To summarize, the overall security from the perspective of an exploit developer on a system with McAfee Application Control installed is quite the same as the security of a standard operating system without McAfee Application Control installed. The security can even be lower in some special situations if McAfee Application Control is installed when it makes sense to abuse the RWX-allocated memory region as mentioned earlier. Only if the underlying operating system is unpatched and therefore does not contain support for forced ASLR the security is higher because McAfee Application Control implements this feature.

During the research it was possible to exploit various memory corruption vulnerabilities on a Windows XP, Windows 7 and Windows 8.1 system with McAfee Application Control installed. Most of the exploits were developed a long time before auditing McAfee Application Control and successfully executed without any modification.

Exploitation of installed ZIP application

In the previous chapters techniques to exploit memory corruption vulnerabilities on systems with McAfee Application Control were described. However, before bypassing these mitigations such a memory corruption vulnerability must be found. Identifying such a flaw can be considered as being not too difficult because the application is often used in environments where critical patches are missing (e.g. SCADA environments because of reliability requirements). This shows why it's so important to apply updates just in time. Another fact to consider is that local memory corruption vulnerabilities are often not fixed. This is due to the fact that exploitation of such a vulnerability typically brings no additional advantage for an attacker because it can't lead to a privilege escalation. Only code in the context of another application can be executed, however, the same goal can be achieved by using a code injection technique. Therefore such vulnerabilities are often marked as unimportant and not get fixed. These vulnerabilities are very useful if application whitelisting should be bypassed because they are easy to find and they can be used to achieve code execution.

Finding such a vulnerability heavily depends on the target system and exploitation is therefore very specific. To overcome this problem a generic approach was developed. For a generic method the attacked executable must be present on all systems. This restricts the selection to application specific executables. Therefore the installation folder of McAfee Application Control was searched for other executables. A ZIP-application from 1999 was found to be installed per default by McAfee Application Control.

Because the executable was compiled in 2001 it doesn't support typical security features such as DEP, ASLR or SafeSEH. Finding a vulnerability in it is quite trivial, a simple call such as the following leads to a buffer overflow:

```
1 zip.exe -r a.zip
2 aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa \n
3 aaaaaaaaaaaaaaaaaaaaaa*reduced*
```

2.3 Bypassing User Account Control (UAC)

Starting with Windows Vista Microsoft introduced the concept of User-Account-Control (UAC). The idea behind this technology is to limit the privileges of applications even if they are started by an administrative user. If an administrator logs into the system two different privilege tokens are created – one for a normal user and one for an administrative user. Typically applications are started using the normal user token. If an application requires addition privileges from the administrative token a special prompt must be confirmed by the end-user.

Some techniques from the next chapter require administrative privileges because code should be injected into privileged services. To apply these techniques, UAC must somehow be bypassed. Therefore, it's important to first discuss how UAC can be bypassed on systems with McAfee Application Control installed. Please note that the technique to bypass the write protection also works with accounts with standard privileges. Only some special cases require administrative privileges, e.g. injecting code into a service.

The typical and most commonly used UAC bypass (which is for example used by Meterpreter) is not working because McAfee Application Control prevents loading of not whitelisted libraries. The following text gives a very short overview about different UAC bypass methods. This text doesn't give a complete overview nor are all details discussed in depth. Instead, the basic idea behind the techniques is described to discuss why certain methods are or are not working with McAfee Application Control enabled.

What's commonly shared with many UAC-bypasses is the fact that auto-elevated processes are abused.

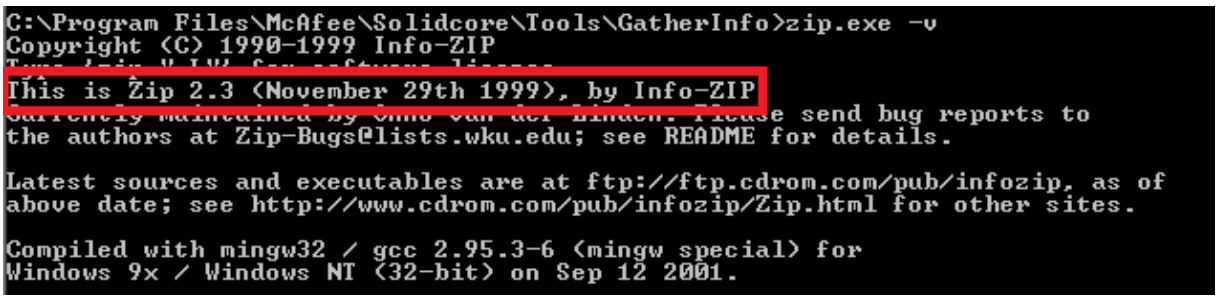


Figure 1: ZIP application from 1999 gets installed on all systems

Microsoft gave a set of programs special privileges to allow them to auto-elevate UAC requests. This was done to minimize the number of displayed UAC prompts to the end-user. If an attacker manages to execute code in the context of such an auto-elevated process UAC is bypassed. However, it's not possible to directly inject code into such a process because of the missing administrative privileges. Microsoft gave a second set of applications special permissions. These applications (e.g. explorer.exe, notepad.exe, calc.exe, and so on) can interact with auto elevated COM objects to do specific actions like creating a new directory in a protected folder without prompting an UAC dialog. Since these applications run with the same privileges as the attacker supplied code it's possible to inject code into their process space using the standard technique with VirtualAllocEx, WriteProcessMemory and CreateRemoteThread. After that it's possible to create files and directories inside protected folders (e.g. `C:\Windows\system32`) by abusing the COM objects.

To completely bypass UAC a final step is required which injects code into one of the auto-elevated processes. To do this, a simple DLL injection technique is used. If an executable imports a library, a specific search path is used to find it (see (6) for more information). Code can be injected into such an auto elevated process by placing a malicious library very early into the search path to force its load instead of the valid one. This is typically not possible because the search path consists of protected folders. However, by applying the above discussed technique files can be written to such locations.

More information on such bypasses can be found at (7) and (8). Especially (8) is a great resource and source code for many of the next discussed techniques can be found there. Over time, more methods were developed, however, the basic underlying technique is in most cases still the same. E.g. it's possible to abuse the WUSA (Windows Update Standalone Installer) application to extract compressed files to a protected directory instead of injecting code to notepad.exe or explorer.exe. The attacked auto-elevated executable also changed with different methods (e.g. abusing cliconfg.exe instead of sysprep.exe), but all these techniques share the concept that a newly created library is forced to be loaded into the process space of an auto-elevated process.

Exactly this is prevented by McAfee Application Control because the newly created library is not whitelisted on the system and therefore it can't be loaded. However, other techniques exist to bypass UAC. One example is the AppCompact / Shim redirection technique. The following text gives a very quick overview about the technique.

Microsoft introduced the concept of shims with the Application Compatible Toolkit. The idea behind this feature is to provide the ability to make old applications compatible with newer operating systems. For example on Windows XP documents are stored at `C:\documents` whereas on Windows 7 documents are stored at `C:\Users\<username>\documents`. To make old applications compatible with this new file structure (and other changes) it's possible to install a shim. This shim sits between the application and the libraries by hooking the IAT (import address table) and modifies parameters to fit to the new structure. It's not only possible to change paths, it's also possible to inject libraries or redirect execution to another executable (more information on shims can be found at (9) and (10)). For example, a shim can be created with a redirectEXE rule which redirects cmd.exe to calc.exe. As soon as a user tries to open cmd.exe the calculator pops up instead. The same technique can be used to redirect one of the auto-elevated executables to another executable (e.g. PowerShell.exe). Because the manifest (including the auto-elevate permissions) is used from the original file the newly spawned process has administrative privileges because UAC prompts are auto-elevated. Since this approach does not require loading newly created files, it also works on systems with McAfee Application Control enabled. This bypass is only applicable on x86 systems because 64-bit Windows systems do not support the redirectEXE rule.

Another method which works on 64-bit and 32-bit Windows systems is to permanently disable UAC. The downside of this approach is that it requires a reboot of the system. It is called simba (8) and abuses the undocumented ISecurityEditor COM object. This COM object can be accessed from explorer.exe and can be used to set the access permissions for a registry key. The attack works by injecting code into explorer.exe which then uses the ISecurityEditor COM object to make the following registry path writeable:

`MACHINE\\SOFTWARE\\Microsoft\\Windows\ \CurrentVersion\\policies\\system`

After that, keys can be created and modified to permanently disable UAC.

Many other UAC bypass techniques exist which are also very likely to work on McAfee Application Control enabled systems (e.g. see (11) and (12)). Because the above discussed techniques were already working no further effort was spent in finding further methods.

3 3. Bypassing write and read protection

3.1 Code injection into update-processes

A main feature of McAfee Application Control is to prevent write actions on whitelisted files. This is significant since an attacker could simply overwrite the content of a whitelisted application to execute his own code.

McAfee Application Control is designed to store the name and path of all whitelisted applications in a database. As soon as a file is started (or a library loaded) the path and filename are checked against the whitelisted database entries. Using this approach additional overhead (e.g. calculation of the hash of an executable) is not required and therefore the system performance is not affected too heavily. Please note that the documentation mentions that hashes are also stored in the database (which is stored encrypted at `<drive>\Solidcore\scinv`). During the tests the hashing mechanism could not be verified, however, it's still possible that hashing occurs but this is done in a transparent way to the end-user and therefore also to the attacker. This means that an attacker can just overwrite the content of a file by using the following technique without worrying about possibly incorrect hash sums.

The described approach of storing the absolute path together with the write protection has a second »benefit«. Often applications include an update-process which ensures that the latest version of the application is installed. If this updater downloads and replaces the main executable the application would not be able to start anymore because hashes changed. A system administrator would have to whitelist the application again. A better approach is to automate the process of re-whitelisting updated executables to lower the maintenance work of administrators and end-users.

To implement this, McAfee Application Control allows to specify a list of special update-processes. These processes are identified based only on their names (but must be solidified and marked as executable). They are allowed to overwrite any file on the system and therefore can be used to completely bypass the write protection offered by McAfee Application Control.

From the perspective of an attacker such a bypass is not required but it makes things a lot easier. For example, as described in chapter 2, it's possible to achieve code execution. However, using the discussed approach only shellcode can be executed. With this shellcode it's possible to do everything which an attacker wants but writing such shellcode is quite time consuming. Typically an attacker already has a bunch of tools for various tasks (e.g. dumping information from the system, stealing credentials, starting a key-logger, further compromise the internal network and so on). Because of the application whitelisting these tools can't be started but as soon as the write protection is bypassed it's possible to overwrite a whitelisted application with the content of the required tool.

A list of pre-configured default updaters can be dumped using the »sadmin updaters list« command. They mainly consist of different updaters for McAfee products, Apache, Apple, Adobe Flash player, Oracle Java, Mozilla Firefox, any many more.

The last missing step is to force one of that update processes to overwrite the content of a whitelisted application. To achieve this code in the context of an update-process must be controlled. This can be done by using one of the many well-known code injection techniques. The simplest approach is to open a handle to the process and then allocate memory for that process by using the VirtualAllocEx() function. Next, shellcode which overwrites a whitelisted file (e.g. simple CopyFileA-shellcode) can be copied to the newly allocated memory space of the target process by using the WriteProcessMemory() function. Finally, the shellcode can be executed on behalf of the update-process by starting a new thread using the CreateRemoteThread() function.

The above described attack can be hot-fixed by removing all update-rules and therefore an attacker can't migrate to an update-process. Another problem would be if no default updater is installed on the system. Since the default rules contain »spoolsv.exe» there is always a standard-process which is running with updater-privileges. However, »spoolsv.exe« is running as SYSTEM and therefore UAC must be bypassed to inject code into that process. Different techniques to achieve this were already described in chapter 2.3.

Updaters also have a second capability. The command `sadmin read-protect -i C:\secret.txt` can be used to make the file secret.txt read-protected. However, updaters can still read such protected files and therefore read-protection can easily be bypassed as well.

3.2 Code injection into scsrvc.exe

Another injection target is »scsrvc.exe« which is the main service of McAfee Application Control. Injecting code into this process has some special advantages, however, since this process runs as SYSTEM, UAC must first be bypassed.

After injecting code into the `scsrvc.exe` process the password file can be read. This file is stored at `C:\ProgramFiles\McAfee\Solidcore\passwd` and is read-protected. Reading this file is even forbidden for update-processes. However, »scsrvc.exe« is allowed to read the file.

Moreover, `scsrvc.exe` has the permission to remove the file. As soon as the file is removed, it's possible to use the `sadmin.exe` command without supplying a password and therefore all rules can be modified or disabled.

It's also possible to modify rules directly by changing registry values from the scsrvc.exe process. The configuration is stored in the following path:

An attacker can for example change the »TrustedVolumeRules« to add a directory from which any executable can be started. This can be used to completely bypass write and code execution protection in an easy way.

4 Kernel driver vulnerabilities

Several kernel vulnerabilities were identified during a simple fuzzing test. McAfee Application Control loads the kernel driver `C:\Windows\system32\drivers\swin1.sys` which has several weaknesses when handling one of the following IOCTL-codes:

- 0xb37031f0
- 0xb37031f8
- 0xb37031fc
- 0xb370320c
- 0xb3703200
- 0xb3703204
- 0xb3703208
- 0xb3703214

These vulnerabilities can be triggered from user land and can be used to crash the system with a bluescreen.

5 Conclusion

As shown, it's easily possible to bypass the protections provided by McAfee Application Control. In most cases the application successfully prevented an ongoing attack, however, by applying only minimal changes it was always possible to bypass the checks. Several methods were presented which can be used to achieve full code execution. Techniques to bypass read and write protection and how UAC can be bypassed to apply them were also discussed. By combining all these techniques (first gain basic code execution, then escalate to full code execution, after that bypass UAC and then inject code into the main service to create a trusted volume) it's possible to start any not whitelisted application. Therefore, it's possible for an advanced attacker to bypass the implemented protections. While McAfee Application Control helps to prevent attacks from the mass, additional security in the case of targeted attacks that especially occur in SCADA environments, can't be expected.

Moreover, several weaknesses were identified. These range from allocating RWX regions in the memory space of all protected applications over installing applications from 1999 to critical kernel vulnerabilities which can be used to completely crash the system. Because of the typical area of application of McAfee Application Control such an attack on the reliability of a system can cause significant problems. In most cases such a tool is only used if the security of a critical system should be increased. Exactly in such situations the reliability of the system is very important and should therefore be secured. By installing McAfee Application Control the user is tricked in weighting in deceptive additional security but in reality the application tears holes in the overall security of the system.

Out of our experience we at SEC Consult consider it necessary for critical infrastructures to regularly install new updates, use only software reviewed by security professionals and further increase the awareness of end users with security trainings. For such systems it's not enough to solely rely on a security layer such as application whitelisting. Rather, the underlying security of the system itself must be increased. We do not see a reason for not using application whitelisting if the software is secure and doesn't tear holes in the overall system security but it's important to understand that it doesn't replace robust security measures.

6 References

1. https://github.com/trustedsec/social-engineer-toolkit
2. http://support.microsoft.com/en-us/kb/2639308
3. https://msdn.microsoft.com/en-us/library/windows/desktop/ms682586%28v=vs.85%29.aspx
4. http://www.pretentiousname.com/misc/win7_uac_whitelist2.html
5. https://github.com/hfiref0x/UACME
6. http://blogs.technet.com/b/askperf/archive/2011/06/17/demystifying-shims-or-using-the-app-compat-toolkit-to-make-your-old-stuff-work-with-your-new-stuff.aspx
7. http://www.alex-ionescu.com/?p=39
8. https://code.google.com/p/google-security-research/issues/detail?id=118
9. https://code.google.com/p/google-security-research/issues/detail?id=222
10. https://github.com/schierlm/JavaPayload
11. BSidesCHS2013Session02, JavaShellcodeExecution,RyanWincey

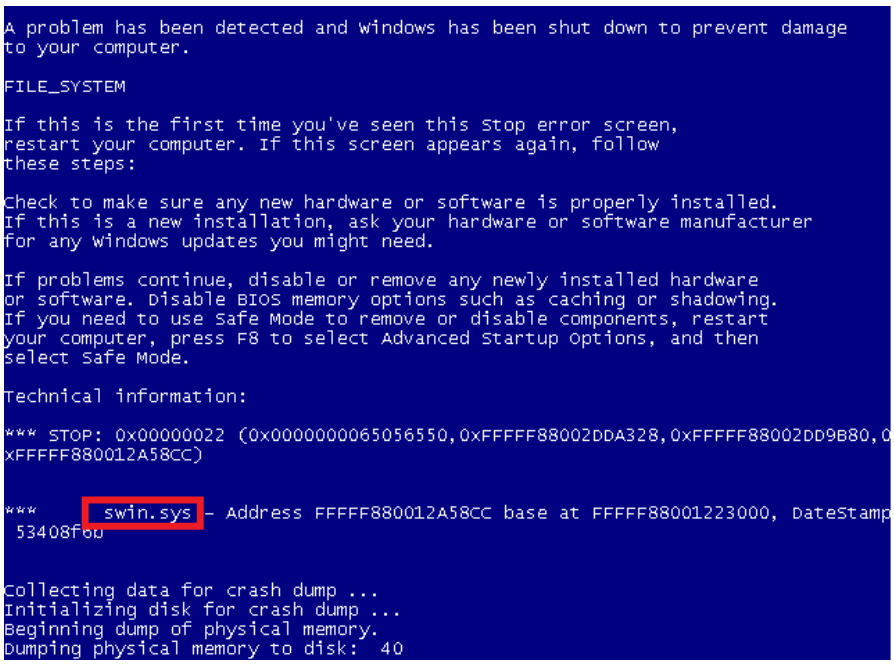


Figure 2: Flaw in swin.sys leads to a bluescreen

12. http://blog.didierstevens.com/2009/05/06/shellcode-2-vbscript/
13. https://msdn.microsoft.com/en-us/library/office/ee691831%28v=office.14%29.aspx

7 About the Author

René Freingruber has been working as a professional security consultant for SEC Consult for several years. He operates research in the fields of malware analysis, reverse engineering and exploit development. During his bachelor thesis he developed hundreds of exploits to study different mitigation techniques implemented by modern operating systems and how they can be bypassed by attackers. With topics such as bypassing Microsoft's EMET toolkit he has already spoken at various big security conferences including RuxCon, ToorCon, ZeroNights, DeepSec, NorthSec, IT-Secx and 31C3.

About the Vulnerability Lab

Members of the SEC Consult Vulnerability Lab perform security research in various topics of technical information security. Projects include vulnerability research and the development of cutting edge security tools and methodologies, and are supported by partners like the Technical University of Vienna. The lab has published security vulnerabilities in many high-profile software products, and selected work has been presented at top security conferences like Blackhat and DeepSec.

For more information, see `http://www.sec-consult.com/`