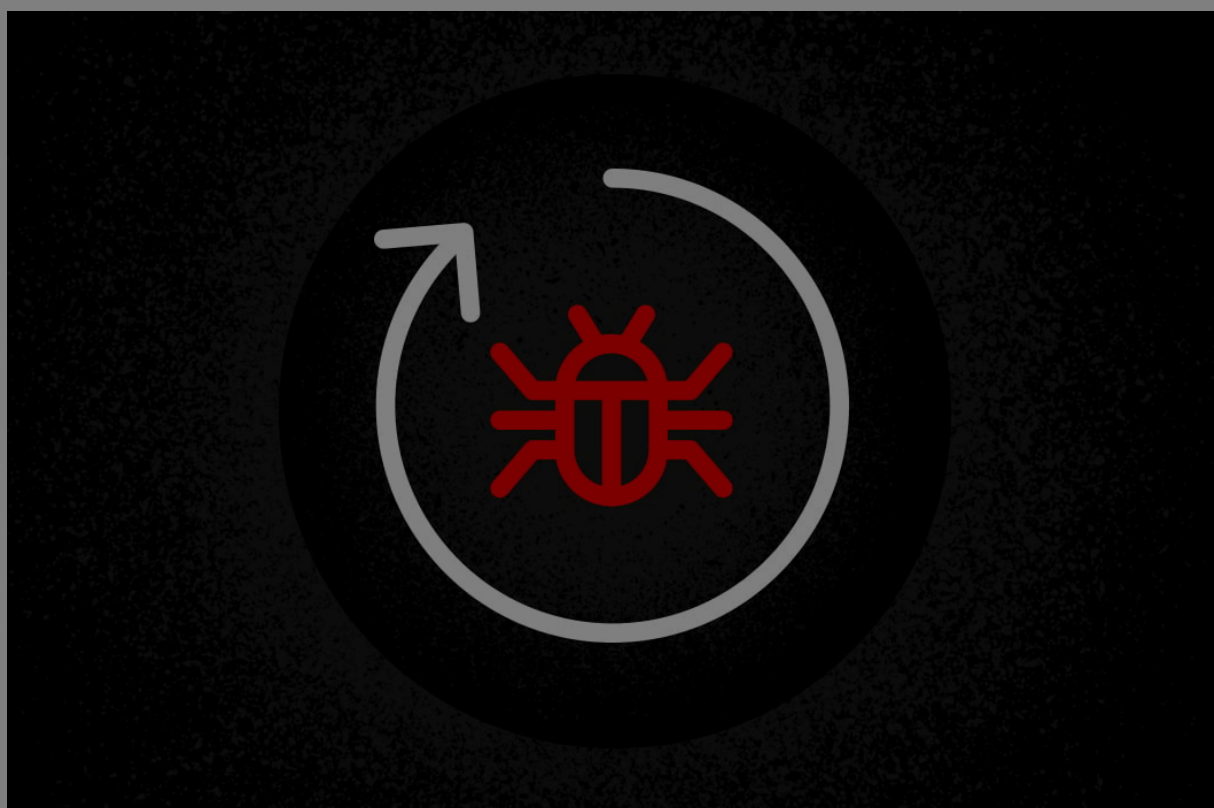


The Windows Restart Manager: How It Works and How It Can Be Hijacked, Part 1














Malware utilizes a multitude of techniques to avoid detection, and threat actors are continuously uncovering and exploiting new methods of attack. One of the less common techniques includes the exploitation of the Windows Restart Manager. To stay ahead of malicious authors, it is important to be aware of them and understand how they work.

The Restart Manager is a library for reducing required reboots during software updates. Files that may be impacted during an update can be locked by various applications, preventing the update's process from modifying them. This can result in a reboot that forces applications to release the locks they have on files targeted by the updater. As an alternative, the Restart Manager enables processes to release the locks on targeted files by terminating processes that are using them if required conditions are met. However, this mechanism can be hijacked to serve malicious purposes.

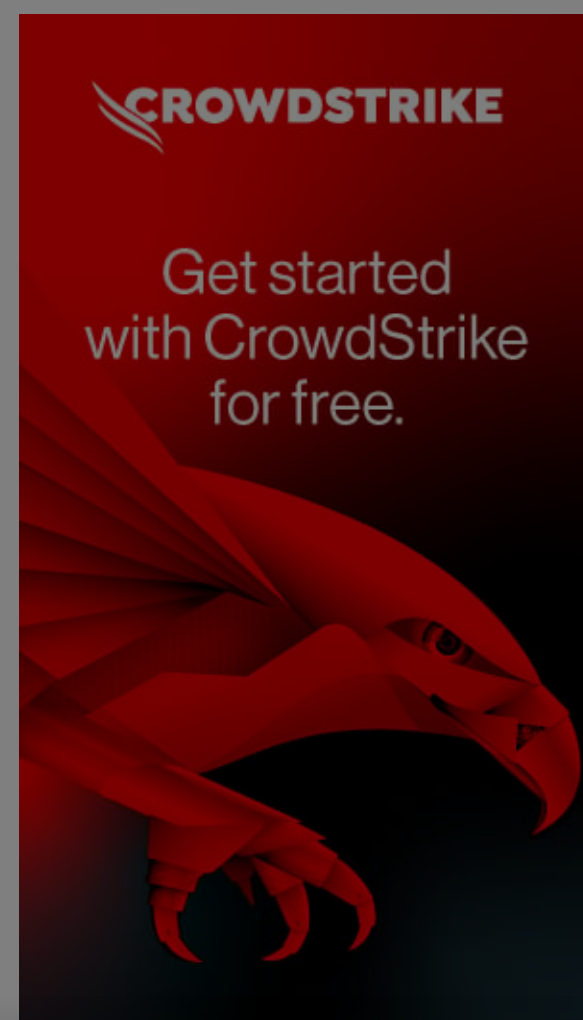
In this two-part series, we:

- Examine the Restart Manager's mechanisms using the example of a software installer.
- Showcase how the Restart Manager can be used maliciously, explaining why and how malware authors employ this specific technique.
- Discuss how processes can be protected from malicious use of the

CATEGORIES

	Cloud & Application Security	104
	Counter Adversary Operations	184
	Endpoint Security & XDR	307
	Engineering & Tech	78
	Executive Viewpoint	162
	Exposure Management	84
	From The Front Lines	190
	Identity Protection	37
	Next-Gen SIEM & Log Management	91
	Public Sector	37
	Small Business	8

CONNECT WITH US



ABOUT COOKIES ON THIS SITE

By clicking "Accept All Cookies", you agree to the storing of cookies on your device to enhance site navigation, analyze site usage, and assist in our marketing efforts. [Cookie Notice](#)

Accept All Cookies

Reject All

Cookie Settings

process cannot access the file because it is being used by another process that have been mapped or files that have been opened without the FILE_SHARE_READ | FILE_SHARE_WRITE mode. In such cases, the file might be “locked” by the process, leading to the rejection of other processes requesting read or write access to that file.

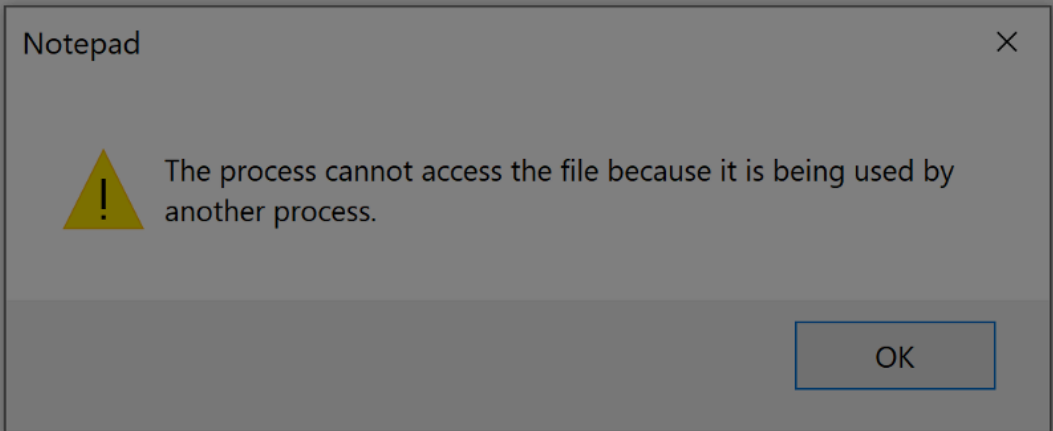


Figure 1. Error message displayed by Windows when a process tries to open a file already mapped by another process (click to enlarge)

If other processes need to access a “locked file,” they can request a reboot of the whole system to free the file from its locks. Beyond the interruption to user operations that OS reboots generate, reboots can also be resource-intensive. To address these pain points, beginning with Windows Vista, Microsoft introduced **the Restart Manager**, allowing applications to shut down processes locking resources without requiring a reboot.

Use of the Restart Manager

The Restart Manager is implemented in Windows through the “Rstrtmgr.dll” library, stored in %Windows%\System32.

Processes interact with the Restart Manager via what are called “**sessions**,” in which they register a set of **resources** and receive resulting information pertaining to them. **Resources** represent targets that need to be accessed by a process. As shown in Figure 2, a resource can be a file, a process, or a service, and a session can be composed of one or more of each of these. The role of the Restart Manager is to determine which processes are currently blocking a given resource, referred to as an **affected application**, and store the information about the applications into a **list**.

Users can author software to create a Restart Manager session, register a set of files, processes or services that they need to use, and determine if other processes are currently preventing them from doing so. If there are any, the Restart Manager would provide the list of the affected applications and the software can request the shutdown of those applications. This enables the software to check, prior to performing its operations, that nothing would impede their execution, which is important for procedures that shouldn’t be interrupted, such as updates.

September 25, 2024

Recognizing the Resilience of the CrowdStrike Community

September 25, 2024

CrowdStrike Drives Cybersecurity Forward with New Innovations Spanning AI, Cloud, Next-Gen SIEM and Identity Protection

September 18, 2024

SUBSCRIBE

Sign up now to receive the latest notifications and updates from CrowdStrike.

Sign Up



See CrowdStrike Falcon® in Action

Detect, prevent, and respond to attacks—even malware-free intrusions—at any stage, with next-generation endpoint protection.

See Demo

ABOUT COOKIES ON THIS SITE

By clicking “Accept All Cookies”, you agree to the storing of cookies on your device to enhance site navigation, analyze site usage, and assist in our marketing efforts. [Cookie Notice](#)

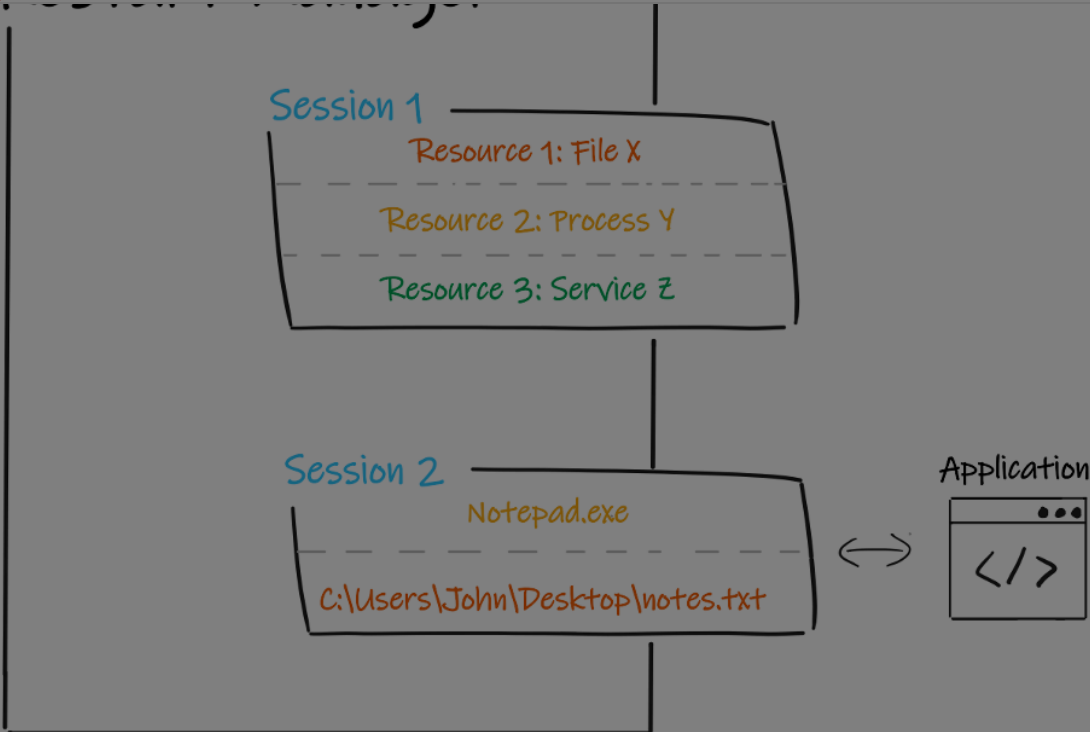


Figure 2. Architecture of the Restart Manager (click to enlarge)

To interact with the Restart Manager, one process would typically perform the following operations:

1. Create a Restart Manager session
2. Register a resource that can be a file, a service or a process
3. Retrieve through the list returned by the Restart Manager the affected applications blocking the registered resources

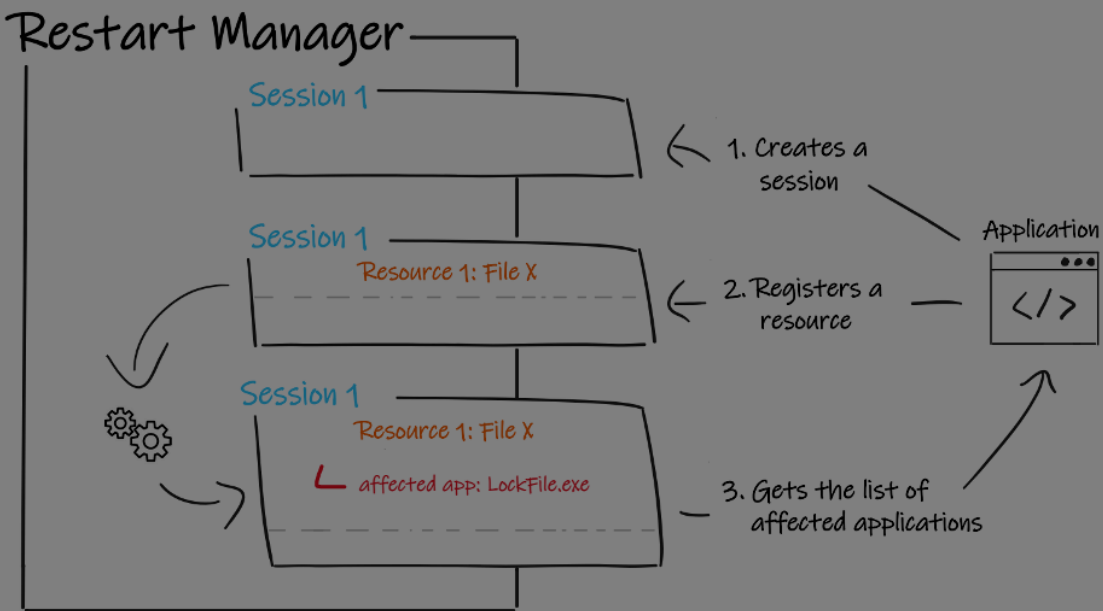


Figure 3. Typical use of the Restart Manager by an application (click to enlarge)

Let's see how processes technically interface with the Restart Manager and explain how the functions from the Restart Manager API can be used.

The first step to use the library is to create a **session** using [RmStartSession\(\)](#)¹:

```
DWORD RmStartSession(  
  
[out] DWORD *pSessionHandle,
```

ABOUT COOKIES ON THIS SITE

By clicking "Accept All Cookies", you agree to the storing of cookies on your device to enhance site navigation, analyze site usage, and assist in our marketing efforts. [Cookie Notice](#)

```

DWORD RmRegisterResources(
    [in]          DWORD          dwSessionHandle,
    [in]          UINT           nFiles,
    [in, optional] LPCWSTR []    rgsFileNames,
    [in]          UINT           nApplications,
    [in, optional] RM_UNIQUE_PROCESS [] rgApplications,
    [in]          UINT           nServices,
    [in, optional] LPCWSTR []    rgsServiceNames);

```

Depending on the nature of the resource to register, different types of information are required. For instance, registering a process requires the initialization of a `RM_UNIQUE_PROCESS` structure that gathers the process id and the process creation time:

```
typedef struct _RM_UNIQUE_PROCESS {  
  
    DWORD dwProcessId;  
  
    FILETIME ProcessStartTime;  
  
} RM_UNIQUE_PROCESS, *PRM_UNIQUE_PROCESS;
```

After registering one or more resources, the process can get the list of the affected applications (the applications currently blocking the resource) through the Restart Manager using the function `RmGetList()`³:

```

DWORD RmGetList(
    [in]          DWORD          dwSessionHandle,
    [out]         UINT           *pnProcInfoNeeded,
    [in, out]     UINT           *pnProcInfo,
    [in, out, optional] RM_PROCESS_INFO [] rgAffectedApps,
    [out]         LPDWORD        lpdwRebootReasons);

```

`PmGetList()` returns a list of `PM_PROCESS_INFO` structures, which gather the

ABOUT COOKIES ON THIS SITE

By clicking "Accept All Cookies", you agree to the storing of cookies on your device to enhance site navigation, analyze site usage, and assist in our marketing efforts. [Cookie Notice](#)

```
RM_APP_TYPE      ApplicationType;

ULONG            AppStatus;

DWORD            TSSessionId;

BOOL             bRestartable;

} RM_PROCESS_INFO, *PRM_PROCESS_INFO;
```

The RM_PROCESS_INFO structure contains key information for the Restart Manager to determine if a shutdown of the targeted application is requested by the process:

- **ApplicationType** defines the nature of the application:
 - Stand-alone process with or without a top-level window,
 - Console application
 - Windows Service
 - Critical application that the Restart Manager has determined cannot be shut down.
 - Note: This terminology is not to be confused with a **system critical process**, which is a process that would “*force a system reboot if they terminate*.”⁴ The Restart Manager may classify an affected application as a “critical application” because it is a:
 - System critical process that can’t be terminated
 - Process that does not have permission to shut down the application
 - One of its affected application is the owner of an active session with the Restart Manager
- RmUnkownApp, dedicated for applications that do not fall into any other category
- **AppStatus** indicates the application’s current context of execution is:
 - Running
 - Being restarted by the Restart Manager
 - Stopped by the Restart Manager or an external action
 - Encountering errors on Stop/Restart

The value returned through the AppStatus can be a composition of these values, combined with an OR operator.

- **bRestartable** determines whether the affected application can be restarted or not.

Along with the list of affected applications, RmGetList() specifies in the

ABOUT COOKIES ON THIS SITE

By clicking “Accept All Cookies”, you agree to the storing of cookies on your device to enhance site navigation, analyze site usage, and assist in our marketing efforts. [Cookie Notice](#)

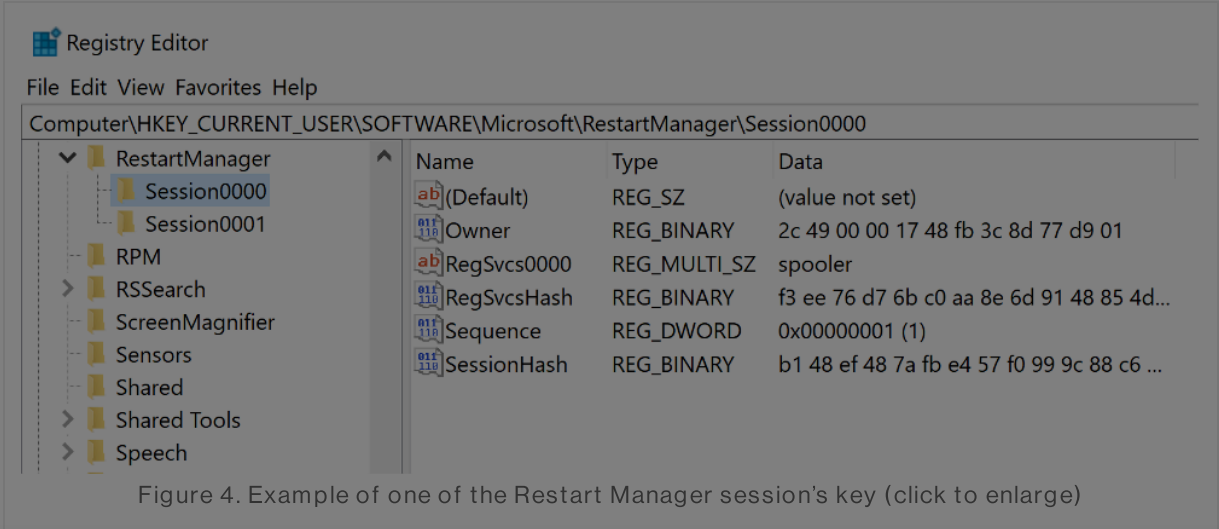
```
RmRebootReasonCriticalProcess = 0x4,  
  
RmRebootReasonCriticalService = 0x8,  
  
RmRebootReasonDetectedSelf  
  
} RM_REBOOT_REASON;
```

At this point, the process can request a shutdown of the affected application(s) using [RmShutdown\(\)](#)⁵:

```
DWORD RmShutdown(  
  
    [in]          DWORD          dwSessionHandle,  
  
    [in]          ULONG          lActionFlags,  
  
    [in, optional] RM_WRITE_STATUS_CALLBACK fnStatus  
  
);
```

Mechanisms of RmShutdown()

As the Restart Manager is able to legitimately terminate processes, it also provides malware authors with an opportunity to hijack its termination mechanism. When requested by one process, the `RmShutdown()` function will first retrieve internally the Restart Manager session associated with the session handle given in the parameters and will then perform internal checks to make sure the data is up-to-date. In addition to internal checks, the Restart Manager queries the registry hive of the current session under the key `HKCU\Software\Microsoft\RestartManager` (Figure 4) to synchronize internal data and registry information. Hives of each session are created at the execution of `RmStartSession()` and updated at each step of the Restart Manager execution with the internal data of the session.



The Restart Manager will then perform a set of checks to determine potential

ABOUT COOKIES ON THIS SITE

By clicking "Accept All Cookies", you agree to the storing of cookies on your device to enhance site navigation, analyze site usage, and assist in our marketing efforts. [Cookie Notice](#)

application.

Scenario 1: GUI Applications

If the affected application is a GUI application such as notepad.exe, the shutdown process is divided into three steps, implemented in three distinct undocumented functions of the Restart Manager internal C++ class called “ActionStrategy”:

- 1. ActionStrategy::ShutdownPrepAction()
- 2. ActionStrategy::ShutdownProcAction()
- 3. ActionStategy::WaitForProcsStops()

They mostly rely on the use of the Windows API function [SendMessageTimeoutW\(\)](#).⁸

SendMessageTimeoutW() is designed to send a **message** that represents information to be handled by a GUI window. The system and applications use messages as communication channels to notify applications about user inputs, window resizes or other requests such as shutdown procedures. Every message is transferred with the following parameters:

- The window handle of the recipient window
- A message identifier that represents the type of information to be transmitted
- Optional parameters if required

Within ShutdownPrepAction(), first a call to SendMessageTimeoutW() is performed with the message identifier [WM_QUERYENDSESSION \(0x11\)](#).⁹ This message is a request for the target application to ask if the application is ready to end the session. If the application returns FALSE, meaning that it is not ready, what happens depends on the way the shutdown has been requested. The second argument of RmShutdown(), IActionFlags can be the value “RmForceShutdown,” setting up a forced shutdown. When the shutdown is not forced, if the application is not ready to end, the shutdown is canceled. However, if the shutdown is forced, regardless of whether or not the application is ready to end, the Restart Manager moves to the next part of the procedure, similar to when the application is ready to end and has returned TRUE to the first message.

The second part of this procedure happens within the function ShutdownProcAction() that performs another call to SendMessageTimeoutW() with the value [WM_ENDSESSION \(0x16\)](#).¹⁰ This message notifies the application that the session needs to end, implicitly meaning that the application can be shut down by the system in a short amount of time if the application does not exit itself. It gives the application a timeout, offering the possibility for the application to do a last cleanup before ending, which can be useful, for example

ABOUT COOKIES ON THIS SITE

By clicking “Accept All Cookies”, you agree to the storing of cookies on your device to enhance site navigation, analyze site usage, and assist in our marketing efforts. [Cookie Notice](#)

Restart Manager sends the CTRL_C_EVENT to the affected application. This notification will be processed by the console’s control handler, which by default calls ExitProcess().

Scenario 3: Applications Associated with Services

If the affected application is a service, the applied method differs, as services do not have a GUI window and thus SendMessageTimeoutW() could not apply. To shut down a target service, two core functions of ActionStrategy are involved:

- 1. ActionStrategy::ShutdownService
- 2. ActionStrategy::TerminateProc

ShutdownService() first checks, using the Windows API function QueryServiceStatus(), that the service isn’t in SERVICE_STOPPED (0x1) state. If it is not, the function performs a call to [ControlService\(\)](#)¹² to notify the service that it should stop:

```
// extract of the pseudo-code from ActionStrategy::StopService()
SC_HANDLE hService;
DWORD ErrCode;
SERVICE_STATUS ServiceStatus;

if (ServiceStatus.dwCurrentState != SERVICE_STOP_PENDING
    && !ControlService(hService, SERVICE_CONTROL_STOP, &ServiceStatus)) // if non successful
{
    ErrCode = GetLastError();
}
```

Figure 5. Implementation of the service stop from ActionStrategy::StopService(), Rstrtmgr.dll (click to enlarge)

Then, regardless of the success of the service stop, ActionStrategy::TerminateProc() attempts to terminate the affected application process using [TerminateProcess\(\)](#)¹³:

```
// extract of the pseudo-code from ActionStrategy::TerminateProc()
DWORD EventCause = 0, ErrCode = 0;
HANDLE hProc;
RMProcInfoInternal* RMProcInfo;

hProc = (char *)RMProcInfo->hProc;
EventCause = WaitForSingleObject(hProc, 0);

if (EventCause)
{
    if (EventCause == WAIT_TIMEOUT)
    {
        if (TerminateProcess(RMProcInfo->hProc, 0)) // if succesful
            return ErrCode;
    }
}
```

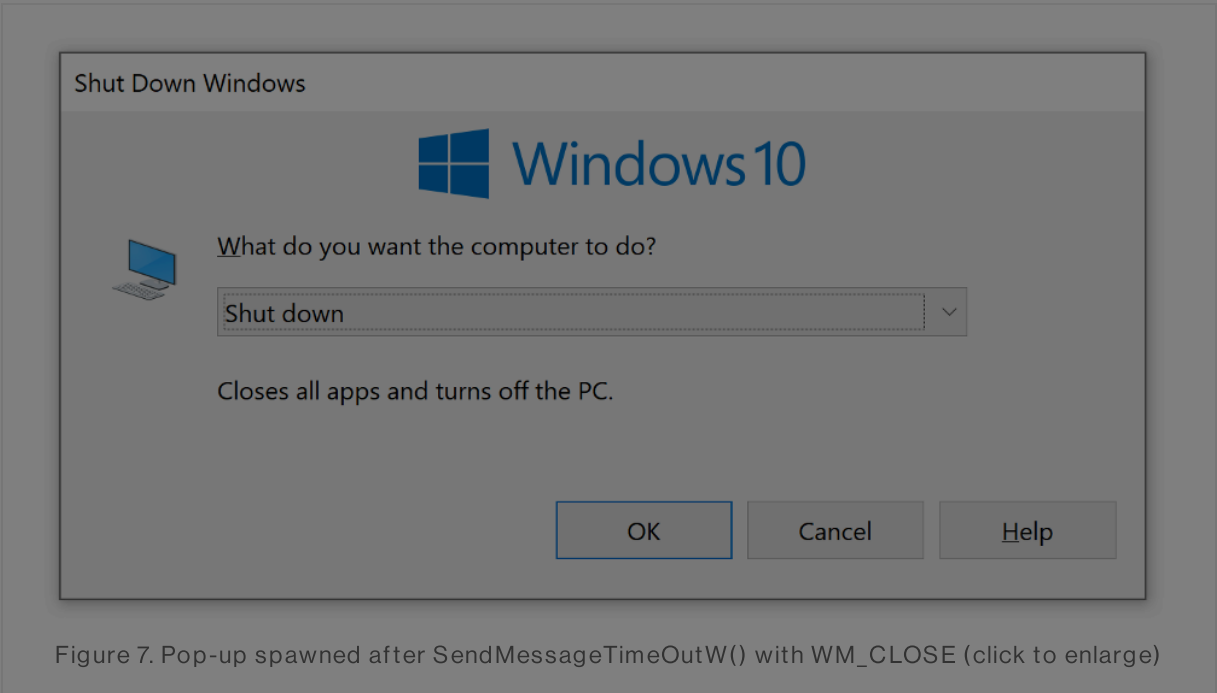
Figure 6. Implementation of the process termination from ActionStrategy::TerminateProc(), Rstrtmgr.dll (click to enlarge)

ABOUT COOKIES ON THIS SITE

By clicking “Accept All Cookies”, you agree to the storing of cookies on your device to enhance site navigation, analyze site usage, and assist in our marketing efforts. [Cookie Notice](#)

message WM_CLOSE (0x10) if the second call detected that the application isn’t ending.

Indeed, when sending SendMessageTimeoutW() with WM_CLOSE (0x10) to the explorer.exe process with no graphic window currently open, the OS will spawn a pop-up offering to shut down the system.



When no graphic window is found on the system, explorer.exe interprets the message as a request to “close” the system, which is the opposite of what the Restart Manager strives to accomplish. In this case, the Restart Manager enforces another procedure when the affected application is explorer.exe, limiting RmShutdown to the sending of WM_QUERYENDSESSION and WM_ENDSESSION.

Legitimate Use Cases

Outside of internal Microsoft applications, some installers leverage the Restart Manager to make sure no other application is currently using the files required for the install. To identify some of them, we used a software called [ninite](#),¹⁴ which allows users to select a set of applications among commonly downloaded software and generates a binary that will install all of them at once.

Using this binary, we used [API Monitor](#)¹⁵ to monitor the parent installer process and its children processes, hooking calls made to the Restart Manager to catch installers using it. To this end, we first set up the hooked functions in API Monitor (top left corner of Figure 8 below) and launched the ninite installer.

ABOUT COOKIES ON THIS SITE

By clicking “Accept All Cookies”, you agree to the storing of cookies on your device to enhance site navigation, analyze site usage, and assist in our marketing efforts. [Cookie Notice](#)

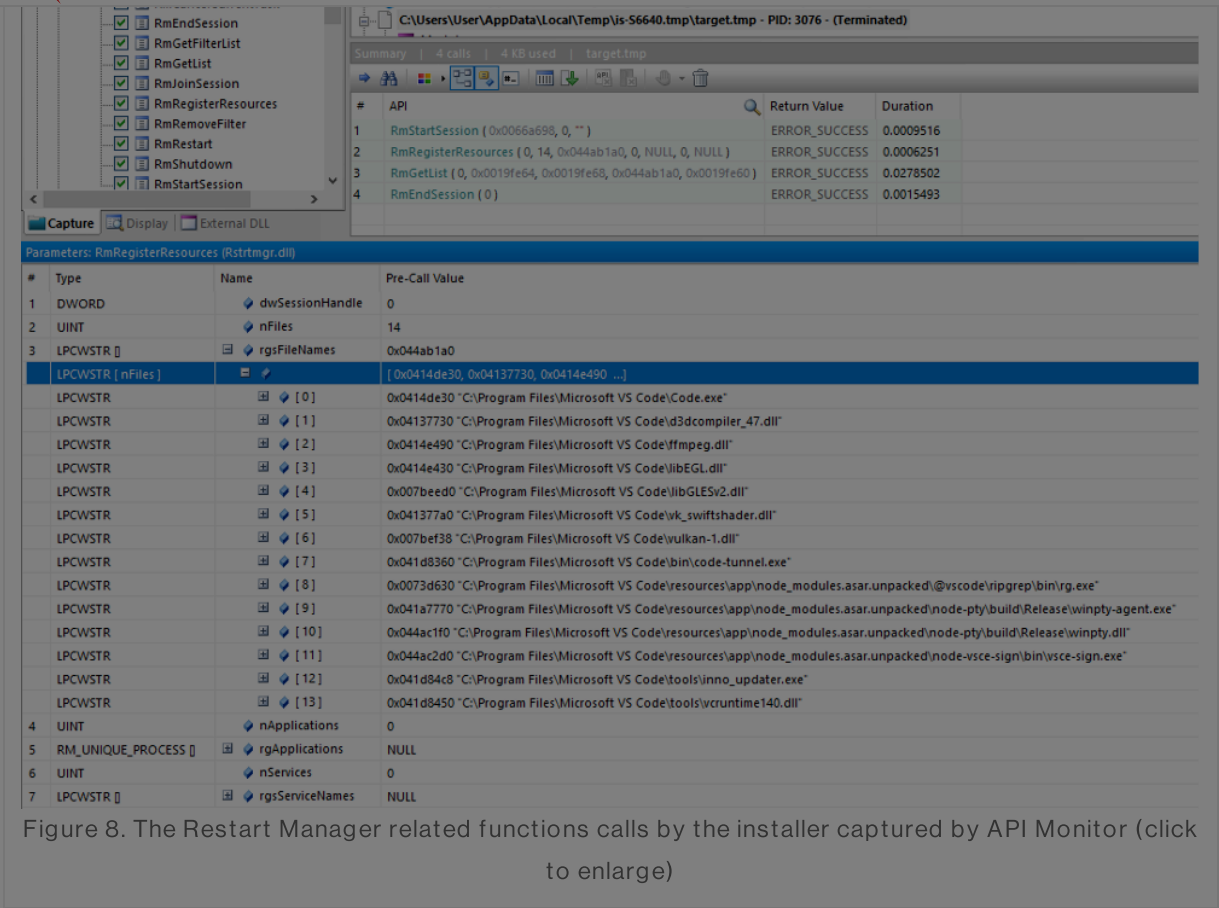


Figure 8. The Restart Manager related functions calls by the installer captured by API Monitor (click to enlarge)

In the top right corner, we can see the installer and its children processes, each installing software from an executable renamed “target.exe,” which launches a “target.tmp” process. If the names of the software installed are hidden by the “target.exe” and “target.tmp” formats, we can use the icons of the binaries to see what software is installed through them. In the middle window, among the hooked calls done by the Visual Studio Code installer, we can see that the installer performed four calls to the Restart Manager:

- RmStartSession()
- RmRegisterResources()
- RmGetList()
- RmEndSession()

Let’s take a closer look at the arguments of the RmRegisterResources() call displayed in the bottom window to observe what files the installer wanted to ensure are not used by another process. The installer registered no process or services, but in a Restart Manager session registered 14 binary files, including .exe and .dll binaries that are commonly used for installation or updates, such as “inno_updater.exe” (*filename with the index 12*).

This example shows a typical use of the Restart Manager for one installer, validating prior to the installation or update that the binaries about to be upgraded are not currently blocked by other applications.

Summary

In this blog, we reviewed what the Restart Manager is and how it works. This library enables programs to verify that no applications will block the resources they need prior to executing certain operations, notably in the case of

ABOUT COOKIES ON THIS SITE

By clicking “Accept All Cookies”, you agree to the storing of cookies on your device to enhance site navigation, analyze site usage, and assist in our marketing efforts. [Cookie Notice](#)

Additional Resources

- To learn more about how threat actors are using Microsoft components in attacks, join us at Fal.Con 2023, the can’t-miss cybersecurity experience of the year. Register now and meet us in Las Vegas, Sept. 18-21!
- Get a comprehensive look at the evolving techniques of today’s adversaries in our new report, Nowhere to Hide: CrowdStrike 2023 Threat Hunting Report.
- Experience the benefits of Falcon for yourself. Start your free trial of CrowdStrike Falcon® Prevent today.

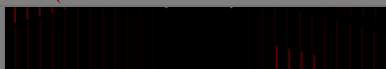
- https://learn.microsoft.com/en-us/windows/win32/api/restartmanager/nf-restartmanager-rmstartsession
- https://learn.microsoft.com/en-us/windows/win32/api/restartmanager/nf-restartmanager-rmregisterresources
- https://learn.microsoft.com/en-us/windows/win32/api/restartmanager/nf-restartmanager-rmgetlist
- https://devblogs.microsoft.com/oldnewthing/20180216-00/?p=98035
- https://learn.microsoft.com/en-us/windows/win32/api/restartmanager/nf-restartmanager-rmshutdown
- https://learn.microsoft.com/en-us/windows/win32/api/restartmanager/nf-restartmanager-rmrestart
- https://learn.microsoft.com/en-us/windows/win32/api/restartmanager/nf-restartmanager-rmaddfilter
- https://learn.microsoft.com/en-us/windows/win32/api/winuser/nf-winuser-sendmessagetimeoutw
- https://learn.microsoft.com/en-us/windows/win32/shutdown/wm-queryendsession
- https://learn.microsoft.com/en-us/windows/win32/shutdown/wm-endsession
- https://learn.microsoft.com/en-us/windows/win32/winmsg/wm-close
- https://learn.microsoft.com/en-us/windows/win32/api/winsvc/nf-winsvc-controlservice
- https://learn.microsoft.com/en-us/windows/win32/api/processthreadsapi/nf-processthreadsapi-terminateprocess
- https://ninite.com/
- http://www.rohitab.com/apimonitor

X Tweet

f Share

ABOUT COOKIES ON THIS SITE

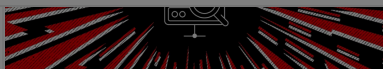
By clicking “Accept All Cookies”, you agree to the storing of cookies on your device to enhance site navigation, analyze site usage, and assist in our marketing efforts. Cookie Notice



Tech Analysis: Channel File May Contain Null Bytes



EMBERSim: A Large-Scale Databank for Boosting Similarity Search in Malware Analysis



CrowdStrike Falcon Next-Gen SIEM Unveils Advanced Detection of Ransomware Targeting VMware ESXi Environments

« How CrowdStrike Uses Similarity-Based Mapping to Understand Cybersecurity Data and Prevent Breaches

The Windows Restart Manager: How It Works and How It Can Be Hijacked, Part 2 >>



ABOUT COOKIES ON THIS SITE

By clicking "Accept All Cookies", you agree to the storing of cookies on your device to enhance site navigation, analyze site usage, and assist in our marketing efforts. [Cookie Notice](#)