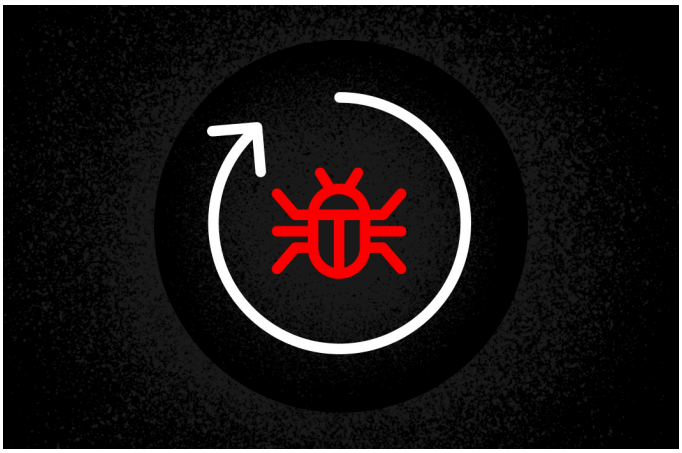
The Windows Restart Manager: How It Works and How It Can Be Hijacked, Part 1

August 25, 2023 | Mathilde Venault | Engineering & Tech



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.

Featured
Recent
Video
Category
Start Free Trial

this mechanism can be hijacked to serve malicious purposes.

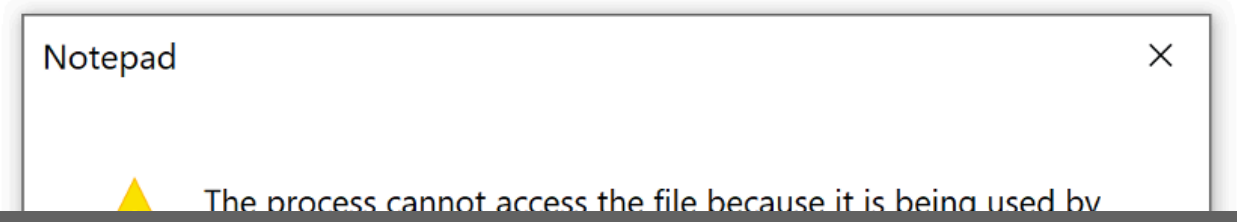
In this two-part series, we:

how malware authors employ this specific technique.
- Discuss how processes can be protected from malicious use of the Restart Manager.

Architecture and Functionalities

Origin

Each operating system (OS) has a unique way of handling simultaneous access to files. In the case of Windows, depending on how a file has been opened, one process can have an exclusive access to the file — as in the case of files 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.



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

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

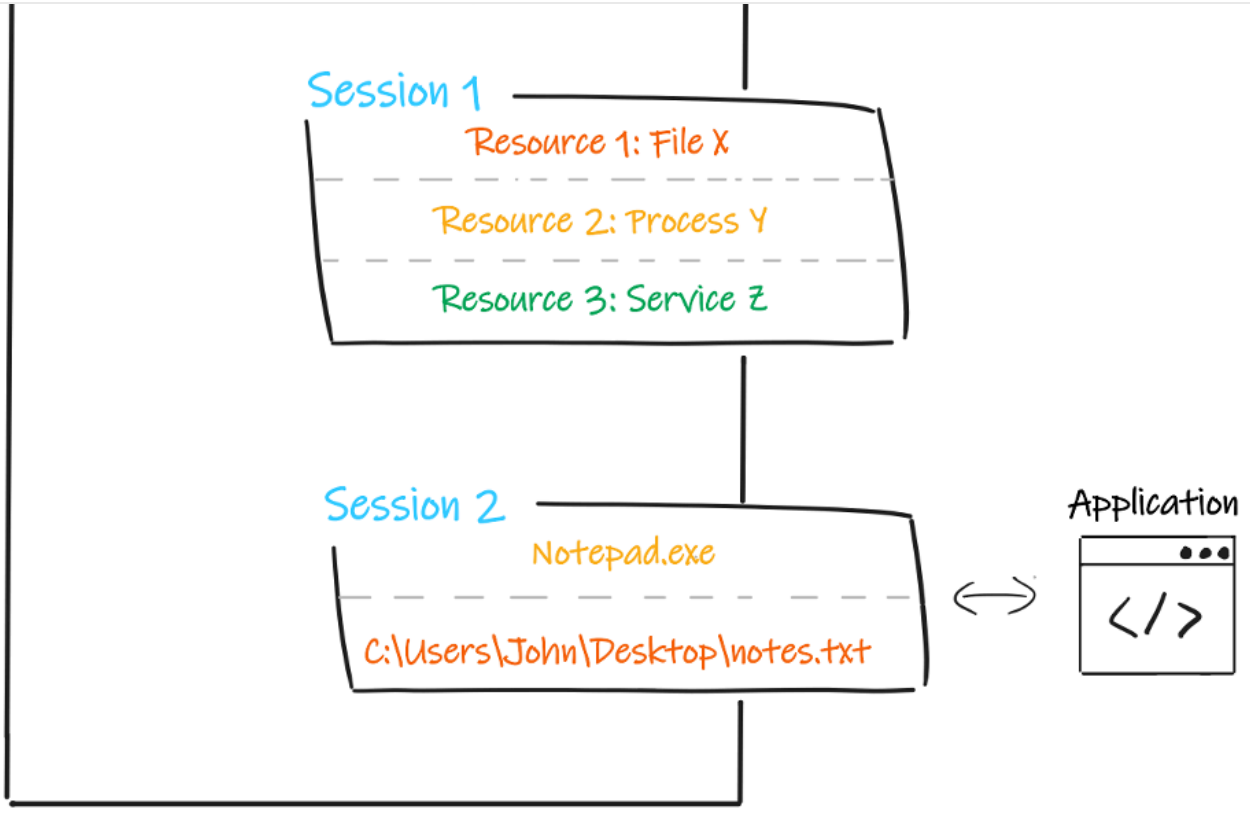


Figure 2. Architecture of the Restart Manager (click to enlarge)

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

1. Create a Restart Manager session

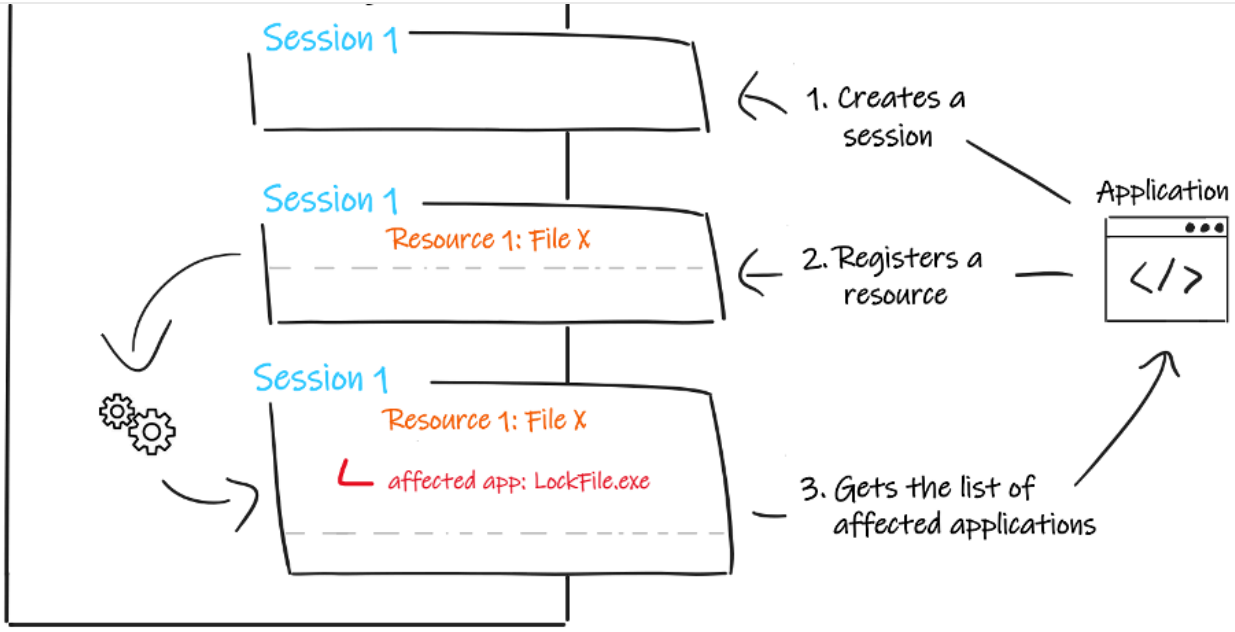


Figure 3. Typical use of the Restart Manager by an application (click to enlarge)

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,
```

Featured
Recent
Video
Category
Start Free Trial

required, but the Restart Manager can simultaneously handle a maximum number of 64 sessions for the whole system. Once the session is created, a process can register one or more resources per session through the function RmRegisterResource()[2]:

```
  [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;
```



```
DWORD RmGetList(

  [in]                DWORD             dwSessionHandle,
```

```
  [in, out]           UINT               *pnProcInfo,

  [in, out, optional] RM_PROCESS_INFO [] rgAffectedApps,

  [out]               LPDWORD            lpdwRebootReasons);
```

RmGetList() returns a list of RM_PROCESS_INFO structures, which gather the following information for each affected application:

```
typedef struct _RM_PROCESS_INFO {

  RM_UNIQUE_PROCESS Process;

  WCHAR             strAppName[CCH_RM_MAX_APP_NAME + 1];

  WCHAR             strServiceShortName[CCH_RM_MAX_SVC_NAME + 1];

  RM_APP_TYPE       ApplicationType;
```

Featured
Recent
Video
Category
Start Free Trial

```
} 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:

- 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

Featured
Recent
Video
Category
Start Free Trial

reason can be one of the following:

```
typedef enum _RM_REBOOT_REASON {
```

```
  RmRebootReasonPermissionDenied = 0x1,

  RmRebootReasonSessionMismatch = 0x2,

  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,
```

Featured
Recent
Video
Category
Start Free Trial

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

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.

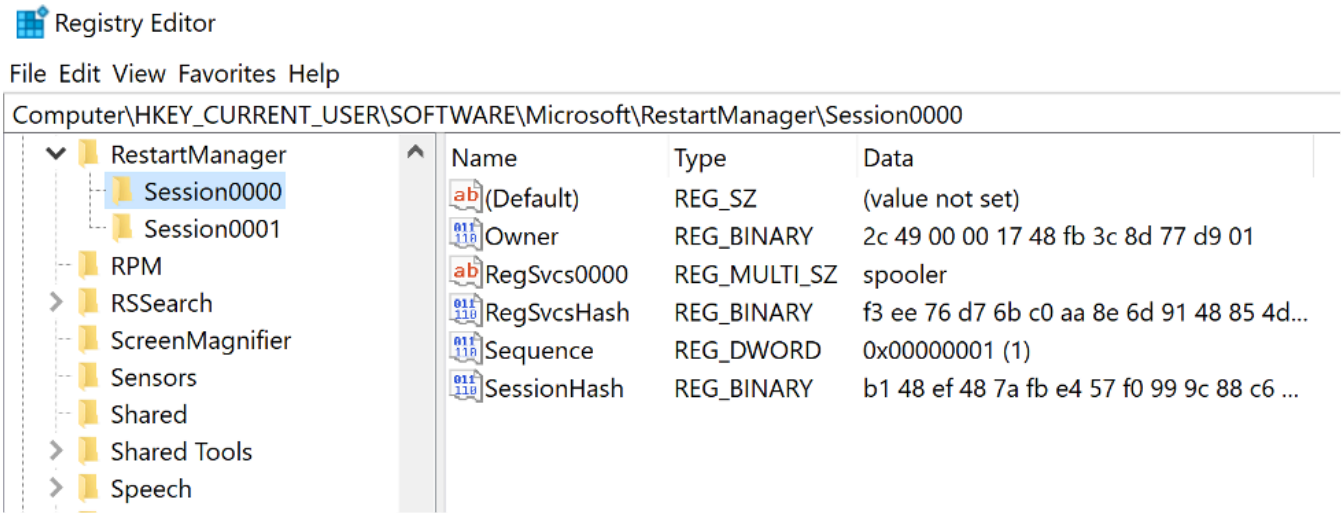


Figure 4. Example of one of the Restart Manager session's key (click to enlarge)

The Restart Manager will then perform a set of checks to determine potential changes to consider before the shutdown process, such as:

- If an OS reboot is required
- If there are modifications requested by the owner of the session via RmAddFilter()[7]
- If the affected application isn't a critical application from the Restart Manager

Featured
Recent
Video
Category
Start Free Trial

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":

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(), lActionFlags 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

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 to save user data.

Scenario 2: Console Applications

As messages are designed to be sent to GUI window handles, the Restart Manager needs to adjust when the affected application is a console application and instead will send a **notification.** To signify the termination request, the 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:

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]:

```
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)

Scenario 4: explorer.exe

Finally, if the affected application is Windows Explorer, a dedicated function named ActionStrategy::ExplorerShutdownProcAction() is called. This function relies partly on the same mechanism as for Scenario 1 applications, first performing a call to

offering to shut down the system.

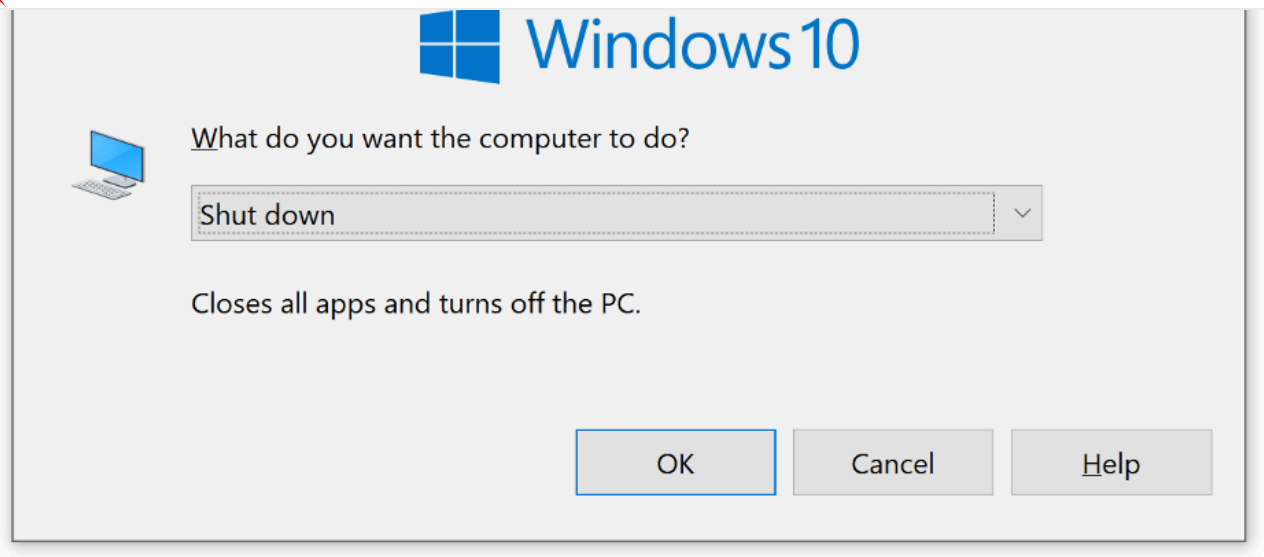


Figure 7. Pop-up spawned after SendMessageTimeOutW() with WM_CLOSE (click to enlarge)

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

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.

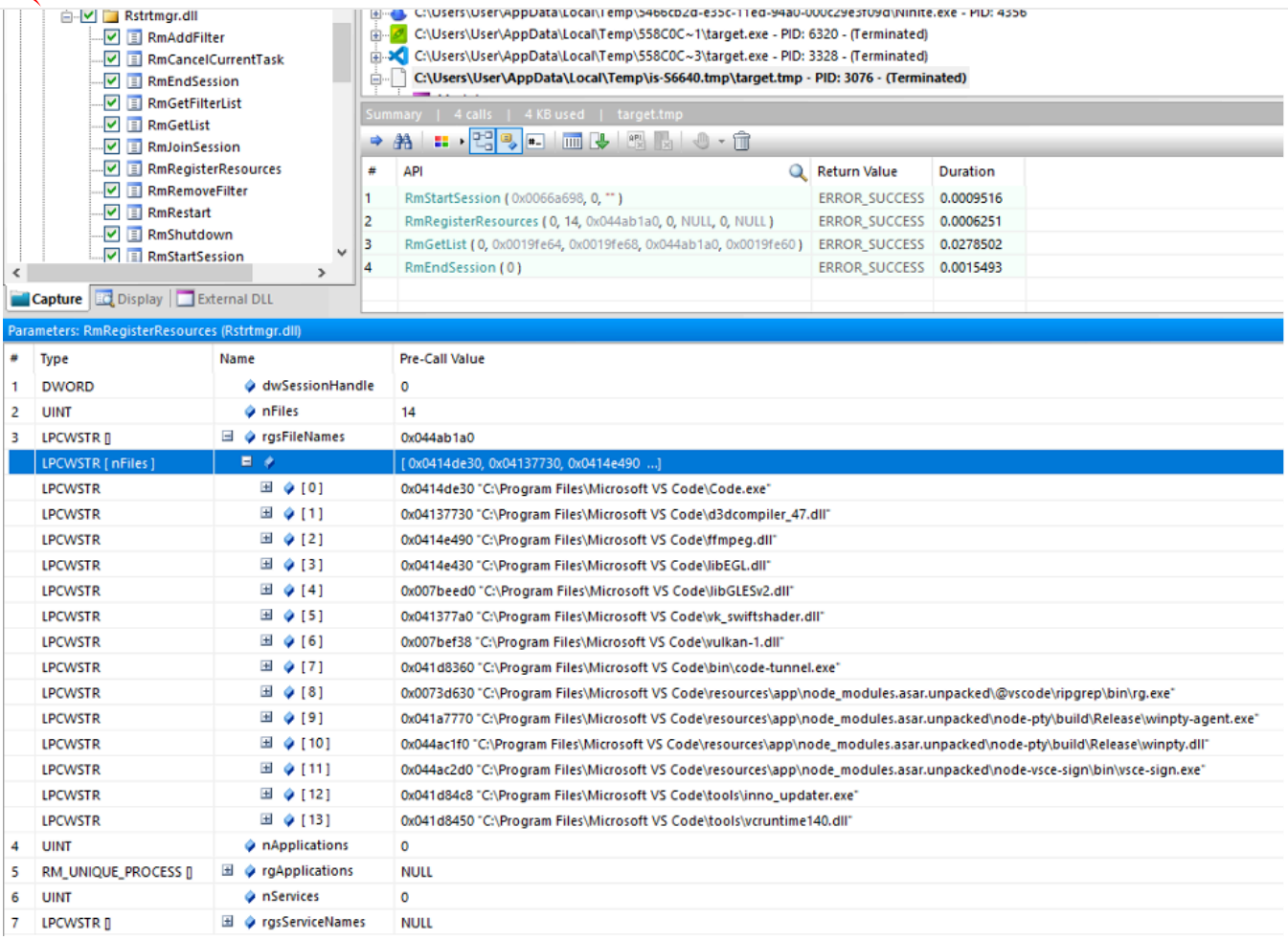


Figure 8. The Restart Manager related functions calls by the installer captured by API Monitor (click to enlarge)

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"

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

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*).

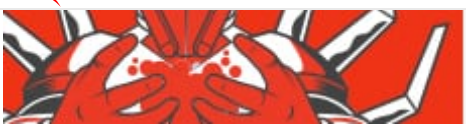
This example shows a typical use of the Restart Manager for one installer, validating prior to the installation or update that the binaries about to be upgraded are not currently blocked by other applications.

Summary

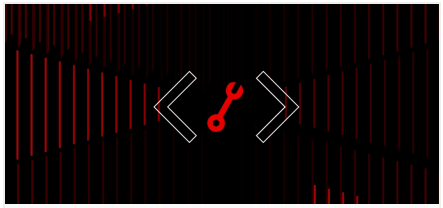
In this blog, we reviewed what the Restart Manager is and how it works. This library enables programs to verify that no applications will block the resources they need prior to executing certain operations, notably in the case of operations that should not be interrupted. We also detailed how a process can use the Restart Manager to request the shutdown of affected applications to free the locks currently blocking the resource they need. In the second part of this series, we will explore how this functionality can be hijacked by malicious authors and examine real-world examples. We will also explain how an application can be protected against these malicious techniques and how the CrowdStrike Falcon platform gives users visibility into attacks that exploit the Restart Manager’s termination mechanism.

2. https://learn.microsoft.com/en-us/windows/win32/api/restartmanager/nf-restartmanager-rmregisterresources
3. https://learn.microsoft.com/en-us/windows/win32/api/restartmanager/nf-restartmanager-rmgetlist
4. https://devblogs.microsoft.com/oldnewthing/20180216-00/?p=98035
5. https://learn.microsoft.com/en-us/windows/win32/api/restartmanager/nf-restartmanager-rmshutdown
6. https://learn.microsoft.com/en-us/windows/win32/api/restartmanager/nf-restartmanager-rmrestart
7. https://learn.microsoft.com/en-us/windows/win32/api/restartmanager/nf-restartmanager-rmaddfilter
8. https://learn.microsoft.com/en-us/windows/win32/api/winuser/nf-winuser-sendmessagetimeoutw
9. https://learn.microsoft.com/en-us/windows/win32/shutdown/wm-queryendsession
10. https://learn.microsoft.com/en-us/windows/win32/shutdown/wm-endsession
11. https://learn.microsoft.com/en-us/windows/win32/winmsg/wm-close
12. https://learn.microsoft.com/en-us/windows/win32/api/winsvc/nf-winsvc-controlservice
13. https://learn.microsoft.com/en-us/windows/win32/api/processthreadsapi/nf-

Featured
Recent
Video
Category
Start Free Trial



Related Content



Tech Analysis: Channel File May Contain Null Bytes

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

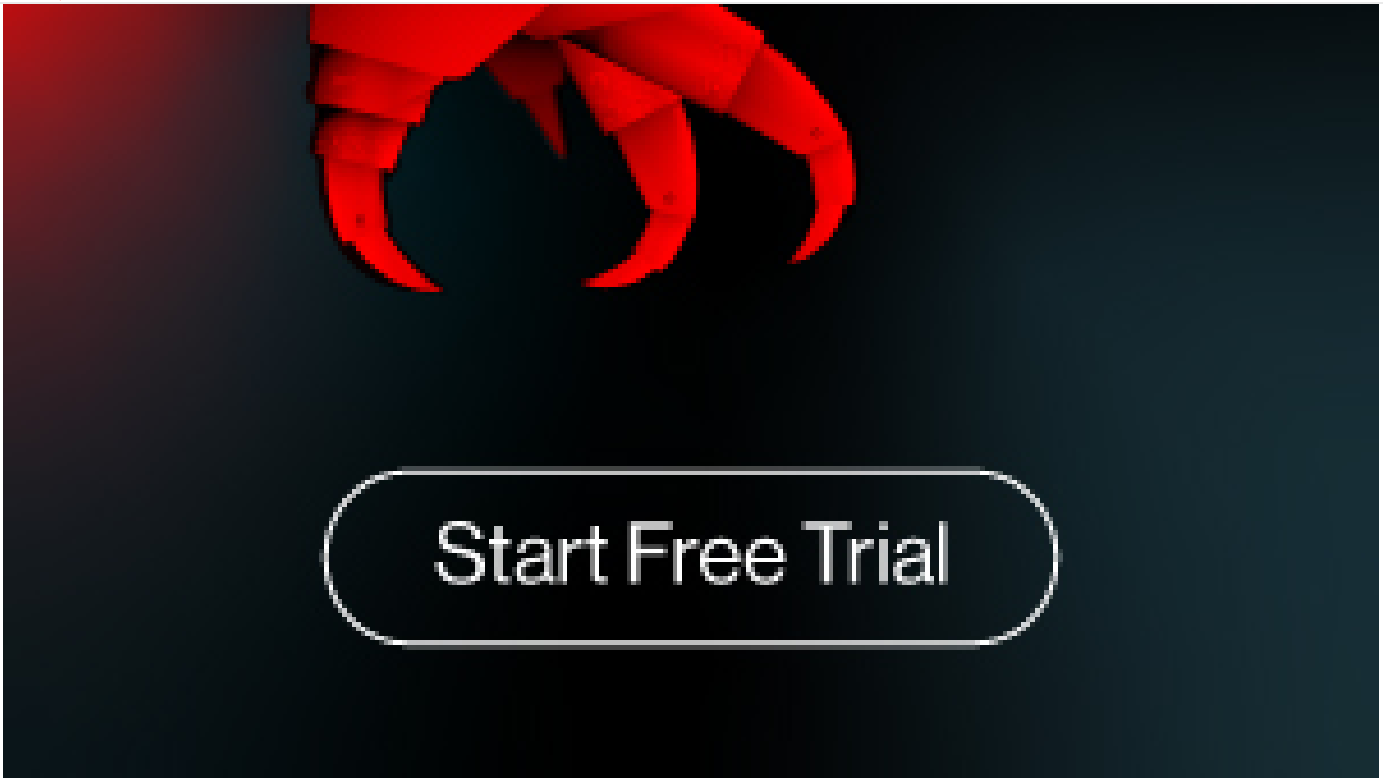
CrowdStrike Falcon Next-Gen SIEM Unveils Advanced Detection of Ransomware Targeting VMware ESXi Environments

CATEGORIES

Featured
Recent
Video
Category
Start Free Trial

Exposure Management 84
From The Front Lines 190
Identity Protection 37

CONNECT WITH US



FEATURED ARTICLES

October 01, 2024

CrowdStrike Named a Leader in 2024 Gartner® Magic Quadrant™ for Endpoint Protection Platforms

September 25, 2024

Featured
Recent
Video
Category
Start Free Trial

SUBSCRIBE

« 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 »

Featured
Recent
Video
Category
Start Free Trial