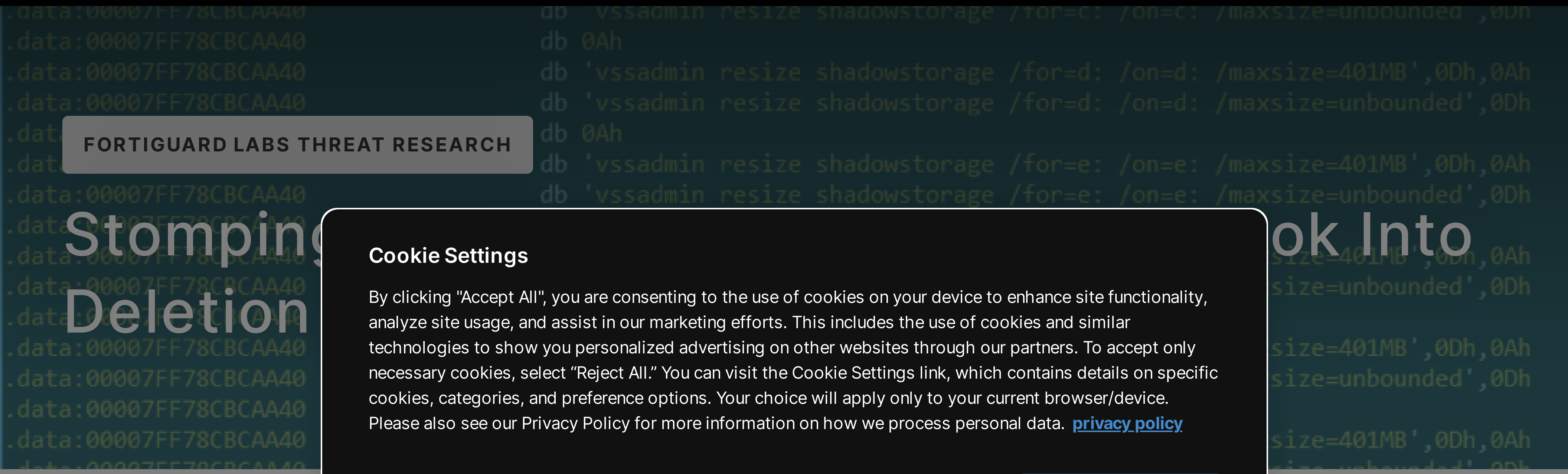


FORTIGUARD LABS THREAT RESEARCH

Stomping ... ok Into Deletion

By Ben Hunter | April 2

FortiGuard Labs Threat Analysis

Affected Platforms:	Windows
Impacted Users:	Any organization
Threat Severity:	High

Ransomware has been a dominant threat to organizations for several years now, causing unparalleled damage estimated in the billions of dollars. To be extremely effective, a common action carried out by ransomware is to erase volume backups (i.e. shadow copies), thereby preventing victims from being able to recover any files that had been encrypted.

In this blog, we review existing methods used by ransomware to delete shadow copies in order to give defenders a recap of the techniques they need to protect themselves and their digital resources against. Additionally, we expose new methods that can potentially be used by ransomware. Sharing these methods will allow defenders to deploy appropriate detections for these potential future techniques, as it's only a matter of time before we'll encounter them in the wild.

VSS Architecture

Before we get started, there are several important elements readers should be familiar with with regards to a Volume Shadow Copy architecture.

Volume Shadow Copy Service (VSS): The service is responsible for orchestrating all actions between the entities that perform shadow copy-related operations, such as the relevant writers and providers. VSS is implemented using COM (Component Object Model) technology, which the reader should have basic familiarity with for the latter parts of this article.

VSS Writers: These components are responsible for supplying a steady pipeline of data to be backed up by the service. Windows has a long list of "In-Box VSS Writers" for various software components.

VSS Providers: These components create and maintain the actual shadow copies objects. There are two main types of providers: "*Hardware Providers*" and "*Software Providers*".

A diagram of VSS architecture that illustrates the components and their relationships can be seen in Figure 1.

Figure 1: General architecture diagram from MSDN

The System Provider

The ***System Provider*** is the default provider on Windows, currently implemented as a software provider. It is the most common target when dealing with shadow copies deletion attempts. The system provider employs the copy-on-write (C-o-W) mechanism so snapshots save only the changes being made on the volume. Those changes are saved in a designated "diff area'' storage location, which is usually on the same volume. But they can also be saved on any NTFS volume big enough to contain them.

Existing Deletion Methods

There are two approaches for deleting shadow copies. The first is to explicitly delete shadow copies using command-line utilities, or programmatically in various ways (which we'll describe later in this article).

The second approach takes an indirect route, as it relies on the fact it is possible to control the size of the "diff area". If the existing snapshots exceed the size of the newly resized diff area, the provider will drop snapshots to free up space, as explained in the documentation in Figure 2. (note that the minimum possible size to set is 320MB).

Note that administrator privileges are required to work with VSS.

Figure 2: Resizing warning from MSDN

There are a couple of well-known command-line utilities that can manage VSS: vssadmin.exe, which has been shipped with every fresh Windows installation starting with Windows XP, and WMIC.exe (WMI Command-line), which provides access to Windows Management Instrumentation. Actors evolved their usage of these utilities over time to counter detection by defenders.

The first and most commonly used method is to run vssadmin with the delete command:

```
vssadmin delete shadows /all /quiet
```

Some infamous ransomware families that were observed utilizing this method are Ryuk, WannaCry, Dharma, RobinHood, Hermes, Phobos, and Locky.

After this method became widely known, and defended against, attackers started utilizing WMIC.exe to delete shadow copies, using the following parameters:

```
wmic shadowcopy delete /nointeractive
```

This method is also very popular among ransomware authors, and is used by GandCrab, Robinhood, Phobos, LockBit, Rapid, and JCry.

Resizing, on the other hand, is a relatively new approach among ransomware families compared to the previous two methods, and is also not as common in the wild even though it's also performed with vssdamin:

```
vssadmin resize shadowstorage /for=<backed volume> /on=<backup location volume> /maxsize=<new size>
```

Figure 3: Example of resize commands strings used by Ryuk

This method is used by a few ransomware families, such as Nemty, Ryuk, Hermes, Rapid, and MegaCortex (the only one to actually use the minimum size), and we expect to see wider use of it in the future.

The problem with this method is that we don't necessarily know how much space in the diff area is used, which could potentially lead to a scenario that, in spite of the resize, some snapshots may not get deleted. Since the System Provider uses C-o-W, changes are saved to the diff area as they occur and before a snapshot is created. To overcome this issue, after the resize operation the diff area should be filled with data in an amount equal to or greater than its shrunken size, and followed by the creation of a new snapshot in order to trigger deletion of all existing snapshots.

The latest innovation among ransomware authors is the preference to invoke deletion of shadow copies directly from their code (or scripts). Running PowerShell one-liners is favored by ransomware, enumerating and deleting all instances of shadow copies in one simple line of code.

This is conveniently achieved with WMI's Win32_ShadowCopy class and the helpful PowerShell cmdlets to access WMI objects, as seen in the following examples:

1. `Get-WmiObject Win32_ShadowCopy | % { $_.Delete() }`
2. `Get-WmiObject Win32_ShadowCopy | Remove-WmiObject`

`Get-WmiObject` cmdlet is used by Nemty and Sodinokibi. Sodinokibi runs PowerShell as a child process with a base64 encoded command line parameter that decodes to:

```
Get-WmiObject Win32_Shadowcopy | ForEach-Object { $_Delete(); }
```

In the next sections, we'll cover deletion methods that have yet to be spotted ITW.

New Deletion Methods

In this section, we discuss new methods to delete shadow copies that can potentially be used by ransomware in the future. The first method presents additional PowerShell tricks to trigger deletion, while the second and third methods take advantage of the behavior and internal workings of the VSS architecture.

1. More PowerShell Tradecraft

Although we've yet to encounter any ransomware that uses this, it's possible to use CIM cmdlets instead of the WMI ones:

```
Get-CimInstance Win32_ShadowCopy | Remove-CimInstance
```

When CIM-based calls

PowerShell cmdlets als CimInstance. New ones can be created vi

It's also possible to run Powershell scripts; for instance SyncAppv ture based detection.

Instead of using a know s from within your own process using .NET Framework. A few tools have already implement it (UnmanagedPowerShell and SharpPick).

2. Invoking COM Objects

WMI can be used programmatically via COM instead of command-line tools or PowerShell, and as we mentioned earlier, the VSS architecture itself is COM-based so a more straightforward approach can be used to operate those objects directly.

If we look under the hood of vssadmin, we can quickly notice that it uses the VSSCoordinator COM class with the IVssCoordinator interface to issue a deletion request to the VSS service. This object is implemented in a proxy DLL named vss_ps.dll.

Figure 4: Registry lookup of the proxy DLL for the IVssCoordinator interface

Following this finding, the obvious direction is to use the same COM object directly, hence, leaving the vssadmin binary redundant. The relevant interface definitions appear in vscoordint.idl, which was shipped in older Windows SDK releases. The delete function definition is:

```
HRESULT DeleteSnapshots(
VSS_ID SourceObjectId,
VSS_OBJECT_TYPE eSourceObjectType,
BOOL bForceDelete,
LONG* plDeletedSnapshots,
VSS_ID* pNondeletedSnapshotID)
```

`SourceObjectId` is the GUID identifier of the shadow copy itself and the `eSourceObjectType` is set to `VSS_OBJECT_SNAPSHOT`.

There are also documented VSS objects that provide all the necessary functionality to manage snapshots. One of them is the IVssSoftwareSnapshotProvider interface, which can also be used to invoke the desired deletion operation. Changing the size of the diff area can be done using the IVssDifferentialSoftwareSnapshotMgmt interface.

3. Direct Device Access

When deleting snapshots, the execution flow gets to the provider (swprv.dll) that sends IOCTLs to volsnap.sys. The kernel driver doesn't perform any access checks, either on device open requests or on the IOCTLs sent to it, to validate that the source is the provider service process, so it's possible to send those IOCTLs from any process, disposing of even the COM layer.

Using ProcMon we can easily track the operations performed by the provider:

1. Open a handle to the shadow copy volume (for example \Device\HarddiskVolumeShadowCopy1).
2. Send the undocumented IOCTL_VOLSNAP_SET_APPLICATION_INFO.
3. Set the volume as hidden with IOCTL_VOLUME_SET_GPT_ATTRIBUTES if it is not already.
4. Next, the volume is disabled by sending FSCTL_DISMOUNT_VOLUME and IOCTL_VOLUME_OFFLINE.
5. Open a handle to the backed volume (i.e. C:, the "Original Volume" in the shadow copy's properties).
6. Send the undocur f the snapshot to delete in the input

Querying shadow copie COM objects). Steps 1-4 are optional, but ke a failure or unexpected results.

Figure 5: ProcMon log of the IOCTL deletion call

In order to resize the diff area, the provider sends the undocumented `IOCTL_VOLSNAP_SET_MAX_DIFF_AREA_SIZE` (0x53C028) to the original volume, as presented in Figure 6.

Figure 6: The undocumented IOCTL under the ChangeDiffAreaMaxmiumSize in swprv.dll

Even if the driver would validate that the source of requests is the provider's service, injecting code into it to perform the actions described above will bypass that check.

Detection

The methods we described operate at different layers, and each present a number of options for detection mechanisms.

The first place to start is with command-line parameters scanning, which is a very elementary solution. While it covers the majority of techniques that ransomware uses today, defenders have to take into account diverse styles of syntax and be able to handle new forms of obfuscation and the discovery of new LOLBins. Furthermore, there are known techniques to hide command-line parameters.

For a broader approach, some form of COM monitoring is required.

An intuitive solution for this can be to search for any anomalies of libraries being loaded in unexpected processes. For example, if vss_ps.dll gets loaded by processes other than vssadmin.exe or wmiprvse.exe (when WMI is used), they can be considered suspicious. However, this can generate false-positives as it doesn't assure us that vss_ps.dll is actually being used and that shadow copies are being deleted. Monitoring the actual COM objects will grant more accurate results. Monitoring access to the relevant keys in the registry is possible, but has the same issues with regards to false-positives. Instead, using COM object proxies and filters may prove to be the adequate solution.

The ultimate solution would be to monitor the final step in the execution flow, as it's the common intersection point in all the methods - the device IO control calls to the driver. Those are supposed to originate from the provider service. Installing hooks in every process on functions such as DeviceIoControl and ZwDeviceIoControlFile to monitor those calls is possible, but they can be evaded without much effort by attackers. Consuming ETW events to detect those IOCTL codes is a more feasible possibility. The last option is to use a kernel driver and perform IRP filtering, which also supports the ability to block any potentially malicious request.

Solution

The FortiEDR platform is capable of detecting the existing and new techniques outlined in this document.

Summary

In this article, we revie utilities to various programmatic forms.

Almost all of those me ocumented. We demonstrated an alter s while accessing the shadow copy device d

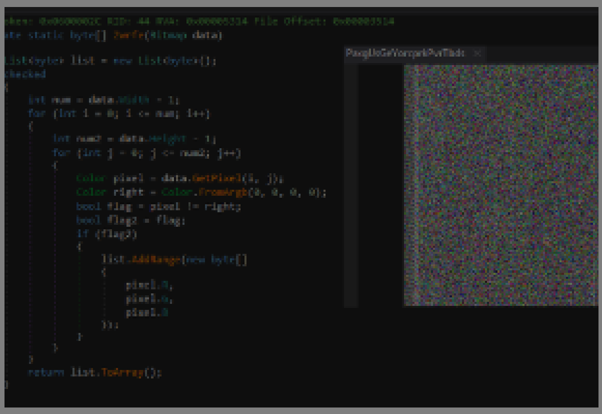
We suggested different elying on command-line parameters alone f the shadow copies devices themselves.

To conclude, this research highlights the premise that defenders must be vigilant to stay ahead of the bad guys and constantly adopt new detection methodologies and capabilities. Moreover, conducting offensive research is just as important to defenders as it is to attackers in order to stay ahead of the curve.

Learn more about FortiGuard Labs threat research and the FortiGuard Security Subscriptions and Services portfolio. Sign up for the weekly Threat Brief from FortiGuard Labs.

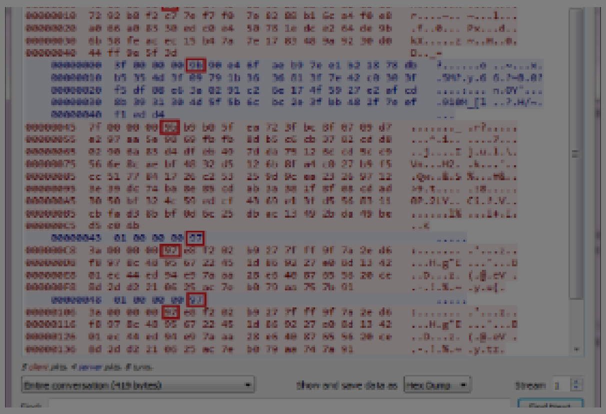
Learn more about Fortinet's free cybersecurity training initiative or about the Fortinet Network Security Expert program, Network Security Academy program, and FortiVet program.

Related Posts



FORTIGUARD LABS THREAT RESEARCH

Deconstructing an Evasive Formbook Campaign Leveraging COVID-19 Themes



FORTIGUARD LABS THREAT RESEARCH

NetWire RAT Targeting Taxpayers is Spreading via Legacy Microsoft Excel 4.0 Macro

FORTIGUARD LABS THREAT RESEARCH

Preparing for the Surge in Attacks Targeting Remote Workers