Yours Truly, Signed AV Driver: Weaponizing An Antivirus Driver

Home → Aon's Cyber Labs → **Yours Truly, Signed AV Driver: Weaponizing an Antivirus Driver**

As we head into 2022, ransomware groups continue to plague our digital environment with new and interesting techniques to bypass Antivirus (AV) and Endpoint Detection and Response (EDR) solutions and ensuring the successful execution of their ransomware payloads.

In December 2021, Stroz Friedberg's Incident Response Services team engaged in a Digital Forensics and Incident Response (DFIR) investigation and environment-wide recovery of a Cuba ransomware incident. We discovered novel indicators of compromise (IOCs) utilizing an interesting technique. Here, as part of the Cuba's toolset, the threat actor group executed a script that abused a function in an Avast® Anti Rootkit kernel driver to terminate popular AV and EDR processes.

While the use of kernel drivers to target and kill AV and EDR solutions[1] prior to encryption has been known and discussed for some time, the abuse of a signed and valid driver from an Antivirus vendor[2] was surprisingly effective and ironic.

At the time of writing this article, there are three different versions of the same attack. They are listed below in the order of implementation complexity:

- A self-contained PowerShell script, dropped alongside the Avast driver, that installs and loads the driver and executes a small number of functions to control the driver.
- An executable that unpacks and loads in memory a small executable to control the driver. Within this blog, we refer to this executable as the controller. Additional tools are used to install and load the Avast driver in the infected system.
- A batch script that installs a service to load the Avast kernel driver, then launches a PowerShell script to decode, load and execute the controller in memory.

This article delves into the implementation of the third variant of the attack where the attacker uses a batch script as described in the third bullet point above.

The Staging – Batch Script

The first stage of the hijack starts with the threat actor dropping three files, a batch script, a PowerShell script, and an Avast driver, within the target system's "C:\Windows" and

CONTACT US

First Name: *

Last Name: *

Business Email: *

Business Phone:

Job Title: *

Company: *

Location: *

--Please select--

How may we help you?

The Obfuscation – PowerShell Loader Script

The PowerShell script contains multiple layers of obfuscation which, when executed, decodes its contents, and rebuilds the controller. Once the PowerShell script finishes rebuilding the controller, it utilizes Windows Application Programming Interfaces (APIs) to load and execute the controller in memory.

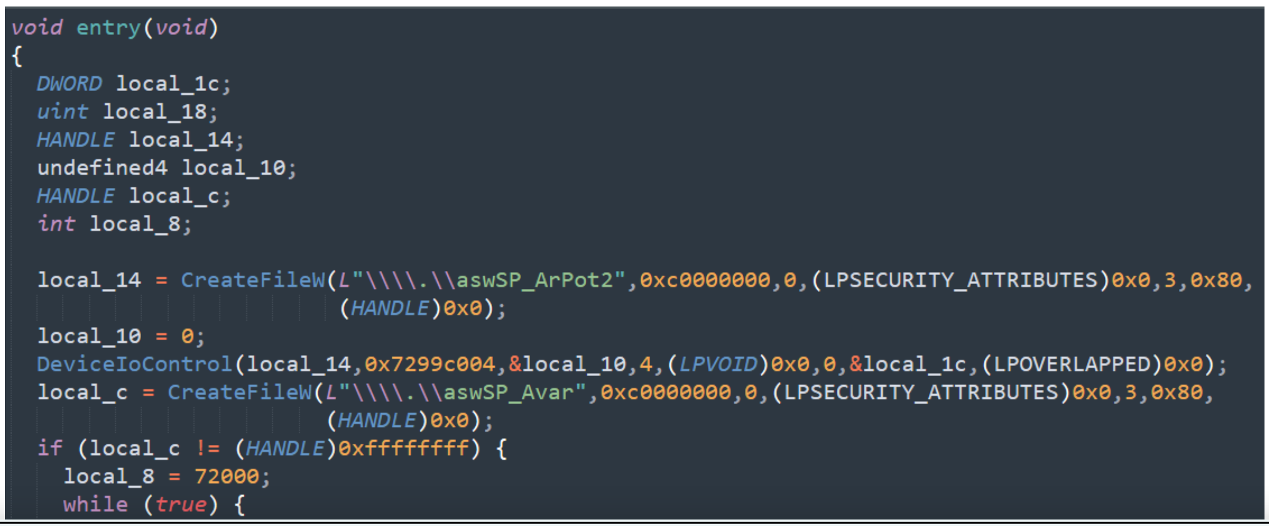
```
 Add-Type -TypeDefinition @'
     using System;
     using System.Diagnostics;
     using System.Runtime.InteropServices;
     public static class RANDOMSTRING1 {
          [DllImport("kernel32.dll")]
          public static extern IntPtr VirtualAlloc(IntPtr RANDOMSTRING2, uint RANDOMSTRING3,
uint RANDOMSTRING4, uint RANDOMSTRING5);

          [DllImport("PowrProf.dll")]
          public static extern IntPtr EnumPwrSchemes(IntPtr RANDOMSTRING6, IntPtr
RANDOMSTRING7);
     }
 '@

 Function ouBmwjaLNIuXYiiWYYxZt() {
 return (([regex]::Matches('[Redacted_Base64_String…]
```

The Controller – Malicious Portable Executable (PE)

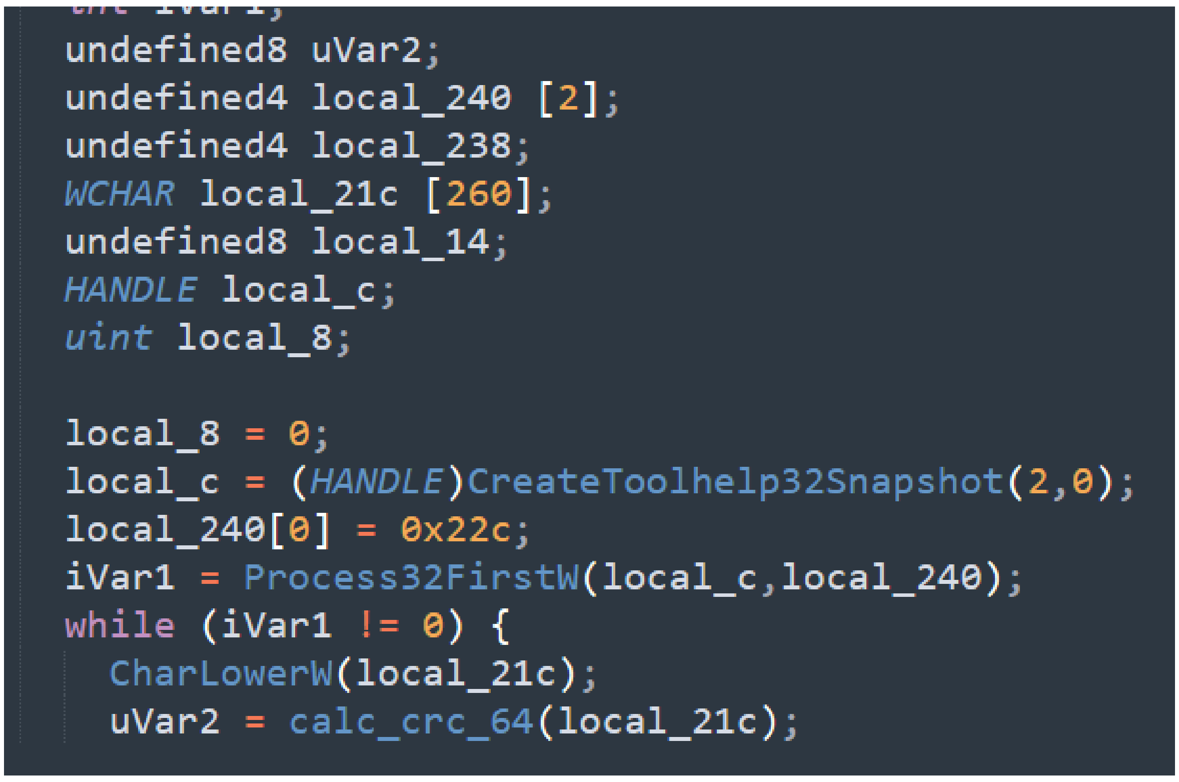
The small (~5KB in size) PE loaded into memory proved to be simple, yet effective. The executable is designed to collate a list of actively running processes, then compare them to an obfuscated hardcoded list of CRC64 checksum values of AV and EDR processes names. If any process name directly correlates to an entry in the hardcoded list, an I/O Control (IOCTL) code is sent to the Avast driver, resulting in the termination of the process.

Disassembling the sample on Ghidra provides insight into the hashing and comparison functions of the controller:

1. The initial function creates a handle to reference the recently installed Avast driver via the *CreateFileW* API. If the driver handle returns as valid, the executable calls a function to find and terminate processes.

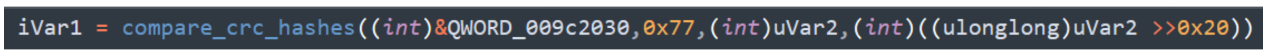


```
void entry(void)
{
  DWORD local_1c;
  uint local_18;
  HANDLE local_14;
  undefined4 local_10;
  HANDLE local_c;
  int local_8;

  local_14 = CreateFileW(L"\\\\.\\aswSP_ArPot2",0xc0000000,0,(LPSECURITY_ATTRIBUTES)0x0,3,0x80,
                         (HANDLE)0x0);
  local_10 = 0;
  DeviceIoControl(local_14,0x7299c004,&local_10,4,(LPVOID)0x0,0,&local_1c,(LPOVERLAPPED)0x0);
  local_c = CreateFileW(L"\\\\.\\aswSP_Avar",0xc0000000,0,(LPSECURITY_ATTRIBUTES)0x0,3,0x80,
                        (HANDLE)0x0);
  if (local_c != (HANDLE)0xffffffff) {
    local_8 = 72000;
    while (true) {
```



```
  int iVar1;
  undefined8 uVar2;
  undefined4 local_240 [2];
  undefined4 local_238;
  WCHAR local_21c [260];
  undefined8 local_14;
  HANDLE local_c;
  uint local_8;

  local_8 = 0;
  local_c = (HANDLE)CreateToolhelp32Snapshot(2,0);
  local_240[0] = 0x22c;
  iVar1 = Process32FirstW(local_c,local_240);
  while (iVar1 != 0) {
    CharLowerW(local_21c);
    uVar2 = calc_crc_64(local_21c);
```

3. The executable then cycles through the hardcoded list of CRC64 checksum values (QWORD_009c2030), each of which represents the name of known AV or EDR processes. In the sample discussed in this article, the hardcoded list contained 119 (0x77) CRC64 checksum values.



```
iVar1 = compare_crc_hashes((int)&QWORD_009c2030,0x77,(int)uVar2,(int)((ulonglong)uVar2 >>0x20))
```

4. If the sample finds a match, it calls the Avast process termination function passing the handle of the Avast driver, and the matching process ID.

5. The *DeviceIoControl* API is called, which sends the *0x9988c094* IOCTL code to the Avast driver, along with the process ID. This results in the Avast driver terminating the process at Kernel level, bypassing tamper protection implemented in most AV and EDR products.

The Kill – Avast IOCTL Code

Figure 1. IOCTL code comparison within Avast driver's code, which calls its process terminating function

Figure 2. Avast driver's process terminating function

This IOCTL code and function can be found on multiple versions of the *aswArPot.sys* Avast driver, including the version distributed on Avast products as of December 2021. However, it was confirmed, through behavioral analysis, that the latest distributed versions of the Avast driver are not susceptible to this abuse. Upon contacting the Avast Bug Bounty team, we have

Description: **Avast Anti Rootkit**

Original Name: **aswArPot.sys**

Internal Name: **aswArPot**

File Version: **21.1.187.0**

Date signed: **2021-02-01 14:09:00**

The Targets

Different implementations of this driver's abuse, found either on VirusTotal or on the engagements, contain different lists of targeted processes.

- The smallest PowerShell script implementation targets only one specific process.
- Three "early" versions of the PE found on VirusTotal contained clear-text strings of the targeted processes, along with the transparently named PDB path "*F:\\Source\\WorkNew19\\KillAV\\Release\\KillAV.pdb*". These versions contained 53, 72 and 88 targeted processes with the PE compilation timestamps of October 28th, November 2nd and November 3rd, 2021, respectively.
- The PE with the longest target list of all, found during the Cuba ransomware incident, contained 110 unique targeted processes (after removing duplicates) represented as CRC64 checksum values. It also contained the newest PE compilation timestamp of all.

Utilizing the HashDB API service from OpenAnalysis[3], we were able to recover the clear-text strings corresponding to the hardcoded CRC64 checksums of the latter sample mentioned above. The list contains process names from well-known AV and EDR vendors, which include, amongst others, processes names from SentinelOne®, Cylance®, Avast®, Carbon Black®, Sophos®, McAfee®, and Malwarebytes®.

Below is the list of 110 targeted processes found in the latest PE:

agentsvc.exe	mfemms.exe	SophosSafestore64.exe
alsvc.exe	msmpeng.exe	sophosui.exe
avastsvc.exe	notifier.exe	ssdvagent.exe
avastui.exe	ntrtscan.exe	sspservice.exe
avp.exe	paui.exe	svcgenerichost.exe
avpsus.exe	pccntmon.exe	swc_service.exe
bcc.exe	psanhost.exe	swi_fc.exe
bccavsvc.exe	psuamain.exe	swi_service.exe
ccsvchst.exe	psuaservice.exe	tesvc.exe

dsa.exe	sbamtray.exe	sophossafestore.exe
efrservice.exe	sbpimsvc.exe	sophoslivequeryservice.exe
epam_svc.exe	scanhost.exe	sophososquery.exe
epwd.exe	sdcservice.exe	sophosfimservice.exe
hmpalert.exe	SEDService.exe	sophosmtrextension.exe
hostedagent.exe	sentinelagent.exe	sophoscleanup.exe
idafserverhostservice.exe	SentinelAgentWorker.exe	sophos ui.exe
iptray.exe	sentinelhelperservice.exe	cloudendpointservice.exe
klnagent.exe	sentinelservicehost.exe	cetasvc.exe
logwriter.exe	sentinelstaticenginescanner.exe	endpointbasecamp.exe
macmnsvc.exe	SentinelUI.exe	wscommunicator.exe
macompatsvc.exe	sepagent.exe	dsa-connect.exe
masvc.exe	sepWscSvc64.exe	responseservice.exe
mbamservice.exe	sfc.exe	epab_svc.exe
mbcloudea.exe	smcgui.exe	fsagentservice.exe
mcsagent.exe	SophosCleanM64.exe	endpoint agent tray.exe
mcsclient.exe	sophosfilescanner.exe	easervicemonitor.exe
mctray.exe	sophosfs.exe	aswtoolssvc.exe
mfeann.exe	SophosHealth.exe	avwrapper.exe
mfemactl.exe	SophosNtpService.exe	

Future Functionality?

The PE found during the Cuba ransomware incident also contains a second smaller chunk of CRC64 checksums, which correspond to the names of three specific ransomware executables utilized by Cuba Ransomware: “a.exe”, “anet.exe” and “aus.exe”. These checksums sit next to unutilized strings “**/c del**“, “**>> NUL**” and “**\\system32\\cmd.exe**“. These strings are never referenced. Along with the recent iterations and enhancements on observed versions of this PE, these strings indicate that a potential future version of this executable could include a function to automatically delete the ransomware executables from disk.

Below is a list of publicly found samples:

File Name	SHA256 Hash
eset.ps1	8fcfa67e1fde51f7d99c3714f80b7672a0bb0d31c6cafdb5e7670b845d4dee98
A82.exe	4306c5d152cdd86f3506f91633ef3ae7d8cf0dd25f3e37bec43423c4742f4c42
KillAV.exe	aeb044d310801d546d10b247164c78afde638a90b6ef2f04e1f40170e54dec03
c5e3b725080712c175840c59a37a5daa.virus	f68cea99e6887739cd82865f9b973664117af14c1a25d4917eec25ce4b26a381

ATT&CK® Mapping

Execution

- T1059.001 – Command and Scripting Interpreter – PowerShell
- T1106 – Native API
- T1569.002 – System Services – Service Execution
- T1204.002 – User Execution – Malicious File

Defense Evasion

- T1458.002 – Abuse Elevation Control Mechanism – Bypass User Access Control
- T1140 – Deobfuscate/Decode Files or Information
- T1211 – Exploitation for Defense Evasion
- T1574.010 – Hijack Execution Flow – Service File Permissions Weakness
- T1562.001 – Impair Defense – Disable or Modify Tools
- T1036.005 – Masquerading – Match Legitimate Name or Location
- T1027.001 – Obfuscated Files or Information – Binary Padding
- T1027.002 – Software Packing
- T1055.002 – Process Injection – Portable Execution Injection
- T1218 – Signed Binary Proxy Execution

Discovery

- T1057 – Process Discovery

Acknowledgements: Special thanks to Aon's Cyber Solutions team member, Nhan Huynh, for assistance with revising content and ensuring accuracy.

[1] How DoppelPaymer Hunts and Kills Windows Processes, https://www.crowdstrike.com/blog/how-doppelpaymer-hunts-and-kills-windows-processes

[2] Signed Binary Proxy Execution – T1218, https://attack.mitre.org/techniques/T1218/

[3] https://hashdb.openanalysis.net/

Change Cookie Preferences | Careers | Contact Us | Client Login | Privacy | Legal | Cookies