ModPipe POS Malware: New Hooking Targets Extract Card Data

kroll.com/en/insights/publications/cyber/modpipe-pos-malware-new-hooking-targets-extract-card-data

Send Message Send Message
Thank you

One of our experts will contact you shortly.

Sorry, something went wrong :(Please try again later!

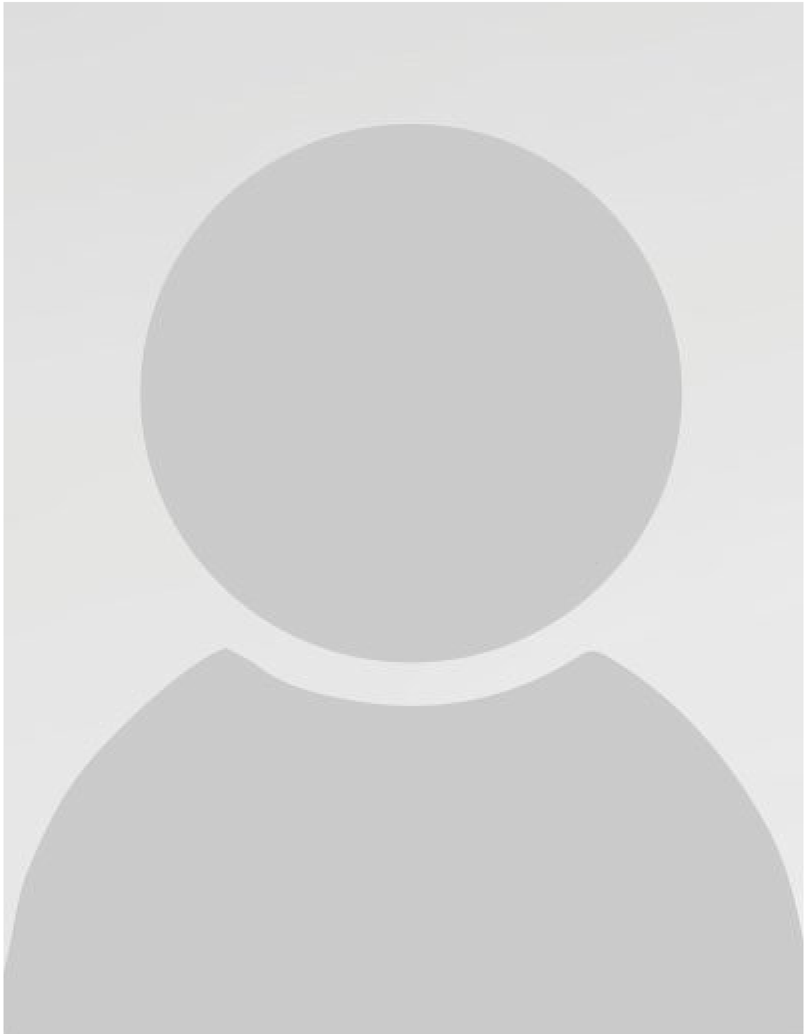
Please try again later!

Contact details

I would like to receive periodic news, reports, and invitations from Kroll, a Duff & Phelps.
chevron Path 2

Cyber Risk

Thu, Jun 2, 2022



Sean Straw

Kroll’s incident responders have seen threat actor groups becoming increasingly sophisticated and elusive in the tactics, techniques and procedures they employ to steal payment card data. One common method is to “scrape” the Track 1 or Track 2 data stored on the card’s magnetic stripe, which provides the cardholder account and personal information criminals need to make fraudulent “card-not-present” (CNP) transactions. This no longer requires physical access to a card or using a rogue device near an ATM to skim and store card data. The ongoing migration of point of sale (POS) management systems to the cloud provides actors with a rapidly expanding attack surface that is rife with opportunities for exploitation.

The Kroll Malware Analysis and Reverse Engineering team has observed recent ModPipe malware activity targeting payment card information. ModPipe continues to use the JHook hooking module to steal card information when it is encrypted or decrypted on card processing servers. Kroll analyzed a recent sample that hooked the **memcpy** function within **rsaenh.dll**, a Microsoft cryptographic library. Previous reporting had identified **memcpy** as a possible hooking target for ModPipe. Included in the analysis is a Ghidra script created by Kroll analysts to quickly map process memory into the correct virtual address space using Volatility 3.

ModPipe Analysis

ModPipe Background

As first reported in November 2020, ModPipe is a modular backdoor that runs within system processes and uses named pipes to communicate between modules. ModPipe has historically targeted devices running ORACLE MICROS Restaurant Enterprise Series (RES) 3700 POS, but the capabilities it employs could be used to target other systems. The persistent loader is an executable that unpacks an inner payload before injecting the core ModPipe module into a system process. In Kroll's analysis, this process was typically **lsass.exe**, though **wininit.exe** and **services.exe** have also been identified as additional targets.

Once injected, it has been noted that the unpacked core module will attempt to connect to legitimate URLs to confirm network connectivity before establishing command-and-control (C2) connections and then downloading and executing additional modules from a remote, actor-controlled server. Along with the C2 and injection modules, the following modules have been previously identified:

- GetMicInfo – A module to steal database passwords and sensitive information related to MicrosPOS software
- ModScan – A module designed to scan IP addresses
- ProcList – Gets a list of running processes and loaded modules

At the time of initial analysis, analysts believed that additional modules most likely exist, but the modules had not yet been seen in the wild.

JHook Module Background

In April 2021, researchers reported that they had identified two additional modules used by ModPipe malware, including the JHook module. The JHook module is designed to replace legitimate function calls within a process with malicious JHook functions that are designed to target two legitimate functions for scraping:

- **CryptDecrypt** – a decryption function within the **advapi32.dll** library
- **memcpy** – a common C function to copy data from one memory location to another

When the JHook module is run, these functions can be replaced by scraping functions which call the original legitimate function after reviewing the data and looking for Track 1 and Track 2 card data. Once the target data is located, the JHook scraping function copies the identified track data to a buffer before sending the data to the core module for exfiltration.

The researchers reported that the configuration they had identified targeted the **CryptDecrypt** function within **dbsecurity2.dll** using the format provided in Figure 1.

```
<HOOK_TYPE>{
<TARGET_MODULE>|<HOOK_TARGET_DLL>:<HOOK_TARGET_FUNC>:<HOOK_FUNC>
```

Figure 1 – JHook format used to target “CryptDecrypt” function within dbsecurity2.dll

Our Findings

We have analyzed an instance of ModPipe that was configured to hook the **memcpy** function within **rsaenh.dll**, a Microsoft cryptographic library, in the **CCS.exe** process of a Micros RES POS system. The configured hook is provided in Figure 2.

```
HookImport{
\rsaenh.dll|ntdll.dll:memcpy:JHook_memcpy
```

```
HookImport{
\rsaenh.dll|ntdll.dll:memcpy:
JHook_memcpy
```

Figure 2 – JHook Format Identified by Kroll to Hook “memcpy” Function within “rsaenh.dll”

The module is configured to hook the memcpy function of **ntdll.dll** within **rsaenh.dll**. Interestingly, Kroll has not yet identified why or where the rsaenh.dll library is used within the MicroPOS **CCS.exe** process. Dependency analysis of the **CCS.exe** executable did not identify any locations where the executable or its libraries listed a function from **rsaenh.dll**. Further analysis is underway to identify the specific use of **rsaenh.dll** within the MicroPOS **CCS.exe** process. However, as the library provides cryptographic functionality, it is reasonable to assume that some portion of the library is used for encrypting or decrypting track data. Kroll identified stolen track data within memory to confirm that despite the unclear origin of **rsaenh.dll**, the module had found some success in stealing track data previously.

To hook the **memcpy** function, the malware replaces the entry for the function within the Import Address Table (IAT) of **rsaenh.dll** with the address of a trampoline function created by the module in a new section of readable/writeable/executable memory (Figure 3).

```
                    fn_Trampoline_memcpy
05591000 68 10 46        PUSH       _memcpy
         ba 77
05591005 ff 15 14        CALL       dword ptr [->f_Trampoline_memcpy_return]
         10 59 05
0559100b e9 80 27        JMP        f_JHook_memcpy
         b1 fd
                    f_Trampoline_memcpy_return
05591010 c3              RET
05591011 00 00 00        db[3]
                    PTR_f_Trampoline_memcpy_return_05591014
05591014 10 10 59 05     addr       f_Trampoline_memcpy_return
```

Figure 3 – The Trampoline Function

When called, the scraping function checks to ensure that the **count** argument of **memcpy** is at least 10 and that a **src** argument has been provided before inserting an exception handler into the exception handler's chain (Figure 4).

```
                    f_JHook_memcpy

030a3790 83 7c 24        CMP        dword ptr [ESP + p_src],0x0
         0c 00
030a3795 0f 84 d0        JZ         f_JHook_CryptDecrypt::return
         00 00 00
030a379b 83 7c 24        CMP        dword ptr [ESP + count],10
         10 0a
030a37a0 0f 82 c5        JC         f_JHook_CryptDecrypt::return
         00 00 00
030a37a6 68 6b 38        PUSH       f_JHook_CryptDecrypt::return
         0a 03
030a37ab 60              PUSHAD
030a37ac 9c              PUSHFD
030a37ad 89 e5           MOV        EBP,ESP
030a37af 64 ff 35        PUSH       dword ptr FS:[0x0]
         00 00 00 00
030a37b6 68 3e 39        PUSH       f_jhook_exception_handler
         0a 03
030a37bb 6a 00           PUSH       0x0
030a37bd 64 89 25        MOV        dword ptr FS:[0x0],ESP
```

Figure 4 – The Beginning of the "JHook_memcpy" Function

The exception handler increments a counter before setting the instruction counter to be near the end of the function. This effectively catches and ignores any errors from the scraping routine (Figure 5).

```
                          f_jhook_exception_handler
030a393e ff 05 58             INC         dword ptr [dword_error_count]
         54 0a 03
030a3944 8b 44 24 0c          MOV         EAX,dword ptr [ESP + ContextRecord]
                          Sets ContextRecord->Eip value
030a3948 c7 80 b8             MOV         dword ptr [EAX + 0xb8],f_JHook_CryptDecrypt::return_restore_reg
         00 00 00
         5e 38 0a 03
030a3952 31 c0                XOR         EAX,EAX
030a3954 c3                   RET
```

Figure 5 – The Registered Exception Handler

As the earlier researchers noted, after copying the identified data to an internal buffer, the scraping function increments a counter common to both the **JHook_memcpy** and **JHook_CryptDecrypt** functions. At the time of this writing, the internal buffer within memory had tracked approximately double the number of cards identified in the initial Common Point of Purchase notification.

Mapping Memory within Ghidra

As part of Kroll's analysis, analysts reviewed a memory dump of a system with ModPipe and the JHook module active. To facilitate the reverse engineering of the module, Kroll developed a short Ghidra script to take the output of the Volatility 3 **vadinfo** module and quickly map the memory to the correct locations within the Ghidra CodeBrowser.

When a process of interest has been identified, all the process memory can be dumped using the Volatility 3 **vadinfo** module to a directory, with the standard output of the module written to a text file. The module includes the VPN start and end for each memory section, source file when available and the memory protection (Figure 6).

```
py E:\tools\volatility3\vol.py -f .\memory_dump -o 644_vadinfo
windows.vadinfo --pid 644 --dump > vadinfo.out
```

Figure 6 – Command to dump memory using the vadinfo module

After opening a section of memory and setting the language for the CodeBrowser, the Ghidra script can be run, prompting for the output directory (**644_vadinfo**) and output file (**vadinfo.out**). The script will then create the appropriate memory map for all available dump files (Figure 7).

Memory Map - Image Base: 030a0000

Memory Blocks

Name	Start	End	Length	R	W	X	Volatile	Overlay	Type	Initialized	Byte Source	Source
pid.644.vad.0x9a000...	009a0000	009a1ffe	0x1fff	☑	☑	☐	☐	☐	Default	☑	File: pid.644.vad.0x9a0000-0x...	
pid.644.vad.0x9b000...	009b0000	009bfffe	0xffff	☑	☑	☐	☐	☐	Default	☑	File: pid.644.vad.0x9b0000-0x...	
pid.644.vad.0x9c000...	009c0000	009c7ffe	0x7fff	☑	☑	☐	☐	☐	Default	☑	File: pid.644.vad.0x9c0000-0x...	
pid.644.vad.0x9d000...	009d0000	009e8ffe	0x18fff	☑	☐	☐	☐	☐	Default	☑	File: pid.644.vad.0x9d0000-0x...	
pid.644.vad.0x9f000...	009f0000	009f3ffe	0x3fff	☑	☐	☐	☐	☐	Default	☑	File: pid.644.vad.0x9f0000-0x9...	
pid.644.vad.0xa0000...	00a00000	00a00ffe	0xfff	☑	☐	☐	☐	☐	Default	☑	File: pid.644.vad.0xa00000-0x...	
pid.644.vad.0xa1000...	00a10000	00a75ffe	0x65fff	☐	☑	☑	☐	☐	Default	☑	File: pid.644.vad.0xa10000-0x...	\MICROS\Common\Bin\CCS.exe
pid.644.vad.0xa8000...	00a80000	00abfffe	0x3ffff	☑	☑	☐	☐	☐	Default	☑	File: pid.644.vad.0xa80000-0x...	
pid.644.vad.0xac000...	00ac0000	00bbfffe	0xfffff	☑	☑	☐	☐	☐	Default	☑	File: pid.644.vad.0xac0000-0x...	
pid.644.vad.0xbc000...	00bc0000	00bc0ffe	0xfff	☑	☑	☐	☐	☐	Default	☑	File: pid.644.vad.0xbc0000-0x...	
pid.644.vad.0xbd000...	00bd0000	00bd0ffe	0xfff	☑	☐	☐	☐	☐	Default	☑	File: pid.644.vad.0xbd0000-0x...	
pid.644.vad.0xbe000...	00be0000	00be0ffe	0xfff	☑	☐	☐	☐	☐	Default	☑	File: pid.644.vad.0xbe0000-0x...	
pid.644.vad.0xbf000...	00bf0000	00bf0ffe	0xfff	☑	☐	☐	☐	☐	Default	☑	File: pid.644.vad.0xbf0000-0xb...	
pid.644.vad.0xc0000...	00c00000	00dffffe	0x1fffff	☑	☑	☐	☐	☐	Default	☑	File: pid.644.vad.0xc00000-0x...	
pid.644.vad.0xe0000...	00e00000	02e00ffe	0x2000fff	☑	☑	☐	☐	☐	Default	☑	File: pid.644.vad.0xe00000-0x...	
pid.644.vad.0x2e100...	02e10000	02e4fffe	0x3ffff	☑	☑	☐	☐	☐	Default	☑	File: pid.644.vad.0x2e10000-0...	
pid.644.vad.0x2e500...	02e50000	02e50ffe	0xfff	☑	☐	☐	☐	☐	Default	☑	File: pid.644.vad.0x2e50000-0...	
pid.644.vad.0x2e600...	02e60000	02e60ffe	0xfff	☑	☐	☐	☐	☐	Default	☑	File: pid.644.vad.0x2e60000-0...	
pid.644.vad.0x2e700...	02e70000	02f6fffe	0xfffff	☑	☑	☐	☐	☐	Default	☑	File: pid.644.vad.0x2e70000-0...	
pid.644.vad.0x2fb00...	02fb0000	02fb0ffe	0xfff	☑	☐	☐	☐	☐	Default	☑	File: pid.644.vad.0x2fb0000-0x...	
pid.644.vad.0x2fc00...	02fc0000	02fcfffe	0xffff	☑	☑	☐	☐	☐	Default	☑	File: pid.644.vad.0x2fc0000-0x...	
pid.644.vad.0x2fd00...	02fd0000	03094ffe	0xc4fff	☑	☐	☐	☐	☐	Default	☑	File: pid.644.vad.0x2fd0000-0x...	\Windows\System32\locale.nls
ram	030a0000	030bbffe	0x1bfff	☑	☑	☑	☐	☐	Default	☑	File: pid.644.vad.0x30a0000-0...	Binary Loader
pid.644.vad.0x30c00...	030c0000	031bfffe	0xfffff	☑	☑	☐	☐	☐	Default	☑	File: pid.644.vad.0x30c0000-0...	

Figure 7 – Memory Map After Running the Script

This enables analysts to easily cross-reference code from different sections of memory, such as the JHook trampoline, the JHook **memcpy** function and the modified IAT within **rsaenh.dll** (Figure 8).

```
                  undefined fn_Trampoline_memcpy()
     undefined      AL:1           <RETURN>
                  fn_Trampoline_memcpy                    XREF[6]:     057859a4(*), 05785a08(*),
                                                                       057861fc(*), 05786220(*),
                                                                       fn_Trampoline_memcpy:7264d012(T),
                                                                       fn_Trampoline_memcpy:7264d012(j),
                                                                       72665280(*)
05591000 68 10 46     PUSH      _memcpy
         ba 77
05591005 ff 15 14     CALL      dword ptr [->f_Trampoline_memcopy_return]      undefined f_Trampoline_memcopy_r
         10 59 05
0559100b e9 80 27     JMP       f_JHook_memcpy                                 void f_JHook_memcpy(void * retur
         b1 fd
```

Figure 8 – Trampoline function showing references to different sections of memory mapped by the script

Detection and Mitigation

As of this publication, ModPipe is considered advanced, but not commonly used. There has been limited research on the malware, with no proof of a parent group or what threat actors are doing with the stolen data after exfiltration. ModPipe relies on elevated privileges and would generally fall within the actions-on-objectives within the Kroll Intrusion Lifecycle.

Hardening steps like network segmentation, enabling multifactor authentication and utilizing endpoint detection and monitoring that can identify malware such as ModPipe can help prevent threat actors from achieving the necessary level of access to deploy ModPipe.

Endpoint detection can also be configured to look for unusual network communications from system processes.

Ensuring your organization has good overall cybersecurity hygiene can prevent a threat actor from gaining access to your network, thus preventing malware from being placed on your server. These 10 essential cybersecurity controls can help improve your security posture and increase your cyber resilience. For more information, contact us via our 24x7 cyber incident response hotlines or connect with us through our Contact Us page.

Indicators of Compromise

Type	Value
Filename	Mljpmi.exe
MD5	34d03f1900e759e2a670c42d7af14a3d
SHA1	10ea1722991a787fef9a3a2545f7bcc7d0b4f42c
SHA256	74fd2d70b085d70bd883854ae9ebd1684f3de824995acd4df1993d3d469c62ec
URL	ouidji12345.ddns[.]net/gettime.html
IP Address	185.206.147.15
IP Address	89.163.246.135

Type	Value
Filename	Mljpmi.exe
MD5	34d03f1900e759e2a670c42d7af14a3d
SHA1	10ea1722991a787fef9a3a2545f7bcc7d0b4f42c
SHA256	74fd2d70b085d70bd883854ae9ebd1684f3de824995acd4df1993d3d469c62ec
URL	ouidji12345.ddns[.]net/gettime.html
IP Address	185.206.147.15
IP Address	89.163.246.135

Kroll's expertise establishes whether data was compromised and to what extent. We uncover actionable information, leaving you better prepared to manage a future incident.

Data Collection and Preservation

Improve investigations and reduce your potential for litigation and fines with the strict chain-of-custody protocol our experts follow at every stage of the data collection process.