



# MalumPOS

## History and Characteristics

TrendLabs Security Intelligence Blog

Jay Yaneza  
Trend Micro Threats Analyst

June 2015

# Contents

Introduction .....	3
Threat Details .....	3
<i>Attack Methodology</i> .....	3
<i>MalumPOS Capabilities</i> .....	3
<i>Use of Regular Expressions</i> .....	6
Stealth Attempts.....	8
Improved Variants .....	9
Solutions and Recommendations.....	11
Indicators.....	11

## TREND MICRO LEGAL DISCLAIMER

The information provided herein is for general information and educational purposes only. It is not intended and should not be construed to constitute legal advice. The information contained herein may not be applicable to all situations and may not reflect the most current situation. Nothing contained herein should be relied on or acted upon without the benefit of legal advice based on the particular facts and circumstances presented and nothing herein should be construed otherwise. Trend Micro reserves the right to modify the contents of this document at any time without prior notice.

Translations of any material into other languages are intended solely as a convenience. Translation accuracy is not guaranteed nor implied. If any questions arise related to the accuracy of a translation, please refer to the original language official version of the document. Any discrepancies or differences created in the translation are not binding and have no legal effect for compliance or enforcement purposes.

Although Trend Micro uses reasonable efforts to include accurate and up-to-date information herein, Trend Micro makes no warranties or representations of any kind as to its accuracy, currency, or completeness. You agree that access to and use of and reliance on this document and the content thereof is at your own risk. Trend Micro disclaims all warranties of any kind, express or implied. Neither Trend Micro nor any party involved in creating, producing, or delivering this document shall be liable for any consequence, loss, or damage, including direct, indirect, special, consequential, loss of business profits, or special damages, whatsoever arising out of access to, use of, or inability to use, or in connection with the use of this document, or any errors or omissions in the content thereof. Use of this information constitutes acceptance for use in an “as is” condition.

## Introduction

Point-of-sale (PoS) malware is now a major threat for organizations in the retail and hospitality industries. The newest PoS malware family to target these sectors is the *MalumPOS* family, which has been configured to steal information from various applications used in the hospitality industry.

The currently observed configuration of MalumPOS is to collect data from PoS systems running the Oracle® MICROS® Platform. Oracle claims that this platform is in use in over 300,000 locations globally.<sup>1</sup> Many of these sites are located in the United States, putting millions of customers in North America at risk of financial fraud.

## Threat Details

### *Attack Methodology*

Various means can be used to plant MalumPOS onto affected machines, which will not be discussed in this brief. Whatever the case may be, it is installed as a malicious service *NVIDIA® Display Driver* (sometimes stylized as *NVIDIA Display Driv3r*).

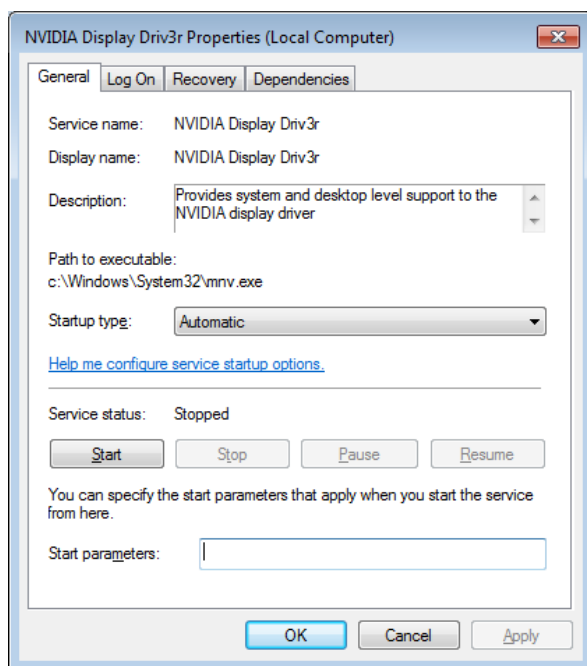


Figure 1. Installed service of MalumPOS

### *MalumPOS Capabilities*

MalumPOS is written in the Delphi programming language. It can monitor running processes on an affected system, and scrape the memory contents of targeted processes.

<sup>1</sup> Oracle Corporation. (2014). *Oracle and MICROS Systems*. Last accessed on June 05, 2015, <http://www.oracle.com/us/corporate/acquisitions/micros/index.html>.

The targeted processes, as well as the location of the saved information, are set via a downloaded configuration file:

- Up to 100 processes can be targeted
- In the samples we've seen, the scraped credit card information is saved in the file named `C:\Windows\system32\nsvc.dll`. Like the scraper itself, this is named to appear to be part of NVIDIA drivers. The contents of the file are encrypted.

```

if ( DuplicateHandle(hCurrentProcess1, hCurrentThread, hCurrentProcess, hServiceHandle, 0, 0, 2u) )
{
    *hServiceEvent = CreateEventA(0, -1, 0, 0);
    if ( *hServiceEvent )
    {
        UpdateServiceStatus(4u);
        dwThreadId = 0;
        InitService(); // Initialize (Load configuration)
        while ( WaitForSingleObject(*hServiceEvent, 0x1388u) == 258 )
        {
            // Start a thread for memory scrapper
            hScrapperThread = CreateScrapperThread(0, 0, (int)MemoryScrapper, (DWORD *)&dwThreadId, 0, 0);
            CloseHandle_0(hScrapperThread);
            WaitForSingleObject(hScrapperThread, 0xFFFFFFFF);
        }
        ReleaseTStringList();
        CloseHandle_0(*lpTargetHandle);
        *lpTargetHandle = 0;
        CloseHandle_0(*hServiceEvent);
        *hServiceEvent = 0;
        result = UpdateServiceStatus(1u);
    }
}

```

Figure 2. Main thread of MalumPOS

The configuration file is loaded at the beginning, as can be seen in the screenshot below.

004133F9	75 F2	JNE SHORT 29e45b1b9bdbe9cbc6da7e52259c214.004133ED	
004133FE	B8 A0B94100	MOV EAX, OFFSET 29e45b1b9bdbe9cbc6da7e52259c214.00413400	
00413400	BA 4C354100	MOV EDX, 29e45b1b9bdbe9cbc6da7e52259c214.0041354C	ASCII "DGH"
00413405	E8 4E0FFFFFFF	CALL 29e45b1b9bdbe9cbc6da7e52259c214.00404358	
0041340A	B8 A4B94100	MOV EAX, OFFSET 29e45b1b9bdbe9cbc6da7e52259c214.0041340F	
0041340F	BA 58354100	MOV EDX, 29e45b1b9bdbe9cbc6da7e52259c214.00413558	ASCII "DELAM&RPMs"
00413414	E8 3F0FFFFFFF	CALL 29e45b1b9bdbe9cbc6da7e52259c214.00404358	
00413419	C705 A8E94100	MOV DWORD PTR [29e45b1b9bdbe9cbc6da7e52259c214.41E9], 0	
00413423	B8 ACB94100	MOV EAX, OFFSET 29e45b1b9bdbe9cbc6da7e52259c214.00413428	
00413428	E8 D70FFFFFFF	CALL 29e45b1b9bdbe9cbc6da7e52259c214.00404304	
0041342D	B8 B0B94100	MOV EAX, OFFSET 29e45b1b9bdbe9cbc6da7e52259c214.00413432	
00413432	BA 6C354100	MOV EDX, 29e45b1b9bdbe9cbc6da7e52259c214.0041356C	ASCII "nsvc.dll"
00413437	E8 1C0FFFFFFF	CALL 29e45b1b9bdbe9cbc6da7e52259c214.00404358	
0041343C	B8 E4B94100	MOV EAX, OFFSET 29e45b1b9bdbe9cbc6da7e52259c214.00413441	
00413441	E8 BE0FFFFFFF	CALL 29e45b1b9bdbe9cbc6da7e52259c214.00404304	
00413446	B8 B8B94100	MOV EAX, OFFSET 29e45b1b9bdbe9cbc6da7e52259c214.0041344B	
0041344B	BA 80354100	MOV EDX, 29e45b1b9bdbe9cbc6da7e52259c214.00413580	ASCII "TMPLOCs"
00413450	E8 030FFFFFFF	CALL 29e45b1b9bdbe9cbc6da7e52259c214.00404358	
00413455	B8 BCB94100	MOV EAX, OFFSET 29e45b1b9bdbe9cbc6da7e52259c214.0041345A	
0041345A	BA 90354100	MOV EDX, 29e45b1b9bdbe9cbc6da7e52259c214.00413590	ASCII "frmweb.exe"
0041345F	E8 F40FFFFFFF	CALL 29e45b1b9bdbe9cbc6da7e52259c214.00404358	
00413464	E8 8BF9FFFF	CALL 29e45b1b9bdbe9cbc6da7e52259c214.00412DF4	
00413469	E8 C2F9FFFF	CALL 29e45b1b9bdbe9cbc6da7e52259c214.00412E30	
0041346E	C605 7CB84100	MOV BYTE PTR [29e45b1b9bdbe9cbc6da7e52259c214.41B87], 0	

Figure 3. Configuration loading of MalumPOS



0041B88A	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	1E	10	.....▲▶
0041B89A	15	19	02	07	0B	14	1D	06	0F	0C	12	16	1B	1F	04	08		Ⓢ↓θ·Ⓢπ#*Ⓢ.ⓈL+Ⓢ♦□
0041B8AA	0D	11	1A	03	18	01	0A	13	1C	05	0E	17	00	09	87	AA		.◀+◀+θ.!!L◀ⓈⓈ.◀◀
0041B8BA	C6	E9	25	49	65	88	A4	C7	E3	26	42	66	82	A5	C1	E4		fθ%Ieēñ  π&Bfēñ+Σ
0041B8CA	20	43	60	83	9F	C2	DE	21	3D	7D	A0	BC	DF	4C	68	8C		C'āfτ  †=)ā'■Lh i
0041B8DA	A8	CB	E7	2A	46	69	85	A9	C5	E8	24	47	63	86	A2	E2		◀πr*#Fīār†ⓈⓈGōāōΓ
0041B8EA	41	64	80	A3	C0	FF	5E	81	9D	DD	3C	5F	7B	9E	BA	FA		AdQū ◀ ^ūⓈ◀ ◀ (A
0041B8FA	59	7C	98	BB	FB	3E	5A	99	D8	FC	38	5B	77	9A	B6	D9		Y ūq r>Z0τ'8[wū
0041B90A	F5	39	55	78	94	B7	D3	F6	32	56	72	95	B1	D4	F0	33		J9Uxōη μ+2UxōⓈⓈ=3
0041B91A	4F	73	8F	B2	CE	F1	2D	50	6D	90	AC	CF	EB	2E	4A	8A		OsAⓈⓈ†Ⓢ-Pmēk†Ⓢ.Jē
0041B92A	AD	C9	EC	28	4B	75	B5	F4	37	53	76	92	D2	31	54	93		†r◀(Ku††7SvEπ1Tō
0041B93A	71	B0	8E	6C	AB	89	C8	67	A6	E5	29	45	84	A7	C3	E6		◀ⓈA ⓈēhⓈgⓈσ)Eāō hγ
0041B94A	22	62	A1	C4	E0	23	3F	7F	F9	58	44	61	7E	BE	E1	FD		"b i-◀#?θ·XDa"◀β²
0041B95A	40	5C	9B	DB	FE	3A	5D	79	9C	B8	F8	3B	57	7A	96	B9		θ\◀■◀:Jyē†°:Wzū
0041B96A	DC	97	D6	36	D7	DA	F7	D5	34	B3	51	6E	8B	AE	CA	ED		■ūπ6†r◀F4 Qn i◀◀◀
0041B97A	48	EA	F2	D0	2F	91	B4	D1	30	6F	EE	4D	70	AF	EF	2B		Hō²μ/◀†τQōeMπ>>n+
0041B98A	4E	CD	8D	CC	6A	2C	35	74	F3	52	6B	27	BD	BF	00	00		N= †j,St≤Rk'μj..

Figure 7. Look-up table used to decrypt the scraped data

The encryption attempts to hide the data on the system machine, as the stolen information requires decryption to be viewed by an analyst.

The memory dumping procedure itself is slightly different compared to other PoS threats. The procedure flows as follows.

- The CreateToolHelp32Snapshot would be dynamically imported and the API address would be saved for reuse.
- It searches all matched processes and save into a list.
- It would proceed to check the list one by one and scrape credit data into a secondary list; the data is encrypted at same time.
- If the secondary list isn't empty, it would proceed to write data into the configured data storage file (i.e., nvsvc.dll).

## Use of Regular Expressions

In order to detect potential credit card information, MalumPOS uses regular expressions (regexes) to search for strings that match credit card numbers and other information relevant to attackers.

Magnetic stripe cards store this information in a format defined by the ISO/IEC 7813:2006.<sup>2</sup> Two separate magnetic tracks are used, which requires the usage of two regexes. The use of regexes to gather information about credit card transactions is a common practice used by many PoS malware families.

<sup>2</sup> International Organization for Standardization. (2013). *Standards Catalogue*. "ISO/IEC 7813:2006." Last accessed on June 05, 2015, [http://www.iso.org/iso/home/store/catalogue\\_ics/catalogue\\_detail\\_ics.htm?csnumber=43317](http://www.iso.org/iso/home/store/catalogue_ics/catalogue_detail_ics.htm?csnumber=43317).

The regular expression to detect Track 1 information `((b|B)[0-9]{13,19})\^[A-Za-z\s]{0,30}\V[A-Za-z\s]{0,30}\^(1[1-9])((0[1-9])|(1[0-2]))[0-9\s]{3,50}[0-9]{1}` is illustrated below.

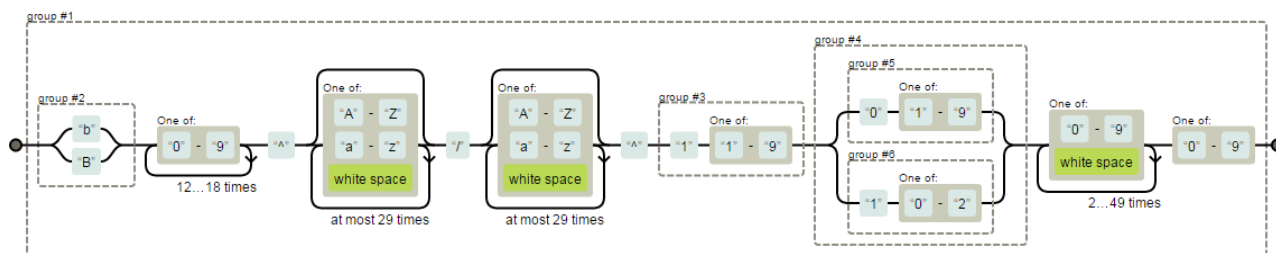


Figure 8. Regular expression used for Track 1 data, visualized with <http://regexper.com/>

It skips the Start sentinel (usually “%”) and immediately starts with the format code “B” to indicate credit/debit card. After that, it matches the primary account number (PAN) then matches the name. The next section (seen in Figure 8 as group #3) is the “expiration date” that effectively matches cards that expires between the years 2011 and 2019. The rest of the numbers (service code, discretionary data) are matched normally, and the end sentinel (generally “?”) is skipped.

For Track 2, the regex `((3-9){1}[0-9]{14,15}[D=](1[1-9])((0[1-9])|(1[0-2]))[0-9]{8,30})` is used. This is illustrated in the diagram below.

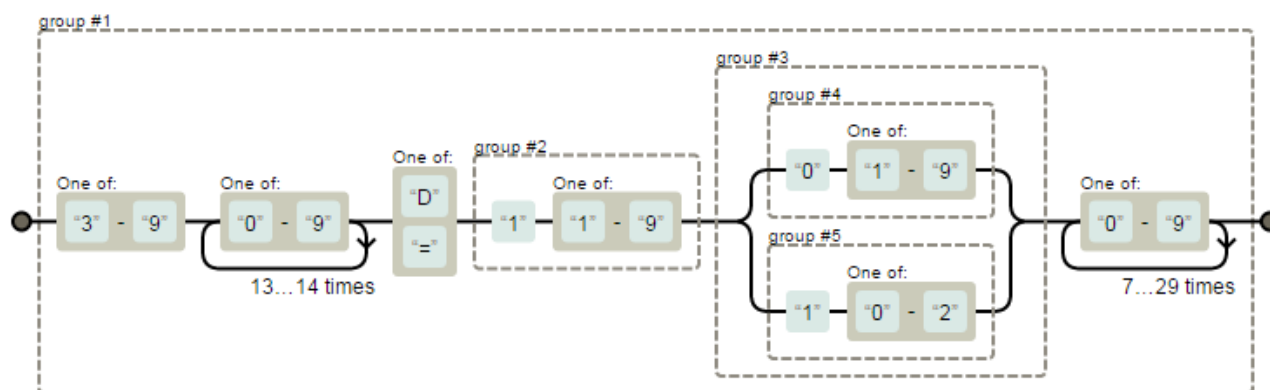


Figure 9. Regular expression used for Track 2 data, visualized with <http://regexper.com/>

It skips the start sentinel (usually “;”) and ensures that the start of the credit card number starts in a value between 3 and 9. This way, MalumPOS selectively looks for Visa (starts with 41), MasterCard (starts with 51), American Express (starts with 30, 34, or 37), Discover Cards (starts with 60 or 65), Diner’s Club (starts with 30, 36, or 38), and some JCB cards (starts with 35).

The “expiration date” (seen in Figure 9 as group #2) only matches cards that expires between the years 2011 and 2019. It then proceeds to evaluate the service code and discretionary data normally, and the end sentinel (generally “?”) is skipped from matching.

By modifying the ranges being used in the definition of the regular expression that is meant to match credit card data, the threat actors who use MalumPOS effectively validated the expiration dates of the credit cards, as well as the targeted specific credit card lines.

While regex expressions are commonly used by PoS malware, the specific expressions used here were first seen in the *Rdasrv* malware family. This is an older malware family that was documented in our earlier white paper on PoS RAM scraper malware.<sup>3</sup>

```

Track1_data dd 0FFFFFFFh, 69h
            db '((b|B)[0-9]{13,19}\^[A-Za-z\s]{0,30}\^[A-Za-z\s]{0,30}\^(1[1-9])('
            ; DATA >REF: search_for_CC_data+37Io
            db '(0[1-9])|(1[0-2]))[0-9\s]{3,50}[0-9]{1}',0
            align 4
Track2_data dd 0FFFFFFFh, 40h
            db '([3-9]{1}[0-9]{14,15}[0-9]{1}[1-9])((0[1-9])|(1[0-2]))[0-9]{8,30}',0
            ; DATA >REF: search_for_CC_data+85Io
            align 10h
off_4191D0 dd offset unk_4191D4 ; DATA >REF: open_process_and_read_mem+64Io
unk_4191D4 db 11h ; open_process_and_read_mem+149Io ...
            db 2
            db '.2'
            db 1

```

Figure 10. The regex of *Rdasrv* is the same with MalumPOS

The similarities between MalumPOS and *Rdasrv* became more apparent upon closer inspection:

1. The main scraper is installed as a service, and installed with an “install” switch
2. Selectively inspects processes that have been hardcoded (or, in this case, configured) within the binary
3. It calls `OpenProcess` to obtain a handle to the specified target process
4. There is an overlap with the target industry: *Rdasrv* is seen to target processes of PoS systems within the food services and hospitality industries; MalumPOS within the hospitality industry
5. As there is no data exfiltration functionality, the configured data storage file is probably collected by the threat actor with some other piece of malware.

This suggests that MalumPOS and *Rdasrv* are somehow linked, although this cannot be fully proven. What is clear is that the persons operating MalumPOS had prior information about their target’s environment as they are able to customize binaries based on the target’s PoS systems, plant them within the target’s environment, and manually collect the stored data.

## Stealth Attempts

MalumPOS uses various techniques in order to prevent itself from being detected. To some degree, these techniques are not particularly sophisticated and are an attempt to hide in “plain sight.” As mentioned earlier, MalumPOS uses filenames that are designed to make it look like part of legitimate NVIDIA software. It also uses the following techniques:

1. The files collected have an old time stamp: *1992-06-19 17:22:17*. While this attempt would be effective for incident responders who are looking for newer files dropped in the system, it made it very obvious once the file is analyzed.
2. Some of the APIs are loaded dynamically; imported APIs are processed via `GetProcAddress`. This is a visible and perhaps basic attempt to avoid evade-static analysis tools.

<sup>3</sup> Numaan Huq. (2014). *Trend Micro Security Intelligence*. “PoS RAM Scraper Malware: Past, Present, and Future.” Last accessed on June 05, 2015, <http://www.trendmicro.com/cloud-content/us/pdfs/security-intelligence/white-papers/wp-pos-ram-scraper-malware.pdf>.



```

bool GetMemoryDumpAPI()
{
    if ( !hLibrary )
    {
        hLibrary = GetModuleHandleA_1("kernel32.dll");
        if ( hLibrary )
        {
            pCreateToolhelp32Snapshot = (int (__stdcall *)(_DWORD, _DWORD))GetProcAddress_0(
                hLibrary,
                "CreateToolhelp32Snapshot");

            pHeap32ListFirst = (int)GetProcAddress_0(hLibrary, "Heap32ListFirst");
            pHeap32ListNext = (int)GetProcAddress_0(hLibrary, "Heap32ListNext");
            pHeap32First = (int)GetProcAddress_0(hLibrary, "Heap32First");
            *( _DWORD *)pHeap32Next = GetProcAddress_0(hLibrary, "Heap32Next");
            *( _DWORD *)pToolhelp32ReadProcessMemory = GetProcAddress_0(hLibrary, "Toolhelp32ReadProcessMemory");
            pProcess32First = (int (__stdcall *)(_DWORD, _DWORD))GetProcAddress_0(hLibrary, "Process32First");
            pProcess32Next = (int (__stdcall *)(_DWORD, _DWORD))GetProcAddress_0(hLibrary, "Process32Next");
            pProcess32FirstW = (int)GetProcAddress_0(hLibrary, "Process32FirstW");
            pProcess32NextW = (int)GetProcAddress_0(hLibrary, "Process32NextW");
            pThread32First = (int)GetProcAddress_0(hLibrary, "Thread32First");
            pThread32Next = (int)GetProcAddress_0(hLibrary, "Thread32Next");
            pModule32First = (int)GetProcAddress_0(hLibrary, "Module32First");
            pModule32Next = (int)GetProcAddress_0(hLibrary, "Module32Next");
            pModule32FirstW = (int)GetProcAddress_0(hLibrary, "Module32FirstW");
            pModule32NextW = (int)GetProcAddress_0(hLibrary, "Module32NextW");
        }
    }
    return hLibrary && pCreateToolhelp32Snapshot;
}

```

Figure 11. Dynamically loading APIs

While these two facts were attempts to hide the binaries related to MalumPOS, they can also be used as characteristics to identify and single out files related to this family.

## Improved Variants

In analyzing the main PoS scraper, we also encountered two similar files from the same the threat actor, one that looked like a test binary (sha1: *fe713f9bb90b999250c3b6a3bba965d603de32a3*), and another with an attempt to act as a client stub in a client-server implementation (sha1: *d0b3562d868694fd1147e15483f88f3a78ebedfb*). We have included the first file within our detection of TSPY\_MALUMPOS.SM, but let's take a few moments to look into what seems to be a client-server version of which we were able to analyze the client stub.

The client-server version functions very similarly to the main PoS scraper, but it is clear that the threat actor wanted to have means of remote control.

```

004540B2 8045 AC LEA EAX,DWORD PTR SS:[EBP-54]
004540B5 B9 70484500 MOV ECX,cc.00454870 ASCII "log.ini"
004540BA 8B15 687F4500 MOV EDX,DWORD PTR DS:[457F68]
004540C0 E8 E306FBFF CALL cc.004047A8
004540C5 8B40 AC MOV ECX,DWORD PTR SS:[EBP-54]
004540C8 B2 01 MOV DL,1
004540CA A1 B8F64100 MOV EAX,DWORD PTR DS:[41F6B8]
004540CF E8 94B6FCFF CALL cc.0041F768
004540D4 8945 E0 MOV DWORD PTR SS:[EBP-20],EAX
004540D7 68 80484500 PUSH cc.00454880 ASCII "Error"
004540DC 8045 DC LEA EAX,DWORD PTR SS:[EBP-24]
004540DF 50 PUSH EAX
004540E0 B9 90484500 MOV ECX,cc.00454890 ASCII "Name"
004540E5 BA 00484500 MOV EDX,cc.004548A0 ASCII "PARAMS"
004540EA 8B45 E0 MOV EAX,DWORD PTR SS:[EBP-20]
004540ED 8B18 MOV EBX,DWORD PTR DS:[EAX]
004540EF FF13 CALL DWORD PTR DS:[EBX]
004540F1 68 80484500 PUSH cc.00454880 ASCII "Error"
004540F6 8045 D8 LEA EAX,DWORD PTR SS:[EBP-28]
004540F9 50 PUSH EAX
004540FA B9 B0484500 MOV ECX,cc.004548B0 ASCII "InterfacesIP"
004540FF BA 00484500 MOV EDX,cc.004548A0 ASCII "PARAMS"
00454104 8B45 E0 MOV EAX,DWORD PTR SS:[EBP-20]
00454107 8B18 MOV EBX,DWORD PTR DS:[EAX]
00454109 FF13 CALL DWORD PTR DS:[EBX]
0045410B 6A 00 PUSH 0
0045410D B9 C8484500 MOV ECX,cc.004548C8 ASCII "Port"
00454112 BA 00484500 MOV EDX,cc.004548A0 ASCII "PARAMS"

```

Figure 12. Client-server version of MalumPOS

The settings of the client-server version require a file called "log.ini" for it to function. This file would contain the following settings:

[PARAMS]	Notes :
Name=AAAAA	// to identify the affected endpoint
InterfacesIP=11.11.1.1	// IP address of the server
Port=80	// TCP port number where the server would be listening on

The file "log.ini" would be further populated with the processes it has already evaluated, and then it would send similar information to the configured IP and port. While we are currently not sure if the aforementioned client stub TSPY\_MALUMPOS.A is actively being widely used, it is already functional in its current state. If this would be used, then the whole infection would be complete with the data exfiltration phase as the information would be sent back to the configured IP address.

## Solutions and Recommendations

Given the characteristics of MalumPOS, threat actors can potentially configure future versions of binaries according to their target's environment with ease. While Trend Micro now detects all binaries pertinent to this threat, we are also providing a YARA rule that you can use to look for indicators related to MalumPOS, should you have endpoint monitoring software like Trend Micro™ Deep Discovery Endpoint Sensor.<sup>4</sup>

For more details about PoS malware and how to enhance your security posture, please read "Defending Against PoS RAM Scrapers: Current Strategies and Next-Gen Technologies."<sup>5</sup>

## Indicators

The following indicators are used by the threat actor for the main PoS phase:

Filename	Hash	Detection	Targets
mnv.exe	757ae5eed0c5e229ad9bae586f1281b5de053767	TSPY_MALUMPOS.SM	Oracle Forms process
nvsvc.exe	2cf2f41d2454b59641a84f8180fd7e32135a0dbc	TSPY_MALUMPOS.SM	MICROS 9700 VISAD Driver MICROS 9700 SSL GW
nvsvc.exe	f720bf7d6dbfc4c7bea21d6a3fd0b88f4fe52a4a	TSPY_MALUMPOS.SM	Oracle Forms process
nvsvc.exe	798bc2d91293c18af7e99ba7c9a4fd3010051741	TSPY_MALUMPOS.SM	Web-based PoS systems accessed through Microsoft™ Windows® Internet Explorer
nvsvc.exe	90e85b471b64667dbcde3aee3fa504c0d4b0ad35	TSPY_MALUMPOS.SM	Shift4 Corporation Universal Transaction Gateway PAR Springer-Miller Systems

<sup>4</sup> Trend Micro Incorporated. (2014). *Trend Micro Security and Risk Management*. "Trend Micro Deep Discovery Endpoint Sensor." Last accessed on June 05, 2015, <http://www.trendmicro.com/us/enterprise/security-risk-management/deep-discovery/index.html#endpoint-protection>.

<sup>5</sup> Trend Micro Incorporated. (March 11, 2015). *Trend Micro Security Intelligence*. "Defending Against PoS RAM Scrapers: Current Strategies and Next-Gen Technologies." Last accessed on June 05, 2015, <http://www.trendmicro.com/vinfo/us/security/news/cybercrime-and-digital-threats/defending-against-pos-ram-scrappers-strategies-and-technologies>.

On the other hand, these indicators are part of a seemingly “test” phase for the threat actor:

Filename	Hash	Detection	Notes
rdp.exe	fe713f9bb90b999250c3b6a3bba965d603de32a3	TSPY_MALUMPOS.SM	Looks like a test
winini.exe	d0b3562d868694fd1147e15483f88f3a78ebedfb	TSPY_MALUMPOS.A	Client stub

The YARA rule:

```
rule PoS_Malware_MalumPOS : MalumPOS
{
  meta:
    author = "Trend Micro, Inc."
    date = "2015-05-25"
    description = "Used to detect MalumPOS memory dumper"
    sample_filetype = "exe"
  strings:
    $string1 = "SOFTWARE\\Borland\\Delphi\\RTL"
    $string2 = "B)[0-9]{13,19}\\"
    $string3 = "[A-Za-z\\s]{0,30}\\\[A-Za-z\\s]{0,30}\\"
    $string4 = "TRegExpr(exec): ExecNext Without Exec[Pos]"
    $string5 = /Y:\\PROGRAMS\\.{20,300}\\.pas/ nocase
  condition:
    all of ($string*)
}
```

Trend Micro Incorporated, a global leader in security software, strives to make the world safe for exchanging digital information. Our innovative solutions for consumers, businesses and governments provide layered content security to protect information on mobile devices, endpoints, gateways, servers and the cloud. All of our solutions are powered by cloud-based global threat intelligence, the Trend Micro™ Smart Protection Network™, and are supported by over 1,200 threat experts around the globe. For more information, visit [www.trendmicro.com](http://www.trendmicro.com).

©2015 by Trend Micro, Incorporated. All rights reserved. Trend Micro and the Trend Micro t-ball logo are trademarks or registered trademarks of Trend Micro, Incorporated. All other product or company names may be trademarks or registered trademarks of their owners.



Securing Your Journey  
to the Cloud

10101 N. De Anza Blvd.  
Cupertino, CA 95014

U.S. toll free: 1 +800.228.5651  
Phone: 1 +408.257.1500  
Fax: 1 +408.257.2003