MalumPOS

History and Characteristics

TrendLabs Security Intelligence Blog

Jay Yaneza
Trend Micro Threats Analyst

June 2015

Contents

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.[1] 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*).

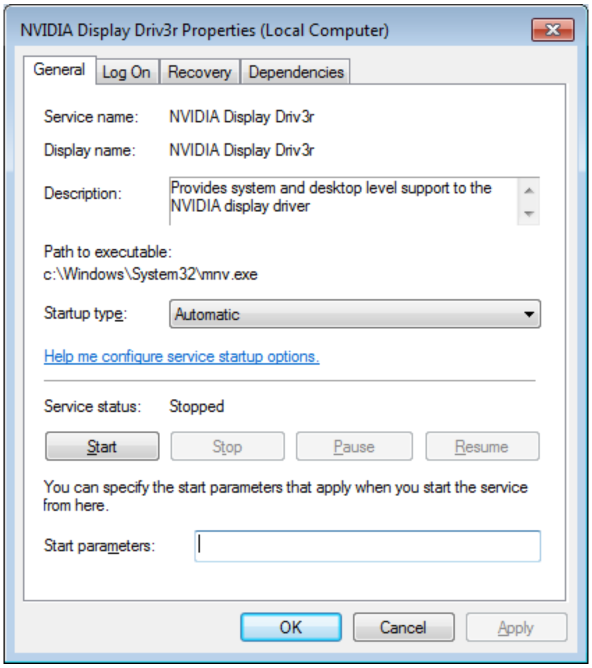

Figure 1. Installed service of MalumPOS

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.

[1] 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\nvsvc.dll*. Like the scraper itself, this is named to appear to be part of NVIDIA drivers. The contents of the file are encrypted.

Figure 2. Main thread of MalumPOS

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

Figure 3. Configuration loading of MalumPOS

The scraper then goes through up to 100 running processes to look for data that can be scraped.

```
if ( fProcess32First((int)hObject, (int)&pPROCESSENTRY32) )
{
  do
  {
    dwCount = 100;                          // Maximum Number is 100
    asSearchProcess = (int *)asMatchArray;
    do
    {
      LStrFromArray((int)&asExecFile, (int)&szExeFile, 260);
      LowerCase(asExecFile, (int)&asProcessName);
      LowerCase(*asSearchProcess, (int)&asMatchProcess);
      StrCmp(asProcessName, asMatchProcess);  // Check ProcessName
      if ( bMatched )
      {
        if ( *asSearchProcess )
        {
          hProcess = OpenProcess(0x1F0FFFu, 0, dwProcessId);
          if ( hProcess )
          {
```

Figure 4. Looks for 100 processes at a time

A simple substitution cipher is used to encrypt the contents of the file containing the stolen information.

```
char *__usercall EncryptFunction@<eax>(char *data@<eax>)
{
  signed int EOF; // edx@2

  if ( *(_DWORD *)data )
  {
    data = checkResult(*(char **)data);
    EOF = (signed int)&data[*((_DWORD *)data - 1)];
    while ( (signed int)data <= EOF )
    {
      *data = (*lookupTable)[(unsigned __int8)*data];
      ++data;
    }
  }
  return data;
}
```

Figure 5. The encryption function

```
C:\Workspace>type c:\windows\system32\nvsvc.dll
¿ΣΣΣΣΣΣΣΣΣΣΣΣΣΣ{Ç*dÇÑÇ*dÇ{Σ!┴Σ     Σ
```

Figure 6. Example of the scraped data

The configured storage file would then be later collected and deciphered with the use of a look-up table.

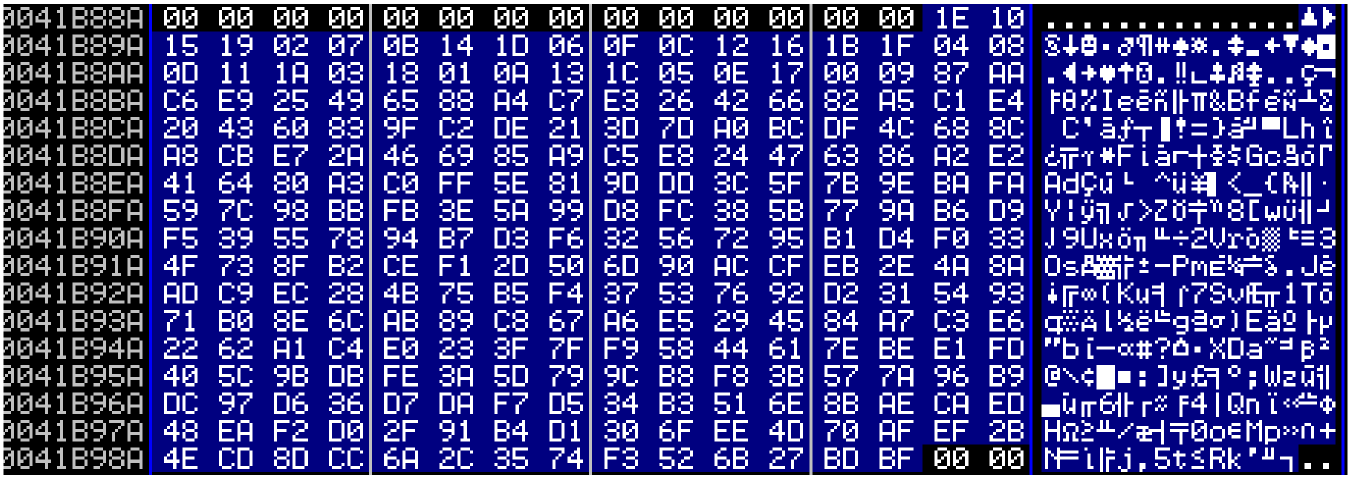

Figure 7. Look-up table used to decrypt the scraped data

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.

a. The CreateToolHelp32Snapshot would be dynamically imported and the API address would be saved for reuse.
b. It searches all matched processes and save into a list.
c. 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.
d. 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.[2] 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.

[2] 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.

The regular expression to detect Track 1 information ((b|B)[0-9]{13,19}\^[A-Za-z\s]{0,30}\/[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.

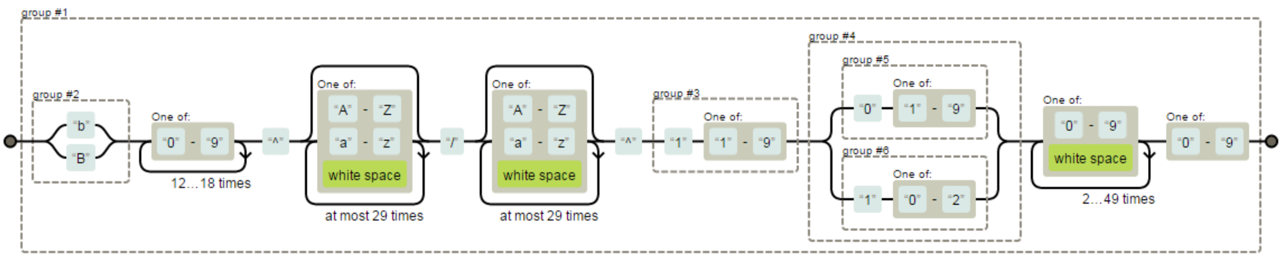

Figure 8. Regular expression used for Track 1 data, visualized with *http://regexper.com/*

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.

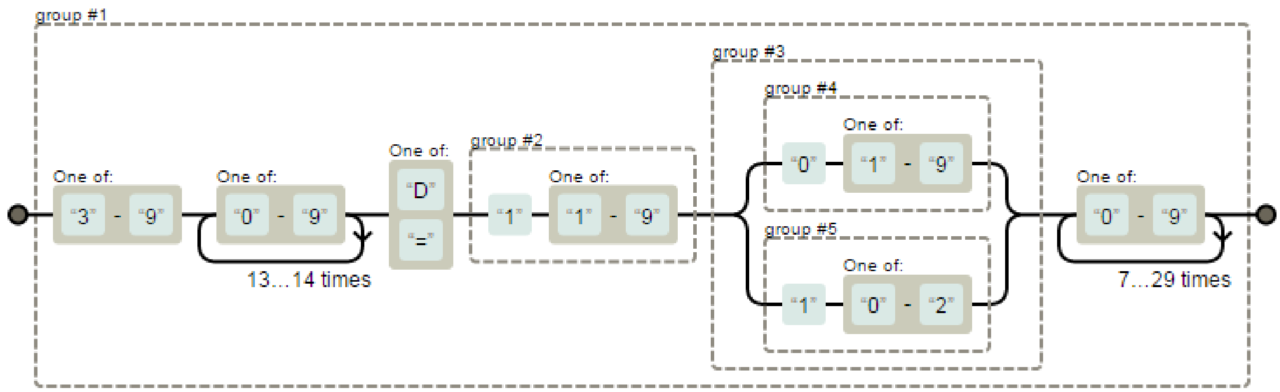

Figure 9. Regular expression used for Track 2 data, visualized with *http://regexper.com/*

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 *Rdaserv* malware family. This is an older malware family that was documented in our earlier white paper on PoS RAM scraper malware.[3]

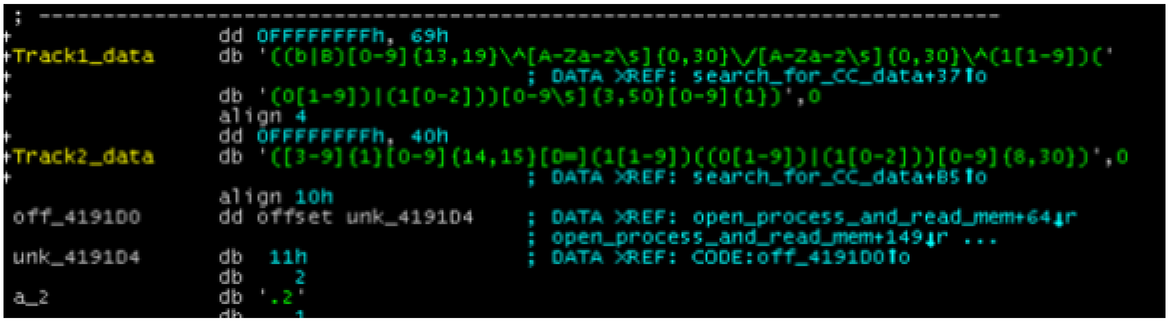

Figure 10. The regex of Rdasrv is the same with MalumPOS

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.

[3] 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.

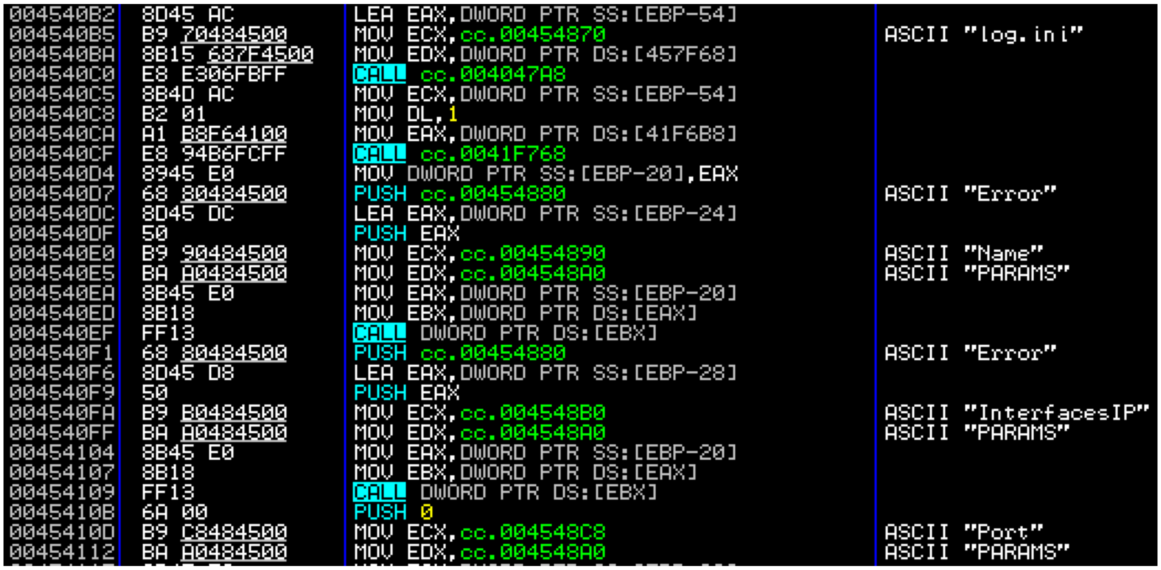

Figure 12. Client-server version of MalumPOS

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=11.11.1.1.1	// 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.[4]

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."[5]

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

[4] 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.

[5] 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-scrapers-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.

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