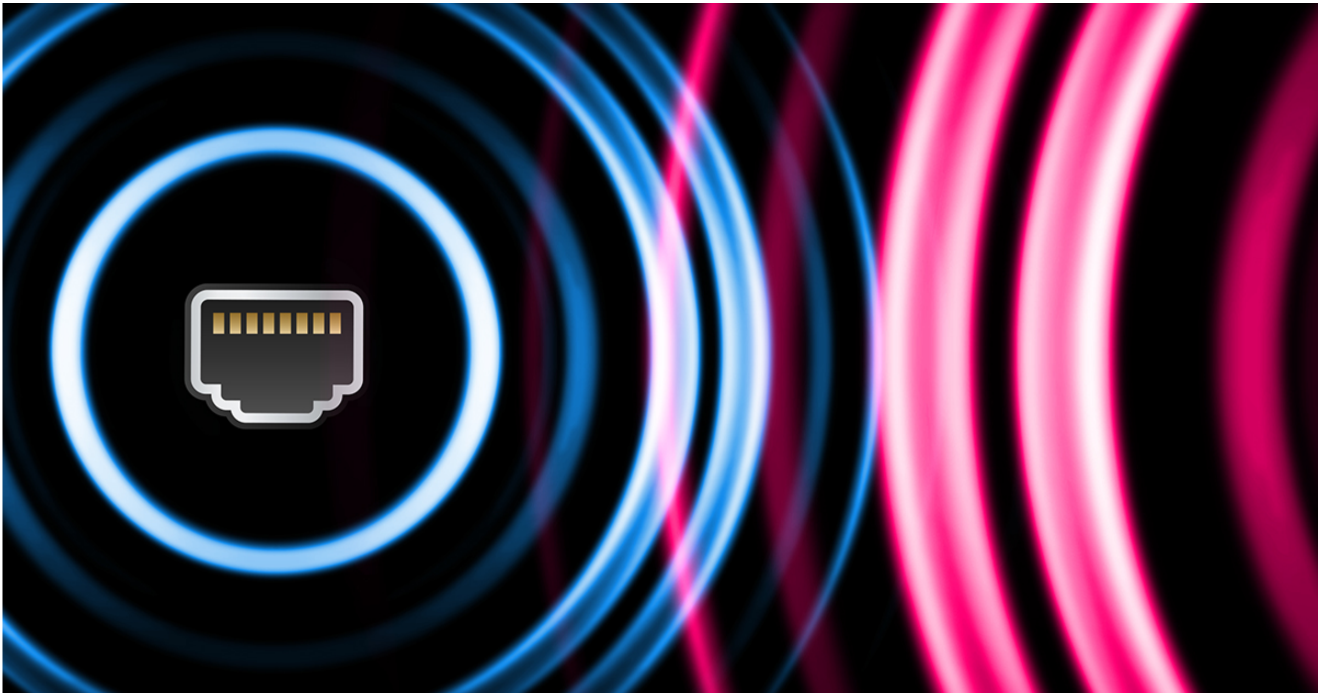
Pingback: Backdoor At The End Of The ICMP Tunnel

trustwave.com/en-us/resources/blogs/spiderlabs-blog/backdoor-at-the-end-of-the-icmp-tunnel/



Loading...

Blogs & Stories

SpiderLabs Blog

Attracting more than a half-million annual readers, this is the security community's go-to destination for technical breakdowns of the latest threats, critical vulnerability disclosures and cutting-edge research.

Introduction

In this post, we analyze a piece of malware that we encountered during a recent breach investigation. What caught our attention was how the malware achieved persistence, how it used ICMP tunneling for its backdoor communications, and how it operated with different modes to increase its chances of a successful attack. Malware using ICMP is not new but is relatively uncommon. Because of this, and the presence of certain strings, we decided to name this malware 'Pingback'. Below we demonstrate how Pingback's protocols work and also provide sample code on how we interacted with the malware.

We begin by looking at how Pingback achieves persistence through DLL hijacking.

Persistence through DLL Hijacking

DLL (Dynamic Link Library) hijacking is a technique that involves using a legitimate application to preload a malicious DLL file. Attackers commonly abuse the Windows DLL Search Order and take advantage of this to load a malicious DLL file instead of the legitimate one.

The file we investigated was a DLL file called ***oci.dll***. We knew that the file was suspicious during our initial triaging, but we could not figure how it was loaded into the system because the DLL was not loaded through traditional rundll32.exe.

```
File Size:                 66.00 KiB (67584  bytes)
File Entropy:              5.95211
Created:                   03/23/2021  16:40:49:221
Modified:                  03/18/2021  15:00:54:00
Accessed:                  03/23/2021  16:40:49:221
SSDEEP:                    1536:b8TTUkboM8m64G95k3TdvdNs5Dpqi1d52cI:b8TTLbT36x9AY5r1d53
CRC32:                     B6535337
ImpHash:                   69A080EB62533E53F1ABDE958E9FB49D
MD5:                       264C2EDE235DC7232D673D4748437969
SHA1:                      0190495D0C3BE6C0EDBAB0D4DBD5A7E122EFBB3F
SHA256:                    E50943D9F361830502DCFDB00971CBEE76877AA73665245427D817047523667F
Authentihash(PE256):31BCF95DD6AC543D640030DE7D91A9C66A7D785EA7DCA0F3C3E768D0164F3A85
ProductVersion:            10, 2, 0, 1
FileVersion:               10, 2, 0, 1
FileDescription:           Oracle Call interface
ProductName:               Oracle Call interface
InternalName:              OCI
LegalCopyright:            Copyright (C) 2010
```

***Figure 1**: **oci.dll** file information*

We found out later that it got loaded through a legitimate service called **msdtc** (a.k.a Microsoft Distributed Transaction Coordinator). This service, as the name suggests, coordinates transactions that span multiple machines, such as databases, message queues, and file systems.

It turns out the **msdtc** service indirectly loads *oci.dll* through MSDTCTM.DLL that loads an ODBC library to support Oracle databases called MTXOCI.DLL. This library searches for and tries to load three Oracle ODBC DLLs which include *oci.dll*, *SqlLib80.dll*, and *xa80.dll*.

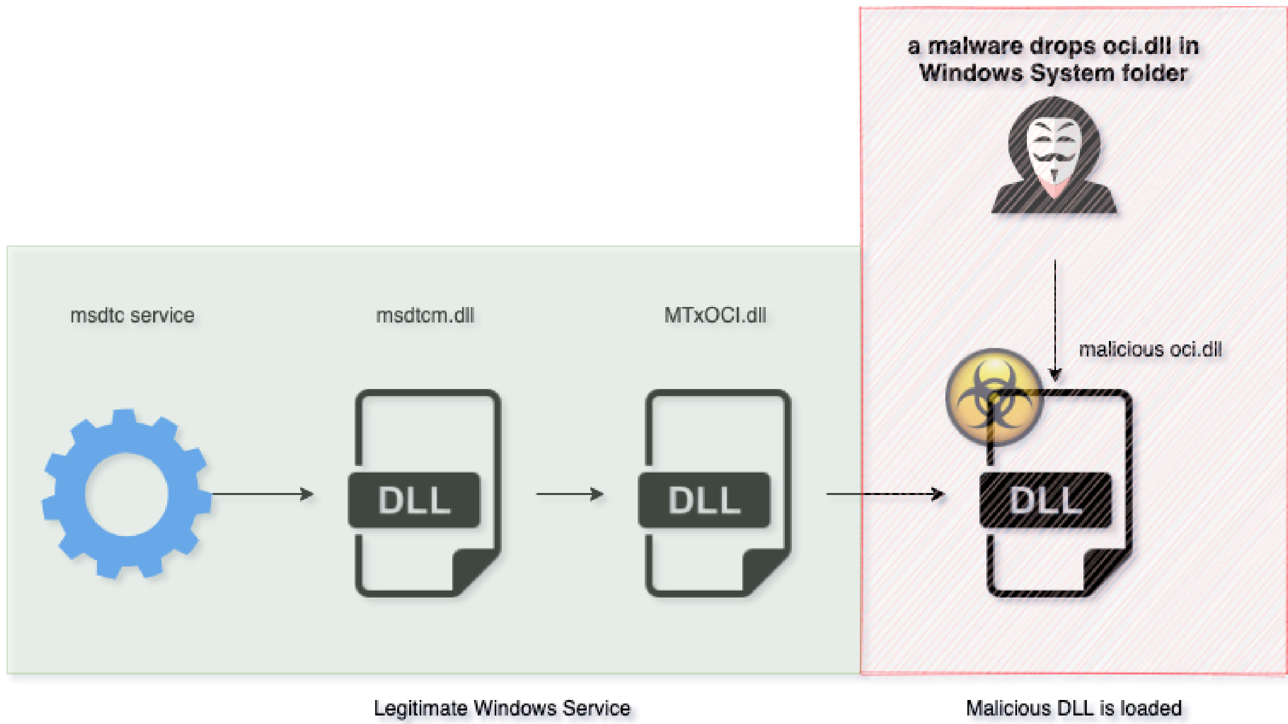


***Figure 2**: Malicious oci.dll is indirectly loaded by msdtc service*

```
if ( RegOpenKeyExA(HKEY_LOCAL_MACHINE, "SOFTWARE\\Microsoft\\MSDTC\\MTxOCI", 0, 0x20019u, &hKey) )
{
  v5 = 0x101i64;
  v6 = Data;
  do
  {
    if ( v5 == 0xFFFFFFFF80000103ui64 )
      break;
    v7 = v6["xa80.dll" - (char *)Data];
    if ( !v7 )
      break;
    *v6++ = v7;
    --v5;
  }
  while ( v5 );
  if ( !v5 )
    --v6;
  *v6 = 0;
  v8 = 0x101i64;
  v9 = v29;
  do
  {
    if ( v8 == 0xFFFFFFFF80000103ui64 )
      break;
    v10 = v9["SQLLib80.dll" - (char *)v29];
    if ( !v10 )
      break;
    *v9++ = v10;
    --v8;
  }
  while ( v8 );
  if ( !v8 )
    --v9;
  *v9 = 0;
  v11 = v28;
  do
  {
    if ( v1 == 0xFFFFFFFF80000103ui64 )
      break;
    v12 = v11["oci.dll" - (char *)v28];
    if ( !v12 )
      break;
    *v11++ = v12;
    --v1;
```

Figure 3: *MTxOCI.DLL loads three plugin DLLs that support the Oracle ODBC interface*

By default, the three Oracle DLLs do not exist in the Windows system directory. So, in theory, an attacker with system privileges can drop a malicious DLL and save it using one of the DLL filenames that *MTxOCI* loads. We have experimented with dropping all three DLL filenames but only ***oci.dll*** was successfully loaded by the service.

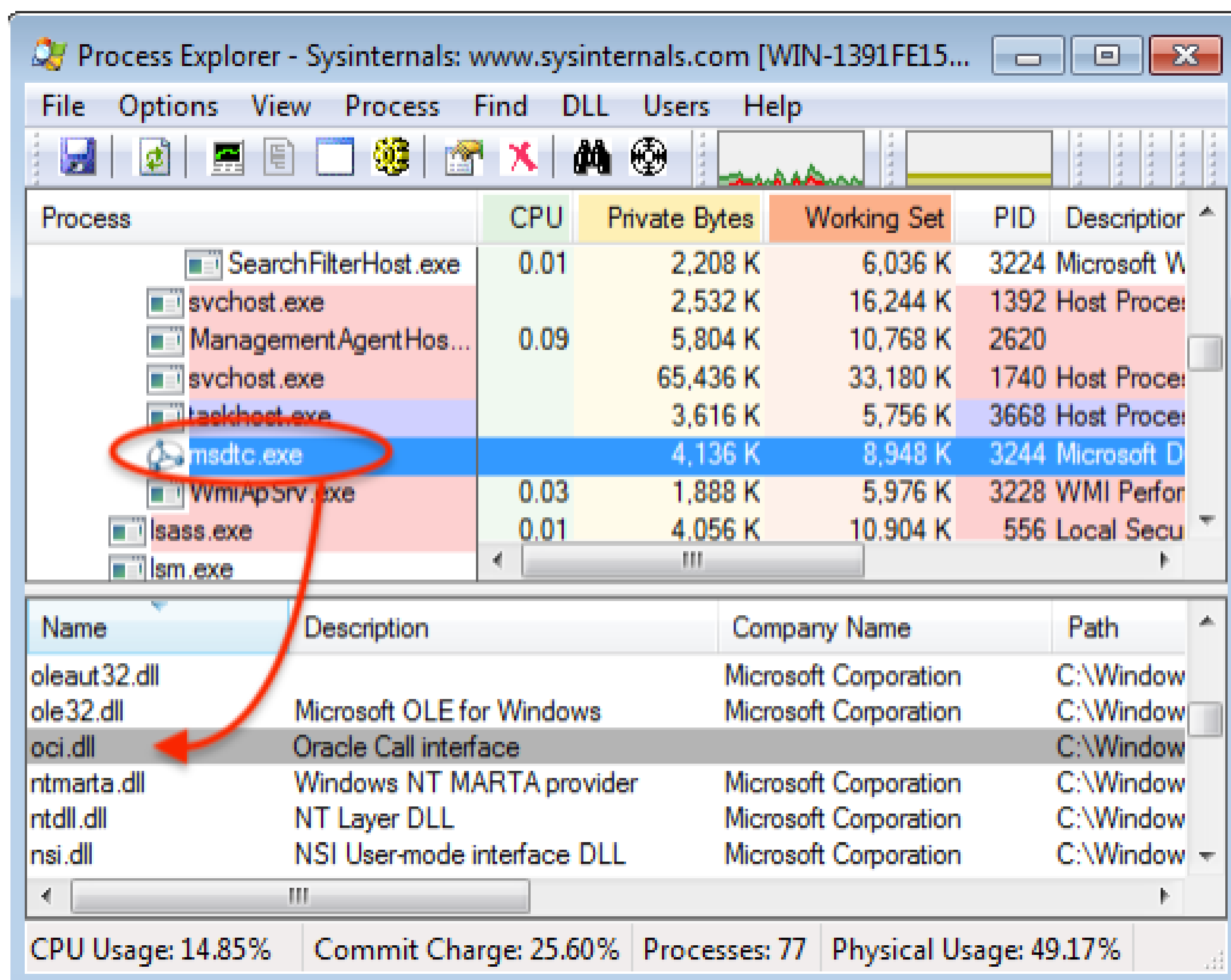


***Figure 4**: oci.dll runs in the background loaded by msdtc.exe*

msdtc by default does not run during start-up. To remain persistent, the **msdtc** service needs to be configured to start automatically, so the attacker would need system privileges to reconfigure the **msdtc** startup type. It can be done manually using SC command, via malicious scripts, or through a malware installer.

Our theory is that a separate executable installed this malware. In fact, after a bit of hunting, we found a sample in VirusTotal with similar IOCs that installs ***oci.dll*** into the Windows System directory and then sets **msdtc** service to start automatically.

```
if ( v5 == -1 || (v5 & 0x10) == 0 )
{
  WinExec("sc config msdtc start= auto", 0);
  Sleep(0x1F4u);
  WinExec(
    "reg add HKLM\\SYSTEM\\CurrentcontrolSet\\Services\\msdtc /v objectname /t REG_SZ /d \"LocalSystem\" /f",
    0);
}
else
{
  WinExec("sc config msdtc obj= Localsystem start= auto", 0);
}
Sleep(500u);
WinExec("sc start msdtc", 0);
```

***Figure 5**: A loader configuring **msdtc** service to start automatically*

We also observed during our analysis that in a VMware environment, the VM Tools service also loads MTXOCI and eventually loads the malicious OCI.DLL.

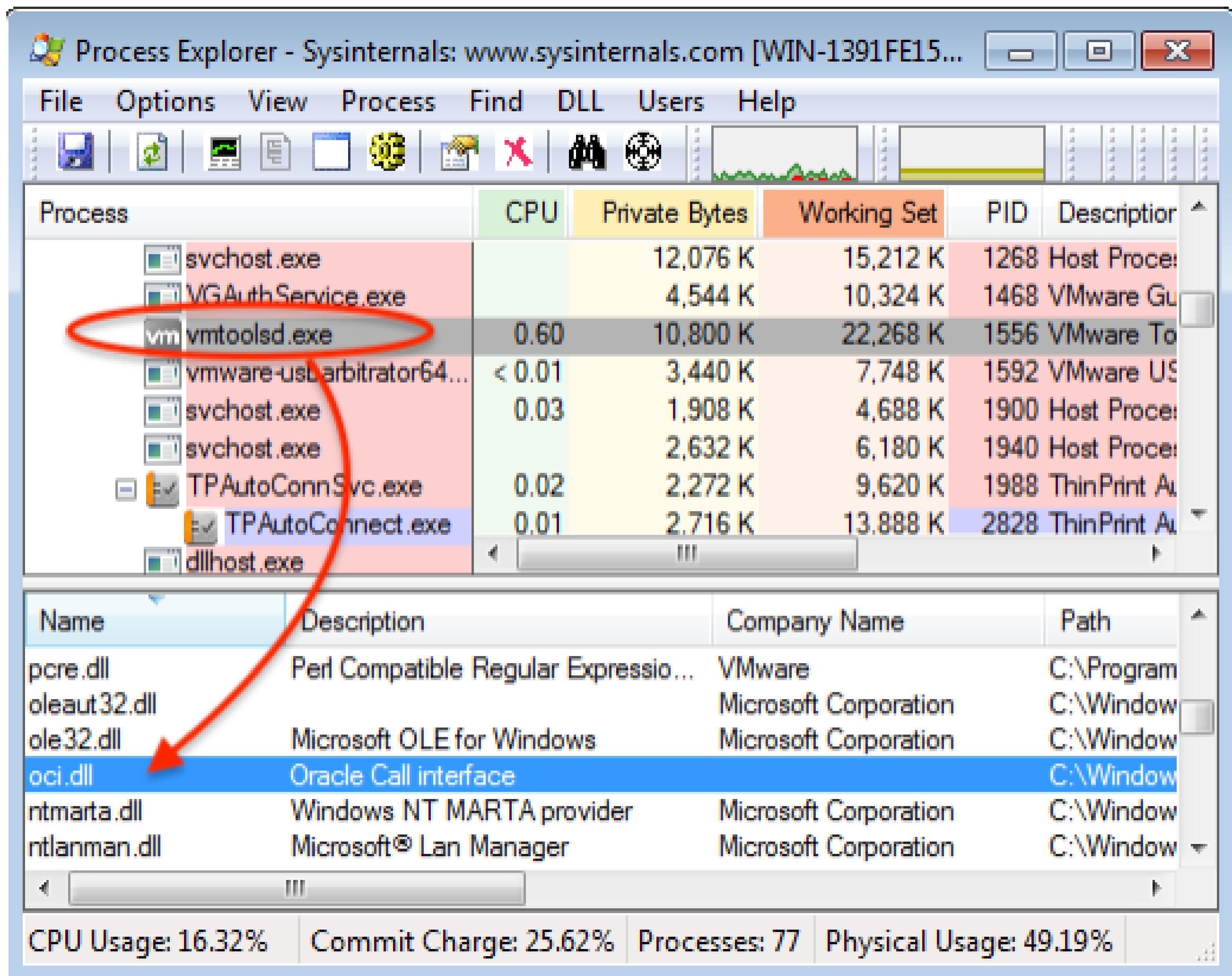


***Figure 6**: In Process Explorer, we found that OCI.DLL is also loaded by VMTools service in a VMware Environment*

So that is the DLL loading part. But before turning our attention to Pingback itself and its operation, let us first lay out what is ICMP and how ICMP tunneling works.

ICMP Foundation

The Internet Control Message Protocol (ICMP) is a network layer protocol mainly used by network devices for diagnostic and control purposes. It is used in utilities such as ping to determine reachability and roundtrip time, traceroute, and path MTU discovery to avoid packet fragmentation and enhance performance. It can also be misused by malicious actors to scan and map a target’s network environment. This is one of the reasons why there are some debates over whether ICMP should be disabled or not. In most cases, users do not pay attention to ICMP packets either as they do not manifest open ports on the machine.

The malware, Pingback, at the center of our investigation, oci.dll, uses the ICMP protocol for its main communication. This has the effect of being hidden from the user as ports cannot be listed by netstat. Below we detail how Pingback uses the ICMP protocol to pass data back and forth between the infected host and the attacker's host. A technique called ICMP tunneling.

To explain ICMP tunneling, let us first understand an ICMP packet:

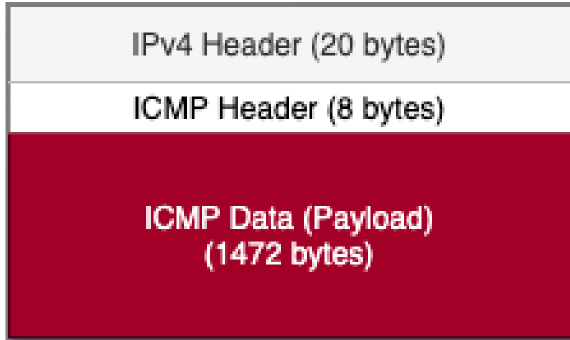


Figure 7*: A diagram of an ICMP packet. ICMP data size varies, we assume that the IP maximum transmission unit is 1500 bytes. The packet size limit for an ICMP Data is maximum allowed size of an IPv4 network packet, minus the 20 byte IP header and 8 byte ICMP header.*

An ICMP packet is built on top of the IP layer and has an 8 byte ICMP header. The packet size limit for ICMP data is a maximum allowed size of an IPv4 network packet, minus the 20 byte IP header and 8 byte ICMP header. Or approximately 64K. The ICMP data is determined by the message type. The message types are defined here: https://www.iana.org/assignments/icmp-parameters/icmp-parameters.xhtml

An ICMP tunnel mainly uses these two types:

Code	Type	Description
0	Echo Reply	ping reply
8	Echo	ping

In the diagram below, A echo packet header defines the ICMP type, code, checksum, identifier and sequence number. And lastly, the ICMP data section is where an attacker can piggyback an arbitrary data to be sent to a remote host. The remote host replies in the same manner, by piggyback an answer into another ICMP packet and sending it back.

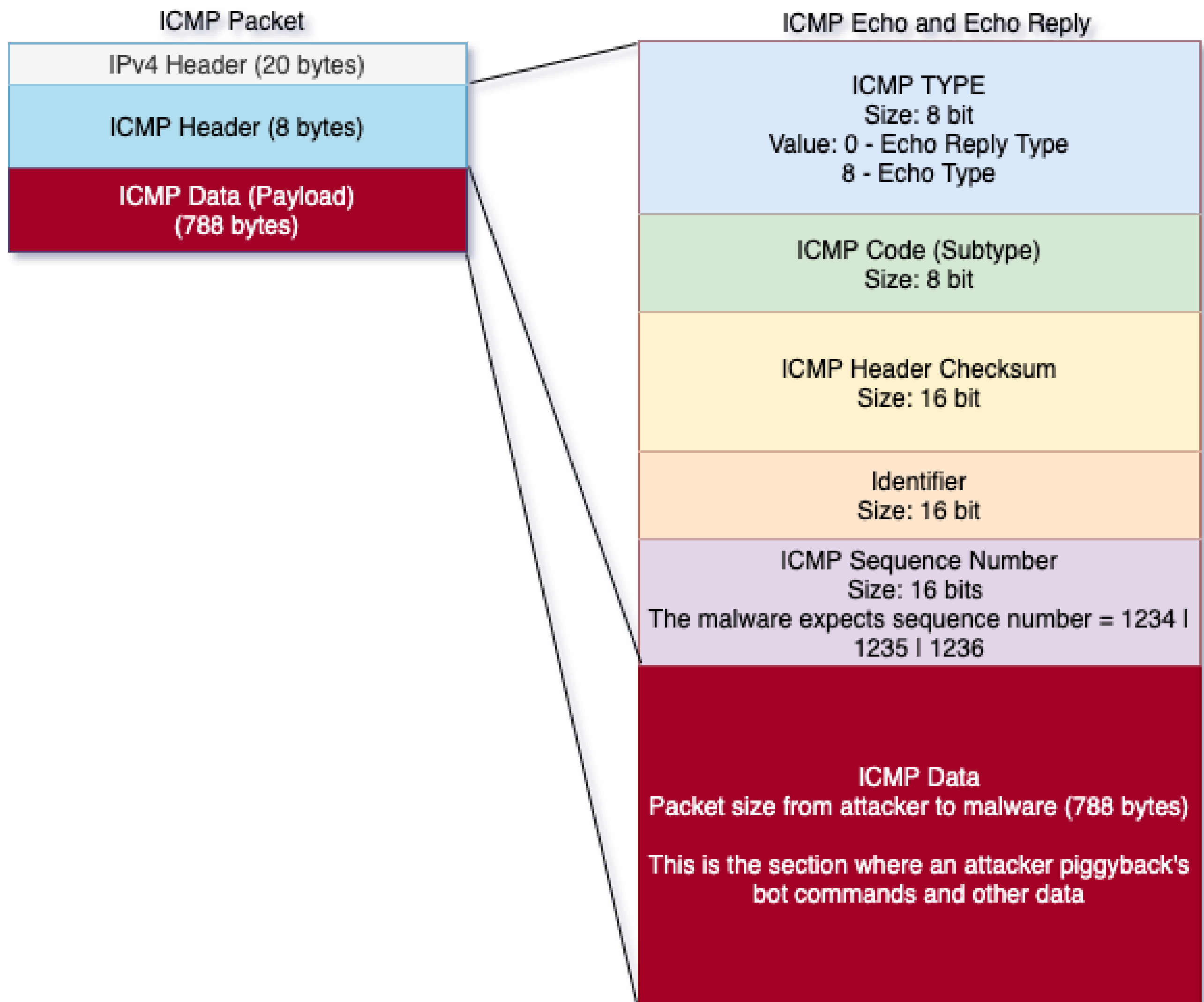


Figure 8: *ICMP packet. The size of the ICMP data sent by the attacker is always 788 bytes*

```
No.  Time            Source          Destination     Protocol Length Info
  15 47.831394572    192.168.38.131  192.168.38.133  ICMP     830    Ec…
 116 139.921172608   192.168.38.131  192.168.38.133  ICMP     830    Ec…

▸ Frame 15: 830 bytes on wire (6640 bits), 830 bytes captured (6640 bits) on inte
▾ Ethernet II, Src: VMware_ec:e3:52 (00:0c:29:ec:e3:52), Dst: Broadcast (ff:ff:ff
  ▸ Destination: Broadcast (ff:ff:ff:ff:ff:ff)
  ▸ Source: VMware_ec:e3:52 (00:0c:29:ec:e3:52)
    Type: IPv4 (0x0800)
▸ Internet Protocol Version 4, Src: 192.168.38.131, Dst: 192.168.38.133
▾ Internet Control Message Protocol
    Type: 8 (Echo (ping) request)
    Code: 0
    Checksum: 0xfc70 [correct]
    [Checksum Status: Good]
    Identifier (BE): 1 (0x0001)
    Identifier (LE): 256 (0x0100)
    Sequence number (BE): 53764 (0xd204)
    Sequence number (LE): 1234 (0x04d2)
  ▸ [No response seen]
  ▾ Data (788 bytes)
      Data: 7368656c6c000000000000000000000000000000000000000…
      [Length: 788]

ICMP Header
ICMP Data

0020  26 85 08 00 fc 70 00 01  d2 04 73 68 65 6c 6c 00   &····p·· ··shell·
0030  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00   ········ ········
0040  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00   ········ ········
0050  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00   ········ ········
0060  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00   ········ ········
0070  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00   ········ ········
0080  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00   ········ ········
0090  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00   ········ ········
00a0  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00   ········ ········
00b0  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00   ········ ········
```

***Figure 9**: Packet capture of the attacker's ICMP packet*

ICMP Tunneling

Now that we have laid out the foundation and how an attacker can piggyback data on the ICMP packets, we are ready to explain exactly how exactly this malware works.

Pingback specifically uses the echo (ping) request or type 8 ICMP message. It starts a sniffer for every IP address on the host, spawning a thread to sniff packets on each individual IP address. To distinguish between its own packets and other packets, the sniffer ignores anything else that's not an ICMP echo packet and does not contain the ICMP sequence number 1234, 1235 or 1236. It also ignores packets not destined for the specified IP address.

It then interprets the data in the following format:

```
struct ICMPData {
    char   cmd[10];         // bot command (see appendix for more details)
    char   args[512];       // extra parameter, but haven't seen it used by the malware
    char   cmd_line[258];       // command line(see appendix for more details)
    unsigned long  dest_port;  // destination port
    char   dest_addr[4];        // destination IP address
}
```

***Figure 10**: Malware's ICMP data is represented by this C structure. See appendix below for detailed information of the **cmd** and **cmd_line** fields.*

The sequence is used as a message type for each ICMP data. It currently supports 3 message types:

- 1234 – the packet contains a command or data
- 1235 and 1236 – used for pure ICMP packet communication only. 1235 being the data has been received at the other end, and 1236 as new data has been received by the malware.

Pingback supports several commands including:

- **shell** – execute a shell
- **download** – 3 different modes of download are provided:
    - Mode 1: Infected host connects back to the attacker's host (works well if incoming TCP connections are blocked by firewall)
    - Mode 2: Infected host opens a socket on a specified port and waits for the attacker to connect.
    - Mode 3: Purely ICMP-based, but this is very slow and the current implementation is not very reliable in terms of flow control.
- **upload** – also supports 3 different modes, similar to the Download command.
- **exec** – execute a command on the infected host.

This is interesting, you can see Pingback uses a combination of ICMP for initiating any of the commands and TCP for better performance and reliability. A pure ICMP mode is also provided but is not very reliable.

Protocol example

To download a file in mode 1, the attacker performs:

- Create a socket and listen on a port
- Send the following ICMP packet:
    - ICMP echo request with sequence 1234
    - Payload contains: "download", name of file to download and, IP address and port of the newly created socket, this is where the malware will stream the file to
- Wait for connections and receive data. There is an additional protocol here which includes is reasonably simple (send and receive) and wait for "END\x00" string.

We have provided a source to demonstrate all three modes and most of the commands supported by Pingback.

Source available here:

https://github.com/SpiderLabs/pingback

We have also prepared a video to demonstrate how our client interacts with the malware running in an isolated infected system.

Watch Video At:

https://youtu.be/OlzgEVk3dig

Final Words

ICMP tunneling is not new, but this particular sample piqued our interest as a real-world example of malware using this technique to evade detection. ICMP is useful for diagnostics and performance of IP connections in the real world. It is very useful to have them enabled but must be balanced by real-world threats. While we are not suggesting that ICMP should be disabled, we do suggest putting in place monitoring to help detect such covert communications over ICMP.

For network administrators and technical audience, a rule can be implemented to check if a packet is an ICMP echo (type 8), the data size is 788 bytes or greater and check for ICMP sequence number: 1234, 1235 or 1236. Backdoor command strings such as “download”, “upload”, “exec”, “exep”, “rexec”, “shell” that found in an ICMP data packet can also be flagged. Trustwave Managed IDS devices can also detect this malicious traffic.

Finally, this malware did not get into the network through ICMP but rather utilizes ICMP for its covert bot communications. The initial entry vector is still being investigated.

Appendix

cmd – bot commands and may be any of the following:

- *exep* (execute process) – execute a binary/command on the remote host
- *download* (download mode 1) - attacker's initial connection is done via ICMP and appears as a ping packet. The ICMP echo packet contains data that specifies the attacker's host and port to where the malware connects back. The ICMP data also contains a file path that the attacker requests. Using the host and port information, the malware creates a new socket, then transmits the requested file back to the attacker.
- *upload* (upload mode 1) – attacker's initial connection is done via ICMP. The malware receives the initial connection then connects back to attacker's host and port specified in the ICMP Echo packet. It then receives the file from the attacker to be saved in the infected system's local disk
- *download2* – (download mode 2), initial ICMP packet is sent by the attacker. The ICMP echo packet contains the requested filename and path in the infected system. It also contains a port number where the malware will bind and listen to. The malware then waits for the attacker to connect, afterward, it begins transmitting the requested file.
- *upload2* – (upload mode 2), initial ICMP packet is sent by the attacker. The ICMP echo packet contains the filename of the file to be received. It also contains a port number where the malware will bind and listen to. The malware then creates the file in the remote host and waits for the attacker to connect. Once connection is established, the attacker begins transmitting the file content to the remote host
- *download3* – (download mode 3), a file is sent to the attacker purely through ICMP data. Although this mode is more covert as it appears as ping packets only, this is slower than using TCP directly as only 1 packet can be transmitted at a time. The malware has to wait for acknowledgment from the attacker's end.
- *upload3* – (upload mode 3), same as download mode 3 – although the attacker uploads the file purely through ICMP. Also slower and unreliable but more covert than other modes.
- *shell* – request malware to connect back to the attacker with a shell. Initial request is done via ICMP packet containing information including attacker's host IP and port to where the malware makes a TCP connection.

cmd_line - In *exep* command, this variable holds the command to be executed on the remote host. While in *download* and *upload* command, this variable contains the remote file name.

IOC:

File:

Filename: **Oci.dll**
SHA256: E50943D9F361830502DCFDB00971CBEE76877AA73665245427D817047523667F
PDB path: c:\Users\XL\Documents\Visual Studio 2008\Projects\PingBackService0509\x64\Release\PingBackService0509.pdb

Network:

Source: <Attacker IP address>
Destination: <Target host>
ICMP Type: 8
Sequence Number: 1234|1235|1236
Data size: 788 bytes

PCAP: https://github.com/SpiderLabs/IOCs-IDPS/tree/master/Pingback

Yara:

```
rule PingBack
{
   meta:
      description = "This rule detects PingBack malware"
      author = "Trustwave SpiderLabs"
      date = "May 4th, 2021"



   strings:
      $string1 = "Sniffer ok!" ascii
      $string2 = "lock2" ascii
      $string3 = "recvfrom failed" ascii
      $string4 = "rexec" ascii
      $string5 = "exep" ascii
      $string6= "download" ascii
      $string7 = "download2" ascii
      $string8 = "download3" ascii
      $string9 = "upload" ascii
      $string10 = "upload2" ascii
      $string11 = "upload3" ascii
      $string12 = "cmd.exe" ascii
      $string13 = "PingBackService" ascii

      condition:
      all of them
```

```
}
```