PRELIMINARY ANALYSIS OF 2005 DFRWS FORENSIC CHALLENGE

by RossettoeCioccolato <rossetoecioccolato@yahoo.com>
August 1, 2005

As per your request on August 1, 2005 I began a forensic examination of two memory "images" acquired from the notebook computer of Professor Goatboy on Sunday, June 5, 2005. The memory "images" were generated by running a tool, dd.exe, by a certain author. The particulars of the case are stated more fully at http://www.dfrws.org/2005/challenge/.

EVIDENCE EXAMINED*:*

1. Physical memory "image" identified as dfrws2005-physical-memory1.dmp (MD5 = 2d767dbc338075f7c7594894716f3290)

2. Physical memory "image" identified as dfrws2005-physical-memory2.dmp (MD5 = dbca88eeb7b8dbd42f406a405e6f56cf)

EQUIPMENT LIST:

1. *Kntlist.exe*, proprietary software developed by *GMG Systems, Inc*. which parses physical memory of certain *Microsoft Windows*™ operating systems in terms of its internal structure.

2. *Wordpad.exe*, a text editor that comes with most modern Microsoft Windows systems.

3. Virtual PC with Service Pack 1 installed.

4. Windbg, a part of the Debugging Tools for Windows version 6.5.3.7 (http://www.microsoft.com/whdc/devtools/debugging/installx86.mspx).

METHOD:

Originally *kntlist* was not designed to analyze any version of Microsoft Windows prior to Windows 2000 with Service Pack 4 installed. Substantial differences exist in the structure of physical memory on Microsoft Windows systems from one version to another and between service packs of the same version. Accordingly we requested copies of the presumptive Windows kernel (*ntoskrnl.exe*) and important networking components (*tcpip.sys* and *afd.sys*). These files then were used to identify the specific Windows version in use on Professor Goatboy's computer. The md5 checksums of the provided files were compared with the md5 checksums of corresponding files taken from known Windows 2000 versions until a match was found. The provided files were found to match files taken from a default installation of Windows 2000 with Service Pack 1 installed.

Accordingly, a virtual machine was set up with Windows 2000 SP1 and a crash dump was generated. The crash dump was loaded into *Windbg* and used to locate the RVA of important symbols relevant to the investigation such as *PsActiveProcessListHead* and *ObpDirectoryObjectRoot*. Based on this analysis specific modifications were made to *kntlist* to support Windows 2000 SP1.

After these modifications *kntlist* was run against dfrws2005-physical-memory1.dmp and dfrws2005-physical-memory2.dmp. The output from this tool as well as log files and sha1 checksums generated by *kntlist* are attached to this report as *dfrws2005-physical-memory1.txt*, *dfrws2005-physical-memory1.log*, *dfrws2005-physical-memory1.sha1*, *dfrws2005-physical-memory2.txt*, *dfrws2005-physical-memory2.log* and *dfrws2005-physical-memory2.sha1*. Additionally, *kntlist* was run against a memory "image" taken from the default Windows 2000 SP1 virtual machine described in the preceding paragraph. This "default image" was used as a reference system in a known good state. During analysis running executables were discovered that are consistent with a Sony OEM installation. Ideally we would like to use a default Sony OEM installation as a reference system; however, that was not available to us within the time frame of this release.

Stated briefly, *kntlist* walks the *PsActiveProcessList* and develops a list of known active processes. Each active process contains a reference to an object or handle table which stores references to resources that are in use by the process. Object tables also are linked in a list that originates at the *HandleTableListHead*. Each object table contains a reference to the process that is charged with the table's resources (ie. the "quota process"). The contents of the active process and handle table lists are correlated and any discrepancies are noted. Next each object in each object table is recorded and searched for references to active processes and threads. If a Windows kernel object contains a reference to an active process that was not found in the active process list, this fact is noted. There is also an object directory that originates at *ObpRootDirectoryObject*. Each object in the object directory was recorded and searched for references to active processes and threads. If an object in the object directory was found to reference an active process that was not found in the active process list, the fact is noted. The loaded and unloaded module lists also are searched and the contents of these lists are enumerated. Most driver objects contain a reference to the module from which they were loaded. The object directory is searched for driver objects. For each driver object the loaded module is noted and compared with the loaded module list. If a module is found that is not contained in the loaded module list that fact is noted. The IDT, GDT and system service tables are enumerated and a loaded module is located for each function pointer in these tables. Relevant fields from each processor control region are noted. The location of each system pagefile is recorded. The system time and time zone is recorded as well as the system boot time. Finally, networking components are enumerated such as the interface list, arp list and the address object, TCB and TWTCB tables. The setting for *SynAttackProtect* is noted.

Pattern searches may also be employed to locate cloaked processes, threads, drivers and other objects of interest that cannot be identified using the means described above. The current release of *kntlist* does not employ pattern searches, however.

On Microsoft Windows™ systems processes serve primarily as the source of resources. Task scheduling occurs on a per thread basis. While a great deal of attention has been focused on locating "cloaked" processes, there is no requirement that malicious code create any process (at least once initial infection has occurred). The true focus of forensic analysis on Windows systems needs to be the thread. Both the kernel and (if it exists) user mode stacks for each thread are given to facilitate analysis of the thread's saved context and stack back trace. Knowing what code a thread currently is executing is extremely value. Currently this analysis must be performed by hand, however, even though the location of the relevant information is given.

In examining the output from *kntlist*, a convention should be noted: The physical address for important symbols is presented in parenthesis after the virtual address. For example, the location of *PsActiveProcessHead* is given as follows:

PsActiveProcessHead: 0x8046B980(46b980)

The hexadecimal value 0x46b980 represents the physical offset of this symbol in the memory "image."

The output from *kntlist* has only been available for a few hours. Accordingly we will only make a few preliminary observations.

PRELIMINARY FINDINGS:

1. Time is an important parameter in digital forensic investigation. The system time was determined using the *shared user data* and was found to be 2005-06-05 14:53:46Z in the first memory "image" and 2005-06-05 15:18:39Z in the second "image." The time zone bias was 144000000000 in both images. The local time was determined by subtracting the time zone bias from the system time and was found to be 2005-06-05 10:53:46 in the first "image" and 2005-06-05 11:18:39 in the second "image." (Note: The output from *kntlist* erroneously indicates the local time as Zulu time. However the value for the time is local time.) The boot time in the first image is 2005-06-05 00:32:27Z and 2005-06-05 15:00:56Z in the second "image." The presentation of the challenge from DFRWS does not provide any basis for evaluating the accuracy of the system clock on the subject system. Accordingly, it must be considered only presumptive time.

2. The *IFList* in both memory dumps contained two interfaces with MAC-addresses of 08-00-46-02-22-f0 and 08-00-46-18-65-ad, respectively.

3. The arp cache in both memory dumps is significant and needs to be further investigated. The first memory dump contains two dynamic arp cache entries. The first is bound to local interface 08-00-46-02-22-f0 and the following remote interface:

```
ArpCacheEntry: 0xFCCA5828 (12c2828)
CreateTime: 0x4
InetAddress: 92.0.94.0
PhysicalAddress: 00-00-00-00-00-00
CacheLife: 0x201800e
Type: dynamic
```

The second arp cache entry is bound to local interface 08-00-46-18-65-ad and the following remote interface:

```
ArpCacheEntry: 0xFCCA62D0 (12c32d0)
CreateTime: 0xfcc9ae30
InetAddress: 18.0.0.0
PhysicalAddress: 20-6e-58-f0-00-00
CacheLife: 0xfcca6378
Type: dynamic
```

The *IFList* of the second memory dump contains the same two interfaces described above. The arp cache contains a single entry that is bound to local interface 08-00-46-18-65-ad:

```
ArpCacheEntry: 0xFCA37968 (1054968)
CreateTime: 0xcc0cf
InetAddress: 192.168.0.5
PhysicalAddress: 00-00-e2-8a-c4-6b
CacheLife: 0x34
Type: dynamic
```

Dynamic arp cache entries are labile and indicate recent network activity.

4. The TWTCB (ie. "time wait" TCB) table in the first memory image was empty. The TWTCB table in the second "image" contained 4 entries all to the network interface with IP address 192.168.0.5:

```
TWTCB: 0xFF1A2CC8 (364ecc8)
Connection: 0x200a8c0:604-->0x500a8c0:8504 192.168.0.2:1030-->192.168.0.5:1157
SomeSequenceNumber1: 0x718b796b
SomeSequenceNumber2: 0x718b796b

TWTCB: 0xFF192B28 (3992b28)
Connection: 0x200a8c0:9cad-->0x500a8c0:8104 192.168.0.2:44444-->192.168.0.5:1153
SomeSequenceNumber1: 0x70b75eaf
SomeSequenceNumber2: 0x70b75eaf

TWTCB: 0xFF1A36E8 (362e6e8)
Connection: 0x200a8c0:404-->0x500a8c0:8304 192.168.0.2:1028-->192.168.0.5:1155
```

```
SomeSequenceNumber1: 0x71756a79
SomeSequenceNumber2: 0x71756a79

TWTCB: 0xFF2007E8 (4dd87e8)
Connection: 0x200a8c0:504-->0x500a8c0:8404 192.168.0.2:1029-->192.168.0.5:1156
SomeSequenceNumber1: 0x71814e57
SomeSequenceNumber2: 0x71814e57
```

Time wait control blocks are used to store information about a TCPIP connection while the local host waits for the remote host to acknowledge that the connection is being closed. They are extremely labile and indicate network activity that occurred either while the second memory "image" was being acquired or shortly before.

One possibility is that these TWTCB's might have been created by Daniels by writing the memory image to a to a network drive. However, in that case we would not expect the TCB'to be generated until after the memory was acquired. To rule out this possibility the command line of dd.exe was examined and the destination was located in the object directory. The destination was found to be a local fixed drive:

```
\??\F:
OBJECT: 0xFCCC6B70(12e3b70)  Type: 3 SymbolicLink
SecurityDescriptor: 0xE1007E38(15a8e38)
        Revision: 1
        Sbz1:     0
        Control:  DaclPresent SelfRelative
        O: S-1-5-32-544
        G: S-1-5-18
        D:(A;;CCRC;;;WD)(A;;CCSDRCWDWO;;;SY)(A;;CCSDRCWDWO;;;BA)(A;;CCRC;;;RC)
Target: \Device\HarddiskVolume3
TargetObject: 0xFCCCAE70
```

Also, the source port for one the TWTCBs (TCP 44444) was identified as belonging to a known trojan, as will be explained below.

Unfortunately, TWTCBs do not contain a reference to the process that originated them.

5. The address object table was examined in each "image." Thirteen listening interfaces were found in the first "image." Fifteen listening interfaces were found in the second "image." In the interest of time we will only discuss interfaces that are not found on a default Windows 2000 installation.

TCP ports 641 and 653 were open and listening on all interfaces:

```
Address Object: 0xFF162E48 (7b35e48)
Local Address: 0x0:8d02 0.0.0.0:653
Protocol: 6 MCastIF: 0x0
Flags1: 0x88  Flags2: 0x4
AssociatedConnections: { -:-} {0xFF162548:FF1635A8}
```

```
Address Object: 0xFF163E48 (836e48)
Local Address: 0x0:8102 0.0.0.0:641
Protocol: 6 MCastIF: 0x0
Flags1: 0x88  Flags2: 0x4
AssociatedConnections: { -:-} {0xFF163608:FF164C28}
```

TCP ports 3000 and 44444 also were open and listening over all interfaces:

```
Address Object: 0xFF16B008 (65d6008)
Local Address: 0x0:b80b 0.0.0.0:3000
Protocol: 6 MCastIF: 0x0
Flags1: 0x88  Flags2: 0x4
AssociatedConnections: { -:-} {0xFF1C4128:FF1C4128}

Address Object: 0xFF17DE48 (3082e48)
Local Address: 0x0:9cad 0.0.0.0:44444
Protocol: 6 MCastIF: 0x0
Flags1: 0x88  Flags2: 0x4
AssociatedConnections: { -:-} {0xFF15A488:FF2518A8}
```

Unfortunately, address objects on Windows 2000 do not contain a reference to the originating process. However, we searched the handle tables and object directory for references to these address objects. References to the first two address objects were found in the handle table of a process entitled *tgcmd.exe*:

```
OBJECT: 0xFF162D08(7b35d08) Type: 26 File
Object Header: 0xFF162CF0
GrantedAccess: 1f01ff PointerCount: 2 HandleCount: 1
SecurityDescriptor: (null)
Path: Tcp
Type: TDI_CONNECTION_FILE
TDI Context: 0xFF163068 (836068)
        ConnectionHandle: 0x4E00004D

        Connection Object: 0xFF1635A8 (8365a8)
        ControlChannel: 0x00000000 (1)
        LocalAddressObject: 0xFF162E48 (7b35e48)          ConnectionId: 0x4e
        AfdEndpoint: 0xFF162D88 (7b35d88)
        ProcessId: 0x3f4 tgcmd.exe
        TableLock: 0xFF277168 (5dc0168)
        ConnectionHandle: 0x4E00004D


OBJECT: 0xFF163AC8(836ac8) Type: 26 File
Object Header: 0xFF163AB0
GrantedAccess: 1f01ff PointerCount: 2 HandleCount: 1
SecurityDescriptor: (null)
Path: Tcp
Type: TDI_CONNECTION_FILE
```

```
TDI Context: 0xFF163A68 (836a68)
        ConnectionHandle: 0x4B00004A

        Connection Object: 0xFF163A08 (836a08)
        ControlChannel: 0x00000000 (1)
        LocalAddressObject: 0xFF163E48 (836e48)             ConnectionId: 0x4b
        AfdEndpoint: 0xFF163B48 (836b48)
        ProcessId: 0x3f4 tgcmd.exe
        TableLock: 0xFF277168 (5dc0168)
        ConnectionHandle: 0x4B00004A

        Address Object: 0xFF163E48 (836e48)
        Local Address: 0x0:8102 0.0.0.0:641
        Protocol: 6 MCastIF: 0x0
        Flags1: 0x88  Flags2: 0x4
        AssociatedConnections: { -:-} {0xFF163608:FF164C28}
```

While the presence of these listening endpoints might at first appear suspicious, an Internet search revealed that a process by this name is a typical part of Sony OEM installations. The process may be carved out of memory and compared with the Sony installation to rule in or out this hypothesis.

The third address object corresponding to TCP port 3000 was located in the handle table belonging to a process entitled *nc.exe*:

```
OBJECT: 0xFF16B2C8(65d62c8) Type: 26 File
Object Header: 0xFF16B2B0
GrantedAccess: 1f01ff PointerCount: 2 HandleCount: 1
SecurityDescriptor: (null)
Path: Tcp
Type: TDI_CONNECTION_FILE
TDI Context: 0xFF16B268 (65d6268)
        ConnectionHandle: 0xB700005F

        Connection Object: 0xFF1C4128 (6758128)
        ControlChannel: 0x00000000 (1)
        LocalAddressObject: 0xFF16B008 (65d6008)            ConnectionId: 0xb7
        AfdEndpoint: 0xFCA256E8 (10426e8)
        ProcessId: 0x448 nc.exe
        TableLock: 0xFF277168 (5dc0168)
        ConnectionHandle: 0xB700005F
```

Examination of the *EPROCESS* block for *nc.exe* revealed that *nc.exe* was started with the following command line:

```
 + 448  nc.exe
        Source: from_active_process_list
        Eprocess Block: 0xFF16E3C0 (0x625d3c0)
        CreateTime: 0x1c5696a664c750 2005-06-05 01:00:54Z
        [...]
        DirectoryTableBase: 0x600d000
        Process Environment Block: 0x7FFDF000
        Command Line: "c:\winnt\system32\nc.exe" -L -p 3000 -t -e cmd.exe
        Section Base Address: 0x00400000

        SectionFileName: \winnt\system32\nc.exe 0xe1e9de48 (0x6971e48)
        Handle Table: 0xFF15D528 (0xa0f528) Count: 96   TableCode: 0xE2095000
```

The section file name is consistent with the process name. The command line indicates that a process, purportedly *netcat*, was used to start a remote command shell. However in that case *cmd.exe* should be the next entry in the active process list but was not found. Another command shell was found which started well before "netcat." One possibility is that the command shell was started and terminated for some reason. Another possibility is that the command shell is still running but "cloaked" by some means that was not discovered by *kntlist*. A third possibility is that this process is not *netcat* at all and the command line doesn't have the same meaning. This anomaly is something that needs to be investigated further.

The fourth endpoint was found in a handle table to reference *UMGR32.exe*:

```
        OBJECT: 0xFF1D2028(6755028) Type: 26 File
        Object Header: 0xFF1D2010
        GrantedAccess: 1f01ff PointerCount: 2 HandleCount: 1
        SecurityDescriptor: (null)
        Path: Tcp
        Type: TDI_CONNECTION_FILE
        TDI Context: 0xFF150948 (c95948)
                ConnectionHandle: 0x87000055

                Connection Object: 0xFF28FD88 (575bd88)
                ControlChannel: 0x00000000 (1)
                LocalAddressObject: 0xFF17DE48 (3082e48)            ConnectionId: 0x87
                AfdEndpoint: 0xFF1861A8 (2daf1a8)
                ProcessId: 0x29c UMGR32.EXE
                TableLock: 0xFF277168 (5dc0168)
                ConnectionHandle: 0x87000055
```

The section file name of *UMGR32.exe* is WINNT\System32\UMGR32.EXE 0xe1bf45a8 (0x93e5a8). This is consistent with the default behavior of BO2K and is highly suspicious.

A handle table also is suspicious. It is the second table in the handle table list. The process id is 0. No process EPROCESS block was found that references this table. A common idiom is for user processes to create files or TCP ports and pass the object to a kernel mode process. This needs to be investigated in this case.

```
2. TABLE: 0xFCE25668(1442668):
        Table: 0xE1003000
        QuotaProcess:
        ProcessId: 0
        HandleCount: 62
        CapturedHandleCount: 62
```

```
TableLevel: 2
StrictFIFO: No
```

The handle table of *UMGR32* also is significant since it contains references to dialup networking components and the fax service. The same references were found in the second memory "image" so this must be default behavior for this application. If Professor GoatBoy's laptop has a dialup connection the phone logs from this site need to be examined to determine if there were any unaccounted outbound telephone calls.

6. Three cloaked processes were discovered during the investigation. An additional copy of the first application, *dfrws2005.exe*, was found running. The second two were named *metasploit.exe*. The thread list for each of these processes was empty and the exit time indicated that they exited shortly after starting. However threads belonging to these processes were found in the object table or handle tables. It was impossible to determine if they were still being scheduled. One possibility is that they are remote threads. It is also possible that *DKOM* was used to remove these processes from the active process list. In that case the process resources would not be cleaned up.

As time is up for submission I will now conclude these observations.

Regards,

RossettoeCioccolato.