Scan of the Month – 32

(September 2004)

www.honeynet.org

Chetan Ganatra
ganatra@speedpost.net

Introduction

The challenge is to analyze a home-made malware binary to identify its inner workings, understand its purpose and capabilities. During the process various challenge questions are to be answered to describe the process adopted and the rationale behind each steps.

Initial Setup

On a hardened test machine (Windows XP Professional) download the binary "RaDa.zip" from the challenge website and compute the MD5 checksum to verify the value against the value provided at the site.

```
C:\Sotm> md5sum RaDa.zip
a75de27ee59ab60e148efe7feee5dd3f *rada.zip
```

Once the integrity of the file was verified it was checked for presence of any known virus patterns using Symantec anti-virus (Corporate Edition with latest virus signatures). Result for the above check implied that the piece of software was not a known virus or worm or a variant. The zip file contains an executable with similar name as "RaDa.exe". This file was also further checked for presence of any known viral traits however no virus/worm was detected.

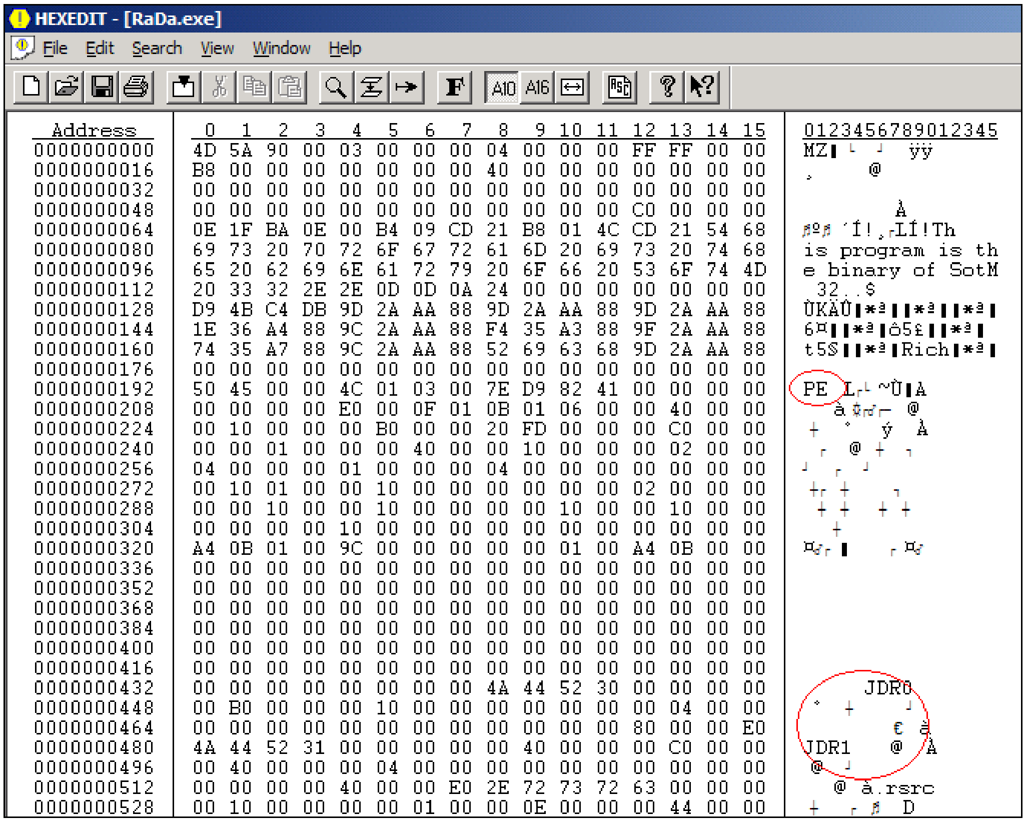
Replies to the challenge questions

Q1. Identify and provide an overview of the binary, including the fundamental pieces of information that would help in identifying the same specimen.

A1. Preliminary analysis of the binary by right-clicking and checking the properties option gives the below details

File size:	20992 byte(s)
FileVersion:	1.00
CompanyName:	Malware
InternalName:	RaDa
OriginalFilename:	RaDa
ProductName:	RaDa
ProductVersion:	1.00
Language:	English (US)

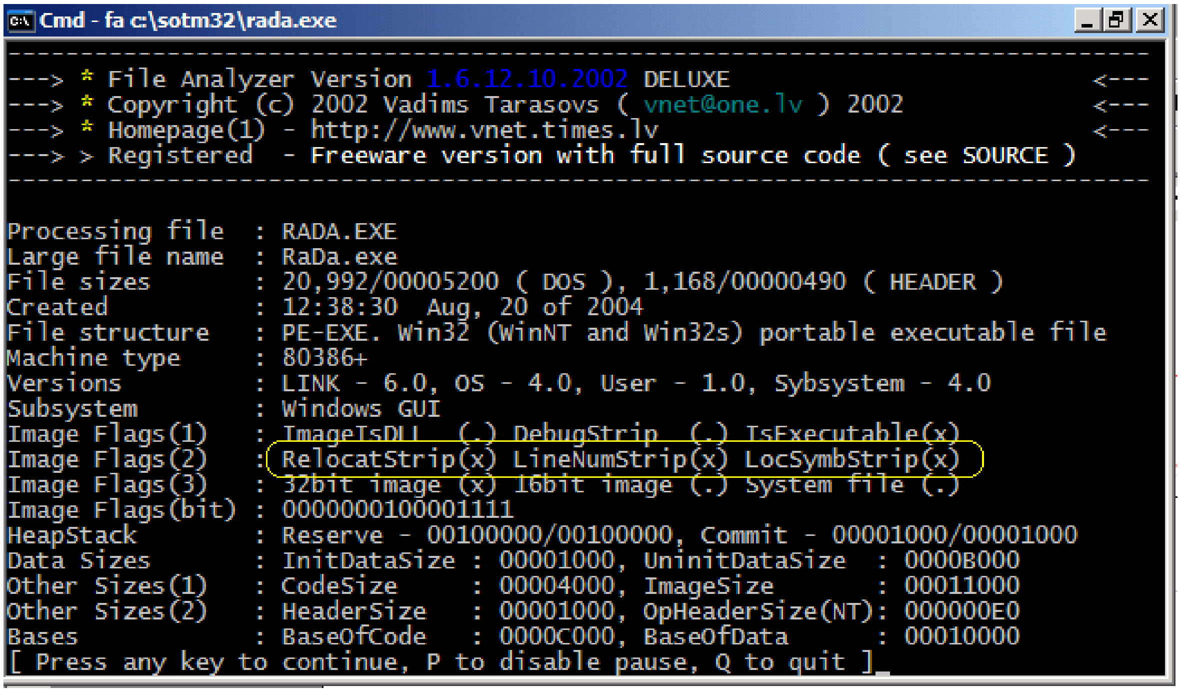
Further viewing the exe via a hex editor reveals the PE header and some unusual section headings like “JDR0” and “JDR1”. This should be a sufficient clue of a binary being modified/manipulated by a specialized software. The snap shot of the above findings is as below.



```
 Address     0  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15   0123456789012345
0000000000  4D 5A 90 00 03 00 00 00 04 00 00 00 FF FF 00 00   MZ▌ └   ┘   ÿÿ
0000000016  B8 00 00 00 00 00 00 00 40 00 00 00 00 00 00 00   ¸       @
0000000032  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0000000048  00 00 00 00 00 00 00 00 00 00 00 00 C0 00 00 00               À
0000000064  0E 1F BA 0E 00 B4 09 CD 21 B8 01 4C CD 21 54 68   ♬º♬ ´Í!¸┌LÍ!Th
0000000080  69 73 20 70 72 6F 67 72 61 6D 20 69 73 20 74 68   is program is th
0000000096  65 20 62 69 6E 61 72 79 20 6F 66 20 53 6F 74 4D   e binary of SotM
0000000112  20 33 32 2E 2E 0D 0D 0A 24 00 00 00 00 00 00 00    32..  $
0000000128  D9 4B C4 DB 9D 2A AA 88 9D 2A AA 88 9D 2A AA 88   ÙKÄÛ▌*ª▌▌*ª▌▌*ª▌
0000000144  1E 36 A4 88 9C 2A AA 88 F4 35 A3 88 9F 2A AA 88   6¤▌▌*ª▌ô5£▌▌*ª▌
0000000160  74 35 A7 88 9C 2A AA 88 52 69 63 68 9D 2A AA 88   t5§▌▌*ª▌Rich▌*ª▌
0000000176  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0000000192  50 45 00 00 4C 01 03 00 7E D9 82 41 00 00 00 00   PE  L┌┘ ~Ù▌A
0000000208  00 00 00 00 E0 00 0F 01 0B 01 06 00 00 40 00 00       à ☼┌♂┌─  @
0000000224  00 10 00 00 00 B0 00 00 20 FD 00 00 00 C0 00 00    ┼   °   ý   À
0000000240  00 00 01 00 00 00 40 00 00 10 00 00 00 02 00 00     ┌   @  ┼   ┐
0000000256  04 00 00 00 01 00 00 00 04 00 00 00 00 00 00 00   ┘   ┌   ┘
0000000272  00 10 01 00 00 10 00 00 00 00 00 00 02 00 00 00    ┼┌  ┼      ┐
0000000288  00 00 10 00 00 10 00 00 00 00 10 00 00 10 00 00     ┼  ┼    ┼  ┼
0000000304  00 00 00 00 10 00 00 00 00 00 00 00 00 00 00 00       ┼
0000000320  A4 0B 01 00 9C 00 00 00 00 00 01 00 A4 0B 00 00   ¤♂┌ ▌     ┌ ¤♂
0000000336  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0000000352  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0000000368  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0000000384  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0000000400  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0000000416  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0000000432  00 00 00 00 00 00 00 00 4A 44 52 30 00 00 00 00           JDR0
0000000448  00 B0 00 00 00 10 00 00 00 00 00 00 00 04 00 00    °   ┼       ┘
0000000464  00 00 00 00 00 00 00 00 00 00 00 00 80 00 00 E0               €  à
0000000480  4A 44 52 31 00 00 00 00 00 40 00 00 00 C0 00 00   JDR1     @   À
0000000496  00 40 00 00 00 04 00 00 00 00 00 00 00 00 00 00    @   ┘
0000000512  00 00 00 00 40 00 00 E0 2E 72 73 72 63 00 00 00       @  à.rsrc
0000000528  00 10 00 00 00 00 01 00 00 0E 00 00 00 44 00 00    ┼     ┌  ♬   D
```

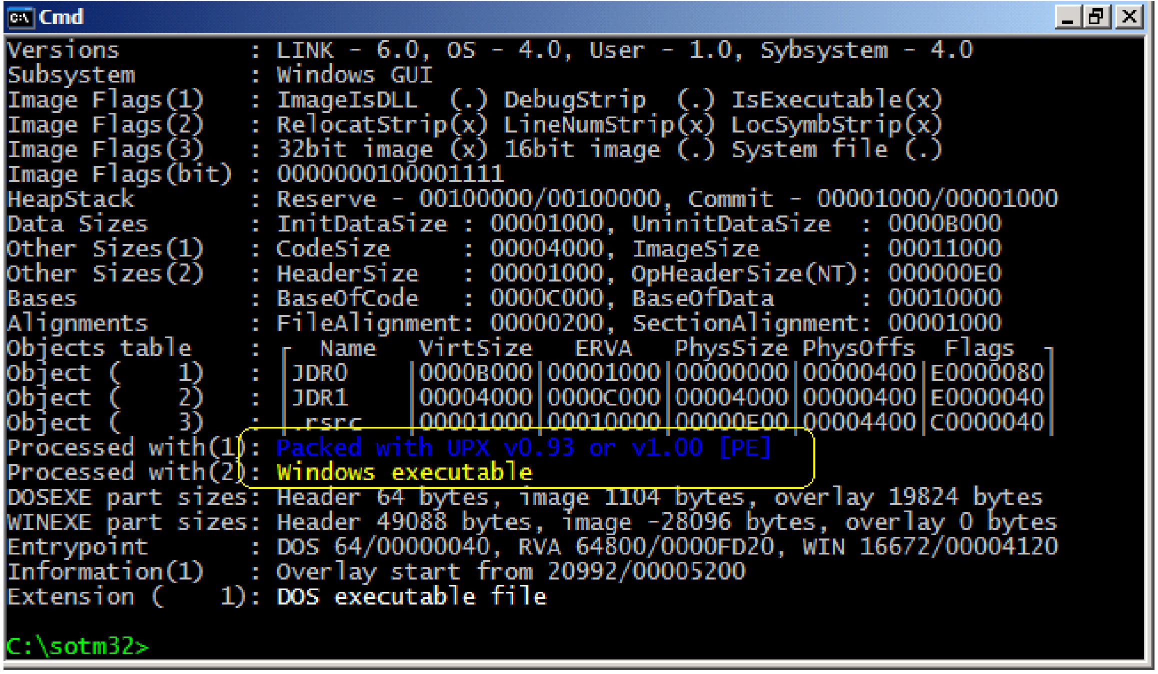
To check if the executable file was compressed or encrypted using any known exe packers or crypters we pass the file through File Analyzer software called (fa) available at http://www.vnet.times.lv

Below is the first page output of the above command. The output reveals that the relocation table, line numbers and location symbols are stripped from the binary, probably to make reverse engineering and debugging difficult.



```
-------------------------------------------------------------------------------
---> * File Analyzer Version 1.6.12.10.2002 DELUXE                        <---
---> * Copyright (c) 2002 Vadims Tarasovs ( vnet@one.lv ) 2002            <---
---> * Homepage(1) - http://www.vnet.times.lv                             <---
---> > Registered  - Freeware version with full source code ( see SOURCE )
-------------------------------------------------------------------------------

Processing file   : RADA.EXE
Large file name   : RaDa.exe
File sizes        : 20,992/00005200 ( DOS ), 1,168/00000490 ( HEADER )
Created           : 12:38:30  Aug, 20 of 2004
File structure    : PE-EXE. Win32 (WinNT and Win32s) portable executable file
Machine type      : 80386+
Versions          : LINK - 6.0, OS - 4.0, User - 1.0, Sybsystem - 4.0
Subsystem         : Windows GUI
Image Flags(1)    : ImageIsDLL   (.) DebugStrip   (.) IsExecutable(x)
Image Flags(2)    : RelocatStrip(x) LineNumStrip(x) LocSymbStrip(x)
Image Flags(3)    : 32bit image  (x) 16bit image  (.) System file (.)
Image Flags(bit)  : 0000000100001111
HeapStack         : Reserve - 00100000/00100000, Commit - 00001000/00001000
Data Sizes        : InitDataSize : 00001000, UninitDataSize   : 0000B000
Other Sizes(1)    : CodeSize     : 00004000, ImageSize        : 00011000
Other Sizes(2)    : HeaderSize   : 00001000, OpHeaderSize(NT): 000000E0
Bases             : BaseOfCode   : 0000C000, BaseOfData       : 00010000
[ Press any key to continue, P to disable pause, Q to quit ]_
```

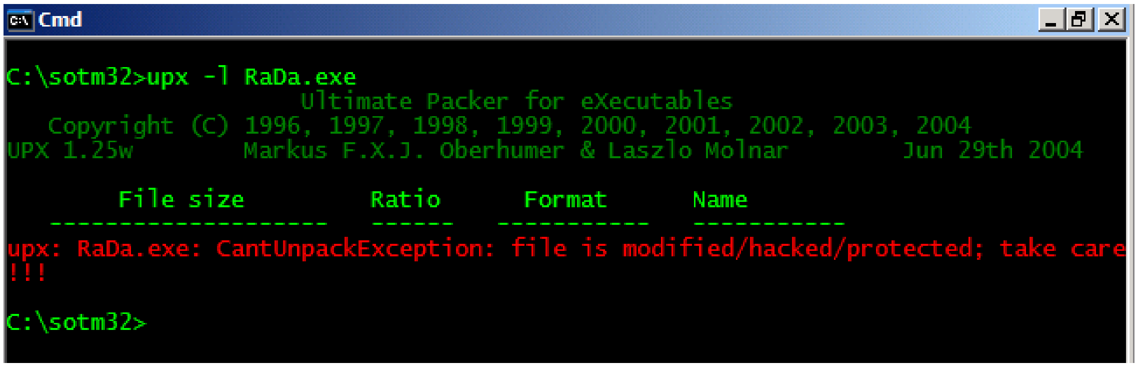
Below is the snap shot for the second page of output



```
Versions          : LINK - 6.0, OS - 4.0, User - 1.0, Sybsystem - 4.0
Subsystem         : Windows GUI
Image Flags(1)    : ImageIsDLL   (.) DebugStrip   (.) IsExecutable(x)
Image Flags(2)    : RelocatStrip(x) LineNumStrip(x) LocSymbStrip(x)
Image Flags(3)    : 32bit image  (x) 16bit image  (.) System file (.)
Image Flags(bit)  : 0000000100001111
HeapStack         : Reserve - 00100000/00100000, Commit - 00001000/00001000
Data Sizes        : InitDataSize : 00001000, UninitDataSize   : 0000B000
Other Sizes(1)    : CodeSize     : 00004000, ImageSize        : 00011000
Other Sizes(2)    : HeaderSize   : 00001000, OpHeaderSize(NT): 000000E0
Bases             : BaseOfCode   : 0000C000, BaseOfData       : 00010000
Alignments        : FileAlignment: 00000200, SectionAlignment: 00001000
Objects table     : ┌  Name    VirtSize   ERVA    PhysSize PhysOffs  Flags  ┐
Object (     1)   : │JDR0     │0000B000│00001000│00000000│00000400│E0000080│
Object (     2)   : │JDR1     │00004000│0000C000│00004000│00000400│E0000040│
Object (     3)   : │.rsrc    │00001000│00010000│00000E00│00004400│C0000040│
Processed with(1) : Packed with UPX v0.93 or v1.00 [PE]
Processed with(2) : Windows executable
DOSEXE part sizes: Header 64 bytes, image 1104 bytes, overlay 19824 bytes
WINEXE part sizes: Header 49088 bytes, image -28096 bytes, overlay 0 bytes
Entrypoint        : DOS 64/00000040, RVA 64800/0000FD20, WIN 16672/00004120
Information(1)    : Overlay start from 20992/00005200
Extension (     1): DOS executable file

C:\sotm32>
```

The last observation gives significant information about the executable. It detects the executable being packed with UPX v0.93 or v1.00 [PE].

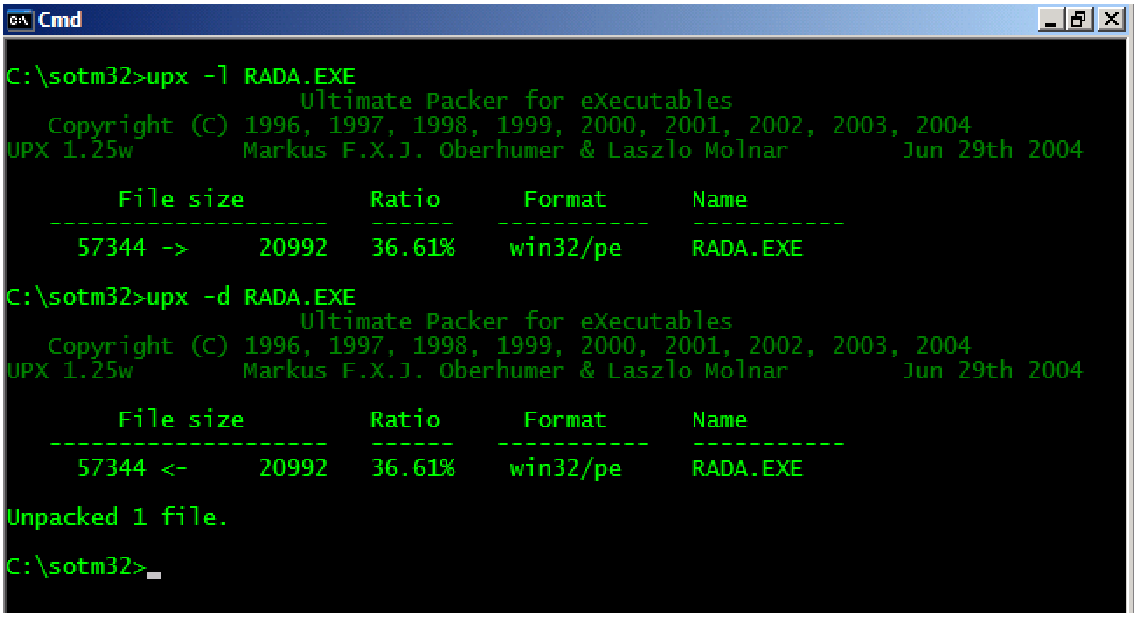
UPX is "Ultimate Packer for executables" available at http://upx.sourceforge.net. UPX compress executable files and reduces its size significantly. We tried to test the binary with upx, however the test failed as shown in the below snap shot. The binary seemed to be modified or protected after being compressed.



```
C:\sotm32>upx -l RaDa.exe
                       Ultimate Packer for eXecutables
   Copyright (C) 1996, 1997, 1998, 1999, 2000, 2001, 2002, 2003, 2004
UPX 1.25w        Markus F.X.J. Oberhumer & Laszlo Molnar        Jun 29th 2004

        File size         Ratio      Format      Name
   --------------------   ------   -----------   -----------
upx: RaDa.exe: CantUnpackException: file is modified/hacked/protected; take care
!!!

C:\sotm32>
```

After having analyzed the binary several times in a hex editor and several trial and errors at debugging the executable, the binary was successfully decompressed after changing the before mentioned section headers from JRD to UPX, which is the default for files compressed with upx.

Below is the snap shot for successful decompression of the RaDa.exe binary

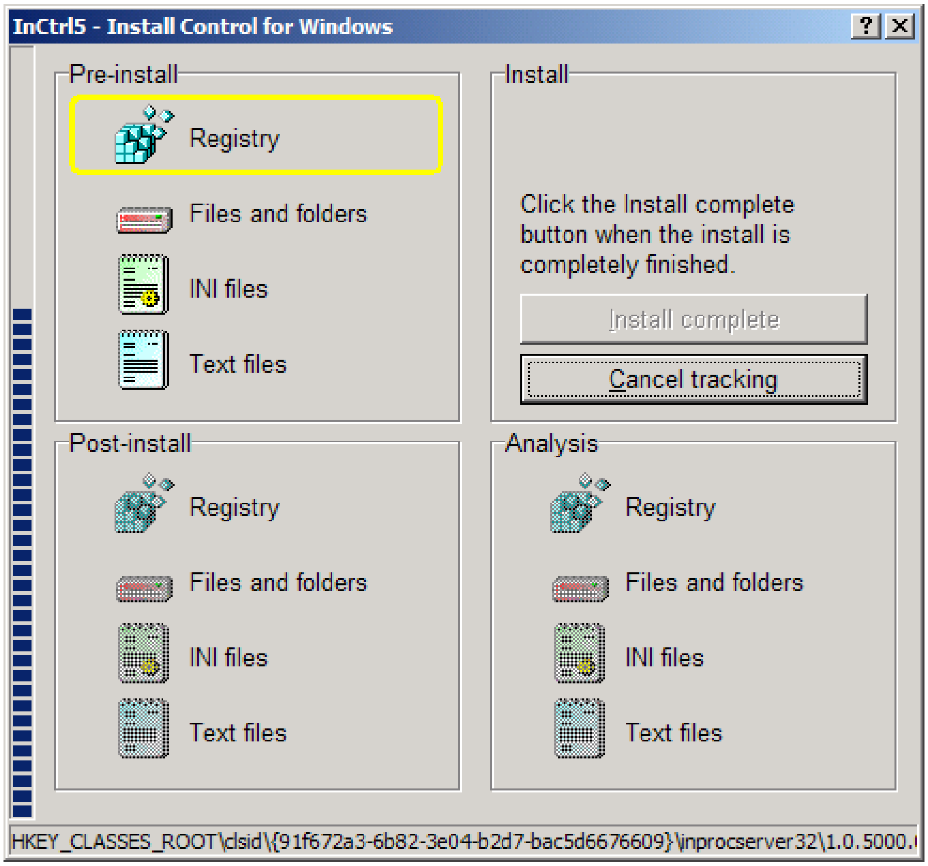


```
C:\sotm32>upx -l RADA.EXE
                       Ultimate Packer for eXecutables
   Copyright (C) 1996, 1997, 1998, 1999, 2000, 2001, 2002, 2003, 2004
UPX 1.25w        Markus F.X.J. Oberhumer & Laszlo Molnar        Jun 29th 2004

        File size         Ratio      Format      Name
   --------------------   ------   -----------   -----------
     57344 ->     20992   36.61%    win32/pe     RADA.EXE

C:\sotm32>upx -d RADA.EXE
                       Ultimate Packer for eXecutables
   Copyright (C) 1996, 1997, 1998, 1999, 2000, 2001, 2002, 2003, 2004
UPX 1.25w        Markus F.X.J. Oberhumer & Laszlo Molnar        Jun 29th 2004

        File size         Ratio      Format      Name
   --------------------   ------   -----------   -----------
     57344 <-     20992   36.61%    win32/pe     RADA.EXE

Unpacked 1 file.

C:\sotm32>
```

Q2. Identify and explain the purpose of the binary.

A2. The binary is a compressed installer, which, if not currently installed on local drive, installs it self on the current root drive, adds an entry in the registry to execute itself on every reboot and in the end copies itself in C:\RaDa\bin directory. With in the C:\RaDa directory two other directories namely “bin” and “tmp” are also created. The RaDa.exe binary is placed in the bin directory.

The above observations are made by using a tool called InCtrl5 that enables us to monitor and compare system drives, registry and ini files during installations.



Click the Install complete button when the install is completely finished.

HKEY_CLASSES_ROOT\clsid\{91f672a3-6b82-3e04-b2d7-bac5d6676609}\inprocserver32\1.0.5000.

Below are few of the significant changes as revealed in the InCtrl5 report file

Registry Added:

```
HKLM\SOFTWARE\Microsoft\Windows\CurrentVersion\Run "RaDa"
      New type: REG_SZ
      New data: C:\RaDa\bin\RaDa.exe
```

Folders added: 3

```
      c:\RaDa
      c:\RaDa\bin
```

```
c:\RaDa\tmp
```

Files added:

```
c:\RaDa\bin\RaDa.exe
     Date: 8/20/2004 12:38 PM
     Size: 20,992 bytes
```

The string "*Starting DDoS Smurf remote attack...*" found in the rada.exe binary suggest that its some kind of flooding tool. Further analysis done on the data captured by network sniffer "ngsniff" revealed below facts

RaDa.exe periodically tries to initiate connection to 10.10.10.10 on port 80.

Destination IP:	10.10.10.10
Destination Port:	80
Source Port:	4400+ (Source port increased after every request)

After extracting Unicode strings from the above binary few of the below partial IP address were also found:

```
http://192.168.
http://172.16.
http://10.
```

Probably these are the IP prefix for randomly generated IP addresses to be used during the attack. During the test period only 10.10.10.10 IP address was found to be used.

Text found in the binary as listed below also suggest usage of Vbscripts to upload data files to a server via HTTP. This forms could be used to transfer the malware to vulnerable machines.

```
Upload file using http And multipart/form-data
Copyright (C) 2001 Antonin Foller, PSTRUH Software
[cscript|wscript] fupload.vbs file url [fieldname]
  file ... Local file To upload
```

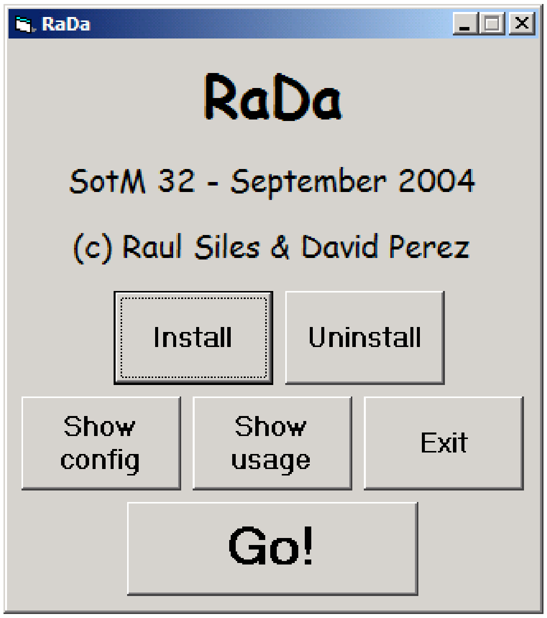
Q3. Identify and explain the different features of the binary. What are its capabilities?

A3. Below are few listed features of the binary as observed during the analysis.

1. If the machine is not currently infected, the binary would install itself on the system and add a registry key to automatically invoke itself at every reboot and copy itself to the specific root location. Exact details of the installation are given in the Answer 2 above.
2. Once executed, manually or automatically at reboot, the binary tries to flood IP address 10.10.10.10 on port 80 after a fixed time interval.
3. Below is a list of commands line arguments as supported by the binary

```
--verbose
--visible
--server
--commands
--cgipath
--cgiget
--cycles
--help
--installdir
--noinstall
--uninstall
--authors
```

As the option name suggest the binary can be provided with the above additional arguments to get more information related to the specific command. The binary also supports a GUI interface, which can be invoked by using the `--gui` parameter. Below is the snap shot of the gui interface



4. The binary also has the capability to upload files via HTTP and multipart/form-data using windows scripting. The actual code can be retrieved from the string reference found in the binary. The script code used in the binary can be found at below url:

   `http://www.motobit.com/tips/detpg_uploadvbsie.htm`

   The script uses ADODB.Recordset and InternetExplorer.Application components. These components are used, as they are readily available on all windows machine and are capable of dealing with binary data.

Q4. Identify and explain the binary communication methods. Develop a Snort signature to detect this type of malware being as generic as possible, so other similar specimens could be detected, but avoiding at the same time a high false positives rate signature.

A4. As inferred from the binary disassembly the malware uses two mode of communication.

1. For sending data collected or to transfer itself from the victim machine the binary uses HTTP multipart/form-data to upload or transfer binary files to other vulnerable hosts. The offset location and the abnormal section headings in the original binary like "JDR0" and "JDR1" can be used as a signature pattern to identify this binary and its variants using similar patching techniques.
2. For generating SYN flood it uses TCP/IP with a incremental source port starting at 4400+ port number and targeting 10.10.10.10 IP address. Further reference have been found, which indicates the binary may use http://192.168. and http://172.16. for the attack.

   Below is a partial list of the sniffer log generated using ngSniff. (Few IP header details are removed clarity).

```
IP HEADER 172.16.15.28 -> 10.10.10.10
------------------------------------------
 IP->version: 4
 IP->ihl: 5
 IP->tos: 0
 IP->tot_len: 48
 IP->id: 31250
 IP->frag_off: 64
 IP->ttl: 128
 IP->protocol: 6
 IP->checksum: 3609

TCP HEADER
----------
 TCP->sport: 4755
 TCP->dport: 80
 TCP->seq: 1550194942
 TCP->ack: 0
 TCP->off: 7
 TCP->flags: SYN
 TCP->window: 64240
 TCP->checksum: 38465
 TCP->urp: 0
```

```
IP HEADER 172.16.15.28 -> 10.10.10.10
------------------------------------------
 IP->version: 4
 IP->ihl: 5
 IP->tos: 0
 IP->tot_len: 48
 IP->id: 31506
 IP->frag_off: 64
 IP->ttl: 128
 IP->protocol: 6
 IP->checksum: 3353

TCP HEADER
----------
 TCP->sport: 4756
 TCP->dport: 80
 TCP->seq: -769802498
 TCP->ack: 0
 TCP->off: 7
 TCP->flags: SYN
 TCP->window: 64240
 TCP->checksum: 31113
 TCP->urp: 0
```

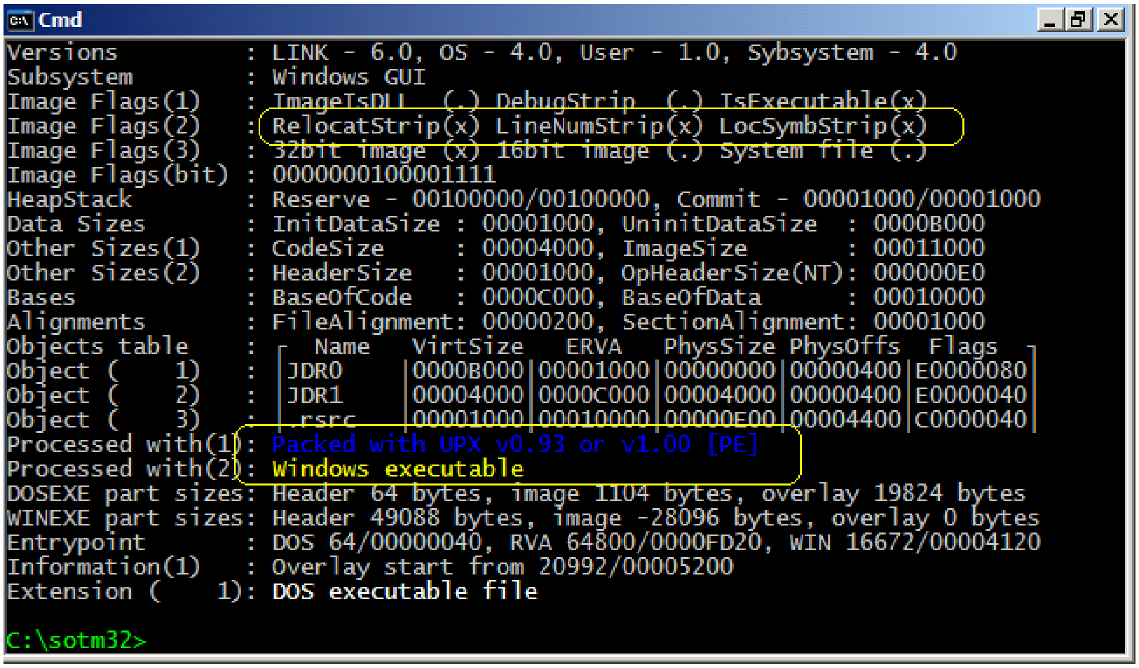
Snort signature for identifying this binary communication is as suggested below.

```
alert tcp any 4400:5000 -> 10.10.10.10 80 (msg: "RaDa.EXE DoS
attempt"; flags: S; ack: 0; classtype: DoS; Priority:1; react:
Block; reference: http://www.honeynet.org/scans/scan32;)

alert ip any 4400:5000 -> 10.10.10.10 80 (msg: "RaDa.EXE DoS
attempt"; ip_proto: tcp; fragoffset: 64; reference:
http://www.honeynet.org/scans/scan32;)
```

Q5. Identify and explain any techniques in the binary that protect it from being analyzed or reverse engineered.

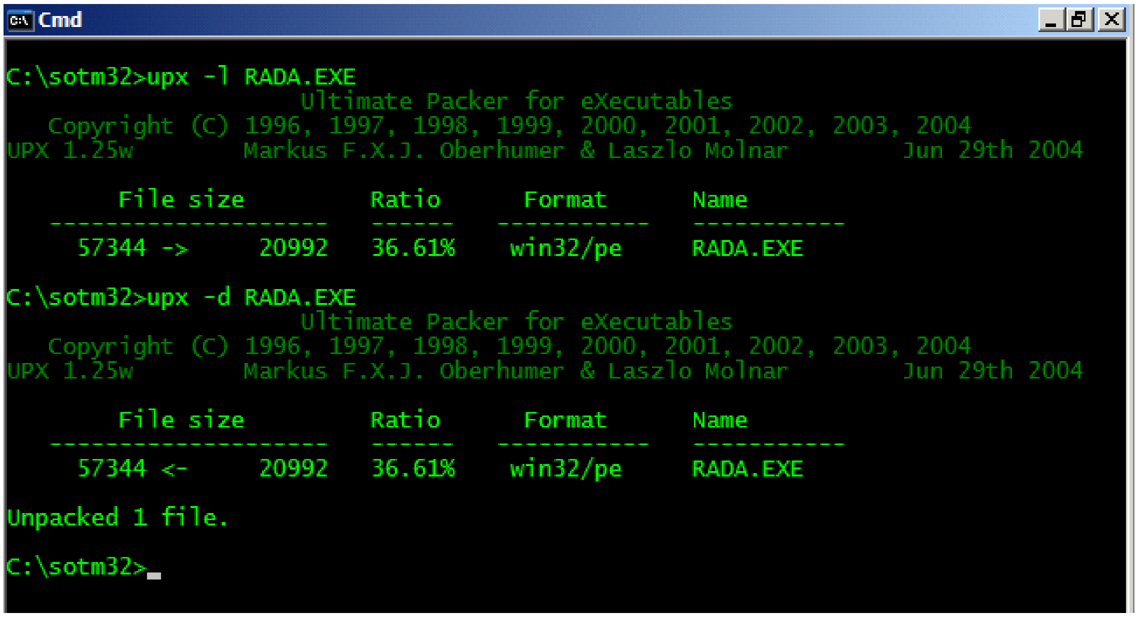
A5. Below are few of the techniques identified during the analysis

1. Details regarding Relocation, Line numbers and Symbol tables were stripped from the binary to remove any debugging information. This makes the task of debugging binaries more difficult
2. The executable "RaDa.exe" was packed using UPX v0.93 or v1.00. Additionally the default UPX section headers were changed to JDR to defeat known packer detection tools. Replacing the section heading from JDR to UPX made it possible to decompress the binary using the latest version of UPX v1.25. Below are few snap shots that reveal the above information.



```
Versions           : LINK - 6.0, OS - 4.0, User - 1.0, Sybsystem - 4.0
Subsystem          : Windows GUI
Image Flags(1)     : ImageIsDLL   (.) DebugStrip  (.) IsExecutable(x)
Image Flags(2)     : RelocatStrip(x) LineNumStrip(x) LocSymbStrip(x)
Image Flags(3)     : 32bit image (x) 16bit image (.) System file (.)
Image Flags(bit)   : 0000000100001111
HeapStack          : Reserve - 00100000/00100000, Commit - 00001000/00001000
Data Sizes         : InitDataSize : 00001000, UninitDataSize   : 0000B000
Other Sizes(1)     : CodeSize     : 00004000, ImageSize        : 00011000
Other Sizes(2)     : HeaderSize   : 00001000, OpHeaderSize(NT): 000000E0
Bases              : BaseOfCode   : 0000C000, BaseOfData       : 00010000
Alignments         : FileAlignment: 00000200, SectionAlignment: 00001000
Objects table      : ┌  Name    VirtSize   ERVA    PhysSize PhysOffs  Flags ┐
Object (     1)    : │JDR0     │0000B000│00001000│00000000│00000400│E0000080│
Object (     2)    : │JDR1     │00004000│0000C000│00004000│00000400│E0000040│
Object (     3)    : │.rsrc    │00001000│00010000│00000E00│00004400│C0000040│
Processed with(1): Packed with UPX v0.93 or v1.00 [PE]
Processed with(2): Windows executable
DOSEXE part sizes: Header 64 bytes, image 1104 bytes, overlay 19824 bytes
WINEXE part sizes: Header 49088 bytes, image -28096 bytes, overlay 0 bytes
Entrypoint         : DOS 64/00000040, RVA 64800/0000FD20, WIN 16672/00004120
Information(1)     : Overlay start from 20992/00005200
Extension (     1): DOS executable file

C:\sotm32>
```

Rada.exe after modifying the section headers.



```
C:\sotm32>upx -l RADA.EXE
                     Ultimate Packer for eXecutables
   Copyright (C) 1996, 1997, 1998, 1999, 2000, 2001, 2002, 2003, 2004
UPX 1.25w        Markus F.X.J. Oberhumer & Laszlo Molnar        Jun 29th 2004

        File size         Ratio      Format      Name
   --------------------   ------   -----------   -----------
     57344 ->     20992   36.61%    win32/pe     RADA.EXE

C:\sotm32>upx -d RADA.EXE
                     Ultimate Packer for eXecutables
   Copyright (C) 1996, 1997, 1998, 1999, 2000, 2001, 2002, 2003, 2004
UPX 1.25w        Markus F.X.J. Oberhumer & Laszlo Molnar        Jun 29th 2004

        File size         Ratio      Format      Name
   --------------------   ------   -----------   -----------
     57344 <-     20992   36.61%    win32/pe     RADA.EXE

Unpacked 1 file.

C:\sotm32>_
```

3. In one instance while analyzing the Unicode strings from the binary, a reference was found to a registry key to check the path where VMWare tools are installed. This may be one of the detection techniques used to check and exit if virtual host, which are generally used for testing is found. However during the review no significant trail was found to check this key.

   `HKLM\Software\VMware, Inc.\VMware Tools\InstallPath`

Q6. Categorize this type of malware (virus, worm...) and justify your reasoning.

A6. Considering the recent variants and techniques of virus and worms found in the wild, the strict difference between definitions of virus and worms has diminished. Most of the recent malware displays hybrid traits of infecting like a virus and spreading like worms.

This binary may be categorized as a virus as it needs some action to be performed by the victim in terms of downloading the malware and executing the same once. The above functionality could have been automated by combining certain other exploit mechanisms like download and execute via IE browser with out user’s consent.

Since the prime functionality as revealed from the binary was to conduct a DoS on a targeted server. The binary does not depict traits of a worm to spread itself. However we have found evidence within the malware to upload binary data via HTTP. Probably this mechanism is used to spread copies of it to vulnerable hosts.

Q7. Identify another tool that has demonstrated similar functionality in the past.

A7. As inferred from the binary this functionality is also demonstrated by various DoS tools based on Smurf attack.

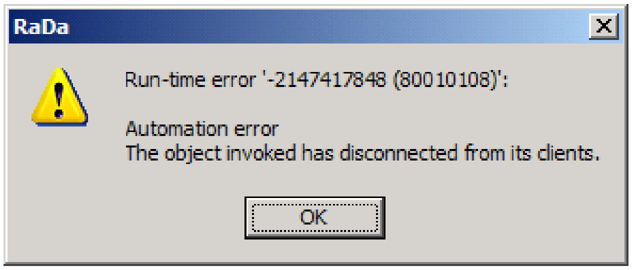
Several packet crafting tools like dsniff, hping, packetX etc can easily be used and modified to simulate similar functionality as demonstrated by RaDa.

Q8. Suggest detection and protection methods to fight against the threats introduced by this binary.

A8. Combination of various detection mechanisms is required to defend such binaries with hybrid threats. We suggest a proactive approach by having a stringent access control and content filtering policies at the gateway level to prohibit users from downloading or uploading binary data in any form.

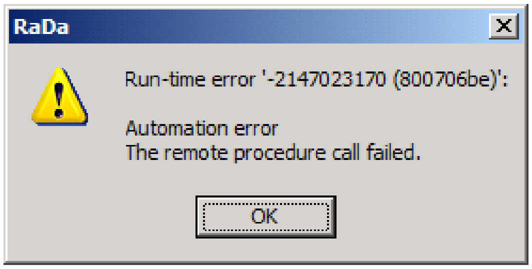
Once identified, generic IDS rules may be installed on gateway servers to detect and block offending hosts.

Considering the usage of client side scripts for uploading data it is advisable to harden clients machine by disabling windows scripting. On one of the hardened test machine with minimum software installed, the binary "rada.exe" could not execute properly even once. Few of the error message displayed during various installation attempts are as below



Run-time error '-2147417848 (80010108)':

Automation error
The object invoked has disconnected from its clients.



Run-time error '-2147023170 (800706be)':

Automation error
The remote procedure call failed.

Bonus Question:

Q9. Is it possible to interrogate the binary about the person(s) who developed this tool? In what circumstances and under which conditions?

A9. It is evident from the text found in the binary about the authors of the binary.

Below is the string as found in the decompressed binary.

```
Authors: Raul Siles & David Perez, 2004
```

Few of the components found in the binary are compiled from various sources already available on the net. The vbscript uploading form embedded in the binary can be found at http://www.motobit.com/tips/detpg_uploadvbsie.htm authored by Antonin Foller, of PSTRUH Software

UPX packer is used to compress the finalized binary.

Q10. What advancements in tools with similar purposes can we expect in the near future?

A10. Various existing tools already display advance techniques in terms of anti-debugging and anti-detection capabilities. Many of such techniques were not found to be implemented in the current binary.

The malware opted to install itself in system root drive, it would have been more difficult for a casual observer to witness its presence if the binary would have been installed in a unsuspicious or benign location.

The binary was compressed using a readily available and well-known compression software upx, which made its detection easy at an early stage. Run-time encryption was not employed. Using proprietary or modified versions of compressors and encryptions would have made disassembly and analysis a very difficult task.

It was observed that when rada.exe was specifically executed couple of times, it did not check for its already instance running and the same can be seen in the below snapshot. No attempts were made to hide from system process listing.

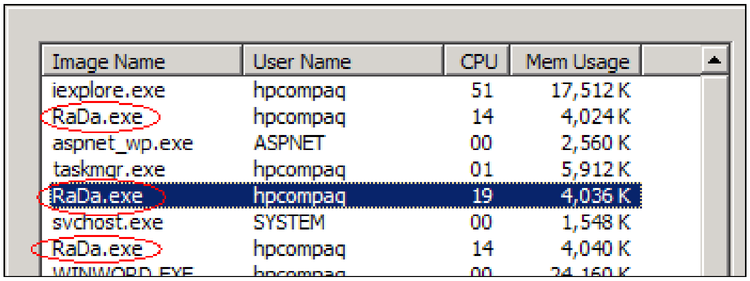


Image Name	User Name	CPU	Mem Usage
iexplore.exe	hpcompaq	51	17,512 K
RaDa.exe	hpcompaq	14	4,024 K
aspnet_wp.exe	ASPNET	00	2,560 K
taskmgr.exe	hpcompaq	01	5,912 K
RaDa.exe	hpcompaq	19	4,036 K
svchost.exe	SYSTEM	00	1,548 K
RaDa.exe	hpcompaq	14	4,040 K
WINWORD.EXE	hpcompaq	00	24,160 K

Usage of windows scripting is always prone to various client side configuration settings. Coding the transfer routines in a high-level language would have made identification and analysis difficult.

We can expect malware tools of future to be carefully designed with intent to be deceptive and thwart known identifying and debugging techniques.