References:
http://www.phrack.org/issues.html?issue=60&id=7
The Shellcoder's Handbook Second Edition Chapter 13
http://www.blackhat.com/presentations/bh-dc-08/FX/Whitepaper/bh-dc-08-fx-WP.pdf
http://www.coresecurity.com/content/killing-the-myth-cisco-ios

Naming convention programs :
naft_* : Python modules
naft-*: Python programs

naft-gfe.py: Network Appliance Forensic Toolkit - Generic Frame Extraction
naft-icd.py: Network Appliance Forensic Toolkit - IOS Core Dumps
naft-ii.py: Network Appliance Forensic Toolkit - IOS Image

naft-gfe.py: Network Appliance Forensic Toolkit - Generic Frame Extraction

This tool extracts frames from files by searching for ARP frames and IPV4 headers with valid checksums, and stores the extracted frames in a PCAP file.

Examples:

naft-gfe.py xp-laptop-2005-06-25.img.pcap xp-laptop-2005-06-25.img

```
20120217-202633: Start
20120217-202633: Reading file xp-laptop-2005-06-25.img
20120217-202649: Searching for IPv4 packets
20120217-203019: Searching for ARP frames
20120217-203020: Writing PCAP file xp-laptop-2005-06-25.img.pcap
20120217-203020: Number of identified frames: 5525
20120217-203020: Number of identified packets: 1571
20120217-203020: Number of frames in PCAP file: 3537
20120217-203020: Done
```

naft-gfe.py test.pcap MacMountainLion_10_8_3_AMDx64.vmem

```
20131013-131437: Start
20131013-131437: Reading file MacMountainLion_10_8_3_AMDx64.vmem
20131013-131445: File is too large to fit in memory
20131013-131445: Done
```

By default, naft-gte loads the complete dump file in memory. If there is not enough memory in your machine (or if you are using 32-bit Python), you will get an error.

naft-gfe.py -b result.pcap MacMountainLion_10_8_3_AMDx64.vmem

```
20131013-131524: Start
20131013-131524: Buffering file MacMountainLion_10_8_3_AMDx64.vmem
20131013-131525: Processing buffer 0x0 size 101 MB 4%
20131013-131525: Searching for IPv4 packets
20131013-131528: Searching for ARP Ethernet frames
20131013-131529: Processing buffer 0x6400000 size 101 MB 9%
```

```
20131013-131529: Searching for IPv4 packets
...
20131013-131634: Processing buffer 0x76c00000 size 101 MB 97%
20131013-131634: Searching for IPv4 packets
20131013-131636: Searching for ARP Ethernet frames
20131013-131637: Processing buffer 0x7d000000 size 48 MB 100%
20131013-131637: Searching for IPv4 packets
20131013-131639: Searching for ARP Ethernet frames
20131013-131639: Writing PCAP file test.pcap
20131013-131639: Number of identified frames:     112
20131013-131639: Number of identified packets:     77
20131013-131639: Number of frames in PCAP file:   184
20131013-131639: Done
```

Use option -b if a file is too large to load in memory. This --buffer option will load and search the dump file in blocks (by default 100MB). To prevent the search algorithms from missing split packets (packets that start in one block and end in the next block), an overlap buffer is used (by default 1MB, which is much larger than the largest IPv4 packet). To change the default size of the buffer and the overlap buffer, use options -S and -O respectively.

naft-gfe.py -d r870-coreiomem.pcap r870-coreiomem

```
Number of identified frames: 229
Number of identified packets: 12
Number of frames in PCAP file: 241
```

Option –d keeps duplicate frames.
By default naft-gfe will discard identical frames, and only include one copy of the frame in the PCAP file.

naft-gfe.py -t r870-coreiomem.bt r870-coreiomem.pcap r870-coreiomem

```
Number of identified frames: 229
Number of identified packets: 12
Number of frames in PCAP file: 220
```

Option –t creates a template for the 010 Editor:

```
// Generated
local int iCOLOR = 0x95E8FF; // Color used for highlighting
local int iToggleColor = iCOLOR;
void ToggleBackColor()
{
	if (iToggleColor == iCOLOR)
		iToggleColor = cNone;
	else
		iToggleColor = iCOLOR;
	SetBackColor(iToggleColor);
}
ToggleBackColor();
BYTE unknown1[3014];
ToggleBackColor();
BYTE frame1[74];
ToggleBackColor();
BYTE unknown2[250];
```

```
ToggleBackColor();
BYTE frame2[46];
…
```

Option -o uses the OUI list to ignore frames with MAC addresses from organizations that are not in the OUI list (http://standards.ieee.org/develop/regauth/oui/oui.txt).

Option -p searches for IPV4 headers with options. By default, naft-gfe will only identify IPV4 headers 5 units (32 bits) long. Longer headers are ignored, except when option -p is used.

naft-icd.py: Network Appliance Forensic Toolkit - IOS Core Dumps

This tool analyses IOS core dumps. In this version of the tool, we assume the memory is not corrupted (e.g. heap corruption).

Tested with IOS 12.4 and 15.0.

naft-icd.py takes a command as option:

- regions
- cwstrings
- heap
- frames
- processes
- integritycheck
- checktext
- events
- history

naft-icd.py regions r870-core
This command displays the regions found in core dump r870-core

Example:

```
Start      End        Size       Name
0x80000000 0x8002008B     131212 begin
0x8002008C 0x81A92DB7   27733292 text
0x81A92DB8 0x82CE8103   19223372 data
0x82CE8104 0x82F0AEEB    2239976 bss
0x82F0AEEC 0x873FFFFF   72306964 heap
```

naft-icd.py –w regions r870-core
Option –w writes the regions to separate files.

naft-icd.py cwstrings r870-core
This command displays CW_* strings found in the core dump.

Example:

```
CW_BEGIN:              -gs-advipservicesk9-m
```

```
CW_END:                -gs-advipservicesk9-m
CW_FAMILY:             C870
CW_FEATURE:            IP|FIREWALL|VOICE|PLUS|SSH|3DES
CW_IMAGE:              C870-ADVIPSERVICESK9-M
CW_MAGIC:
CW_MEDIA:              RAM
CW_SYSDESCR:
Cisco IOS Software, C870 Software (C870-ADVIPSERVICESK9-M), Version 12.4(6)T5,
RELEASE SOFTWARE (fc1)
Technical Support: http://www.cisco.com/techsupport
Copyright (c) 1986-2006 by Cisco Systems, Inc.
Compiled Sat 07-Oct-06 01:08 by kellythw
CW_VERSION:            12.4(6)T5
```

Option –a (--raw) is used to force the tool to search the complete file, not only the DATA region.

naft-icd.py heap r870-core
This command displays the blocks in the heap.

Example:

```
Address      Bytes     Prev     Next Ref     PrevF    NextF Alloc PC  what
82F0AEEC 0000004100 00000000 82F0BF20 001  -------- -------- 814689A0  8253A7A8
82F0BF20 0000002052 82F0AEEC 82F0C754 001  -------- -------- 81468A18  8253A7A8
82F0C754 0000014052 82F0BF20 82F0FE68 000  0        0        8015A97C  81A9B9DC
82F0FE68 0000004100 82F0C754 82F10E9C 001  -------- -------- 814689A0  8253A7A8
82F10E9C 0000120192 82F0FE68 82F2E44C 000  0        0        80374074  81A9B9DC
```

naft-icd.py –r heap r870-core
Option –r resolves names.

Examples:

```
Address      Bytes     Prev     Next Ref     PrevF    NextF Alloc PC  what
82F0AEEC 0000004100 00000000 82F0BF20 001  -------- -------- 814689A0  SSH2
Buffer
82F0BF20 0000002052 82F0AEEC 82F0C754 001  -------- -------- 81468A18  SSH2
Buffer
82F0C754 0000014052 82F0BF20 82F0FE68 000  0        0        8015A97C
(coalesced)
82F0FE68 0000004100 82F0C754 82F10E9C 001  -------- -------- 814689A0  SSH2
Buffer
82F10E9C 0000120192 82F0FE68 82F2E44C 000  0        0        80374074
(coalesced)
82F2E44C 0000020004 82F10E9C 82F332A0 001  -------- -------- 8004985C  Managed
Chunk Queue Elements
82F332A0 0000010004 82F2E44C 82F359E4 001  -------- -------- 8125AC20  List
Elements
82F359E4 0000005004 82F332A0 82F36DA0 001  -------- -------- 8125AC60  List
Headers
82F36DA0 0000000048 82F359E4 82F36E00 001  -------- -------- 81A895E8  *Init*
```

naft-icd.py –f “TTY data” heap r870-core
Option –f resolves names and selects blocks with the given name (“TTY data” in this example).

```
Address     Bytes     Prev     Next Ref     PrevF    NextF Alloc PC  what
82F3915C 0000004356 82F389CC 82F3A290 001  -------- -------- 8083D5A0  TTY data
8301B44C 0000004356 8301B400 8301C580 001  -------- -------- 8083D5A0  TTY data
83BFC824 0000004356 83BFC7D8 83BFD958 001  -------- -------- 8083D5A0  TTY data
83BFDB8C 0000004356 83BFD958 83BFECC0 001  -------- -------- 8083D5A0  TTY data
83BFECC0 0000004356 83BFDB8C 83BFFDF4 001  -------- -------- 8083D5A0  TTY data
83C13D78 0000004356 83C0F884 83C14EAC 001  -------- -------- 8083D5A0  TTY data
83C14EAC 0000004356 83C13D78 83C15FE0 001  -------- -------- 8083D5A0  TTY data
83C15FE0 0000004356 83C14EAC 83C17114 001  -------- -------- 8083D5A0  TTY data
```

naft-icd.py –f “TTY data” –s heap r870-core
Options –s dumps strings (ASCII, not UNICODE) found in the data of the block and can only be used together with option –f.

```
82F3915C 0000004356 82F389CC 82F3A290 001  -------- -------- 8083D5A0  TTY data
 82F3AE82: -line)#
 82F3A26A: CSCdr01929
 82F3AB6C: ordnumberone
 82F3AC71: ordnumberone
 82F3AD72: ordnumberone
 82F3B1F3:  --More--
 82F3AE75: r870#
 82F3AB59: -./:@_
 82F3B19D: passwordnumberone
```

Notice the presence of the password (passwordnumberone) used to log on via the console.

naft-icd.py –f “TTY data” –d heap r870-core
Options –d dumps the data of the block and can only be used together with option –f.

```
Address      Bytes     Prev     Next Ref     PrevF    NextF Alloc PC  what
82F3915C 0000004356 82F389CC 82F3A290 001  -------- -------- 8083D5A0  TTY data
 82F3918C: 43 53 43 64 72 30 31 39 32 39 00 00 00 00 00 00  CSCdr01929......
 82F3919C: 00 00 00 01 00 00 00 00 00 00 00 00 00 00 00 00  ................
 82F391AC: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
 82F391BC: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
```

naft-icd.py -f Init -s -g "CMD: " heap r870-core
Option –g greps the strings and can be used to display the history of commands and the IOS events:

```
 Address      Bytes     Prev     Next Ref     PrevF    NextF Alloc PC  what
831B17BC 0000032776 831B1720 831B97F4 001  -------- -------- 808B03EC  Init
 831B9DCC: CMD: ' rsakeypair TP-self-signed-3912050618' 00:00:10 UTC Fri Mar 1
2002
 831B99CF: CMD: 'service timestamps log datetime msec' 00:00:10 UTC Fri Mar 1
2002
 831BA3D6: CMD: ' logging synchronous' 00:00:10 UTC Fri Mar 1 2002
 831BAFE1: CMD: 'show regio' 08:12:07 UTC Fri Mar 1 2002
 831BB1E2: CMD: 'login ' 08:24:37 UTC Fri Mar 1 2002
 831B9FE5: CMD: ' no ip address' 00:00:10 UTC Fri Mar 1 2002
 831B9BE8: CMD: '   network 192.168.1.0 255.255.255.0' 00:00:10 UTC Fri Mar 1
2002
 831B99F4: CMD: PASSWORD statement not printed
 831BA5CC: CMD: ' no inservice' 00:00:10 UTC Fri Mar 1 2002
```

naft-icd.py -f Init -s -g ": %" heap r870-core

```
 Address      Bytes     Prev     Next Ref     PrevF    NextF Alloc PC  what
831B17BC 0000032776 831B1720 831B97F4 001  -------- -------- 808B03EC  Init
 831BABD0: *Mar  1 08:00:34.987: %LINK-3-UPDOWN: Interface FastEthernet2,
changed state to up
 831BA9EA: *Mar  1 08:00:33.403: %SNMP-5-COLDSTART: SNMP agent on host r870 is
undergoing a cold start
 831B9827: *Mar  1 00:00:06.379: %VPN_HW-6-INFO_LOC: Crypto engine: onboard 0
State changed to: Initialized
 831BAE20: *Mar  1 08:00:36.011: %LINEPROTO-5-UPDOWN: Line protocol on
Interface FastEthernet0, changed state to up
 831BAC23: *Mar  1 08:00:34.991: %LINK-3-UPDOWN: Interface FastEthernet1,
changed state to up
 831BA824: *Mar  1 08:00:33.099: %LINEPROTO-5-UPDOWN: Line protocol on
Interface FastEthernet0, changed state to upCMD: 'no access-list 199' 08:00:33
UTC Fri Mar 1 2002
 831BAA2A: *Mar  1 08:00:33.439: %SSH-5-ENABLED: SSH 1.99 has been enabled
```

naft-icd.py –y IOS_canary.yara heap r870-core

Option –y scans the heap with YARA rules. –y specifies the YARA rule(s) to run. A file, a directory or an @-file can be specified.

```
Address      Bytes     Prev     Next Ref     PrevF    NextF Alloc PC  what
83AB9498 0000004100 83AB9444 83ABA4CC 001  -------- -------- 80B5CC7C  8253709C
 YARA rule: IOS_canary
```

Rule IOS_canary.yara searches for a canary value inside the blocks.

```
rule IOS_canary
{
    strings:
        $canary = {FD 01 10 DF}
    condition:
        $canary
}
```

Option –yarastrings prints the found strings:

```
naft-icd.py heap -y IOS_canary.yara --yarastrings r870-core
 Address      Bytes     Prev     Next Ref     PrevF    NextF Alloc PC  what
83AB9498 0000004100 83AB9444 83ABA4CC 001  -------- -------- 80B5CC7C  8253709C
 YARA rule: IOS_canary
  000040 $canary:
  fd0110df
  '\xfd\x01\x10\xdf'
  000094 $canary:
  fd0110df
  '\xfd\x01\x10\xdf'
  000108 $canary:
  fd0110df
  '\xfd\x01\x10\xdf'
  00015c $canary:
  fd0110df
  '\xfd\x01\x10\xdf'
  0001d0 $canary:
  fd0110df
  '\xfd\x01\x10\xdf'
```

When using YARA, option –decoders can be used to decode the content of the blocks. Decords are Python programs like decoder_add1, decoder_rol1 and decoder_xor1.

Example:

```
naft-icd.py -y IOS_canary.yara --decoders decoder_xor1 heap r870-core
 Address      Bytes     Prev     Next Ref     PrevF    NextF Alloc PC  what
83A125C4 0000016388 83A10E20 83A165F8 001  -------- -------- 81284F40  82471FDC
 YARA rule (decoder: XOR 1 byte key 0x5C): IOS_canary
83AB9498 0000004100 83AB9444 83ABA4CC 001  -------- -------- 80B5CC7C  8253709C
 YARA rule: IOS_canary
```

Option –D dumps all blocks to a separate file.

naft-icd.py frames r870-core r870-coreiomem r870-coreiomem.pcap
Command frames extract frames from r870-coreiomem to PCAP file r870-coreiomem.pcap. Unlike naft-gfe.py, this command uses data found in the heap (*Packet Header*) to locate frames in iomem.

Example:

```
Address      Bytes     Prev     Next Ref     PrevF    NextF Alloc PC  what
82FD4248 0000000884 82FD40BC 82FD45EC 001  -------- -------- 8030CA24  *Packet
Header*
 07400BCA: A1 8E 00 1F 6C D0 21 AF 81 00 00 01 08 00 45 00  ....l.!.......E.
 07400BDA: 00 38 07 14 00 00 FF 01 30 FB C0 A8 01 64 C0 A8  .8......0....d..
 07400BEA: 01 01 03 01 BE 09 00 00 00 00 45 00 00 39 0D F5  ..........E..9..
 07400BFA: 00 00 7F 11 BC F3 C0 A8 01 01 D0 43              ..DEL........C
```

naft-icd.py processes r870-core
Command processes extracts the Process Array blocks to show the running processes.

Example:

```
   1 Cwe 80049B5C        0        3       0 5552/6000   0 Chunk Manager
   2 Csp 80371B90        8      341      23 2640/3000   0 Load Meter
   3 Mwe 8118AB24        4     1725       2 5300/6000   0 Spanning Tree
   4 Lst 80046D90    14780      841   17574 5484/6000   0 Check heaps
   5 Cwe 8004F930        0        1       0 5672/6000   0 Pool Manager
   6 Mst 808278AC        0        2       0 5596/6000   0 Timers
```

Option –d dumps the Process block.

naft-icd.py integritycheck r870-core
Command integritycheck checks the integrity of the heap.

Example:

```
   Check start magic:
   OK
   Check end magic:
   OK
   Check previous block:
   OK
   Check next block:
   OK
```

naft-icd.py checktext r870-core c880data-universalk9-mz.150-1.M5.bin

Command checktext compares the instructions in the code region of the core dump with the instructions in the code section of the image. These should be identical. Differences indicate changes in memory.

Example:

```
CW_SYSDESCR are equivalent:

Cisco IOS Software, C880 Software (C880DATA-UNIVERSALK9-M), Version
15.0(1)M5, RELEASE SOFTWARE (fc2)
Technical Support: http://www.cisco.com/techsupport
Copyright (c) 1986-2011 by Cisco Systems, Inc.
Compiled Wed 23-Feb-11 19:52 by prod_rel_team

text region and section are identical
```

naft-icd.py events r870-core
The events command dumps the events found in the coredump.

Example:

```
*Nov 30 07:52:19.011: %LINEPROTO-5-UPDOWN: Line protocol on Interface ATM0,
changed state to down
*Nov 30 15:58:08.293: %LINK-3-UPDOWN: Interface ATM0, changed state to up
*Nov 30 15:58:08.293: %LINK-3-UPDOWN: Interface ATM0, changed state to up
*Nov 30 15:58:09.293: %LINEPROTO-5-UPDOWN: Line protocol on Interface ATM0,
changed state to up
*Nov 30 15:58:16.689: %DIALER-6-BIND: Interface Vi2 bound to profile Di1
*Nov 30 15:58:16.689: %LINK-3-UPDOWN: Interface Virtual-Access2, changed
state to up
*Nov 30 15:58:16.689: %LINK-3-UPDOWN: Interface Virtual-Access2, changed
state to up
*Nov 30 15:58:17.617: %LINEPROTO-5-UPDOWN: Line protocol on Interface
Virtual-Access2, changed state to up
```

naft-icd.py history r870-core
The history command dumps the history log found in the coredump.

Example:

```
22:07:40 UTC Tue Nov 29 2011: show region
22:08:01 UTC Tue Nov 29 2011: show memory 0x800200E4
22:08:04 UTC Tue Nov 29 2011: show memory
22:08:18 UTC Tue Nov 29 2011: exit
23:08:38 UTC Tue Nov 22 2011: show region
23:09:35 UTC Tue Nov 22 2011: show memory
23:09:46 UTC Tue Nov 22 2011: show memory io
23:10:38 UTC Tue Nov 22 2011: exit
```

naft-ii.py: Network Appliance Forensic Toolkit - IOS Image

This tool analyses IOS image files, like this:

naft-ii.py -v c870-advipservicesk9-mz.124-6.T5.bin

```
CW_VERSION:             12.4(6)T5
CW_FAMILY:              C870
CW_FEATURE:             IP|FIREWALL|VOICE|PLUS|SSH|3DES
CW_IMAGE:               C870-ADVIPSERVICESK9-MZ
```

```
CW_SYSDESCR:           Cisco IOS Software, C870 Software (C870-ADVIPSERVICESK9-M
Z), Version 12.4(6)T5, RELEASE SOFTWARE (fc1)
Technical Support: http://www.cisco.com/techsupport
Copyright (c) 1986-2006 by Cisco Systems, Inc.
Compiled Sat 07-Oct-06 01:08 by kellythw
Entry point:           0x80020000
Number of sections:    6
Embedded MD5:          4e684ef5df1284652ef03b80b1058fab
Compressed size:       15961881
Checksum compressed:   0xA5B9F319
Calculated checksum:   0xA5B9F319 (identical)
Uncompressed size:     46957376
Image name:            C870-ADV.BIN
Checksum uncompressed: 0xBE713840
Calculated checksum:   0xBE713840 (identical)
  0         0 0 00000000          0 ''
 11 .text   1 7 00000144      13428 '\x94!\xff\xf8|\x08\x02\xa6'
 17 .rodata 1 2 000035B8       2080 '\nError :'
 33 .data   1 3 00003DD8        952 '\x00\x00\x00\x00\x00\x00\x00\x00'
  0         7 0 00F3D0C0         52 '\x00\x00\x00\r\x00\x00\x00\x18'
 33 .data   1 3 00004190   15961904 '\xfe\xed\xfa\xce\x02\xcc\x83@'
```

The tools accepts these options:

```
Options:
  --version             show program's version number and exit
  -h, --help            show this help message and exit
  -v, --verbose         verbose output
  -x, --extract         extract the compressed image
  -i, --idapro          extract the compressed image and patch it for IDA Pro
  -s, --scan            scan a set of images
  -r, --recurse         recursive scan
  -e RESUME, --resume=RESUME
                        resume an interrupted scan
  -m MD5DB, --md5db=MD5DB
                        compare md5 hash with provided CSV db
  -l LOG, --log=LOG     write scan result to log file
```

-m uses Cisco's MD5 database found here:
http://www.cisco.com/c/en/us/support/docs/csr/cisco-sr-20080516-rootkits.html