# How to catch a chameleon

Steven Seeley
steven@immunityinc.com
@net__ninja

# C:\> whoami /all?

- mr_me
- Security Researcher @ Immunity Inc
- A member of Corelan Security Team
- A python developer
- A new age exploit developer, started with Win32 not Unix :->

# Agenda

- What is 'heaper' ?
- Motivations
- Meta – data attack techniques covered by the tool
- Functional design
- Using heaper -
  - Analyze windows structs
  - Dump function pointers
  - Find writable pointers
  - Analyze the allocator state

# Agenda – cont

- Demo – Adobe Photoshop CS5 TIFF image parsing heap buffer overflow
- More on using heaper -
    - Analyzing the freelistInUse struct
    - Hooking the heap manager
    - Patching/updating/configuring heaper
    - Detecting potential meta-data attack options
- Demo – IE Fixed COL span heap buffer overflow

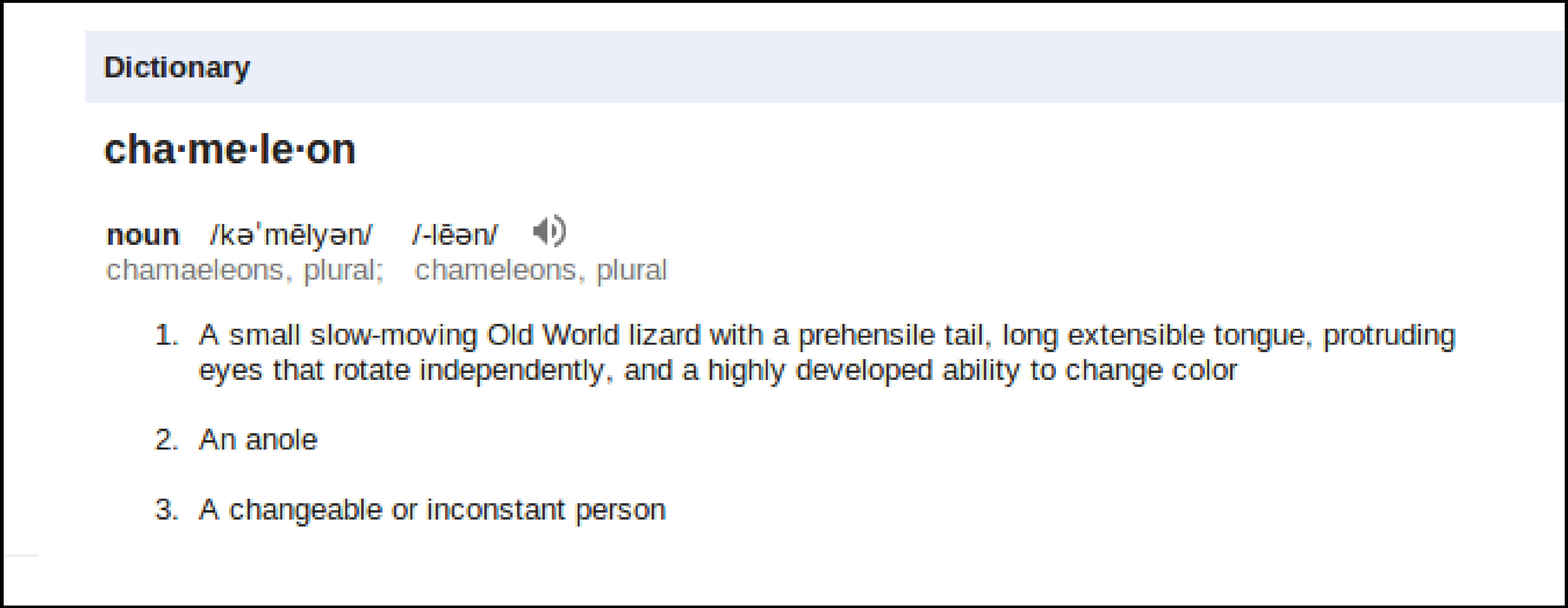
# Agenda – cont

- Limitations
- Future work
- Conclusion

But first.

An entomologist's lesson.

# Definition of a chameleon?



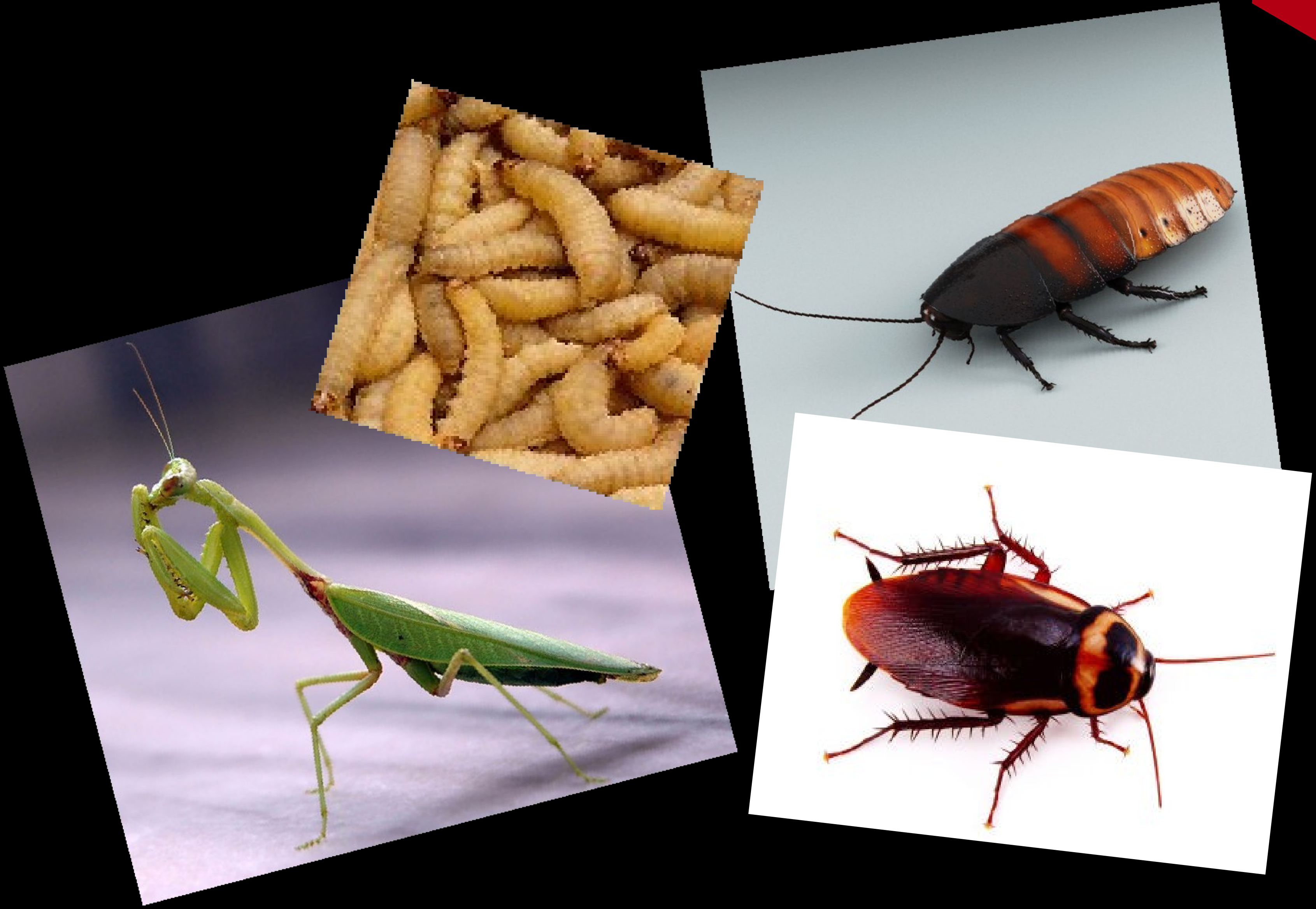
Dictionary

**cha·me·le·on**

**noun** /kəˈmēlyən/ /-lēən/ 🔊
chamaeleons, plural; chameleons, plural

1. A small slow-moving Old World lizard with a prehensile tail, long extensible tongue, protruding eyes that rotate independently, and a highly developed ability to change color
2. An anole
3. A changeable or inconstant person

# A chameleon's diet



# Similarities to the heap

| Chameleon | Heap manager |
|---|---|
| Slow moving | Slow evolution of security in heap managers for some vendors * |
| Protruding, rotating eyes | Symptoms of long debugging sessions |
| Ability to change color rapidly | Ability to change its state rapidly |
| Kills and eats bugs | Difficultly leads to disclosure, in hope of other researchers demonstrating exploitation |

* Some, meaning mostly mobile platform vendors with some exemptions

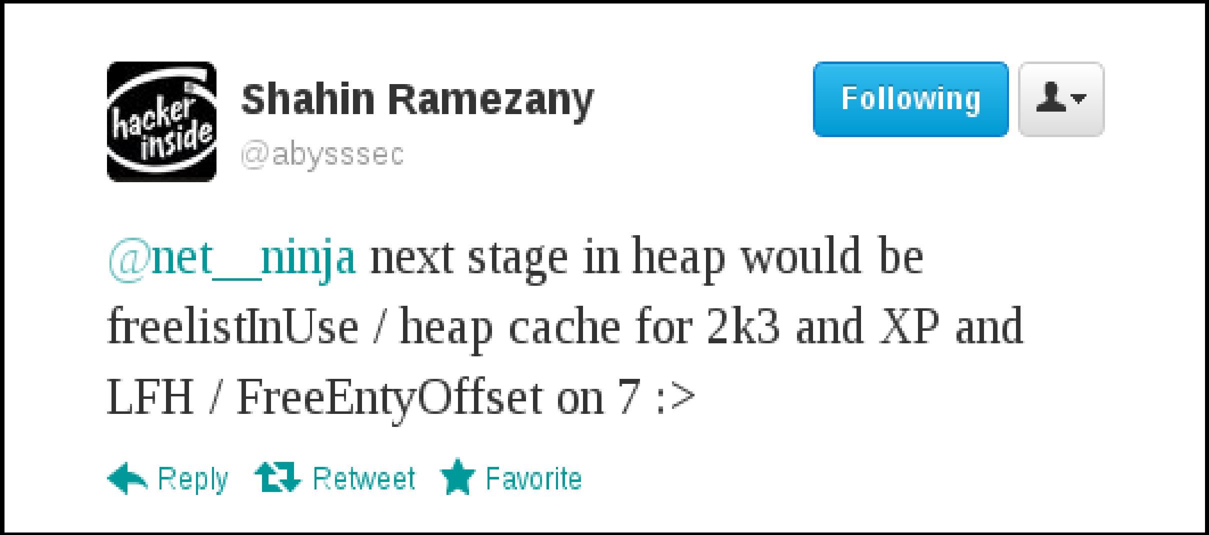
# What is Heaper

- A multi platform win32 heap analysis tool
- A plug-in for Immunity Debugger
- Developed in Python using immlib/heaplib
- An offensive focused tool:
    - Visualize the heap layout
    - Determine exploitable conditions using meta-data
    - Find application specific heap primitives
    - Find application specific function pointers
    - Modify heap structures on the fly for simulation
    - etc

# Motivations

- 3-6 months developing a heap exploit **VS** 3-6 months developing a heap analysis tool
- Meta-data attacks live longer than heap overflow bugs
- Many good heap exploit techniques exist, however often supported by poor or scattered documentation.
- Part of my self learning of advanced user mode memory corruption attacks

# Motivations



Shahin Ramezany
@abysssec

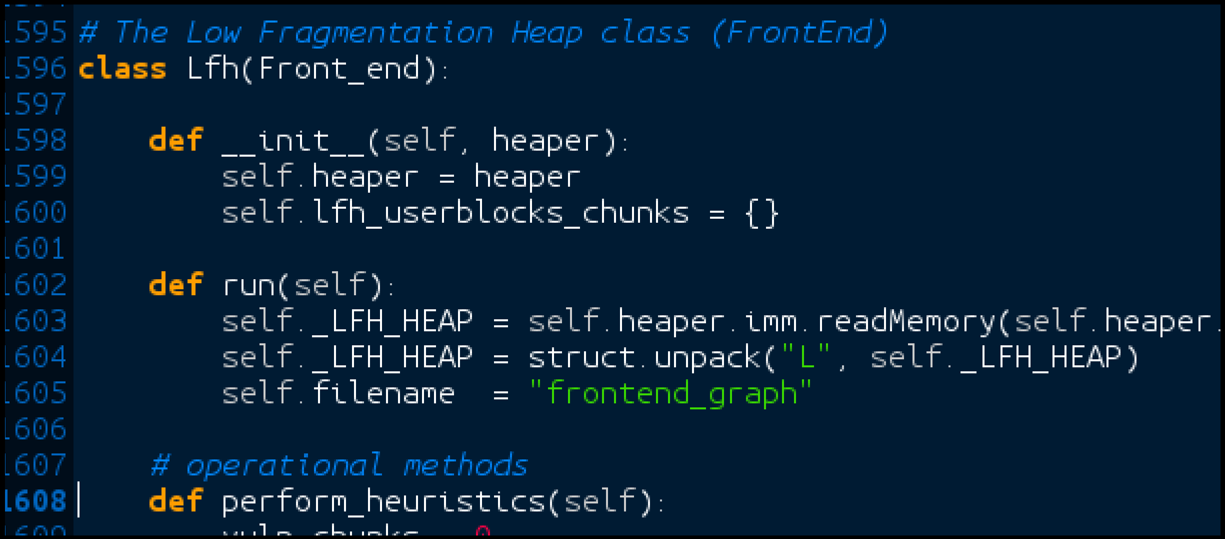
@net__ninja next stage in heap would be freelistInUse / heap cache for 2k3 and XP and LFH / FreeEntyOffset on 7 :>

# Heap exploit techniques

| Technique | Platform | Difficulty* | Reliability* | Supported |
|---|---:|---:|---:|---:|
| Coalesce unlink() | NT 5.[0/1] | 10% | 100% | **Yes** |
| VirtualAlloc block unlink() | NT 5.[0/1] | Unknown | Unknown | **No** |
| Lookaside head overwrite | NT 5.2 | 50-60% | Unknown | **Yes** |
| Freelist insert/search/relink | NT 5.2 | Unknown | Unknown | **Yes** |
| Bitmap flip | NT 5.2 | 50-60% | Unknown | **Yes** |
| Heap cache desycronisation | NT 5.2 | 90% | Unknown | **No** |
| Critical section unlink() | NT 5.2 | 50% | 70% | **No** |
| FreeEntryOverwrite | NT 6.[0/1] | 50% | 60% | **Yes** |
| Segment Offset | NT 6.[0/1] | 50% | 80% | **Yes** |
| Depth De-sync | NT 6.[0/1] | 50% | 70% | **Yes** |
| UserBlocks Overwrite | NT 6.2 | 90% | 40% | **No** |
| Application data | ANY | Unknown | Unknown | **Yes** |

Difficulty/Reliability* - estimate based on own research, will vary depending on context

# Functional design



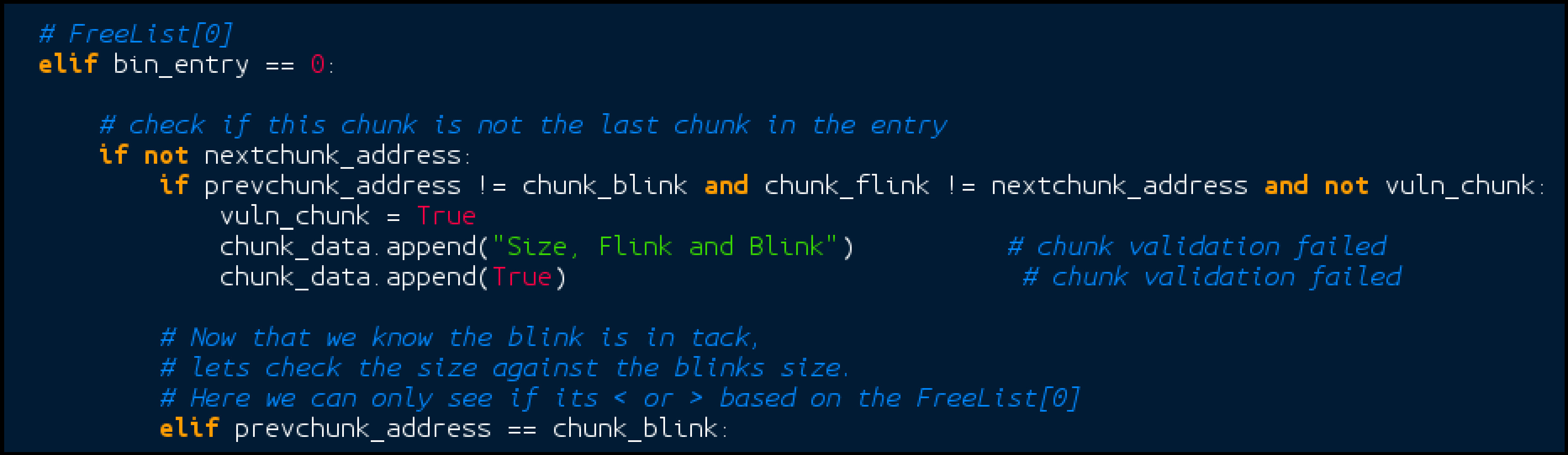
```
# The Low Fragmentation Heap class (FrontEnd)
class Lfh(Front_end):

    def __init__(self, heaper):
        self.heaper = heaper
        self.lfh_userblocks_chunks = {}

    def run(self):
        self._LFH_HEAP = self.heaper.imm.readMemory(self.heaper.
        self._LFH_HEAP = struct.unpack("L", self._LFH_HEAP)
        self.filename  = "frontend_graph"

    # operational methods
    def perform_heuristics(self):
        vuln_chunks = 0
```

- Object oriented design
- Easily extend-able
- Chunk validation based on allocator ordering & categorization
- General heuristics check per allocator



```
# FreeList[0]
elif bin_entry == 0:

    # check if this chunk is not the last chunk in the entry
    if not nextchunk_address:
        if prevchunk_address != chunk_blink and chunk_flink != nextchunk_address and not vuln_chunk:
            vuln_chunk = True
            chunk_data.append("Size, Flink and Blink")          # chunk validation failed
            chunk_data.append(True)                              # chunk validation failed

        # Now that we know the blink is in tack,
        # lets check the size against the blinks size.
        # Here we can only see if its < or > based on the FreeList[0]
        elif prevchunk_address == chunk_blink:
```

# Functional design

## Chunk validation:

- Lets say we have chunk 0xBADF00D in FreeList[0].
- We know relative offsets:
  - 0xBADF00D+0x0 is the size
  - 0xBADF00D+0x2 is the previous chunks size
  - 0xBADF00D+0x4 is the cookie
  - 0xBADF00D+0x8 is the Flink/Blink

Therefore, we can validate the chunk based on its positioning!

# Functional design

## Chunk validation:

-> Windows 2000/XP FreeList[0]

```
If not (previous_chunk_size < current_chunk_size) or not (next_chunk_size > current_chunk_size) or not (previous_chunk_addr != next_chunk_addr):
    chunk overwrite detected!
```

-> Windows 7 LFH (size is encoded)

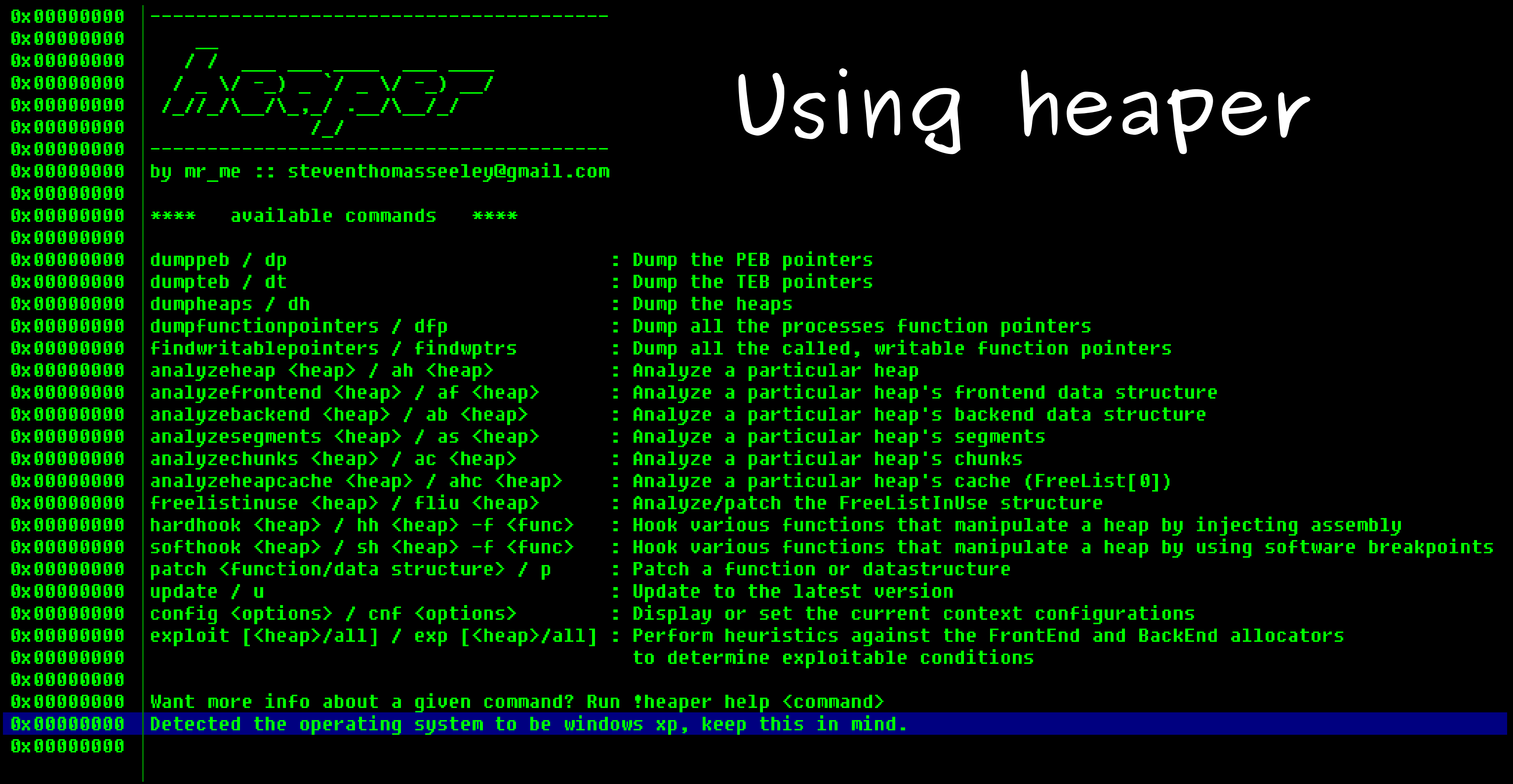
```
result = "%x" % (encoded_header ^ self.heaper.pheap.EncodingKey)
if (int(a+block.BaseIndex) == 0x7f or int(a+block.BaseIndex) == 0x7ff):
    decoded_size = int(result[len(result)-4:len(result)],16)
    if decoded_size >  int(a+block.BaseIndex):
        chunk overwrite detected!
```

# Functional design

## Graphing:

- We all know that little green men are hard to understand
- Uses pydot/graphviz/pyparser (the same engine in PaiMei RE framework)
- Again, extensible, graphing is done in its own method using a customized struct based on the allocator type (LFH/Freelist/ListHint/Lookaside)
- chunk validation is applied within the graphing engine too
-

# Using heaper



```
0x00000000 | ----------------------------------------
0x00000000 |    __
0x00000000 |   / /  ___ ___ ____  ___ ____
0x00000000 |  / _ \/ -_) _ `/ _ \/ -_) __/
0x00000000 | /_//_/\__/\_,_/ .__/\__/_/
0x00000000 |              /_/
0x00000000 | ----------------------------------------
0x00000000 | by mr_me :: steventhomasseeley@gmail.com
0x00000000 |
0x00000000 | ****   available commands   ****
0x00000000 |
0x00000000 | dumppeb / dp                            : Dump the PEB pointers
0x00000000 | dumpteb / dt                            : Dump the TEB pointers
0x00000000 | dumpheaps / dh                          : Dump the heaps
0x00000000 | dumpfunctionpointers / dfp              : Dump all the processes function pointers
0x00000000 | findwritablepointers / findwptrs        : Dump all the called, writable function pointers
0x00000000 | analyzeheap <heap> / ah <heap>          : Analyze a particular heap
0x00000000 | analyzefrontend <heap> / af <heap>      : Analyze a particular heap's frontend data structure
0x00000000 | analyzebackend <heap> / ab <heap>       : Analyze a particular heap's backend data structure
0x00000000 | analyzesegments <heap> / as <heap>      : Analyze a particular heap's segments
0x00000000 | analyzechunks <heap> / ac <heap>        : Analyze a particular heap's chunks
0x00000000 | analyzeheapcache <heap> / ahc <heap>    : Analyze a particular heap's cache (FreeList[0])
0x00000000 | freelistinuse <heap> / fliu <heap>      : Analyze/patch the FreeListInUse structure
0x00000000 | hardhook <heap> / hh <heap> -f <func>   : Hook various functions that manipulate a heap by injecting assembly
0x00000000 | softhook <heap> / sh <heap> -f <func>   : Hook various functions that manipulate a heap by using software breakpoints
0x00000000 | patch <function/data structure> / p     : Patch a function or datastructure
0x00000000 | update / u                              : Update to the latest version
0x00000000 | config <options> / cnf <options>        : Display or set the current context configurations
0x00000000 | exploit [<heap>/all] / exp [<heap>/all] : Perform heuristics against the FrontEnd and BackEnd allocators
0x00000000 |                                           to determine exploitable conditions
0x00000000 |
0x00000000 | Want more info about a given command? Run !heaper help <command>
0x00000000 | Detected the operating system to be windows xp, keep this in mind.
0x00000000 |
```

Steven Seeley - Ruxcon 2012

# Analyze windows structs

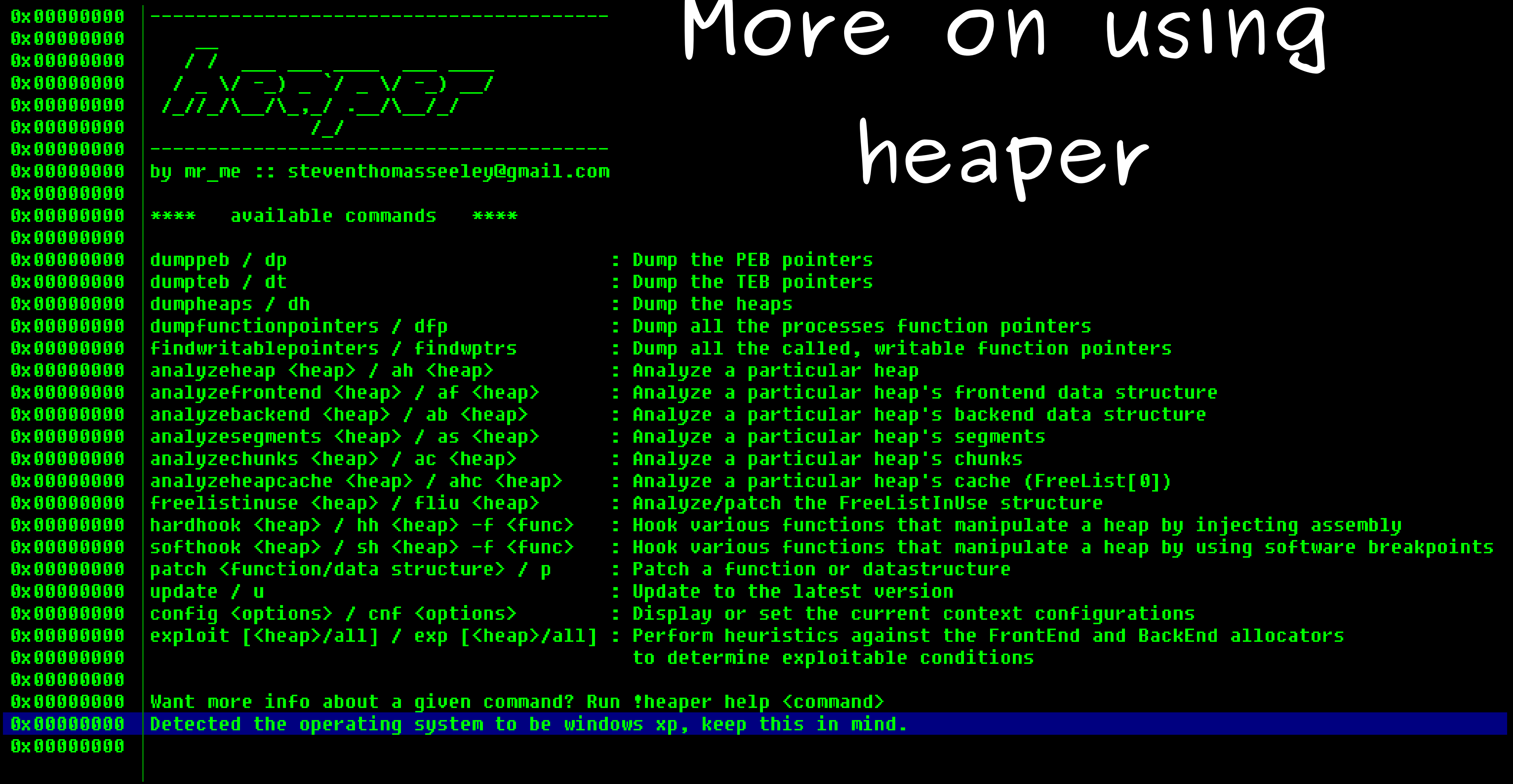
# Dump function pointers

# Find writable pointers

# Analyze the allocator state

# Demo
# Adobe Photoshop CS5 TIFF image parsing heap buffer overflow

# More on using heaper



```
0x00000000  ----------------------------------------
0x00000000      __
0x00000000     / /_  ___  ____ _____  ___  _____
0x00000000    / __ \/ _ \/ __ `/ __ \/ _ \/ ___/
0x00000000   / / / /  __/ /_/ / /_/ /  __/ /
0x00000000  /_/ /_/\___/\__,_/ .___/\___/_/
0x00000000                  /_/
0x00000000  ----------------------------------------
0x00000000  by mr_me :: steventhomasseeley@gmail.com
0x00000000
0x00000000  ****   available commands   ****
0x00000000
0x00000000  dumppeb / dp                             : Dump the PEB pointers
0x00000000  dumpteb / dt                             : Dump the TEB pointers
0x00000000  dumpheaps / dh                           : Dump the heaps
0x00000000  dumpfunctionpointers / dfp               : Dump all the processes function pointers
0x00000000  findwritablepointers / findwptrs         : Dump all the called, writable function pointers
0x00000000  analyzeheap <heap> / ah <heap>           : Analyze a particular heap
0x00000000  analyzefrontend <heap> / af <heap>       : Analyze a particular heap's frontend data structure
0x00000000  analyzebackend <heap> / ab <heap>        : Analyze a particular heap's backend data structure
0x00000000  analyzesegments <heap> / as <heap>       : Analyze a particular heap's segments
0x00000000  analyzechunks <heap> / ac <heap>         : Analyze a particular heap's chunks
0x00000000  analyzeheapcache <heap> / ahc <heap>     : Analyze a particular heap's cache (FreeList[0])
0x00000000  freelistinuse <heap> / fliu <heap>       : Analyze/patch the FreeListInUse structure
0x00000000  hardhook <heap> / hh <heap> -f <func>    : Hook various functions that manipulate a heap by injecting assembly
0x00000000  softhook <heap> / sh <heap> -f <func>    : Hook various functions that manipulate a heap by using software breakpoints
0x00000000  patch <function/data structure> / p      : Patch a function or datastructure
0x00000000  update / u                               : Update to the latest version
0x00000000  config <options> / cnf <options>         : Display or set the current context configurations
0x00000000  exploit [<heap>/all] / exp [<heap>/all]  : Perform heuristics against the FrontEnd and BackEnd allocators
0x00000000                                             to determine exploitable conditions
0x00000000
0x00000000  Want more info about a given command? Run !heaper help <command>
0x00000000  Detected the operating system to be windows xp, keep this in mind.
0x00000000
```

# Analyze the freelistinuse

# hook the heap manager

# Patch/update/config

# Demo
# IE Fixed col span heap buffer overflow

# Detecting potential meta-data attack options

- We know it can be hard to understand the little green men…
- Answer: visualize a the heap layout for:
  - chunk overwrites
  - Heap primitives
- Can be separated based on bin size (good for large heap structures).

# Limitations

- Does not analyze LFH on XP
- Does not analyze LFH on Windows 8
- Supports only a limited number of meta-data attacks for now
- Does not log analysis findings external to the debugger
- Needs a decent heap search function

# Conclusion

- Run-time analysis of the heap to detect meta-data attack conditions is complex
- Some form of SMT solver maybe more applicable to this type of analysis :->
- Immunity will continue to be a leader in the development and application of heap exploitation techniques
-

Miami