CPU: 0 PID: 3851 Comm: insmod Tainted: GF D O 3.13.0-32-generic #57~precise1-Ubuntu
Hardware name: VMware, Inc. VMware Virtual Platform/440BX Desktop Reference Platform, BIOS 6.00 05/20/2014
task: ffff88002a94afe0 ti: ffff880003214000 task.ti: ffff880003214000
RIP: 0010:[<ffffffffa0388611>] [<ffffffffa0388611>] anti_forensic_init+0x81/0x1b0 [suterusu]
RSP: 0018:ffff880003215dd8 EFLAGS: 00010246
RAX: 0000000000000000 RBX: 0000000000000000 RCX: ffffffffa03a21a0
RDX: 0000000000000018 RSI: 00000000000000d0 RDI: 0000000000017ac0
RBP: ffff880003215dd8 R08: ffff88003fc17ac0 R09: ffffffffa0386140
R10: 0000000000000779 R11: 205d74696b6c6f66 R12: ffffffffa001d000
R13: 0000000000000000 R14: 0000000000000000 R15: 0000000000000000
FS: 00007fdee9f76700(0000) GS:ffff88003fc00000(0000) knlGS:0000000000000000
CS: 0010 DS: 0000 ES: 0000 CR0: 0000000080050033
CR2: 0000000000000000 CR3: 0000000051da000 CR4: 00000000001407f0
Stack:
ffff880003215de8 ffffffffa001a06a ffff880003215e68 ffffffff8100215a
0000000002329010 0000000000000000 ffff880003215e38 ffffffff8105cb33
0000000000000000 0000000000000000 0000000002329010 ffffffffa001d000

$ Circumventing Forensic Live-acquisition Tools

> Rootkits for dubious defensive purposes

> Yonne de Bruijn (yonne.debruijn@os3.nl)

Digital Forensics

$ retrieve (evidentiary) data | form chain of evidence | prove stuff

› like normal forensics, but digital…

$ non-volatile sources: persistent storage

$ volatile sources: network settings, RAM

$ offline (dead) acquisition: device is turned off

$ online (live) acquisition: device is turned on

Live-Acquisition

$ Reasons

> Full Disk Encryption (FDE)

> Leave system running to reduce investigation “noise”

$ Process

> order of volatility: first most volatile storage

> Windows: plethora of information

> Linux: …

Anti-Forensics (AF)

$ Increase difficulty of digital forensic process

  > or completely prevent

$ common techniques:

  > data hiding (slack space)

  > data destruction

  > encryption

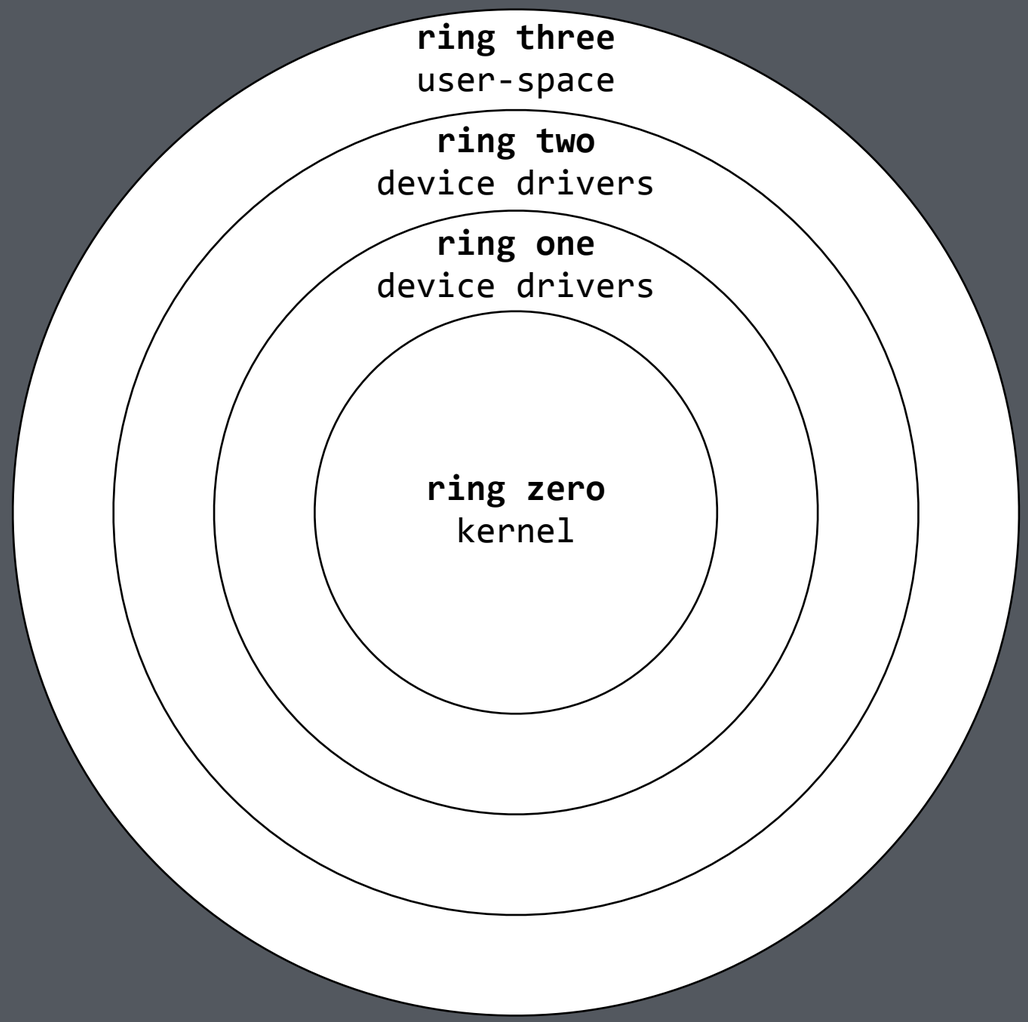
Rootkit - Theory

$ “A rootkit is a tool that is designed to hide itself and other processes, data, and/or activity on a system” (Blunden, 2013)

$ Usually: persistent backdoors, root access, i.e. malicious stuff

$ Ring Zero Rootkit: highest privileges, can intercept commands from user-space

$ Hooking system calls: attach own code to system call



Research Question

$ What acquisition tools are available and used?

$ How can a system defend against those tools?

$ Can the tools or procedures be improved?

Forensics Wiki

$ imagers:

> dd

> or: dcfldd (forensic counterpart)

> or: dd_rescue

> or: pretty much any block-level copy tool

Silk Road

$ Online Drug Market

  > Alleged owner arrested in 2013

  > Sentenced to life, based on, amongst others: live-acquisition

$ Reddit users retrieved court transcripts

  > suspect used Ubuntu + FDE –> live-acquisition

  > forensic toolkit: tar, dd, a camera running 40 minutes slow, and a good batch of ignorance

Dutch High Tech Crime Unit

$ dd

$ cp/tar, or other common copy tools

$ FTK/EnCase –> standard

$ Encrypted evidence:

  › try publicly known exploits

  › other channels (maybe less secure in the past)

  › most effective? –> “rubber hose” decryption

$ don’t often encounter AF

  › if they do –> simple stuff

Problem

$ Tools run on suspect system:

- > insecure environment
- > use system tools –> might be patched to return garbage
- > bring own tools –> might taint evidentiary system
  - > and… still using system kernel

Related Work

$ DDefy (Bilby, 2006) Windows rootkit

  > defensive rootkit –> not just for attacking?

    > actively prevents *dd* from acquiring certain files

$ Bunden (2009) warns for AF rootlets

$ Stüttgen & Cohen (2013) identified, exploited and patched issues in memory acquisition

Common Prevention

- $ Check:
  - › */sysfs* and */proc* for loaded kernel modules (LKM)
  - › common signs of encryption –> implies don’t turn off device
  - › other scripts:
    - › sometimes used to null route logs, shred data
  - › check for known AF applications
  - › does not seem hardened against advanced rootkits

Putting it together

$ focus: tar, dd/dcfldd and FTK Imager
  CLI –> proprietary tool

$ goal: intercept tools and present
  different data, preferably without
  crashing them

$ weapon: ring zero rootkit

  > easy to develop, could just as well
    run directly in kernel

Considerations

$ Control: must be hard to detect (no control application!)

  > hook open system call and parse for magic control strings

$ Hide traces:

  > hide fake data

  > hide rootkit from */sysfs* and */proc*

Interception

$ Return fake data:

  > tar: other user directory -> framing

  > imagers: from clean image located in filesystem

$ Trigger based, i.e. need a detection mechanism

Command Detection

$ Hook system calls used by tools

$ Parse calls for *magic strings*:

> if(open(/dev/sda, params)

> { fake = open(/clean,params);

> return fake; }

$ Comparable for *tar*

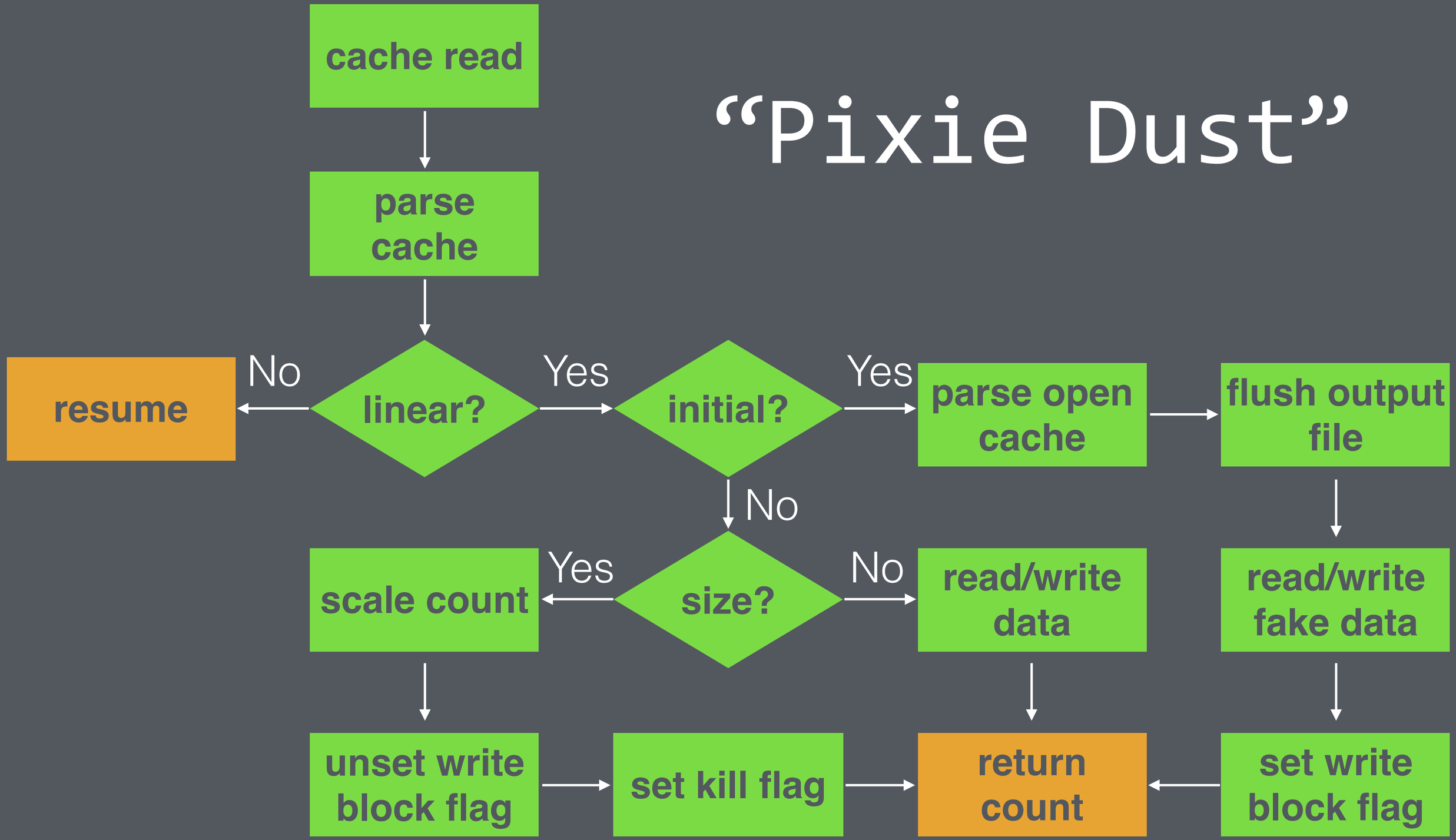
$ Success?

Demo

Linear Detection

- $ Command detection not very flexible, easily broken
  - > better add some “pixie dust” to harden it!
- $ What if we could detect behaviour…
  - > note: only implemented for dd/dcfldd

“Pixie Dust”


cache read
parse cache
linear?
No
resume
Yes
initial?
Yes
parse open cache
flush output file
No
size?
Yes
scale count
No
read/write data
read/write fake data
unset write block flag
set kill flag
return count
set write block flag

Demo

Results

$ Success!:

- detects and intercepts dd/dcfldd
- no data from */dev/sda* present in output file!

$ Issues:

- output image is corrupted
- horrible effect on read/write speeds
- Some false-positives:
  - i.e. sometimes dumps fake MBR in *nano* editor

Prevention (1)

$ Move acquisition to kernel:

  > no need for interception-sensitive system calls

    > direct access to virtual file system (VFS): vfs_read/vfs_write

$ Encrypt communication between user-space and kernel-space (linux/crypt.h)

  > prevents parsing/changing the system calls

Prevention (2)

$ Problem: very rootkit minded

$ What if directly included in kernel –> anti-forensic kernel

› need dedicated hardware solutions

› can not utilise kernel –> directly talk to hardware

› so… how ‘bout hiding in firmware?

Conclusion (1)

- $ Forensics toolkit: commonly available tools
- $ Anti-forensic scenarios: many! in full control
  - $ Prevention
    - > rootkits: create secure environment
    - > tweaked kernel: difficult
      - > hardware based acquisitions

Conclusion (2)

- Realistic threat to digital forensic process
  - not yet seen in the wild
    - but technologically skilled attacker is certainly capable

Future Work

$ Extend linear detection to tar

$ Develop acquisition kernel-model

$ Implement code directly in kernel –>
anti-forensic kernel!

$ Debug, clean and expand

$ Questions?

> source-code: https://bitbucket.org/yonne/aftoolkit