

# De-Anonymizing Live CDs through Physical Memory Analysis

Andrew Case

Senior Security Analyst



Digital Forensics Solutions

# Speaker Background

- Computer Science degree from the University of New Orleans
- Former Security Consultant for Neohapsis
- Worked for Digital Forensics Solutions since 2009
- Work experience ranges from penetration testing to reverse engineering to forensics investigations/IR to related research

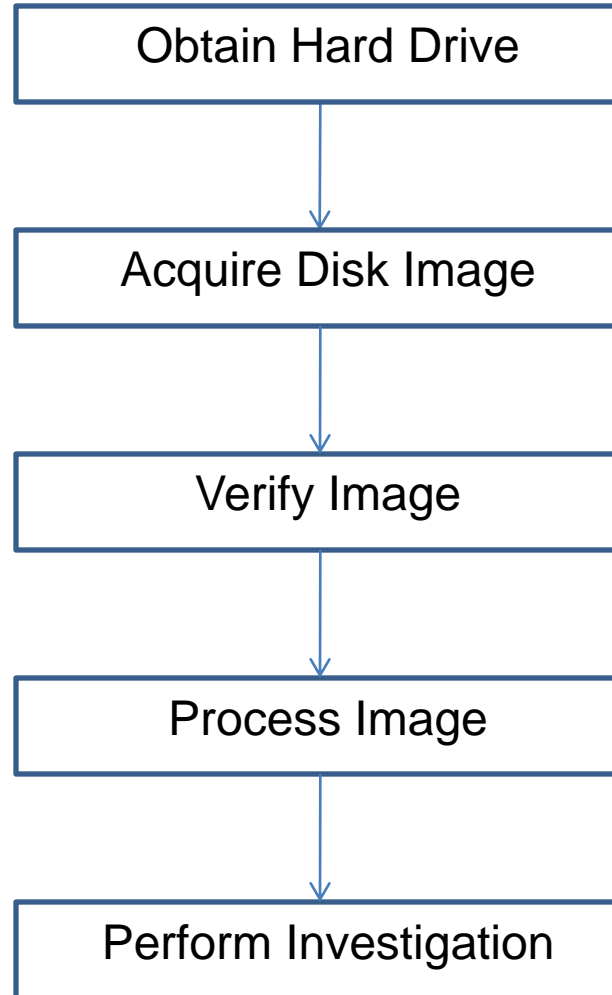


# Agenda

- Discuss Live CDs and how they disrupt the normal forensics process
- Present research that enables traditional investigative techniques against live CDs
- Discuss issues with Tor's insecure handling of memory and present preliminary memory analysis research



# Normal Forensics Process



# Traditional Analysis Techniques

- Timelining of activity based on MAC times
- Hashing of files
- Indexing and searching of files and unallocated space
- Recovery of deleted files
- Application specific analysis
  - Web activity from cache, history, and cookies
  - E-mail activity from local stores (PST, Mbox, ...)

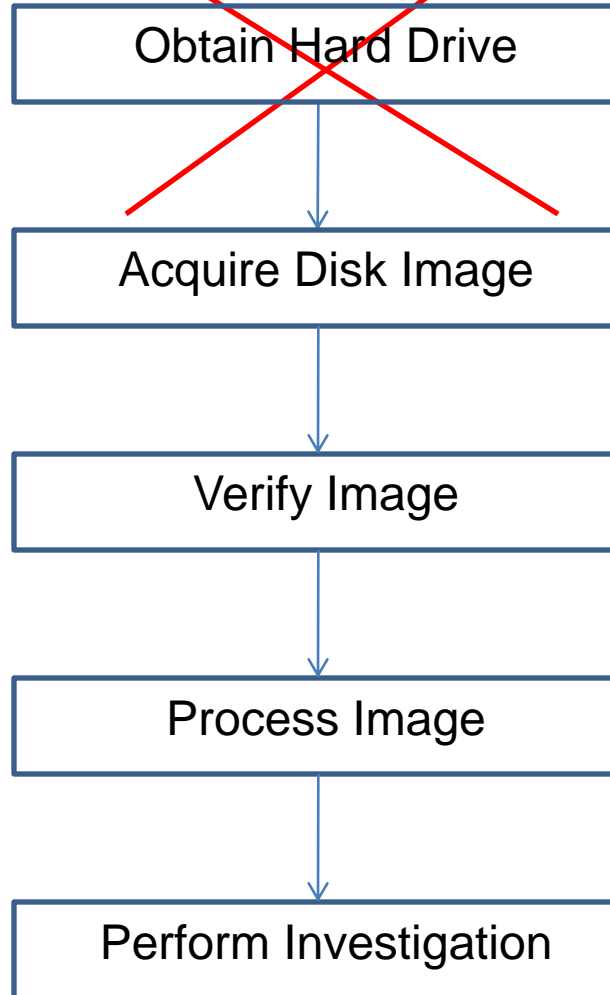


# Problem of Live CDs

- Live CDs allow users to run an operating system and all applications entirely in RAM
- This makes traditional digital forensics (examination of disk images) impossible
- All the previously listed analysis techniques cannot be performed



# The Problem Illustrated



# No Disks or Files, Now What?

- All we can obtain is a memory capture
- With this, an investigator is left with very limited and crude analysis techniques
- Can still search, but can't map to files or dates
  - No context, hard to present coherently
- File carving becomes useless
  - Next slide
- Good luck in court



# File Carving

- Used extensively to recover previously deleted files/data
- Uses a database of headers and footers to find files within raw byte streams such as a disk image
- Finds instances of each header followed by the footer
- Example file formats:
  - JPEG - \xff\xd8\xff\xe0\x00\x10 - \xff\xd9
  - GIF - \x47\x49\x46\x38\x37\x61 - \x00\x3b



# File Carving Cont.

- File carving relies on contiguous allocation of files
  - Luckily modern filesystems strive for low fragmentation
- Unfortunately for memory analysis, physical pages for files are almost never allocated contiguously
  - Page size is only 4k so no structured file will fit
  - Is the equivalent of a completely fragmented filesystem



# People Have Caught On...

- The Amnesic Incognito Live System (TAILS) [1]
  - “No trace is left on local storage devices unless explicitly asked.”
  - “All outgoing connections to the Internet are forced to go through the Tor network”
- Backtrack [2]
  - “ability to perform assessments in a purely native environment dedicated to hacking.”



# What It Really Means...

- Investigators without deep kernel internals knowledge and programming skill are basically hopeless
- It is well known that the use of live CDs is going to defeat most investigations
  - Main motivation for this work
  - Plenty anecdotal evidence of this can be found through Google searches



# What is the Solution?

- Memory Analysis!
  - It is the only method we have available...
- This Analysis gives us:
  - The complete file system structure including file contents and metadata
  - Deleted Files (Maybe)
  - Userland process memory and file system information



# Goal 1: Recovering the File System

- Steps needed to achieve this goal:
  1. Understand the in-memory filesystem
  2. Develop an algorithm that can enumerate directory and files
  3. Recover metadata to enable timelining and other investigative techniques



# The In-Memory Filesystem

- AUFS (AnotherUnionFS)
  - <http://aufs.sourceforge.net/>
  - Used by TAILS, Backtrack, Ubuntu 10.04 installer, and a number of other Live CDs
  - Not included in the vanilla kernel, loaded as an external module



# AUFS Internals

- Stackable filesystem
  - Presents a multilayer filesystem as a single one to users
  - This allows for files created after system boot to be transparently merged on top of read only CD
- Each layer is termed a branch
  - In the live CD case, one branch for the CD, and one for all other files made or changed since boot



# AUFS Userland View of TAILS

```
# cat /proc/mounts
```

```
aufs / aufs rw,relatime,si=4ef94245,noxino
```

```
/dev/loop0 /filesystem.squashfs squashfs
```

```
tmpfs /live/cow tmpfs
```

```
tmpfs /live tmpfs rw,relatime
```

Mount  
points  
relevant  
to AUFS

```
# cat /sys/fs/aufs/si_4ef94245/br0
```

```
/live/cow=rw
```

```
# cat /sys/fs/aufs/si_4ef94245/br1
```

```
/filesystem.squashfs=rr
```

The  
mount  
point of  
each  
AUFS  
branch



# Forensics Approach

- No real need to copy files from the read-only branch
  - Just image the CD
- On the other hand, the writable branch contains every file that was created or modified since boot
  - Including metadata
  - No deleted ones though, more on that later



# Linux Internals Overview I

- struct dentry
  - Represents a directory entry (directory, file, ...)
  - Contains the name of the directory entry and a pointer to its inode structure
- struct inode
  - FS generic, in-memory representation of a disk inode
  - Contains address\_space structure that links an inode to its file's pages
- struct address\_space
  - Links physical pages together into something useful
  - Holds the search tree of pages for a file



# Linux Internals Overview II

- Page Cache
  - Used to store *struct page* structures that correspond to physical pages
  - `address_space` structures contain linkage into the page cache that allows for ordered enumeration of all physical pages pertaining to an inode
- Tmpfs
  - In-memory filesystem
  - Used by TAILS to hold the writable branch



# Enumerating Directories

- Once we can enumerate directories, we can recover the whole filesystem
- Not as simple as recursively walking the children of the file system's root directory
- AUFS creates hidden dentries and inodes in order to mask branches of the stacked filesystem
- Need to carefully interact between AUFS and tmpfs structures



# Directory Enumeration Algorithm

- 1) Walk the super blocks list until the “aufs” filesystem is found
  - This contains a pointer to the root dentry
- 2) For each child dentry, test if it represents a directory

If the child is a directory:

- Obtain the hidden directory entry (next slide)
- Record metadata and recurse into directory

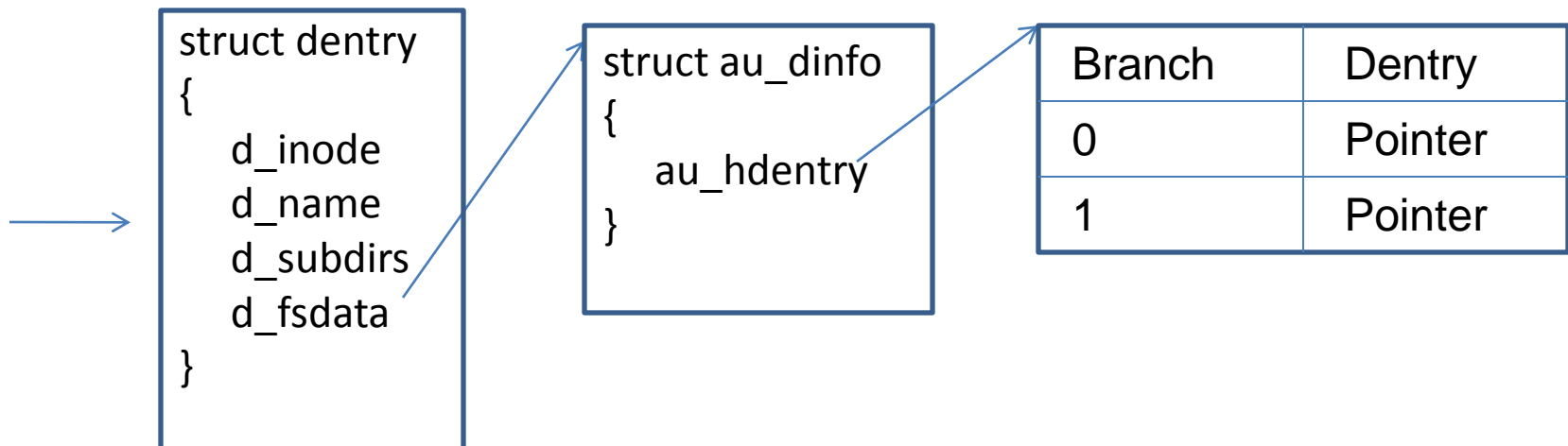
If the child is a regular file:

- Obtain the hidden inode and record metadata



# Obtaining a Hidden Directory

- Each kernel dentry stores a pointer to an *au\_dinfo* structure inside its *d\_fsdata* member
- The *di\_hdentry* member of *au\_dinfo* is an array of *au\_hdentry* structures that embed regular kernel dentries



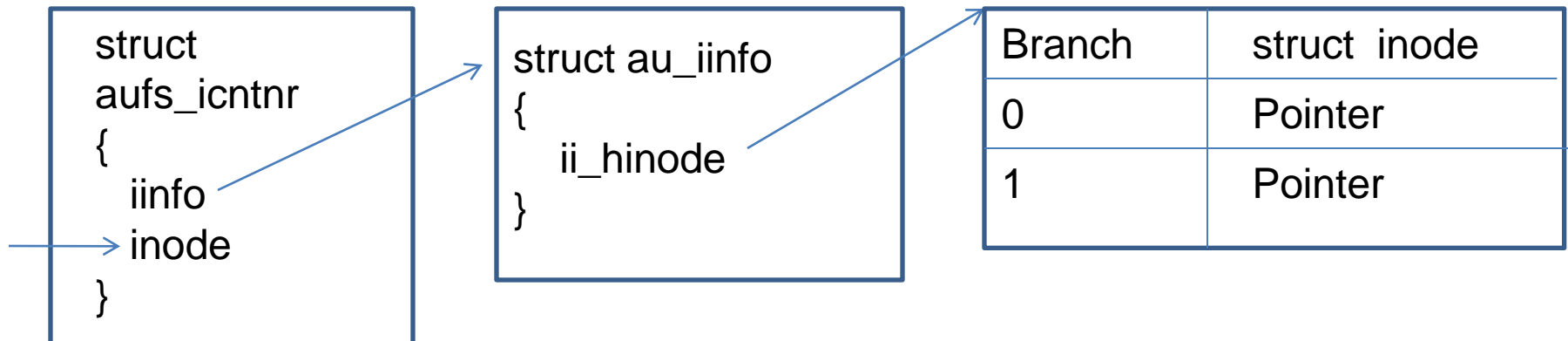
# Obtaining Metadata

- All useful metadata such as MAC times, file size, file owner, etc is contained in the hidden inode
- This information is used to fill the *stat* command and *istat* functionality of the Sleuthkit
- Timelining becomes possible again



# Obtaining a Hidden Inode

- Each aufs controlled inode gets embedded in an *aufs\_icntnr*
- This structure also embeds an array of *au\_hinode* structures which can be indexed by branch number to find the hidden inode of an exposed inode



# Goal 2: Recovering File Contents

- The size of a file is kept in its inode's *i\_size* member
- An inode's *page\_tree* member is the root of the radix tree of its physical pages
- In order to recover file contents this tree needs to be searched for each page of a file
- The lookup function returns a *struct page* which leads to the backing physical page



# Recovering File Contents Cont.

- Indexing the tree in order and gathering of each page will lead to accurate recovery of a whole file
- This algorithm assumes that swap isn't being used
  - Using swap would defeat much of the purpose of anonymous live CDs
- Tmpfs analysis is useful for every distribution
  - Many distros mount /tmp using tmpfs, shmem, etc



# Goal 3: Recovering Deleted Info

- Discussion:
  1. Formulate Approach
  2. Discuss the *kmem\_cache* and how it relates to recovery
  3. Attempt to recover previously deleted file and directory names, metadata, and file contents



# Approach

- We want orderly recovery
- To accomplish this, information about deleted files and directories needs to be found in a non-standard way
  - All regular lists, hash tables, and so on lose track of structures as they are deleted
- Need a way to gather these structures in an orderly manner
  - *kmem\_cache* analysis to the rescue!



# Recovery through *kmem\_cache* analysis

- A *kmem\_cache* holds all structures of the same type in an organized manner
  - Allows for instant allocations & deallocations
  - Used for handling of process, memory mappings, open files, and many other structures
- Implementation controlled by allocator in use
  - SLAB and SLUB are the two main ones



# *kmem\_cache* Internals

- Both allocators keep track of allocated and previously de-allocated objects on three lists:
  - *full*, in which all objects are allocated
  - *partial*, a mix of allocated and de-allocated objects
  - *free*, previously freed objects\*
- The free lists are cleared in an allocator dependent manner
  - SLAB leaves free lists in-tact for long periods of time
  - SLUB is more aggressive



# *kmem\_cache* Illustrated

- */proc/slabinfo* contains information about each current *kmem\_cache*
- Example output:

<i># name</i>	<i>&lt;active_objs&gt;</i>	<i>&lt;num_objs&gt;</i>
<i>task_struct</i>	<i>101</i>	<i>154</i>
<i>mm_struct</i>	<i>76</i>	<i>99</i>
<i>filp</i>	<i>901</i>	<i>1420</i>

The difference between *num\_objs* and *active\_objs* is how many free objects are being tracked by the kernel



# Recovery Using *kmem\_cache* Analysis

- Enumeration of the lists with free entries reveals previous objects still being tracked by the kernel
  - The kernel does not clear the memory of these objects
- Our previous work has demonstrated that much previously de-allocated, forensically interesting information can be leveraged from these caches [4]



# Recovering Deleted Filesystem Structure

- Both Linux kernel and aufs directory entries are backed by the *kmem\_cache*
- Recovery of these structures reveals names of previous files and directories
  - If *d\_parent* member is still in-tact, can place entries within file system



# Recovering Previous Metadata

- Inodes are also backed by the *kmem\_cache*
- Recovery means we can timeline again
- Also, the dentry list of the AUFS inodes still have entries (strange)
  - This allows us to link inodes and dentries together
  - Now we can reconstruct previously deleted file information with not only file names & paths, but also MAC times, sizes, inode numbers, and more



# Recovering File Contents – Bad News

- Again, inodes are kept in the *kmem\_cache*
- Unfortunately, page cache entries are removed upon deallocation, making lookup impossible
  - A large number of pointers would need to stay intact for this to work
- This removes the ability to recover file contents in an orderly manner
- Other ways may be possible, but will require more research



# Summary of File System Analysis

- Can completely recover the in-memory filesystem, its associated metadata, and all file contents
- Ordered, partial recovery of deleted file names and their metadata is also possible
- Traditional forensics techniques can be made possible against live CDs
  - Making such analysis accessible to all investigators



# Implementation

- Recovery code was originally written as loadable kernel modules
  - Allowed for rapid development and testing of ideas
  - 2nd implementation was developed for Volatility
- Vmware workstation snapshots were used to avoid rebooting of the live CD and reinstallation of software
  - TAILS doesn't include development tools/headers
  - This saved days of research time



# Testing

- Output was compared to known data sets
  - Directories and files with scripted contents
  - Metadata was compared to the stat command
  - File contents were compared to scripted contents
- Deleted information was analyzed through previously allocated structures
  - While a file was still allocated, its dentry, inode, etc pointers were saved
  - File was deleted and these addresses were examined for previous data



# Memory Analysis of Tor



# Tor Overview

- Used by millions of people worldwide to perform anonymous Internet communications
- Anonymity of communications is essential to whistleblowers, journalists from nations without freedom of the press, and to a number of other professions
- Any recovery of Tor related activity can have dire consequences for such people



# One Slide Technical Overview

- Tor encrypts and sends traffic from clients to a number of other hosts before being sent to the recipient destination
- Only the final Tor endpoint can decrypt the actual packet contents
  - All others can only decrypt necessary routing information
- The endpoint used is changed at regular intervals to ensure that a compromise does not remove all anonymity



# Tor Analysis Motivation

- Forensics/IR Perspective
  - TAILS and a number of other live CDs use Tor to avoid network forensics
  - Not being able to obtain or reconstruct traffic can make certain investigation scenarios impossible
  - If memory analysis can reveal useful evidence then the inability to perform network analysis is not as painful



# Tor Analysis Motivation

- Privacy Perspective
  - Tor provides an extremely useful platform to perform anonymous communications
  - To ensure that communications are indeed secure, memory analysis needs to be performed on all systems that process unencrypted data



# Analyzing Memory Activity of Tor

- Analysis reveals that Tor does not always securely erase memory after its used
- Sound Familiar?
- Since we have access to the process memory of Tor we should be able to recover data of interest....
  - Papers discussing how to recover userland process memory are referenced in the white paper



# Initial Setup & Analysis

- Privoxy is a Tor-aware HTTP proxy
- Tor was installed along with Privoxy on the test virtual machine
- wget was then configured to use Privoxy which would relay the information to Tor
- Before digging into source code, performed the Poor Man's Test (next slide)



# The Poor Man's Test

1. Used wget to recursively download digitalforensicsolutions.com
2. Verified Tor network connections closed
3. Used memfetch [3] to dump the heap of the tor process
4. Ran strings on heap file
5. `# grep -c digitalforensics strings-output`  
7

Looking good so far....



# Initial Analysis Results

- Analysis revealed that HTTP headers, downloaded page contents, server information, and more were contained in its memory
- It seemed that the last used HTTP header was kept in memory
  - Possibly a single buffer used for this?
  - Numerous instances were found for the other types of data



# Interesting Output from Strings

## 1) HTTP REQUEST

GET /incidence-response.html HTTP/1.0

Referer: http://www.digitalforensicssolutions.com/

User-Agent: Wget/1.12 (linux-gnu)

Accept: \*/\*

Host: [www.digitalforensicssolutions.com](http://www.digitalforensicssolutions.com)

## 2) HTML fragments from downloaded webpage

<h2>Evidence Preservation</h2>

<p>Our evidence preservation methodology provides an exact copy of any digital evidence and ensures that the authenticity and integrity of both the duplicate copy and the original data source is preserved.</p>

<h2>Evidence Custody</h2>



# Digging Deeper into Tor

- After seeing the previous results, source code analysis was performed
- Again, orderly collection of data is our goal
- Much more analysis is possible than what was covered in this initial analysis
- Still on-going research...



# Developed Analysis Scripts

- Two Python scripts were developed that pull information from a Tor process
  - The first enumerates and obtains the Tor freelist
  - The second enumerates Tor cells



# Script 1 - Walking Tor's freelist

- Tor keeps “chunks” in its global *freelist* in order to provide fast allocation of new memory
  - Very similar to the workings of the *kmem\_cache*
  - The script enumerates the freelist array and dumps all memory contained



# Freelist Structure

```
typedef struct chunk_freelist_t {  
    size_t alloc_size; // size of chunk  
    int cur_length; // number on list  
    chunk_t *head;  
}
```

*freelist* is an instance of this structure

```
typedef struct chunk_t {  
    struct chunk_t *next;  
    size_t datalen;  
    char *data;  
} chunk_t;
```

Each chunk is represented by a `chunk_t`



# Script 2- Tor's Cell Pool Cache

- In Tor, all data is sent and received as a packed cell
- *cell\_pool* is a memory pool that holds cells allocated and deallocated by Tor
  - Unless the pool is cleaned
- Walking of this pool enumerates every cell structure including its contents (payload)
- Unfortunately the payloads are encrypted



# Cell Pool Structures & Enumeration

- *cell\_pool* is of type *mp\_pool\_t*
- The recovery script walks the three *mp\_chunk\_t* lists as well as the doubly linked list contained in each *mp\_chunk\_t*
- This leads to the type-agnostic *mem* buffer of each chunk

```
struct mp_pool_t {  
    struct mp_chunk_t *empty_chunks,  
    *used_chunks, *full_chunks;  
    size_t item_alloc_size; }
```

```
struct mp_chunk_t {  
    mp_chunk_t *next;  
    mp_chunk_t *prev;  
    size_t mem_size;  
    char mem[1]; }
```



# Recovery of Packed Cells

- *mp\_chunk\_t* structures hold type-agnostic data
- In the cell pool these are represented by a:  
typedef struct packed\_cell\_t {  
    struct packed\_cell\_t \*next;  
    char body[CELL\_NETWORK\_SIZE];  
} packed\_cell\_t;
- Walking the next list retrieves reachable packed cells



# Conclusion

- Memory Analysis of Live CDs is no longer difficult
- Use of the presented research enables traditional forensics techniques to be used
- As if we didn't know already, applications are really bad about handling of sensitive data in memory



# Future Work – Live CD Filesystems

- Integrate analysis code into Volatility
- Test against more Live CDs / aufs configurations
  - aufs has a number of configuration options
- Look into stackable filesystems used by other Live CDs
  - Unionfs is a good target (used by Debian, Gentoo, etc)



# Future Work - Tor

- Work on recovery of encrypted Tor cells
  - Need to find the encrypted key, match to packed cell, and then decrypt the payload section
- Tor developers are aware of the memory handling issues, response will determine amount of further work possible



# Comments? Questions?

- Full details of work are in our whitepaper
- Contact: [andrew@digdeeply.com](mailto:andrew@digdeeply.com)



# References

- [1] <https://amnesia.boum.org/>
- [2] <http://www.backtrack-linux.org>
- [3] [lcamtuf.coredump.cx/soft/memfetch.tgz](http://lcamtuf.coredump.cx/soft/memfetch.tgz)
- [4] A. Case, *et al*, "Treasure and Tragedy in *kmem\_cache* Mining for Live Forensics Investigation," *Proceedings of the 10th Annual Digital Forensics Research Workshop (DFRWS 2010)*, Portland, OR, 2010.

