How the Leopard hides his Spots

OS X Anti-Forensics

the.grugq@gmail.com

Overview

- Introduction
- Strategies
- Tactics
- Conclusion

Introduction

- the grugq <the.grugq@gmail.com>
- Anti forensic researcher since 1999
- Independent security researcher
- Based in Thailand

Why Anti Forensics?

- Forensics is integral to infosec spectrum
- No research => no fixes
  - Forensics remains vulnerable and insecure
- Still “green field” research

Forensics (in 1 slide)

- Forensics only exists within an investigation
- Preserve data in original state
- Extract evidentiary data from snapshot
- Present evidence

Anti Forensics

Data is evidence

Reduce the quantity and quality of evidentiary data

Strategies

Data Destruction

They can’t find what isn’t there

- Remove evidentiary data *ex post facto*
- Difficult to do properly
  - Systems scatter data everywhere
- Scorched earth is the best policy

Data Hiding

They can't find what they can't see

- Store data outside scope of tools
- Not a long term solution
  - Suffers from bug death
- Don’t forget to encrypt

Data Contraception

They can't find what was never there

- Don’t create evidentiary data
- Avoid contact with the disk
  - Requires planning and discipline
  - see haxh, hacking harness*

* http://www.tacvoip.com/tools.html

Tactics

Data Hiding Tactics

- Requirements:
  - Hidden - obviously
  - Robust - don’t want data to vanish
- Exploit structured storage bugs

Structured Storage

File System Fundamentals

- Comprised of data and metadata
- Pair data streams with readable names
- Internal structured metadata organises:
  - System level metadata/data
  - User level metadata/data
- OS level CRUD

Components

- `Header` - Global FS layout / properties
- `Block` - Smallest atomic component
- `Node` - Metadata for a single file + block list(s)
- `Map` - Link names to nodes

Example: NTFS

- `Header` - Boot Block
- `Block` - Cluster
- `Node` - Master File Table entry (File)
- `Map` - Master File Table Directory (Folder)

Example: FAT

- `Header` - Boot Block
- `Block` - Cluster
- `Node` - Directory entry + FAT chain
- `Map` - Directory file

Attacking Structured Storage

- Allocate space
  - Exploit bugs
    - Parsing
    - Interpretation
    - Presentation
- Inject data into that space

FISTing

- File system Insertion and Subversion Technique
- Generic technique for exploit structure storage
  - Find a hole and FIST it

What holes can I FIST?

FIST sized Holes

- Special files
  - Handled implicitly
- Slack space
  - Typically inaccessible
- `reserved`
  - reserved for hacker use, only!

Forensic Tool Bugs

- Incomplete/Ignorant implementations
  - Unused structured storage “features”
- Logic bugs
  - Edge cases get ignored
- Security bugs
  - buffer overflows, `int` wraps, etc.

FISTing for All

- Any structured data storage can be FISTed
  - File Systems
  - Application file formats

Assaulting OS X

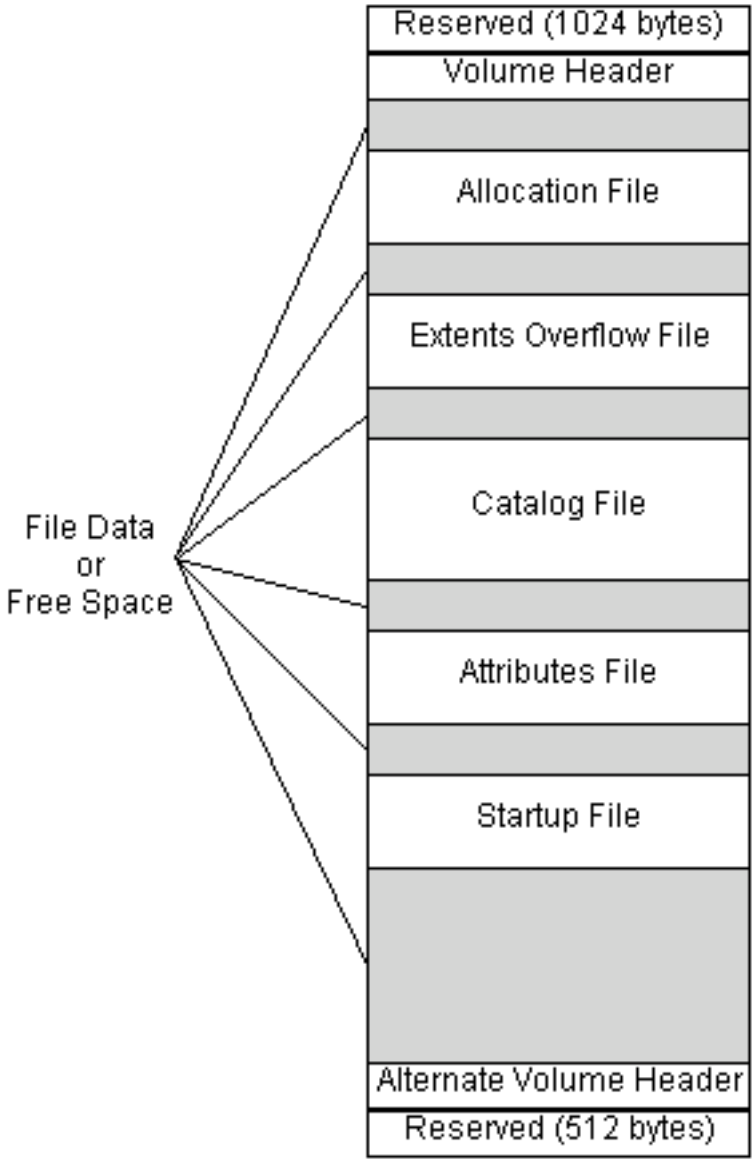
OS X Attacks

- File system attacks
  - HFS+
- Application file format attacks

HFS+ Induction

HFS+

- Hierarchical File System Plus (HFS+)
- Introduced with OS X
- Strongly influenced by HFS
- Complex on-disk structure
  - B*trees
- Technical Note 1150


Reserved (1024 bytes)
Volume Header
Allocation File
Extents Overflow File
Catalog File
Attributes File
Startup File
Alternate Volume Header
Reserved (512 bytes)
File Data
or
Free Space

Components

- `Header` - Volume Header
- `Block` - Block
- `Node` -
  - block lists in extents
  - meta data in catalog file entries
- `Map` - catalog file entries

HFS+ Core Concepts

- Data Forks
  - Extents
- B*trees
  - Nodes

Data Forks

- Store data stream location information
  - Size of user data
  - Block location + order

Data Forks

```
struct HFSPlusForkData {
    UInt64                  logicalSize;
    UInt32                  clumpSize;
    UInt32                  totalBlocks;
    HFSPlusExtentDescriptor extents[8];
};

struct HFSPlusExtentDescriptor {
    UInt32                  startBlock;
    UInt32                  blockCount;
};
```

Special Files

- `allocationFile` - block allocation bitmap
- `catalogFile` - file/directory meta data
- `extentsFile` - fragmented file block lists
- `startupFile` - (optional) kernel loader
- `attributesFile` - extended attributes

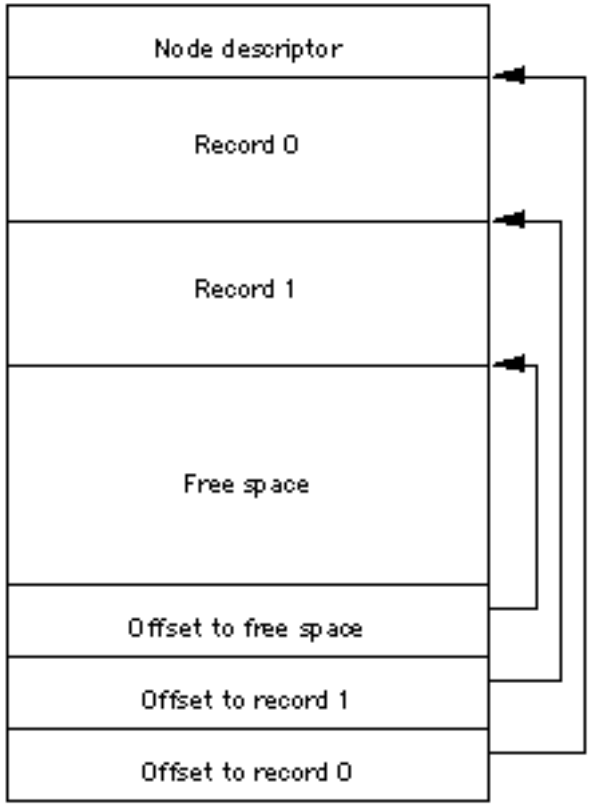
B*trees

- Used for most file system meta data
- Binary tree
- Stored on disk within a “special file”
- File is divided into equal sized nodes
  - Node address == index

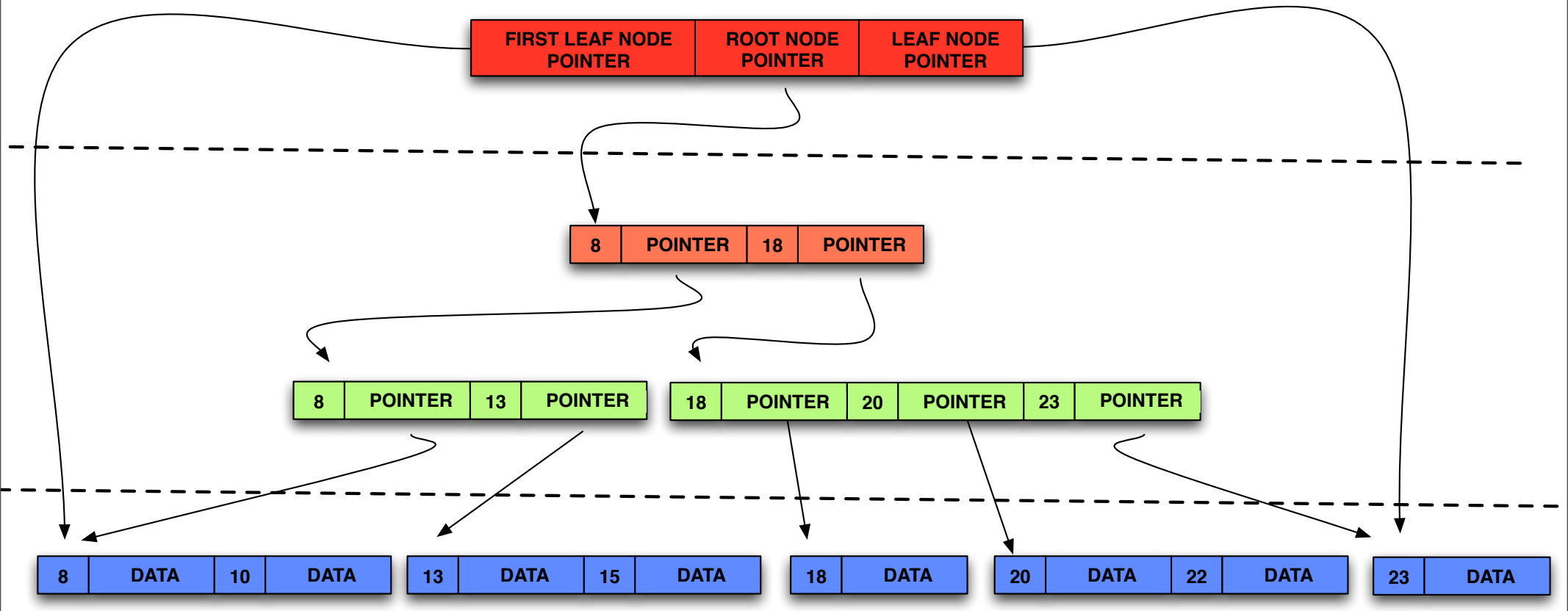
B*tree Nodes

- `Header` node - B*tree metadata
- `Map` nodes - node allocation bitmap
- `Index` nodes - key:node pointer
- `Leaf` nodes - key:data record

B*tree Node


Node descriptor
Record 0
Record 1
Free space
Offset to free space
Offset to record 1
Offset to record 0

B*tree


FIRST LEAF NODE POINTER
ROOT NODE POINTER
LEAF NODE POINTER
8
POINTER
18
POINTER
8
POINTER
13
POINTER
18
POINTER
20
POINTER
23
POINTER
8
DATA
10
DATA
13
DATA
15
DATA
18
DATA
20
DATA
22
DATA
23
DATA

HFS+ Attacks

File Allocation Attacks

Bad Blocks File

- Allocated extents within the `extentsFile`
- Special CNID => 5
- Prevents kernel allocating bad blocks for userfiles
- Lame attack, way too obvioous

`startupFile`

- Used for certain (archaic) systems
  - Typically 0 length file
- Can be arbitrary size
- Kinda like bad blocks file, only more explicit

HFS Wrapper

- HFS+ volumes embedded within HFS
- HFS volume marks HFS+ space in “bad blocks file”
- Slack space after embedded HFS+ volume

B*tree Internals Attacks

Excessive Map Nodes

- Map nodes contain node allocation bitmap
- Stored as linked list of nodes
- Can exceed the size required for all nodes

B*tree Free Space

- B*tree nodes can contain free space
- Use freespace for data storage

When Unicode Attacks

Zero Width Unicode

- UTF-16 has non-glyph characters
  - `\x00\x00 NULL CHARACTER`
  - `\x20\0B ZERO WIDTH SPACE`
- Invisible data storage

UTF-16 is where?

- iPod iTunes db (mhbd), extensive UTF16
- File names: HFS+, NTFS, FAT32
- Inside documents
- … etc.

ZWU

- (Z triple U) iPod mhbd attack
- Max size is 255 UTF-16 chars
- 8 chars per byte
- ~31 bytes per string (theoretical max)
- approx 100k per 1k tracks

```
>>> import utfool
>>> s = "hello world"
>>> u = utfool.tounicode(s)
>>> print u'"%s"' % u
""
>>> len(u)
88
>>> s == utfool.fromunicode(u)
True
```

Application File Formats

Browser Cookies

Browser Cookies

- What is a cookie? url: { key: value }
  - value is a base64 encoded encrypted binary blob

Browser Cookies

- What is a cookie? url: { key: value }
  - value is a base64 encoded encrypted binary blob
- What do we want to store?
  - encrypted binary blobs…

Mozilla Cookies File

- Stored in an SQLite database
  - Can access with `sqlite3` tool
- Very simple schema

moz_cookies Schema

```
CREATE TABLE moz_cookies (
            id INTEGER PRIMARY KEY,
            name TEXT,
            value TEXT,
            host TEXT,
            path TEXT,
            expiry INTEGER,
            isSecure INTEGER,
            isHttpOnly INTEGER,
            lastAccessed INTEGER
            )
```

Cookies HOWTO

```
sqlite> insert into moz_cookies host,value
('www.lolitapictures.com',
'eW91J3JlIGxlZXQ=');
```

SQLite

File Format

- File is an array of pages
  - Header stored in first page
  - Free pages in a linked list
- Pages contain cells
- Cells linked in b-trees

SQLeez

- Free space within SQLite files
  - Expand in size, shrink after vacuum
- AUTO-VACUUM can destroy

SQLeez

- Demo

Future

- Out of the file system, into the files
- Application specific attacks are harder to detect
  - More diverse attack space

Q & A

Thank you.