

Forensic Discovery of hosts and networks

Wietse Venema

IBM T.J.Watson Research

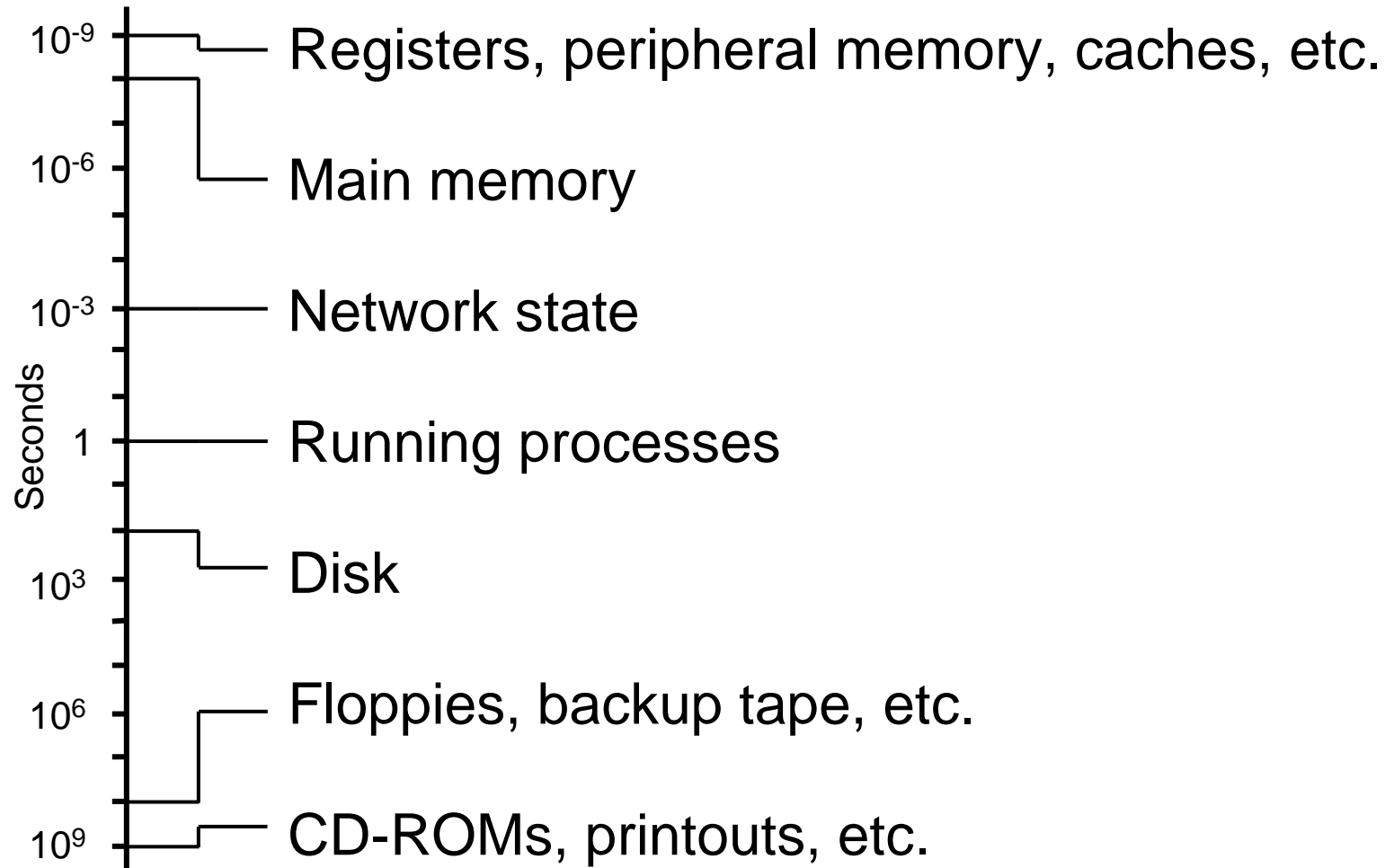
Hawthorne, New York, USA

Overview

- Basic concepts.
- Time from file systems and less conventional sources.
- Persistence of deleted data on disk and in main memory.
- Recovering WinXP/Linux encrypted files without key.
- Recovering file encryption keys from memory.
- Book text and software at author websites:
 - <http://www.porcupine.org/>
 - <http://www.fish2.com/>

Order of Volatility

(from nanoseconds to tens of years)



Most files are accessed rarely

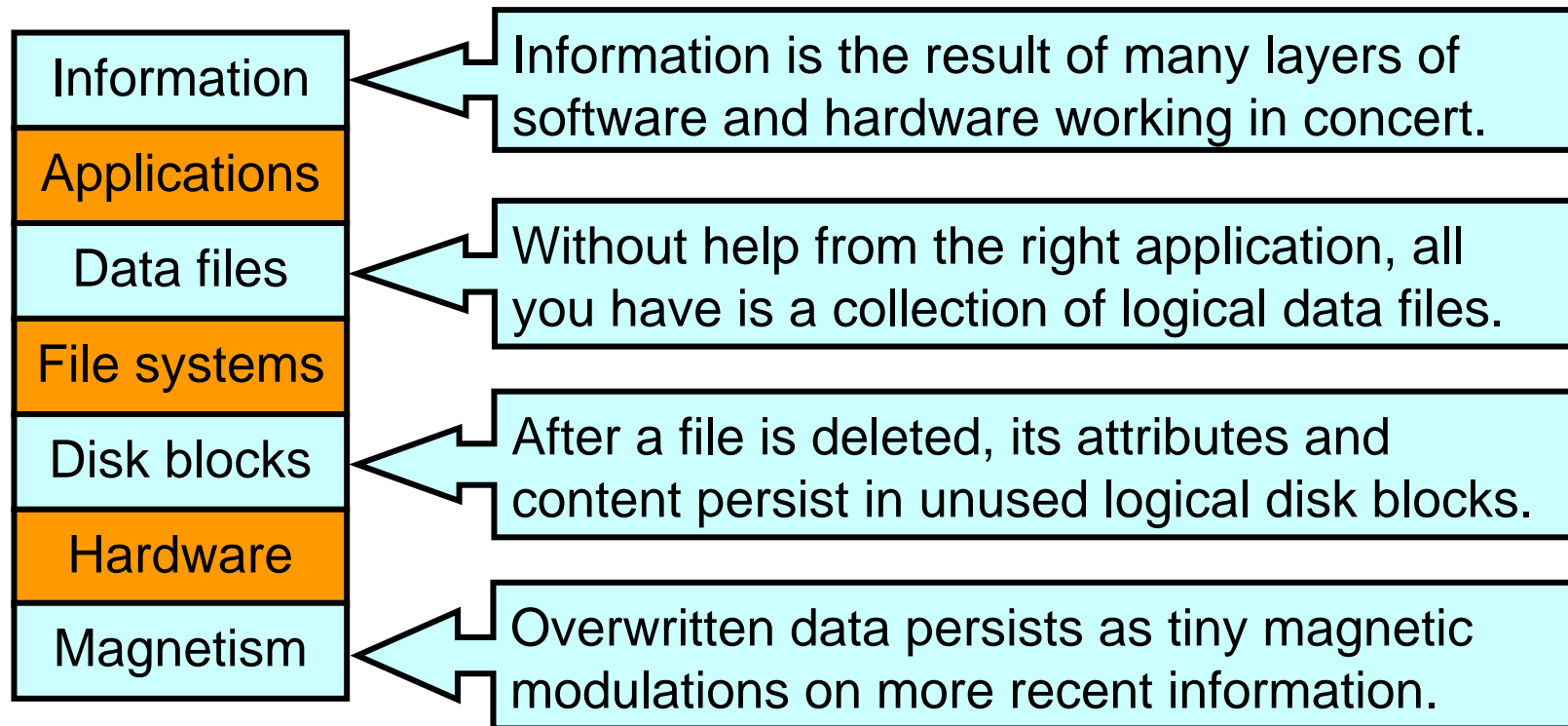
	<i>www.things.org</i>	<i>www.fish2.com</i>	<i>news.earthlink.net</i>
less than 1 day	3 %	2 %	2 %
1 day – 1 month	4 %	3 %	7 %
1 - 6 months	9 %	1 %	72 %
6 months – 1 year	8 %	19 %	7 %
more than 1 year	77 %	75 %	11 %

Numbers are based on file read access times.

Erosion paradox

- Information disappears, even if you do nothing.
Examples: logfiles and last file access times.
- Routine user or system activity affects the same files again and again - literally stepping on its own footprints.
- Footprints from unusual behavior stand out, and for a relatively long time.

Fossilization and abstraction layers



- Information deleted at layer N persists at layers $N-1$, etc.
- It becomes frozen in time; older data sits in lower layers.

Cost of an investigation

(not entirely serious)

<i>Effort</i>	<i>Skill level</i>	<i>Time</i>
Do nothing	None	Almost none
Minimal	Install system s/w	< 1 Day
Recommended	Junior sysadmin	1-2 Days
Serious	Senior sysadmin	Days – weeks
Fanatical	Expert sysadmin	Months

MACtimes Introduction

What are MACtimes?

(name coined by Dan Farmer)

- **Mtime** Time of last modification
(Write/truncate, create/delete directory entry).
- **Atime** Time of last access
(Read/execute file, look up directory entry).
- **Ctime** Time of last attribute change
(Owner, permission, refcount, size, etc.).
- **dtime** Time of file deletion (LINUX).

Getting MACtimes

```
($dev,$inode,$mode,$nlink,$uid,$gid,$rdev,$size,  
$atime,$mtime,$ctime,$blksz,$blks) = lstat($file);
```

- *lstat()* returns file attributes (above example is in Perl).
- Works in *UFS*, *Ext*fs*, *NTFS*, etc. (even *FAT*).
- TCT¹ Command: “*grave-robber -m*” or “*mactime -d*”.

¹The Coroner’s toolkit, see references at end

Example – login session (what the user sees)

```
$ telnet sunos.fish2.com
Trying 216.240.49.177...
Connected to sunos.fish2.com.
Escape character is '^]'.

SunOS UNIX (sunos)

login: zen
Password:
Last login: Thu Dec 25 09:30:21 from flying.fish2.com

Welcome to ancient history!
$
```

Question: Why does this example use a 15 year old computer?

Example – login session (MACtime view)

Time	Size	MAC	Permission	Owner	Group	File name
19:47:04	49152	.a.	-rwsr-xr-x	root	staff	/usr/bin/login
	32768	.a.	-rwxr-xr-x	root	staff	/usr/etc/in.telnetd
19:47:08	272	.a.	-rw-r--r--	root	staff	/etc/group
	108	.a.	-r--r--r--	root	staff	/etc/motd
	8234	.a.	-rw-r--r--	root	staff	/etc/ttytab
	3636	m.c	-rw-rw-rw-	root	staff	/etc/utmp
	28056	m.c	-rw-r--r--	root	staff	/var/adm/lastlog
	1250496	m.c	-rw-r--r--	root	staff	/var/adm/wtmp
19:47:09	1041	.a.	-rw-r--r--	root	staff	/etc/passwd
19:47:10	147456	.a.	-rwxr-xr-x	root	staff	/bin/sh

(m=modified, a=read/execute, c=status change)

Uses for MACtimes

- Profiling user activity (activity footprint).
- Understanding systems (execution footprint).
- Improving system security (used/unused files).
- Dead or alive (deleted/existing file attributes).

MACtime Limitations

- Shows only the *last time* something happened.
- Easy to forge: UNIX `utime()`, Windows `SetFileTime()`.
- Digital Alzheimer's. Data erodes over time.
- Only unusual behavior persists.

Analyzing time from hosts and networks

Real intrusion, location and time
details anonymized

Discovery of an intrusion

- A host compromise was discovered on August 20.
 - An “extra” ssh (encrypted login) network service had been installed without proper authorization.
 - This service accepted connections on TCP port 33332, instead of the normal ssh service on TCP port 22.
- The system administrators backed up the system, and then alerted the security staff.
 - The backup destroyed all the file read/execute access times.
- The security staff quarantined the machine and examined the disk with the Coroner’s Toolkit.

MACtime post-intrusion analysis

- Apparently, the rogue ssh service was installed July 19.

Jul 19					
Time	Size	MAC	Permissions	Owner	File name
16:29:47	655360	m..	-rw-r--r--	root	/dev/ssh/sshdlinux.tar
16:30:13	655360	..c	-rw-r--r--	root	/dev/ssh/sshdlinux.tar
16:30:16	395	..c	-rwxrw-r--	5002	/dev/ssh/ssh.sh
	880	..c	-rw-r--r--	5002	/dev/ssh/ssh_config
	537	..c	-rw-----	5002	/dev/ssh/ssh_host_key
	341	..c	-rw-r--r--	5002	/dev/ssh/ssh_host_key.pub
16:30:17	695	..c	-rw-r--r--	5002	/dev/ssh/sshd_config
16:30:20	1024	m.c	drwxr-xr-x	root	/dev/ssh

m=modified, a=read/execute, c=status change

Jul 19
Install ssh

Aug 20
Discovery

Network flows to the rescue

- Fortunately, the security staff routinely collected network flow information for incident response.
- They used Argus to summarize an entire connection as one network connection record:
 - Start/end time, protocol type, source/destination host+port.
 - Packet count and byte count (not used in our examples).
 - Connection states (from initial handshake to disconnect).
 - No network connection content.

Jul 19
Install ssh

Aug 20
Discovery



Network flow analysis

- Step 1: zoom in on victim-related network activity when the rogue ssh service was installed (July 19, 16:30).

start	end	proto	source	destination	status
16:25:53-16:30:26		tcp	suspect1.44445	victim.21	sSEfR
16:28:34-16:29:36		tcp	victim.1466	suspect2.21	sSEfC
16:29:30-16:29:36		tcp	suspect2.20	victim.1467	sSEfC
16:30:47-16:47:16		tcp	suspect1.1023	victim.33332	sSEfC

- Step 2: determine the purpose of each connection.
- Step 3: zoom out and look for other network flows with similar properties (same host, etc.).

Jul 19
Install ssh

Aug 20
Discovery



Episode 1: rogue ssh service

- Purpose of each network connection:
 - Intruder enters via the FTP service (port 21). See next slide.
 - Intruder downloads the sshlinux file with FTP (ports 21, 20).
 - First connection to the rogue ssh service (port 33332).

start	end	proto	source	destination	status
16:25:53-16:30:26		tcp	suspect1.44445	victim.21	sSEfR
16:28:34-16:29:36		tcp	victim.1466	suspect2.21	sSEfC
16:29:30-16:29:36		tcp	suspect2.20	victim.1467	sSEfC
16:30:47-16:47:16		tcp	suspect1.1023	victim.33332	sSEfC

- More connections to the rogue ssh service until the machine was quarantined.



Episode 2: prior activity

- Earlier *suspect1* activity suggests the existence of a previously-installed trapdoor.
 - Open the door by connecting *from* TCP port 44445.

start	end	proto	source	destination	status
<i>...lots of of suspect1.44445 records omitted for brevity...</i>					
16:25:32		tcp	suspect1.44445	other.110	s
16:25:49		tcp	suspect1.44445	victim.110	sR
16:25:53-16:30:26		tcp	suspect1.44445	victim.21	sSEfR

- Activity from TCP port 44445 spans almost one year, beginning on August 22.



Initial compromise, 1 year earlier

- Apparently, the initial compromise happened August 21.
 - Gain entrance via a DNS server vulnerability (TCP port 53).
 - Download two malware files with FTP (TCP port 21, 20).
 - First connection from suspect port 44445 to victim port 21

start	end	proto	source	destination	status
23:59:55	00:29:48	tcp	suspect3.1882	victim.53	sSEfR
00:08:32	00:09:04	tcp	victim.1027	suspect4.21	sSEfC
00:08:42	00:09:04	tcp	suspect4.20	victim.1028	sSEfC
00:11:08	00:13:26	tcp	victim.1029	suspect4.21	sSEfC
00:12:07	00:12:13	tcp	suspect4.20	victim.1030	sSEfC
00:13:38	00:13:35	tcp	suspect4.44445	victim.21	sSEfR



Host and network forensics

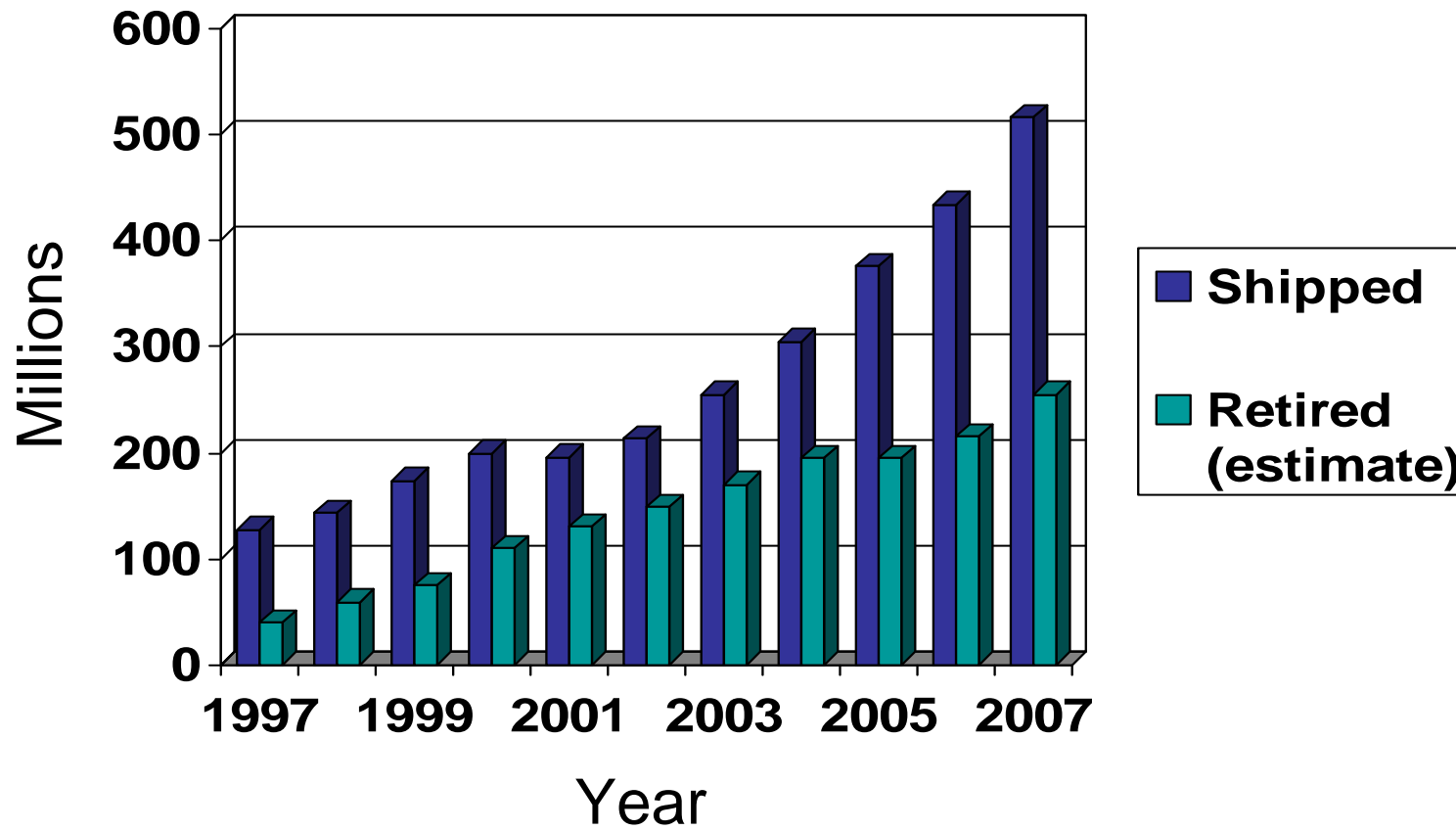
- Host forensics gives a *close-up* view, but information is lost over time, and intruders may hide or remove traces.
- Network forensics gives an *independent* view, but the amount of information you can keep long-term is limited.
- Combinations of host, netflow and IDS form the current state of the art in botnet detection.



Intermezzo 1: Information on retired disks

Global hard disk market

(Millions shipped, source: Dataquest, iSuppli)



Long-term collection of retired disks

- Experiment: buy used drives, mainly via Ebay.
- Time frame: 1998 - 2006.
- 1005 Drives purchased.
- 750 Drives included in “corpus” for research.
- 449 Drives contained active file system.
- 324 Drives had more than 5 files.
- 882 GB of file content was recovered.

Simson Garfinkel, DFRWS 2007

<http://www.dfrws.org/2007/proceedings/p2-garfinkel.pdf>

Results from early informal survey (Garfinkel & Shelat)

- Time frame: November 2000 - August 2002.
- 158 Drives purchased.
- 129 Drives still worked.
- 51 Drives “formatted”, leaving most data intact.
- 12 Drives overwritten with fill pattern.
- 75 GB of file content was found or recovered.

IEEE Privacy & Security January/February 2003, <http://www.simson.net/clips/academic/2003.IEEE.DiskDriveForensics.pdf>

What information can be found on a retired disk

- One drive with 2868 account numbers, access dates, balances, ATM software, but no DES key.
- One drive with 3722 credit card numbers.
- Corporate memoranda about personnel issues.
- Letter to doctor from cancer patient's parent.
- Email (17 drives with more than 100 messages).
- 675 MS Word documents.
- 566 MS Powerpoint presentations.
- 274 MS Excel spreadsheets.

Persistence of deleted information

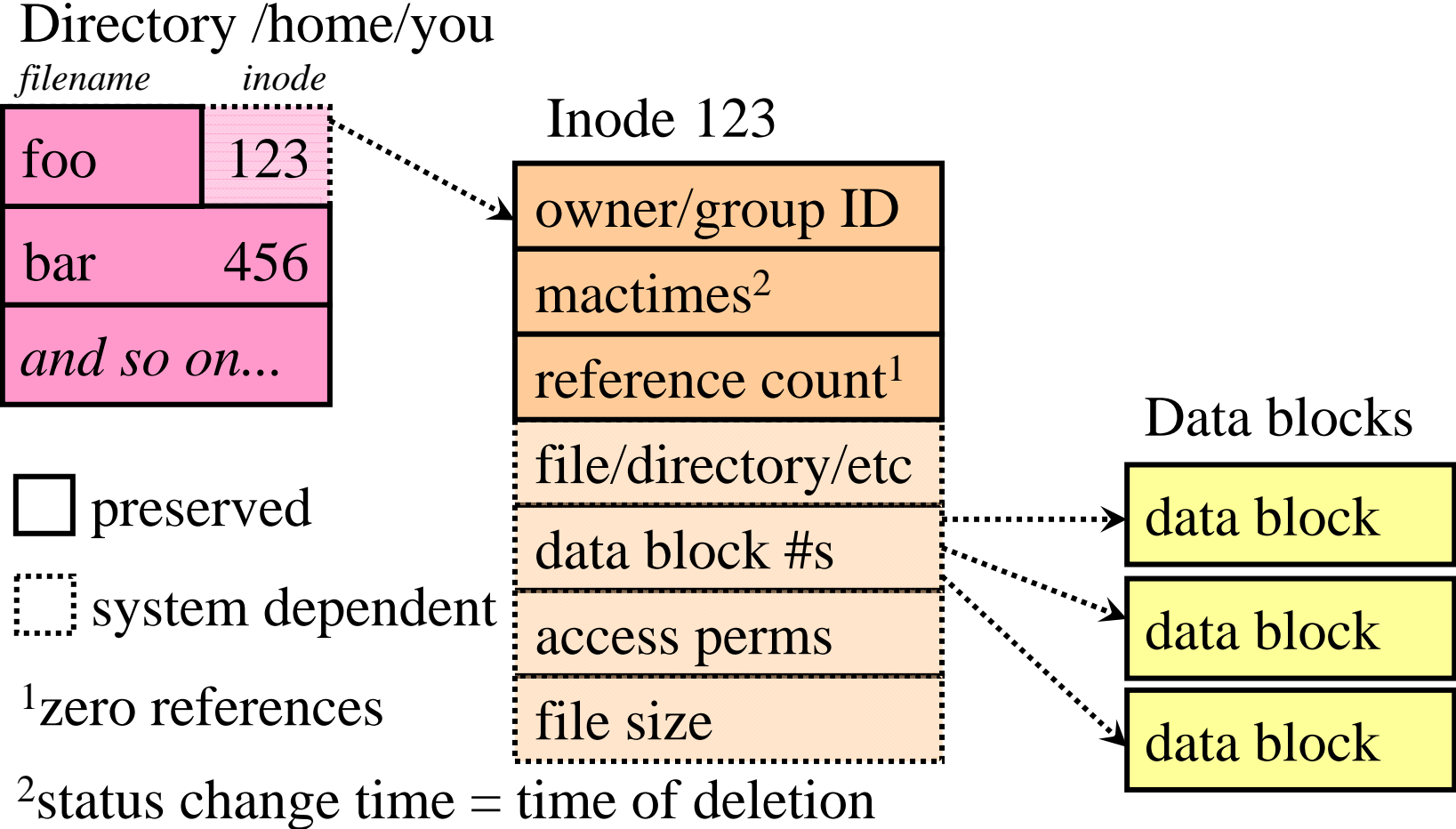
Overview

- Persistence of deleted file information.
- Persistence of information in main memory.
- Recovering encrypted Linux/WinXP files without key.

Persistence of deleted file information

Why deleted file data can be more persistent than existing file data

Deleting a file destroys structure but not content



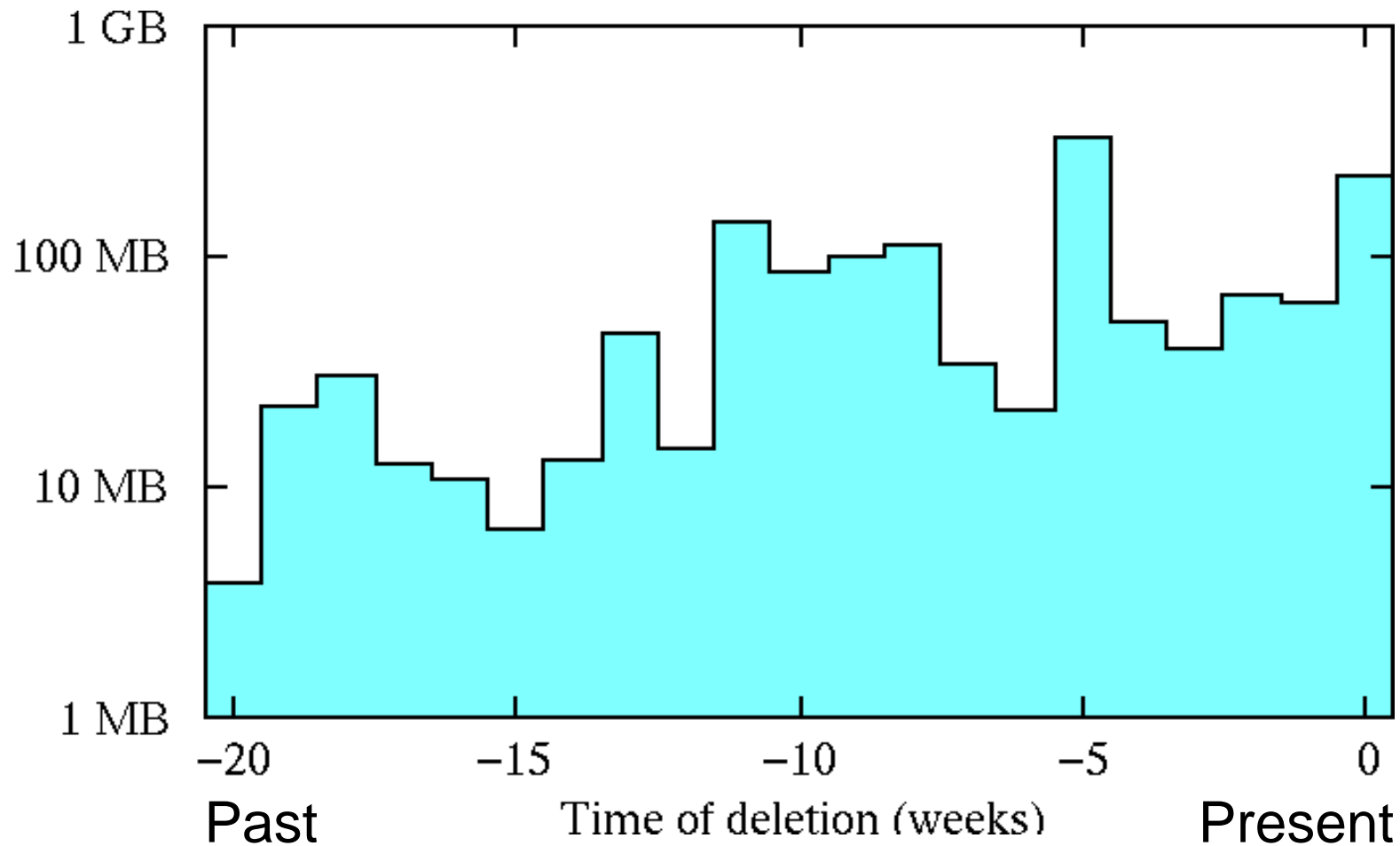
Measurements with deleted file content (not so easy)

- Problem: deleted file content rarely gives hints about the time of deletion (exception: short-lived logfiles).
- We actually have to measure when a file is deleted, and how long it takes before its content is overwritten.
- Experiment with half-dozen file systems: every night, hash every disk block¹ and record its status (*allocated*, *free*, *metadata*)². Keep doing this for several months. Result: 100 MBytes of data.

¹We kept only 16 bits of each hash, to save space.

²Data collection tools are at the book website.

Persistence of deleted file *content* (dedicated UNIX server, 9GB half full)



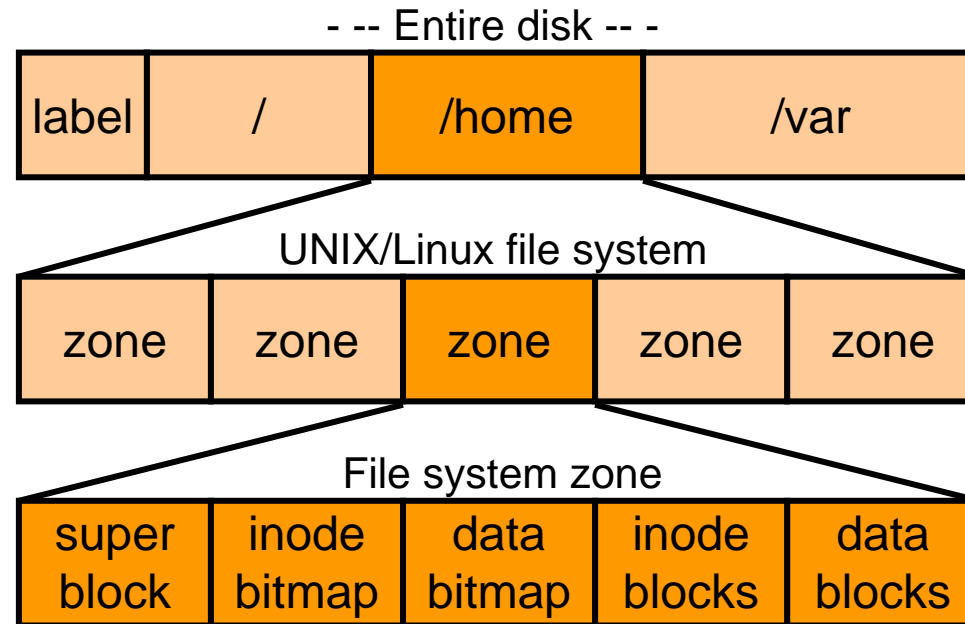
Summary: persistence of deleted file content

Machine	File system	Half-life
spike.porcupine.org ¹	entire disk	35 days
flying.fish.com ²	/	17 days
flying.fish.com ²	/usr	19 days
www.porcupine.org ¹	entire disk	12 days

¹FreeBSD ²Linux

Why deleted file data can be more persistent than existing file data

- Existing files are easy to access, and easy to modify. Deleted files are less accessible, and become fossils.



- File system locality protects deleted data. Example: a deleted file in zone X survives write activity in zone Y.

Main Memory Persistence

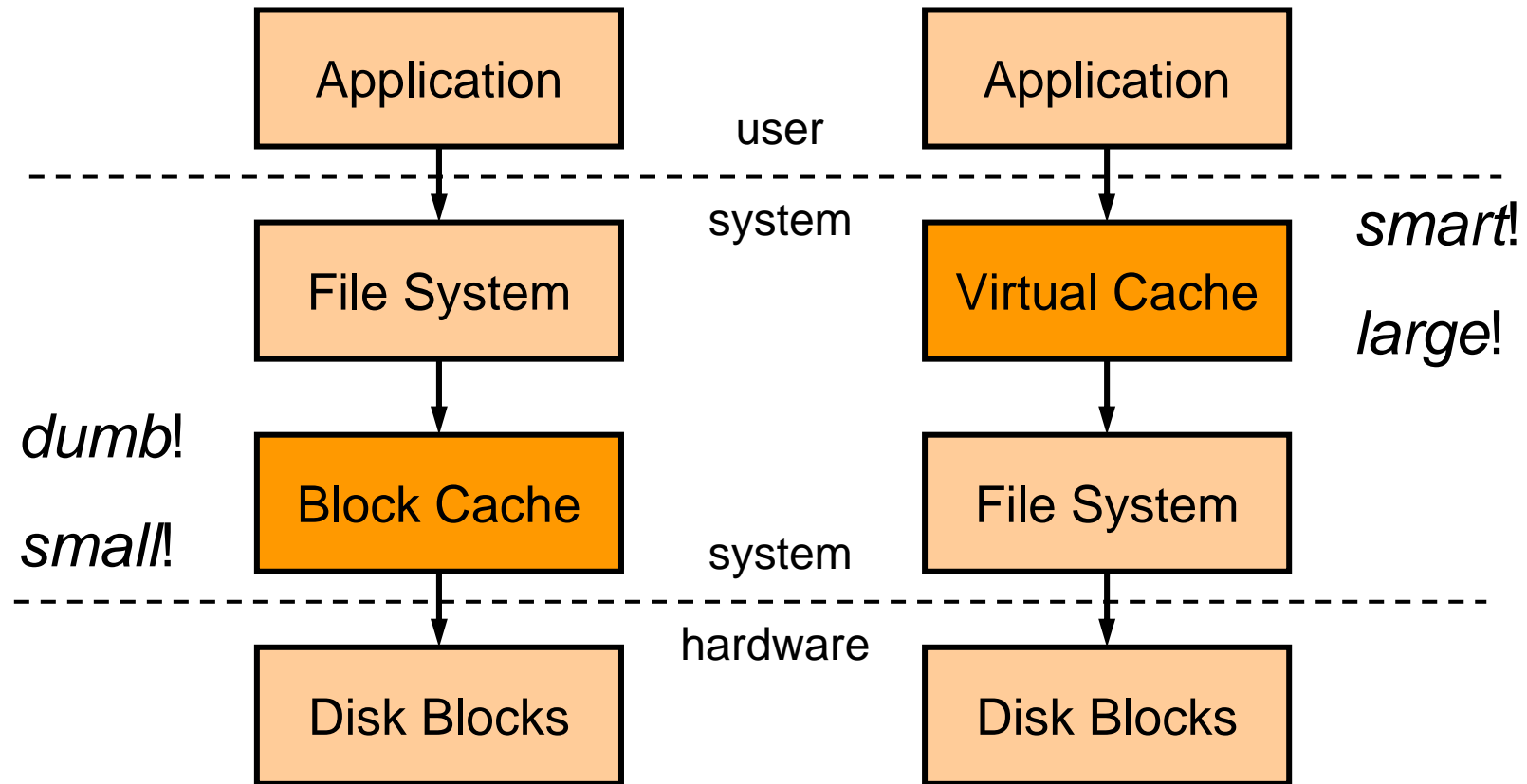
Recovering Linux/WinXP
encrypted files without the key

Information in main memory

- Running processes¹.
- Terminated processes¹.
- Kernel memory.
- Recently active files/directories (file cache).
- Deleted files (from processes¹ or from file cache).
- Different persistence properties.

¹Some information may be found in swap files.

Old block cache \Leftrightarrow Virtual cache (owned by system, not by applications)



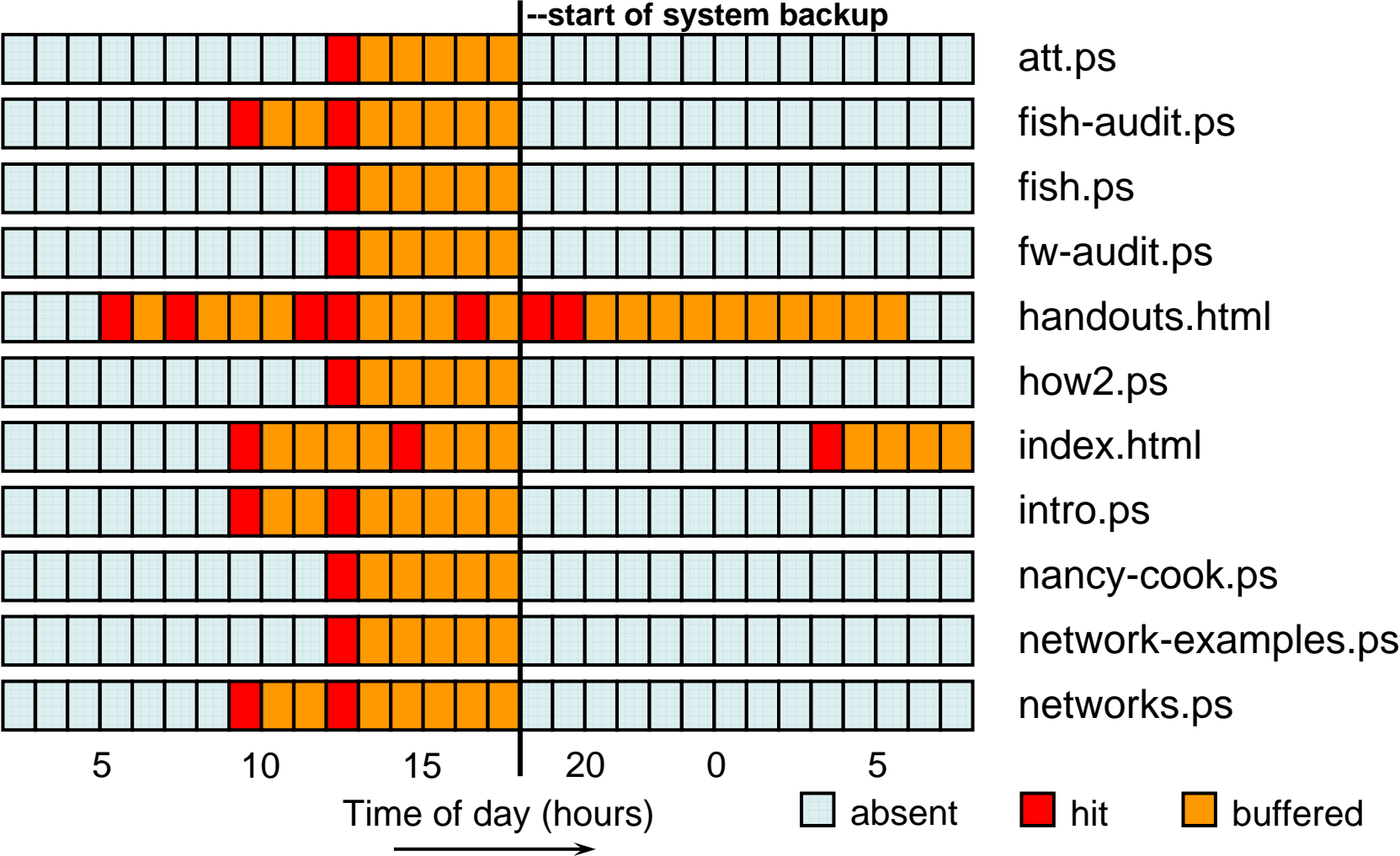
DOS, Win95/98/ME, old BSD BSD, Linux, Solaris, WinNT/2K/XP/etc

File caching in main memory measurements

- Disk blocks are cached in memory in page size chunks.
- Every hour, compute a hash of every 1kbyte memory block¹.
- Once: compute hashes of 1kbyte file blocks, zero-padding short blocks.
- Off-line analysis: compare memory hashes with file hashes to get an idea of what is cached in memory.
- Minor impact from collisions, such as all-zero blocks.

¹Tools are at the book website.

File caching in main memory (low-traffic web pages, FreeBSD)



Recovering WinXP/Linux encrypted files without key

Two experiments with remarkably
similar outcomes

Experiment 1: Windows/2K/XP EFS

- *EFS*¹ provides encryption by *file* or by *directory*. Encryption is enabled via an Explorer property dialog box or via the equivalent system calls.
- With encryption by directory, files are encrypted *before* they are written to disk.
- Is unencrypted content of *EFS* files cached in main memory?
- If yes, for how long?

¹EFS=Encrypting File System

Experiment 1: create encrypted file

- Create “encrypted” directory *c:\templencrypted*.
- Download 350kB text file via FTP, with content:

```
00001 this is the plain text
00002 this is the plain text
...
11935 this is the plain text
11936 this is the plain text
```

- Scanning the disk (from outside the VMware virtual machine) confirms that no plaintext is written to disk.

Experiment 1: search memory dump

- Log off from the Windows/XP console and press Ctrl/ScrollLock twice for memory dump¹ (160 MB).
- Analyze dumped memory with standard UNIX tools.
- 99.6% of the plain text was found undamaged.

¹Microsoft KB 254649: Windows 2000 memory dump options.

Experiment 2: Linux eCryptfs

- *eCryptfs*¹ provides encryption by *file system*.
- Standard with kernel version 2.6.19 and later.
- Files are encrypted *before* they are written to disk.
- Is unencrypted content of *eCryptfs* files cached in main memory?
- If yes, for how long?

¹<http://ecryptfs.sourceforge.net/>

Experiment 2: create encrypted file

(tested with kernel 2.6.15)

- Mount *eCryptfs* file system on */mnt*.
- Run a simple script that generates 29 MB of easy to recognize text:

```
00000 This is the plain text
00001 This is the plain text
...
99998 This is the plain text
99999 This is the plain text
```

- Scanning the disk (from outside the VMware virtual machine) confirms that no plaintext is written to disk.

Experiment 2: search memory dump

- Unmount the file system and dump the VMware guest¹ memory (256 MB) with the *pcat* command from TCT.
- Analyze dumped memory with standard UNIX tools.
- 99% Of the plaintext was found undamaged. One hour later, 96% of the plaintext still persisted.

¹Kernel 2.6 /dev/mem and /proc/kcore appear to be crippled. SBO?

Conclusion - recovering encrypted files without key

- Good: file system encryption provides privacy by encrypting file content before it is written to disk.
- Bad: unencrypted content stays cached in main memory even after the user has logged off (WinXP) or after the file system is unmounted (Linux).
- Similar experiments are needed for other encrypting file systems, but we expect to find similar plaintext caching behavior.

Intermezzo: Recovering keys with “cold boot” attacks

J. Alex Halderman et al, USENIX
Security 2008



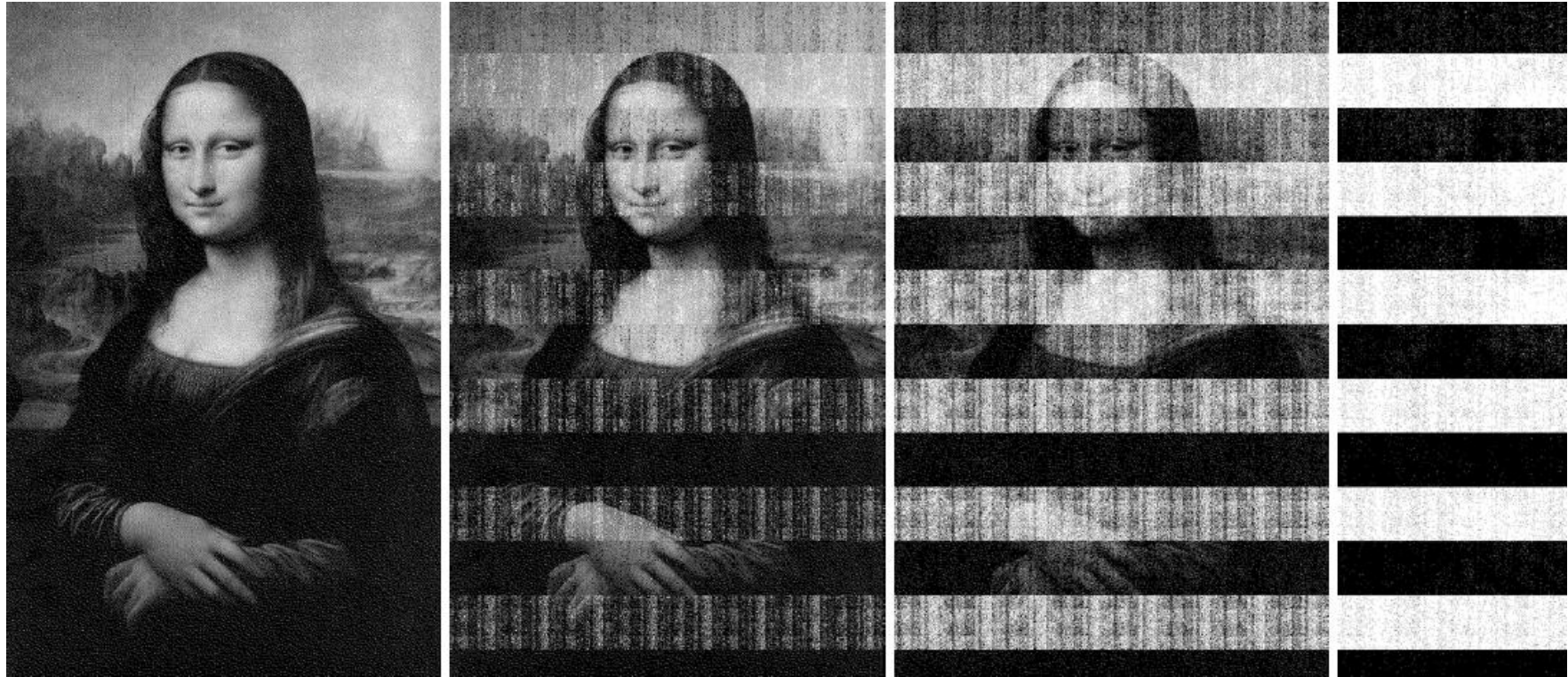
Motivation for cold boot attacks

- Problem: encrypted file systems require user input to unlock their keys¹.
 - Examples: BitLocker, FileVault, TrueCrypt, PGPDisk, etc.
- Solution: extract the unlocked keys from the memory of the running machine.
- Different techniques, different degrees of accuracy:
 - Shutdown the running OS, press RESET, or toggle power briefly. Then boot memorydump software from USB or network.
 - This loses some or all information depending on BIOS etc.
 - **Remove memory modules and examine on different system.**

¹BitLocker “basic TPM mode” requires no user input to unlock keys.

Data loss after power loss

(best-case example with the least loss)



Examples of data loss at -50°C (inverted spray can technique)

Vendor	Year	Memory	Time	Error %
Dell	1999	SDRAM	300s	0.000095
Toshiba	2001	DDR	600s	0.000036
Dell	2003	DDR	360s	0.00144
IBM	2006	DDR2	80s	0.18 ¹

¹This is still less than 0.5 bit error per 256-bit key.

Repairing cryptographic keys

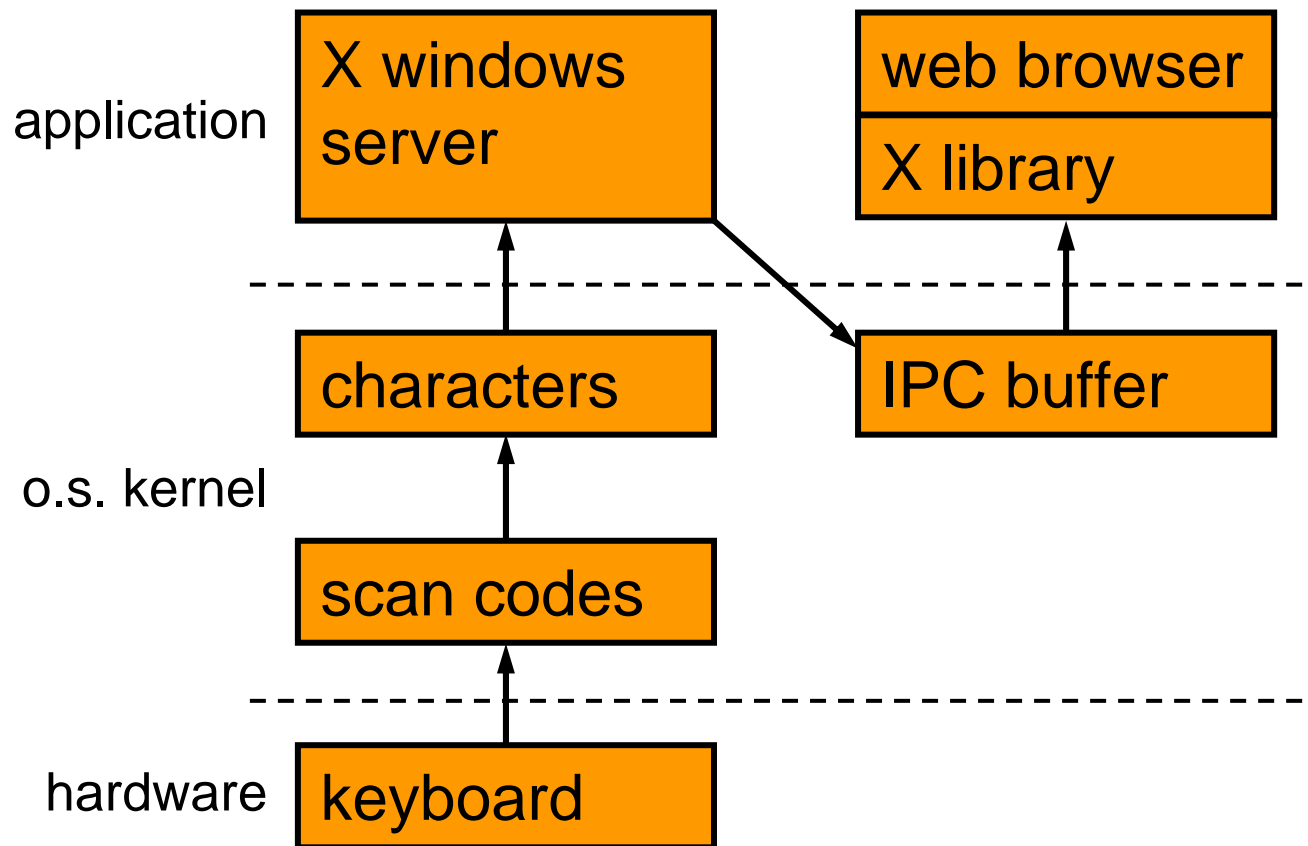
- To improve performance, encryption algorithms keep intermediate results (the so-called “key schedules”) in main memory.
- This information helps “cold boot” attackers to repair keys even when 10% or more of their bits are damaged.
- Examples: DES, AES, RSA.

Long-term memory persistence

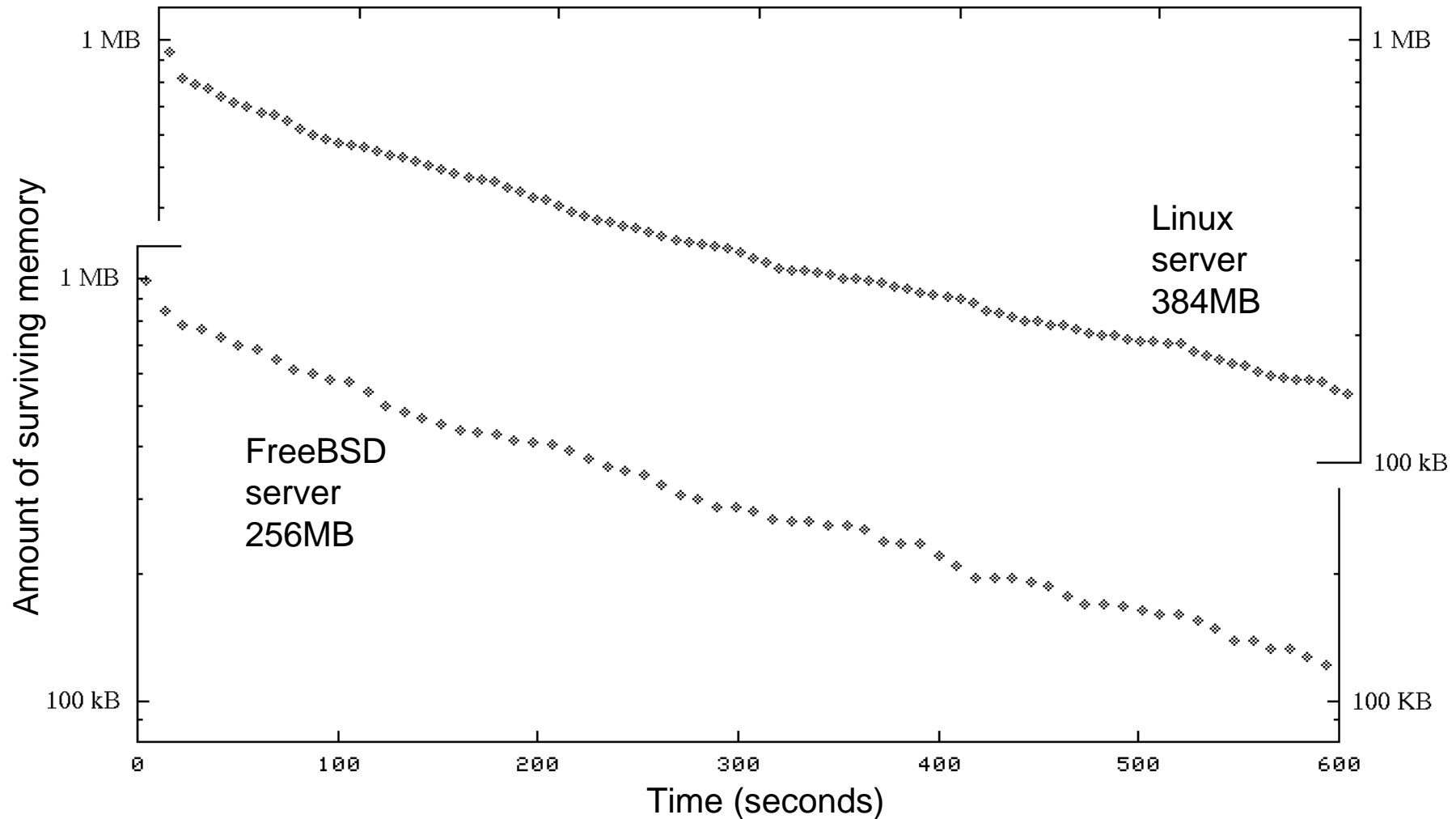
Relevant for “always on”
machines.

Trail of secrets across memory

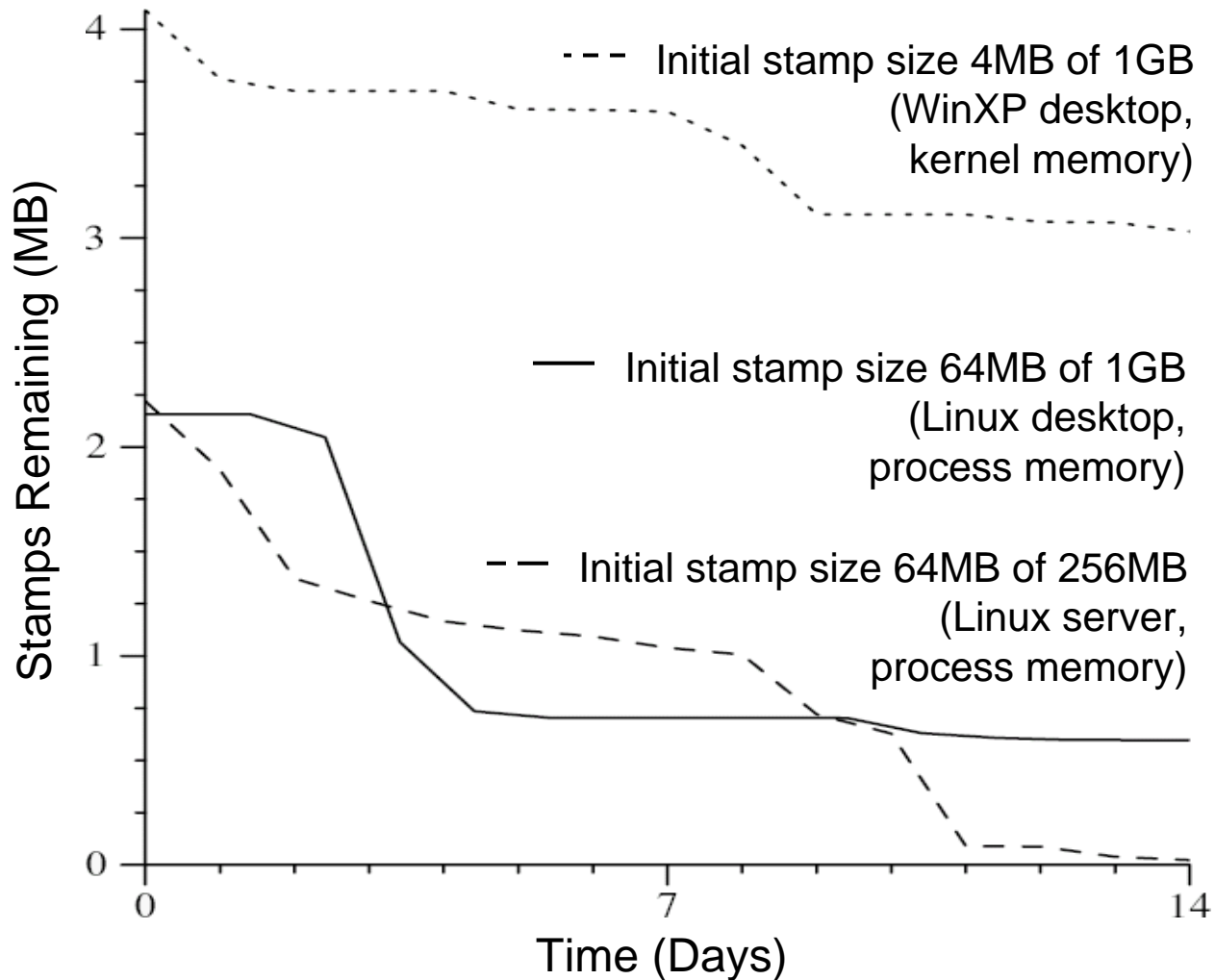
(Chow *et al.*, USENIX Security 2004)



Short-term memory persistence after process termination (1MB stamp)



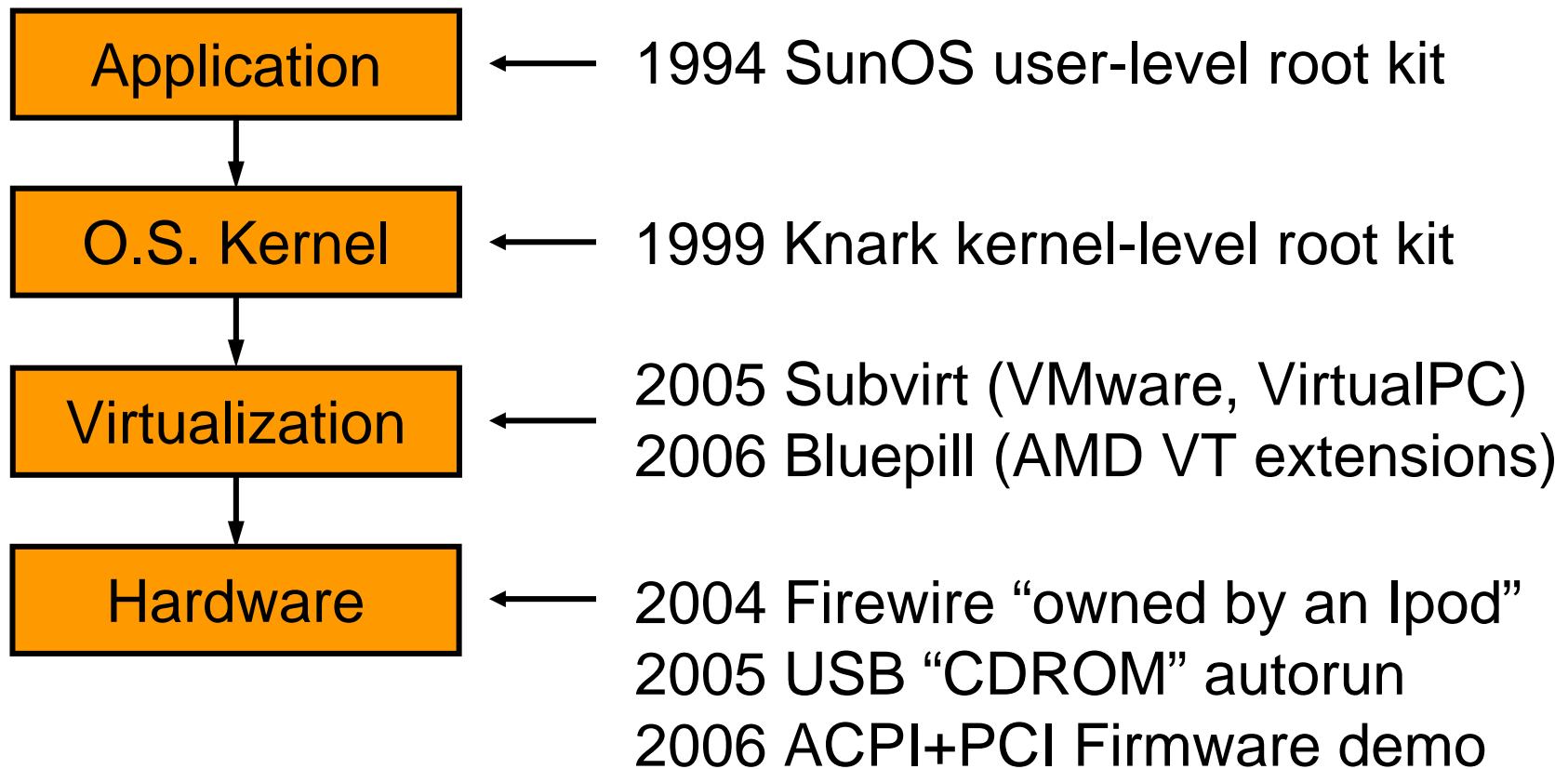
Long-term memory persistence (Chow *et al.*, USENIX Security 2005)



Concluding remarks

Trustworthiness of information

Progression in subversion



Conclusion

- Deleted file information can survive for a year or more, even with actively used file systems.
- Main memory becomes a *primary* source of forensic information, especially with infection of *running* processes or *running* operating system kernels.
- Hardware is becoming softer¹ all the time, as complexity of systems increases and product life cycles shorten.
- Protect your wired and wireless ports.

¹Field upgradable firmware.

Pointers

- Dan Farmer, Wietse Venema: “*Forensic Discovery*”, Addison-Wesley, Dec. 2004; “The Coroner’s Toolkit”, <http://www.porcupine.org/>, <http://www.fish2.com/>
- Jim Chow *et al.*: “*Shredding Your Garbage*”, USENIX Security 2005; “*Understanding Data Lifetime via Whole System Simulation*”, USENIX Security 2004.
- Cold boot website. <http://citp.princeton.edu/memory/>
- Brian Carrier, Sleuthkit. <http://www.sleuthkit.org/>
- Michael Cohen, PyFlag. <http://www.pyflag.net/>
- Digital forensics research website. <http://www.dfrws.org/>