



# Systems and Internet Infrastructure Security

Network and Security Research Center  
Department of Computer Science and Engineering  
Pennsylvania State University, University Park PA

# Shredding Your Garbage: Reducing Data Lifetime Through Secure Deallocation

Jim Chow, Ben Pfaff, Tal Garfinkel, Mendel Rosenblum

Presented by Kevin Butler  
5 April 2007

# Where Does Data Go to Die?

- Old but sensitive data permeates systems
  - ▶ Throughout user and kernel space
- How long does this data sit around for?
  - ▶ Seconds? Minutes? Hours? Days? Weeks?
- What are the implications of all this sensitive data being accessible past the end of its useful life?
- How do we get rid of it?



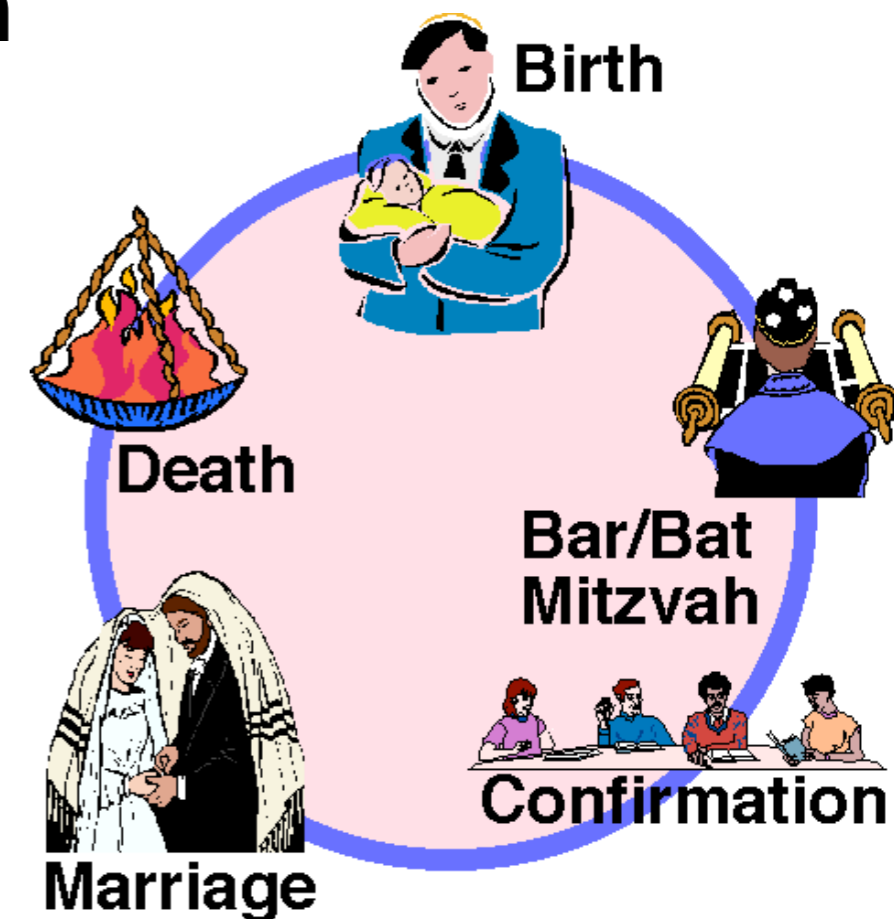
# Data Leakage

- Direct compromise of a system
- Software bugs that leak memory
- Unintended interactions (dumps, logs, etc.)
- Accidental reuse of data (dirty pages)
- Unanticipated leaks to disk or NAS
- *Applications!*
  - ▶ For the most part, not designed to deal with sensitive data
  - ▶ OSes, libraries, runtimes have problems

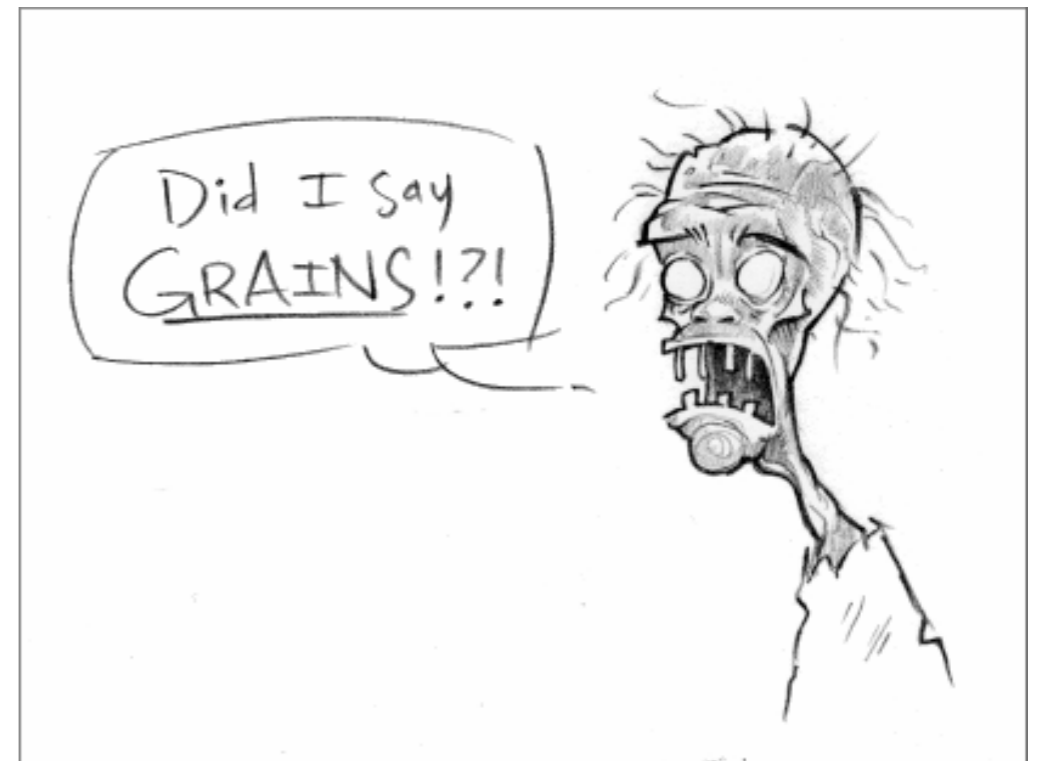


# Life Cycles of Data

- What is a life cycle when talking about data?
- Ideal: First write after allocation to last read before deallocation
- Natural: First write after allocation to first write after next allocation
- Secure Deallocation: First write after allocation until explicit deallocation



- Why does some data stick around?
- Effects
  - ▶ data still around 1-2 weeks after last usage
- *Holes* caused by slab allocator
  - ▶ Another term for this?
- Warm vs. cold reboot
  - ▶ How is memory affected?
  - ▶ Why does a ThinkPad act differently?



# Secure Deallocation

- What is the process?
  - ▶ Zero out sensitive information when it's finished being used
- Where is the best place to do deallocation?
  - ▶ Every layer!
  - ▶ Applications: Best knowledge of where sensitive data is and when to clean it; complex and laborious to identify all spots for deallocation
  - ▶ Compilers: Static requests vs. Libraries: dynamic requests
  - ▶ OS: final spot where clearing can be done

# How to Clear Data

- In compilers/libraries:
  - ▶ *free* call zeroes allocated heap data
  - ▶ on stack, zero activation frames or all data below SP
- In kernel:
  - ▶ use semantic info to selectively clear structs
  - ▶ user space memory, I/O buffers
  - ▶ zeroing large areas of memory (e.g., pages) has performance tradeoffs
  - ▶ what is this similar to?



- Secure deallocation makes data last about 1.3 times longer on average than ideal case but much less time than natural lifetime would be
  - ▶ e.g., Mozilla: 11s ideal, 21s secure, 40s natural
  - ▶ Thunderbird: 5s ideal, 10s secure, 34s natural
- Anything pop out at you looking at the results?
- Kernel clearing strategies: < 7% overhead for heap clearing, <2% for stack clearing
  - ▶ but if stack clearing needs to be done right after allocation, overhead between 10-40%

- Problem identified, mechanisms created for solving the problem
- What would your next paper be?
  - ▶ apply to different domains (VMMs, programming language runtimes)
  - ▶ go lower: build into OS (or hardware? ephemeral mem)
  - ▶ Consider effect of new technologies (NVRAM?)
  - ▶ parameterize and tune algorithms
  - ▶ expand aspect of work (rather than C, look at GC languages)

- Insight: many (most) applications don't think about security (even those that explicitly deal with it)
- Minimize reliance on users/coders who will make bad decisions... minimize complexity to them
- Defence in depth (many things may need modification to ensure complete coverage)
- I like puppet shows

