Making Information Flow Explicit in HiStar
Lecture 25, cs262a

Ion Stoica & Ali Ghodsi
UC Berkeley
April 23, 2018

Today's Paper

Making Information Flow Explicit in HiStar,
Nickolai Zeldovich, Silas Boyd-Wickizer, Eddie Kohler, and David Mazières
https://people.csail.mit.edu/nickolai/papers/zeldovich-histar.pdf

Motivation

- Security vulnerabilities discovered in all kinds of apps
    - Buffer overflows, format string issues, SQL injection, JS injection, temp file races, integer overflows
- Security implemented at many different levels
    - Web app implements its own logic, e.g. private Facebook posts
    - Web servers implement access to different directories (.htaccess)
    - OS implements its own ACLs, users, SU, …
    - Hardware implements security, page tables, etc
- Bugs could exist anywhere, high level info can be leaked at any level!
    - Meltdown leaking secret webapp info to another tenant

Main idea

- Small kernel (20k LoC) that controls information flow
  - Don’t care about bugs in programs, make sure kernel isn’t buggy
  - Control the information flow between potentially buggy programs
  - Seen this idea before?

- Example
  - Antivirus needs to scan all your files.
  - It will see confidential information.
  - If the AV code is malicious, it can communicate that code out over the Internet
  - Kernel can simply now allow AV to send info anywhere

Military research in the 70s: **Bell LaPadula**

- Bell Lapadula
  - Preserve confidentiality
  - Subjects reading/writing Objects
  - Subjects and Objects given a level, e.g. 1…4 (unclassified…top secret)
- No read up
  - Subject at level i cannot read object at level j when $i < j$
  - e.g. anonymous user reading root’s files (could leak /etc/passwd)
- No write down
  - Subject at level i cannot write object at level j when $i > j$
  - e.g. root writing to /user/anonymous (could leak secret info to anonymous)

Military research in the 70s: **Biba**

- **Biba**
  - Preserve integrity / trustworthiness
  - Who would you trust when receiving information?
- No write up
  - Subject at level i cannot write object at level j when $i < j$
  - Cannot authoritatively provide information to the upper levels
- No read down
  - Subject at level i cannot read object at level j when $i > j$
  - Cannot trust information from lower levels

Military Operating Systems

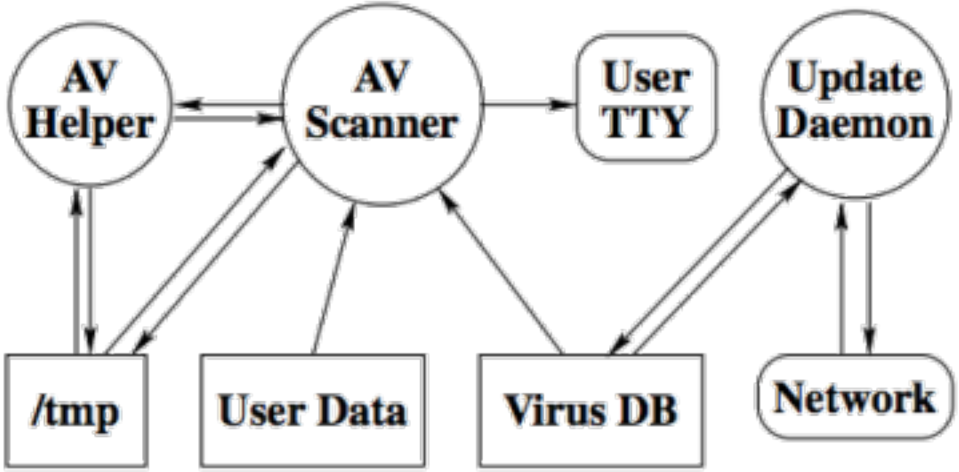
- Early OS:s implemented these ideas for file systems
  - Policies on how top secret or classified information could be handled
  - Reading and writing of files were protected
  - How is it different from today’s file systems and their security?
- Application level concepts wouldn’t get these benefits
  - OS doesn’t know about the app data
  - HiStar exposes these security mechanisms to all apps
  - **Information Flow** as basic OS mechanisms exposed to apps!
  - Can implement Bell Lapadula and Biba in any app!

Information Flow Control (IFC)

- How should we track information flow?
  - Associate a **Label** with the data
  - Label follows data when it moves around
  - Labels determine what you can do with the data
    - e.g. SSN cannot be sent to any other computer

Example: Virus Scanner

- AV Scanner/Helper
  - Read virus DB (signatures)
  - Read **all** files
  - Read/write tmp files
  - Write to screen scan status

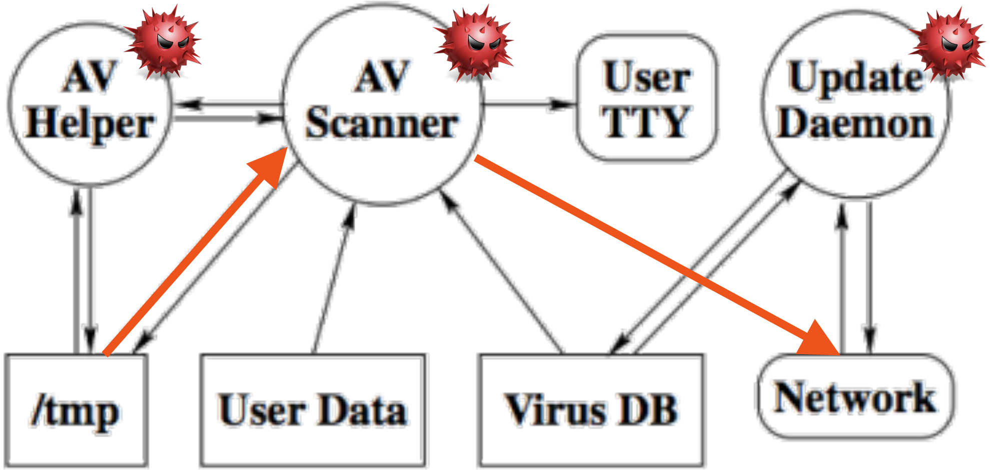


- Update daemon
  - Read/write data to Internet to fetch latest virus DB
  - Write to the virus DB to update it

- Can we protect files from corruption and leakage to outside?

Problems today

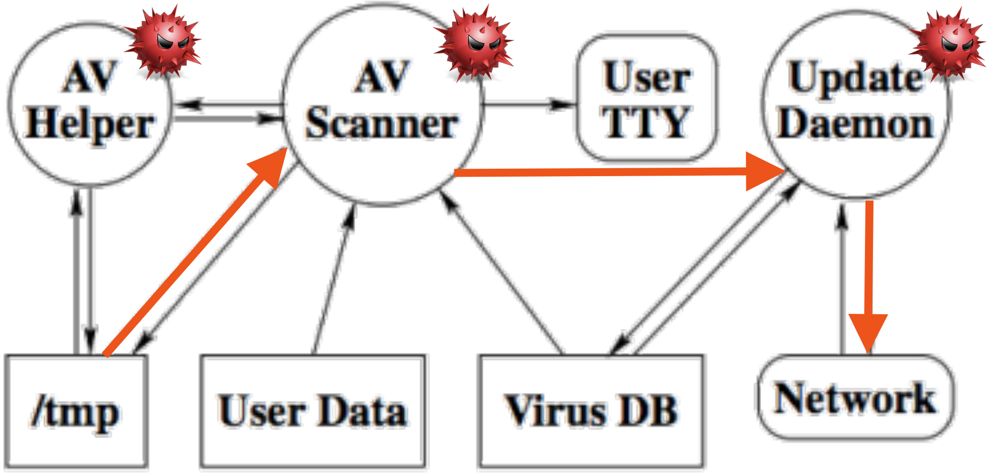
- Any process can get hacked
- Ways in leaks could happen?
  - Send private data to Internet



- Prevent AV scanner to communicate with the Internet!

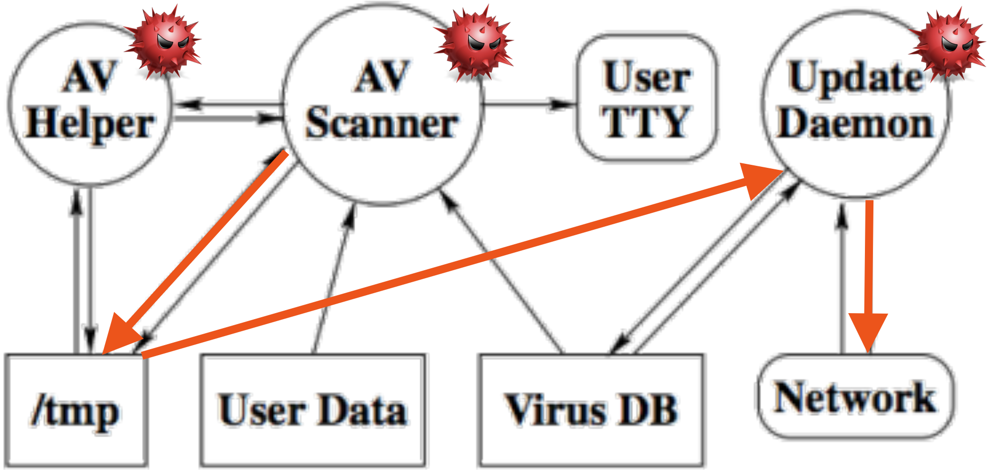
Problems today

- Any process can get hacked
- Ways in leaks could happen?
  - Collude with Update DM
  - Update DM needs Inet
- Prevent IPC too!



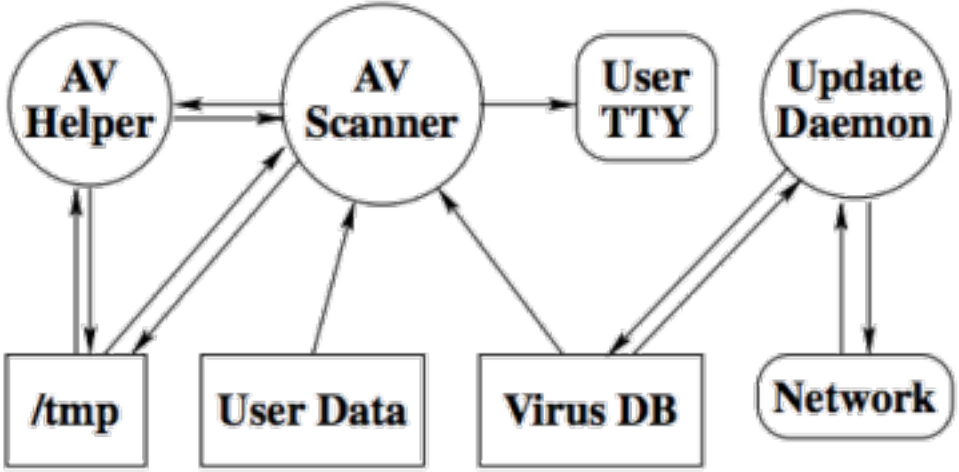
Problems today

- Any process can get hacked
- Ways in leaks could happen?
  - AV write data to tmp files
  - Update DM read tmp files



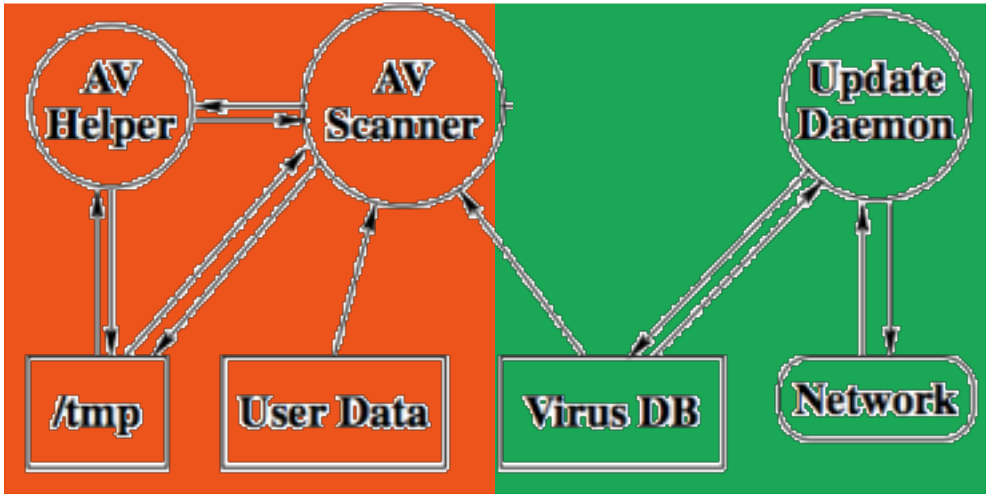
Problems today

- OS today have protection
  - File systems with RBAC
  - Process protection
  - Memory protection
- What's the problem?
  - They **ignore** information flow
  - Process P can read a secret file it has access to and write it to a public file
  - P does so either maliciously or by getting hacked, e.g. buffer overflow
- OS:s allow violating Bell Lapadula (no write down violated)



Information Flow to save us!

- Information Flow Solution
  - Files & processes colored
  - Label private stuff RED
  - Label public stuff GREEN
- Enforce the arrows in the chart



Kernels Objects

Six kernel objects

- Segment (data itself), array of bytes
- Thread
- Address space
- Device (network)
- Gate (IPC)*
- Container (“directory”), ever kernel object inside a container

All of Unix implemented on top of the 6 objects!

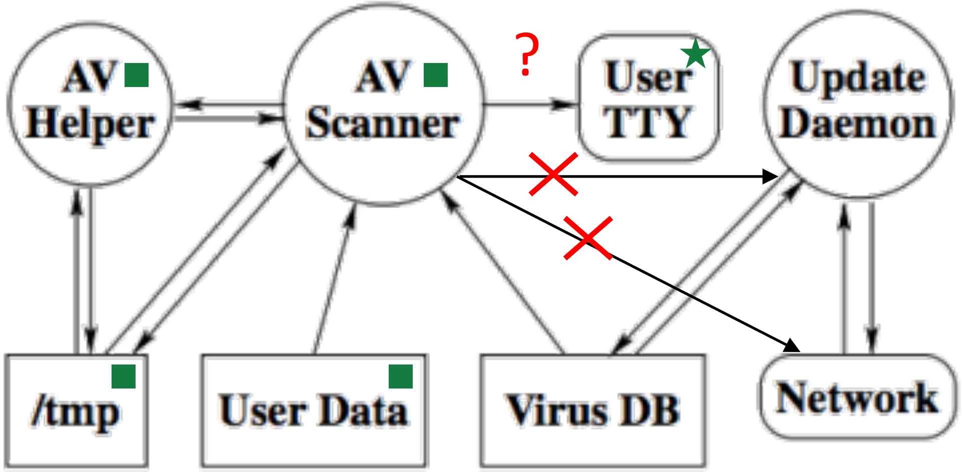
Information Flow: **Labels & Categories**

Every Kernel Object has a *label*

- Label tells you the security property of the information inside an object
- Since an object (e.g. Thread) might contain multiple types of information, labels contain multiple *Categories* (think of a category as a color)

HiStar will only allow kernel objects to interact (information to flow) if two kernel objects have “consistent” labels, i.e. implement Bell Lapadula/Biba

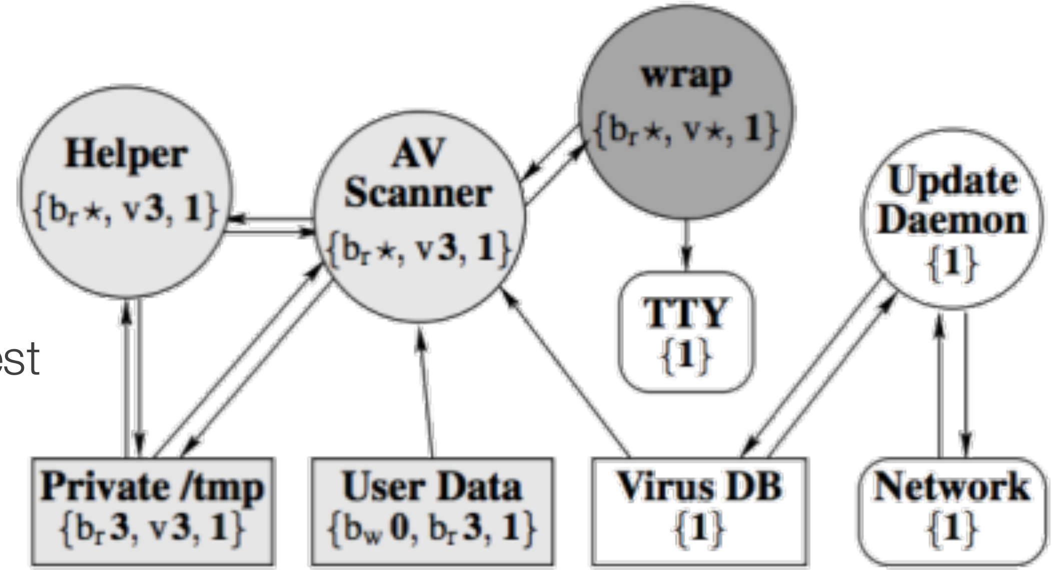
Antivirus example fixed with information flow



Great. But how do we get the AV result to the terminal screen?

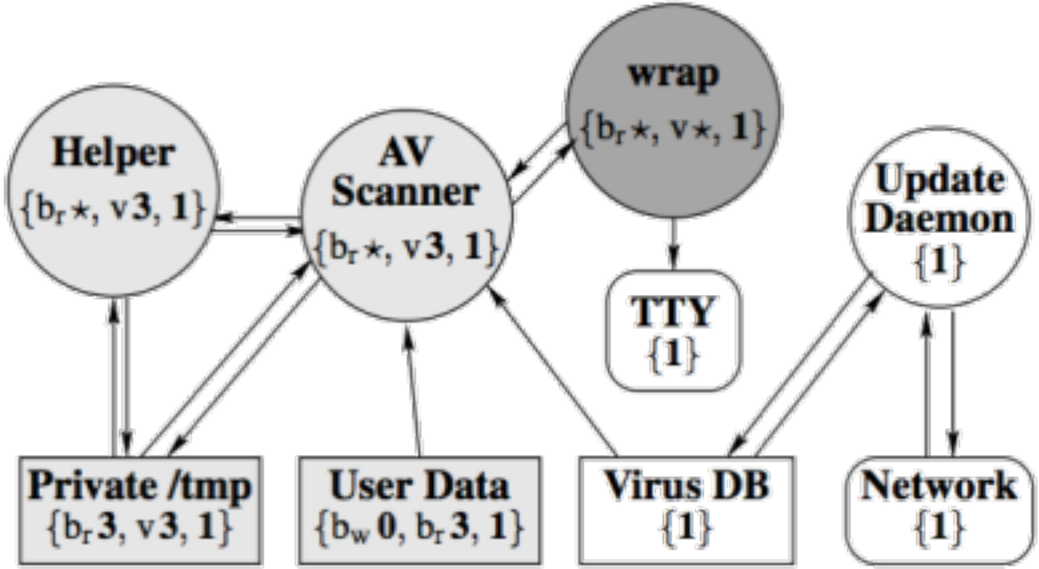
- Process creating a category (color) *owns* ★ it: it can declassify it and bypass restrictions on that category.
- Small 140 line wrapper script extracts the AV output and prints it

AV explained in full



- Colors (categories) have levels, ★ own, 0 lowest, 1 default, 4 highest
- Bob marks all User Data as color $\{b_w 0, b_r 3, 1\}$, i.e. color $bob_{write}=0$, $bob_{read}=3$, all other colors = 1
- wrap creates and ★ owns v (virus) category, and owns read category (can downgrade and bypass v and r restrictions)
- wrap spawns virus scanner and helper with v level 3, /tmp with v level 3
- **AV/helper cannot communicate with network, update daemon, because they don't have color v=3, it can write secrets to /tmp (but others cannot read it)**
- **AV/helper cannot corrupt files, because they don't have b_w permission**
- **All communication through trusted 140 line wrap**

Unix vs IFC



How is this different from Unix? We just have to be careful with permissions, right?

- No, HiStar tracks *information flow*!
- Any information flow out of AV gets tainted as special virus permission v3
- If you don't have v3, AV scanner cannot leak to you. No matter how buggy AV/Helper is, no matter how many buffer overflows or malicious code snippets it has!
- In Unix, AV scanner might by mistake leak information to a public file, screen, or network

Conclusion

- Current OSs have too many aspects that need to be secured
    - Sloppy code in many places lead to vulnerabilities
- HiStar offers a minimalistic kernel that exposes information flow
    - Six objects in the kernel
    - Each object has a label (categories/colors)
    - Information flow controlled between objects
    - All of Unix implemented on top of these few abstractions
    - New applications can implement security using these abstractions

Zooming out

- Radically different approach to OSs.
    - DAC vs MAC
- Attacks the problem with a small microkernel
    - Everything else implemented on top of it
    - All future applications benefit from the security of HiStar
- Great bold paper! Realistic implementation.
    - Why didn’t it have more impact?