

Asbestos

Petros Efstathopoulos, Maxwell Krohn, Steve VanDeBogart, Cliff Frey,
David Ziegler, Eddie Kohler, David Mazières, Frans Kaashoek, and
Robert Morris

MIT, NYU, UCLA

Motivation

- **Most software cannot be trusted**
 - Massive, complex systems no one fully understands
 - Not written by security-conscious programmers
 - Even good programmers make mistakes
- **Yet this is what people develop and want to run**
- **What can be done?**
 - SRS: Diversity, Replication, Healing, ...
 - **Asbestos: Address problem through better OS interface**

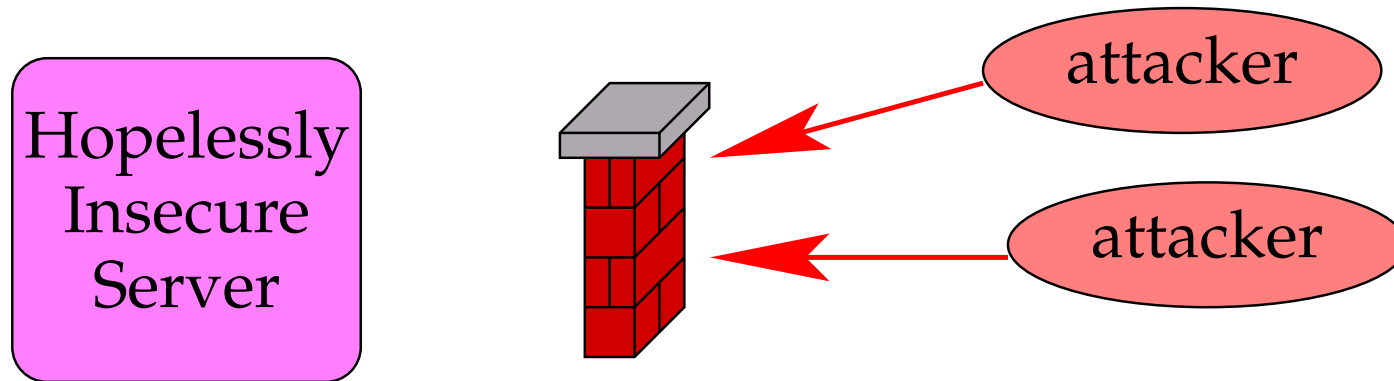
Examples

- **PayMaxx web site divulges social security numbers**
 - Test record had SSN 000-00-0000 and no password
 - After login, could access records by consecutive serial number
- **CSU food center divulges social security numbers**
 - Admins discovered files containing personal information
 - No way to know if information was downloaded
- **The list goes on...**
 - CardSystems loses 40,000,000 credit card numbers
 - Recommendation letters for 10,000 Stanford applicants stolen
 - Jacobsen compromises T-mobile, steals secret service mail

Goals

- **Address a wide range of software flaws...**
 - Buffer overruns
 - Trojaned machines
 - SQL injection
- **...without trusting/understanding whole application**
- **Attacks we *don't* address:**
 - Social Engineering
 - Find a UPS box full of hard drives
 - Sniffed passwords

Analogy: Firewalls



- **Your machine runs hopelessly insecure software**
 - Can't fix it—no source or too complicated
 - *Can* reason about network traffic
- **Block interaction with network attackers**
- **Take similar approach within single machine**
 - Control software by “filtering” process interactions

An approach: Interposition

- **Limit what a process can interact with**
 - On Unix, use of chroot has improved security
 - But heavy-weight, clunky, and incomplete isolation
- **Capability-based systems less clunky**
 - Include other benefits (like lack of ambient authority)
- **Plan9-like namespace could make capabilities easy to manipulate**
- **Problem: Achieving end-to-end security guarantees**
 - Hard to guarantee a process won't indirectly exercise capability
 - E.g., Keynix must trust all multi-level objects

An approach: Interposition

- **Limit what a process can interact with**
 - On Unix, use of chroot has improved security
 - But heavy-weight, clunky, and incomplete isolation
- **Capability-based systems less clunky**
 - Include other benefits (like lack of ambient authority)
- **Plan9-like namespaces could make capabilities easy to manipulate**
- **Problem: Achieving end-to-end security guarantees**
 - Hard to guarantee a process won't indirectly exercise capability
 - E.g. Keyring must trust all multi-level objects

Example: PayMaxx web site

- Users authenticate themselves with a password
- Web server accesses database...
- ...potentially performs many functions
 - E.g., generate PDF files of W2 forms
- **How to prevent user *A* from seeing user *B*'s data?**
 - Any process with database access could leak information
 - Server very probably has bugs
- **Problem calls for Mandatory Access Control (MAC)**

Traditional MAC

- **Could try traditional (orange-book-like) OS**
 - Security administrator creates one compartment per user
 - Run one instance of web server per compartment
 - Pw checker & DB need sanitization privs – in TCB
- **Problem: Scalability**
 - Need one compartment & process per user
 - Must create compartments on-the-fly with new users
- **Problem: Centralized control**
 - Can't be deployed without sanitization privs

Programming Languages

- **Could implement server in a language like JiF**
 - Set up principles corresponding to users
 - Label data and output channels appropriately
- **Advantages:**
 - “Tainting” applies to data, not whole process
 - Much less state required per active user
 - Appropriate sanitization can be done outside of TCB
- **Disadvantages:**
 - Can’t dynamically create new users
 - JiF effectively becomes your OS (might or might not be practical)

Asbestos design goals

- **Scalability:**
 - Handle 1,000s of compartments
- **Decentralized control:**
 - Even unprivileged processes can create compartments on-the-fly
 - Sanitization can be performed outside the TCB
- **Fine granularity:**
 - Closer to JiF than to traditional MAC
- **Support multiple security models:**
 - Should support OS-level fine-grained MAC
 - Should equally support capability-like DAC

Implications of goals

- **Anybody can create a compartment**
 - Achieved through 61-bit handles, unique until reboot
 - Labels map handles to sensitivity levels
- **Specify policies about two processes (e.g., *P* can't talk to netd)**
 - Labels have asymmetric default values on handles
- **Decentralized declassification**
 - A special sensitivity level, \star , allows a process to escape a particular compartment
- **Discretionary policies**
 - Possession of \star can be used like discretionary capabilities

Asbestos Labels

- A **label** is a function $L : \mathcal{H} \rightarrow \{\star, 0, 1, 2, 3\}$
 - \mathcal{H} is the space of 61-bit *handles* (= compartments)
 - $\{\star, 0, 1, 2, 3\}$ are *sensitivity levels* (also used for integrity)
 - 0 corresponds to high integrity, 3 high secrecy
- Write $L = \{h_1 v_1, h_2 v_2, \dots, h_n v_n, v\}$ iff
 $\forall i \leq n, L(h_i) = v_i$ and $\forall h \notin \{h_i\}, L(h) = v$
- As usual, labels form a lattice:
 - $L_1 \leq L_2$ iff $\forall h \in \mathcal{H}, L_1(h) \leq L_2(h)$ where $\star < 0 < 1 < 2 < 3$
 - $\text{LUB}(L_1, L_2) = L$ iff $\forall h, L(h) = \max(L_1(h), L_2(h))$
- Other notation:
 - Define LUB^\star as LUB with $0 < 1 < 2 < 3 < \star$
 - Let $L^0 = \text{LUB}(L, \{0\})$, $L^\star = \text{LUB}^\star(L, \{3\})$

Process labels

- Each process P has a *send* and a *receive label*
 - Intuitively the receive label, P_R , is like “clearance”
 - The send label, P_S , tracks information received [HiWat]
- When P sends a message to Q :
 - Asbestos requires that $P_S \leq Q_R$
 - Asbestos increases $Q_S \leftarrow \text{LUB}^*(P_S^0, Q_S)$
- \star denotes ability to escape a compartment
 - E.g., if before send $P_S = \{j\mathbf{3}, k\mathbf{2}, \mathbf{1}\}$ and $Q_S = \{j\star, \mathbf{1}\}$,
 - Then after receive $Q_S = \{j\star, k\mathbf{2}, \mathbf{1}\}$.
- \star can be re-granted in a discretionary way
 - If $P_S(h) = \star$, P can lower $Q_S(h)$ or raise $Q_R(h)$

Creating handles

- **Any (unprivileged) process P can create a handle**
 - Kernel chooses unique h
 - Sets $P_S(h) \leftarrow \star$; now P “owns” h
- **Handles can also be used for communication**
 - Messages sent to communication handles go to creator
- **Each handle h has a *receive label* h_R**
 - Default value of $h_R = \{h\ 0, 3\}$
 - If Q created h , can set h_R to arbitrary value
 - If P sends to h , Asbestos requires $P_S \leq h_R$ and $P_S \leq Q_R$
- **By default, send label $\{h\ \star\}$ like capability for h**

Example: Isolation

- **Goal:** P wants to create isolated process Q
 - Q should be allowed to communicate only with P
- P creates two handles:
 - j – secrecy handle, prevents others from observing Q
 - k – integrity handle, prevents Q from observing others

Labels	P	Q	Others
<i>Send</i>	$\{j^*, k^*, 1\}$	$\{j\mathbf{3}, k\mathbf{0}, 1\}$	$\{j\mathbf{1}, k\mathbf{1}, 1\}$
<i>Receive</i>	$\{j\mathbf{3}, k\mathbf{2}, 2\}$	$\{j\mathbf{3}, k\mathbf{0}, 2\}$	$\{j\mathbf{2}, k\mathbf{2}, 2\}$

Example: Isolation

- **Goal:** P wants to create isolated process Q
 - Q should be allowed to communicate only with P
- P creates two handles:
 - j – secrecy handle, prevents others from observing Q
 - k – integrity handle, prevents Q from observing others

Labels	P	Q	Others
<i>Send</i>	$\{j^*, k^*, 1\}$	$\{j\mathbf{3}, k\mathbf{0}, 1\}$	$\{j\mathbf{1}, k\mathbf{1}, 1\}$
<i>Receive</i>	$\{j\mathbf{3}, k\mathbf{2}, 2\}$	$\{j\mathbf{3}, k\mathbf{0}, 2\}$	$\{j\mathbf{2}, k\mathbf{2}, 2\}$

Asymmetric defaults

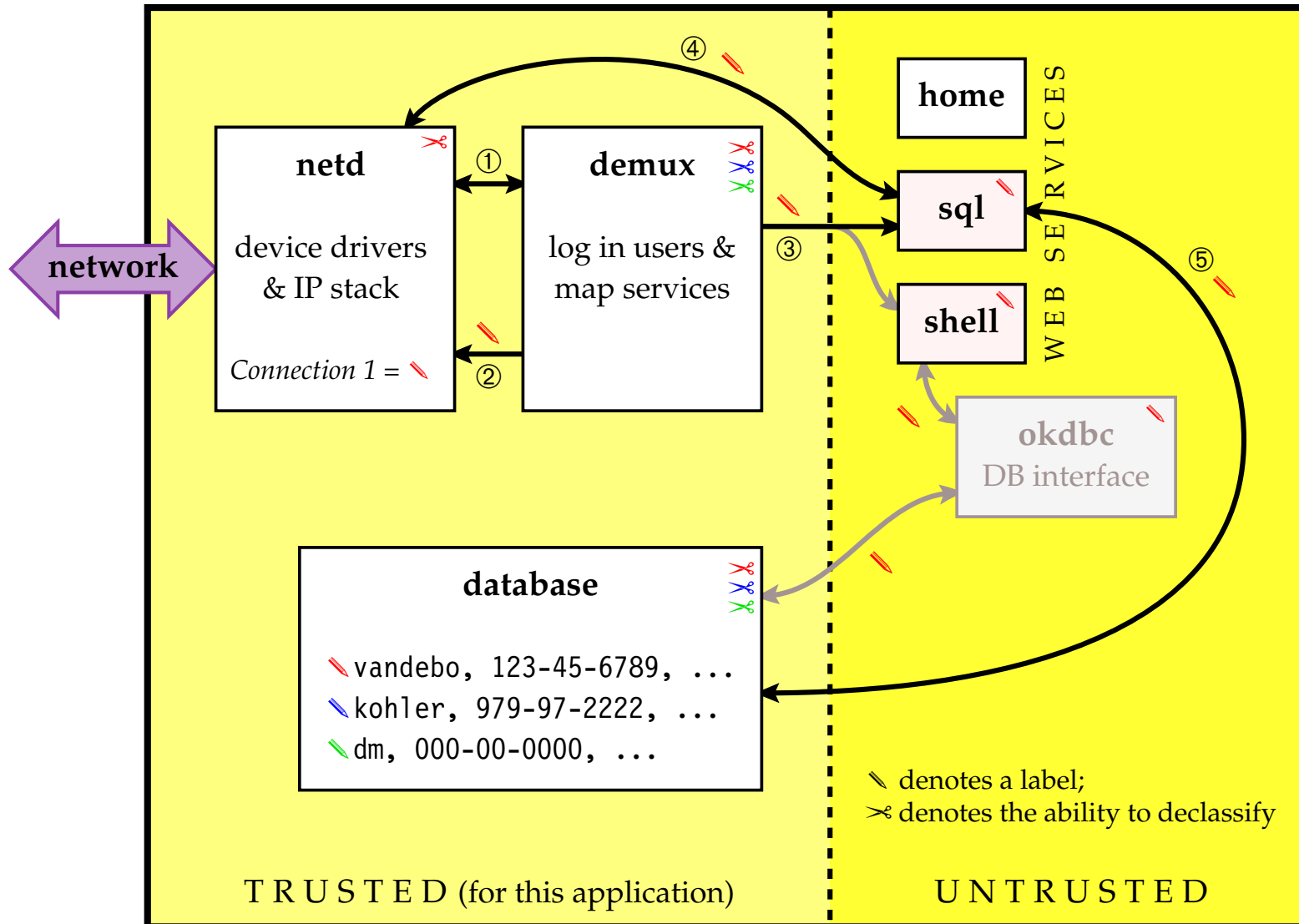
- The default send label is $P_S = \{1\}$
- The default receive label is $P_R = \{2\}$
- Asymmetry allows for process separation
- Example: Prevent P from sending messages to Q

Labels	P	Q	Others
<i>Send</i>	$\{j2, 1\}$	$\{1\}$	$\{1\}$
<i>Receive</i>	$\{2\}$	$\{j1, 2\}$	$\{2\}$

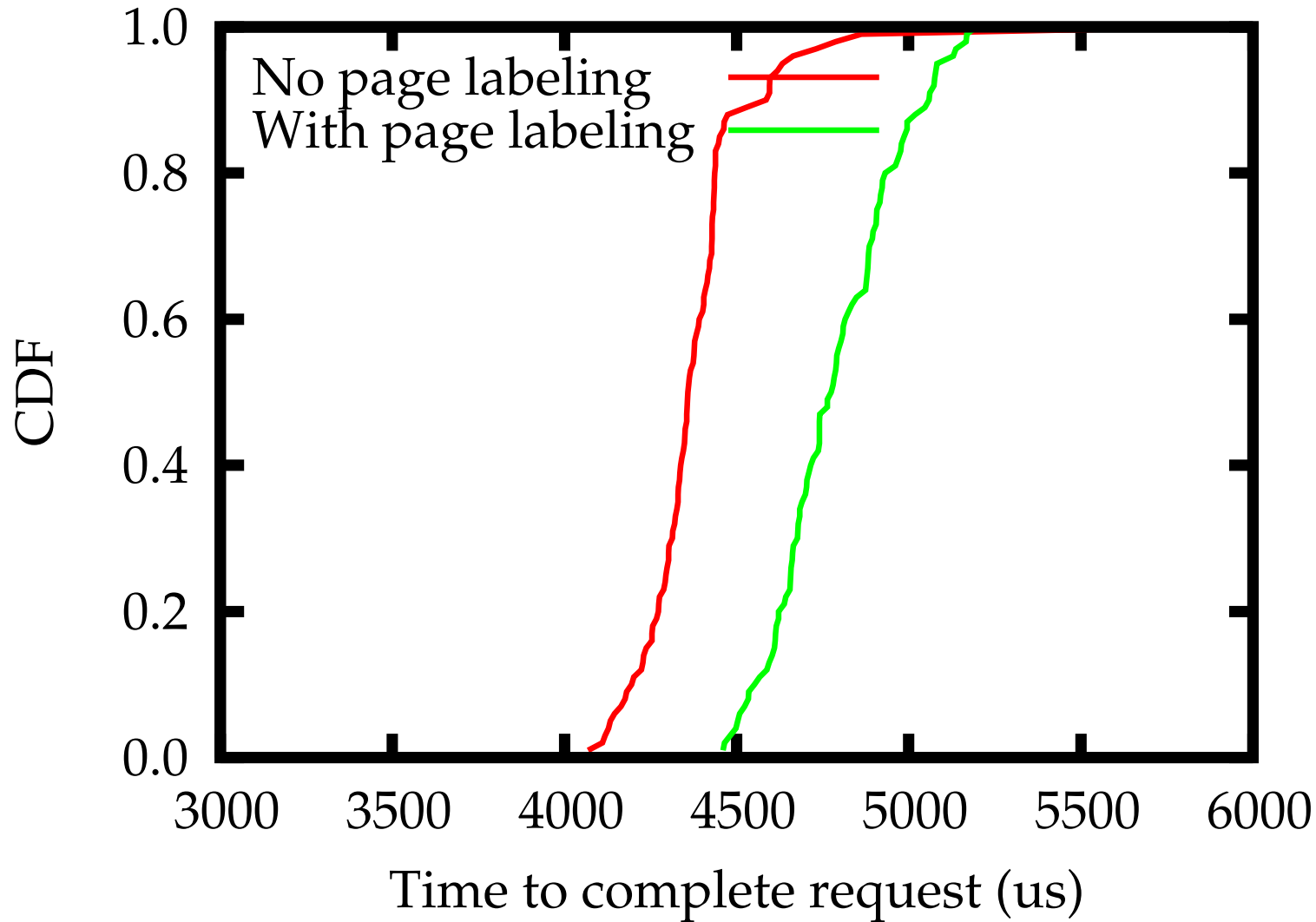
Avoiding Label Accumulation

- **Problem: Processes might accumulate restrictions**
 - E.g., Web server sees multiple users' data
 - Should we prevent it from talking to network again?
- **Solution: Virtual memory page labeling**
 - Keep track of which pages contain “tainted” data
- **Two new system calls:**
 - `vm_save` – Save process's label, wait for message
 - `vm_restore` – Revert state to time of `vm_save`
- **Allocated pages preserved across identical labels**
 - Allows processes to keep per-user state
 - Prototype web server requires 1 page per active user

Asbestos web server



Initial performance



Memory utilization

- **W/o page labeling, need 20 physical pages per user**
 - 1 page directory, 4 page tables
 - 1 user stack, 1 user exception stack, 1 BSS, 1 heap
 - Other pages arguably artifacts of implementation
- **Page labeling Asbestos:**
 - 1 simultaneous user requires 1,264 physical pages
 - 2 simultaneous users require 1,265 physical pages
 - ⋮
 - 25 simultaneous users require 1,288 physical pages
 - **~ 1 page per external connection**

Design space

- **Sub-process information flow control + dynamic compartments + decentralized declassification a win**
- **Given multiple “address spaces” per process...
...many possible design points**
- **We plan to explore the end points:**
 - Make address spaces more like processes
 - Kernel doesn't even provide a process abstraction

Event processes

- **Forked from existing processes**
 - Most memory copy on write
 - Explicit shared-memory regions
- **Scheduling is mutually exclusive**
 - Only one PC/trapframe for all event processes in one process
- **Can be forked by another process**
 - When event must be handled by new label
- **Status: Implemented, untested**
 - Programming model similar to page labeling
 - Implementation seems much simpler

Gates/Segments model

- **Decouple execution threads and address spaces**
 - I.e., one thread can go through multiple labels/address spaces
- ***Segments* – Virtual memory with fixed label**
 - Can cheaply copy a segment to give it a new label
- ***Gates* – Protected control transfer**
 - Specifies virtual address space in terms of segments
 - Labels protect invocation
 - Can copy a gate to give it more capabilities
- ***Multi-level environments* – lookup tables**
- **Status: Still in design phase**
 - “Acetone” prototype with only one address space
(proof of concept, not practical with 48-bit address space)

Conclusions

- **Two biggest threats today:**
 - Buggy code
 - Malware (Spyware trojan horses, viruses, etc.)
- **Both addressed by MAC**
- **Asbestos MAC designed for modern Internet applications**
 - Decentralized mechanisms for applications outside TCB
 - Scalable to many compartments
 - Sub-process granularity minimizes memory usage