Labels and Event Processes in the Asbestos Operating System

Petros Efstathopoulos* Maxwell Krohn† Steve VanDeBogart*
Cliff Frey† David Ziegler† Eddie Kohler* David Mazières‡
Frans Kaashoek† Robert Morris†

*UCLA †MIT ‡Stanford

http://asbestos.cs.ucla.edu
Target Application

E-commerce web server

• Thousands of users
• Users change quickly
• Necessarily reachable
• Historically vulnerable
  - PayMaxx divulged social security numbers
  - Stanford 10,000 recommendation letters
  - CardSystems exposed 40 million credit card numbers
What's the Problem?

• Bugs in web server or web application

• Many types of bugs allow access to server data
  - Buffer overflows
  - Missing access check
  - SQL injection

• It's nearly impossible to eliminate all bugs

• Want to minimize exposed data despite bugs

• Traditional OSs don't provide appropriate security mechanisms for this
The Problem

- If Bob compromises the system, he can access Alice's data.

Alice
123 Main St.
4275-8204-4009-7915

Bob
456 Elm St.
5829-7640-4607-1273
The Problem

- If Bob compromises the system, he can access Alice's data

Alice
123 Main St.
4275-8204-4009-7915

Bob
456 Elm St.
5829-7640-4607-1273
The Problem

- If Bob compromises the system, he can access Alice's data.

The problem involves a scenario where Bob has compromised the system, allowing him to access Alice's data. The diagram illustrates the flow from Bob to the kernel via the cloud, accessing Alice's data through `/submit_order.cgi`. The addresses and contact numbers are:

**Alice**
- Address: 123 Main St.
- Contact: 4275-8204-4009-7915

**Bob**
- Address: 456 Elm St.
- Contact: 5829-7640-4607-1273
The Problem

- If Bob compromises the system, he can access Alice's data

Alice
123 Main St.
4275-8204-4009-7915

Kernel

/submit_order.cgi

Alice
123 Main St.
4275-8204-4009-7915

Bob
456 Elm St.
5829-7640-4607-1273
The Goal: User Isolation

• Bob should not be able to access Alice's data without Alice's permission
  - Alice and Bob’s data is isolated

• Complications
  - Even if there are bugs in the applications
  - Alice's data may travel through several processes

• To isolate, must prevent inappropriate data flow

• Application designer defines inappropriate
Virtual Machine Isolation

Kernel

/submit_order.cgi

Alice
123 Main St.
4275-8204-4009-7915

VMM

Kernel

/submit_order.cgi

Bob
456 Elm St.
5829-7640-4607-1273
Virtual Machine Tradeoffs

- Strict partitioning of off-the-shell software
- But...
  - Course-grained sharing
  - Resource challenges
- Isolation should be an OS feature
Desired Behavior

Kernel

/submit_order.cgi

Alice
123 Main St.
4275-8204-4009-7915

Bob
456 Elm St.
5829-7640-4607-1273
Desired Behavior

Kernel

Alice
123 Main St.
4275-8204-4009-7915

/submit_order.cgi

Alice
123 Main St.
4275-8204-4009-7915

Bob
456 Elm St.
5829-7640-4607-1273
Desired Behavior

Kernel

Alice
123 Main St.
4275-8204-4009-7915

/submit_order.cgi

Alice
123 Main St.
4275-8204-4009-7915

Bob
456 Elm St.
5829-7640-4607-1273
Information Flow Control

- Information flow control solves this kind of problem
Label data with its owner (contaminate with respect to its owner)
Information Flow Control

Kernel

/submit_order.cgi

Alice
123 Main St.
4275-8204-4009-7915

Bob
456 Elm St.
5829-7640-4607-1273

Keep track of who the connection is for
Track the information as it moves around the operating system.
Information Flow Control

Kernel

/reject

Base access control decisions on labels

/submit_order.cgi

Alice
123 Main St.
4275-8204-4009-7915

Bob
456 Elm St.
5829-7640-4607-1273
Information Flow Control Systems

• Conventional multi-level security
  - Kernel-enforced information flow control across processes
  - A handful of levels and compartments: “secret, nuclear”
  - Inflexible, administrator-established policies
  - Central authority, no privilege delegation

• Language-enforced information flow (Jif)
  - Applications can define flexible policies at compile time
  - Enforced within one process

• Asbestos
  - Applications can define flexible policies
  - Kernel-enforced across all processes
Approaches

Within a process

Across processes

Policy defined by:

Application

Kernel

Conventional MLS

Asbestos

Top-Secret

Jif

10 1
Asbestos Contributions

• New message passing operating system
  – Requires rewrite of applications for new security mechanism

• Asbestos labels
  – New security mechanism to track information flow control
  – Designed to support application defined policies

• Event processes
  – A new process abstraction
  – Prevent accumulation of contamination

• Good performance
  – Wrote a web server that uses Asbestos labels & event processes
  – Acceptable performance with strong security properties up to at least 10,000 sessions
Asbestos Compartments

- A compartment is a kind of contamination / label type
- Example has two compartments: Alice & Bob
- Alice might have multiple compartments
  - Financial secrets
  - Romantic secrets
- Compartments can overlap
- Application defines compartment policy
  - Kernel enforces policy
Asbestos Labels Build Flexible Applications

• Application can create compartments without privilege
  – Application created users are isolated with the same mechanism as login users
  – Applications can easily sub-divide privilege

• Applications can delegate rights for compartments
  – Decentralized declassification like Jif

• Applications can choose different policies
  – Mandatory Access Control
  – Discretionary Access Control
  – Capabilities
  – More...
Label Basics

• Each process has a send and receive label
  – The send label tracks current contamination
  – The receive label tracks contamination limits (clearance)

• Rules track information flow

• Similar to IX
Basic Example

Alice's ahttpd

Bob's ahttpd

cgi script

Backend DB

Send Label

Recv Label
Basic Example

Alice's ahttpd  Bob's ahttpd  cgi script  Backend DB

Send Label

Recv Label
Basic Example

Rule 1:
The kernel contaminates the message with all of the sender's contamination

Send Label

Recv Label
Rule 2:
The kernel validates that the destination has clearance to receive the contamination of the message.
Rule 3:
At delivery, the destination takes on the contamination of the message
Basic Example

Alice's ahttpd

Bob's ahttpd

cgi script

Backend DB

Send Label

Recv Label
Implementing Clearance Checks

- How does the clearance check work?
- Labels form a lattice
- Partial ordering
  - Sender's send label must be less than or equal to the destination's receive label
- Send label updated with a least upper bound operator
Limiting Bug Impact

Alice's ahttpd
Bob's ahttpd
cgi script
Backend DB

Send Label
Recv Label

![Diagram showing flow between Alice's and Bob's ahttpd, a cgi script, and a backend DB, with labels indicating the impact of a bug.]
Limiting Bug Impact

Alice's ahttpd → Bob's ahttpd → cgi script → Backend DB

Send Label: Biohazard
Recv Label: Biohazard, None
Limiting Bug Impact

Alice's ahttpd

Bob's ahttpd
cgi script

Backend DB

Send Label

Recv Label
Limiting Bug Impact

Alice's ahttpd → Bob's ahttpd → cgi script → Backend DB

Send Label

Recv Label
Limiting Bug Impact

Alice's ahttpd  Bob's ahttpd  cgi script  Backend DB

Send Label Recv Label
Limiting Bug Impact

Alice's ahttpd

Bob's ahttpd

cgi script

Backend DB

Send Label

Recv Label
Application Defined Policies

- Where did the compartments come from?
- How did the labels get set the way they are?
- In traditional multi-level security systems, the system operator does these things
- Asbestos labels provide a decentralized and unprivileged method to set these initial conditions
Compartment Creation

Alice's ahttpd
Bob's ahttpd
cgi script
Backend DB

Send Label
Recv Label
Compartment Creation

Alice's ahttpd → password → Bob's ahttpd → cgi script → Backend DB

Send Label

Recv Label
Any process that creates a compartment gets privilege with respect to that compartment:

- Declassify data
- Grant clearance
- Delegate privilege
Declassify Receive

Alice's ahttpd
Bob's ahttpd
cgi script
Backend DB

User
Kernel
Send Label
Recv Label
Optional Labels

- Process can attach optional (discretionary) labels to messages
  - $C_s$ – Contaminate Send
  - $D_r$ – Declassify Receive
  - $D_s$ – Declassify Send
  - $V$ – Verify
Declassify receive grants clearance for a compartment to another process.
Declassify Receive

The kernel checks that processes have the privilege needed to grant clearance
Declassify Receive

Alice's ahttpd → Bob's ahttpd → cgi script → Backend DB

Send Label

Recv Label

$D_R = \text{biohazard}$
Declassify Receive

Alice's ahttpd
Bob's ahttpd
cgi script
Backend DB

$D_R =$
Declassify Receive

Alice's ahttpd

Bob's ahttpd

cgi script

Backend DB

Send Label

Recv Label

$D_R =$
Declassify Receive

Alice's ahttpd
Bob's ahttpd
cgi script
Backend DB

Send Label
Recv Label
Contaminate Send

Alice's ahttpd

Bob's ahttpd

cgi script

Backend DB

Send Label

Recv Label

$C_s =$
No privilege needed for $C_s$ – it can only add processes to a compartment.
Contaminate Send

Alice's ahttpd
Bob's ahttpd
cgi script
Backend DB

Send Label
Recv Label

$C_s =$
Contaminate Send

Alice's ahttpd

Bob's ahttpd

cgi script

Backend DB

Send Label

Recv Label

C_s =
Contaminate Send

Alice's ahttpd
Bob's ahttpd
cgi script
Backend DB

Send Label

Recv Label

\[ C_s = \]
Contaminate Send

Alice's ahttpd ➔ Bob's ahttpd ➔ cgi script ➔ Backend DB

Send Label ➔ C_s ➔ Recv Label

= Biohazard icon
CGI Setup

Alice's ahttpd

Bob's ahttpd

cgi script

Backend DB

D_R =

Send Label

Recv Label
Bob Setup

- Alice's ahttpd
- Bob's ahttpd
- cgi script
- Backend DB

Send Label
- Biohazard

Recv Label
- Biohazard
Bob Setup

Alice's ahttpd
Bob's ahttpd
cgi script
Backend DB

Application Trust

Send Label
Recv Label
Declassification

- Information flow control keeps users data completely disjoint
- Alice wants to export some of her data, like her profile
  - But all her data is in her compartment
- How can she safely declassify her data?
- Alice must trust all process that can do so
- To minimize declassification bugs, we build declassifiers as simple, single purpose programs
Declassification

Alice's ahttpd

Bob's ahttpd

Alice's profile declassifier

Backend DB

Send Label

Recv Label
Declassification

The process must have privilege for the compartment to use both $D_S$ and $D_R$. 

$D_S = $ [Red] $D_R = $ [Red]
Declassification

Alice's ahttpd → Bob's ahttpd

Alice's profile declassifier

Backend DB

Send Label

Recv Label
Declassification

Alice's ahttpd
Bob's ahttpd
Alice's profile declassifier
Backend DB

profile

Send Label
Recv Label

biohazard
biohazard
Declassification

Alice's ahttpd → Bob's ahttpd → Alice's profile declassifier → Backend DB

Send Label: "biohazard"
Recv Label: "biohazard"
Since the process is privileged in Alice's compartment, it doesn't get contaminated.
Other Label Features

Other label details and features only in the paper

- **Verify label on messages**
  - Allows a process to prove it has labels at specific levels

- **Integrity tracking**
  - Enabled by level 0

- **Different default level for send & receive labels**
  - Enables interesting isolation policies
Label Implementation

- Contamination & Privilege = Label level (*, 0-3)

- \[ = \{A *, B 3, 1\}\]

- A & B are compartment names

- Trailing 1 = Neutral in all other compartments
  - Including those that haven't been created yet

- Label representation linear in # compartments
Combating Process Over-Contamination

- One process per user per service
  - Lots of heavy weight context switches
  - Lots of memory

- Combine processes to get one process per service?
  - Become too contaminated to function
  - Or too privileged

- Many processes are similar

- Programming style help?
while (1) {
    event = get_next_event();
    user = lookup_user(event);
    if (user not yet seen)
        user.state = create_state();
    process_event(event, user);
}

- State isolated to data structures
- Stack not used from event to event
- Execution state has nice preemption points
Event Process Abstraction

```c
ep_checkpoint(&msg);
if (!state.initialized) {
    initialize_state(&state);
    state.reply = new_port();
}
process_message(&msg, &state);
ep_yield(); // revert to checkpointed memory
```

- Fork memory state for each new session
  - Memory isolation is the same as fork
  - Small differences anticipated, stored efficiently (diff)

- Event loop allows shared execution state
  - Allows light weight context switches
Where's the fork?

• Explicit fork would be a covert channel
  – A process with a secret, N, can create N new sessions

    ep_checkpoint(&msg);
    if (!state.initialized) {
        new_sessions++;
        fork_memory(&state);
        state.reply = new_port();
    }
    process_message(&msg, &state);
    ep_yield();

• Combine memory-fork with message reception

• Fork when the message is for a new session
Communication Terms

- Communication end points are “ports”
  - Similar to Mach ports

- One process has receive rights for a port
  - Process with receive rights considered the current owner of the port
Event process created port = existing session

- Choose the event process based on the owner of the destination port
- Restore any address space modifications
- Use the send and receive label specific to the event process
Port created before `ep_checkpoint()` = new session

- Create a new event process
- Clone the address space & send and receive labels
- Event process must create a new port
Event Process Messages

Port created before `ep_checkpoint()` = new session

- Create a new event process
- Clone the address space & send and receive labels
- Event process must create a new port
Event Process Messages

Port created before `ep_checkpoint()` = new session

- Create a new event process
- Clone the address space & send and receive labels
- Event process must create a new port

\[ D_R = \]
Port created before `ep_checkpoint()` = new session

- Create a new event process
- Clone the address space & send and receive labels
- Event process must create a new port
Event Process Messages

Port created before `ep_checkpoint()` = new session

- Create a new event process
- Clone the address space & send and receive labels
- Event process must create a new port
Event Processes Work

• Event process isolate state
  - Used so that each event process is only contaminated by one user
  - One process per service with one event process per user

• Even at 10,000 event processes, state is stored efficiently

• Little additional programmer overhead because event processes fit into event driven programming style
It Works!

- Built it for x86
- Uses the e1000 network driver from Linux
- LWIP network stack & SQLite database as a backend for the web server
- Built OKWS like web server
Web Server Architecture

netd → demux → ahttpd-idd

worker_1 → db-proxy → Database

worker_N → db-proxy → Database
Performance Hypotheses

- Is the memory overhead from event processes mild, even at 10,000 sessions?
- Despite better security properties, is the performance of the OK web server on Asbestos comparable with Apache?
- Does the per connection kernel overhead increase at most linearly with the number of sessions?
Experimental Setup – Memory

- How much memory do event processes use?
- Shopping cart application
  - Session state stored in event process
  - One event process per user
- Active session – Adding an item to the shopping cart
- Cached session – Deciding if you really want an item

Click!

Hmm
Event Processes Conserve Memory

- Includes user and kernel memory
- Not too many active sessions on a large website
Experimental Setup – Throughput

• Simple character generation service
  - Not interested in application overhead
  - One event process per session (user)

• Compare to Apache & Mod-Apache
  - Varied concurrency to get best case performance

• Apache
  - Service runs as a CGI script
  - Connections are isolated into processes
  - Processes are not isolated or jailed on the system

• Mod-Apache
  - Service runs inside Apache process
For 16 sessions, 150% of Apache
For 10,000 session, 75% of Apache
Label Cost Linear in Label Size

- Throughput benchmark
- DB performance fixed
- Label cost starts small but outstrips OKWS cost around 6500 sessions
- Declassifiers label size $O(\#\text{sessions})$
Future Work

- Minimizing label costs
- Easing programmability
- Label persistence
- More applications
Related Work

• Inspired by Jif
• Dynamic labels: IX, LOMAC
• Integrity: Biba
• Capabilities: KeyKOS, EROS
Conclusion

• Asbestos labels make MAC more practical
  - Labels provide decentralized compartment creation & privilege
  - Event processes avoid accumulation of contamination

• The OK web server on Asbestos
  - Performs comparably to Apache
  - Provides better security properties than Apache