An Operating System Analog to the Perl Data Tainting Functionality

Dana M. Madsen

Overview

- **Background**
- **The Tainting File System Concept**
- **Related Work**
- **Implementation Approach**

Background

Recent Internet-Related Security Incidents

- **Fast-Propagating Worms and Viruses**
  - **– ILOVEYOU and Melissa**
- **Trojan Horse Software**
  - **– Back Orifice 2000**
- **Flaws in Java and Active-X Security**
  - **– JVM type confusion vulnerabilities (Apr and Oct 1999)**
  - **– Bubbleboy virus exploiting flawed Active-X controls**

Background (cont)

Conclusions from Recent Internet Security Incidents

- **Security Depends on User Vigilance and Competence**
  - **– Scanning email attachments for viruses**
  - **– Judging whether downloaded software is safe**
  - **– Assessing whether Internet sites are safe to visit**
- **Incomplete O/S-Level Notion of Data Trustworthiness**
  - **– Trust is based on user identity, not the source of the data**
- **Hence Poor Protection Against Untrustworthy Data**
  - **– Untrustworthy code/data could trigger malicious actions with full permissions and identity of victim user**

Tainting File System Concept

- **Add a new file attribute reflecting trustworthiness**
- **If the file is executable:**
  Constrain the execution of that file
- **If the file contains data:**
  Constrain the execution of all processes reading that file
- **Enforce flexible policies governing how file trust attribute assigned and processes constrained**

Primary Emphasis

Protect Inattentive or Unskilled Users who,

Without Malicious Intent,

Introduce Malicious Content into the File System

Any additional protection against malicious users is a positive side effect.

Tainting Complements Other Security Technologies

Intended for Defense in Depth Strategy

- **Added protection at O/S level for failures in:**
  - **Firewalls and intrusion detection systems**
  - **Application-level security mechanisms**
- **Unified trustworthiness policy at O/S level**
  - **Underlies and undergirds all installed application software**
  - **Reconciles different application-level policies and mechanisms**
  - **Covers office productivity suites, browsers, sandboxes, etc.**

Tainting Can Harden Existing Security Techniques

Technique	*Augmented with Tainting*
Generic Software Wrappers	Security *tailored* to a specific combination of user, resources, *and* data trustworthiness.
Sandboxing	Helps contain security-related bugs in application level sandboxes.
Firewalls	Added protection against malicious mobile code that gets past the firewall.
Role-Based Access Control	Support policies limiting trusted roles to trust-worthy files. Mark files modified by less trusted roles as untrustworthy.

Tainting Impact on O/S Trustworthiness

Potentially Introduces Two New Security Issues

Effects on Existing Applications	Trust-Based Attacks
Existing applications may not react well to constraints imposed based on trustworthiness. Non-robust applications especially vulnerable. E.g., not checking for errors returned by system calls.	Tainting creates new trust-based "channels of influence" on processes. E.g., maliciously constrain a process by illicitly affecting file trust-worthiness. Reduce vulnerability by applying least privilege principle to user profiles.

Low Degree of Vulnerability -- Situation-Dependent

Tainting and Traditional MLS and Integrity Policies

- **Bell-La Padula Confidentiality Model**
  - Tainting and classification are distinct concepts.
  - Tainting software could be adapted to support a military security policy.
- **Biba Integrity Model**
  - Tainting enforces a low water mark policy for subjects and objects.
  - Also provides security functionality by constraining active processes.

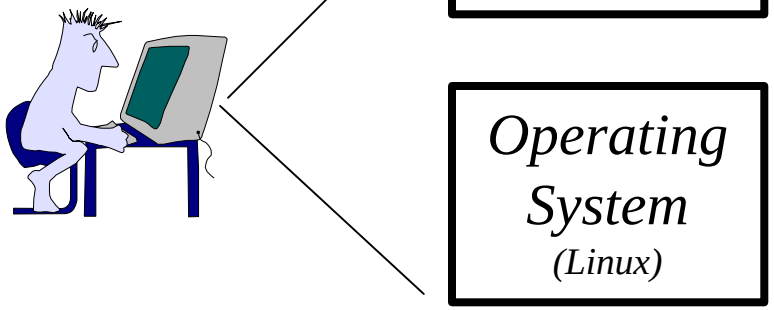
Existing Concepts Similar to Tainting

Concept	*Relation to Tainting*
LOMAC **(Fraser et al)**	LOMAC emphasizes the threat of malicious users, compromised root daemons, and viruses. Tainting focuses on the inattentive or unskilled user. LOMAC enforces low water mark policy for subjects only. Tainting covers subjects and objects.
Application Level Isolation **(Fayad et al)**	Application-level scheme focuses on *isolating* untrustworthiness. Tainting supports *dynamic interaction* between varying levels of trustworthiness.

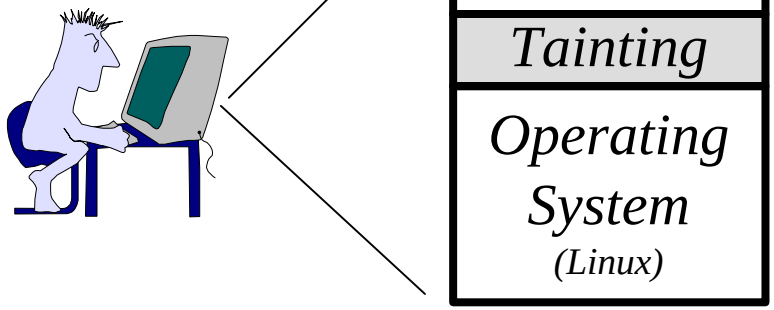
Implementation Approach

Overlay onto Existing Operating System
(No Source Code Modifications)

Before



After

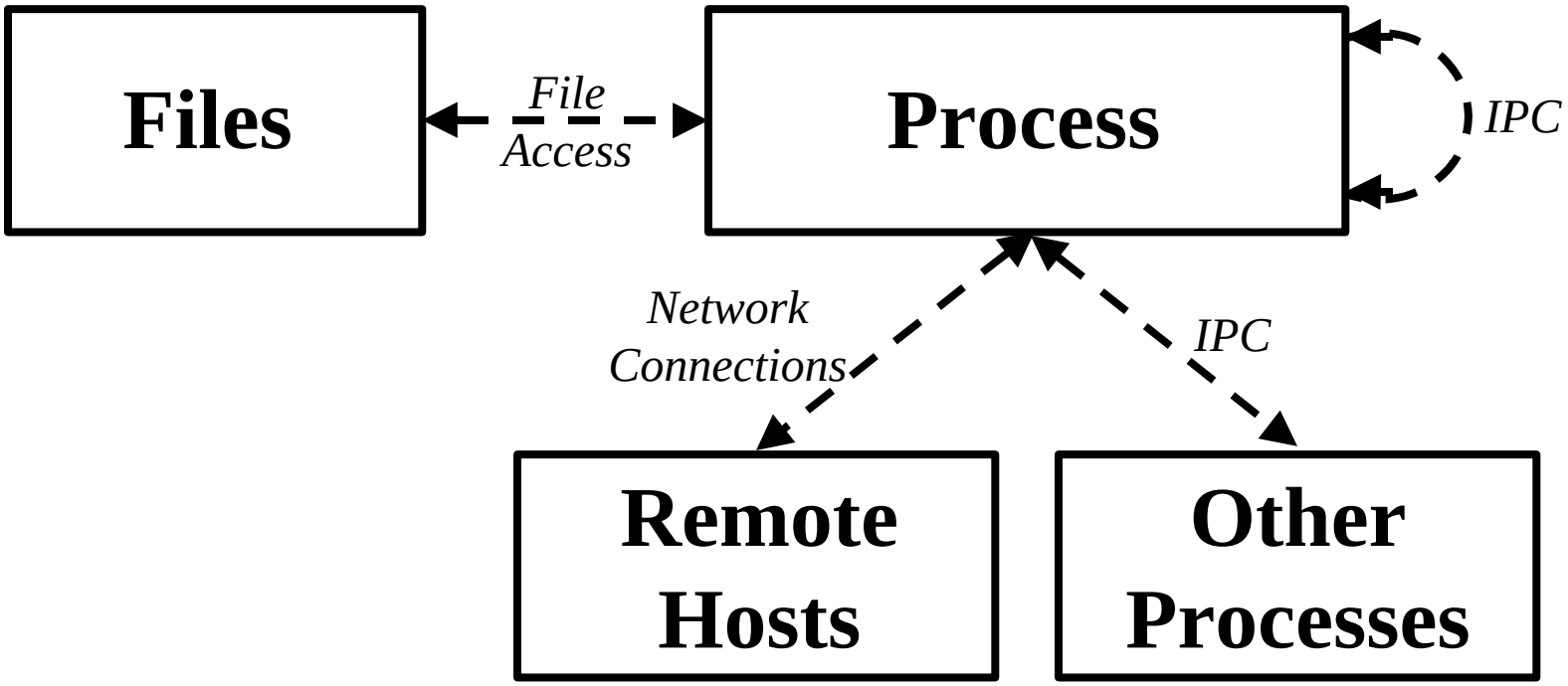


Use Loadable Kernel Modules

Implementation Approach

Setting the File Trust Attribute

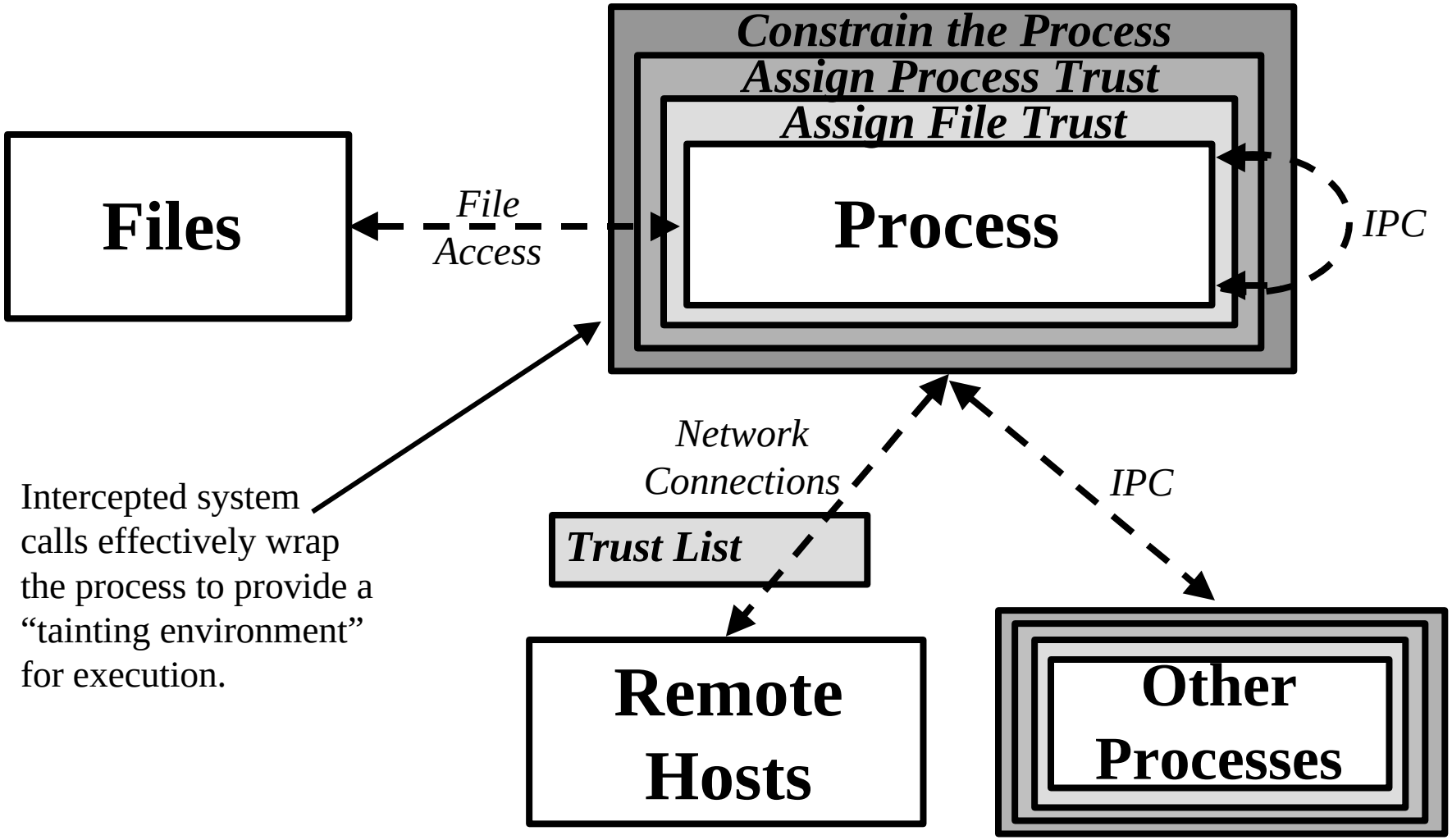
Requires Tracking the Conditions of File Creation/Modification



Must also Assign Trust to Processes and Network Connections

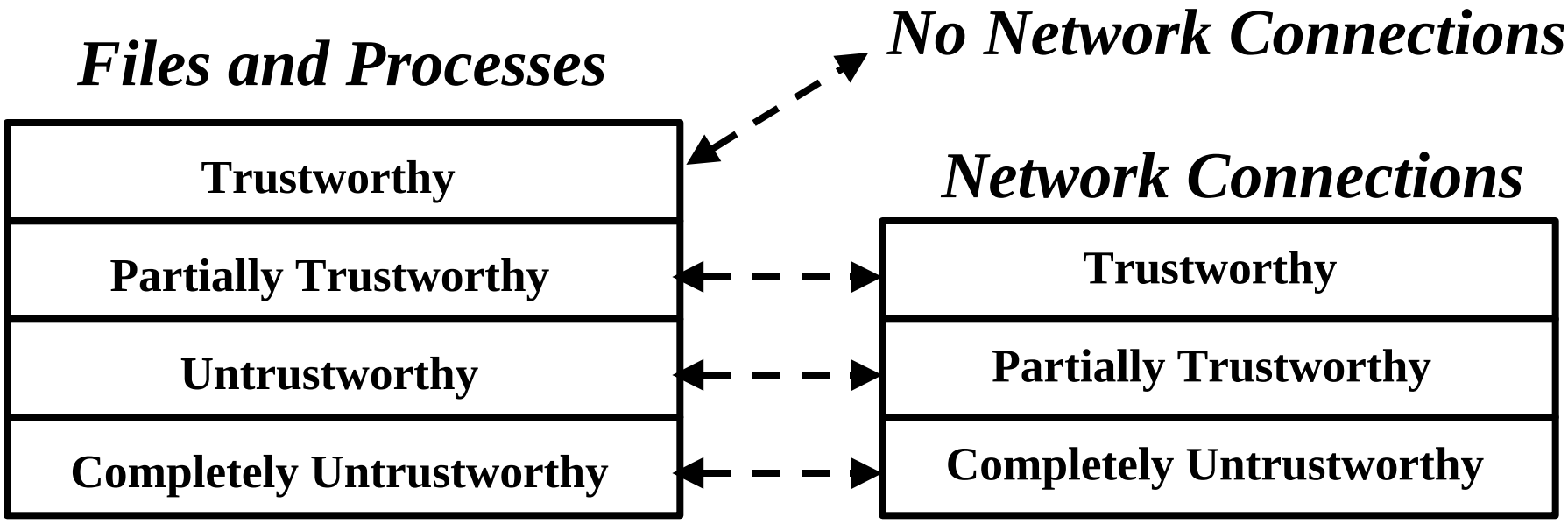
Implementation Approach

Assign/Enforce Trust by Intercepting System Calls



Example Trustworthiness Policy

Levels of Trustworthiness and Their Relationships



Example Constraints on Active Processes

Trustworthy	Any file created will by default have its "world" permissions cleared.
Partially Trustworthy	• Granted same access as "world" to any file or directory stored on the user's account. • Data cannot be written to a completely untrustworthy remote host. • Should not have *suid* capability.
Untrustworthy	Executes in "read-only" mode. Cannot create, delete, write, or change permissions of files. Cannot write to network connections. No *suid* capability.
Completely Untrustworthy	Process is automatically and immediately halted.

Example Scenario

Lighter gray shades represent lower trustworthiness.

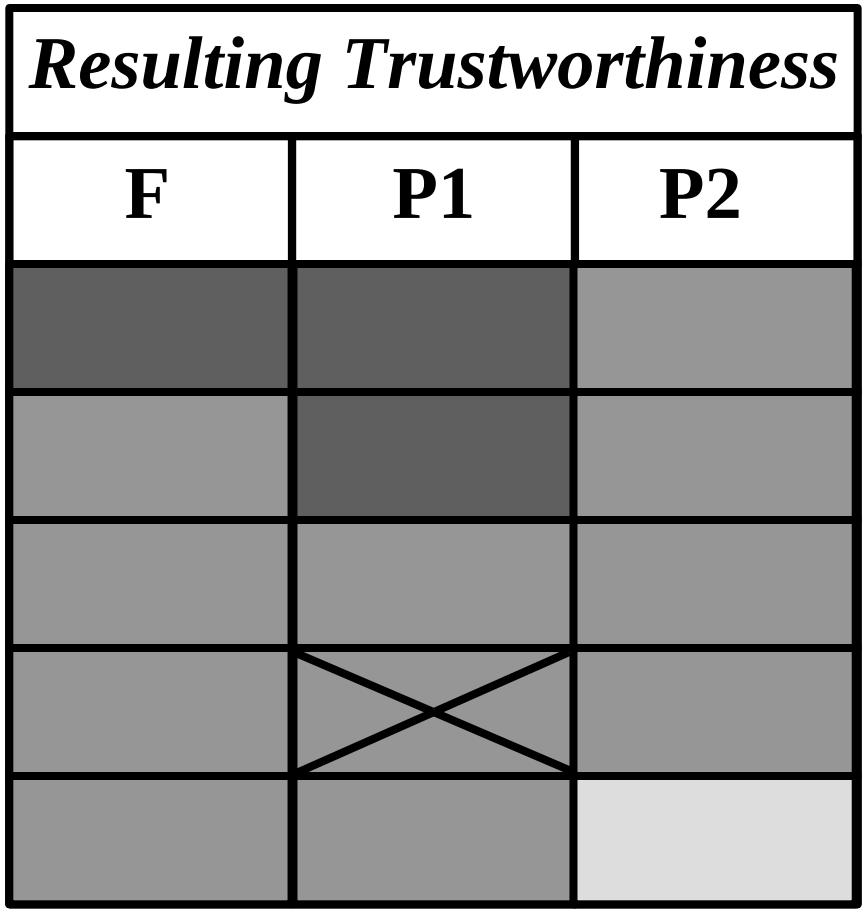
File *F* is created by trustworthy process *P1*.

Partially trustworthy process *P2* writes to *F*.

Trustworthy process *P1* reads F.

Partially trustworthy process *P1* attempts action *A* (not allowed at this trust level).

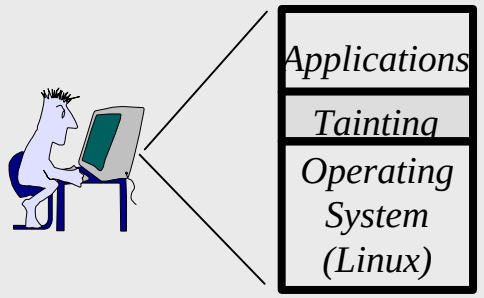
Partially trustworthy process *P2* opens a network connection with untrustworthy host *H*.



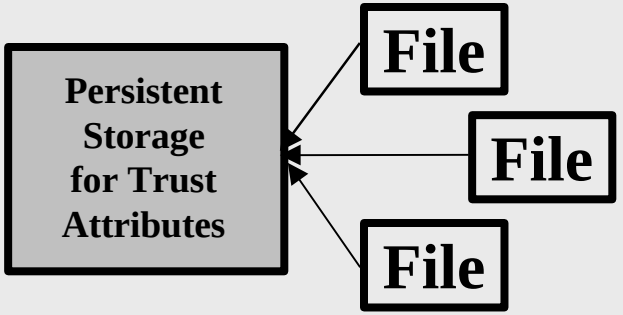
Protecting Tainting Functionality from Attack

Two Major Requirements in Linux

Protect Loadable Kernel Modules Used for Tainting



Securely Store the File Trust Attributes



Protecting Tainting Functionality from Attack

The Challenge in Linux

No clear way to distinguish between authorized and unauthorized root-level users.

YET

Root Can:

Load and unload kernel modules.

Access any file in the file system.

Result: Malicious Root can Undermine Tainting

Protecting Tainting Functionality from Attack

We Duck this Issue for Now!

- **Tainting provides additional protection for inattentive or unskilled users.**

- **Present concept not specifically intended to defend against malicious users and intruders.**

Conclusion

Tainting ...

- **Addresses deficiencies in general purpose operating systems used on the Internet.**
- **Gives added protection for inattentive or unskilled users.**
- **Is one element of a defense in depth strategy -- complements existing network security techniques.**
- **Future Work:**
  - **Complete Linux implementation**
  - **Port to Windows 9x/NT environment**
  - **Improve support for tailoring the trustworthiness policy**