Securing Enterprise Networks with Traffic Tainting

Anirudh Ramachandran
Nick Feamster
Yogesh Mundada
Mukarram bin Tariq

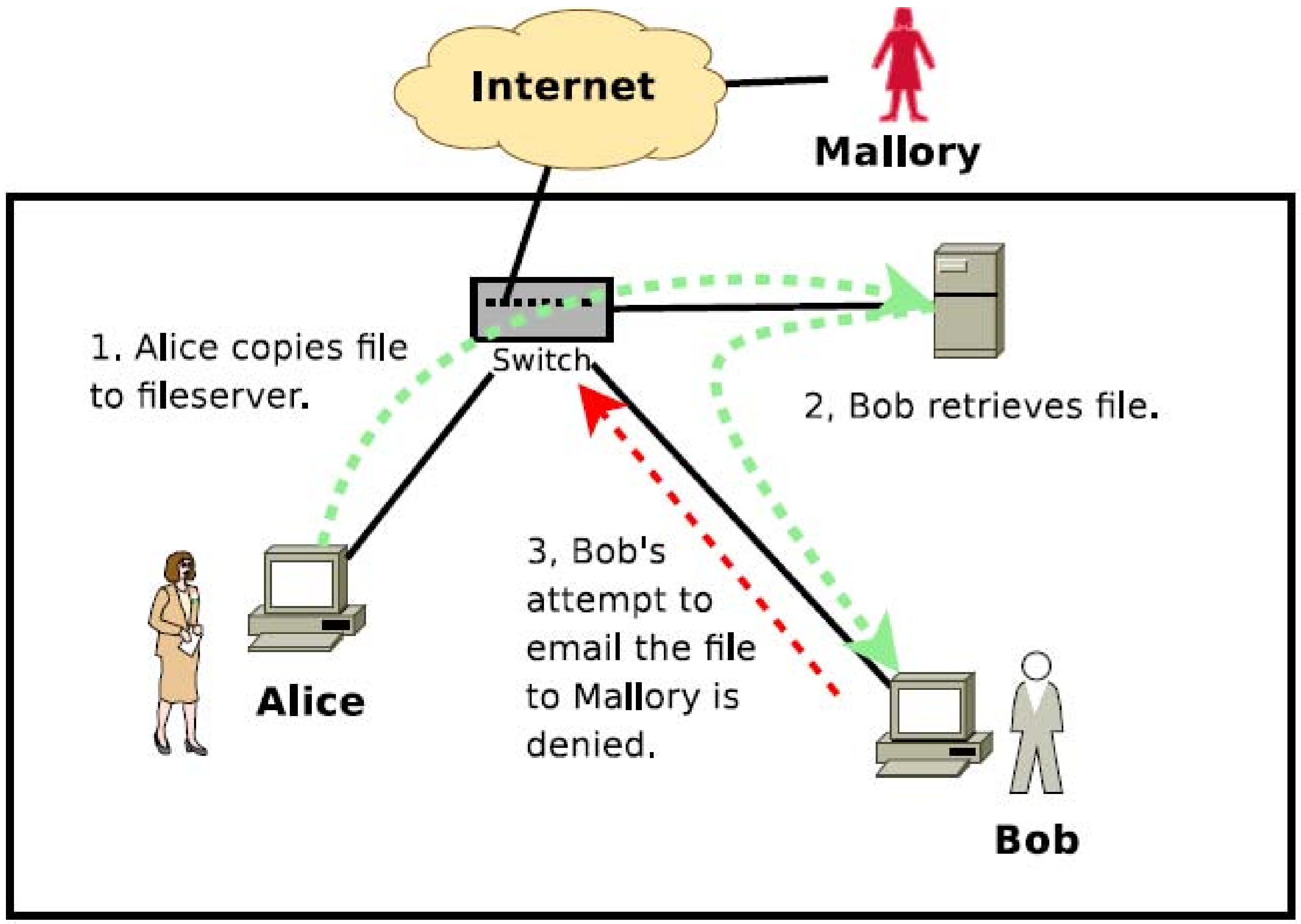
Motivation

- Main goal: Control the flow of traffic within an enterprise network

- Two scenarios
  - Preventing confidential documents from leaving the enterprise
    ~1/3 of companies victims of insider fraud

  - Controlling the spread of malware
    Damages from malware exceed $13 Billion

Scenario #1: Confidential Documents


Internet
Mallory
Switch
1. Alice copies file to fileserver.
2, Bob retrieves file.
3, Bob's attempt to email the file to Mallory is denied.
Alice
Bob

Existing Approaches

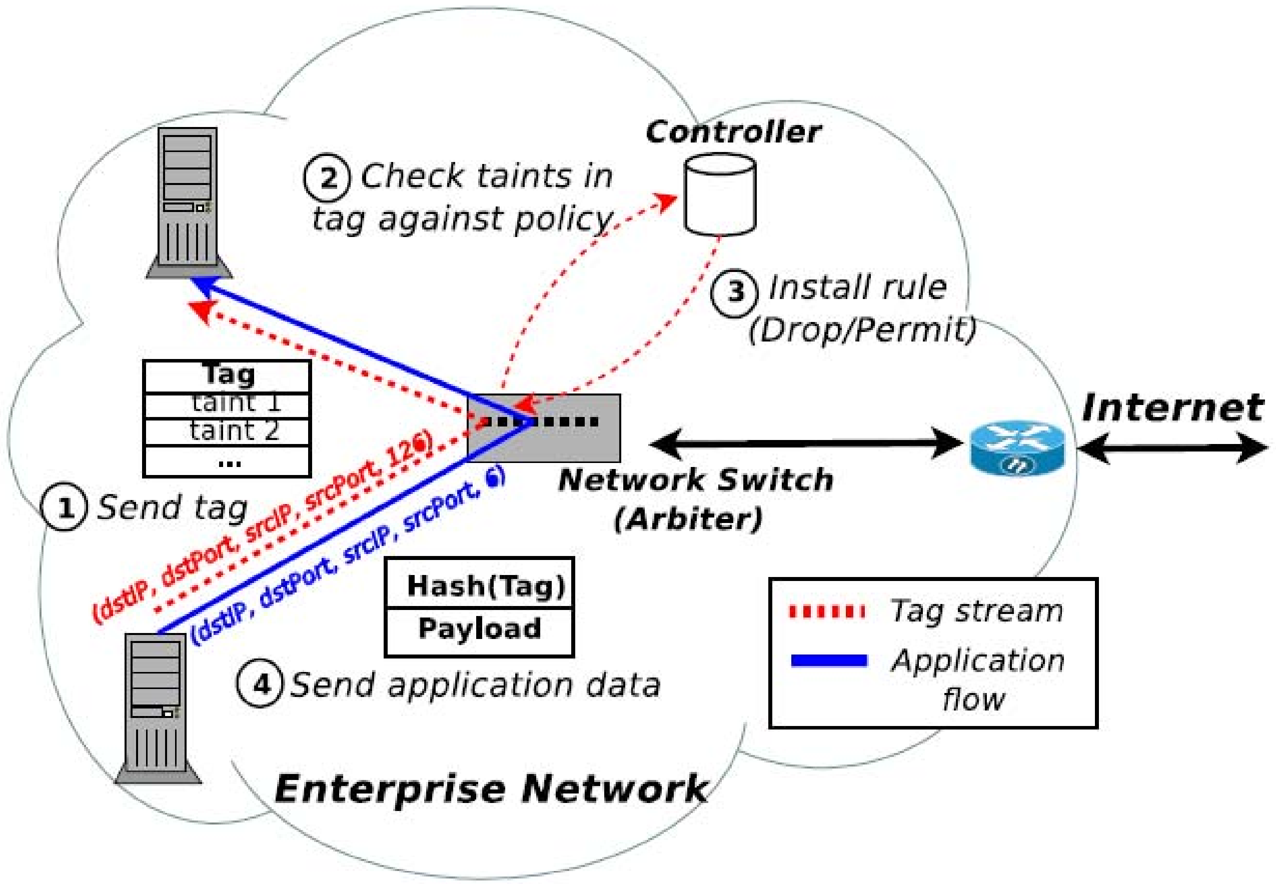
- Network firewalls
  - Inspecting content may require deep-packet inspection: difficult at high-speed
- Host firewalls
  - Must implement policies on host
- Restricted use (or separate machines)

Scenario #2: Malware Spreading

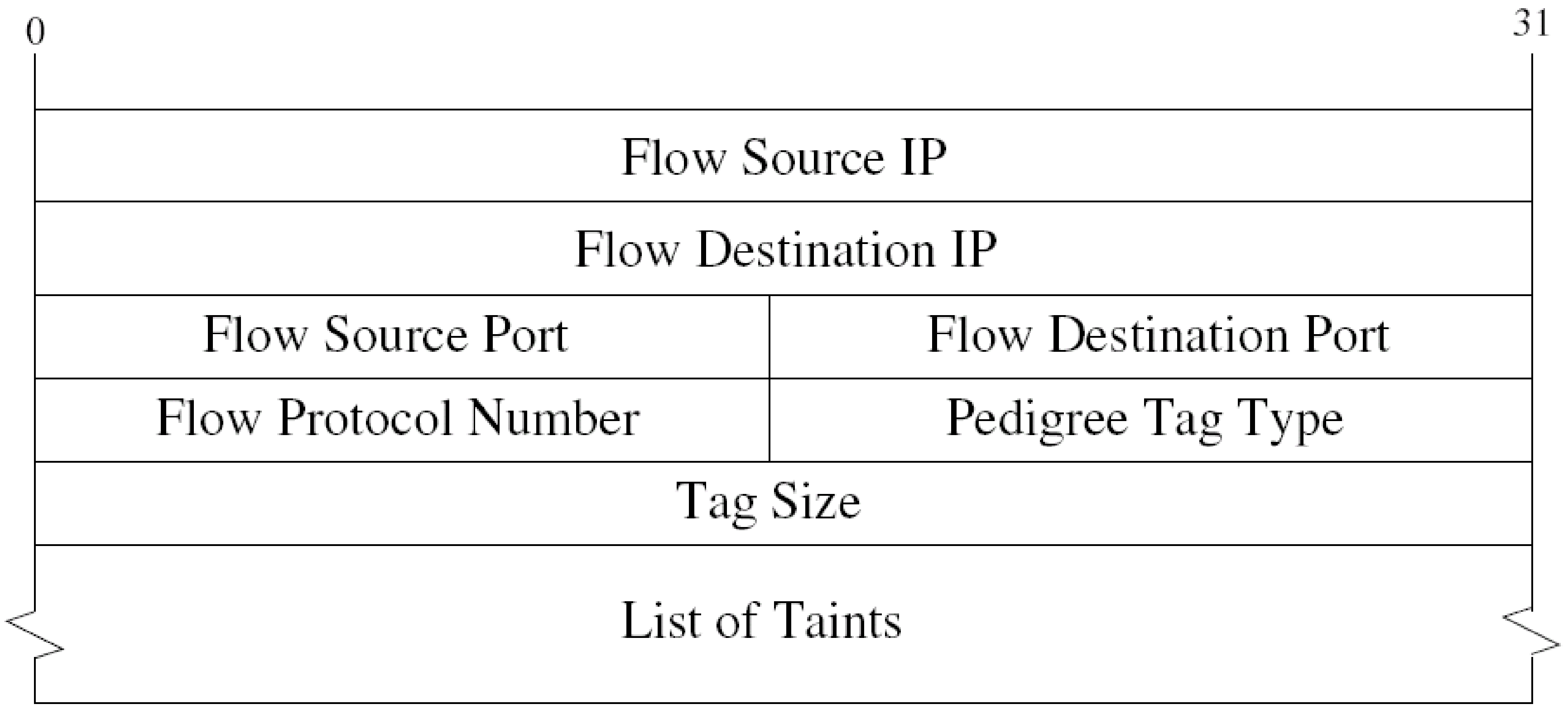
- Malware enters enterprise over thenetwork (*e.g.*, remote exploit, Web application), mobile device, etc.
- System administrators rely on virus scanners, host AV, etc.
  - Problem: Payloads may change, hard to keep AV up-to-date

Pedigree Design

- Trusted **tagging component** on host
- **Arbiter** on network switch



Tag Structure and Function

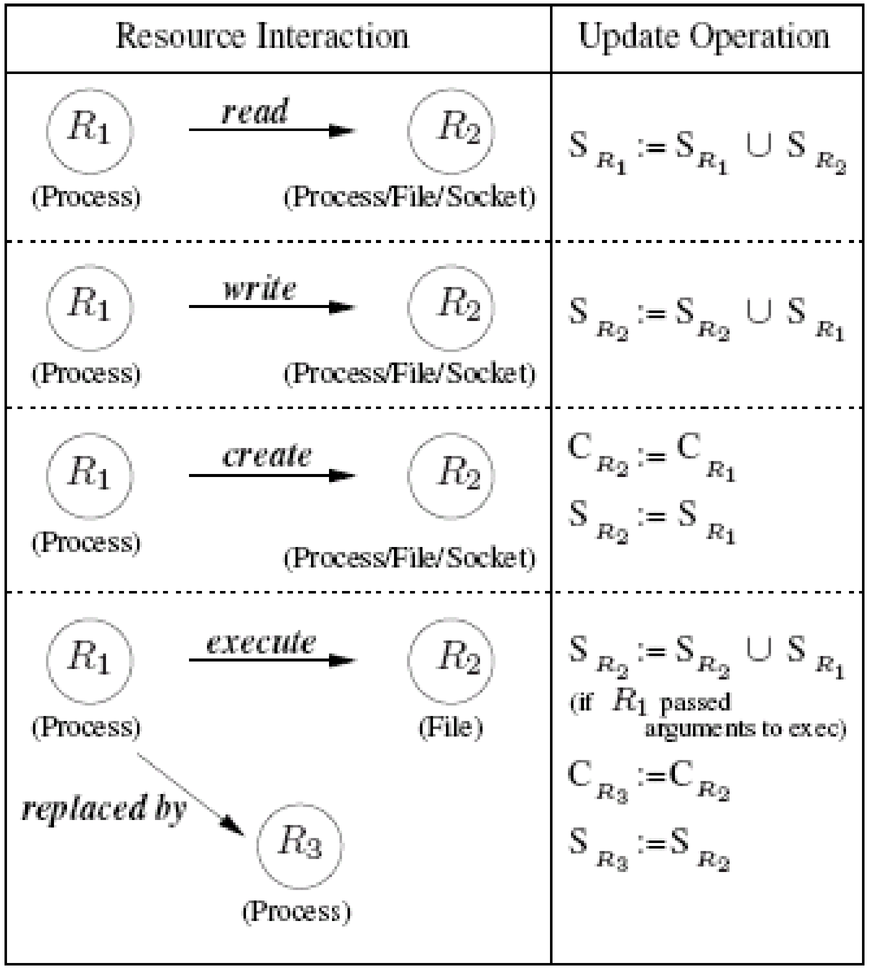


Design Decisions

- Specify and enforce policy in the network (not at the host).
- Taint files and processes.
- Implement tagger as a kernel module.
- Use a separate control channel to associate tags with network connections.

Transferring Taints

- System calls (e.g., `read`, `write`) intercepted, used to track taints
- Sets of taints stored in separate “tag store”
  - Mounted on separate device
- **Implementation:** Linux Security Modules



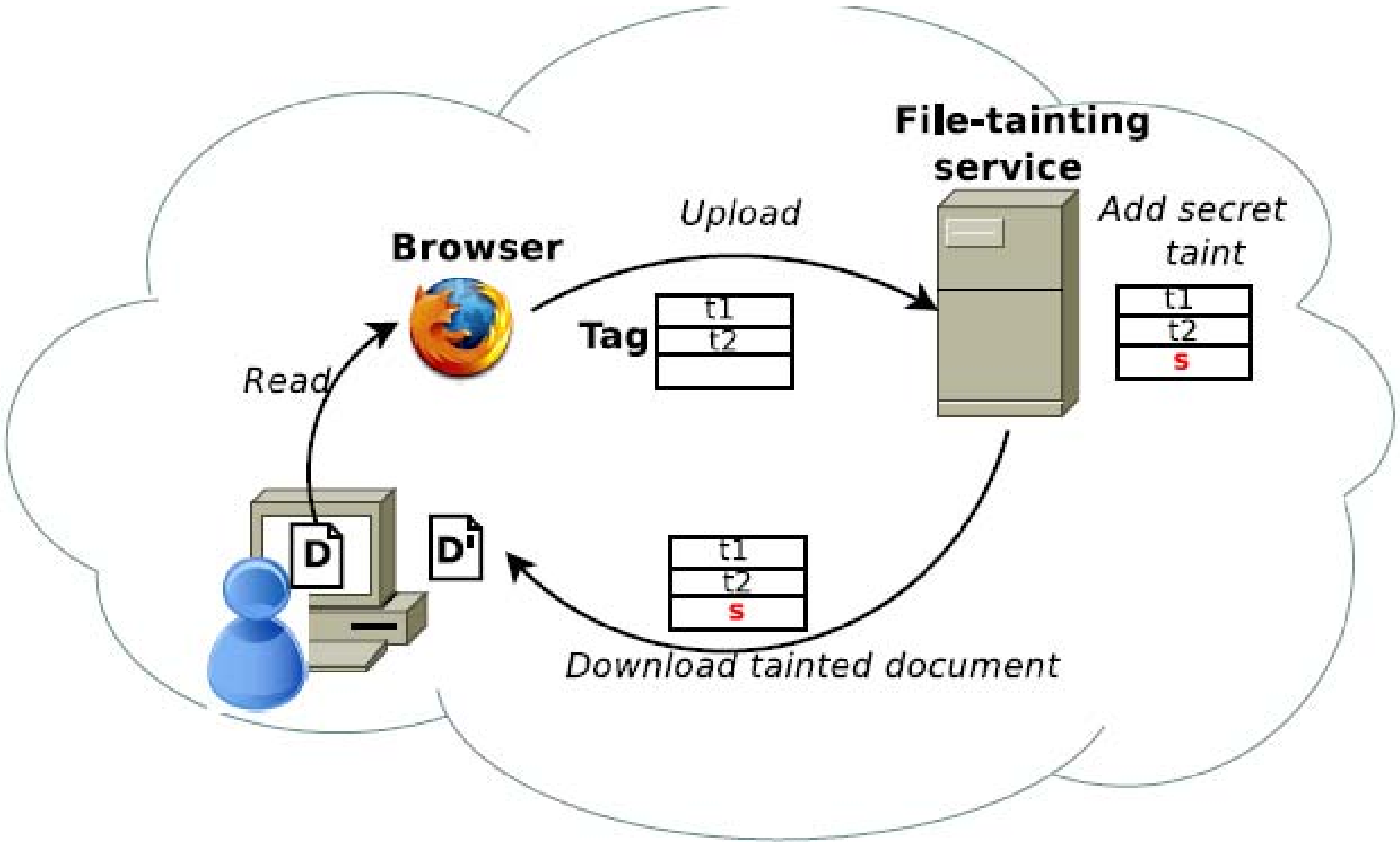
Resource Interaction	Update Operation
R_1 (Process) $\xrightarrow{\textit{read}}$ R_2 (Process/File/Socket)	$S_{R_1} := S_{R_1} \cup S_{R_2}$
R_1 (Process) $\xrightarrow{\textit{write}}$ R_2 (Process/File/Socket)	$S_{R_2} := S_{R_2} \cup S_{R_1}$
R_1 (Process) $\xrightarrow{\textit{create}}$ R_2 (Process/File/Socket)	$C_{R_2} := C_{R_1}$ $S_{R_2} := S_{R_1}$
R_1 (Process) $\xrightarrow{\textit{execute}}$ R_2 (File) R_1 (Process) $\xrightarrow{\textit{replaced by}}$ R_3 (Process)	$S_{R_2} := S_{R_2} \cup S_{R_1}$ (if R_1 passed arguments to exec) $C_{R_3} := C_{R_2}$ $S_{R_3} := S_{R_2}$

Assumptions and Trust Model

- Network elements don’t modify tags
- End host has a **trusted component**
  - Privileged process
  - Kernel module
  - Hypervisor
  - Outside the host

Scenario: Exfiltration Prevention

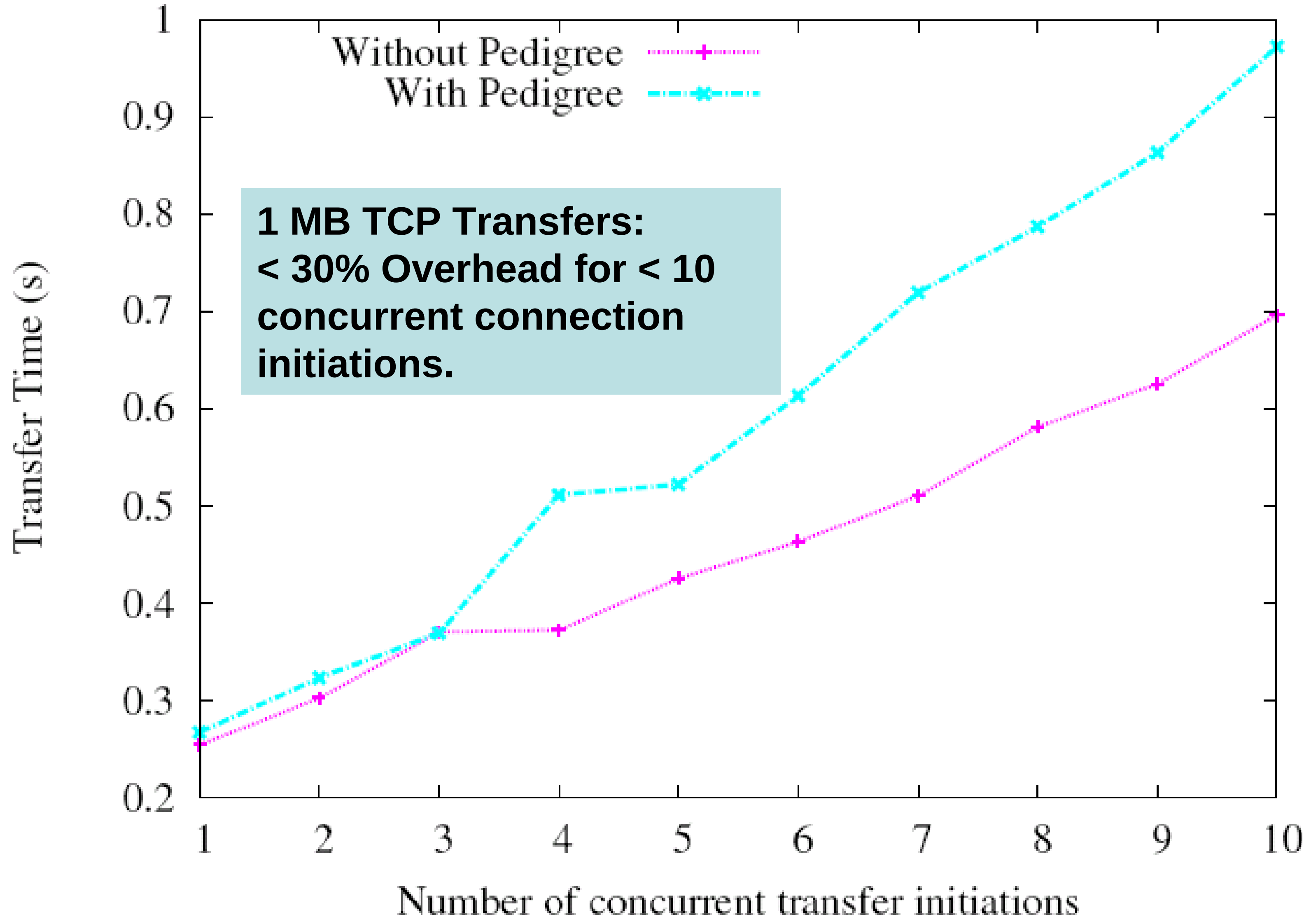
- Users can use a tainting service to assign security classes to files.



Concerns

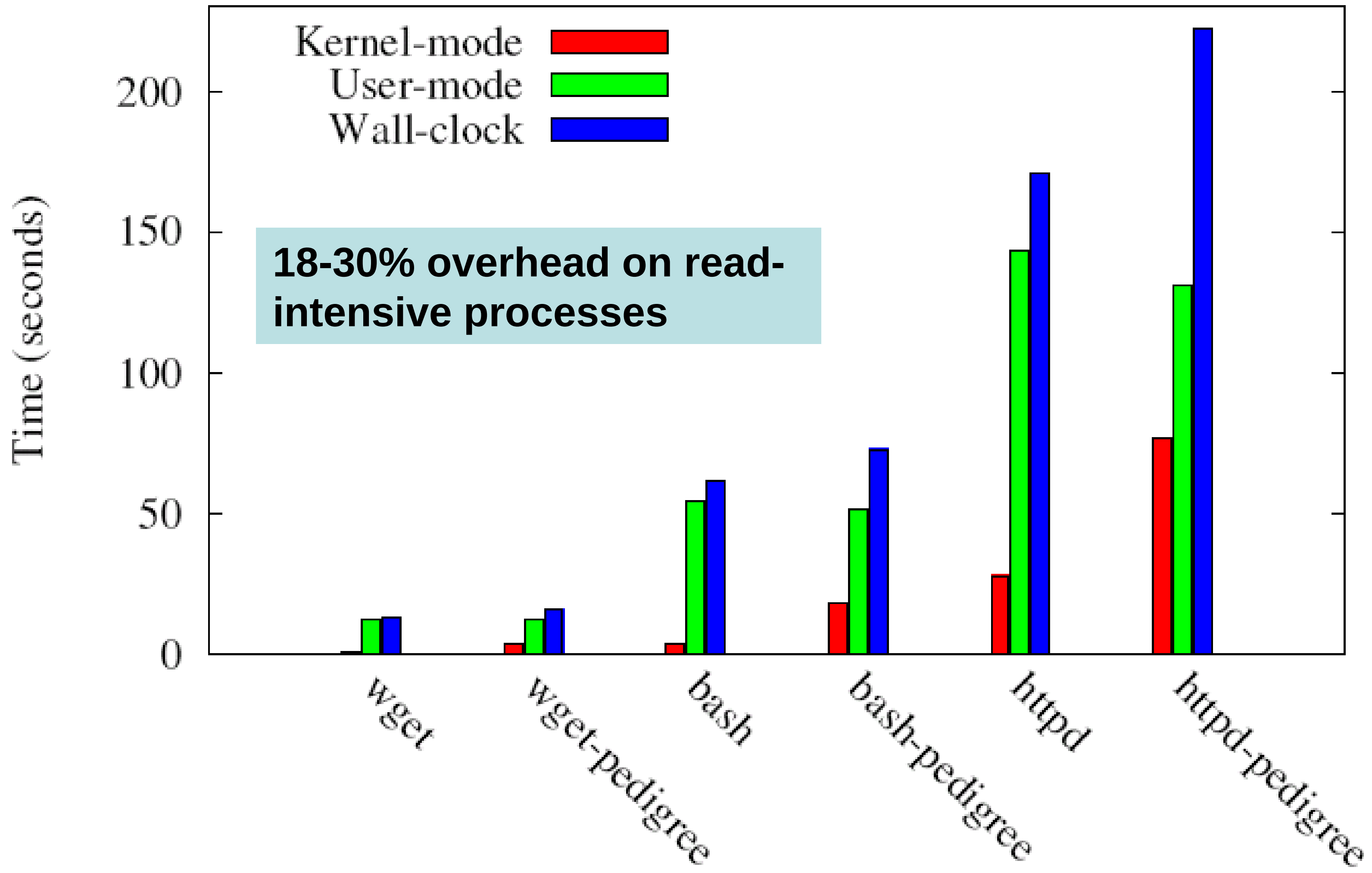
- Performance Overhead
  - Connection setup overhead
  - System call overhead
  - Storage overhead
- Overflow of taint set
  - Size of taint set could become quite large
- How to identify taints that reflect a certain class of traffic?

Connection Setup



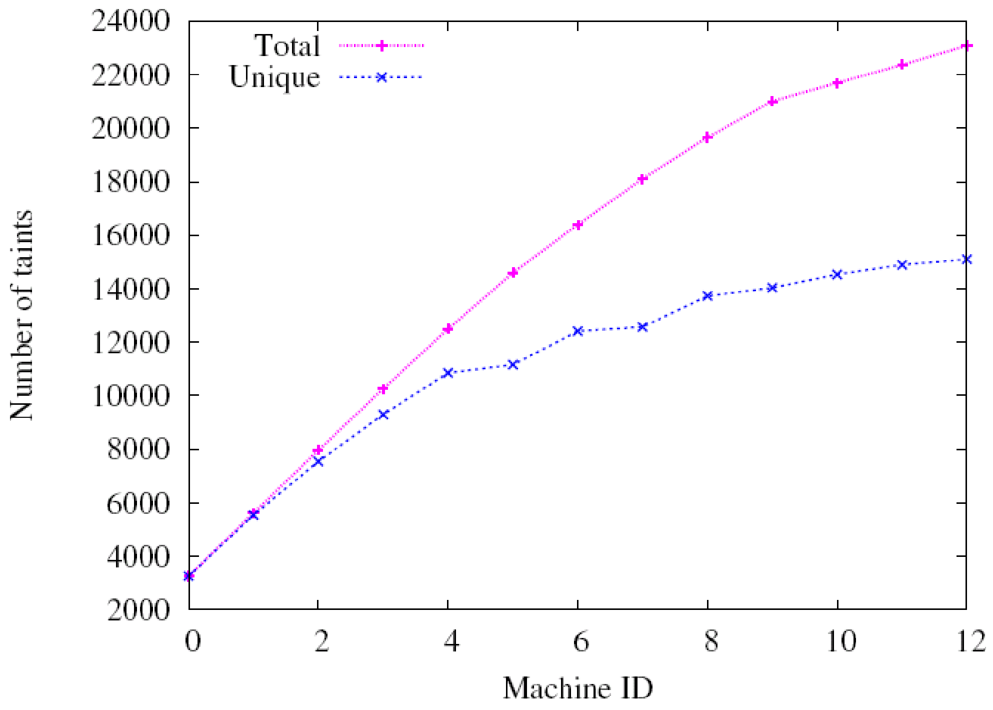
1 MB TCP Transfers:
< 30% Overhead for < 10 concurrent connection initiations.

System Call Overhead



18-30% overhead on read-intensive processes

How Many Taints?



- Our research group: 15,000 unique binaries
- Ways to deal with large sets of taints
  - Compression (Bloom filter)
  - Aggregation (Second-level taints)
  - “Bottom” security level

Summary

- Enterprises need to control information flow within their networks
  - Data leak/loss prevention
  - Malware containment

- **Idea:** Track information flow across processes. Implement control in network.