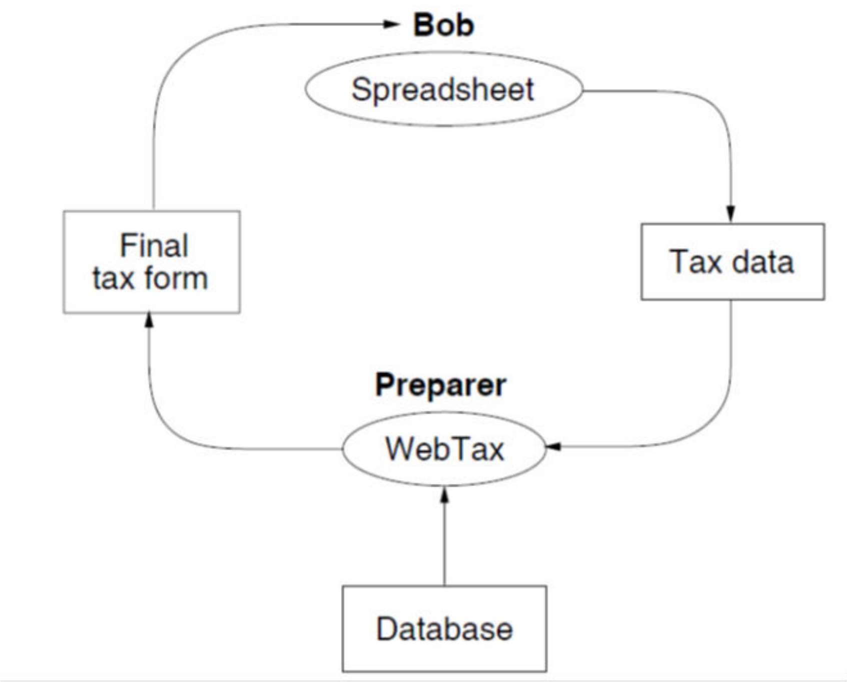
Decentralized Information Flow Control

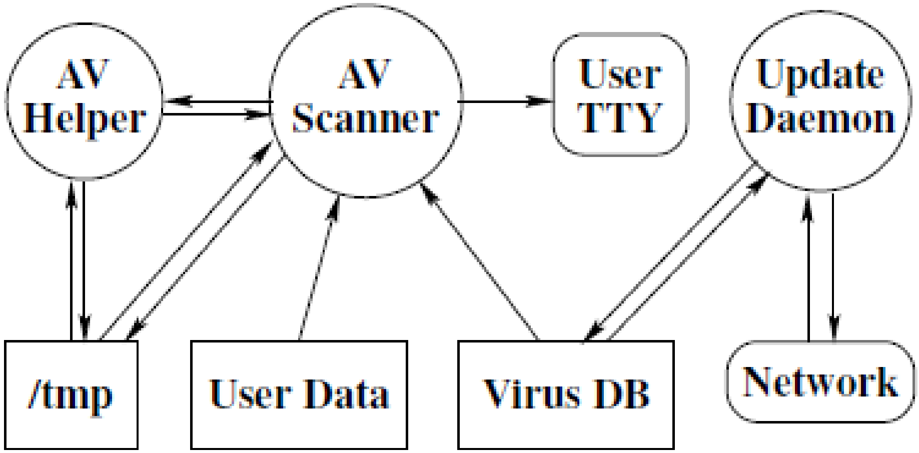
RK Shyamasundar
Tata Institute of Fundamental Research
Mumbai
shyam@tifr.res.in

Confidentiality Example (Contd)



- Principal Preparer – distributor of WebTax-may have privacy interests
- WebTax application computes the final tax form using a proprietary database, shown at the bottom (owned by Preparer).
  - this might, contain secret algorithms for minimizing tax payments.
  - Since this principal is the source of the WebTax software, it trusts the program not to distribute the proprietary database through malicious action,
  - However, the program might leak information because it contains bugs.

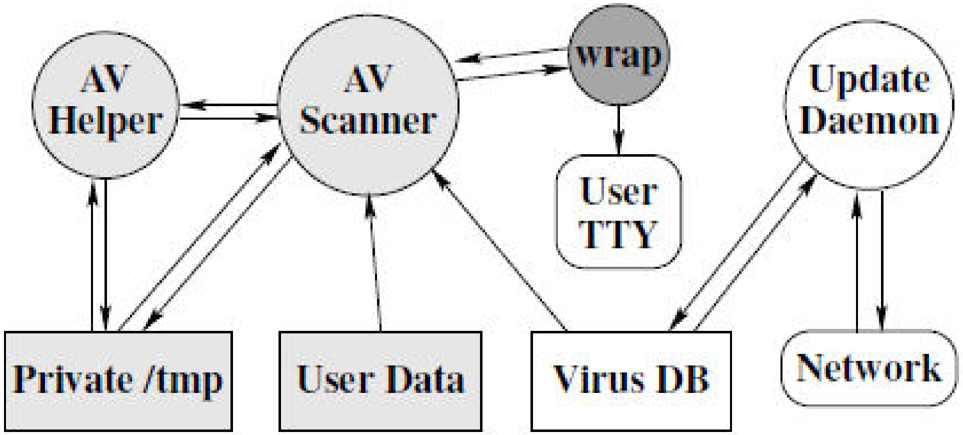
Enforcing data security policy while executing untrusted code



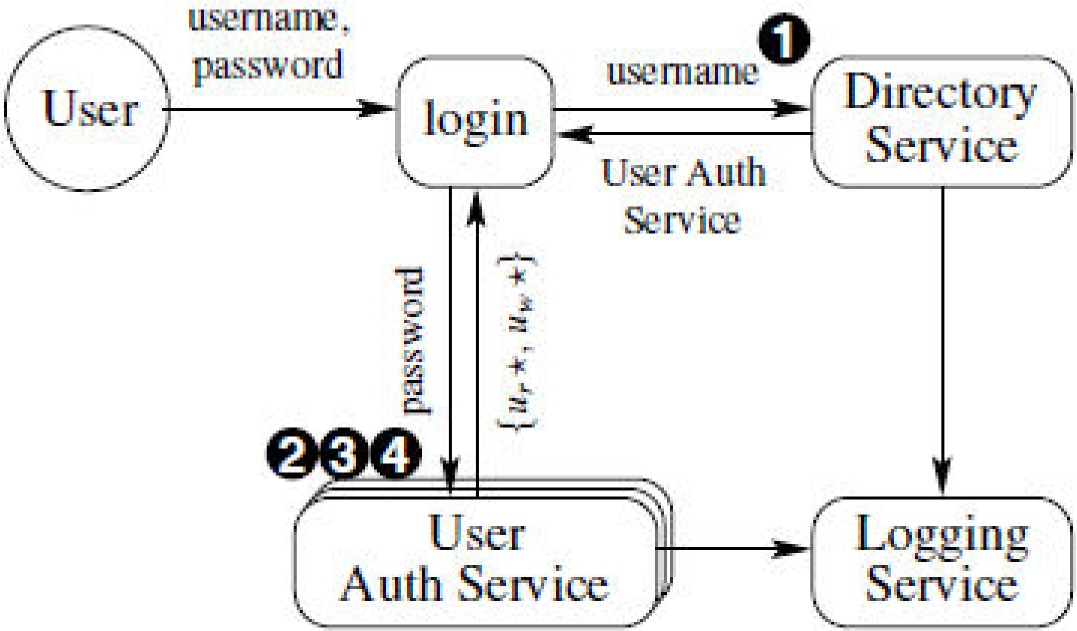
- Lightly Shaded – Confidential
- Unshaded – non-confidential
- Dark Shaded- Special privileges

to relay the scanner’s confidential output to the terminal.

- Circles: Processes
- Rectangles: Files/Dir
- Rounded Rect: Devices



Security Guarantees with OS like HiStar



Access Control

- Matrix Model
  - Evolution of access control in traditional OS
- Discretionary Access Control
  - controlling accesses to resources : traditional operating systems
  - Role based access control
- Mandatory Access Control
  - the control of information flow between distributed nodes on a system wide basis rather than only individual basis like discretionary control
- Information Flow Control
  - how information is disseminated or propagated from one object to another
  - security classes of all entities must be specified clearly and class of an entity never changes after it has been created
  - All permissible information flow paths among them are regulated using unambiguous security rules
  - Distributed Information Control
    - Language Based
    - OS based – distribution across nodes

Access Matrix model (AMM)

- The access matrix model provides a framework for describing discretionary access control
- First proposed by Lampson for the protection of resources within the context of operating systems, and later refined by Graham and Denning, the model was subsequently formalized by Harrison, Ruzzo, and Ullman (HRU model), who developed the access control model proposed by Lampson to the goal of analyzing the complexity of determining an access control policy

- The original model is called access matrix since the authorization state, meaning the authorizations holding at a given time in the system, is represented as a matrix

- The matrix therefore gives an abstract representation of **protection** in systems

mac-ifip.pptx

Main issues with AMM

Confinement problem:

- How to determine whether there is any mechanism by which a subject authorized to access an object may leak information contained in that object to some other subjects not authorized to access that object.

Another disadvantage:

- No semantics of information in the objects are considered;
  - thus the security sensitivity of an object is hardly expressed by that model.

Discretionary Access Control

- Discretionary Access Control (DAC)
- Discretionary policies enforce access control on the basis of the identity of the requestors and explicit access rules that establish who can, or cannot, execute which actions on which resources
- They are called discretionary as users can be given the ability of passing on their privileges to other users, where granting and revocation of privileges is regulated by an administrative policy

DAC: Vunerabilities

- In defining the basic concepts of discretionary policies, we have referred to access requests on objects submitted by users, which are then checked against the users' authorizations
- Although it is true that each request is originated because of some user's actions, a more precise examination of the access control problem shows the utility of separating users from subjects

- Users are passive entities for whom authorizations can be specified and who can connect to the system
- Once connected to the system, users originate processes (subjects) that execute on their behalf and, accordingly, submit requests to the system
- Discretionary policies ignore this distinction and evaluate all requests submitted by a process running on behalf of some user against the authorizations of the user
- This aspect makes discretionary policies vulnerable from processes executing malicious programs exploiting the authorizations of the user on behalf of whom they are executing

DAC: Vulnerabilities

- In particular, the access control system can be bypassed by Trojan Horses embedded in programs
- A Trojan Horse is a computer program with an apparently or actually useful function, which contains additional hidden functions that exploit the legitimate authorizations of the invoking process
- A Trojan Horse can improperly use any authorizations of the invoking user, for example, it could even delete all files of the user
- This vulnerability of Trojan Horses, together with the fact that discretionary policies do not enforce any control on the flow of information once this information is acquired by a process, makes it possible for processes **to leak information to users not allowed to read it**
- All this can happen without the cognizance of the data administrator/owner, and despite the fact that each single access request is controlled against the authorizations

Discretionary and Mandatory Access Control

- Discretionary Access Control (DAC)
- Discretionary policies enforce access control on the basis of the identity of the requestors and explicit access rules that establish who can, or cannot, execute which actions on which resources
- They are called discretionary as users can be given the ability of passing on their privileges to other users, where granting and revocation of privileges is regulated by an administrative policy

- Mandatory Access Control (MAC)
- Mandatory access control refers to a type of access control by which the operating system constrains the ability of a subject or initiator to access or generally perform some sort of operation on an object or target.

- Need: In the context of networked distributed systems, it is necessary to broaden the scope to include the control of information flow between distributed nodes on a system wide basis rather than on an individual basis as in discretionary control

MAC vs DAC

- Mandatory access control, this security policy is centrally controlled by a security policy administrator; users do not have the ability to override the policy and, for example, grant access to files that would otherwise be restricted.

- Discretionary access control (DAC), which also governs the ability of subjects to access objects, allows users the ability to make policy decisions and/or assign security attributes.

Security Classifications

- In multilevel mandatory policies, an access class is assigned to each object and subject
  - Access class is one element of a partially ordered set of classes
- The partial order is defined by a dominance relationship denoted ≥
- In the most general case, the set of access classes can simply be any set of labels that together with the dominance relationship defined on them form a POSET (partially ordered set)

Information Flow Control

- System entities are partitioned into security classes
- The security classes of all entities must be specified explictly and the class of an entity seldom changes after it has been created(changes sometimes made by the system administration)

- **Information Flow control is concerned with how information is disseminated or propagated from one object to another**.

Lattice Model

- Lattice: consists of a finite partially ordered set together with a least upper bound and greatest lower bound operator on the set.
- Information is permitted to flow from a lower class to upper class.
- (Details – See earlier lectures)

Protecting Privacy in a Distributed Network

- Downloading of untrusted code are particularly in need of a better security model
- E.g., Java supports downloading of code from remote sites,
  - possibility that the downloaded code will transfer confidential data to those sites.
  - Java attempts to prevent these transfers by using its compartmental sandbox security model,
    - It largely prevents applications from sharing data.
    - Different data manipulated by an application have different security requirements, but a compartmental model restricts all data equally.

Decentralized Label Model

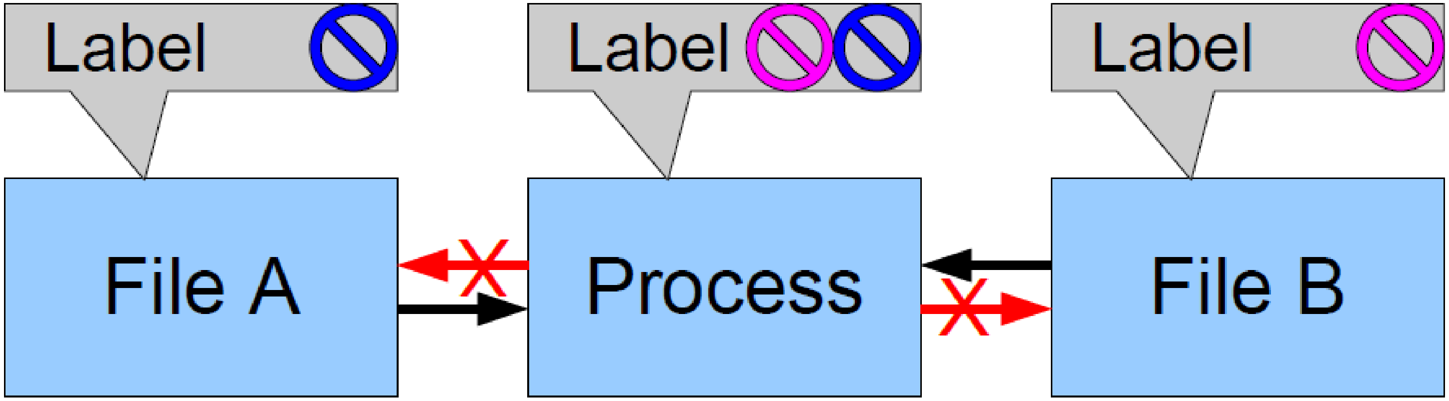
Myers and Liskov (2000)

- addresses the weaknesses of earlier approaches to the protection of confidentiality in a system containing untrusted code or users, even in situations of mutual distrust.
- allows users to control the flow of their information without imposing the rigid constraints of a traditional MLS
- It defines a set of rules that programs must follow in order to avoid leaks of private information
- Protects confidentiality for users and groups rather than for a monolithic organization
- Introduces a richer notion of declassification.
  - In the earlier models it was done by a trusted subject; in this model principals can declassify their own data

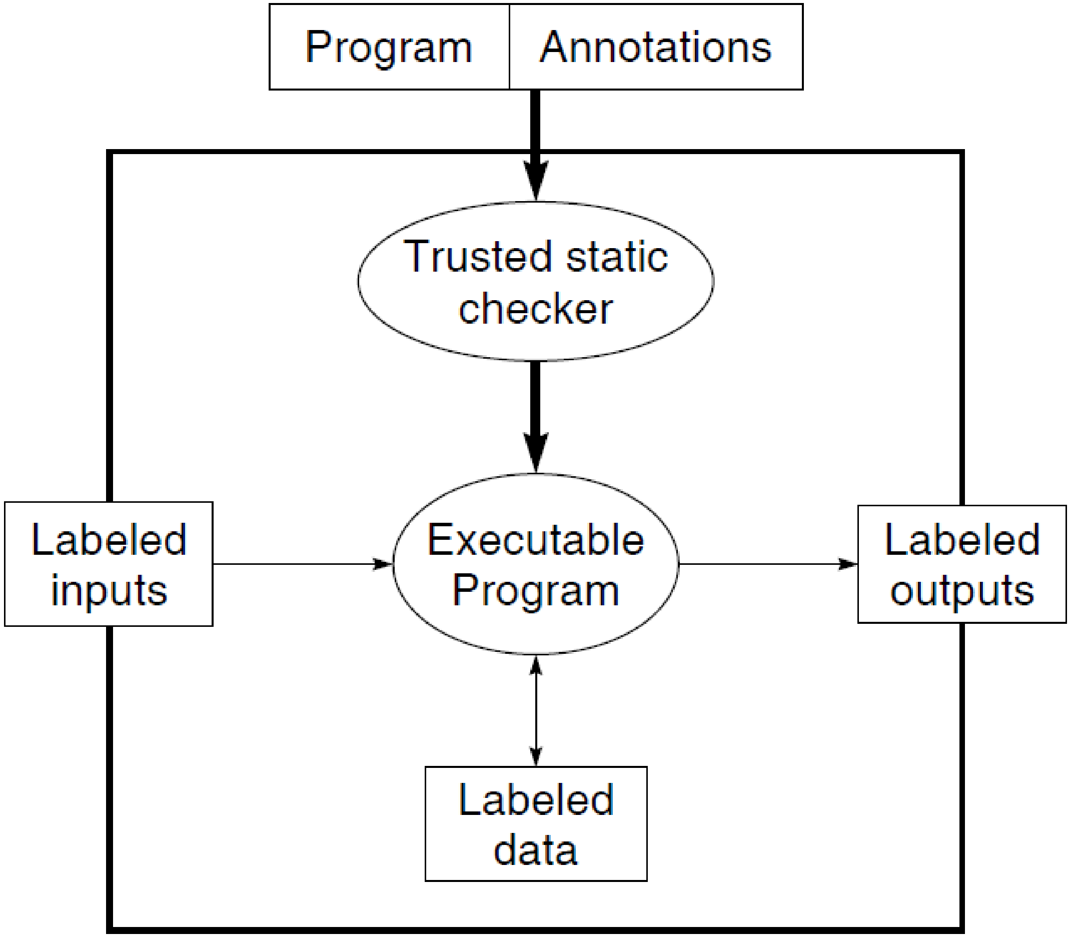
Labels control information flow

🟦 Color is category of data (e.g. my files)

🛇 Blue data can flow only to other blue objects



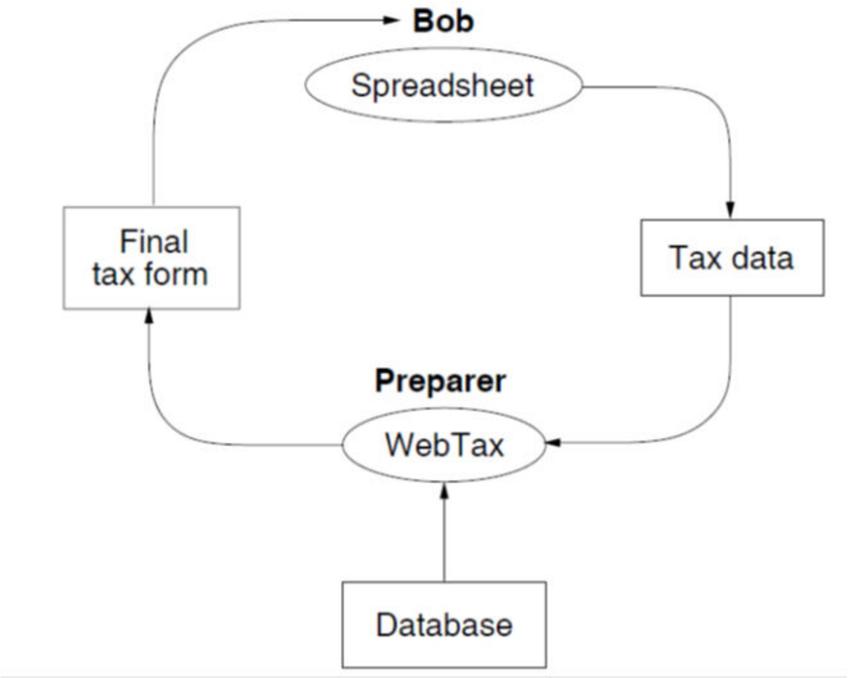
Trusted Execution Model



Confidentiality Example

- Bob is preparing his tax form using both a spreadsheet program (trusted and he can give full authority) and a piece of software called "WebTax" which he doesn't not trust WebTax to respect his privacy.
- Bob wants to prepare his final tax form using WebTax, by transmitting his tax data from the spreadsheet to WebTax and receive a final tax form as a result, while being protected against WebTax leaking his tax information.

Confidentiality Example (Contd)



- Principal Preparer – distributor of WebTax-may have privacy interests
- WebTax application computes the final tax form using a proprietary database, shown at the bottom (owned by Preparer).
  - this might, contain secret algorithms for minimizing tax payments.
  - Since this principal is the source of the WebTax software, it trusts the program not to distribute the proprietary database through malicious action,
  - However, the program might leak information because it contains bugs.

Can it preserve confidentiality?

- Difficult to prevent some information about the database contents from leaking back to Bob, particularly if Bob makes a large number of requests and then carefully analyze the resulting tax forms.
  - This leak is not a practical problem if Preparer can charge Bob a per-form fee that exceeds the value of the information Bob obtains through each form.
- Ensuring Confidentiality by preparer

1. It needs protection against accidental or malicious release of information from the database by paths other than through the final tax form.

2. It needs the ability to sign off on the final tax form, confirming that the information leaked in the final tax form is sufficiently small or scrambled by computation that the tax form may be released to Bob.

- Bob and Preparer do need to trust the execution platform
- Assuming the platform is trusted Bob must check webtax code and ensure that it does not leak - DIFFICULT

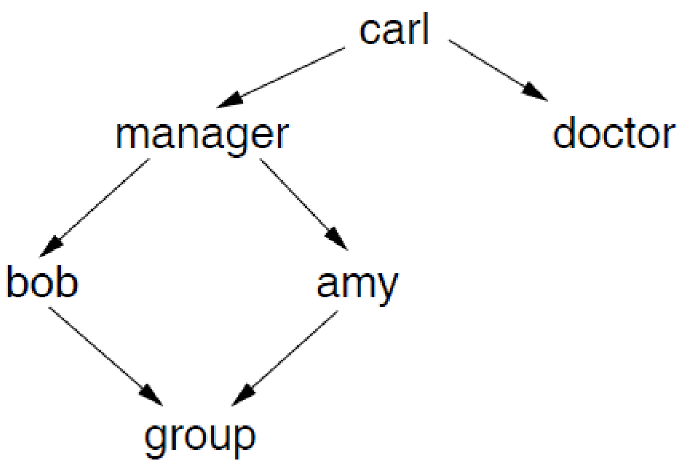
Label Model: **allows Bob and**

Preparer to inspect the Webtax program efficiently and simply to determine whether it violates their security requirements.

Label Model

- Principals: users and other authority entities such as groups or roles.
  - users/groups in Unix be represented by principals
- p acts for q is written formally as p≥ q.
  - process has authority to act on behalf of some set of principals
  - The acts for relation is reflexive and transitive, defining a hierarchy or partial order of principals.
  - principal hierarchy changes over time, revocation of acts-for relations is assumed to occur infrequently

Label Model-Principle Hierarchy



- allows Amy and Bob to read data readable by the group and to control data controlled by the group.
- A principal manager representing Bob and Amy's manager is able to act for both amy and bob.
- This principal is one of the roles that a third user, carl is currently enabled to fulfill;
- carl also has a separate role as a doctor.
- carl can use his roles to prevent accidental leakage of information between data stores associated with his different jobs.

Label Model

- The acts-for relation permits delegation of all of a principal’s powers, or none.
- However, the principle of least privilege suggests that it is desirable to separate out the various powers that may be given by one principal to another.
  - Eg., the acts-for relation can be separated into declassifies-for and reads-for relations between principals, as well as other relations less directly connected to information flow control.

Labels

- Principals express their privacy concerns by using labels to annotate programs & data
- label -- set of components that express privacy requirements stated by various principals.
- A component has two parts, an owner and a set of readers, and is written in the form owner: readers.
- Purpose of a label component is to protect privacy of the owner of component.
- The readers of a component are the principals that this component permits to read the data.
- Thus, the owner is a source of data, and the readers are possible destinations for the data.
- Principals not listed as readers are not permitted to read the data.
- a label component is called a policy for use of the data (not be confused with its use for the high-level specification of information flows allowed within the system as a whole (e.g., non-interference policies))
- Policies in decentralized labels, by contrast, apply only to a single data value

Label Model

- L = {o1 : r1, r2; o2 : r2, r3}.
- o1, o2, r1, r2 denote principals.
- Semicolons separate two policies (components) within the label L.
- Owners of these policies are o1 and o2, and the reader sets of the policies are {r1, r2} and {r2, r3}, respectively.
- label structure allows each owner to specify an independent flow policy, and thus retains control over the dissemination of its data.
- Code running with the authority of an owner can modify a policy with that owner;
  - in particular, the program can declassify that data by adding additional readers. Since declassification applies on a per-owner basis, no centralized declassification process is needed, as it is in systems that lack ownership labeling.

Label Model

- Labels are used to control information flow within programs by annotating program variables with labels.
- The label of a variable controls how the data stored within that variable can be disseminated.
- The key to protecting confidentiality is to ensure that as data flows through the system during computation, its labels only become more restrictive: the labels have more owners, or particular owners allow fewer readers.
- If the contents of one variable affect the contents of another variable, there is an information flow from the first variable to the second, and therefore, the label of the second variable must be at least as restrictive as the label of the first.
- The condition can be enforced at compile time by type checking, as long as the label of each variable is known at compile time.
- In other words, the label of a variable cannot change at run time, and the label of the data contained within a variable is always the same as the label of the variable itself.
- If the label of a variable could change, the label would need to be stored and checked at run time, creating an additional information channel and leading to significant performance overhead.
- Immutable variable labels might seem to be a limitation, but there are mechanisms that would restore expressiveness.

Label Model

- When a value is read from a variable x, the apparent label of the value is the label of x; whatever label that value had at the time it was written to x is no longer known when it is read.
  - In other words, writing a value to a variable is a relabeling, and is allowed only when the label of the variable is at least as restrictive as the apparent label of the value being written.

Information leak:

- Giving private data to an untrusted program does not create an information leak—even if that program runs with the authority of another principal—as long as that program obeys all of the label rules described.

- Information can be leaked only when it leaves the system through an output channel, so output channels are labeled to prevent leaks.
- Information can enter the system through an input channel, which also is labeled to prevent leaks of data that enters.
- It is not necessarily an information leak for a process to manipulate data even though no principal in its authority has the right to read it, because all the process can do is write the data to a variable or a channel with a label that is at least as restrictive as the data's label.

Incremental Relabeling

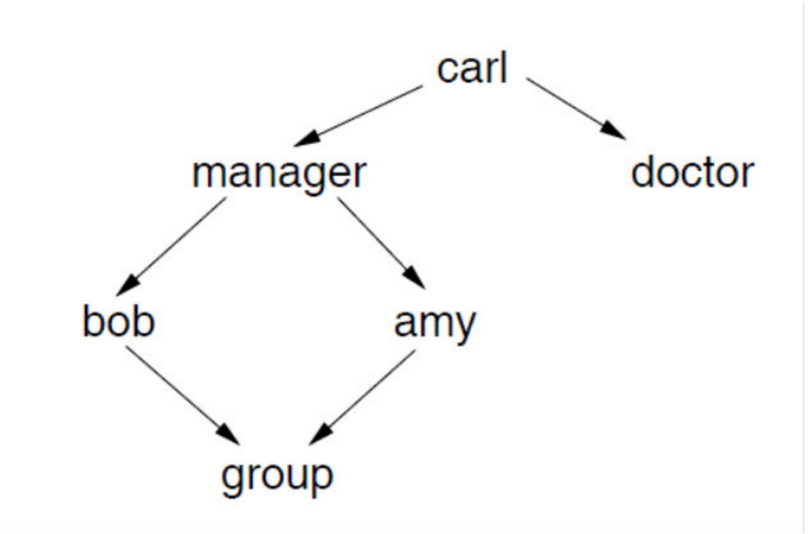
- A value may be assigned to a variable only if the relabeling that occurs at this point is a restriction, a relabeling in which the set of readers permitted by the new label is in all contexts a subset of the readers permitted by the original.
- For example, if the two labels differ only in a single policy, and the readers of that policy in the label of the variable are a subset of the readers of that policy in the label of the value.
  - Eg, a relabeling from {bob : amy, carl} to {bob : amy} is a restriction because the set of readers allowed by bob becomes smaller.
- If a relabeling is a restriction, it is considered to be **safe.**

- **Clearly, a change to a label in which a reader r is removed cannot make the changed policy any less restrictive, and therefore this change is safe.**
  - **Note that the removal of a reader does not necessarily make the policy more restrictive if there is another reader in the policy for which r acts.**

- Remove a reader. It is safe to remove a reader from some policy in the label, as just described.
- Add a policy. It is safe to add a new policy to a label; since all previous policies are still enforced, the label cannot become less restrictive.
- Add a reader. It is safe to add a reader r' to a policy if the policy already allows a reader r that r' acts for. This change is safe because if r'acts for r, it is already effectively considered to be a reader: r'has all of the privileges of r anyway.
- Replace an owner. It is safe to replace a policy owner o with some principal o' that acts for o. This change is safe because the new policy allows only processes that act for o' to weaken it through declassification, while the original policy also allows processes with the weaker authority of o to declassify it.

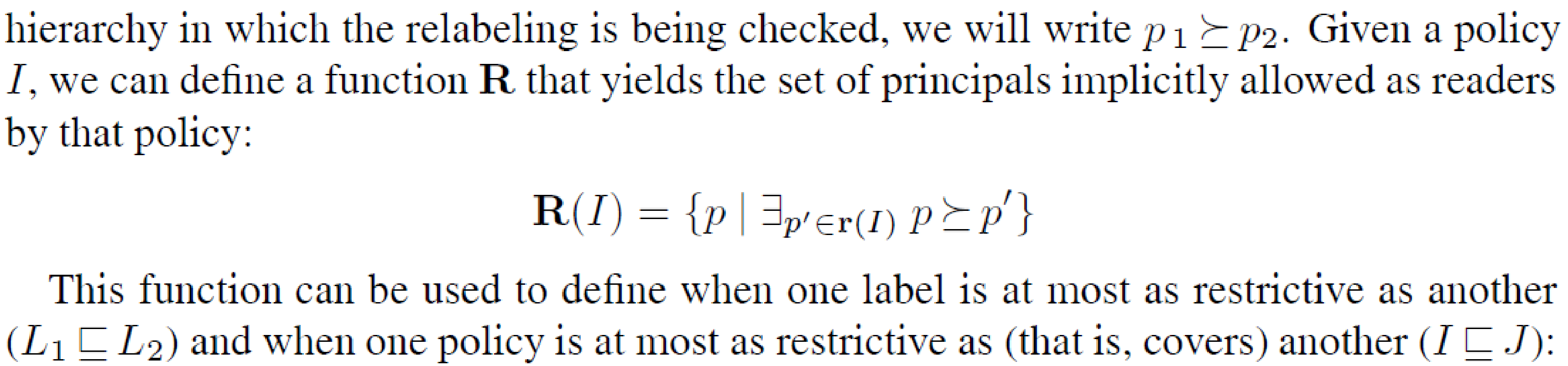
Examples of policy relationships

- 1. {amy : bob, carl} ⊆{amy : carl}
- 2. {amy : bob} ⊆ {amy : }
- 3. {amy : manager} ⊆ {amy : carl}
- 4. {manager : bob} ⊆ {carl : bob}
- 5. {amy : carl}≰ {amy : bob}
- 6. {amy : carl} ≰ {bob : carl}
- 7. {amy : manager} ≰ {amy : bob}
- 8. {manager : bob} ≰{bob : bob}



Complete Relabeling Rule

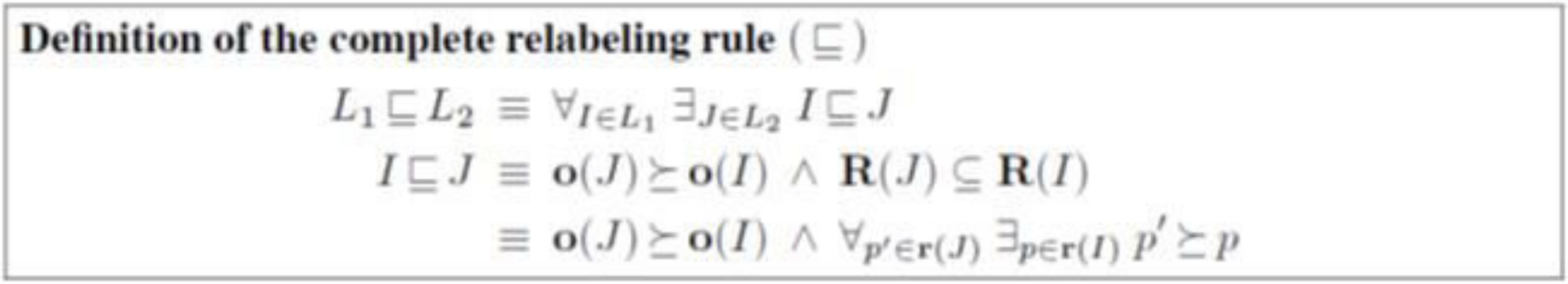
- o(I) denotes the owner of a policy I
- r(I) denotes the set of readers of I
- principal p1 acts for a principal p2 in the principal



hierarchy in which the relabeling is being checked, we will write $p_1 \succeq p_2$. Given a policy I, we can define a function $\mathbf{R}$ that yields the set of principals implicitly allowed as readers by that policy:

$$\mathbf{R}(I) = \{p \mid \exists_{p' \in \mathbf{r}(I)}\ p \succeq p'\}$$

This function can be used to define when one label is at most as restrictive as another $(L_1 \sqsubseteq L_2)$ and when one policy is at most as restrictive as (that is, covers) another $(I \sqsubseteq J)$:

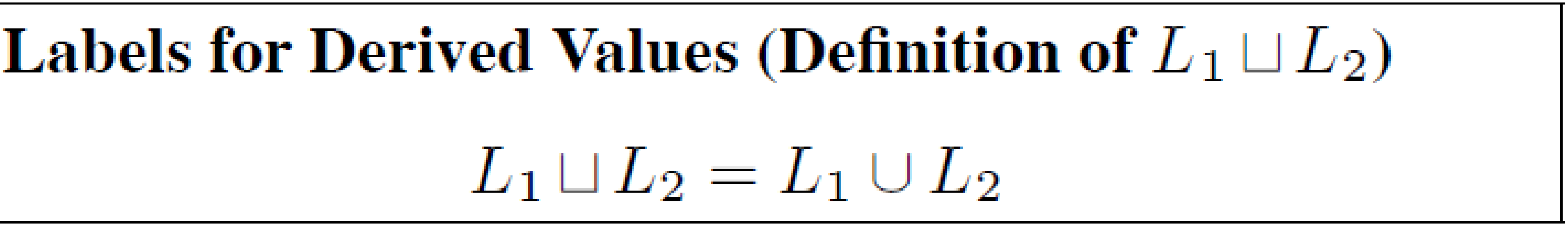


Definition of the complete relabeling rule $(\sqsubseteq)$

$$\begin{aligned} L_1 \sqsubseteq L_2 &\equiv \forall_{I \in L_1} \exists_{J \in L_2}\ I \sqsubseteq J \\ I \sqsubseteq J &\equiv \mathbf{o}(J) \succeq \mathbf{o}(I)\ \wedge\ \mathbf{R}(J) \subseteq \mathbf{R}(I) \\ &\equiv \mathbf{o}(J) \succeq \mathbf{o}(I)\ \wedge\ \forall_{p' \in \mathbf{r}(J)} \exists_{p \in \mathbf{r}(I)}\ p' \succeq p \end{aligned}$$

Join Rule for labels

- Least restrictive set of policies that enforces all the policies in L1 and L2 is simply the union of the two set of policies.
- The least restrictive label is the least upper bound or join of L1 and L2, written as L1 ⊔ L2 (⊕ also has been used to denote the join of two security classes)



Labels for Derived Values (Definition of $L_1 \sqcup L_2$)

$$L_1 \sqcup L_2 = L_1 \cup L_2$$

Join of labels

- {amy : bob} ⊔

  {amy : bob, carl} is

- {amy : bob; amy : bob, carl},

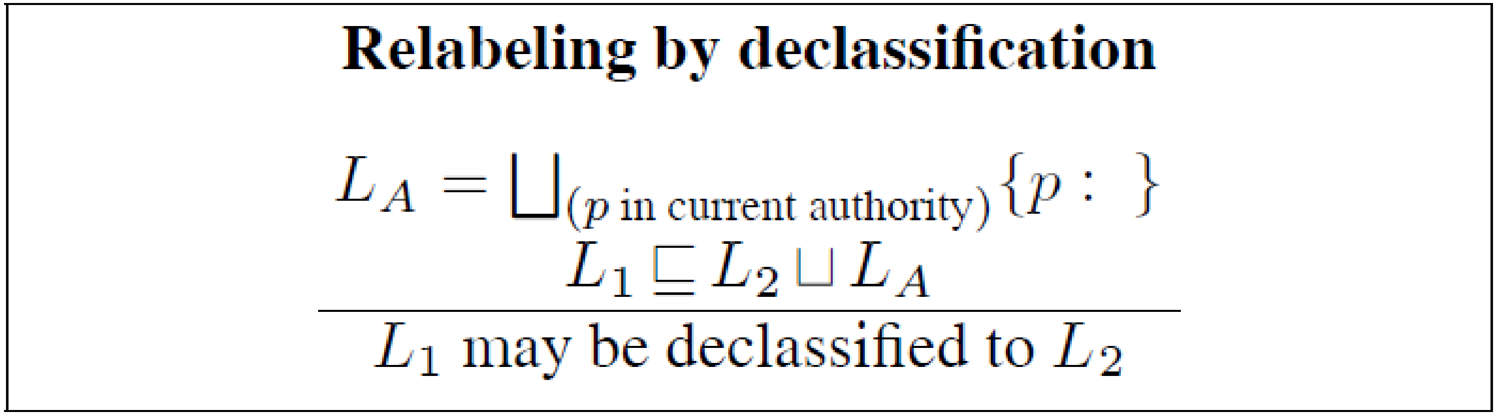
≡{amy : bob},

  – the second policy in the union is covered by the first, regardless of the principal hierarchy

Declassification

- Labels in this model contain information about the owners of labeled data, these owners can retain control over the dissemination of their data, and relax overly restrictive policies when appropriate. This second kind of relabeling is a selective form of declassification.
- Code running with the authority of a principal can declassify data by creating a copy in whose label a policy owned by that principal is relaxed. In the label of the copy, readers may be added to the reader set, or the policy may be removed entirely, which is effectively the same as adding all principals as readers in the policy.
- At any moment while executing a program, a process is authorized to act on behalf of some (possibly empty) set of principals, → referred to as the authority of the process
- A process may weaken or remove any policies owned by principals that are part of its authority. Therefore, the label L1 may be relabeled to L2 as long as $L1 \sqsubseteq L2 \sqcup LA$, where LA is a label containing exactly the policies of the form {p :} for every principal p in the current authority.

Declassification



Relabeling by declassification

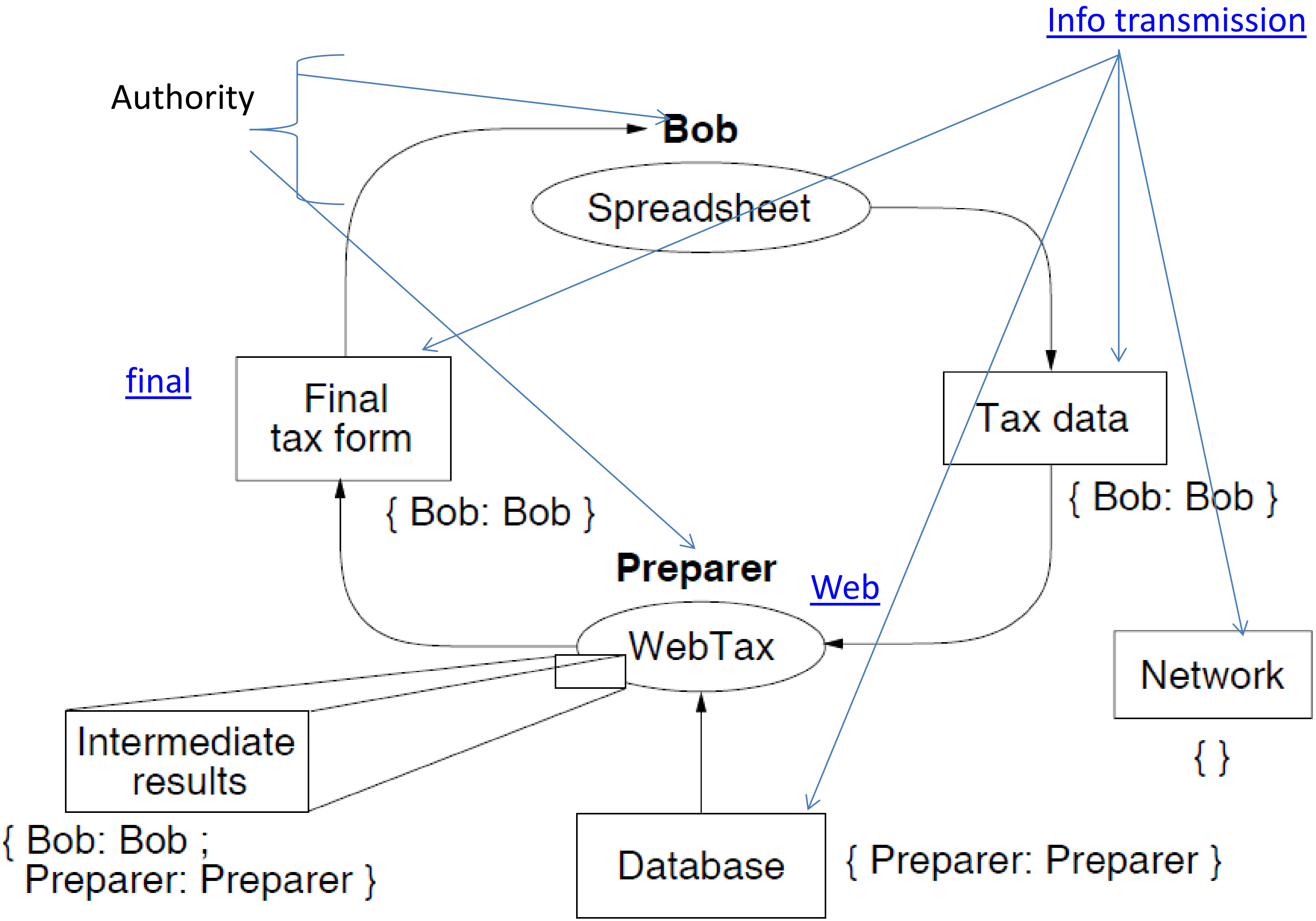
$$L_A = \bigsqcup_{(p \text{ in current authority})} \{p : \ \}$$

$$\frac{L_1 \sqsubseteq L_2 \sqcup L_A}{L_1 \text{ may be declassified to } L_2}$$

- **For all policies J in L1, there must be a policy K in L2 that is at least as restrictive.**
- **declassification rule has the intended effect as for policies J in L1 that are owned by a principal p in the current authority, a more restrictive policy K is found in LA**
- **For other policies J, the corresponding policy K must be found in L2, since the current authority does not have the power to weaken them.**
- **This shows that a label L1 always may be declassified to a label that it could be relabeled to by restriction, because the relabeling condition L1 ⊑ L2 implies the declassification condition L1 ⊑ L2 ⊔LA.**

Channels

- Input and output channels allow data to enter and leave the domain in which the label rules are enforced.
- Channels are half-variables; like variables, they have an associated label and can be used as an information conduit. However, they provide only half the functionality that a variable provides: either input or output.
  - As with a variable, when a value is read from an input channel, the value acquires the label of the input channel.
  - Similarly, a value may be written to an output channel only if the label of the output channel is at least as restrictive as the label on the value; otherwise, an information leak is presumed to occur.



Policy Specification

- The authority to act as Preparer need not be possessed by the entire WebTax application, but only by the part that performs the final release of the tax form.

- By limiting this authority to a small portion of the application, the risk of accidental release of the database is reduced. Thus the WebTax application might have a small top-level routine that runs with the authority of Preparer, while the rest of its code runs with no authority.

Integrity Labels

DUAL

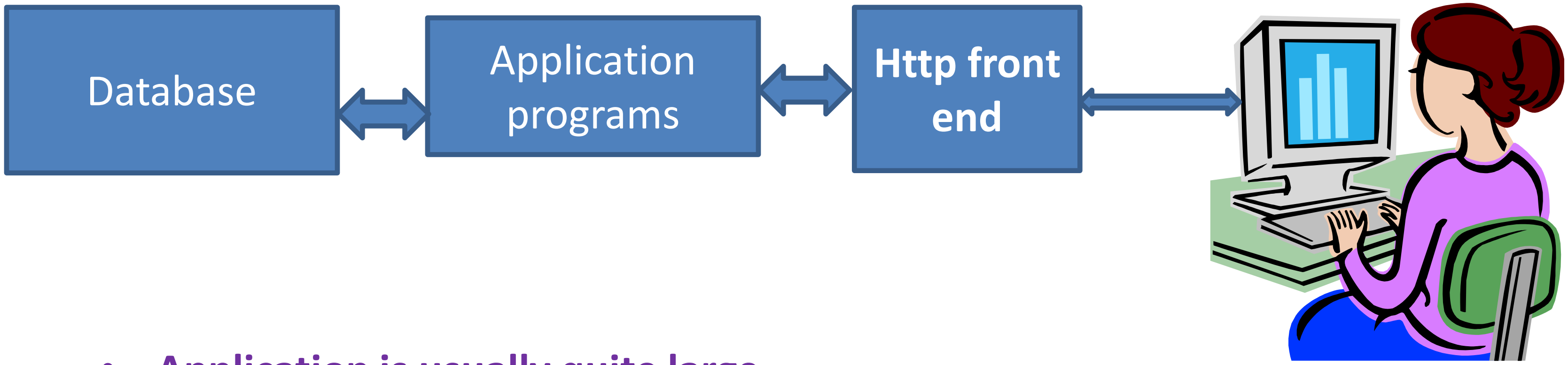
Endorsement

Decentralized declassification in PL (Jif)

- Jif can track information flow at the level of individual variables and perform most label checks, at compile time. It also has the luxury of relying on the underlying operating system for bootstrapping, storage, trusted input files, administration, etc.,
- Jif labels allow different principals to express their security concerns by specifying what other principals are allowed to read or write certain data.
- Language-based techniques largely avoid addressing many practical issues such as trust management, resource allocation, support for heterogeneous systems, and execution of arbitrary machine code.

Securing Distributed Systems with Information Control

- Build secure applications from mostly untrusted code by using information flow control to enforce data security
- Eg. Enforce data security policy when executing untrusted code with access to sensitive data;
  - an untrusted application may be able to read some sensitive data, but it should not be able to surrepitiously export this data from the system
    - Example: virus scanner: accesses all of user's private data but should never their contents to anyone else (Symantec 10.x Anti-virus wormhole (2006) placed millions at risk)



Database

Application programs

Http front end

- Application is usually quite large
  - It uses third party libraries
- It has access to entire user database

- Works properly if all the code is verified
  - Which is impossible
- Bugs in application can enable data stealing

(PayMaxx app code exposed 100,000 users' SSNs)

How do we work in untrusted environments?

- Eliminating bug in all application is impossible
- Can user data be kept secure even if applications are malicious?

YES:

- Track flow of user's data through system
- Only send user's data to that user's browser
- No need to audit/understand application code

HOW: OS'es like Asbestos, HiStar, Flume

Limitation: works only on one machine

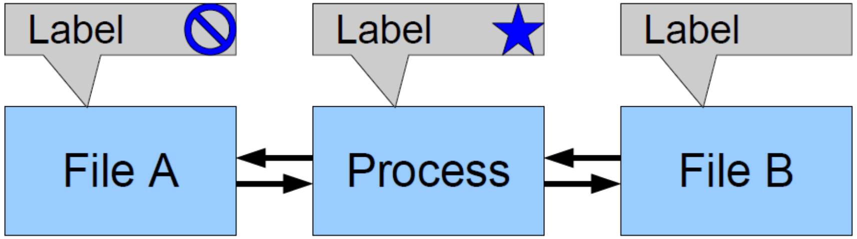
– Web applications need multiple machines for scale

Labels control information flow

🟦 Color is category of data (e.g. my files)

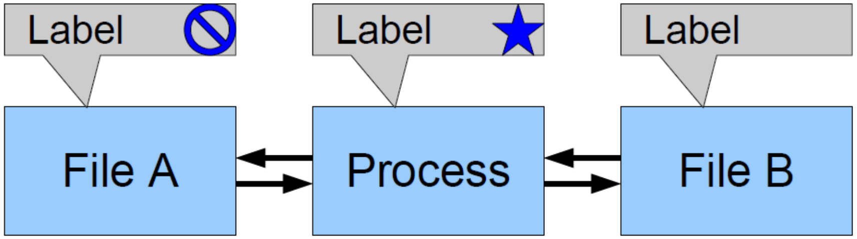
🚫 Blue data can flow only to other blue objects

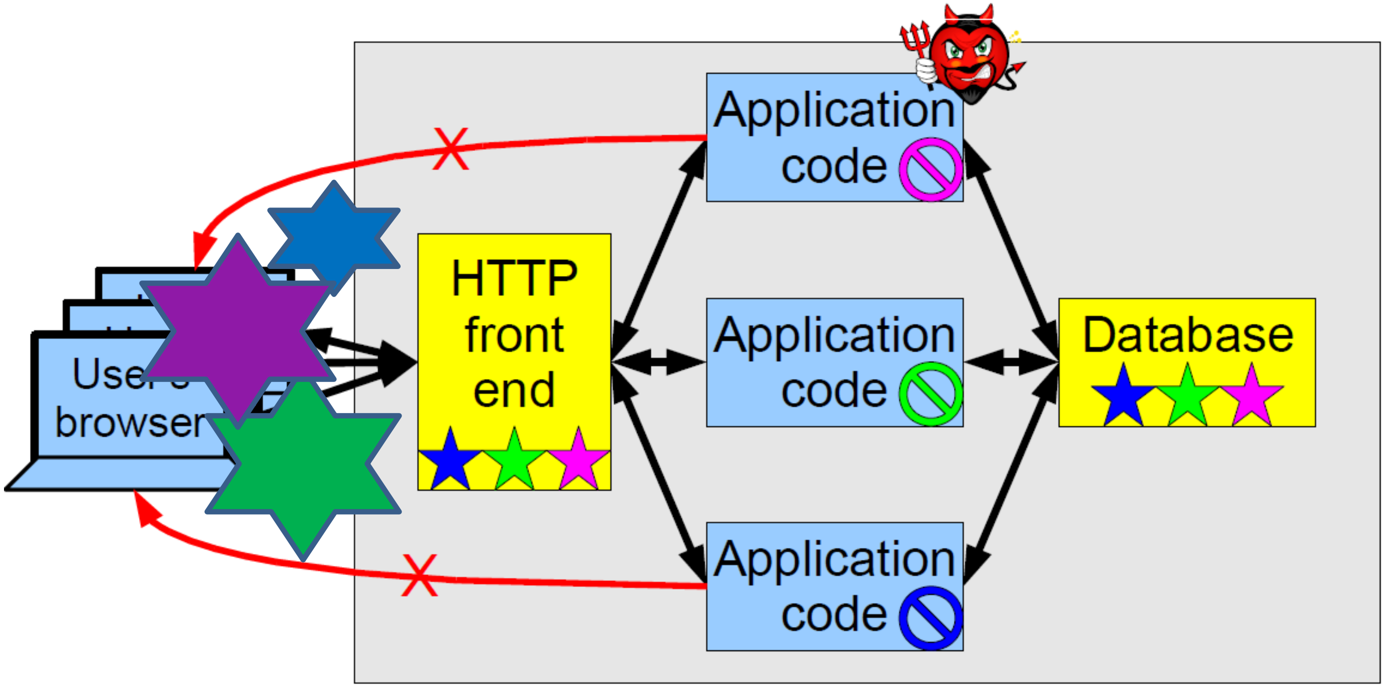
★ Owns blue data, can remove color (e.g. encrypt)



Labels are egalitarian

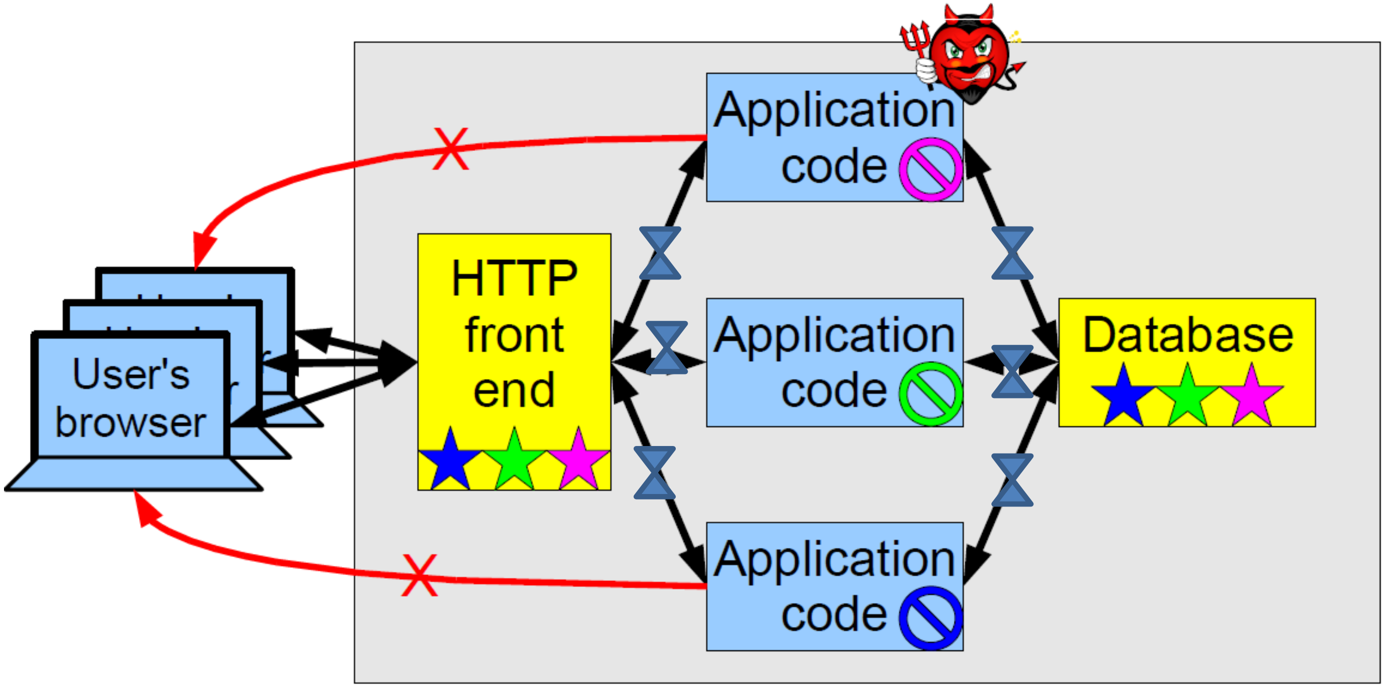
- Any process can request a new category (color)
  - Gets ownership of that category (★)
  - Uses category in labels to control information flow
  - Can grant ownership to others





- **Separate colour for each users data**
- **Track each user's data in app**
- **Labels prevent application code from disclosing data onto network**
- **Front-end uses ownership to send data only to user's browser**

- When the server gets overloaded -- No flow of information

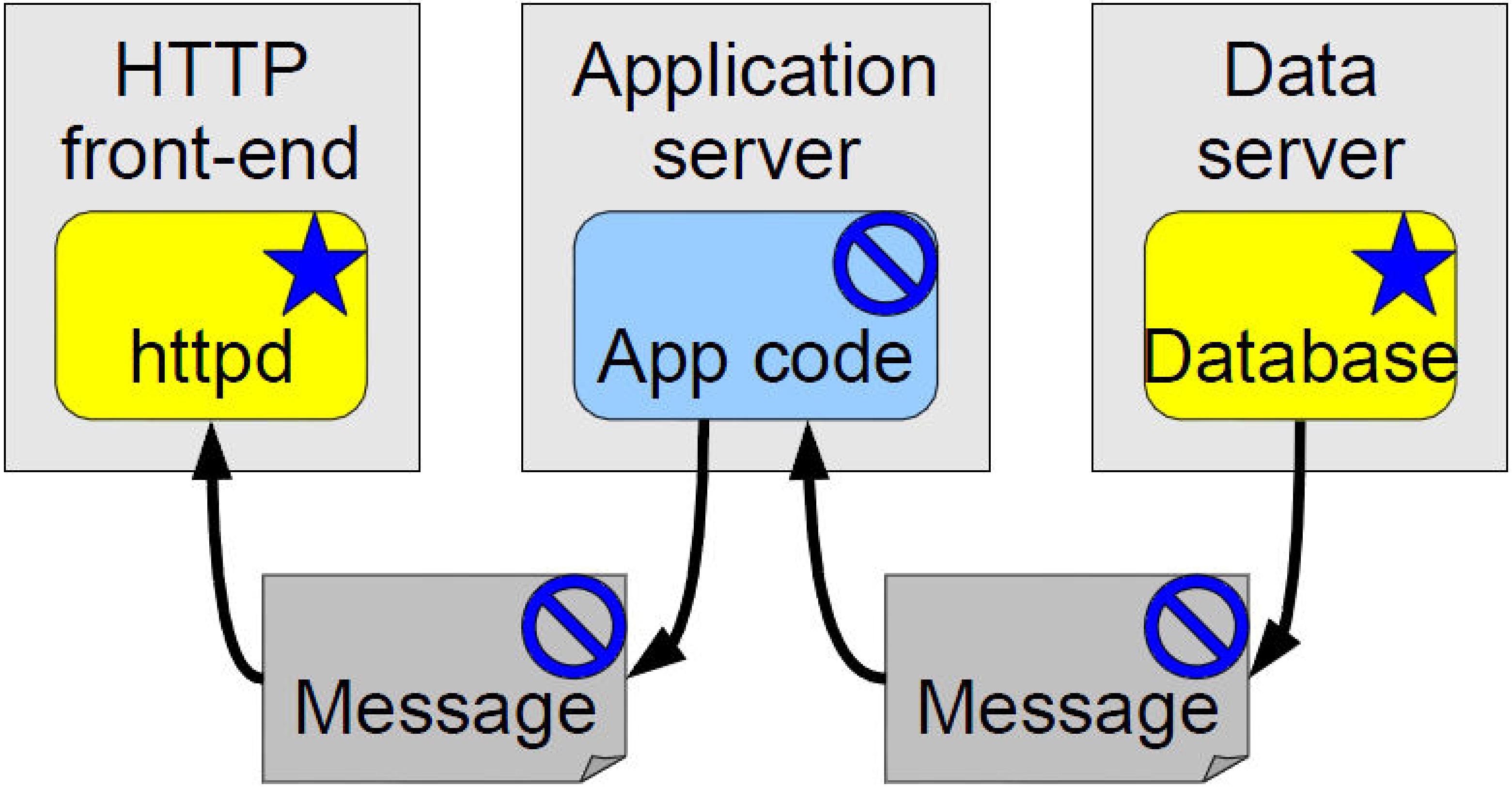


Is there Limitation?

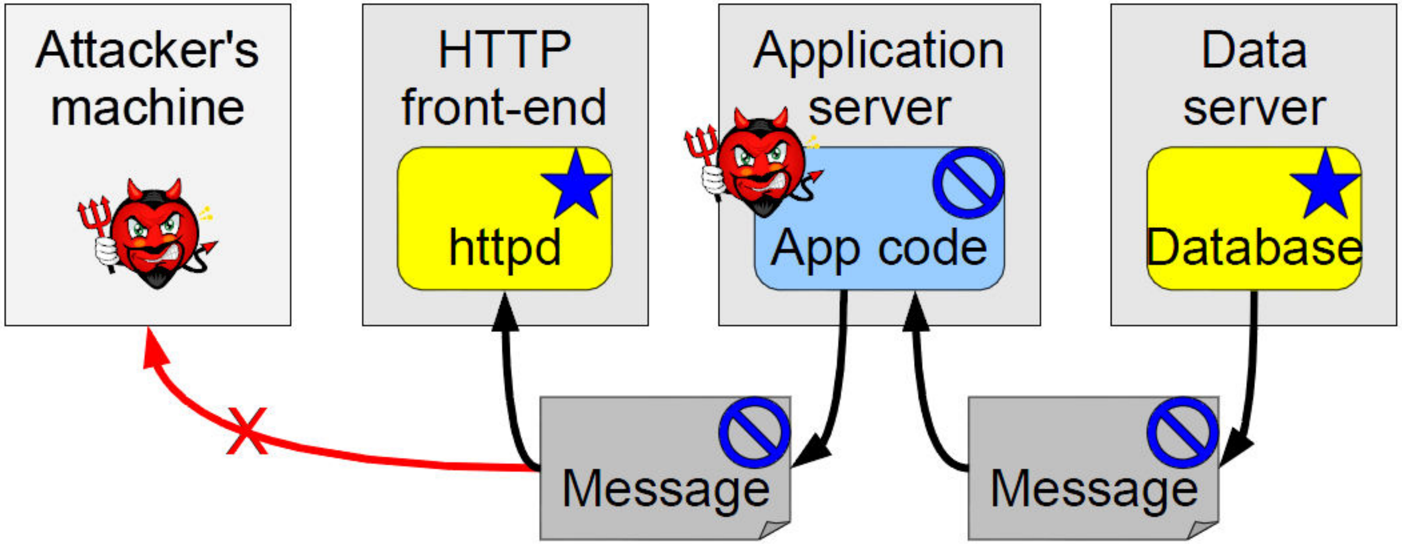
- **OS alone cannot control information flow in distributed system**

- **Find a *safe protocol* for processes to communicate noting that there is no equivalent of a fully-trusted OS kernel that can make all decisions**

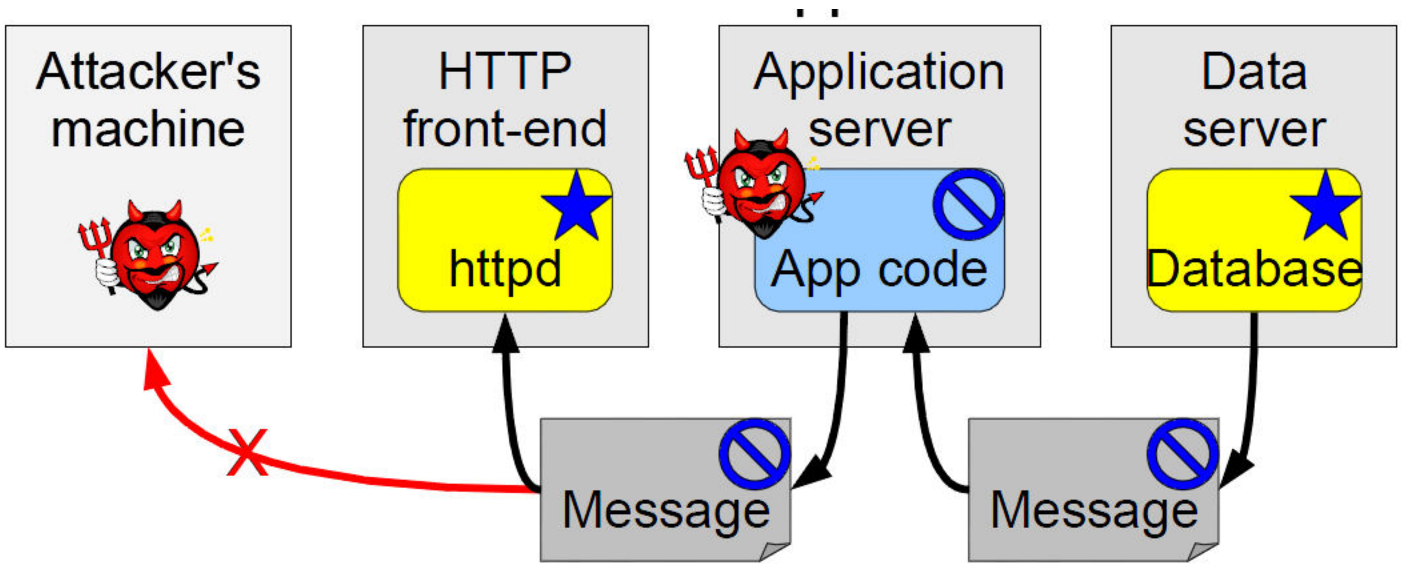
- **Label the data**
- **Each Machine uses its OS to enforce labels locally**



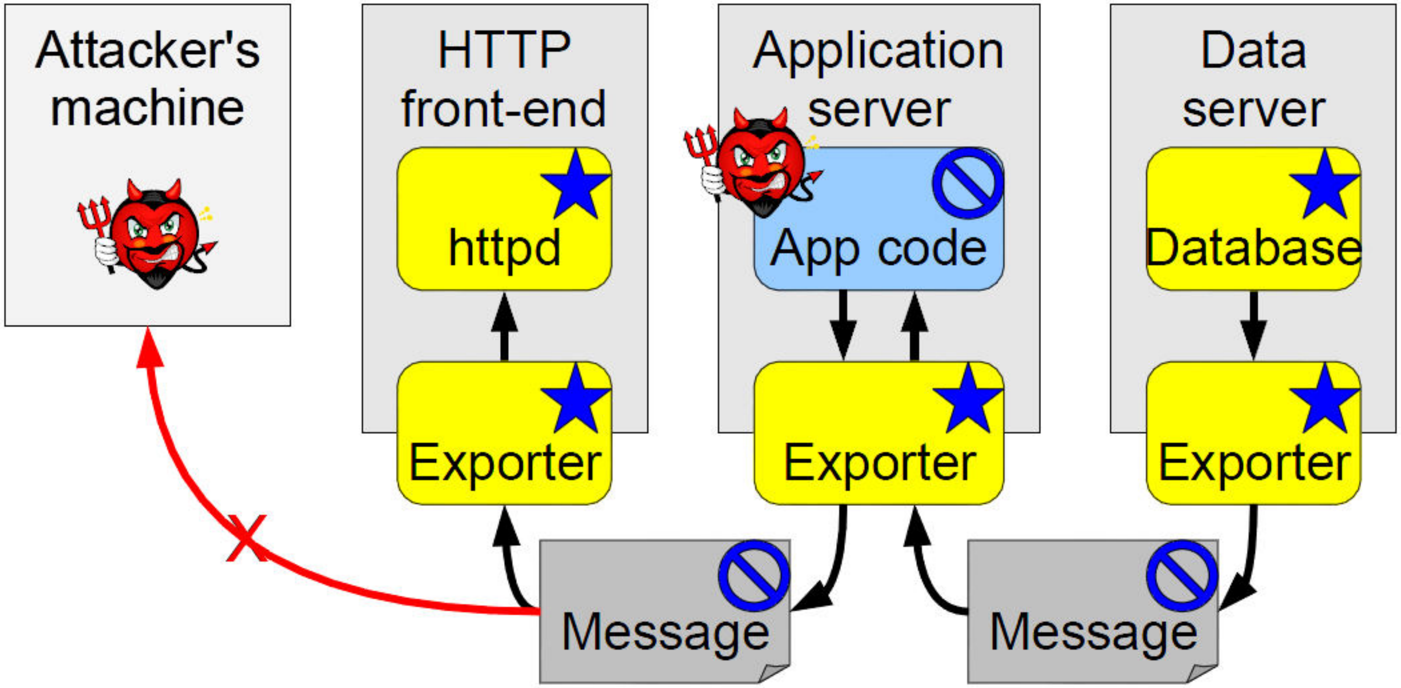
Trust: Decentralized



- **Need to trust the recipient of the message**



- **DB does not trust the application code**
- **DB trusts front app servers with a particular users data**

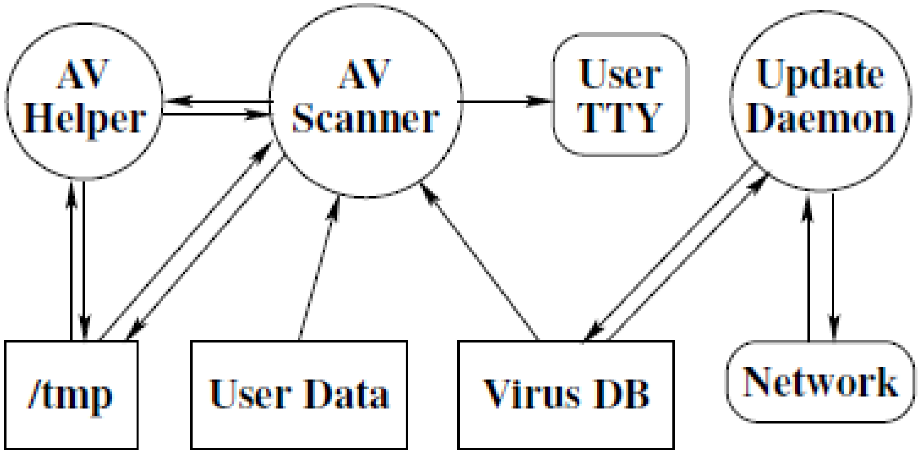


- **Database does not trust the application code**
- **Trusts the Exporter to contain the application code**

Information Flow Control In Operating Systems

Zeldovich

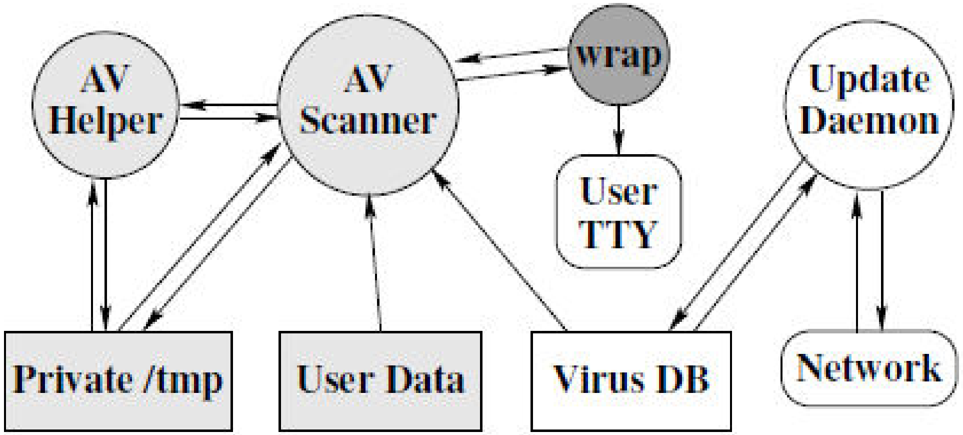
Enforcing data security policy while executing untrusted code



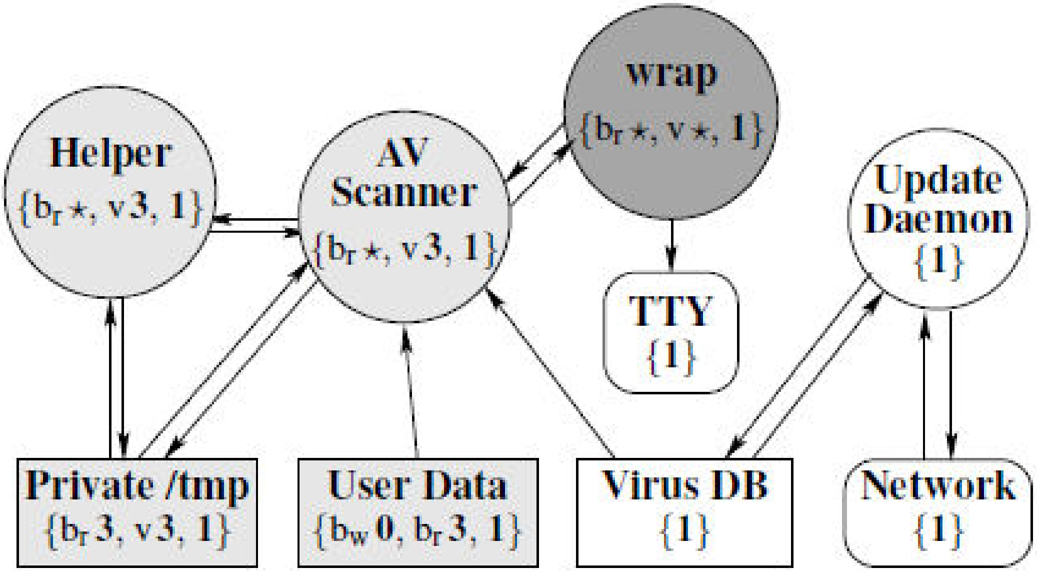
- Lightly Shaded – Confidential
- Unshaded – non-confidential
- Dark Shaded- Special privileges

to relay the scanner’s confidential output to the terminal.

- Circles: Processes
- Rectangles: Files/Dir
- Rounded Rect: Devices



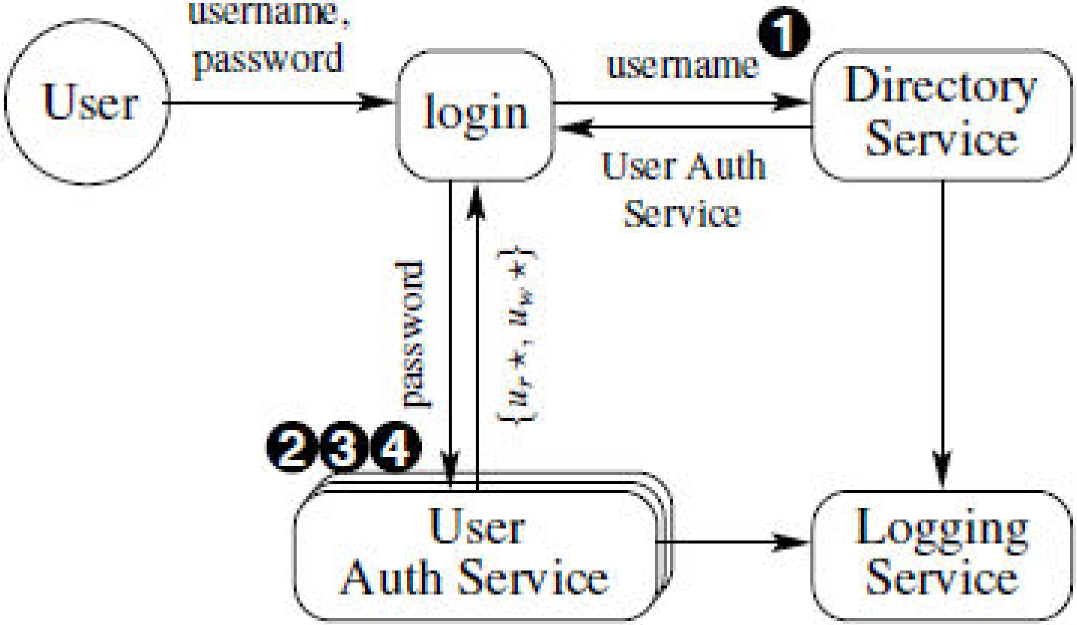
Labels on components of the HiStar ClamAV port.

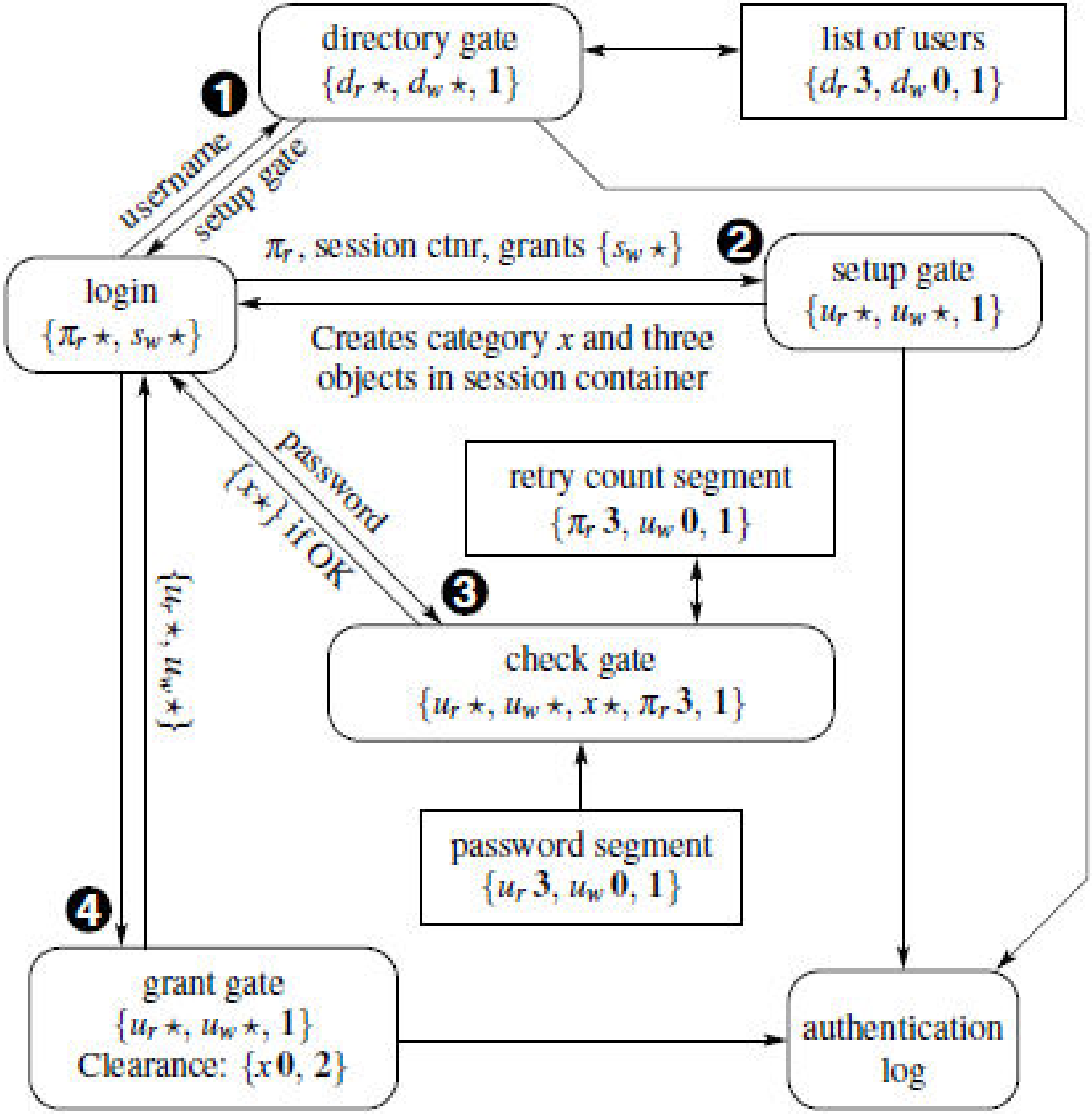


- all objects containing private user data are labeled confidential,
- kernel prevents any information flow from confidential to non-confidential objects.
- Initially, the scanner process is not labeled confidential, and the kernel prevents it from reading any confidential object, as per flow policy specified by the labels.
- If scanner changes to confidential, the kernel will allow it to read confidential objects.
- However, by changing its label to confidential, the scanner loses its ability to write any data to nonconfidential objects, or to change its label back to non-confidential, since threads are only allowed to change their label in a way consistent with information flow restrictions.
- Thus, regardless of the scanner's actions, kernel ensures that confidential data cannot reach non-confidential objects, including the network and the colluding update process.

Security Guarantees with HiStar

- Anti Virus Software

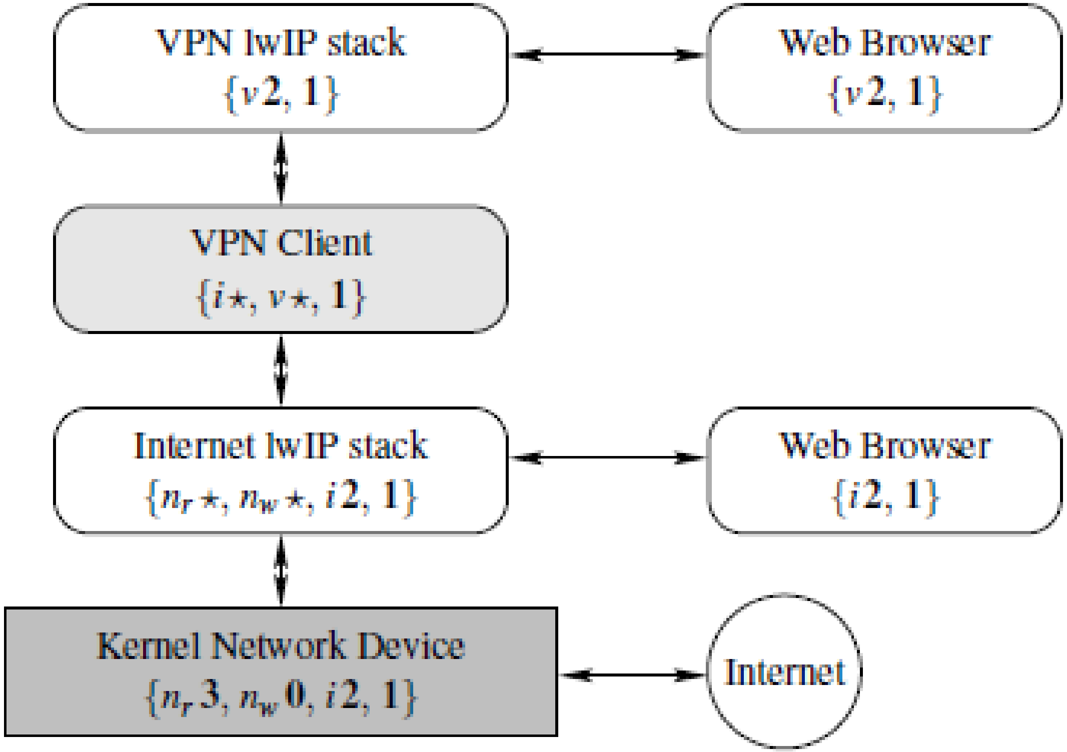



directory gate
$\{d_r \star, d_w \star, 1\}$
list of users
$\{d_r 3, d_w 0, 1\}$
1
username
setup gate
π_r, session ctnr, grants $\{s_w \star\}$
2
login
$\{\pi_r \star, s_w \star\}$
setup gate
$\{u_r \star, u_w \star, 1\}$
Creates category x and three objects in session container
password
$\{x \star\}$ if OK
retry count segment
$\{\pi_r 3, u_w 0, 1\}$
3
check gate
$\{u_r \star, u_w \star, x \star, \pi_r 3, 1\}$
$\{u_r \star, u_w \star\}$
password segment
$\{u_r 3, u_w 0, 1\}$
4
grant gate
$\{u_r \star, u_w \star, 1\}$
Clearance: $\{x 0, 2\}$
authentication log

VPN Isolation

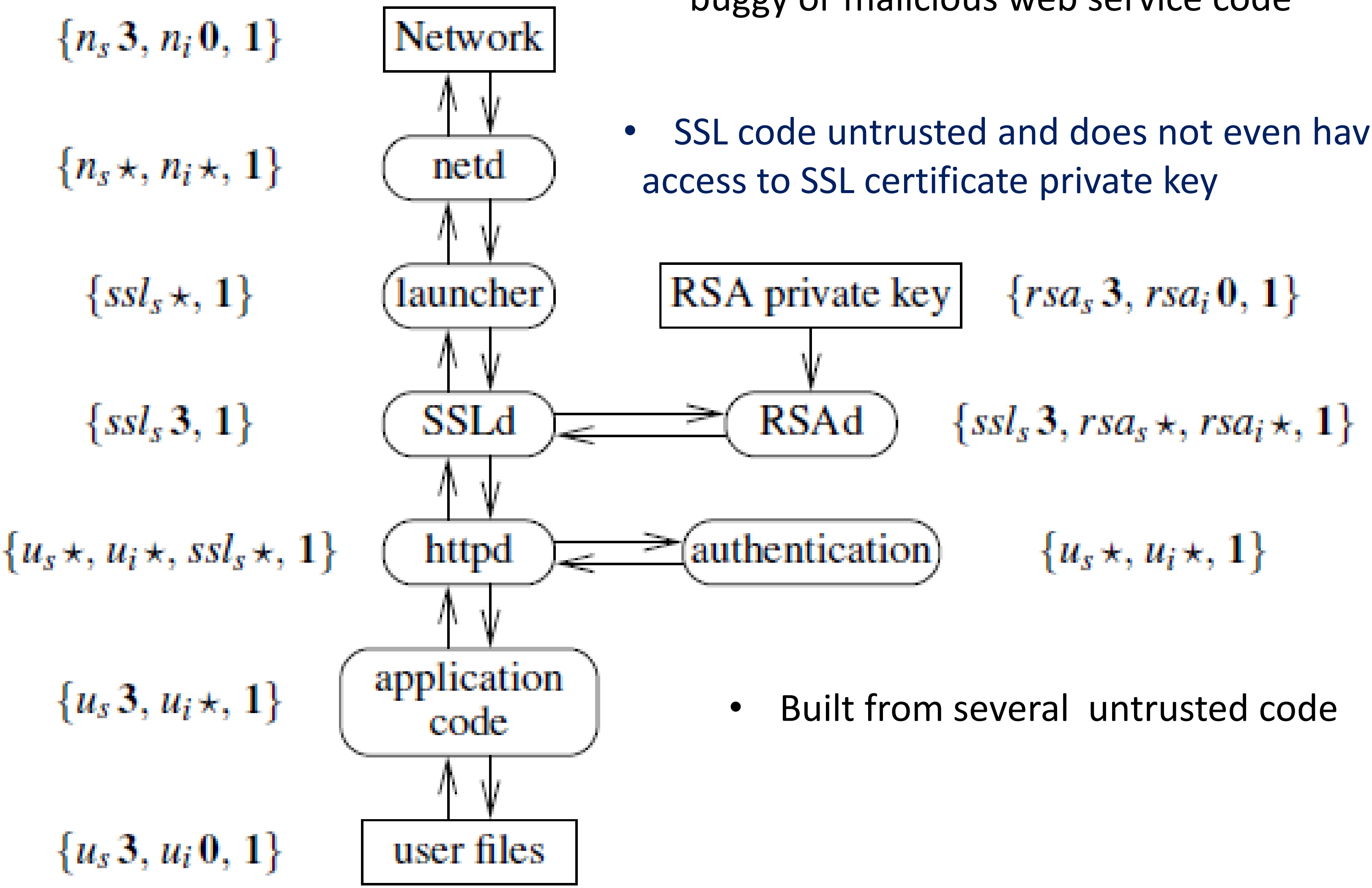
- Networks rely heavily on firewalls for security
  - bridging them to the open Internet poses a serious danger (eg., Slammer worm disabled a safety monitoring system at a nuclear power plant in 2003)
- Usual to connect home machines and laptops to otherwise firewalled networks through encrypted virtual private networks (VPNs).
  - When VPNs let the same machine connect to either side of a firewall, there is a risk having malware either infect internal machines or (eg., Sircam worm) divulge sensitive documents to the world.
- Track origin of data with label and control flow of inf.
  - Tainting anything received from Internet

VPN Client


VPN lwIP stack
$\{v2, 1\}$
Web Browser
$\{v2, 1\}$
VPN Client
$\{i\star, v\star, 1\}$
Internet lwIP stack
$\{n_r\star, n_w\star, i2, 1\}$
Web Browser
$\{i2, 1\}$
Kernel Network Device
$\{n_r 3, n_w 0, i2, 1\}$
Internet

Web Server

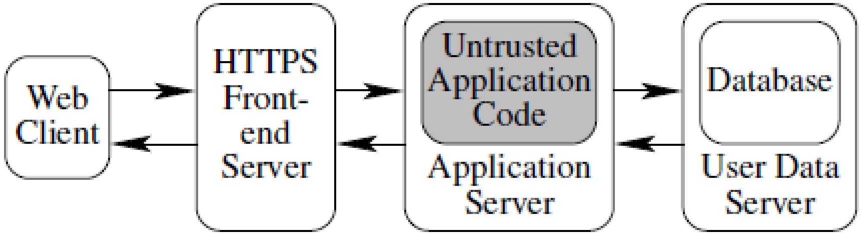
- Isolate Different user's data to tolerate buggy or malicious web service code
- SSL code untrusted and does not even have access to SSL certificate private key

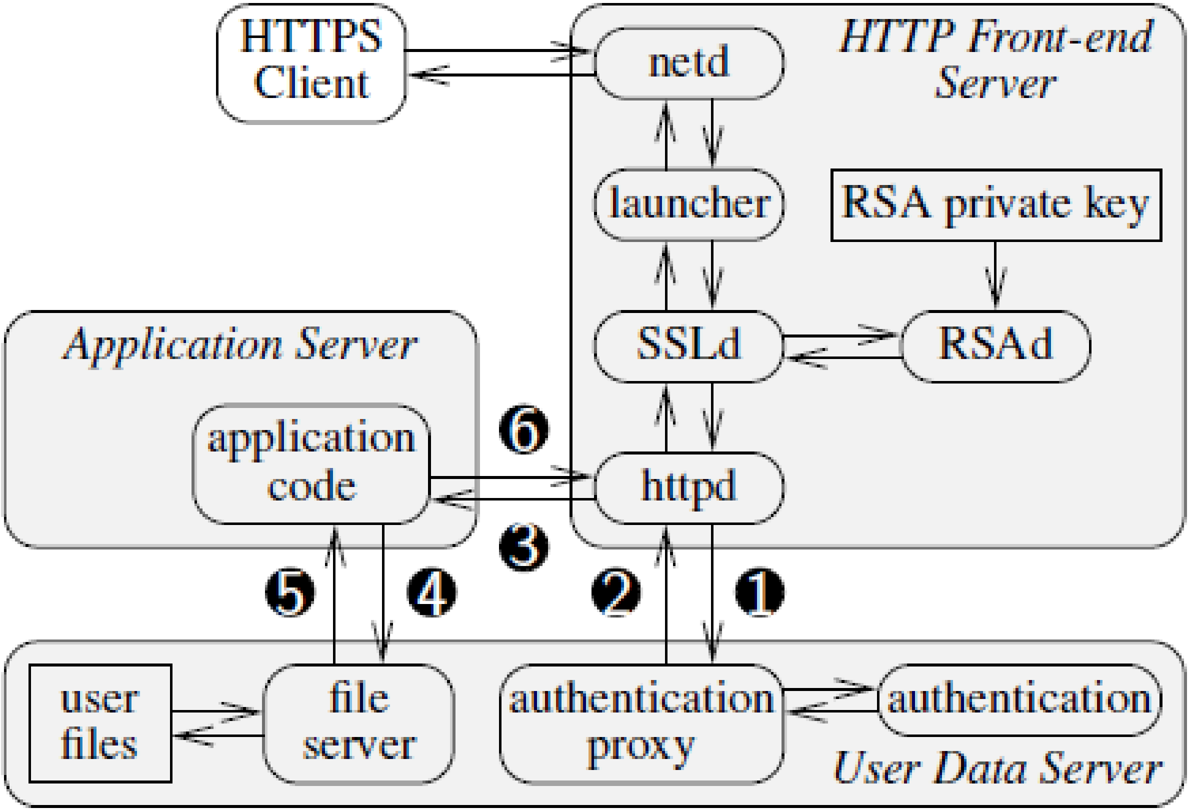

$\{n_s 3, n_i 0, 1\}$
Network
$\{n_s \star, n_i \star, 1\}$
netd
$\{ssl_s \star, 1\}$
launcher
RSA private key
$\{rsa_s 3, rsa_i 0, 1\}$
$\{ssl_s 3, 1\}$
SSLd
RSAd
$\{ssl_s 3, rsa_s \star, rsa_i \star, 1\}$
$\{u_s \star, u_i \star, ssl_s \star, 1\}$
httpd
authentication
$\{u_s \star, u_i \star, 1\}$
$\{u_s 3, u_i \star, 1\}$
application code
$\{u_s 3, u_i 0, 1\}$
user files

- Built from several untrusted code

- User connections are initially handled by the launcher process, which listens for incoming connections from web browsers and allocates resources to handle each request.
- For each request, the launcher spawns an SSLd daemon to handle the SSL connection with the user’s web browser, and an httpd daemon to process the user’s plaintext HTTP request.
- The launcher then proxies data between SSLd and the TCP connection to the user’s browser. SSLd, in turn, uses the RSAd daemon to establish an SSL session key with the web browser, by generating an RSA signature using the SSL certificate private key kept by RSAd.

Distributed Web Server & security




HTTPS Client
HTTP Front-end Server
netd
launcher
RSA private key
SSLd
RSAd
httpd
Application Server
application code
❻
❸
❺
❹
❷
❶
user files
file server
authentication proxy
authentication
User Data Server

- The first tier, the HTTP front-end servers, run components responsible for accepting client connections and handling the HTTP protocol: the launcher, SSLd, RSAd, and httpd.
- The second tier, or application servers, run application-specific logic to handle user requests.
- Finally, the third tier, user data servers, store private user data & perform user authentication.

- Servers in the first two tiers are largely stateless, making it easy to improve overall performance by adding more physical machines.

  This is an important consideration for complex web applications, where simple tasks such as generating a PDF document can easily consume 100 milliseconds of CPU time, when using GNU ghostscript for this purpose.
- The third tier, the user data servers, can also be partitioned over multiple machines, by keeping a consistent mapping from each individual user to the particular user data server responsible for his data.