Securing Distributed Systems with Information Flow Control

CS6204 Privacy and Security
Virginia Tech

-Nikhil Komawar
Oct 6, 2011
Introduction
Information Flow Control
Design
Implementation
Instance applications
Evaluation
We try to design a system which remains secure despite the untrustworthy code.

**Decentralized trust**:
- OS has a completely trusted kernel
- Not the same case for a Distributed System
- One solution: centralize trust mechanism (Not the most effective!)

**Egalitarian Mechanism**:
- Specifically designed for applications
- Even unprivileged code can use DIFC

**No inherent covert channels**:
- Avoid any covert channels in Dstar’s interface to meet different security requirements.
A typical web application.

![Diagram showing the components of a web application: Web Browser, HTTPS Front-end Server, Untrustworthy Application Code, Database, Web Server.]}
Labels:
- Ensure the communication between two processes
- Direction of communication

Processes S and R:
- Flows from S to R \( L_S \subseteq L_R \).
- Flows even from R to S \( L_R \subseteq L_S \).

On a distributed system, label \( L_M \) for a message \( M \) \( L_S \subseteq L_M \subseteq L_R \).

\( L_1 \subseteq L_2 \) if and only if \( L_1 \) contains all the integrity categories in \( L_2 \) and \( L_2 \) contains all the secrecy categories in \( L_1 \).
If information flows to higher label then we will not get anything out of the system.

\[ L_S \sqsubseteq_{O_S} L_M \sqsubseteq_{O_R} L_R. \]

S can send message M to R if

\[ L_1 \sqsubseteq L_2 \text{ always implies } L_1 \sqsubseteq_{O_P} L_2, \]
A process P’s downgrading are also represented as categories: P owns a category $c$ when $c \in O_P$.

$L_1 \subseteq_O L_2$ if and only if $L_1 - O \subseteq L_2 - O$.

Set of categories in DStar message M

$G_M \subseteq O_S$. 
Lattice structure
Each process has third set of categories \( C_P \), besides \( L_P \) and \( O_P \).

\( C_P \), Are called clearance: which gives a right to the process P to raise its own label say \( L_{P}^{new} \).

Now \( L_P \subseteq L_{P}^{new} \subseteq C_P \).
Figure 3: Example use of labels to prevent the PayMaxx application code from inappropriately disclosing data, with two users, A and B. Rounded boxes are processes. Ellipses are messages. Shaded components are labeled with a user’s secrecy category, $a_s$ or $b_s$ for A and B respectively. The front end communicates with the database to authenticate the user and obtain ownership of the user’s secrecy category.
Run on each host: an exporter daemon which provides three requirements:-

- Track labels assigned to and categories owned by each process on the machine
- A process cannot send messages with inappropriate label
- Send and accept network message M when a remote exporter can be trusted
We may have different host Oses Asbestos, HiStar, Flume.

We refer to their categories as *OS categories and OS labels*.

Downgrading privileges: $\mathcal{O}_p$

Corresponding category $\mathcal{L}_p$. 
Local checks, Trust and Addressing

Local Checks

Decentralized Trust

Addressing
Exporter Interface

```c
struct category_name {
    pubkey creator;
    category_type type;
    uint64_t id;
};

struct dstar_message {
    pubkey recipient_exporter;
    slot recipient_slot;
    category_set label, ownership, clearance;
    cert_set certs;
    mapping_set mapset;
    opaque payload;
};

void dstar_send(ip_addr, tcp_port, dstar_message,
             cert_set, mapping_set);
```
- Delegation service
  - Local process to another exporter

- Mapping service
  - Mapping between DStar and local OS categories

- Guarded invocation service
  - Launches executables with specified arguments and privileges

- Resource allocation service
  - Allocates space for data, CPU time for threads
Object types in HiStar

- Segments: 0 or more memory pages
- Containers: similar to directories
- Threads: execute code
- Gates: used for IPC and privilege transfer
DStar and HiStar Mapping

Object is in a container

HiStar category \( c \)

DStar category \( d \)

Binding segment

User-provided container

Mapping container \( m \)

\( L_m = \{ e_s, e_i \} \)

Binding segment \( b \)

Contents = \( (c, d, m, b, g) \)

\( L_b = \{ e_s, e_i \} \)

Gate \( g \)

\( O_g = \{ c \} \)

\( A_g = \{ e_i \} \)
HiStar SSL web server

$L = \{n_s, n_i\}$

$L = \{\}, O = \{n_s, n_i\}$

$L = \{\}, O = \{ssl_s\}$

$L = \{ssl_s\}, O = \{\}$

$L = \{\}, O = \{u_s, u_i, ssl_s\}$

$L = \{\}, O = \{u_s, u_i\}$

$L = \{u_s\}, O = \{u_i\}$

$L = \{u_s, u_i\}$

Network

netd

launcher

SSLe

RSA

authentication

httpd

application code

user files

Fall, 2011 - Privacy&Security - Virginia Tech – Computer Science
## Code complexity

### Lines of C Code

<table>
<thead>
<tr>
<th>Component</th>
<th>Lines of Code</th>
<th>Label</th>
<th>Ownership</th>
<th>Effects of Compromise</th>
</tr>
</thead>
<tbody>
<tr>
<td>netd</td>
<td>350,000</td>
<td>{}</td>
<td>{\ns, ni}</td>
<td>Equivalent to an active network attacker; subject to same kernel label checks as any other process</td>
</tr>
<tr>
<td>launcher</td>
<td>310</td>
<td>{}</td>
<td>{\ssl}</td>
<td>Obtain plaintext requests, including passwords, and subsequently corrupt user data</td>
</tr>
<tr>
<td>SSLd</td>
<td>340,000</td>
<td>{\ssl}</td>
<td>{}</td>
<td>Corrupt request or response, or send unencrypted data to same user’s browser</td>
</tr>
<tr>
<td>RSAd</td>
<td>4,600</td>
<td>{\ssl}</td>
<td>{\rsa}</td>
<td>Disclose the server’s SSL certificate private key</td>
</tr>
<tr>
<td>httpd</td>
<td>300</td>
<td>{}</td>
<td>{\usi, \ui, \ssl}</td>
<td>Full access to data in attacker’s account, but not to other users’ data</td>
</tr>
<tr>
<td>authentication</td>
<td>320</td>
<td>{}</td>
<td>{}</td>
<td>Full access to data of the user whose agent is compromised, but no password disclosure</td>
</tr>
<tr>
<td>application</td>
<td>680,000+</td>
<td>{\usi}</td>
<td>{\ui}</td>
<td>Send garbage (but only to same user’s browser), corrupt user data (for write requests)</td>
</tr>
<tr>
<td>DStar exporter</td>
<td>3,700</td>
<td>{\usi}</td>
<td>{}</td>
<td>Corrupt or disclose any data sent or received via DStar on a machine</td>
</tr>
<tr>
<td>DStar client library</td>
<td>1,500</td>
<td>{\usi}</td>
<td>{}</td>
<td>Corrupt, but not necessarily disclose, data sent or received via DStar by an application</td>
</tr>
</tbody>
</table>
Web application on many HiStar machines
## Size of DStar structures

## Compression of data using zlib

<table>
<thead>
<tr>
<th>Data structure</th>
<th>Raw bytes</th>
<th>Compressed bytes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Public key</td>
<td>172</td>
<td>183</td>
</tr>
<tr>
<td>Category name</td>
<td>184</td>
<td>195</td>
</tr>
<tr>
<td>Category mapping</td>
<td>208</td>
<td>219</td>
</tr>
<tr>
<td>Unsigned delegation certificate</td>
<td>548</td>
<td>384</td>
</tr>
<tr>
<td>Signed delegation certificate</td>
<td>720</td>
<td>556</td>
</tr>
<tr>
<td>Unsigned address certificate</td>
<td>200</td>
<td>203</td>
</tr>
<tr>
<td>Signed address certificate</td>
<td>376</td>
<td>379</td>
</tr>
<tr>
<td>Null message (1)</td>
<td>200</td>
<td>194</td>
</tr>
<tr>
<td>Empty message (2)</td>
<td>1348</td>
<td>623</td>
</tr>
</tbody>
</table>
Microbenchmarks measuring time to sign and verify certificates

<table>
<thead>
<tr>
<th>Operation</th>
<th>Time (msec)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sign a delegation certificate</td>
<td>1.37</td>
</tr>
<tr>
<td>Verify a delegation certificate</td>
<td>0.012</td>
</tr>
<tr>
<td>Sign an address certificate</td>
<td>1.35</td>
</tr>
<tr>
<td>Verify an address certificate</td>
<td>0.011</td>
</tr>
<tr>
<td>Null RPC on same machine</td>
<td>1.84</td>
</tr>
</tbody>
</table>
## Throughput and Latency

<table>
<thead>
<tr>
<th>System</th>
<th>PDF workload</th>
<th>cat workload</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>throughput</td>
<td>latency</td>
</tr>
<tr>
<td>Linux Apache</td>
<td>7.6</td>
<td>137</td>
</tr>
<tr>
<td>HS PS+Auth</td>
<td>6.3</td>
<td>161</td>
</tr>
<tr>
<td>HS PS</td>
<td>6.6</td>
<td>154</td>
</tr>
<tr>
<td>HS no PS</td>
<td>6.7</td>
<td>150</td>
</tr>
<tr>
<td>DS on one machine</td>
<td>5.2</td>
<td>194</td>
</tr>
<tr>
<td>DS on multiple machines</td>
<td>varies</td>
<td>511</td>
</tr>
</tbody>
</table>
### Different configurations

<table>
<thead>
<tr>
<th>Calling machine</th>
<th>Execution machine</th>
<th>Communication</th>
<th>Throughput, req/sec</th>
<th>Latency, msec</th>
</tr>
</thead>
<tbody>
<tr>
<td>Linux same</td>
<td>HiStar same</td>
<td>none</td>
<td>505</td>
<td>2.0</td>
</tr>
<tr>
<td>Linux same</td>
<td>HiStar same</td>
<td>none</td>
<td>334</td>
<td>3.0</td>
</tr>
<tr>
<td>Linux TCP</td>
<td>HiStar DStar</td>
<td>none</td>
<td>160</td>
<td>6.3</td>
</tr>
<tr>
<td>Linux TCP</td>
<td>HiStar DStar</td>
<td>TCP</td>
<td>67</td>
<td>15.7</td>
</tr>
<tr>
<td>Linux TCP</td>
<td>HiStar DStar</td>
<td>DStar</td>
<td>61</td>
<td>20.6</td>
</tr>
</tbody>
</table>

*Note: The table above compares different configurations of calling and execution machines with various communication methods, showing throughput in requests per second and latency in milliseconds.*
DStar:

- Framework for securing distributed systems
- Uses security label mechanisms of DIFC Oses
- Exporter daemon on every machine
- Exporter ensures safe communication
- Trust relation left to the applications
- Self certifying categories – fully trusted machines not needed
- Showed a DStar implementation of a three tired Web Server
- Web server scales well
Thank You!