

CLAMP: **Preventing Large-Scale Data Leaks**

Bryan Parno, Jonathan McCune,
Dan Wendlandt, David Andersen, Adrian Perrig
Carnegie Mellon University

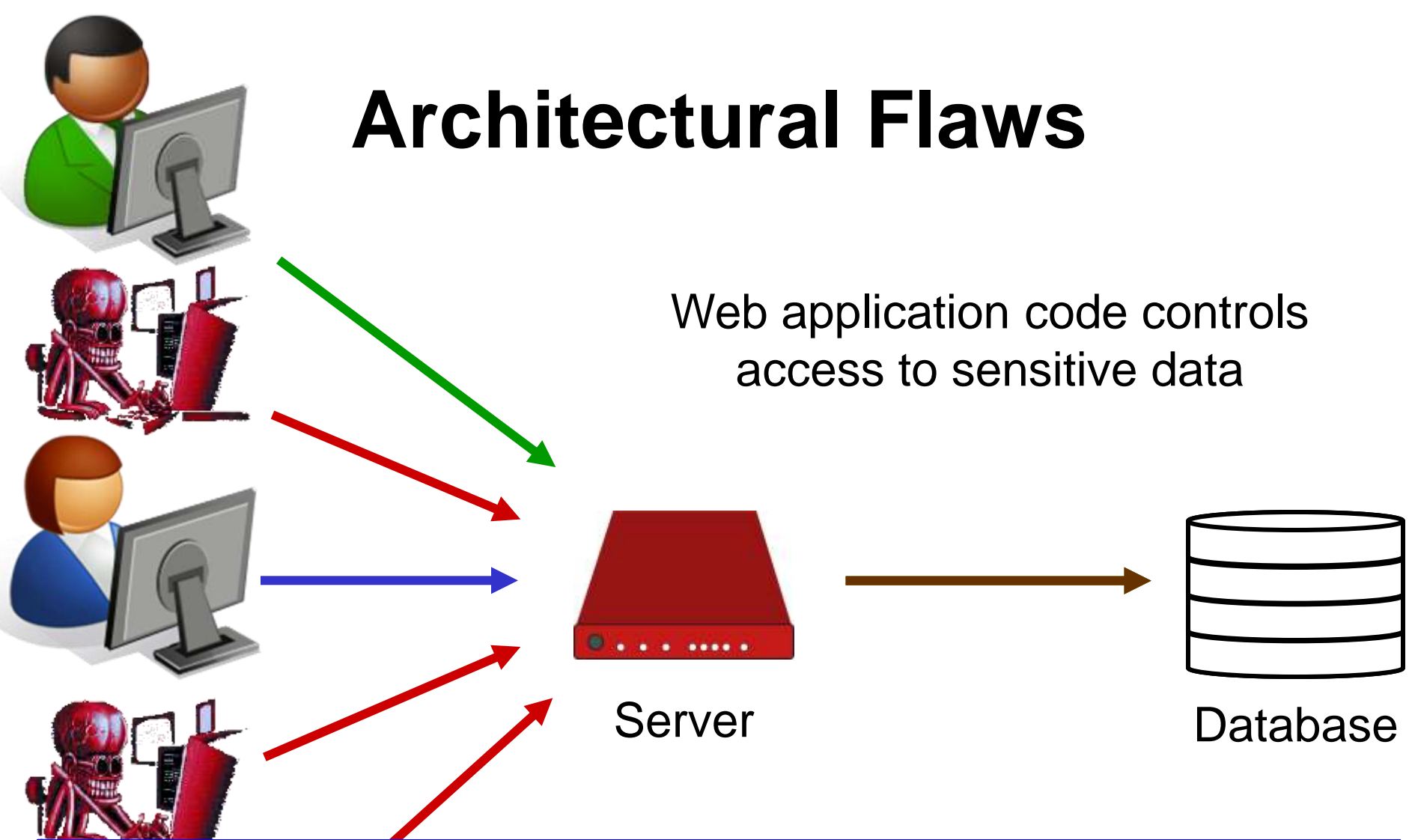
Data Leaks Are All Too Common

- Sample of 2008 remote exploits:
 - Davidson Companies: 226,000 records (name, SSN, acct info)
 - University of Florida: 300,000 records (personal & medical info)
 - Ameritrade: Contact info for 6.3M customers
- Trend of making sensitive data available online
 - Online banking, e-commerce, medical data, etc.
- A single compromise can leak data about millions of customers
 - Highly attractive targets



Architectural Flaws

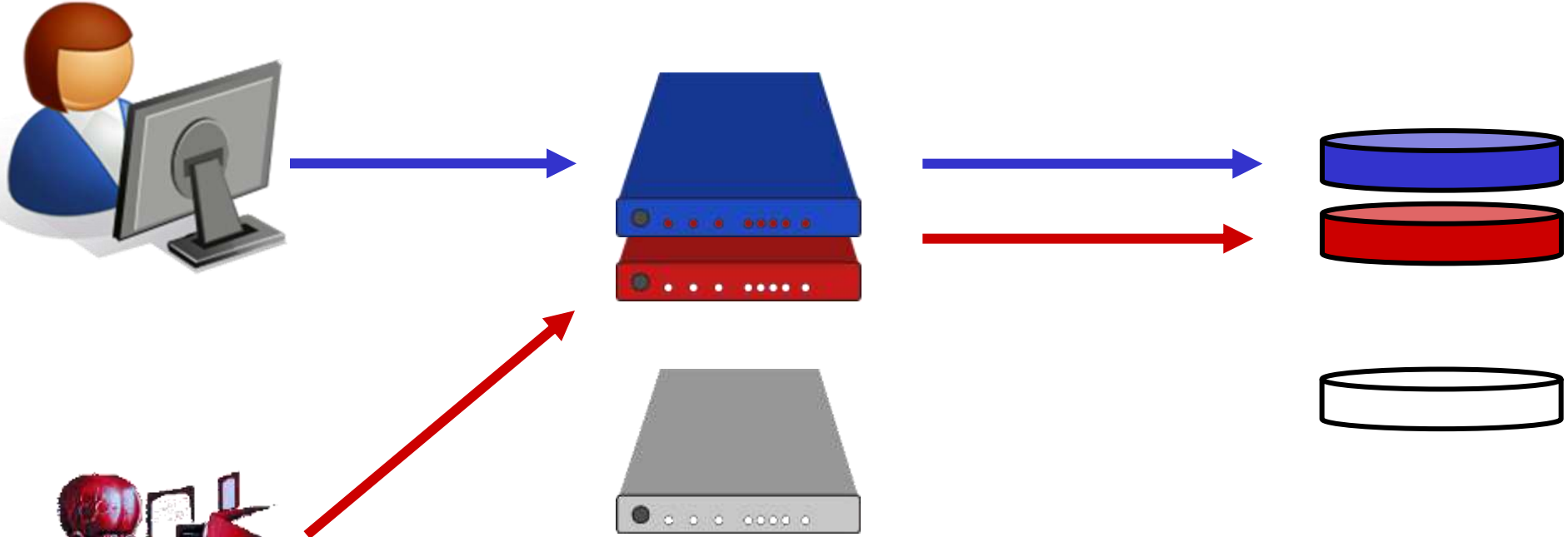
Web application code controls access to sensitive data



Compromised Server → Compromised Data

Clamp

Each customer interacts with a private web server

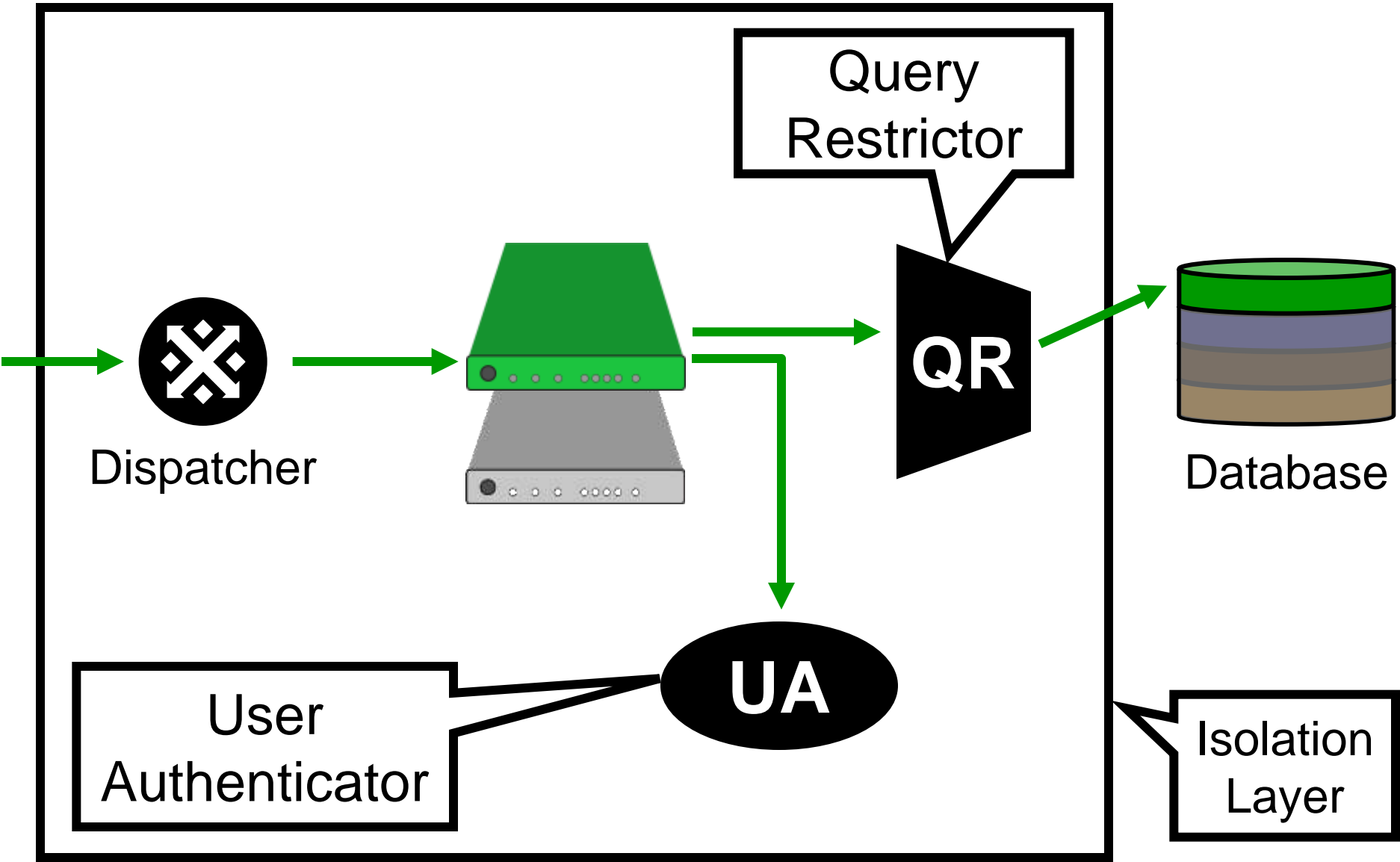


A web server can only access one person's data

Key Ideas

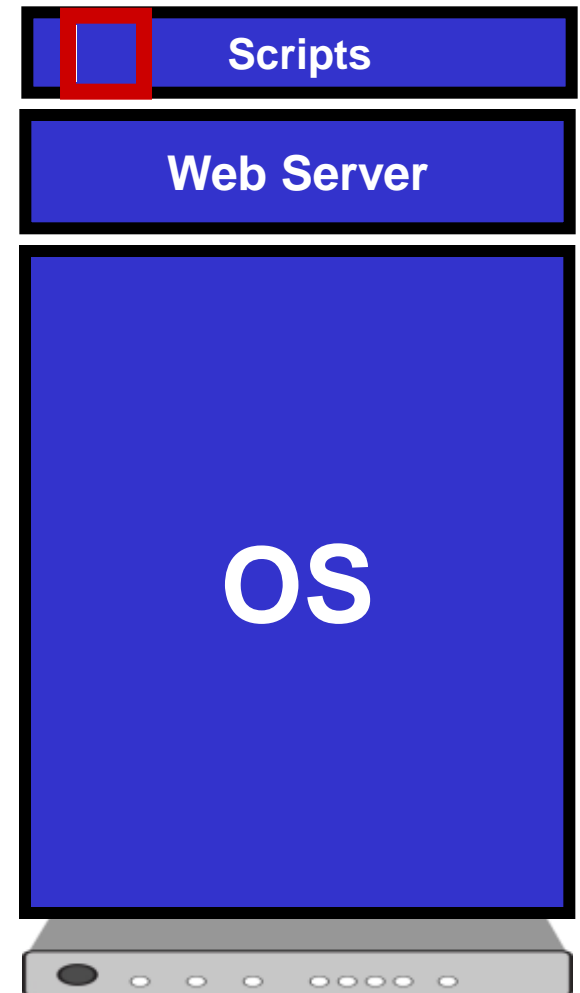
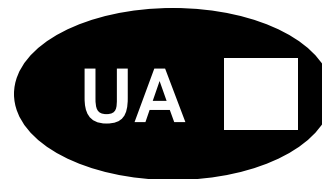
- Bundle everything the user *might touch* into an *isolated* environment
- Many applications continue to work if other users' data is absent
- Focus on *minimizing* developer effort

CLAMP Architecture Overview



User Authenticator (UA)

- Tags each server with an internal UserID
 - Tag is used by the Query Restrictor (QR)
- Prevents a compromised server from masquerading as another user
- We extract authentication code from web app
 - Very straightforward in our experience



Query Restrictor (QR)



QR

- Proxies all connections to the database
- Creates a *restricted view* of the database for each user
 - Uses the *same schema*, but omits other users' sensitive data
 - Intuition: Alice's server can generate webpages even if Bob's credit card is not in the database

Data Access Policies

QR

- Policy defines a user's restricted database
- For each table, specify an SQL "predicate" defining which data belongs to a user



| user | credit_card | expiration |
|-------|---------------------|------------|
| Alice | 4000 1820 1888 8600 | 09/2011 |
| Bob | 5588 3201 2345 6789 | 11/2009 |
| Carol | 4552 7204 1234 5678 | 02/2012 |
| Dave | 5490 1234 5678 9123 | 07/20 |

SELECT * FROM credit_card WHERE UserID = User

UA

UA

Creating Data Access Policies

- Difficult in theory: Private data looks just like Public data
- Typically *easy* in practice!
 - Schemas designed by humans for humans
 - Applications already encode the logic necessary to extract data relevant to a user

Case Study: osCommerce

- Open-source storefront used by *~14,000* online merchants
- Identified *~150 lines* of user auth
 - Easily found in 3 files, e.g., `login.php`
- Identified *7 tables* with sensitive data
 - Developed 3 policy files to describe restrictions
- Total time customizing CLAMP: *~2 hours*

Other Case Studies

- MyPhpMoney: Personal finance manager
 - Added **10 lines** of code for user auth
 - Two data access policies with **<20 lines**
 - Total time: **~3 hours**

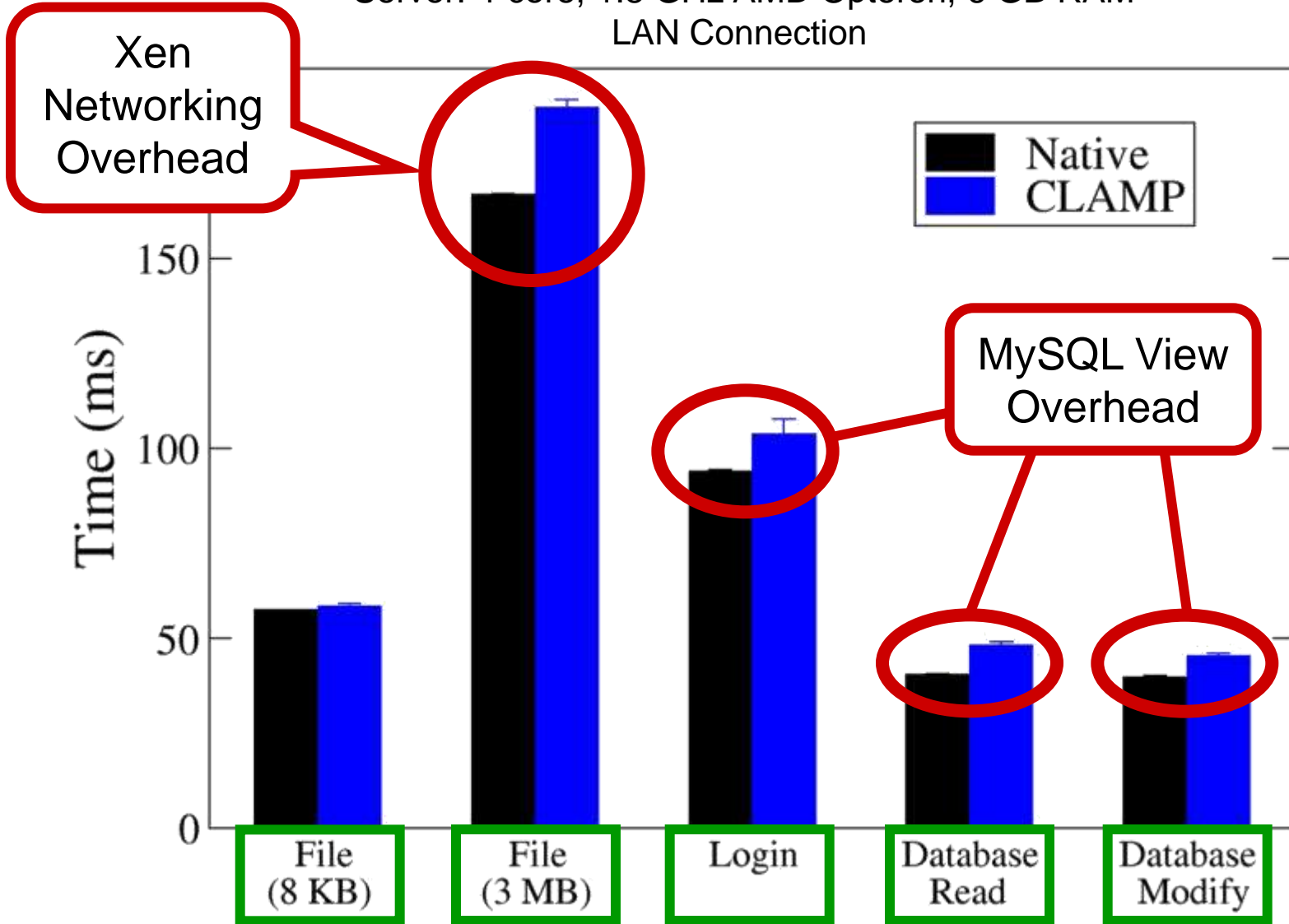


- HotCRP: Academic conference management
 - Added **6 lines** of code for user auth
 - Highly configurable policies required **~2 days**

A screenshot of the HotCRP sign-in page for the Oakland09 conference. The page has a white background with a black border. At the top left, it says "Oakland09 Sign in". On the top right, it shows the date and time: "Tuesday 28 Apr 2009 1:32:18pm EDT" and "Your local time: Tuesday 28 Apr 2009 1:50:21pm". The main content area is divided into two columns. The left column contains a paragraph of text: "Since 1980, the IEEE Symposium on Security and Privacy has been the premier forum for computer security research. Papers **must** be submitted in a form suitable for anonymous review." Below this text are two input fields: "Email" and "Password". The right column is titled "Conference information" and contains several blue hyperlinks: "Deadlines", "Program", "committee", "Conference site", and "26 papers were accepted out of".

Performance: Latency

Server: 4-core, 1.8 GHz AMD Opteron, 6 GB RAM
LAN Connection



Performance: Throughput

- Limited by VMM performance
 - 42% of native on connections/second
 - At most 8 logins/second vs. 85 on native
- Interpretation #1:
Choose an *alternate isolation* layer
 - E.g., SELinux, JVM, etc.
- Interpretation #2:
Performance *will improve* over time
 - Virtualization support constantly improving
 - VMs are embarrassingly parallel