Deadlock Immunity
Enabling Systems To Defend Against Deadlocks

Horatiu Jula, Daniel Tralamazza, Cristian Zamfir, George Candea

Dependable Systems Laboratory
Real Software System
Real Software System
Running

Failed

Real Software System
Real Software System
Real Software System
Real Software System
Real Software System
Real Software System
Running

Failed

Real Software System

time
Real Software with An Immune System
Real Software with An Immune System
Real Software with An Immune System
Real Software with An Immune System
Real Software with An Immune System
Real Software with An Immune System
Real Software with An Immune System
Real Software with An Immune System
Outline

- Immunity Against Deadlocks
- Dimmunix Overview
- Challenges & Solutions
- Discussion & Conclusions
Deadlock Immunity

- Learn executions that lead to deadlock
- Save fingerprints of encountered deadlock patterns into a persistent history
- Avoid executions patterns that have led to deadlock in the past
Example Uses

- **Software vendors**
  - *Example:* Protect a deadlock-free Web browser
    - Users extend browsers with plugins not controlled by the vendor... these can deadlock the browser
    - Deadlock occurs → obtain fingerprint → distribute it to all

- **End users**
  - *Example:* Deadlock-prone closed-source legacy DB server
    - Develop immunity against deadlocks
    - No need to upgrade or wait for patches
void nlShutdown(void)
{
    if(--nlInitCount > 0)
        return;
    if(nlSockets != NULL)
    {
        NLsocket i;
        for(i=0;i<nlNextsocket;i++)
        {
            if(nlSockets[i] != NULL)
            {
                driver->Close(i);
                nlReturnSocket(i);
                nlUnlockSocket(i, NL_BOTH);
                nlFreeSocket(i);
            }
        }
        free(nlSockets);
        nlSockets = NULL;
    }
    nlMutexUnlock(&socklock);
    if(driver != NULL)
    {
        driver->Shutdown();
        driver->initialized = NL_FALSE;
        driver = NULL;
    }
    else
    {
        ....
    }
}

NLboolean nlClose(NLsocket socket)
{
    if(driver)
    {
        if(nlIsValidSocket(socket) == NL_TRUE)
        {
            nlLockSocket(socket, NL_BOTH);
            driver->Close(socket);
            nlMutexLock(&socklock);
            nlReturnSocket(socket);
            nlMutexUnlock(&socklock);
            nlUnlockSocket(socket, NL_BOTH);
            return NL_TRUE;
        }
        else
        {
            nlSetError(NL_INVALID_SOCKET);
            return NL_TRUE;
        }
    }
    nlSetError(NL_NO_NETWORK);
    return NL_FALSE;
}

void nlShutdown(void)
{
    if(--nlInitCount > 0)
        return;
    nlMutexLock(&socklock);
    if(nlSockets != NULL)
    {
        NLsocket i;
        for(i=0;i<nlNextsocket;i++)
        {
            if(nlSockets[i] != NULL)
            {
                nlLockSocket(i, NL_BOTH);
                driver->Close(i);
                nlReturnSocket(i);
                nlUnlockSocket(i, NL_BOTH);
                nlFreeSocket(i);
            }
        }
        free(nlSockets);
        nlSockets = NULL;
    }
    nlMutexUnlock(&socklock);
    if(driver != NULL)
    {
        driver->Shutdown();
        driver->initialized = NL_FALSE;
        driver = NULL;
    }
    else
    {
        ....
    }
}
void nlShutdown(void)
{
  if(--nlInitCount > 0)
    return;
  if(nlSockets != NULL)
  {
    NLsocket i;
    for(i=0;i<nlNextsocket;i++)
    {
      if(nlSockets[i] != NULL)
      {
        driver->Close(i);
        nlReturnSocket(i);
        nlUnlockSocket(i, NL_BOTH);
        nlFreeSocket(i);
      }
    }
    free(nlSockets);
    nlSockets = NULL;
  }
  nlMutexUnlock(&socklock);
  if(driver != NULL)
  {
    driver->Shutdown();
    driver->initialized = NL_FALSE;
    driver = NULL;
  }
  else
  {
    ....
  }
}

NLboolean nlClose(NLsocket socket)
{
  if(driver)
  {
    if(nlIsValidSocket(socket) == NL_TRUE)
    {
      nlLockSocket(socket, NL_BOTH);
      driver->Close(socket);
      nlMutexLock(&socklock);
      nlReturnSocket(socket);
      nlMutexUnlock(&socklock);
      nlUnlockSocket(socket, NL_BOTH);
      return NL_TRUE;
    }
  }
  else
  {
    nlSetError(NL_INVALID_SOCKET);
    return NL_TRUE;
  }
  nlSetError(NL_NO_NETWORK);
  return NL_FALSE;
}
	nlMutexLock(&socklock);
	nlUnlockSocket(socket, NL_BOTH);
	nlMutexLock(&socklock);
Thread 1

nlLockSocket(socket, NL_BOTH);
nlMutexLock(&socklock);

Thread 2

nlMutexLock(&socklock);
nlLockSocket(socket, NL_BOTH);
Thread 1

nlLockSocket(socket, NL_BOTH);
nlMutexLock(&socklock);

Thread 2

nlMutexLock(&socklock);
nlLockSocket(socket, NL_BOTH);

Lock Inversion => Deadlock Bug
nlLockSocket(socket, NL_BOTH)
nlMutexLock(&socklock)
nlMutexLock(&socklock)
nlLockSocket(i, NL_BOTH)
Thread 1

nlClose(socket)

→ nlLockSocket(socket, NL_BOTH)

→ nlMutexLock(socks[socket]->lock)

→ pthread_mutex_lock(sock[socket]->lock->mutex)

→ nlMutexLock(&socklock)

→ pthread_mutex_lock(socklock->p_mutex)

Thread 2

nlShutdown()

→ nlMutexLock(&socklock)

→ pthread_mutex_lock(socklock->p_mutex)

→ nlLockSocket(i, NL_BOTH)

→ nlMutexLock(socks[i]->lock)

→ pthread_mutex_lock(socks[i]->lock->p_mutex)
Thread 1

\texttt{nlClose(socket)}

\texttt{nlLockSocket(socket, NL\_BOTH)}

\texttt{nlMutexLock(socks[socket]->lock)}

\texttt{pthread\_mutex\_lock(sock[socket]->lock->mutex)}

\texttt{nlMutexLock(&socketlock)}

\texttt{pthread\_mutex\_lock(socketlock->p\_mutex)}

\texttt{nlShutdown()}

Thread 2

\texttt{nlMutexLock(&socketlock)}

\texttt{pthread\_mutex\_lock(socketlock->p\_mutex)}

\texttt{nlLockSocket(i, NL\_BOTH)}

\texttt{nlMutexLock(socks[i]->lock)}

\texttt{pthread\_mutex\_lock(socks[i]->lock->p\_mutex)}
Thread 1

nlClose(socket)
  \rightarrow nlLockSocket(socket, NL_BOTH)
  \rightarrow nlMutexLock(socks[socket]->lock)
  \rightarrow pthread_mutex_lock(sock[socket]->lock->mutex)
  \rightarrow nlMutexLock(&socklock)
  \rightarrow pthread_mutex_lock(socklock->p_mutex)

Thread 2

nlShutdown()
  \rightarrow nlMutexLock(&socklock)
  \rightarrow pthread_mutex_lock(socklock->p_mutex)
  \rightarrow nlLockSocket(i, NL_BOTH)
  \rightarrow nlMutexLock(socks[i]->lock)
  \rightarrow pthread_mutex_lock(socks[i]->lock->p_mutex)
Deadlock Pattern

Thread 1

nlClose
- @ nl.c: 710
- nlLockSocket
  - @ nl.c: 347
- nlMutexLock
  - @ mutex.c: 105
  - pthread_mutex_lock
- nlMutexLock
  - @ mutex.c: 105
  - pthread_mutex_lock
- pthread_mutex_lock
  - mutex.c: 105
- @ nl.c: 722
- nlMutexLock
  - mutex.c: 105
  - pthread_mutex_lock
- nlMutexLock
  - mutex.c: 105
  - pthread_mutex_lock

Thread 2

nlShutdown
- @ nl.c: 553
- nlMutexLock
  - mutex.c: 105
  - pthread_mutex_lock
- pthread_mutex_lock
  - mutex.c: 105
  - pthread_mutex_lock
- nlLockSocket
  - @ nl.c: 347
- nlMutexLock
  - mutex.c: 105
  - pthread_mutex_lock
- nlMutexLock
  - mutex.c: 105
  - pthread_mutex_lock
- @ nl.c: 565

Thread 1

Thread 2
Deadlock Pattern

Thread 1

nlClose
  @ nl.c : 710
  → nlLockSocket
  @ nl.c : 347
  → nlMutexLock
  @ nl.c : 722
  → nlMutexLock
  @ mutex.c : 105
  → pthread_mutex_lock

Thread 2

nlShutdown
  @ nl.c : 553
  → nlMutexLock
  @ mutex.c : 105
  → pthread_mutex_lock
  @ nl.c : 565
  → nlLockSocket
  @ nl.c : 347
  → nlMutexLock
  @ mutex.c : 105
  → pthread_mutex_lock

Deadlock Bug
Deadlock Pattern

Thread 1
- nlClose @ nl.c: 710
  - nlLockSocket @ nl.c: 347
    - nlMutexLock @ mutex.c: 105
    - pthread_mutex_lock

Thread 2
- nlShutdown @ nl.c: 553
  - nlMutexLock @ mutex.c: 105
    - pthread_mutex_lock
- Threads

Deadlock Bug
Hangs
Deadlock Pattern

Thread 1
- nlClose
  @ nl.c: 710
- nlLockSocket
  @ nl.c: 347
- nlMutexLock
  @ mutex.c: 105
- pthread_mutex_lock
  @ mutex.c: 105
- nlclose
  @ nl.c: 722
- nlMutexLock
  @ mutex.c: 105
- pthread_mutex_lock
  @ mutex.c: 105

Thread 2
- nlShutdown
  @ nl.c: 553
- nlMutexLock
  @ mutex.c: 105
- pthread_mutex_lock
  @ mutex.c: 105
- nlLockSocket
  @ nl.c: 347
- nlMutexLock
  @ mutex.c: 105
- pthread_mutex_lock
  @ mutex.c: 105

Deadlock Bug
Hangs
Thread 1
- nlClose
  @ nl.c: 710
- nlLockSocket
  @ nl.c: 347
- nlMutexLock
  @ mutex.c: 105
- pthread_mutex_lock
  @ mutex.c: 105
- pthread_mutex_lock
  @ mutex.c: 105
- pthread_mutex_lock
  @ mutex.c: 105
- nlClose
  @ nl.c: 722
- nlMutexLock
  @ mutex.c: 105
- pthread_mutex_lock
  @ mutex.c: 105

Thread 2
- nlShutdown
  @ nl.c: 553
- nlMutexLock
  @ mutex.c: 105
- pthread_mutex_lock
  @ mutex.c: 105
- pthread_mutex_lock
  @ mutex.c: 105
- pthread_mutex_lock
  @ mutex.c: 105
- nlLockSocket
  @ nl.c: 347
- nlMutexLock
  @ mutex.c: 105
- pthread_mutex_lock
  @ mutex.c: 105

Deadlock Pattern
Deadlock Bug
Hangs
Deadlock Pattern

Thread 1
- nlClose
  - nlLockSocket
    - nlMutexLock
      - pthread_mutex_lock
  - @ nl.c : 710
- nlMutexLock
  - @ nl.c : 347
- @ mutex.c : 105
- @ mutex.c : 105
- @ mutex.c : 105
- @ nl.c : 722
- @ nl.c : 553
- @ nl.c : 347
- nlShutdown
  - @ nl.c : 553
  - nlMutexLock
    - @ mutex.c : 105
    - @ mutex.c : 105
  - @ nl.c : 565
  - @ nl.c : 347
  - nlLockSocket
    - nlMutexLock
      - pthread_mutex_lock

Thread 2
- @ nl.c : 347
- @ mutex.c : 105
- @ mutex.c : 105
- @ mutex.c : 105
- @ nl.c : 722
- @ nl.c : 565
- @ nl.c : 565
- @ nl.c : 553

Deadlock Bug
Hangs
Dimmunix

- **Intercepts** calls to lock/unlock
- **Detects** deadlocks automatically
- **Saves** the observed deadlock pattern
- **Avoids** executions that match saved patterns
Dimmunix

- **Intercepts** calls to lock/unlock
- **Detects** deadlocks automatically
- **Saves** the observed deadlock pattern
- **Avoids** executions that match saved patterns

History of patterns = immune system
Interception

- Two versions
  - Java: Direct instrumentation
  - POSIX Threads: Modified pthreads shared library

- Benefits
  - Zero assistance from programmers
  - Zero assistance from users
  - Zero need for source code
Deadlock Detection
Deadlock Detection

- $t_2$
- $l_2$
- $l_1$
- $t_1$

- Thread
- Lock
- Hold edge
- Allow edge
- Request edge
Deadlock Detection

Diagram:
- \( t_2 \) to \( l_2 \)
- \( l_1 \) to \( t_1 \)

Legend:
- \( \text{Thread} \)
- \( \text{Lock} \)
- \( \text{Hold edge} \)
- \( \text{Allow edge} \)
- \( \text{Request edge} \)
Deadlock Detection

Diagram showing threads and locks with edges indicating hold, allow, and request actions.
Deadlock Detection

Diagram:
- $t_2$ (Thread)
- $l_2$ (Lock)
- $l_1$ (Lock)
- $t_1$ (Thread)

Legend:
- Thread
- Lock
- Hold edge
- Allow edge
- Request edge
Deadlock Detection

$\text{t}_2 \xrightarrow{\text{Hold edge}} \text{l}_2 \xrightarrow{\text{Allow edge}} \text{t}_1 \xrightarrow{\text{Hold edge}} \text{l}_1 \xrightarrow{\text{Request edge}} \text{t}_2$

- **Thread**
- **Lock**
- **Hold edge**
- **Allow edge**
- **Request edge**
Fingerprinting Deadlocks

- $t_2$ waits for $l_2$ with $\text{n1Shutdown @nl.c:553}$
- $l_2$ waits for $t_2$ with $\text{n1MutexLock @mutex.c:105}$
- $l_1$ waits for $t_1$ with $\text{n1Close @nl.c:710}$
- $t_1$ waits for $l_1$ with $\text{n1LockSocket @nl.c:347}$
- $l_1$ waits for $t_1$ with $\text{n1MutexLock @mutex.c:105}$
- $t_1$ waits for $l_1$ with $\text{pthread_mutex_lock}$

- $t_2$ waits for $l_2$ with $\text{pthread_mutex_lock}$
Fingerprinting Deadlocks

\[
\begin{align*}
  t_2 & \xrightarrow{nl\text{Shutdown} \ @\ nl.c:553} l_2 \\
  l_1 & \xrightarrow{nl\text{Close} \ @\ nl.c:710} t_1
\end{align*}
\]

\[
\begin{align*}
  & \xrightarrow{nl\text{LockSocket} \ @\ nl.c:347} \\
  & \xrightarrow{nl\text{MutexLock} \ @\ mutex.c:105} \\
  & \xrightarrow{pthread\_mutex\_lock}
\end{align*}
\]
Fingerprinting Deadlocks

\[ t_2 \rightarrow l_2 \]

\[ n1Shutdown @nl.c:553 \]

\[ nlMutexLock @mutex.c:105 \]

\[ pthread_mutex_lock \]

\[ l_1 \rightarrow t_1 \]

\[ nlClose @nl.c:710 \]

\[ nlLockSocket @nl.c:347 \]

\[ nlMutexLock @mutex.c:105 \]

\[ pthread_mutex_lock \]
Fingerprinting Deadlocks

\[ t_2 \rightarrow nl\text{Close} @nl.c:710 \]
\[ nl\text{LockSocket} @nl.c:347 \]
\[ nl\text{MutexLock} @mutex.c:105 \]
\[ pthread\_mutex\_lock \]

\[ l_1 \rightarrow nl\text{Shutdown} @nl.c:553 \]
\[ nl\text{MutexLock} @mutex.c:105 \]
Fingerprinting Deadlocks

- $t_2$ to $l_2$: nlShutdown @nl.c:553, nlMutexLock @mutex.c:105
- $l_1$ to $t_1$: nlClose @nl.c:710, nlLockSocket @nl.c:347, nlMutexLock @mutex.c:105
- pthread_mutex_lock
Fingerprinting Deadlocks

- $t_1$ to $l_1$: `pthread_mutex_lock`
- $l_1$ to $t_2$: `nlLockSocket` @ `nl.c:347`
- $t_2$ to $l_2$: `nlClose` @ `nl.c:710`
- $l_2$ to $t_1$: `nlMutexLock` @ `mutex.c:105`
- $t_1$ to $l_1$: `nlMutexLock` @ `mutex.c:105`
- $l_1$ to $t_2$: `nlMutexLock` @ `mutex.c:105`
- $t_2$ to $l_2$: `nlShutDown` @ `nl.c:553`
Fingerprinting Deadlocks

- `nlShutdown` @ `nl.c:553`
- `nlMutexLock` @ `mutex.c:105`
- `pthread_mutex_lock`

- `nlClose` @ `nl.c:710`
- `nlLockSocket` @ `nl.c:347`
- `nlMutexLock` @ `mutex.c:105`
- `pthread_mutex_lock`
Fingerprinting Deadlocks

History of Signatures
Avoidance

Thread 1

- nlClose
- nlShutdown
- nlMutexLock
- pthread_mutex_lock

Thread 2

- nlClose
- nlLockSocket
- nlMutexLock
- pthread_mutex_lock
Avoidance

Thread 1

nlClose @ nl.c: 710
nlLockSocket

Thread 2

nlShutdown

nlMutexLock

nlClose @ nl.c: 710
nlLockSocket @ nl.c: 347
nlMutexLock @ mutex.c: 105
pthread_mutex_lock
Avoidance

Thread 1

nlClose
@ nl.c: 710

nlLockSocket

Thread 2

nlShutdown
@ nl.c: 553

nlMutexLock
@ mutex.c: 105

pthread_mutex_lock

nlClose
@ nl.c: 710

nlLockSocket
@ nl.c: 347

nlMutexLock
@ mutex.c: 105

pthread_mutex_lock
Avoidance

Thread 1
- nlClose @ nl.c: 710
  - nlLockSocket @ nl.c: 347
    - nlMutexLock

Thread 2
- nlShutdown @ nl.c: 553
  - nlMutexLock @ mutex.c: 105
- pthread_mutex_lock
- nlClose @ nl.c: 710
- nlLockSocket @ nl.c: 347
- nlMutexLock @ mutex.c: 105
- pthread_mutex_lock
Avoidance

Thread 1

nlClose
@ nl.c: 710
→ nlLockSocket
@ nl.c: 347
→ nlMutexLock
@ mutex.c: 105
→ pthread_mutex_lock

Thread 2

nlShutdown
@ nl.c: 553
→ nlMutexLock
@ mutex.c: 105
→ pthread_mutex_lock

nlClose
@ nl.c: 710
nlLockSocket
@ nl.c: 347
nlMutexLock
@ nl.c: 553
nlMutexLock
@ mutex.c: 105
pthread_mutex_lock
Avoidance

Thread 1

nlClose
@ nl.c : 710

nlLockSocket
@ nl.c : 347

nlMutexLock
@ mutex.c : 105

pthread_mutex_lock

Thread 2

nlShutdown
@ nl.c : 553

nlMutexLock
@ mutex.c : 105

pthread_mutex_lock

nlClose
@ nl.c : 710

nlLockSocket
@ nl.c : 347

nlMutexLock
@ mutex.c : 105

pthread_mutex_lock
Avoidance

Thread 1

- nlClose @ nl.c: 710
- nlLockSocket @ nl.c: 347
- nlMutexLock @ mutex.c: 105
- pthread_mutex_lock

Thread 2

- nlShutdown @ nl.c: 553
- nlMutexLock @ mutex.c: 105
- pthread_mutex_lock

nlClose @ nl.c: 710
nlLockSocket @ nl.c: 347
nlMutexLock @ mutex.c: 105
pthread_mutex_lock

nlShutdown @ nl.c: 553
nlMutexLock @ mutex.c: 105
pthread_mutex_lock
Avoidance

Thread 1

nlClose
@ nl.c: 710
→ nlLockSocket
@ nl.c: 347
→ nlMutexLock
@ nl.c: 553
→ pthread_mutex_lock

Thread 2

nlShutdown
@ nl.c: 553
→ nlMutexLock
@ nl.c: 105
→ pthread_mutex_lock

nlClose
@ nl.c: 710
nlLockSocket
@ nl.c: 347
nlMutexLock
@ nl.c: 105
pthread_mutex_lock
Avoidance

Thread 1
- nlClose @ nl.c: 710
- nlLockSocket @ nl.c: 347
  - nlMutexLock @ mutex.c: 105
  - pthread_mutex_lock @ nl.c: 722

Thread 2
- nlShutdown @ nl.c: 553
- nlMutexLock
Avoidance

Thread 1
- nlClose @ nl.c: 710
  - nlLockSocket @ nl.c: 347
    - nlMutexLock @ mutex.c: 105
  - @ nl.c: 722
- nlMutexLock

Thread 2
- nlShutdown @ nl.c: 553
- nlMutexLock @ mutex.c: 105
- pthread_mutex_lock

nlShutdown @nl.c:553
nlMutexLock @mutex.c:105
pthread_mutex_lock

nlClose @nl.c:710
nlLockSocket @nl.c:347
nlMutexLock @nl.c:105
pthread_mutex_lock
Avoidance

Thread 1

nlClose
@ nl.c: 710
→ nlLockSocket
@ nl.c: 347
→ nlMutexLock
@ mutex.c: 105
→ pthread_mutex_lock
@ nl.c: 722
→ nlMutexLock

nlShutdown
@ nl.c: 553
→ nlMutexLock
@ mutex.c: 105
→ pthread_mutex_lock

Thread 2

nlShutdown
@ nl.c: 553
→ nlMutexLock
@ mutex.c: 105
→ pthread_mutex_lock
Avoidance

Thread 1
- nlClose
  @ nl.c: 710
  - nlLockSocket
    @ nl.c: 347
      - nlMutexLock
        @ mutex.c: 105
          - pthread_mutex_lock
            @ nl.c: 722
            - nlMutexLock

Thread 2
- nlShutdown
  @ nl.c: 553
  - nlMutexLock
    @ mutex.c: 105
      - pthread_mutex_lock
Avoidance

Thread 1
- nlClose
  @ nl.c: 710
  - nlLockSocket
    @ nl.c: 347
      - nlMutexLock
        @ mutex.c: 105
        → pthread_mutex_lock
  @ nl.c: 722
  - nlMutexLock

Thread 2
- nlShutdown
  @ nl.c: 553
  - nlMutexLock
    @ mutex.c: 105
    → pthread_mutex_lock

- nlClose
  @ nl.c: 710
- nlLockSocket
  @ nl.c: 347
- nlMutexLock
  @ mutex.c: 105
- pthread_mutex_lock
Avoidance

Thread 1
- nlClose @ nl.c: 710
  - nlLockSocket @ nl.c: 347
    - nlMutexLock @ mutex.c: 105
      - pthread_mutex_lock
  - nlMutexLock @ nl.c: 722
- nlMutexLock @ mutex.c: 105
- pthread_mutex_lock

Thread 2
- nlShutdown @ nl.c: 553
  - nlMutexLock @ mutex.c: 105
    - pthread_mutex_lock
  - nlClose @ nl.c: 710
    - nlLockSocket @ nl.c: 347
      - nlMutexLock @ mutex.c: 105
        - pthread_mutex_lock

Avoidance

Thread 1

nlClose
@ nl.c: 710
→ nlLockSocket
@ nl.c: 347
→ nlMutexLock
@ mutex.c: 105
→ pthread_mutex_lock
@ nl.c: 553
→ nlMutexLock
@ mutex.c: 105
→ pthread_mutex_lock
→ pthread_mutex_unlock

Thread 2

nlShutdown
@ nl.c: 553
→ nlMutexLock
@ mutex.c: 105
→ pthread_mutex_lock

nlClose
@ nl.c: 710
→ nlLockSocket
@ nl.c: 347
→ nlMutexLock
@ mutex.c: 105
→ pthread_mutex_lock
Avoidance

Thread 1

nlClose
@ nl.c: 710

nlLockSocket
@ nl.c: 347

nlMutexLock
@ mutex.c: 105

@ nl.c: 722

nlMutexLock
@ mutex.c: 105

pthread_mutex_lock

Thread 2

nlShutdown
@ nl.c: 553

nlMutexLock
@ mutex.c: 105

nlLockSocket
@ nl.c: 565

nlMutexLock
@ mutex.c: 105

pthread_mutex_lock
Avoidance

Thread 1

nlClose
@ nl.c: 710
  → nlLockSocket
    @ nl.c: 347
      → nlMutexLock
        @ mutex.c: 105
          → pthread_mutex_lock
            @ nl.c: 722
              → nlMutexLock
                @ mutex.c: 105
                  → pthread_mutex_lock

Thread 2

nlShutdown
@ nl.c: 553
  → nlMutexLock
    @ mutex.c: 105
      → pthread_mutex_lock
        @ nl.c: 565
          → nlLockSocket
            @ nl.c: 347
              → nlMutexLock
Avoidance

Thread 1
- nlClose
  @ nl.c: 710
  → nlLockSocket
    @ nl.c: 347
    → nlMutexLock
      @ mutex.c: 105
      → pthread_mutex_lock
    @ nl.c: 722
    → nlMutexLock
      @ mutex.c: 105
      → pthread_mutex_lock
  → pthread_mutex_unlock

Thread 2
- nlShutdown
  @ nl.c: 553
  → nlMutexLock
    @ mutex.c: 105
    → pthread_mutex_lock
    @ nl.c: 565
    → nlLockSocket
      @ nl.c: 347
      → nlMutexLock
        @ mutex.c: 105
        → pthread_mutex_lock
Does It Work?

C/C++
- MySQL
- SQLite
- HawkNL

Java
- Apache ActiveMQ
- Limewire
- MySQL JDBC
- Sun JDK
<table>
<thead>
<tr>
<th>System</th>
<th>Bug#</th>
<th>Deadlock Bug Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MySQL 6.0.4</td>
<td>37080</td>
<td>INSERT and TRUNCATE in two different threads</td>
</tr>
<tr>
<td>SQLite 3.3.0</td>
<td>1672</td>
<td>Race condition in custom recursive lock implementation</td>
</tr>
<tr>
<td>HawkNL 1.6b3</td>
<td>n/a</td>
<td>nlShutdown() called concurrently with nlClose()</td>
</tr>
<tr>
<td>Limewire 4.17.9</td>
<td>1449</td>
<td>Cancel HsqlDB TaskQueue concurrently with program shutdown()</td>
</tr>
<tr>
<td>ActiveMQ 3.1</td>
<td>336</td>
<td>Concurrently create listener and actively dispatch messages to consumer</td>
</tr>
<tr>
<td>ActiveMQ 4.0</td>
<td>575</td>
<td>Concurrent invocation of Queue.dropEvent() and PrefetchSubscription.add()</td>
</tr>
<tr>
<td>MySQL 5.0 JDBC</td>
<td>2147</td>
<td>Connection.close() called concurrently with PreparedStatement.getWarnings()</td>
</tr>
<tr>
<td></td>
<td>14972</td>
<td>Connection.prepareStatement() called concurrently with Statement.close()</td>
</tr>
<tr>
<td></td>
<td>31136</td>
<td>PreparedStatement.executeQuery() called concurrently with Connection.close()</td>
</tr>
<tr>
<td></td>
<td>17709</td>
<td>Statement.executeQuery() concurrent with Connection.prepareStatement()</td>
</tr>
<tr>
<td>System</td>
<td>Bug#</td>
<td>Deadlock Bug Description</td>
</tr>
<tr>
<td>-------------------</td>
<td>-------</td>
<td>----------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>MySQL 6.0.4</td>
<td>37080</td>
<td>INSERT and TRUNCATE in two different threads</td>
</tr>
<tr>
<td>SQLite 3.3.0</td>
<td>1672</td>
<td>Race condition in custom recursive lock implementation</td>
</tr>
<tr>
<td>HawkNL 1.6b3</td>
<td>n/a</td>
<td>nlShutdown() called concurrently with nlClose()</td>
</tr>
<tr>
<td>Limewire 4.17.9</td>
<td>1449</td>
<td>Cancel HsqlDB TaskQueue concurrently with program shutdown()</td>
</tr>
<tr>
<td>ActiveMQ 3.1</td>
<td>336</td>
<td>Concurrently create listener and actively dispatch messages to consumer</td>
</tr>
<tr>
<td>ActiveMQ 4.0</td>
<td>575</td>
<td>Concurrent invocation of Queue.dropEvent() and PrefetchSubscription.add()</td>
</tr>
<tr>
<td>MySQL 5.0 JDBC</td>
<td>2147</td>
<td>Connection.close() called concurrently with PreparedStatement.getWarnings()</td>
</tr>
<tr>
<td></td>
<td>14972</td>
<td>Connection.prepareStatement() called concurrently with Statement.close()</td>
</tr>
<tr>
<td></td>
<td>31136</td>
<td>PreparedStatement.executeQuery() called concurrently w/ Connection.close()</td>
</tr>
<tr>
<td></td>
<td>17709</td>
<td>Statement.executeQuery() concurrent with Connection.prepareStatement()</td>
</tr>
</tbody>
</table>

Avoids real deadlocks in real systems
Challenges

1. Performance
2. Induced Starvation
3. Precision Calibration

See paper for...
- Weak vs. strong immunity
- Software upgrades
- Evaluation of false positives
1. Performance

- **Potential overhead**
  - Intercept every call to lock/unlock
  - Update resource allocation graph
  - Find cycles + make suspend/resume decisions

- **Solution: asynchronous architecture**
  - Do expensive work in a separate thread
  - Take advantage of multi-core CPU
  - Lock-free data structures
Application Thread

lock(L)

unlock(L)

Avoidance

decide

request

acquired

release

go / yield

RAG cache

History

Monitor Thread

async event queue

lock-free

periodically do:

- process events
- search for cycles
- if cycles found
- save to history
1. Performance

![Graph showing lock operations and thread suspends per second vs number of threads. Baseline and Dimmunix data points are plotted.]

- **Lock Operations / Second**: The graph plots lock operations per second against the number of threads.
- **Thread Suspends / Second**: A separate axis plots thread suspends per second.
- **Baseline**: Blue squares represent the baseline performance.
- **Dimmunix**: Red squares indicate Dimmunix performance.
- **Sends**: Grey squares show thread suspends per second.

The graph illustrates how the performance changes with an increasing number of threads, highlighting the efficiency of Baseline and Dimmunix compared to thread suspends.
1. Performance

![Graph showing performance metrics]

- Lock Operations / Second
- Thread Suspends / Second

- Number of Threads

- Baseline
- Dimmunix
- Suspends
1. Performance

![Graph showing Lock Operations vs Number of Threads]

- **Baseline**
- **Dimmunix**
- **Suspends**

**Y-axis:** Lock Operations / Second
**X-axis:** Number of Threads

**Legend:**
- Blue square markers for Baseline
- Orange square markers for Dimmunix
- Grey square markers for Suspends

The graph illustrates the performance comparison between different thread management methods. As the number of threads increases, the lock operations per second and thread suspends per second are monitored. The Baseline method shows a steady performance, while Dimmunix maintains a high efficiency, and the Suspends metric indicates a significant increase in thread suspends as the number of threads grows.
1. Performance

![Graph showing performance metrics]

- **Lock Operations / Second**
- **Thread Suspends / Second**

**X-axis:** Number of Threads

**Y-axis:**
- Lock Operations / Second (0 to 9000)
- Thread Suspends / Second (0 to 250)

Legend:
- Baseline
- Dimmunix
- Suspend
1. Performance

![Graph showing performance over number of threads]

- **Baseline**
- **Dimmunix**
- **Suspends**

**Lock Operations / Second** vs. **Number of Threads**

**Thread Suspends / Second**
1. Performance

![Graph showing performance metrics](image)
1. Performance

![Graph showing performance with overhead range]

Overhead = 0.6% - 4.5%
1. Performance

- Matching signatures from history
  - Hash call stacks and index into precomputed tables
  - Thread-local cache of data structures
1. Performance

- Matching signatures from history
  - Hash call stacks and index into precomputed tables
  - Thread-local cache of data structures

![Graph showing performance of Lock Operations per Second vs. Number of Signatures in History]
1. Performance

- Matching signatures from history
  - Hash call stacks and index into precomputed tables
  - Thread-local cache of data structures
1. Performance

- Matching signatures from history
  - Hash call stacks and index into precomputed tables
  - Thread-local cache of data structures

![Graph showing lock operations per second vs. number of signatures in history. The graph compares Baseline and Dimmunix.]
2. Induced Starvation

- All approaches based on yield/suspend can induce starvation
  - i.e., a thread actively yields, waiting for a blocked thread to make progress

- Never encountered in our tests
Thread
Lock
Hold edge
Request edge
Yield edge

nl_shutdown@nl.c:551
nl_mutex_lock@mutex.c:105
pthread_mutex_lock

nl_close@nl.c:440
nl_lock_socket@nl.c:347
nl_mutex_lock@mutex.c:105
pthread_mutex_lock

pthread_mutex_lock
nl_mutex_lock@mutex.c:105
pthread_mutex_lock

pthread_mutex_lock
nl_mutex_lock@mutex.c:105
pthread_mutex_lock
Thread
Lock
Hold edge
Request edge
Yield edge

nlShutdown @nl.c:551
nlMutexLock @mutex.c:105
pthread_mutex_lock

nlClose @nl.c:440
nlLockSocket @nl.c:347
nlMutexLock @mutex.c:105
pthread_mutex_lock

nlShutdown @nl.c:551
nlMutexLock @mutex.c:105
pthread_mutex_lock

nlMutexLock @mutex.c:105
pthread_mutex_lock

nlMutexLock @mutex.c:105
pthread_mutex_lock
Thread

Lock

Hold edge

Request edge

Yield edge

nlShutdown @nl.c:551
nlMutexLock @mutex.c:105
pthread_mutex_lock

nlClose @nl.c:440
nlLockSocket @nl.c:347
nlMutexLock @mutex.c:105
pthread_mutex_lock
Detect & avoid starvation as if it was a deadlock
3. Precision Calibration

• False positive = avoiding a deadlock that would not have occurred

• Two sources:
  • Matching signatures too generally
  • Input dependencies
3. Precision Calibration

- **Solution: Automatic precision adjustment**
  - Post-factum what-if analysis
  - Heuristic: look for potential lock inversions

- **Input dependencies not captured**
  - Dimmunix focused 100% on control flow
  - This is a requisite for portability of signatures, i.e., for immunity
Outline

- Immunity Against Deadlocks
- Dimmunix Overview
- Challenges & Solutions
- Discussion & Conclusions
Dimmunix Properties

1. Someone will experience first occurrence
   - Afterward can vaccinate all others
2. Cannot affect deadlock-free programs
3. Cannot induce incorrect outputs (non-RT)
4. Must be aware of all synch mechanisms
Related Work

- **Complements other reliability techniques**
  - Static analysis [Flanagan & Leino] [Engler & Ashcraft]
  - Modelchecking [Henzinger et al.] [Havelund et al.]
  - Transactional Memory [Herlihy & Moss]

- **Improves upon previous approaches by increasing avoidance precision**
  - Program transformations [Boronat & Cholvi]
  - Ghostlocks [Zeng & Martin]
  - Gatelocks [Buchbinder et al.]
Dimmunix

- Failure immunity applied to deadlocks
- Effective for both Java and C/C++
- Low overhead (4.5% for 1024 threads)
- Needs no assistance & no source code
- Can use as band-aid and vaccine

http://dimmunix.epfl.ch/