Erlang並行システムでのアプリケーションセキュリティ

力武 健次
NICT インシデント対策グループ

2008年10月9日

本日の話題

- マルチコア時代のセキュリティ要件
- プログラミング言語Erlangの特徴
- Erlangによる並行プログラミング
- Erlangのセキュリティ機能と問題
- 今後Erlang環境上でどのようにセキュリティを強化していくかの考察

マルチコア時代のセキュリティ要件(1)

- CPU単独での性能向上は限界
  - エネルギー消費と放熱の制約
- 単一システムでもマルチコア化
  - Intel Core2Duo/Quadの普及
  - AMD Opteronの普及
- より細かい粒度の並行処理へ
  - ホスト→OSプロセス→OSスレッド

マルチコア時代のセキュリティ要件(2)

- 大量CPUコア同時使用が前提へ
  - 数十数百のCPUで1つの仕事をする
- スレッドプログラミングの難しさ
  - 共有メモリの排他制御が不完全
  - ロックによる性能の低下
- 既存の並列処理環境では不十分
  - 共有メモリの問題は解決できていない

マルチコア時代のセキュリティ要件(3)

- 非同期処理のバグによる脆弱性
  - deadlock, race condition
  - 共有メモリへの不正情報挿入
  - スレッド間のアクセス管理は難しい
- →根本的な発想の転換が必要?
  - 共有メモリをやめてメッセージ交換へ
  - 実行単位間を最大限隔離する

プログラミング言語Erlangの特徴

- 各変数には一度しか代入できない
  - "Ｘ＝Ｘ＋１"はエラー
- プロセス間ではメモリは共有しない
  - メッセージ交換のみで情報共有
- 超軽量プロセスモデル（スレッド相当）
  - OSのスレッドより高速に生成かつ軽量
- OS独立の仮想マシンで資源管理

Erlangでの並行処理例(1)

- 単一コード複数データモデル(SPMD)
  - 同じプログラムを複数データに適用
  - 例: 数値積分の台形公式
    - 積分区間を分割し，それぞれに同一の演算を適用して，その結果を集めて足し合わせる
- Erlangではリスト処理として実行可

Erlangでの並行処理例(2)

- 逐次反復実行=リスト要素毎処理
- 上記処理の並行化
  - –リストを分割しプロセスに割り当てる
  - –各プロセスでは逐次適用(map)
    - プロセスごとの要素数は任意
- 関数名を並行処理用に書き換えるだけで並行処理が可能になる

Erlangでの並行処理例(3): 逐次コード

```
-module(pispmd).
-import(lists, [sum/1, map/2, seq/2]).
-import(nsplit, [npmap/3]).
-export([pitest/2]).
%% map: sequential mapping
%%
pitest(P, N) -> sum(map(
                    fun(X) -> sqinv(X, N) end,
                    seq(0, (N-1)))) / float(N).
%%
sqinv(X, N) ->
      Y = (X + 0.5) / float(N),
      4.0 / (1.0 + (Y * Y)).
```

Erlangでの並行処理例(4): 並列コード

```
-module(pispmd).
-import(lists, [sum/1, map/2, seq/2]).
-import(nsplit, [npmap/3]).
-export([pitest/2]).
%% npmap: n-process parallel mapping
%% P: number of processes
pitest(P, N) -> sum(npmap(P,
                          fun(X) -> sqinv(X, N) end,
                          seq(0, (N-1)))) / float(N).
%%
sqinv(X, N) ->
      Y = (X + 0.5) / float(N),
      4.0 / (1.0 + (Y * Y)).
```

処理速度と同時プロセス数

`pispmd:pitest(P,1000000)` の実行結果

実行時間 [秒]	P: 同時プロセス数		
	1000	10000	100000
単一ノード	19.82	7.44	9.92
デュアルノード	14.83	7.00	21.75

- CPU: Intel Core2Duo 875MHz（FreeBSD powerdで速度制限）
- メモリ: 1.5Gbytes / ノード間は100BASE-TXで結合
- OS: FreeBSD 6.3-RELEASE + 7.0-RELEASE
- Erlang/OTP R12B3を使用
- Erlang VM: SMP有効，スケジューラ2個/ノード，HiPE有効

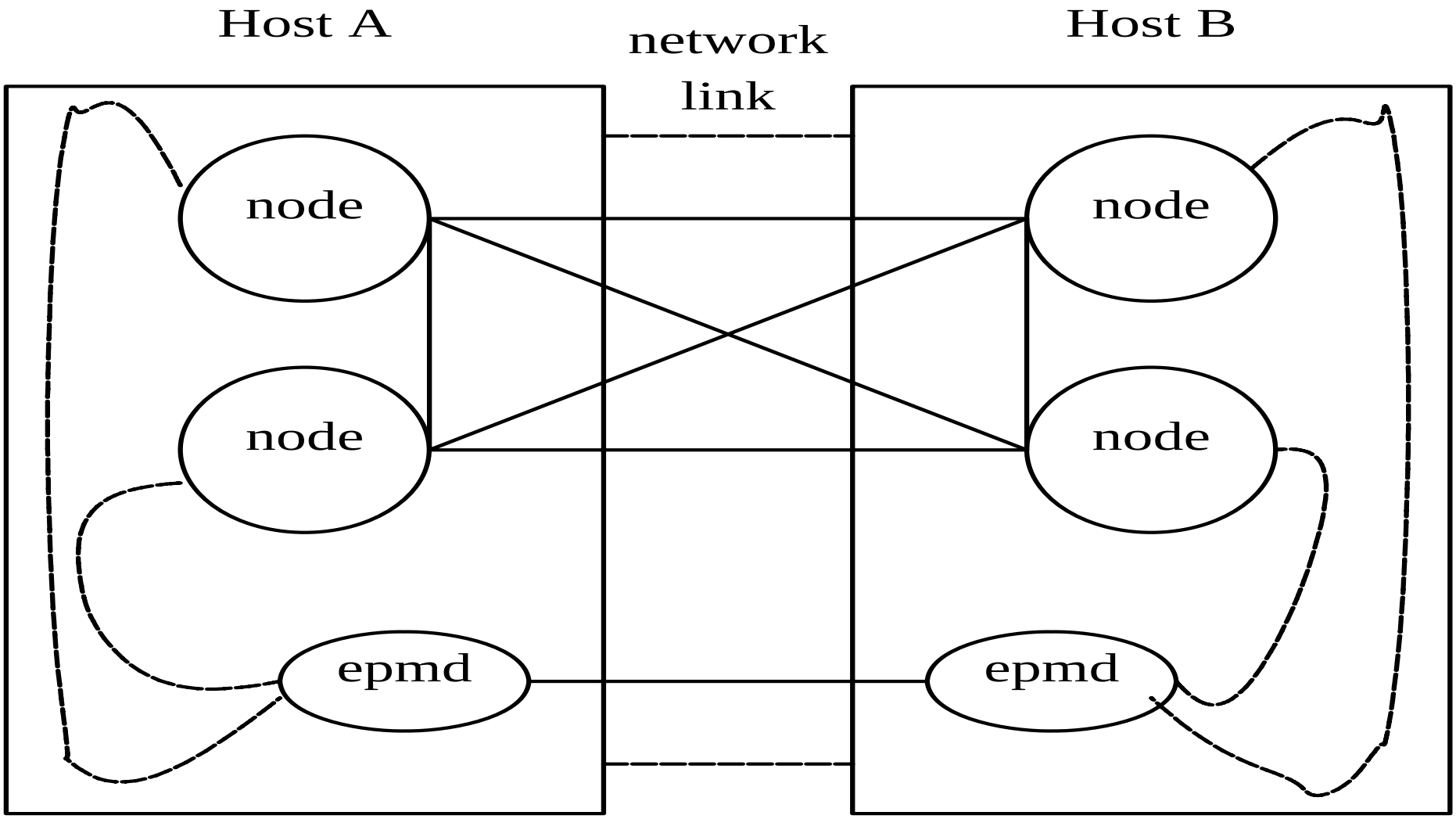
処理速度の決定要素

- 並行化のための処理分割計算量
- プロセス生成と消滅の量と時間
  - 通常同時に数万個程度
- 各プロセスの計算量
  - 大きければオーバヘッドの割合は減る
- プロセス間通信の量と時間
  - メッセージ伝達には時間がかかる

Erlang/OTPの通信モデル

- Open Telecom Platform (OTP)
  - RPCによる複数ノード間の分散処理
    - 資源割当を自動的に行える
  - 耐障害性と処理性能を重視
  - ノード間はメッシュ結合が前提
  - 各ホストは複数ノードを持てる
    - `n1@h1, n2@h2.example.org`

Erlang/OTPのノード間接続例



fully-connected mesh network between nodes
and control link between epmd daemons

Erlang/OTPのセキュリティ(1)

- 同じ内容のcookieかどうかで認証
- RPCマッパ(epmd)を使う
  - 暗号化なし，接続開始時認証のみ
- 通信そのものはVM間で行う
  - トランスポートは選べる，が…
  - 何もしなければ暗号化なしの単純なTCPトランスポートが使われる

Erlang/OTPのセキュリティ(2)

- 暗号化の機能は極めて弱い
  - epmdの暗号化機能はない
  - slave:startによるリモートシェル
    認証でSSHは使えるが，VM間とは別
  - VM間通信はSSLが使える
    - inet_ssl_distドライバ
      - 証明書がないと起動すらしなかった!
    - 同じ内容SSL証明書が各ノードに必要

Erlang/OTPのセキュリティ(3)

- epmdは攻撃に弱い
  - INADDR_ANYにbind()されている
  - ポート番号が既知: DoSに弱い
- epmdの通信の盗聴でVMが攻撃可
  - VM間TCPポート番号はepmdで決定
- ファイアウォール通過制御が難しい
  - RPCやP2P,VoIPなどと同様

Erlang/OTPのセキュリティ(4)

- SSL通信は外部プロセスに依存
  - SSLの処理は並行化できない
- SSL処理が軽ければ無視できる

`pispmd:pitest(P,1000000)` の実行結果

デュアルノード	P: 同時プロセス数		
実行時間[秒]	1000	10000	CPU利用率: (100%/コア) VM: 80% SSL: 10%
SSL未使用	14.58	7.04	
SSLを使用	14.13	6.98	

●Erlang/OTP R12B4を使用

今後の課題(1)

- Erlang/OTP RPCの暗号化対応
  - epmdでの暗号化機能実現
  - 暗号化機能設定の一元化
- 他の暗号化トランスポートでの実装
  - SSH, Secure HTTP, Jabber/XMPP
- 暗号化通信の並行化対応
  - Erlang言語自身による暗号化の実装

今後の課題(2)

- 異なるOSが混在した環境での実験
  - Windows, Linux, FreeBSD
  - 現在でもErlang/OTPは同様に動く
- IPv6環境での実験
  - ErlangはIPv6に対応している，が
  - SSLなどのテストが十分ではない
- OTPのRPC以外の接続手法の検討