

Password Cracking

Fun with Erlang Concurrency

Don't Be Evil!

- For demonstration purposes only
- Not production-ready

Er-what?

Er-what?

- What: Functional language, not Object- Oriented

Er-what?

- What: Functional language, not Object- Oriented
- Who: Ericsson Computer Science Laboratory
- Where: erlang.org. This code: search “erlang_md5crack” on github.com

Why Erlang?

Why Erlang?

- Concurrency

Why Erlang?

- Concurrency
- Distributed

Why Erlang?

- Concurrency
- Distributed
- Designed for failure

Why Erlang?

- Concurrency
- Distributed
- Designed for failure
- Hot-upgradable code

Erlang 101 in 7 minutes

Variables

- They don't vary - really, never

Variables

- ⦿ They don't vary - really, never
- ⦿ $B = 1$ is a statement of fact, not an assignment

Variables

- ⦿ They don't vary - really, never
- ⦿ $B = 1$ is a statement of fact, not an assignment
- ⦿ Mathematically $B = B + 1$ makes no sense, same in Erlang

Variables

- ⦿ They don't vary - really, never
- ⦿ $B = 1$ is a statement of fact, not an assignment
- ⦿ Mathematically $B = B + 1$ makes no sense, same in Erlang
- ⦿ $B = 2$ is similarly illegal

```
1> B = 1.  
1  
2> B = 3 - 2.  
1  
3> B = 2.  
** exception error: no match of right hand side value 2  
4> █
```

Data types

- Number (integer and float)

Data types

- ⦿ Number (integer and float)
- ⦿ Atom (literal):
 - ⦿ hello
 - ⦿ kitty

Tuples

- ⦿ {Term1,.....,TermN}
- ⦿ Shape1 = {square,4}
- ⦿ Shape2 = {circle,5}

Lists

- $X = [1, 2, 3].$
- $Y = [an_atom, \{tuple, tuple\}, "hello", 9.45].$
- $[Z | Rest] = X.$

```
4> X = [1,2,3].  
[1,2,3]  
5> [Z|Rest] = X.  
[1,2,3]  
6> Z.  
1  
7> Rest.  
[2,3]  
8>
```

String: syntactic sugar

- ⦿ An array of integers with ASCII values
- ⦿ Not a data type

```
25> S = [100,111,103].  
"dog"  
26> T = [100,111,103,999].  
[100,111,103,999]  
27> U = "dog".  
"dog"  
28> [V|_] = U.  
"dog"  
29> V.  
100 _
```

Pattern Matching and Recursion

```
countdown(0) ->
    io:format("~n");
countdown(N) ->
    io:format("~p~n", [N]),
    countdown(N-1).
```

```
5> c(misc).
{ok,misc}
6> misc:countdown(3).
3
2
1
0
ok
7> █
```

Fun() with anonymous functions

```
1> Doubler = fun(N) -> 2*N end.  
#Fun<erl_eval.6.13229925>  
2> Doubler(2).  
4  
3> Doubler(15).  
30
```

Concurrent Processes

- Pid = spawn(Fun).

Concurrent Processes

- `Pid = spawn(Fun)` :spawns new function referenced by Pid
- `Pid ! Message` :sends message to process

Concurrent Processes

- `Pid = spawn(Fun)` :spawns function as a process referenced by Pid
- `Pid ! Message` :sends message to process
- `receive...end` :block used by process to receive messages

```
loop() ->
    receive
        {double,N} ->
            io:format("Double ~p is ~p~n",[N, N*2]),
            loop();
        {triple,N} ->
            io:format("Triple ~p is ~p~n",[N, N*3]),
            loop();
        close ->
            io:format("closing...~n");
        Other ->
            io:format("Don't know how to ~p~n",[Other]),
            loop()
    end.
```

```
| 5> Pid = spawn(fun() -> misc:loop() end).  
<0.44.0>
```

```
5> Pid = spawn(fun() -> misc:loop() end).  
<0.44.0>  
6> Pid ! {double,2}.  
Double 2 is 4  
{double,2}  
7> Pid ! {triple,2}.  
Triple 2 is 6  
{triple,2}
```

```
5> Pid = spawn(fun() -> misc:loop() end).  
<0.44.0>  
6> Pid ! {double,2}.  
Double 2 is 4  
{double,2}  
7> Pid ! {triple,2}.  
Triple 2 is 6  
{triple,2}  
8> Pid ! say_fail.  
Don't know how to say_fail  
say_fail  
9> Pid ! close.  
closing...  
close
```

Back to the
Passwords

Problem

- Simple, non-reversible password creation by MD5 algorithm

Problem

- Simple, non-reversible password creation by MD5
- `MD5.hexdigest("hi") = 49f68a5c8493ec2c0bf489821c21fc3b`

Problem

- Simple, non-reversible password creation by MD5
- `MD5.hexdigest("hi") =`
`49f68a5c8493ec2c0bf489821c21fc3b`
- How to determine plaintext when given hash?

Bruteforce Solution

- Check the MD5 hash of every possible string. When hashes match, password found.

Check all strings
between “aa” and “zz”

```
irb(main):005:0> MD5.hexdigest("aa")
=> "4124bc0a9335c27f086f24ba207a4912"
irb(main):006:0> MD5hexdigest("hh")
=> "5e36941b3d856737e81516acd45edc50"
irb(main):007:0> MD5hexdigest("hi")
=> "49f68a5c8493ec2c0bf489821c21fc3b"
```

Exponential Time

- Length 1: $26^1 = 26$ possibilities

Exponential Time

- Length 1: $26^1 = 26$ possibilities
- Length 2: $26^2 = 676$

Exponential Time

- Length 1: $26^1 = 26$ possibilities
- Length 2: $26^2 = 676$
- Length 3: $26^3 = 17,576$
- Length 6: 308,915,776
- Length 7: 8,031,810,176
- Length 8: 5.4 trillion +

Strategy: Divide & Conquer

- 3-letters, single process = search from ‘aaa’ to ‘zzz’

Strategy: Divide & Conquer

- 3-letters, single process = search from ‘aaa’ to ‘zzz’
- 3-letters, 2 processes: ‘aaa’ to ‘mzz’,
‘naa’ to ‘zzz’

Strategy: Divide & Conquer

- 8 processes:

```
[{"aaa","dgm"},  
 {"dgn","gmz"},  
 {"gna","jtm"},  
 {"jtn","mzz"},  
 {"naa","qgm"},  
 {"qgn","tmz"},  
 {"tna","wtm"},  
 {"wtm","zzz"}]
```

Find first letter for string length

- a = 0, z = 25
- aa = 26 (not [0],[0])
- aaa = 702

Find first letter for string length

- $a = 0, z = 25$
- $aa = 26$
- $aaa = 702$
- $\text{first}(3) = 26^2 + 26^1$
- recursively calc 26^{n-1} until $n = 0$

Find first letter for string length

```
first_int(Len) when Len > 0 ->
    first_int(Len-1, 0).

first_int(0, Acc) ->
    Acc;
first_int(Len, Acc) ->
    first_int(Len - 1, Acc + round(math:pow(26, Len))).
```

Divide and Conquer II

- 3 letters, 3 processes:
 - $17576 / 3 = 5858.667$ strings / proc
 - 702 to 6560
 - 6561 to 12419
 - 12420 to 18277

Number to Char Array

- Use hexavigesimal math

Number to Char Array

- Use hexavigesimal math

```
chr_array(I) ->
    chr_array(I, []).|  
  
chr_array(I, L) when I > 25 ->
    R = I rem 26,
    D = I div 26,
    chr_array(D - 1, [R+$a|L]);
chr_array(I, L) ->
    [I + $a|L].
```

- $\text{chr_array}(702) = [97, 97, 97] = \text{"aaa"}$

Spawn N processes

```
decrypt(Crypted, Len, Processes) ->
    CharPartitions = partition_alphabet(Len, Processes),
    Server = self(),
    StartTime = now(),
    lists:foreach(fun({Min,Max}) ->
        spawn(fun() ->
            analyze(Server, Crypted, Min, Max, Len) end)
    end, CharPartitions),
    loop(Processes, StartTime, notfound).
```

Client analyze function

```
analyze(Server, Crypted, Cur, Max, Len) ->
  case erlang:md5(Cur) of
    Crypted ->
      Server ! {found, Cur};
    _ ->
      analyze(Server, Crypted, next(Cur), Max, Len)
  end.
```

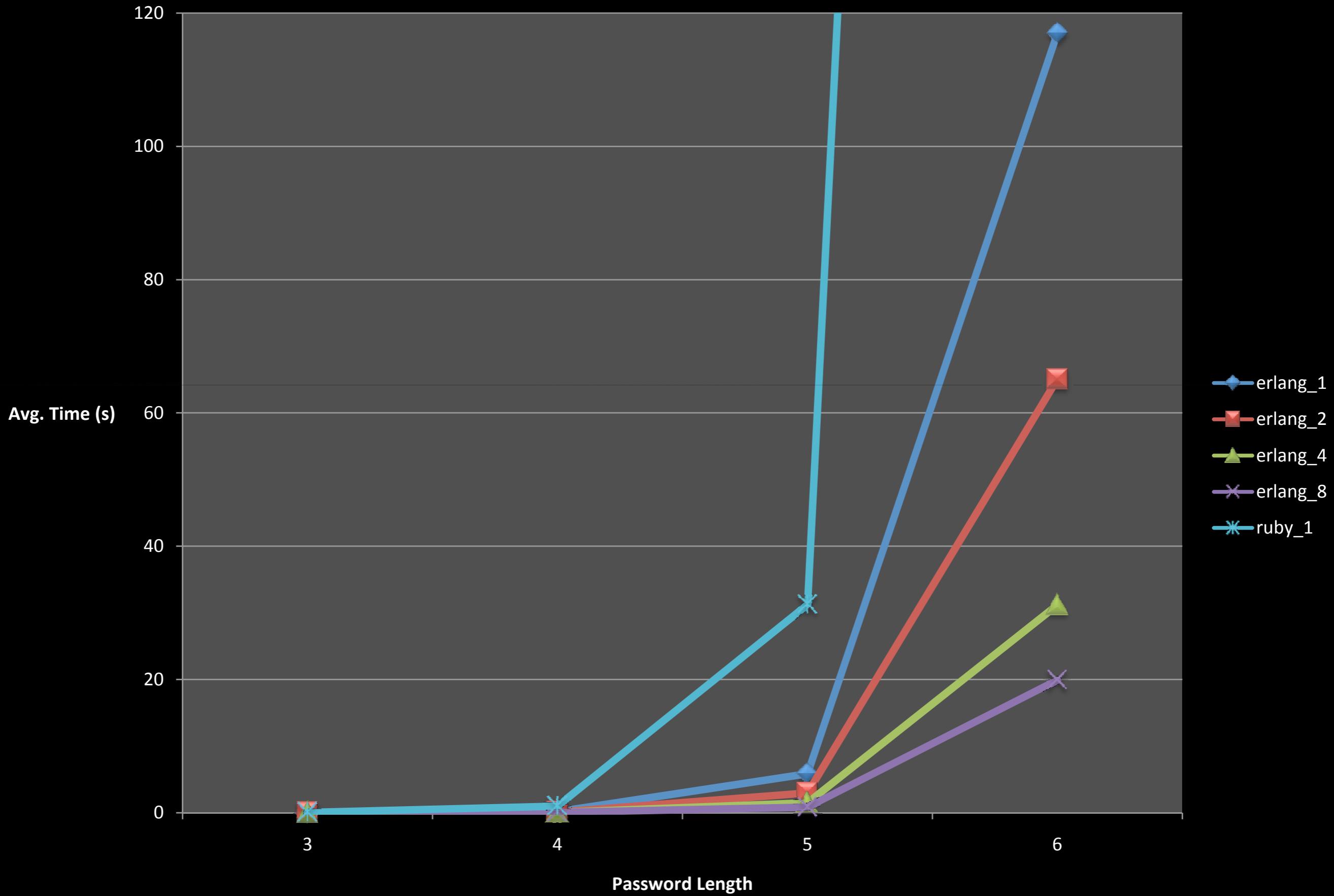
Client analyze function

```
analyze(Server, Crypted, Max, Max, Len) ->
  case erlang:md5(Max) of
    Crypted ->
      Server ! {found, Max};
    _ ->
      Server ! notfound
  end;
analyze(Server, Crypted, Cur, Max, Len) ->
  case erlang:md5(Cur) of
    Crypted ->
      Server ! {found, Cur};
    _ ->
      analyze(Server, Crypted, next(Cur), Max, Len)
  end.
```

Server function

```
loop(0, _Start, Ret) ->
    Ret;
loop(Processes, Start, Ret) ->
    receive
        {found, Password} ->
            Elapsed = timer:now_diff(now(), Start) / 1000 / 1000,
            loop(Processes-1, Start, {found, Password, Elapsed});
        notfound ->
            io:format("Processes remaining: ~p~n", [Processes]),
            loop(Processes-1, Start, Ret)
    end.
```

Password Decryption Times



Avg Decrypt Time for Length 6 Strings

