An NCC Group Publication

Erlang Security 101

Prepared by:

E D Williams

ed.williams ‘at’ nccgroup ‘dot’ com

Contents

1 Background & Introduction

NCC Group's Security Technical Assurance team performs code reviews for clients on numerous different programming languages. Some are well understood from a security perspective (e.g. C, C++, C#, PHP and Python etc.) and some less so. We've been doing Erlang security focused code reviews for over four years and built up a body of knowledge on the subject. It is also understood that proactive developer training and awareness earlier in the development lifecycle can yield significant security benefits.

1.1 Previous Research on Erlang Security

A number of papers have been produced on the security of Erlang[1, 2], the most famous of these being "Application Security of Erlang Concurrent System"[3], the current research is purposefully narrow, this paper will document what to be mindful of when writing Erlang code for a LYME (Linux, YAWS, Mnesia and Erlang) stack or large scale distributed system and more importantly, how to successfully mitigate common security issues.

1.2 What is Erlang?

Erlang was designed and developed by Ericsson to support distributed, fault-tolerant, soft-real-time, non-stop applications in the 1980s[4].

As defined from Wikipedia, "Erlang is a programming language used to build scalable soft real-time systems with requirements on high availability. Some of its uses are in telecoms, banking, e-commerce, computer telephony and instant messaging. Erlang's runtime system has built-in support for concurrency, distribution and fault tolerance[1]."

1.2.1 OTP

An important part of Erlang is OTP, "OTP (Open Telecom Platform) is a collection of libraries for Erlang to do everything from compiling ASN.1 to providing a WWW server. Most projects using "Erlang" are actually using "Erlang/OTP", i.e. the language and the libraries. OTP is also open source[5]."

1.3 Brief History of Erlang

During the 1980s there was a project at the Ericsson Computer Science Laboratory which aimed to find out what aspects of computer languages made it easier to program telecommunications systems. Erlang emerged in the second half of the 80s was the result of taking those features which made writing such systems simpler and avoiding those which made them more complex or error prone[6].

---

[1] http://carlos-trigoso.com/2014/03/01/erlang-application-security/
[2] http://pokingarounderlang.wordpress.com/2009/01/18/erlang-web-security/
[3] http://www.k2r.org/kenji/papers/file-archive/css2008-erlangappsec-final-pub.pdf
[4] http://en.wikipedia.org/wiki/Erlang_%28programming_language%29
[5] http://www.erlang.org/faq/introduction.html
[6] http://www.erlang.org/faq/academic.html

1.4 Who Uses Erlang

While the casual reader may be surprised by the following list of organisations that use Erlang for product development, the more "hard core" Erlang user will not however be surprised[5, 7].

Organisation	Project
Ericsson	Telecommunications Systems
Facebook	Facebook chat service backend
RabbitMQ	AMQP Enterprise Messaging
T-Mobile	SMS and authentication systems
Whatsapp	Messaging for smartphones
Amazon	SimpleDB, Part of Amazon Web Services
Yahoo!	Delicious, part of the social bookmarking service
Huffington Post	Commenting system on HuffPost Live
Rackspace	Manage networking devices

1.5 Basic Syntax and Structures and ubiquitous Hello World

While the Erlang language offers a rich syntax and a wide range of structures, these are beyond the scope of this whitepaper; this section is only intended as a brief introduction into the language. The Erlang website (http://www.erlang.org) is an excellent resource for the seasoned programmer and novice, the author would encourage anybody with an interest in the language to visit this site.

Number Types

Erlang supports basic data types such as integers, floats and more complex structures.

```
1> (9+5)*3.
42
```

Atoms

Atoms are static (or constant) literals.

```
2> edw == edw.
true
```

Tuples

Tuples are a composite data type and are used to store collections of items.

```
3> {abc, def, {0, 1}, ghi}.
{abc,def,{0,1},ghi}
```

[5] http://www.erlang.org/faq/introduction.html
[7] en.wikipedia.org/wiki/erlang_(programming_language)

Lists

Lists and tuples are similar, but whereas a tuple can only be used in a comparison, lists allow a wider variety of manipulation operations to be performed.

```
4> lists:sort([4,5,3,2,6,1]).
[1,2,3,4,5,6]
5> [1,2] ++ [3,4].
[1,2,3,4]
```

Functions and Modules

Functions in Erlang are the basic building blocks, A function is composed of the function name (defined by an atom), and zero or more arguments to the function in parentheses. Modules, just like modules in other languages, are used to collate similar functions together.

```
user@host:~$ cat math.erl
-module(math).
-export([add/2,remove/2]).

add(A,B) ->
    A + B.

remove(A,B) ->
    A - B.
user@host:~$ erl
Erlang R16B03 (erts-5.10.4) [source] [async-threads:10] [kernel-poll:false]

Eshell V5.10.4  (abort with ^G)
1> c(math).
{ok,math}
2> math:add(6,4).
10
3> math:remove(6,4).
2
```

1.5.1 Ubiquitous Hello World*

No introduction of a language would be complete without an example “hello world” script:

```
user@host:~$ cat hello.erl
% Hello World Erlang Script            ← Erlang Comment
-module(hello).                        ← Erlang module name
-export([hello_world/0]).              ← Function name for module with 0 params

hello_world() ->                       ← Start of function
        io:fwrite("Shwd mae byd\n").
user@host:~$ erl -compile hello        ← Compile
user@host:~$ erl -noshell -s hello hello_world -s init stop      ← Run
Shwd mae byd                                                     ← Output
```

* Not quite hello world, but the Welsh equivalent

2 Erlang Language Security Implications

The following section details the common security issues that have been discovered within Erlang code bases and implementations. While the examples are purposefully basic, they are intended to highlight common security-related classes of issue. Each section will also include a specific mitigation for said class.

2.1 Clear Text Node Communication

Nodes are a cornerstone of distributed Erlang, they allow the message passing, linking and monitoring of each subsystem. With this in mind, the security of the communication between distributed nodes, which may well be located in disparate physical locations, is critical. A common method of communication between nodes is through the `rpc:call` function. By default, this traverses the network un-encrypted, allowing a suitably placed attacker the ability to capture data.

Consider the following trivial RPC example, where the erl_test directory is listed via node1, from node2:

```
user@host:~$ erl -sname node2@`hostname`
Erlang R16B03 (erts-5.10.4) [source] [async-threads:10] [kernel-poll:false]

Eshell V5.10.4  (abort with ^G)
(node2@host)1> rpc:call(node1@host1, file, list_dir, ["/erl_test"]).
{ok,["erl_test-file_2","erl_test-file_1","erl_test-file_3"]}
```

These packets can be intercepted, and as can be seen below, while the cookie does not traverse the network, the data does:

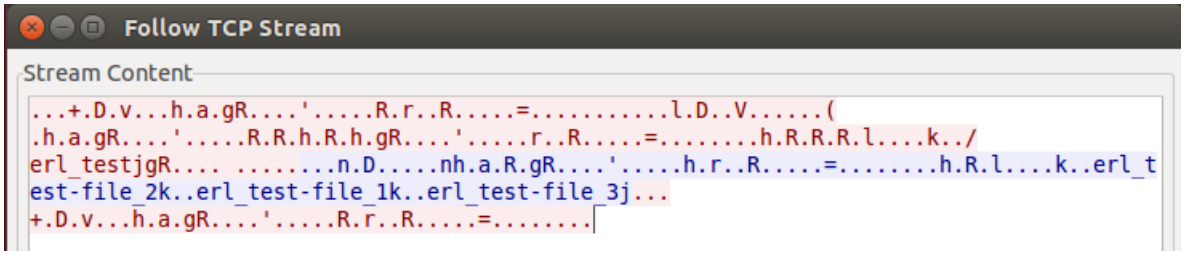

Figure 1: Intercepted RPC Communication

To mitigate this issue, a number of scalable methods can be used to secure communications between disparate nodes:

- An underlying IP mechanism like VPN or IPSEC should be used to connect distributed nodes;
- Use TLS to connect the Erlang nodes[8];
  - use an SSH tunnel to pass all Erlang communications through it;
  - `erl -rsh ssh`

[8] http://www.erlang-factory.com/upload/presentations/214/ErlangFactorySFBay2010-KenjiRikitake.pdf

2.2 Erlang Cookies

For nodes to communicate with one another, they must share the same magic cookie - a password in Erlang land. The use of a magic cookie provides a basic security mechanism to prevent unauthorised access to an Erlang system on another computer.

It is possible to create a node with a magic cookie value of “nocookie”, this should **not** be done; this removes authentication security and can allow anyone on the network access to Erlang nodes and the underlying operating system.

If a magic cookie is set, which it always should be, setting this value via the command line can leave the system vulnerable to cookie theft from local users. The following example is given, where a node is created using a magic cookie on the command line, and a process listing showing the magic cookie is given when queried from a different user account:

```
user@host:~# erl -sname node1 -cookie secret_cookie
Erlang R16B03 (erts-5.10.4) [source] [async-threads:10] [kernel-poll:false]

Eshell V5.10.4  (abort with ^G)
(node1@host)1>
(node1@host)1> os:cmd("ps -ef | grep -v grep | grep -i secret_cookie").
"root      9147  9122  0 10:11 pts/16   00:00:00 /usr/lib/erlang/erts-
5.10.4/bin/beam -- -root /usr/lib/erlang -progname erl -- -home /root -- -sname
node1 -cookie secret_cookie\n"
(node1@host)2>
```

A secure alternative is to use a `$HOME/.erlang.cookie` file, which cannot be enumerated via a process listing, as can be seen below:

```
user@host:~$ ls -al .erlang.cookie
-r-------- 1 ed ed 14 Nov  6 10:30 .erlang.cookie
user@host:~$ cat .erlang.cookie
secret_cookie
user@host:~$ erl -sname node1
Erlang R16B03 (erts-5.10.4) [source] [async-threads:10] [kernel-poll:false]

Eshell V5.10.4  (abort with ^G)
(node1@host)1> os:cmd("ps -ef | grep -v grep | grep -i secret_cookie").
[]
(node2@host)2> erlang:get_cookie().
secret_cookie
(node2@host)3>
```

When using nodes, a magic cookie should always be used; the magic cookie should not be passed via the command line but via the `.erlang.cookie` file.

It can be easy to create overly permissive permissions on a sensitive file, ensure that file level permissions on the `.erlang.cookie` file are strict, such that only the current user can read the file. As show below:

```
user@host:~$ umask
0002
user@host:~$ echo secret_cookie > .erlang.cookie
user@host:~$ ls -al .erlang.cookie
-rw-rw-r-- 1 user user 14 Nov  6 11:12 .erlang.cookie
user@host:~$ chmod 0400 .erlang.cookie
user@host:~$ ls -al .erlang.cookie
-r-------- 1 user user 14 Nov  6 11:12 .erlang.cookie
```


2.3 Erlang Node Context

The user context which Erlang nodes run under is important. Consider the following example; where a node, node1, is run under the context of the root (UID 0) user and no magic cookie is required, this leaves the host and potentially the infrastructure vulnerable:

```
root@host:~# erl -sname node1 -cookie nocookie
Erlang R16B03 (erts-5.10.4) [source] [async-threads:10] [kernel-poll:false]

Eshell V5.10.4  (abort with ^G)
(node1@host)1> os:cmd("id").
"uid=0(root) gid=0(root) groups=0(root)\n"
(node1@host)2> os:cmd("wc -l /etc/shadow").
"39 /etc/shadow\n"
```

Conforming to the principle of least privilege, Erlang nodes should run with the least amount of privileges required, they should **never** be run under the context of root (UID 0).

2.4 Erlang and Crypto

The importance of random numbers and cryptography is especially important given with many applications looking to enhance security for their users[9]; therefore, the generation of random numbers is important for the security of modern day applications.

A common error is to use the random module:

```
1> random:seed(now()).
{8236,26623,17360}
2> {A, B, C} = now().
{1329,311552,420522}
3> random:seed(A,B,C).
{1330,8476,19723}
```

The documentation states:

"*It should be noted that this random number generator is not cryptographically strong. If a strong cryptographic random number generator is needed; for example, crypto:rand_bytes/1 should be used instead*[10]*.*"

An example of which is given below:

```
1> crypto:rand_bytes(24).
<<23,244,102,197,153,106,140,132,231,60,4,163,58,31,193,70,139,61,46,153,51,186,1
45,126>>
2>
```

[9] http://fortune.com/2014/11/18/facebooks-messaging-service-whatsapp-gets-a-security-boost/
[10] http://erlang.org/doc/man/random.html

2.5 Password Hashing

Storing passwords securely is an important aspect of modern applications, all too often there are reports in the press of passwords stored in clear text[11], not using a salt[12] and / or using a weak hashing algorithm like MD5[13].

An Erlang example of an unsalted MD5 hash is given below[14]:

```
Eshell V5.10.4  (abort with ^G)
1> c(md5).
{ok,md5}
2> md5:md5_hex("password").
"5f4dcc3b5aa765d61d8327deb882cf99"
```

As an example of how trivial it is to crack raw MD5 hashes, the password hash was passed to a popular password cracking tool, namely JohnTheRipper[15]:

```
user@host:~/JohnTheRipper/run$ ./john --format=Raw-MD5 h
Loaded 1 password hash (Raw-MD5 [MD5 32/32])
Press 'q' or Ctrl-C to abort, almost any other key for status
password         (?)
1g  0:00:00:00  DONE  (2014-11-06  07:50)  6.250g/s  18.75p/s  18.75c/s  18.75C/s
password
```

As can be seen, this trivial example cracked immediately. For modern day secure applications, the use of MD5, SHA1 and SHA2 alone is not considered appropriate, even with a salt; a more secure alternative is now required. These come in the form of password stretching algorithms, examples of which include bcrypt[16] and PBKDF2[17].

Fortunately, there exist a number of git repositories that can be used to increase the defence in depth of an application; the following example uses bcrypt[18]

```
user@host:~/erlang-bcrypt$ erl -pa ./ebin -s crypto -s bcrypt -smp enabled
Erlang  R16B03  (erts-5.10.4)  [source]  [smp:1:1]  [async-threads:10]  [kernel-
poll:false]

Eshell V5.10.4  (abort with ^G)
1> {ok, Salt} = bcrypt:gen_salt().
{ok,"$2a$12$LxvA.soqBr6jzcl87UNIye"}
2> {ok, Hash} = bcrypt:hashpw("password", Salt).
{ok,"$2a$12$LxvA.soqBr6jzcl87UNIye/vOY5cdDRyJJXb8jeBcdlHbjdYhty9a"}
```

The further reading section 6.1.1 of this whitepaper has a full list of repositories that can be used to secure password hashes for Erlang.

---

[11] http://www.troyhunt.com/2012/07/lessons-in-website-security-anti.html
[12] http://nakedsecurity.sophos.com/2012/06/06/millions-of-linkedin-passwords-reportedly-leaked-take-action-now/
[13] http://nakedsecurity.sophos.com/2010/12/28/mozilla-accidentally-publishes-user-ids-and-passwords/
[14] http://sacharya.com/md5-in-erlang/
[15] http://www.openwall.com/john/
[16] http://en.wikipedia.org/wiki/Bcrypt
[17] http://en.wikipedia.org/wiki/PBKDF2
[18] https://github.com/smarkets/erlang-bcrypt

2.6 External System Calls

It is common practice to make use of external system calls from within Erlang code, this does, however, increase the risk of introducing traditional vulnerability classes. The most common classes identified with external system calls are detailed below:

2.6.1 Full Path on External Calls

Ensuring that all external calls use a full path is important in reducing local users from influencing code logic. Consider the following example, where the `os:cmd` module is performing some functions*:

```
1> os:cmd("chmod 777 important_task.sh"),
1> os:cmd("chown user:user important_task.sh").
```

Under the correct circumstances i.e. if the `$PATH` environmental variable allows, a malicious user would be able to replace OS commands with their own, malicious functions.

2.6.2 Checking Return Status on System Calls and File Operations

Ensuring that system calls "fail safe" increases the assurance of system-related code. Not checking the return code can alter the flow and ultimately the execution of code. An example, given next, which gives flexibility in terms of node communication, is through the use of ports in Erlang:

```
Eshell V5.10.4  (abort with ^G)
1> R = open_port({spawn, "/bin/ls no_file"}, [stderr_to_stdout, binary]).
#Port<0.357>
2> flush().
Shell got {#Port<0.357>,
           {data,<<"/bin/ls: cannot access no_file: No such file or
directory\n">>}}
ok
3>
```

2.7 Export_all

The `export_all` compilation flag will ignore the export module attribute and will instead export all functions defined. In a production environment this is not considered secure for the following reasons[19]:

- Clarity - it's easier to see which functions are intended to be used outside the module.
- Errors - you get warnings for unused functions.
- Optimisation - the compiler might be able to make more aggressive optimisations knowing that not all functions have to be exported.

---

* It goes without saying, but worth repeating, that files should not be given these permissions.

[19] http://stackoverflow.com/questions/7863087/why-is-compileexport-all-bad-practice

3 LYME (software bundle)

LYME is a solution stack composed entirely of free and open-source software to build high-availability heavy duty dynamic web applications. The stack is made up of the following software components:

- Linux – operating system.
- Yaws – web server.
- Mnesia - database.
- Erlang – programming language.

Linux is considered beyond the scope of this whitepaper, and Erlang security is discussed elsewhere within this document.

3.1 YAWS (Yet Another Web Server)

Yaws is a HTTP high performance webserver particularly well suited for dynamic-content web applications[20].

Because Yaws uses Erlang's lightweight threading system, it performs well under high concurrency. A load test conducted in 2002 comparing Yaws and Apache found that with the hardware tested, Apache 2.0.39 with the worker MPM failed at 4,000 concurrent connections, while Yaws continued functioning with over 80,000 concurrent connections[21].

Issues have been discovered with YAWS[22] and its important that that YAWS is running the latest version. At the time of writing, the latest version of YAWS is 1.98. With that in mind, NCC has produced an NSE script that can be used in conjunction with NMAP to quickly identify vulnerable YAWS web servers; an example of which is given below:

```
c:\>nmap --script-updatedb

Starting Nmap 6.45 ( http://nmap.org ) at 2014-11-03 18:18 GMT Standard Time
NSE: Updating rule database.
NSE: Script Database updated successfully.
Nmap done: 0 IP addresses (0 hosts up) scanned in 1.54 seconds

c:\>nmap -sV --script=yaws.nse -p8443 192.168.198.129

Starting Nmap 6.45 ( http://nmap.org ) at 2014-11-03 18:18 GMT Standard Time
Nmap scan report for hotspot.s-bit.nl (192.168.198.129)
Host is up (0.00s latency).
PORT     STATE SERVICE  VERSION
8443/tcp open  ssl/http Yaws httpd 1.89
|_yaws: Your YAWS installation is outdated: 1.89
MAC Address: 00:0C:29:EE:9E:E5 (VMware)

Service detection performed. Please report any incorrect results at http://nmap.
org/submit/ .
Nmap done: 1 IP address (1 host up) scanned in 68.62 seconds
```

3.1.1 Configuring SSL Securely

NCC has produced a whitepaper[23] that details how organisations can avoid SSL issues commonly found during penetration tests; the basic configuration of YAWS is found within the `yaws.conf` file.

---

[20] http://yaws.hyber.org/
[21] http://en.wikipedia.org/wiki/Yaws_%28web_server%29
[22] http://www.cvedetails.com/vulnerability-list/vendor_id-3124/Yaws.html
[23] https://www.nccgroup.com/media/481577/ss-hell-v0-15-2-whitepaper.pdf

3.2 Mnesia

“Mnesia is a distributed, soft real-time database management system written in Erlang[24]”.

As with Erlang, Mnesia was developed by Ericsson for soft real-time distributed and high-availability computing work related to telecoms. It was not intended as a general office-based data processing DBMS, nor to replace SQL-based systems. Instead Mnesia exists to support Erlang, where DBMS-like persistence is required[24].

Data in Mnesia is organized as a set of tables. Each table has a name which must be an atom. Each table is made up of Erlang records. The user is responsible for the record definitions. Each table also has a set of properties[25].

[24] http://en.wikipedia.org/wiki/Mnesia
[25] http://www.erlang.org/doc/man/mnesia.html

4 Mitigation of Detailed Issues

While a number of issues have been identified with specific mitigations within this document, there exist a number of tools that can be used in conjunction with one another to help identify and mitigate security-related issues.

4.1 Dialyzer & Typer

The Erlang language provides two mechanisms that can be used to help determine errors and poor coding practices, namely dialyzer and typer.

4.1.1 Dialyzer

Dialyzer, a DIscrepancy AnalYZer for Erlang programs, is a static analysis tool that identifies software discrepancies such as definite type errors, code which has become dead or unreachable due to some programming error, unnecessary tests, etc. in single Erlang modules or entire sets of applications[26].

4.1.2 Typer

TypEr is a tool that displays and automatically inserts type annotations in Erlang code. It uses Dialyzer to infer variable types[27].

4.1.3 Installation and Usage

For Debian-based systems, dialyzer and typer can be installed via apt-get, as in the following:

```
sudo apt-get install Erlang-dialyzer Erlang-typer
```

Before dialyzer can be used, a PLT (Persistent Lookup Table) file needs to be created, this file stores information about analysed functions. It should be built with commonly used functions and modules, the following example will build the PLT file with only the stdlib and kernel modules (the bare minimum for any sort of useful analysis):

```
sudo dialyzer --build_plt --apps stdlib kernel
```

Once this is done, which can take a while, depending on the modules that have been applied, dialyzer can now be used. As an example, we will use dialyzer with our trivial hello.erl program:

```
user@host:/Erlang$ dialyzer hello.erl
  Checking whether the PLT /home/ed/.dialyzer_plt is up-to-date... yes
  Proceeding with analysis... done in 0m0.41s
done (passed successfully)
```

As can be seen and what was expected, the code was found without any errors.

Similarly, typer can be used against Erlang code, an example of usage is given below:

```
user@host:~$ typer --show hello.erl

%% File: "hello.erl"
%% -----------------
-spec hello_world() -> 'ok'.
```

[26] Man page entry for Dialyzer
[27] Man page entry for typer

4.2 CodeNavi

CodeNavi[28] is a tool designed by NCC Group to manage the security audit of large code projects. Within CodeNavi, these exists custom profiles that can perform dedicated and specific searches against a pre-defined list of commonly found issues and expressions.

For users running CodeNavi 1.2 and less, a basic Erlang.txt file has been created; for the more adventurous user i.e. those running CodeNavi version 1.3 and above, an `Erlang.grepifyv2` file has been created to aid a code review of Erlang code.

A sample screenshot of the Erlang.txt profile, within Grepify.Profiles can be seen below:

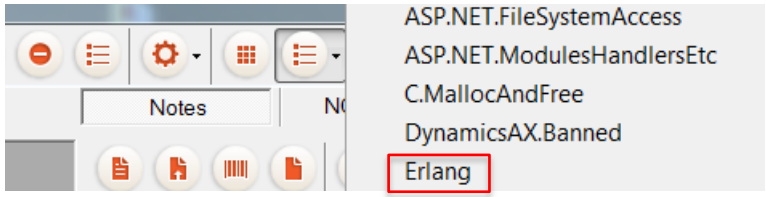

Figure 2: Erlang Profile

View the CodeNavi Getting-Stated page[29] for more information on usage and CodeNavi functionality.

[28] https://github.com/nccgroup/ncccodenavi
[29] https://github.com/nccgroup/ncccodenavi/wiki/Getting-Started

5 Conclusion

Writing secure Erlang code or any code for that matter, is not a trivial task and is often difficult to "fit–in" retrospectively; with that in mind, the following high-level issues should be introduced from an early phase within the design process:

- Ensure that node communications are appropriately encrypted.
- When using magic cookies:
  - Do not use "noccokie"
  - Use `$HOME/.erlang.cookie`
  - Run nodes with the minimum amount of privileges i.e. not root (UID 0)
- When using crypto, use `crypto:rand_bytes.`
- When hashing passwords, ensure a robust hashing algorithm is used like bcrypt or PBKDF2.
- For external system calls:
  - Ensure that the full path is used;
  - System calls and file operations should have their return status checked to ensure logic is maintained.
- Do not `export_all` in a production environment.
- Ensure that all elements of the LYME stack are running the latest, stable release.

When analysing Erlang code bases a number of tools are available:

- Dialyzer
- Typer
- CodeNavi with an Erlang profile

6 Further Reading and References

6.1 Further Reading

Learn You Some Erlang for great good! – http://learnyousomeerlang.com/

Erlang Solutions – https://www.erlang-solutions.com/

Erlang/OTP Professional Services – http://www.erlang-services.com/eng/index.htm

Erlang Course – http://www.erlang.org/course/course.html

Application Security of Erlang Concurrent System – http://www.k2r.org/kenji/papers/file-archive/css2008-erlangappsec-final-pub.pdf

6.1.1 Secure Hashing Resources

Erlang-bcrypt – https://github.com/smarkets/erlang-bcrypt

Erlang-pbkdf2 – https://github.com/whitelynx/erlang-pbkdf2

Erlpass – https://github.com/ferd/erlpass

7 Acknowledgements

The author wishes to thank his colleagues Matt Lewis and Ollie Whitehouse of NCC Group for their peer review and valued suggestions.

