Eliminating application passwords using TLS

D.S. Ljungmark
Modio AB
https://twitter.com/spidler
https://github.com/Spindel/
https://github.com/ModioAB/caramel

MODIO
Secure & efficient energy information

Introductions

"Release Dictator"

D.S. Ljungmark
Modio AB
https://twitter.com/spidler
https://github.com/Spindel/

Systems nerd
Security fellow
Free Software Fan
Opinionated Unixbeard
Social Justice Aficionado

Slides

https://www.modio.se/pages/presentations.html

MODIO
Secure & efficient energy information

Application passwords

Machine to Machine

Service to Service

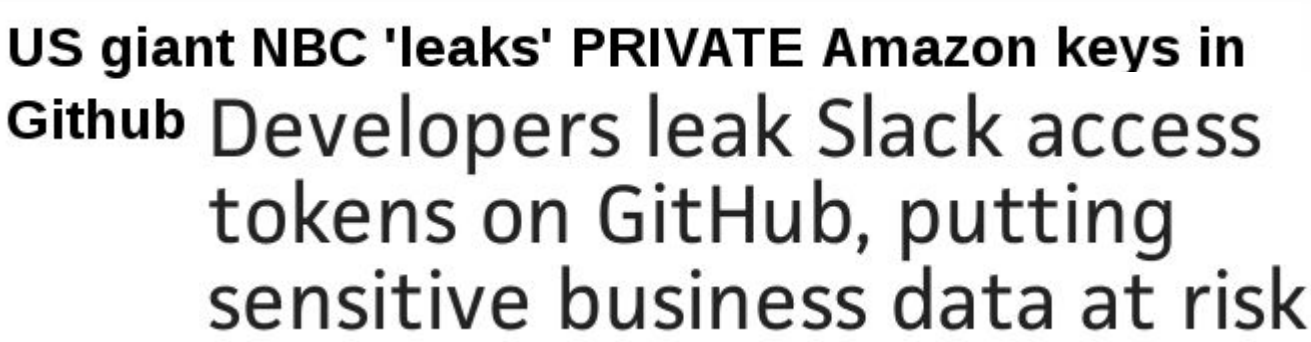
Example Application Passwords

```
mysql://dbadmin:f1$hSt1ck$@db.example.org
```

Amazon ID Keys

Application ID keys

Slack ID token



MODIO
Secure & efficient energy information

Password Drawbacks

- Cleartext on Disk
- Cleartext on Network
- Reused for many things

```
53
54  # Make this unique, and don't share it with anybody.
55  SECRET_KEY = 'b7ocb4t18uebkm%fz*1z(f^jd@rm=cwj^-zt!@guuj&3g0mhwt'
56
```

Like this

```
# Make this unique, and don't share it with anybody.
SECRET_KEY = "verysecret"
```

academy/metacademy-application – settings_local-templat
wing the top five matches. Last indexed on 18 Mar.

```
import config

DEBUG = config.DEBUG
TEMPLATE_DEBUG = DEBUG

# Make this unique, and don't share it with anybody.
SECRET_KEY = '1234!'
```

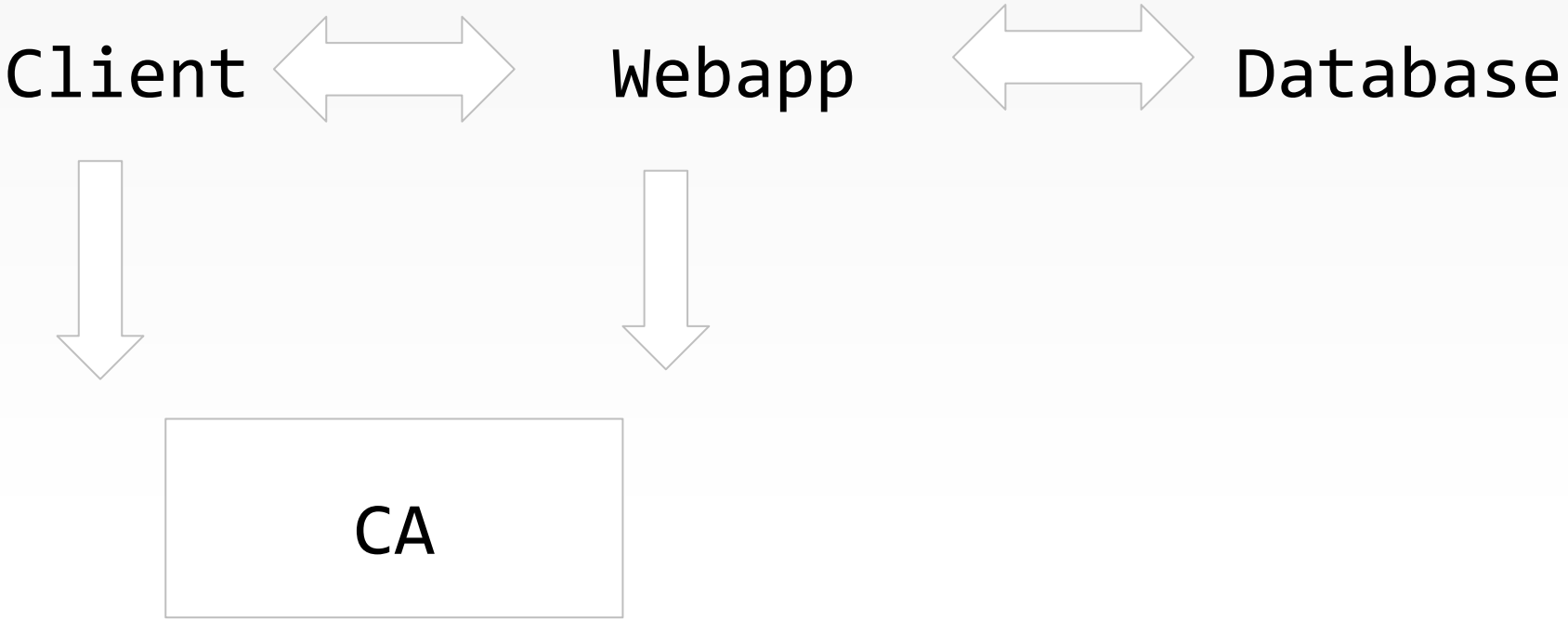
```
 p = '/Users/ross/Pychrondata_experiment/data/isotopedb.sqlite'
= '/usr/local/pychron/isotope.sqlite
 p = '/Users/ross/Sandbox/pychron_test_data/data/isotopedb.sqlite'
 main(url='sqlite:///{}'.format(p) , debug='False', repository='isotc

 url = 'mysql://root:Argon@localhost/isotopedb?connect_timeout=3'
url = 'mysql+pymysql://root:Argon@localhost/isotopedb_dev?connect_tim
#url = 'mysql+pymysql://root:Argon@localhost/pychronlocal?connect_timeout=3'
#url = 'mysql+pymysql://root:Argon@localhost/isotopedb_dev_migrate?connect_timeout=3'
#    url = 'mysql://root:Argon@localhost/isotopedb_FC2?connect_timeout=3'
# url = 'mysql://massspec:DBArgon@129.138.12.131/isotopedb_dev_mod?connect_timeout=3'
#url = 'mysql+pymysql://massspec:DBArgon@129.138.12.160/pychrondata?connect_timeout=3'
# url = 'mysql+pymysql://massspec:DBArgon@129.138.12.160/pychrondata?connect_timeout=3'
#    url = 'mysql+pymysql://root:Argon@localhost/pychrondata_minnabluff?connect_timeout=3'

#     url = 'mysql://root:Argon@localhost/isotopedb_dev?connect_timeout=3'
main(url=url, debug='False', repository='isotopedb/')
```

TLS is not panacea

Our demo architecture



MODIO
Secure & efficient energy information

Introducing Caramel

- Simple CA
- Designed for Client Authorization

https://github.com/ModioAB/caramel

MODIO
Secure & efficient energy information

Alternatives to Caramel

TAuth

- Certificates + Tokens

Dogtag

- Complete CA solution

What you need

- TLS enabled
- CA distribution
- Certificate updates

Drawbacks

- Key management
- Certificate updates
- Distributing custom CA roots
- Configuration...

Advantages

+ Guaranteed encryption
+ Platform independent
+ Implementation supported
+ Hardware accelerated

MODIO
Secure & efficient energy information

Authentication

is not

Authorization

Still need to verify WHICH user

How it works

- Certs are short lived
- Refreshed on CA server
- Built for _client_ certs
- ~~Not using ACME~~

Installing Caramel

```
$ pip install caramel
$ mkdir caramel; cd caramel;
$ cp template.ini ca.ini
$ caramel_initialize_db ca.ini
$ caramel_ca ca.ini
# Set up with reverse proxy, fastcgi or wsgi
```

Our database: PostgreSQL

- Well known
- Good Performant
- Well documented

Enter Let’s Encrypt

```
$ letsencrypt certonly -d do02.skuggor.se
```

PostgreSQL: Enabling TLS

```
listen_addresses = '*'

ssl = true

ssl_cert_file = '/etc/letsencrypt/archive/do02.skuggor.
se/fullchain1.pem'

ssl_key_file = '/etc/letsencrypt/archive/do02.skuggor.
se/privkey1.pem'

+ssl_ciphers = 'ECDHE-ECDSA-AES256-GCM-SHA384:ECDHE-RSA-AES256-GCM-SHA384:ECDHE-ECDSA-
CHACHA20-POLY1305:ECDHE-RSA-CHACHA20-POLY1305:ECDHE-ECDSA-AES128-GCM-SHA256:ECDHE-RSA-
AES128-GCM-SHA256:ECDHE-ECDSA-AES256-SHA384:ECDHE-RSA-AES256-SHA384:ECDHE-ECDSA-AES128-
SHA256:ECDHE-RSA-AES128-SHA256'
```

PostgreSQL: Allowing access

```
# pg_hba.conf
hostssl demoservice demoservice 2a03:b0c0:2:d0::/64    md5
hostnossl      all     all     ::/0 reject
hostnossl      all     all     0.0.0.0/32 reject
```

Connecting webserver=>DB

```
$ psql 'host=do02.skuggor.se port=5432 \
        dbname=demoservice user=demoservice'
# Success
```

Pyramid webapp

```
# on webapp
$ pcreate -t alchemy demoservice
$ cp production.ini demoservice.ini
# demoservice.ini, sqlalchemy.url
postgresql://demoservice:bahumbug@do02.skuggor.se:
5432/demoservice
$ initialize_demoservice_db demoservice.ini
```

Apache TLS config

```
# on webapp
$ letsencrypt --apache -d do01.skuggor.se
```

```
SSLEngine on
SSLProtocol             all -SSLv3 -TLSv1 -TLSv1.1
SSLCipherSuite          ECDHE-ECDSA-AES256-GCM-SHA384:ECDHE-
RSA-AES256-GCM-SHA384:ECDHE-ECDSA-CHACHA20-POLY1305:ECDHE-RSA-
CHACHA20-POLY1305:ECDHE-ECDSA-AES128-GCM-SHA256:ECDHE-RSA-
AES128-GCM-SHA256:ECDHE-ECDSA-AES256-SHA384:ECDHE-RSA-AES256-
SHA384:ECDHE-ECDSA-AES128-SHA256:ECDHE-RSA-AES128-SHA256
SSLHonorCipherOrder     on
SSLCompression          off
SSLSessionTickets       off
```

Apache mod_wsgi

```
WSGIApplicationGroup %{GLOBAL}
WSGIPassAuthorization On
WSGIDaemonProcess pyramid user=ubuntu group=ubuntu threads=4 \
  python-path=/home/ubuntu/env/lib/python3.5/site-packages

WSGIScriptAlias / /home/ubuntu/env/demoservice/pyramid.wsgi

<Directory /home/ubuntu/env/>
     WSGIProcessGroup pyramid
          Order allow,deny
          Allow from all
          Require all granted
</Directory>
```

TLS Hello World: Complete

Next step:

Adding TLS Authentication

Making libpq verify certs

```
# on webservice
$ psql 'host=do02.skuggor.se port=5432 \
  dbname=demoservice user=demoservice \
  sslmode=verify-full \
  sslrootcert=/etc/ssl/certs/ca-certificates.crt'
```

```
# demoservice.ini, sqlalchemy.url postgresql:
//demo...:demo...@bo01.../demoservice?sslmode=verify-
full&sslrootcert=/etc/ssl/certs/ca-certificates.
crt
```


MODIO
Secure & efficient energy information

Fixing bad defaults

```
# on webservice
$ psql 'host=do02.skuggor.se port=5432 \
  dbname=demoservice user=demoservice \
  sslmode=verify-full sslcompression=0 \
  sslrootcert=/etc/ssl/certs/ca-certificates.crt

# demoservice.ini, sqlalchemy.url
sslmode=verify=full&sslcompression=0
```

Deploy CA Cert to DB

```
# On DB
$ curl https://ca.skuggor.se/root.crt \
    -o /etc/ssl/certs/custom.ca.crt

# postgresql.conf
ssl_ca_file = '/etc/ssl/certs/custom.ca.crt'
```

Generating our webapp backend cert

```
# on our webapp machine
$ mkdir secret; cd secret
$ pip install caramel-client
$ caramel-client ca.skuggor.se demoservice
```

Signing the first client

```
# on our CA machine, List all requests
$ caramel_tool ca.ini --list
1 demoservice
cc00d29c3cb89f4bf7e60f39414429e97d50b78bc3a15f7b78ba04fd315a67c2
----------

# sign first
$ caramel_tool ca.ini --sign 1
```

Enable Client cert auth on DB

```
# pg_hba.conf
hostssl demoservice demoservice 2a03:b0c0:2:d0::/64 md5
hostssl demo…  demo… 2a03:b0c0:2:d0::/64  cert clientcert=1
# unchanged
hostnossl      all     all     ::/0 reject
hostnossl      all     all     0.0.0.0/32 reject
```

Test from webapp server

```
$ psql 'host=do02.skuggor.se port=5432 \
        dbname=demoservice user=demoservice \
        sslmode=verify-full sslcompression=0 \
sslrootcert=/etc/ssl/certs/ca-certificates.crt \
sslcert=~/env/demoservice/secret/demoservice.crt \
sslkey=~/env/demoservice/secret/demoservice.key'
```

Deploy in webapp

```
# demoservice.ini, sqlalchemy.url =
    postgresql://demoservice@do02.skuggor.se:
  5432/demoservice?sslrootcert=/etc/ssl/certs/ca-
        certificates.crt&sslmode=verify-
full&sslcompression=0&sslcert=/home/ubuntu/env/demo
          service/secret/demoservice.
crt&sslkey=/home/ubuntu/env/demoservice/secret/demo
                    service.key
```


MODIO
Secure & efficient energy information

Benchmarks!

Because you always ask

Postgres noTLS Benchmark

```
$ ab -c 20 -t 100  https://do01.skuggor.se/
Complete requests:    13091
Requests per second:  130.89 [#/sec] (mean)
Time per request:     152.801 [ms] (mean)
Time per request:     7.640 [ms] (mean, across all
concurrent requests)
```

Postgres TLS auth benchmark

```
$ ab -c 20 -t 100  https://do01.skuggor.se/

Complete requests:    13340
Requests per second:  133.40 [#/sec] (mean)
Time per request:     149.929 [ms] (mean)
Time per request:     7.496 [ms] (mean, across
all concurrent requests)
```

Benchmarks!

No Difference!

Authentication to Webservice

Our client app

```
session = requests.Session()
session.verify = True # Could be your CA file
session.get("https://do01.skuggor.se/")
```

Deploy CA Cert to webapp

```
# On webapp
$ curl https://ca.skuggor.se/root.crt \
    -o /etc/ssl/certs/custom.ca.crt
```

Client to Webapp (Apache)

```
WSGIPassAuthorization On
SSLVerifyClient require
SSLVerifyDepth 3
SSLOptions        +FakeBasicAuth
SSLCACertificateFile "/etc/ssl/certs/custom.ca.crt"
```

Authorization in our webapp

```
environ['REMOTE_USER'] == ssl cert CN
```

Getting a client cert

```
$ mkdir secret; cd secret
$ pip install caramel-client
$ caramel-client ca.skuggor.se democlient
```

Signing the client cert

```
# on CA, list requests
$ caramel_tool ca.ini --list
1 demoservice cc00d29c3cb89f4bf7e60f39414429e97d50b78bc3a15f7b78ba04fd315a67c2 2016-05-10 00:28:46
2 democlient 898d2b23185d0175d426d2045964f816e2744e6c8234d3e34c1325250a6528f5 ----------
$ caramel_tool production.ini --sign 2
```

Adding TLS client auth

```
session = requests.Session()
session.verify = True # Could be your CA file
session.cert = ("secret/democlient.key",
                "secret/democlient.crt")
session.get("https://do01.skuggor.se/")
```

In total

- Database configuration is painful
+ Webserver configuration is simple
+ Webapps work out of the box
+ The crypto part is solved

That’s all folks!

Or is it?

But what if I don’t use Postgres?

Other databases and webservers beyond this slide

Setting up MariaDB

```
# server.cnf
ssl-ca /etc/ssl/certs/custom.ca.crt
ssl-cert /etc/letsencrypt/live/do02.skuggor.se/fullchain.pem
ssl-key /etc/letsencrypt/live/do02.skuggor.se/privkey.pem
ssl-cipher  : TOO big to fit in slide!
```

Caveats : MariaDB/MySQL

- Cannot disable clear text
- SSL requirement per GRANT

Enabling TLS: MongoDB

Concatenate CERT + KEY into single file

```
--sslMode requireSSL
--sslCAFile /etc/ssl/certs/custom.ca.crt
--sslPEMKeyFile /etc/ssl/private/combined.pem
```

Improving defaults: MongoDB

```
--sslDisabledProtocols TLS1_0,TLS1_1
```

What about nginx?

- Poor i18n support (no åäö)
- Custom config needed

nginx, extract CN

```
# nginx.conf
map $ssl_client_s_dn $ssl_client_s_dn_cn {
    default "should_not_happen";
    ~/CN=(?<CN>[^/]+) $CN;
}
```

nginx: enable client auth

```
# per vhost config:
ssl_client_certificate /etc/ssl/certs/custom.ca.crt;
ssl_verify_client on;
ssl_verify_depth 3;

proxy_set_header  X-CLIENT-SSL-CN $ssl_client_s_dn_cn;
proxy_set_header  X-CLIENT-SSL-DN $ssl_client_s_dn;
```