KeyNanny - Credential protection „the Unix Way“

Protecting sensitive information on server systems

2017-07-22 Martin Bartosch, Cynops GmbH

Problems addressed by KeyNanny

- KeyNanny is designed to protect sensitive information (e. g. passwords, secrets, cryptographic key material) on Unix servers
- KeyNanny solves the problem of securely transmitting sensitive information between administrators of different systems (e. g. passwords for system users, such as databases, LDAP directories or web services)

Problems addressed: „data at rest“

- Protection of sensitive information stored
  - in server file systems (in configuration files used by server applications)
  - in system backups
  - on provisioning systems
  - in developer repositories

Problems addressed: „data in transit“

- KeyNanny provides a method of securely transmitting sensitive information between different parties
  (see this slide for process description)

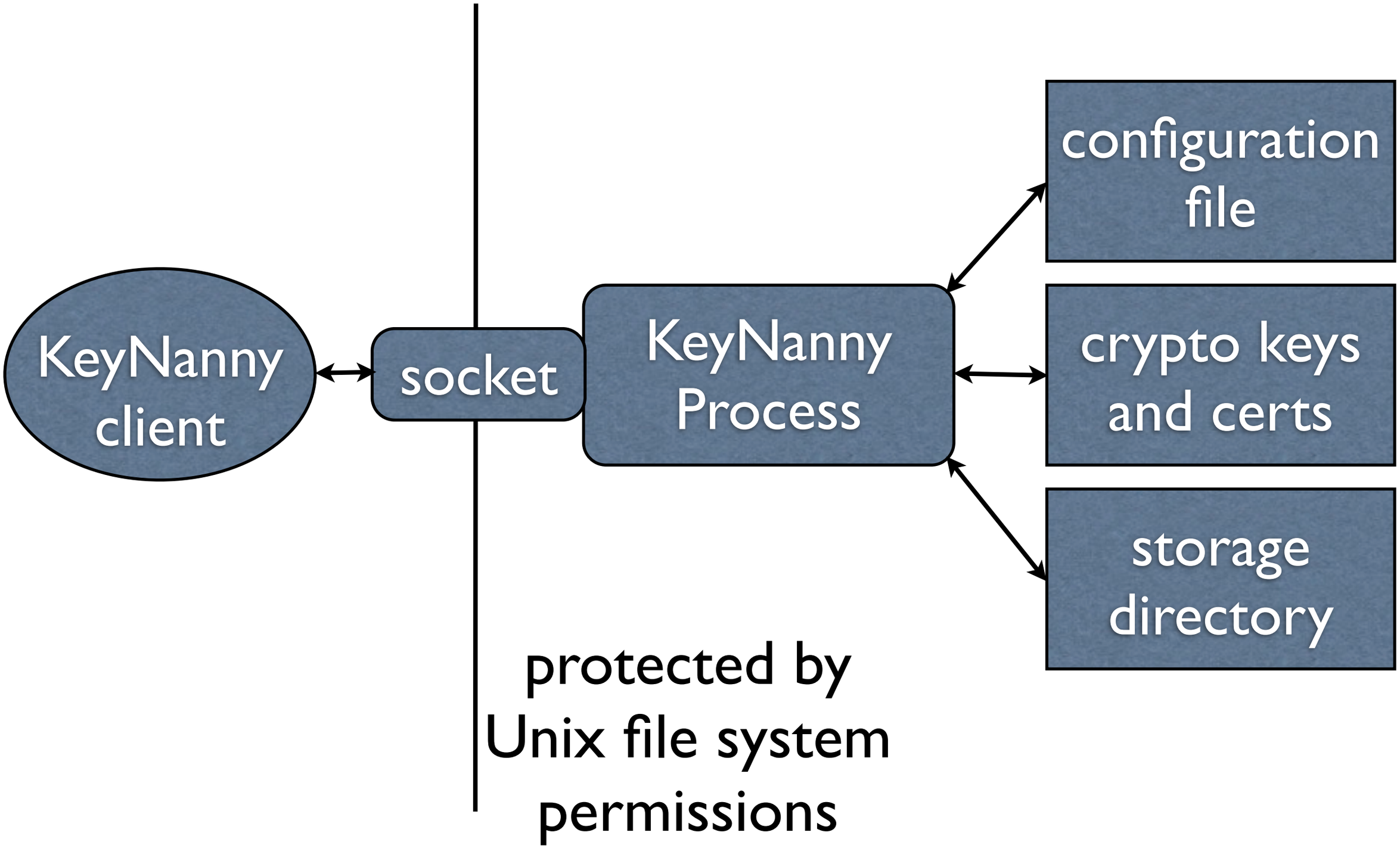
KeyNanny requirements

- Unix-like operating system
  - Unix domain sockets
  - standard Unix security features (file permissions, users, groups)
- Perl
- OpenSSL
- optional but recommended:
  a hardware crypto device (Smartcard, TPM, HSM)

KeyNanny concepts: Architecture

- KeyNanny is implemented as a Unix Daemon
- runs as non-privileged user (typically the same user as consuming application)
- one configuration file per instance
- one application, one KeyNanny daemon
- no limit on number of independent KeyNanny instances on one system

Architecture



KeyNanny concepts: Storage properties

- KeyNanny stores key/value pairs
  - „key“ can be alphanumeric (unique within one KeyNanny instance)
  - „value“ can be text or even binary data
  - no limit on number of key/value pairs

KeyNanny concepts: Storage protection

- implementation: CMS/PKCS#7 encrypted files in dedicated storage directory
  - „key“ is a filename
  - „value“ is the (decrypted) content of the stored file
  - encrypted storage directory contents are not sensitive (may be world readable)

KeyNanny concepts: Process level security

- KeyNanny service is accessible via a dedicated Unix Domain Socket (only on the same host)
- Unix domain socket file is protected with standard Unix file system permissions
- only the application system user should be able to access Keynanny socket
- KeyNanny configuration file can limit client access to read/write, read only or write only
  (write only KeyNanny access may be useful for cases where system administrators should be able to set passwords but not read them)

KeyNanny concepts: Cryptography

- KeyNanny uses common cryptographic standards for protection of stored data
  - Asymmetric encryption (RSA keys, preferably protected with hardware crypto device)
  - X.509v3 certificates
  - CMS/PKCS#7 (container format for encrypted data)
  - multiple encryption certificates possible

KeyNanny clients

- any local Unix process with read/write permissions on KeyNanny socket file can „talk“ to Keynanny
- „talking“ to KeyNanny means speaking the KeyNanny protocol for
  - getting data
  - setting data
  - (... if permitted by KeyNanny configuration)

Native KeyNanny clients

- a „native KeyNanny enabled application“ talks to KeyNanny by daemon socket file
- to obtain stored data it uses the „get“ command, e. g.

  `get mysecret`

  asks KeyNanny daemon to output the value of the stored key „mysecret“ via the socket connection

Enabling applications to use KeyNanny natively

- existing applications can be modified to use Keynanny (source code change required)
- code for connecting to KeyNanny Unix Domain Socket is very similar to reading/writing an ordinary file
- Keynanny protocol is extremely simple:
  - get the value of KEY:

    ```
    get KEY
    ```

  - set KEY to VALUE:

    ```
    set KEY
    VALUE
    <EOF>
    ```

- use this protocol whereever sensitive data is referenced

Supporting existing applications (1)

- modification of application code is not applicable in many cases
  - third party (binary only application)
  - application code not managed in-house
- existing applications typically use configuration files (possibly containing sensitive information stored in the clear)

Supporting existing applications (2)

- replace all ocurrances of sensitive information in config files with named placeholders
- deploy configuration file templates on target system instead of „hot“ configuration files
- resulting templates are no longer sensitive and can be backed up, stored in repositories etc.

Supporting existing applications (3)

- before application startup
  - mount a memory file system (tmpfs)
  - iterate through config file templates
    - get values for all referenced placeholders in template from KeyNanny instance and replace placeholders in template with actual values
    - render resulting configuration file to memory file system
- real application configuration file is a symlink to location of rendered configuration file in memory file system

Exchanging sensitive information (1)

- remote services typically require an application user/password (e. g. database connection, LDAP bind, web service)

- remote admin (e. g. database admin) must set application password and provide it to local admin
- local admin does not really need to know the password - the application does!

Exchanging sensitive information (2)

- solution:
  - send latest KeyNanny certificate to remote admin
  - tell admin to set random password for application user and encrypt it with KeyNanny certificate
  - send resulting CMS/PKCS#7 file to local admin (data is encrypted, non-sensitive!)
  - local admin copies file to KeyNanny instance storage directory with correct filename
  - restart KeyNanny and application

KeyNanny advantages

- allows to remove all sensitive information „in the clear“ from configuration files in repositories and even on host disks
- standard Unix mechanisms
- easy to understand for experienced Unix admins
- decentralized approach
- also solves problem of transmitting credentials
- Open Source, low cost solution

KeyNanny disadvantages

- currently limited to Unix-like operating systems
- system administrator ultimately can read all secrets (inherent in Unix design)
- requires hardware cryptography to get rid of the „final“ secret on host‘s disk (inherent to the problem, not KeyNanny‘s fault)

Examples and use cases

Protecting a server

- application „myapp“ uses a remote database and an LDAP connection
  - myapp database and LDAP credentials are stored in configuration file /etc/myapp/myapp.conf
  - use the configuration file rendering feature to create configuration with passwords before startup
- application „Apache 2.2“ requires an executable printing the private key passphrase on STDOUT on startup (see http://www.modssl.org/docs/2.8/ssl_reference.html#ToC2)
  - use keynanny executable to echo passphrase

Preparing the „myapp“ application configuration

```
$ # note the symlink to the (not yet existing) file in /credentials/myapp/
$ ls -la /etc/myapp
total 11
drwxr-xr-x  2 root root 1024 Jul 22 17:27 .
drwxr-xr-x 74 root root 9216 Jul 22 17:27 ..
lrwxrwxrwx  1 root root   29 Jul 22 17:27 myapp.conf -> /credentials/myapp/myapp.conf
-rw-r--r--  1 root root  375 Jul 22 17:20 myapp.conf.keynanny-template

$ # note how the myapp configuration file template references two keynanny protected variables (which will be
trimmed to remove leading or trailing whitespace) instead of the actual passwords
$ cat /etc/myapp/myapp.conf.keynanny-template
# this is a dummy configuration file for the application myapp
#

[database]
driver: mysql
name: myappdb
user: myapp
password: [% databasepassword | trim %]

[ldap]
uri: ldaps://ldap.example.com:636
base: OU=myapp, O=KeyNanny, C=DE
binddn: cn=myappuser, ou=TechUsers, O=KeyNanny, C=DE
bindpassword: [% ldappassword | trim %]

# ... and many more configuration settings
```

Preparing the „myapp“ KeyNanny instance

```
$ cat /etc/keynanny/myapp.conf
cache_strategy: preload
log: syslog

[crypto]
base_dir: /var/lib/keynanny/crypto

token: cert01
token: cert02

[cert01]
certificate: $(crypto.base_dir)/kn01-cert.pem
key: $(crypto.base_dir)/kn01-key.pem

[cert02]
certificate: $(crypto.base_dir)/kn02-cert.pem
key: $(crypto.base_dir)/kn02-key.pem

[storage]
dir: /var/lib/keynanny/storage/$(namespace)

[server]
user: myapp
group: keynanny
socket_mode: 0700

socket_file: /var/lib/keynanny/run/$(namespace).socket
pid_file: /var/lib/keynanny/run/$(namespace).pid
background: 1
max_servers: 2

[access]
read: 1
write: 0
```

Generating „myapp“ config files in memory file system (1)

```
$ cat /etc/keynanny/myapp.rc
#!/bin/bash
# application specific startup script
# use keynanny to render configuration file templates into temp file system
# arguments to this script, passed by the init script: start, stop

NAME=`basename $0 .rc`

# customize the following settings according to application needs

# keynanny socket file
SOCKETFILE=/var/lib/keynanny/run/$NAME.socket

# base configuration directory to scan for the application, change accordingly
APPLICATION_CONF_DIR=/etc/myapp

# unix user the application is running as (owner of temp file system)
APPLICATION_USER_ID=myapp

# temp file system directory which will be populated with rendered config files
CREDENTIAL_DIR=/credentials/$NAME

# temp file system directory root mode
CREDENTIAL_DIR_MODE=0700

# size of temp file system (in bytes, or use suffixes k for KB, m for MB, g for GB)
CREDENTIAL_DIR_SIZE=16m

# list of all config files belos APPLICATION_CONF_DIR to process
TEMPLATES=`find $APPLICATION_CONF_DIR -type f -name '*.keynanny-template'`

# continued on next page...
```

Generating „myapp“ config files in memory file system (2)

```
# ... continued from previous page
# here we mount a tmp file system on /credentials/myapp

case "$1" in
    start)
      if [ -n "$TEMPLATES" ] ; then
          umount $CREDENTIAL_DIR 2>/dev/null || true
          mkdir -p $CREDENTIAL_DIR
          mount -t tmpfs -o size=$CREDENTIAL_DIR_SIZE,mode=$CREDENTIAL_DIR_MODE,uid=$APPLICATION_USER_ID /dev/null
$CREDENTIAL_DIR
          if [ $? != 0 ] ; then
            echo "ERROR: could not mount temp file system"
            exit 1
          fi
          for file in $TEMPLATES ; do
            TARGETFILE=`basename $file .keynanny-template`
            keynanny --socketfile $SOCKETFILE template $file --outfile $CREDENTIAL_DIR/$TARGETFILE
            chown $APPLICATION_USER_ID $CREDENTIAL_DIR/$TARGETFILE
            chmod 700 $CREDENTIAL_DIR/$TARGETFILE
          done
      fi
      ;;
    stop)
      umount $CREDENTIAL_DIR 2>/dev/null || true
      ;;
esac
```

Let database and LDAP admins set application passwords

From: Myapp Admin
To: Database Admin
Subject: Please set database password for our application user myapp

Dear Database Admin,

Please set a random password for user myapp on database myappdb.
Find attached the KeyNanny certificate of our myapp application server: [kn02-cert.pem]

Please encrypt the database password for this certifiate and send back the encrypted data.
Suggested procedure:

 # Create random password:
PASSWORD=`openssl rand -base64 20`
 # Set database user password for „myapp" to $PASSWORD
 # Encrypt password for our KeyNanny instance:
echo -n $PASSWORD | openssl smime -encrypt -binary -aes256 -outform pem -out databasepassword kn02-cert.pem

Please send us the generated file „databasepassword" via email. This file does not contain sensitive information and can be sent via plain email!

Best regards,

Myapp Admin

***

Send similar email to LDAP Admin, asking setting LDAP bind password and creation of file „ldappassword" instead.

Let database and LDAP admins set application passwords

After receiving the encrypted files „databasepassword" and „ldappassword" from the Database and LDAP admins the myapp administrator simply copies these files to the KeyNanny storage directory and restarts KeyNanny:

```
# cp databasepassword /var/lib/keynanny/storage/myapp/
# cp ldappassword /var/lib/keynanny/storage/myapp/
# /etc/init.d/keynanny stop
# /etc/init.d/keynanny start
# mount | grep credentials
/dev/null on /credentials/apache type tmpfs (rw,size=16m,mode=0700,uid=11109)
/dev/null on /credentials/myapp type tmpfs (rw,size=16m,mode=0700,uid=139)
# ls -la /credentials/myapp/
total 5
drwx------ 2 myapp root   60 Jul 22 22:03 .
drwxr-xr-x 5 root  root 1024 Jul 22 17:23 ..
-rwx------ 1 myapp root  377 Jul 22 22:03 myapp.conf
```

Of course, nothing stops the root user from reading the generated config file:

```
# cat /credentials/myapp/myapp.conf
# this is a dummy configuration file for the application myapp
#

[database]
driver: mysql
name: myappdatabase
user: myapp
password: 0BvPQDW0DkMvH0g64fmpEZ+oalA=

[ldap]
uri: ldaps://ldap.example.com:636
base: OU=myapp, O=KeyNanny, C=DE
binddn: cn=myappuser, ou=TechUsers, O=KeyNanny, C=DE
bindpassword: LEzMD6apZ08tY1FKO+IE86zvzXo=

# ... and many more configuration settings
```

Preparing the „apache“ KeyNanny instance

```
$ cat /etc/keynanny/apache.conf
cache_strategy: preload
log: syslog

[crypto]
openssl: /applications/openssl/1.0.16.0/bin/openssl
base_dir: /var/lib/keynanny/crypto

token: cert01
token: cert02

[cert01]
certificate: $(crypto.base_dir)/kn01-cert.pem
key: $(crypto.base_dir)/kn01-key.pem

[cert02]
certificate: $(crypto.base_dir)/kn02-cert.pem
key: $(crypto.base_dir)/kn02-key.pem

[storage]
dir: /var/lib/keynanny/storage/$(namespace)

[server]
user: wwwown
group: keynanny
socket_mode: 0700

socket_file: /var/lib/keynanny/run/$(namespace).socket
pid_file: /var/lib/keynanny/run/$(namespace).pid
background: 1
max_servers: 2

[access]
read: 1
write: 1
```

Apache is different: passphrase protects local private key file

Apache configuration file references an external program that prints the private key passphrase to STDOUT
See http://www.modssl.org/docs/2.8/ssl_reference.html#ToC2

```
# grep -B3 PassPhrase /etc/apache2/ssl-global.conf
      #   Pass Phrase Dialog:
      #   Configure the pass phrase gathering process.
      #   The filtering dialog program (`builtin' is a internal
      #   terminal dialog) has to provide the pass phrase on stdout.
      SSLPassPhraseDialog  exec:/usr/local/lib/keynanny-apache.sh
```

Apache is different: passphrase protects local private key file

Local admin sets the passphrase (there will be an easy to use command line tool for this purpose later)

```
$ ( echo "set passphrase" ; echo "test1234abc" ) | socat UNIX-CONNECT:/var/lib/keynanny/run/apache.socket -
```

A shell script calls keynanny command line tool and queries „passphrase" of the „apache" KeyNanny:

```
$ ls -la /usr/local/lib/keynanny-apache.sh
-rwxr-xr-x 1 root root 86 Jul 22 17:26 /usr/local/lib/keynanny-apache.sh
$ cat /usr/local/lib/keynanny-apache.sh
#!/bin/bash
keynanny --socketfile /var/lib/keynanny/run/apache.socket get passphrase
```

The script will produce the pass phrase when called:

```
$ whoami
wwwown
$ /usr/local/lib/keynanny-apache.sh
test1234abc
```

This will not work as a different user:

```
$ whoami
vagrant
$ /usr/local/lib/keynanny-apache.sh
Socketfile /var/lib/keynanny/run/apache.socket is not readable at /vagrant/lib//KeyNanny.pm line 18.

# ls -la /var/lib/keynanny/run/
total 4
drwxrwxr-x 2 root   keynanny 1024 Jul 22 22:03 .
drwxr-xr-x 5 root   root     1024 Jul 16 15:26 ..
-rw-r--r-- 1 wwwown keynanny    6 Jul 22 22:03 apache.pid
srwx------ 1 wwwown keynanny    0 Jul 22 22:03 apache.socket
-rw-r--r-- 1 myapp  keynanny    6 Jul 22 22:03 myapp.pid
srwx------ 1 myapp  keynanny    0 Jul 22 22:03 myapp.socket
```