
uts-server - RFC 3161 Timestamp Server

Release 0.1.9

Jul 24, 2017

Contents

1	Dependencies	1
1.1	Runtime dependencies	1
1.2	Build dependencies	1
2	Compilation	3
3	OS specific tips	5
3.1	Debian	5
3.2	CentOS/RHEL	5
3.3	FreeBSD	5
3.4	OpenBSD	5
4	Configuration Parameters	7
4.1	Section [main]	7
4.2	Section [oids]	9
4.3	Section [tsa]	9
5	Full Configuration File	11
6	Deploy	15
6.1	Usage	15
6.2	Running uts-sever	15
7	Changelogs	17
7.1	0.1.9	17
7.2	0.1.8	17
7.3	0.1.7	17
7.4	0.1.6	17
7.5	0.1.5	17
7.6	0.1.4	18
7.7	0.1.3	18
7.8	0.1.2	18
7.9	0.1.1	18
7.10	0.1.0	18
7.11	0.0.3	18
7.12	0.0.2	18
7.13	0.0.1	18

8	Some Goodies	19
8.1	Time-Stamp script combining curl and openssl	19
9	uts-server	23
9.1	License	23
9.2	What is RFC 3161?	23
9.3	Quick (and dirty) Testing	24
9.4	Powered by	24

Runtime dependencies

List of dependencies uts-server relies on to run:

- `OpenSSL`.
- `civetweb`.
- on none GNU LibC, `argp-standalone`

Build dependencies

List of dependencies needed to build uts-server:

- `CMake`
- either `gcc` or `clang`

CHAPTER 2

Compilation

uts-server is compiled using cmake:

```
# If civetweb is already present on the system
$ cmake .
$ make

# If civetweb is not present.
# this will get the master branch of civetweb from upstream and compile it.
# Only for developpment/testing purposes
$ cmake . -DBUNDLE_CIVETWEB=ON
$ make

# Compile with debug flags
# Only for developpment/testing purposes
$ cmake . -DDEBUG=ON
$ make

# Compile statically
# (in some cases, it might be necessary to still
# link some libraries like dl or gcc_s or pthread, if necessary,
# add -DLINK_DL=ON and/or -DLINK_GCC_S=ON and/or -DLINK_PTHREAD=ON)
$ cmake . -DSTATIC=ON # -DLINK_DL=ON -DLINK_GCC_S=ON -DLINK_PTHREAD=ON
$ make
```

Warning: The BUNDLE_CIVETWEB exists only for developpment/testing purposes.

Please compile civetweb externally for building a production binary.

Using this option outside of developpment/testing is a bad idea for the following reasons:

- having an external download in a build process is a bad idea
- recovering the master branch ensures that the build may break randomly
- a build process should be reproducible which is not the case with this option

Debian

The installation requires installing the following packages:

```
# build dependencies  
$ apt-get install libssl-dev cmake clang
```

CentOS/RHEL

The installation requires installing the following packages:

```
# build dependencies  
$ yum install cmake gcc gcc-c++ openssl-devel
```

FreeBSD

The installation requires installing the following packages:

```
# build dependencies  
$ pkg add argp-standalone cmake
```

OpenBSD

The installation requires installing the following packages:

```
# build dependencies
$ pkg_add gcc g++ argp-standalone cmake

# for the test scripts
$ pkg_add python curl
```

To build you must egcc and eg++ (not the old 4.2 gcc in the base system)

```
# set compilers
$ export CC=/usr/local/bin/egcc
$ export CXX=/usr/local/bin/ec++

# then build normally
$ cmake . -DBUNDLE_CIVETWEB=ON && make
```

Configuration Parameters

Section [main]

Main configuration section (mostly http configuration).

Parameter	Description	Example Value
access_control_allow_origin	Comma separated list of IP subnets to accept/deny Ex: -0.0.0.0/0,+192.168.0.0/16 (deny all accesses, only allow 192.168.0.0/16 subnet)	-0.0.0.0/0,+192.168/16
enable_keep_alive	Allows clients to reuse TCP connection for subsequent HTTP requests, which improves performance.	no
listening_ports	Comma-separated list of IP:port tuples to listen on. If the port is SSL, a letter s must be appended. Ex: listening_ports = 80,443s	127.0.0.1:2020
log_level	Loglevel (debug, info, notice, warn, err, emerg, crit)	info
num_threads	Number of worker threads.	50
request_timeout_ms	Timeout for network read and network write operations. In milliseconds.	30000
run_as_user	Switch to given user credentials after startup. Required to run on privileged ports as non root user.	uts-server
ssl_ca_file	Path to a .pem file containing trusted certificates. The file may contain more than one certificate.	/etc/uts-server/ca.pem
ssl_ca_path	Name of a directory containing trusted CA certificates.	/etc/ssl/ca/
ssl_certificate	Path to the SSL certificate file . PEM format must contain private key and certificate.	/etc/uts-server/cert.pem
ssl_cipher_list	List of enabled ciphers for ssl. See https://www.openssl.org/docs/manmaster/apps/ciphers.html or 'man ciphers' for more detailed.	ALL:!eNULL:!SSLv3
ssl_default_verify_paths	Loads default trusted certificates locations set at OpenSSL compile time.	yes
ssl_protocol_version	Sets the minimal accepted version of SSL/TLS protocol according to the table: <ul style="list-style-type: none"> • SSL2+SSL3+TLS1.0+TLS1.1+TLS1.2 -> 0 • SSL3+TLS1.0+TLS1.1+TLS1.2 -> 1 • TLS1.0+TLS1.1+TLS1.2 -> 2 • TLS1.1+TLS1.2 -> 3 • TLS1.2 -> 4 	3
ssl_short_trust	Enables the use of short lived certificates	no
ssl_verify_depth	Sets maximum depth of certificate chain. If client's certificate chain is longer than the depth set here connection is refused.	9
ssl_verify_peer	Enable client's certificate verification by the server.	yes
tcp_nodelay	Enable TCP_NODELAY socket option	0

Section [oids]

Section for declaring OID mapping. Just add <name> = <OID> pairs.

Parameter	Description	Example Value
tsa_policy1		1.2.3.4.1
tsa_policy2		1.2.3.4.5.6
tsa_policy3		1.2.3.4.5.7

Section [tsa]

TSA configuration parameters.

Parameter	Description	Example Value
accuracy	Time-Stamp accuracy. (optional)	secs:1, millisecs:500, microsecs:100
certs	Certificate chain to include in reply. (optional)	\$dir/cacert.pem
clock_precision_digits	Number of decimals for Time-Stamp. (optional)	0
crypto_device	OpenSSL engine to use for signing.	builtin
default_policy	Policy if request did not specify it. (optional)	tsa_policy1
digests	Acceptable message digests. (mandatory) See https://www.openssl.org/docs/manmaster/apps/dgst.html or 'man dgst' to get the list of available digests	md5, sha1, sha224, sha256, sha384, sha512
dir	TSA root directory.	/etc/uts-server/pki
ess_cert_id_chain	Must the ESS cert id chain be included? (optional, default: no)	no
ordering	Is ordering defined for timestamps? (optional, default: no)	yes
other_policies	Acceptable policies. (optional)	tsa_policy2, tsa_policy3
signer_cert	The TSA signing certificat. (optional)	\$dir/tsacert.pem
signer_key	The TSA private key. (optional)	\$dir/private/tsakey.pem
tsa_name	Must the TSA name be included in the reply? (optional, default: no)	yes

CHAPTER 5

Full Configuration File

```
# Section for declaring OID mapping. Just add <name> = <OID> pairs.
[ oids ]

tsa_policy1 = 1.2.3.4.1
tsa_policy2 = 1.2.3.4.5.6
tsa_policy3 = 1.2.3.4.5.7

# Main configuration section (mostly http configuration).
[ main ]

# Comma-separated list of IP:port tuples to listen on.
# If the port is SSL, a letter s must be appended.
#
# Ex: listening_ports = 80,443s
listening_ports = 127.0.0.1:2020

# Allows clients to reuse TCP connection for subsequent
# HTTP requests, which improves performance.
enable_keep_alive = no

# Number of worker threads.
num_threads = 50

# Switch to given user credentials after startup.
# Required to run on privileged ports as non root user.
#run_as_user = uts-server

# Limit download speed for clients.
#
# Throttle is a comma-separated list of key=value pairs:
#
# - *                -> limit speed for all connections
#
# - x.x.x.x/mask -> limit speed for specified subnet
#
```

```
# The value is a floating-point number of bytes per second,
# optionally followed by a k or m character
# meaning kilobytes and megabytes respectively.
#
# A limit of 0 means unlimited rate.
#
# Ex: throttle = *=1k,10.10.0.0/16=10m,10.20.0.0/16=0
throttle = *=0

# Timeout for network read and network write operations.
# In milliseconds.
request_timeout_ms = 30000

# Path to the SSL certificate file .
# PEM format must contain private key and certificate.
#ssl_certificate = /etc/uts-server/cert.pem

# Enable client's certificate verification by the server.
#ssl_verify_peer = yes

# Name of a directory containing trusted CA certificates.
#ssl_ca_path = /etc/ssl/ca/

# Path to a .pem file containing trusted certificates.
# The file may contain more than one certificate.
#ssl_ca_file = /etc/uts-server/ca.pem

# Sets maximum depth of certificate chain.
# If client's certificate chain is longer
# than the depth set here connection is refused.
#ssl_verify_depth = 9

# Loads default trusted certificates
# locations set at OpenSSL compile time.
#ssl_default_verify_paths = yes

# List of enabled ciphers for ssl.
# See https://www.openssl.org/docs/manmaster/apps/ciphers.html
# or 'man ciphers' for more detailed.
#ssl_cipher_list = ALL:!eNULL:!SSLv3

# Sets the minimal accepted version of SSL/TLS protocol
# according to the table:
#
# - SSL2+SSL3+TLS1.0+TLS1.1+TLS1.2 -> 0
#
# - SSL3+TLS1.0+TLS1.1+TLS1.2      -> 1
#
# - TLS1.0+TLS1.1+TLS1.2          -> 2
#
# - TLS1.1+TLS1.2                 -> 3
#
# - TLS1.2                        -> 4
#ssl_protocol_version = 3

# Enables the use of short lived certificates
#ssl_short_trust = no
```



```
# Comma separated list of IP subnets to accept/deny
#
# Ex: -0.0.0.0/0,+192.168.0.0/16
# (deny all accesses, only allow 192.168.0.0/16 subnet)
#access_control_allow_origin = -0.0.0.0/0,+192.168/16

# Enable TCP_NODELAY socket option on client connections.
tcp_nodelay = 0

# Loglevel (debug, info, notice, warn, err, emerg, crit)
log_level = info

# TSA configuration parameters.
[ tsa ]

# TSA root directory.
dir = /etc/uts-server/pki

# OpenSSL engine to use for signing.
#crypto_device = builtin

# The TSA signing certificat. (optional)
signer_cert = $dir/tsacert.pem

# Certificate chain to include in reply. (optional)
certs = $dir/cacert.pem

# The TSA private key. (optional)
signer_key = $dir/private/tsakey.pem

# Policy if request did not specify it. (optional)
default_policy = tsa_policy1

# Acceptable policies. (optional)
other_policies = tsa_policy2, tsa_policy3

# Acceptable message digests. (mandatory)
# See https://www.openssl.org/docs/manmaster/apps/dgst.html
# or 'man dgst' to get the list of available digests
digests = md5, sha1, sha224, sha256, sha384, sha512

# Time-Stamp accuracy. (optional)
accuracy = secs:1, millisecs:500, microsecs:100

# Number of decimals for Time-Stamp. (optional)
clock_precision_digits = 0

# Is ordering defined for timestamps? (optional, default: no)
ordering = yes

# Must the TSA name be included in the reply? (optional, default: no)
tsa_name = yes

# Must the ESS cert id chain be included? (optional, default: no)
ess_cert_id_chain = no
```


Usage

```
$ ./uts-server --help
Usage: uts-server [OPTION...] -c CONFFILE [-d] [-D] [-p <pidfile>]

UTS micro timestamp server (RFC 3161)

  -c, --conffile=CONFFILE    Path to configuration file
  -d, --daemonize             Launch as a daemon
  -D, --debug                 STDOUT debugging
  -p, --pidfile=PIDFILE      Path to pid file
  -?, --help                  Give this help list
      --usage                 Give a short usage message
  -V, --version               Print program version

Mandatory or optional arguments to long options are also mandatory or optional
for any corresponding short options.

Report bugs to Pierre-Francois Carpentier <carpentier.pf@gmail.com>.
```

Running uts-sever

To debug problems with uts-server, run it in the foreground in debug mode:

```
# In debug mode with verbose debugging on stdout
$ ./uts-server -c <path/to/conf> -D
```

To run it as a daemon:

```
# In daemon mode
$ ./uts-server -c <path/to/conf> -d -p <path/to/pidfile>
```


0.1.9

- [fix] add explicit C standard (C99), fixes compilation with olders gcc/cmake

0.1.8

- [impr] add OpenBSD support
- [impr] add LibreSSL support

0.1.7

- [doc] add warnings to explicitly state the BUNDLE_CIVETWEB option as test/dev only
- [impr] add option for easily linking lib pthread (mainly for static linking)

0.1.6

- [fix] option declaration for LINK_GCC_S

0.1.5

- [impr] add support for a static build

0.1.4

- [impr] more portable code

0.1.3

- [impr] add support for FreeBSD

0.1.2

- [fix] adding support for OpenSSL 1.1 (with compatibility with 1.0)

0.1.1

- [fix] correct compilation issues in older gcc/clang caused by missing -D_XOPEN_SOURCE, missing -std and missing headers
- [impr] exit at the first TS_RESP_CTX (OpenSSL TS response context) initialization failed.

0.1.0

- [impr] adding various goodies (init scripts)
- [impr] safer crypto algorithm in configuration file
- [impr] removing useless default_tsa parameter in configuration file

0.0.3

- [fix] memleak on configuration parameters loading

0.0.2

- [fix] Fix loading of certificate in case of relative path

0.0.1

- First version

Time-Stamp script combining curl and openssl

```
#!/bin/sh

RCol='\33[0m'      # Text Reset

# Regular          Bold          Underline          High Intensity
↪BoldHigh Intens
Bla='\33[0;30m';    BBlu='\33[1;30m';    UBlu='\33[4;30m';    IBla='\33[0;90m';
↪BIBla='\33[1;90m';
Red='\33[0;31m';    BRed='\33[1;31m';    URed='\33[4;31m';    IRed='\33[0;91m';
↪BIRed='\33[1;91m';
Gre='\33[0;32m';    BGre='\33[1;32m';    UGre='\33[4;32m';    IGre='\33[0;92m';
↪BIGre='\33[1;92m';
Yel='\33[0;33m';    BYel='\33[1;33m';    UYel='\33[4;33m';    IYel='\33[0;93m';
↪BIYel='\33[1;93m';
Blu='\33[0;34m';    BBlu='\33[1;34m';    UBlu='\33[4;34m';    IBlu='\33[0;94m';
↪BIBlu='\33[1;94m';
Pur='\33[0;35m';    BPur='\33[1;35m';    UPur='\33[4;35m';    IPur='\33[0;95m';
↪BIPur='\33[1;95m';
Cya='\33[0;36m';    BCya='\33[1;36m';    UCya='\33[4;36m';    ICya='\33[0;96m';
↪BICya='\33[1;96m';
Whi='\33[0;37m';    BWhi='\33[1;37m';    UWhi='\33[4;37m';    IWhi='\33[0;97m';
↪BIWhi='\33[1;97m';

SYSLOG=1

help() {
    cat <<EOF

usage: `basename $0` -i <input file> -u <ts server url> \
    -o <output ts file> -O <openssl options> -C <curl options>
```

```

HTTP timestamping client using openssl and curl (RFC 3161)

arguments:

* mandatory:
  -i <input file>:      the input file to timestamp
  -u <ts server url>:   the timestamp server url

* optionnal:
  -l                : enable logging to syslog
  -o <output ts file>: output timestamp file name (default: <input file>.ts)
  -O <openssl options>: openssl additionnal options (man ts for more details)
  -C <curl options>:  curl additionnal options (man curl for more details)

EOF
  exit 1
}

simple_logger(){
  [ $SYSLOG -eq 0 ] && logger -t `basename $0` -p user.$1 $2
}

clean(){
  rm -f -- "$TMPREQ"
}

clean_exit(){
  clean
  exit 1
}

exit_error(){
  msg=$1
  simple_logger err "error, $msg"
  printf "${BIRed}[ERROR]    ${IYel}%s${RCol}\n" "$msg"
  clean_exit
}

info(){
  msg=$1
  simple_logger debug "$msg"
  printf "${BIBlu}[INFO]      ${RCol}%s${RCol}\n" "$msg"
}

success(){
  msg=$1
  simple_logger info "$msg"
  printf "${BIGre}[SUCCESS]  ${RCol}%s${RCol}\n" "$msg"
}

trap clean_exit HUP INT TERM
TMPREQ=`mktemp`

REMOVE_TS=0

while getopts ":lhru:i:o:O:C:" opt; do
  case $opt in
    h) help;;

```



```

1)  SYSLOG=0;;
u)  TS_URL="$OPTARG";;
i)  INPUT_FILE="`readlink -f $OPTARG`";;
o)  OUTPUT_FILE="`readlink -f $OPTARG`";;
O)  OPENSSSL_OPTS="$OPTARG";;
C)  CURL_OPTS="$OPTARG";;
r)  REMOVE_TS=1;;
\?) echo "Invalid option: -$OPTARG" >&2; help; exit 1;;
:)  echo "Option -$OPTARG requires an argument." >&2; help; exit 1;;
esac
done

# If no output file specified, output to <input file>.ts
[ -z "$OUTPUT_FILE" ]    && OUTPUT_FILE="${INPUT_FILE}.tsr"

# Check that input file exists
[ -f "$INPUT_FILE" ]    || exit_error "Input file '$INPUT_FILE' doesn't exist"
# Check that output file doesn't exist
if [ $REMOVE_TS -eq 1 ]
then
    [ -f "$OUTPUT_FILE" ] && rm -f "$OUTPUT_FILE"
else
    ! [ -f "$OUTPUT_FILE" ] || exit_error "Output timestamp file '$OUTPUT_FILE'
    ↳ already exists"
fi
# Check that url is not empty
! [ -z "$TS_URL" ]      || exit_error "Missing timestamp server url"

info "Generating timestamp on file '$INPUT_FILE', to '$OUTPUT_FILE', using server '
    ↳ $TS_URL'"

# Building the timestamp request with openssl
openssl ts $OPENSSSL_OPTS \
    -query -data "$INPUT_FILE" \
    -out "$TMPREQ" || exit_error "Request generation failed"

# Submitting the timestamp request to the RFC 3161 server with curl
curl "$TS_URL" $CURL_OPTS \
    -H "Content-Type: application/timestamp-query" \
    -f -g \
    --data-binary @$TMPREQ \
    -o "$OUTPUT_FILE" 2>/dev/null || exit_error "Timestamp query failed"

openssl ts -verify -data "$INPUT_FILE" -in "$OUTPUT_FILE" 2>&1 | grep -q "asn1_
    ↳ encoding routines" && exit_error \
    "Reponse doesn't appear to be a timestamp response"

success "Timestamp of file '$INPUT_FILE' using server '$TS_URL' succeed, ts written_
    ↳ to '$OUTPUT_FILE'"

clean

```


CHAPTER 9

uts-server

Micro [RFC 3161 Time-Stamp](#) server written in C.

Doc [Uts-Server documentation on ReadTheDoc](#)

Dev [Uts-Server source code on GitHub](#)

License MIT

Author Pierre-Francois Carpentier - copyright © 2016

License

Released under the MIT Public License

What is RFC 3161?

An RFC 3161 time-stamp is basically a cryptographic signature with a date attached.

Roughly, it works as follow:

1. A client application sends an hash of the data it wants to time-stamp to a Time-Stamp authority server.
2. The Time-Stamp authority server retrieves the current date, concatenates it with the hash and uses its private key to create the time-stamp (kind of like a signature).
3. The Time-Stamp authority server returns the generated time-stamp to the client application.

Then a client can verify the piece of data with the time-stamp using the Certificate Authority of the time-stamp key pair (X509 certificates).

It gives a cryptographic proof of a piece of data content, like a file, at a given time.

Some use cases:

- time-stamp log files at rotation time.
- time-stamp file at upload to prove it was delivered in due time or not.

Quick (and dirty) Testing

Here a few steps to quickly trying out uts-server, for production setup, please compile civetweb externally and create proper CA and certificates:

```
# Building with civetweb embedded (will recover civetweb from github).
# Note: the BUNDLE_CIVETWEB option is only here for fast testing purpose
# The recommended way to deploy uts-server in production is to build civetweb
# separately and to link against it.
$ cmake . -DBUNDLE_CIVETWEB=ON
$ make

# Create some test certificates.
$ ./tests/cfg/pki/create_tsa_certs

# Launching the time-stamp server with test configuration in debug mode.
$ ./uts-server -c tests/cfg/uts-server.cnf -D

# In another shell, launching a time-stamp script on the README.md file.
$ ./goodies/timestamp-file.sh -i README.rst -u http://localhost:2020 -r -O "-cert";

# Verify the time-stamp.
$ openssl ts -verify -in README.rst.tsr -data README.rst -CAfile ./tests/cfg/pki/
->tsaca.pem

# Display the time-stamp content.
$ openssl ts -replay -in README.rst.tsr -text
```

Powered by