16. Encryption

Overview

Zabbix supports encrypted communications between Zabbix server, Zabbix proxy, Zabbix agent, zabbix_sender and zabbix_get utilities using Transport Layer Security (TLS) protocol v.1.2. Encryption is supported starting with Zabbix 3.0. Certificate-based and pre-shared key-based encryption is supported.

Encryption is optional and configurable for individual components (e.g. some proxies and agents can be configured to use certificate-based encryption with the server, while others can use pre-shared key-based encryption, and yet others continue with unencrypted communications as before).

Server (proxy) can use different encryption configurations for different hosts.

Zabbix daemon programs use one listening port for encrypted and unencrypted incoming connections. Adding an encryption does not require opening new ports on firewalls.

Limitations

- Private keys are stored in plain text in files readable by Zabbix components during startup.
- Pre-shared keys are entered in Zabbix frontend and stored in Zabbix database in plain text.
- Built-in encryption does not protect communications:
    - between web server running Zabbix frontend and user web browser,
    - between Zabbix frontend and Zabbix server,
    - between Zabbix server (proxy) and Zabbix database.
- Currently each encrypted connection opens with a full TLS handshake, no session caching and tickets are implemented.
- Adding encryption increases time of checks and actions, depending on network latency.
  For example, if packet delay is 100ms then opening a TCP connection and sending unencrypted request takes around 200ms.
  With encryption about 1000 ms are added for establishing TLS connection.
  Timeouts may need to be increased, otherwise some items and actions running remote scripts on agents may work with unencrypted connections but fail with timeout with encrypted.
- Encryption is not supported by network discovery. Zabbix agent checks performed by network discovery will be unencrypted and if Zabbix agent is configured to reject unencrypted connections such checks will not succeed.

Compiling Zabbix with encryption support

To support encryption Zabbix must be compiled and linked with one of four crypto libraries:

- *GnuTLS* (from version 3.1.18)
- *OpenSSL* (versions 1.0.1, 1.0.2, 1.1.0). *OpenSSL* 1.1.1 is supported from Zabbix version 3.0.23.
- *LibreSSL* (tested with versions 2.7.4, 2.8.2) is supported from Zabbix version 3.0.26. *LibreSSL* 2.6.x is not supported. *LibreSSL* is supported as a compatible replacement of *OpenSSL*, the new `tls_*()` *LibreSSL*-specific API functions are not used. Zabbix components compiled with

LibreSSL will not be able to use PSK, only certificates can be used.
- *mbed TLS* (formerly *PolarSSL*)(version 1.3.9 and later 1.3.x). *mbed TLS* 2.x is not currently supported, it is not a drop-in replacement for 1.3 branch, Zabbix will not compile with *mbed TLS* 2.x.

The library is selected by specifying an option to "configure" script:

- `--with-gnutls[=DIR]`
- `--with-openssl[=DIR]` (also used for *LibreSSL*)
- `--with-mbedtls[=DIR]`

For example, to configure the sources for server and agent with *OpenSSL* you may use something like:

```
./configure --enable-server --enable-agent --with-mysql --enable-ipv6 --
with-net-snmp --with-libcurl --with-libxml2 --with-openssl
```

Different Zabbix components may be compiled with different crypto libraries (e.g. a server with *OpenSSL*, an agent with *GnuTLS*).

In our tests *OpenSSL* was the fastest, followed by *GnuTLS*.

If you plan to use pre-shared keys (PSK) consider using *GnuTLS*, newer *OpenSSL* (from 1.1.0) or *mbed TLS* libraries in Zabbix components using PSKs. These libraries support PSK ciphersuites with Perfect Forward Secrecy. Older *OpenSSL* libraries (versions 1.0.1, 1.0.2c) do support PSKs but available PSK ciphersuites do not provide Perfect Forward Secrecy.

Connection encryption management

Connections in Zabbix can use:

- no encryption (default)
- RSA certificate-based encryption
- PSK-based encryption

There are two important parameters used to specify encryption for connections between Zabbix components:

- `TLSConnect`
- `TLSAccept`

`TLSConnect` specifies what encryption to use for outgoing connections and can take one of 3 values (`unencrypted`, PSK, `certificate`). `TLSConnect` is used in configuration files for Zabbix proxy (in active mode, specifies only connections to server) and Zabbix agentd (for active checks). In Zabbix frontend the `TLSConnect` equivalent is *Connections to host* field in *Configuration→Hosts→<some host>→Encryption* tab and *Connections to proxy* field in *Administration→Proxies→<some proxy>→Encryption* tab. If the configured encryption type for connection fails, no other encryption types will be tried.

`TLSAccept` specifies what types of connections are allowed for incoming connections. Connection types are: `unencrypted`, PSK, `certificate`. One or more values can be specified. `TLSAccept` is used in configuration files for Zabbix proxy (in passive mode, specifies only connections from server)

and Zabbix agentd (for passive checks). In Zabbix frontend the TLSAccept equivalent is *Connections from host* field in *Configuration→Hosts→<some host>→Encryption* tab and *Connections from proxy* field in *Administration→Proxies→<some proxy>→Encryption* tab.

Normally you configure only one type of encryption for incoming encryptions. But you may want to switch encryption type, e.g. from unencrypted to certificate-based with minimum downtime and rollback possibility.
To achieve this you can set `TLSAccept=unencrypted,cert` in agentd configuration file and restart Zabbix agent.
Then you can test connection with `zabbix_get` to the agent using certificate. If it works, you can reconfigure encryption for that agent in Zabbix frontend in *Configuration→Hosts→<some host>→Encryption* tab by setting *Connections to host* to "Certificate".
When server configuration cache gets updated (and proxy configuration is updated if the host is monitoring by proxy) then connections to that agent will be encrypted.
If everything works as expected you can set `TLSAccept=cert` in agent configuration file and restart Zabbix agent.
Now the agent will be accepting only encrypted certificate-based connections. Unencrypted and PSK-based connections will be rejected.

In a similar way it works on server and proxy. If in Zabbix frontend in host configuration *Connections from host* is set to "Certificate" then only certificate-based encrypted connections will be accepted from agent (active checks) and `zabbix_sender` (trapper items).

Most likely you will configure incoming and outgoing connections to use the same encryption type or no encryption at all. But technically it is possible to configure it asymmetrically, e.g. certificate-based encryption for incoming and PSK-based for outgoing connections.

For overview, encryption configuration for each host is displayed in Zabbix frontend *Configuration→Hosts* on the right side, in column *AGENT ENCRYPTION*. Configuration display examples:

Example	**Connections TO host**	**Allowed connections FROM host**	**Rejected connections FROM host**
NONE	Unencrypted	Unencrypted	Encrypted certificate and PSK-based
CERT NONE PSK CERT	Encrypted, certificate-based	Encrypted certificate-based	Unencrypted and PSK-based
PSK NONE PSK CERT	Encrypted, PSK-based	Encrypted PSK-based	Unencrypted and certificate-based
PSK NONE PSK CERT	Encrypted, PSK-based	Unencrypted and PSK-based encrypted	Certificate-based
CERT NONE PSK CERT	Encrypted, certificate-based	Unencrypted, PSK or certificate-based encrypted	-

Default is unencrypted connections. Encryption must be configured for each host and proxy individually.

zabbix_get and zabbix_sender with encryption

See man-pages zabbix_get and zabbix_sender for using them with encryption.

Ciphersuites

Ciphersuites are configured internally during Zabbix startup and depend on crypto library, currently they are not user-configurable.

Configured ciphersuites by library type in order from higher to lower priority:

Library	Certificate ciphersuites	PSK ciphersuites
mbed TLS (PolarSSL) 1.3.9	TLS-ECDHE-RSA-WITH-AES-128-GCM-SHA256 TLS-ECDHE-RSA-WITH-AES-128-CBC-SHA256 TLS-ECDHE-RSA-WITH-AES-128-CBC-SHA TLS-RSA-WITH-AES-128-GCM-SHA256 TLS-RSA-WITH-AES-128-CBC-SHA256 TLS-RSA-WITH-AES-128-CBC-SHA	TLS-ECDHE-PSK-WITH-AES-128-CBC-SHA256 TLS-ECDHE-PSK-WITH-AES-128-CBC-SHA TLS-PSK-WITH-AES-128-GCM-SHA256 TLS-PSK-WITH-AES-128-CBC-SHA256 TLS-PSK-WITH-AES-128-CBC-SHA
GnuTLS 3.1.18	TLS_ECDHE_RSA_AES_128_GCM_SHA256 TLS_ECDHE_RSA_AES_128_CBC_SHA256 TLS_ECDHE_RSA_AES_128_CBC_SHA1 TLS_RSA_AES_128_GCM_SHA256 TLS_RSA_AES_128_CBC_SHA256 TLS_RSA_AES_128_CBC_SHA1	TLS_ECDHE_PSK_AES_128_CBC_SHA256 TLS_ECDHE_PSK_AES_128_CBC_SHA1 TLS_PSK_AES_128_GCM_SHA256 TLS_PSK_AES_128_CBC_SHA256 TLS_PSK_AES_128_CBC_SHA1
OpenSSL 1.0.2c	ECDHE-RSA-AES128-GCM-SHA256 ECDHE-RSA-AES128-SHA256 ECDHE-RSA-AES128-SHA AES128-GCM-SHA256 AES128-SHA256 AES128-SHA	PSK-AES128-CBC-SHA
OpenSSL 1.1.0	ECDHE-RSA-AES128-GCM-SHA256 ECDHE-RSA-AES128-SHA256 ECDHE-RSA-AES128-SHA AES128-GCM-SHA256 AES128-CCM8 AES128-CCM AES128-SHA256 AES128-SHA	ECDHE-PSK-AES128-CBC-SHA256 ECDHE-PSK-AES128-CBC-SHA PSK-AES128-GCM-SHA256 PSK-AES128-CCM8 PSK-AES128-CCM PSK-AES128-CBC-SHA256 PSK-AES128-CBC-SHA

Cipher suites using certificates:

	TLS server		
TLS client	*mbed TLS (PolarSSL)*	*GnuTLS*	*OpenSSL 1.0.2*
mbed TLS (PolarSSL)	TLS-ECDHE-RSA-WITH-AES-128-GCM-SHA256	TLS-ECDHE-RSA-WITH-AES-128-GCM-SHA256	TLS-ECDHE-RSA-WITH-AES-128-GCM-SHA256
GnuTLS	TLS-ECDHE-RSA-WITH-AES-128-GCM-SHA256	TLS-ECDHE-RSA-WITH-AES-128-GCM-SHA256	TLS-ECDHE-RSA-WITH-AES-128-GCM-SHA256
OpenSSL 1.0.2	TLS-ECDHE-RSA-WITH-AES-128-GCM-SHA256	TLS-ECDHE-RSA-WITH-AES-128-GCM-SHA256	TLS-ECDHE-RSA-WITH-AES-128-GCM-SHA256

Cipher suites using PSK:

	TLS server		
TLS client	*mbed TLS (PolarSSL)*	*GnuTLS*	*OpenSSL 1.0.2*
mbed TLS (PolarSSL)	TLS-ECDHE-PSK-WITH-AES-128-CBC-SHA256	TLS-ECDHE-PSK-WITH-AES-128-CBC-SHA256	TLS-PSK-WITH-AES-128-CBC-SHA
GnuTLS	TLS-ECDHE-PSK-WITH-AES-128-CBC-SHA256	TLS-ECDHE-PSK-WITH-AES-128-CBC-SHA256	TLS-PSK-WITH-AES-128-CBC-SHA
OpenSSL 1.0.2	TLS-PSK-WITH-AES-128-CBC-SHA	TLS-PSK-WITH-AES-128-CBC-SHA	TLS-PSK-WITH-AES-128-CBC-SHA