mozilla

Winter of Security

Multifactor Authentication and Session Resumption in OpenVPN

Guillaume Destuynder (kang)
Harshvardhan Sharma (harsh1618)
Shivanshu Agrawal (shivanshuag)
Srijan R. Shetty (srijanshetty)

Team

- We’re a group of three undergraduate students currently at the Indian Institute of Technology Kanpur.
- We like contributing back to the open source community as much as we can and this is partly what inspired us to work on MWoS (that and credits for working on open source, what else can one want. And yeah, free T-shirts!).

Virtual Private Network

- Allows two devices to securely communicate with each other over a possibly insecure public network.
- Allows for secure communications between different private networks over an insecure network through tunnelling

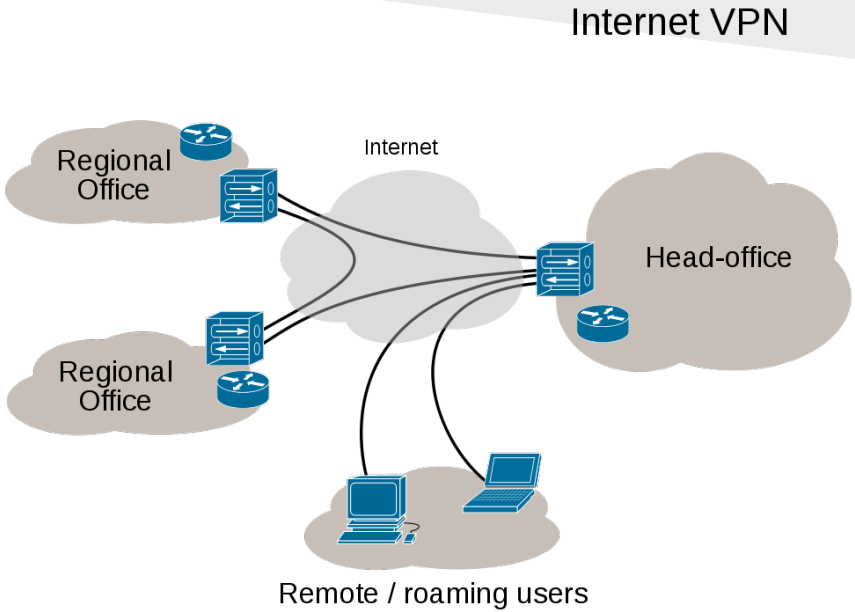


Image Credits: Wikimedia Commons

Mozilla and OpenVPN

- Mozilla uses OpenVPN to allow its employees to securely connect to its Private Network.
- OpenVPN's MFA model currently has the following issues:
  - Only one factor of authentication
  - The password field is reused to implement MFA, i.e., enter OTP in password field. (Thereby motivating the need of true MFA)
  - No session support, user has to enter OTP for every connection

Authentication Modes in OpenVpn

OpenVPN provides for two authentication modes:

- Static Key: In this mode, a key is generated and shared between the users before the establishment of a tunnel.

- TLS: In this mode, a bidirectional session using certificates is established. On a successful TLS/SSL authentication, tunnel keys are established for communication.

Username Password Authentication

OpenVPN also allows users to authenticate using a username and password. The password is checked after successful TLS authentication, during the phase when session keys for the VPN tunnel are established.

Strong Passwords

Multi-Factor Authentication

- MFA relies on the following factors of authentication:

  1. Knowledge Factor : ATM Pins, Passwords
  2. Possession Factor : Smart Cards
  3. Inherence Factor : Biometrics

- The underlying assumption is that it is difficult to get hold of more than one of these three factors.

Session Resumption

- Maintaining a cookie on the client side so that one does not have to authenticate every time one logs in.
- Cookie should have an expiration time.
- This is the standard flow adopted by most websites.

Multifactor Authentication

Challenges

- Understanding the architecture of OpenVPN
- Secure Coding
- Backwards Compatibility
- Multiple Possible MFA types and implementations

MFA Implementation

- Extended the original packet used to exchange key material between client and server for establishing session key

- Added MFA username and password fields

Three types Of MFA Methods

- **PUSH**
  - This will not ask for any credentials from the user.
  - Useful in case of authentication by Push Notifications to a registered smartphone.

Three types Of MFA Methods

- **OTP**
  - Only password is asked from the user.
  - Useful in the case when smartphone of the user has an app which generates the OTP.
  - E.g. Google Authenticator

Three types Of MFA Methods

- **User-Pass**
  - Both username and password is asked from the user.
  - Useful when need to provide an identifier along with the OTP.
  - Also for session support in username, password authentication.

Backwards Compatibility

- Added a flag (bitmask) in the authentication packet header when compiled with MFA support
- Server side config option

```
mfa-backward-compat
```

- When backwards compatibility is enabled, server allows older clients (or clients with MFA disabled) to bypass MFA
- When disabled, auth fails if MFA is not supported
- Can be enabled in the transition phase when all clients have not upgraded

Backwards Compatibility

Compatibility	Server	Client	Action
Enabled	New Server, MFA-enabled	New Client, MFA-Enabled New Client, MFA-Disabled Old Client	MFA-auth auth-failure old-auth
Disabled	New Server, MFA-enabled	Old Client	auth-failure
*	New Server, MFA-disabled	New Client, MFA-Enabled New Client, MFA-disabled	MFA-auth old-auth
*	New Server, MFA-disabled	Old Client	old-auth
*	Old Server	New Client, MFA-Enabled New Client, MFA-Disabled	MFA-auth MFA-auth

Configuration

- Server:

  ```
  mfa-method [mfa-type] [script-file-name] [via-env/via-file]
  Ex: mfa-method otp auth.pl via-file

  mfa-method [method-type]
  plugin [plugin-shared-object-file]
  ```

- Client:

  ```
  mfa-method [method-type]
  ```

Session Support

Session Support

Generate a token which can be used for session resumption.

Similar to web-based session resumption (cookies)

Challenges

Security

- The session token used should be tied to the client’s identity.
- It should not be possible for any entity other than the server to generate a valid token for any client within a reasonable amount of time.

Challenges

End User Experience

- Entire process should be transparent to the user.
- Enabling support for session resumption should require minimal changes to the client and server configuration files.

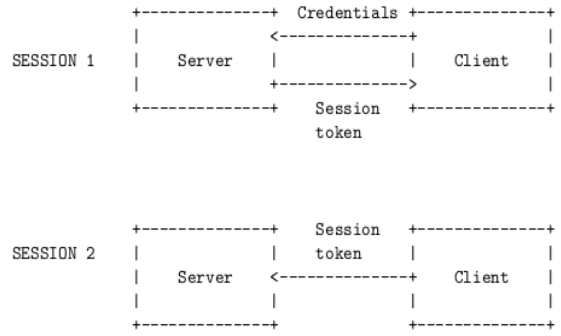
Challenges

Protocol

- The entire procedure of session resumption should fit into the existing OpenVPN protocol so as to maintain backwards compatibility.

Implementation

- Server generates a key (48 bytes) on startup.
- On successful auth, the server generates a token using the key.
- The token and expiry timestamp are sent back to the client. The client stores them on a local file.
- During next authentication, the timestamp and the token are sent instead of the MFA credentials.
- The server verifies that the timestamp and the token are valid.



Session Token Generation

Preliminary

HMAC (Hash based Message Authentication Code) [RFC 2104]

- Uses a cryptographic hash function to generate a ‘tag’ for a message using a secret key.
- Both integrity and authenticity of the data can be verified by re-calculating the HMAC using the same key.

$$HMAC(K, m) = H((K+opad) + H((K+ipad)+m))$$

where H: Hash function (e.g. SHA256, MD5)

K: Key, m: message

ipad,opad: padding bytes

+: concatenation

Session Token Generation

A **Pseudo-random function** (based on RFC 4346 - TLS 1.1) is used to generate the session token

Session token = PRF((CN + Timestamp), Key)

To generate the PRF, the key K is split into two equal parts K_1 and K_2

PRF (K, data) = P_MD5(K_1, data) $\oplus$ P_SHA1(K_2, data)

P_MD5 and P_SHA1 use HMAC_MD5 and HMAC_SHA1 respectively to generate arbitrary length outputs

Session Token Verification

- Use timestamp to check if the token has expired
- Generate a token using the client's CN and the received timestamp. If it matches the token received from the client, MFA authentication succeeds.

The token check ensures that the original token was issued to the same client and the timestamp received is authentic.

No need to store any session information on the server.

Modified Packet

```
+-----------------------------------------------------+
|    TLS plaintext packet (if key_method == 2):       |
+-----------------------------------------------------+
|  Literal 0 (4 bytes).                               |
|  key_method type (1 byte).                          |
|  key_source structure (pre_master only              |
|      defined for client -> server).                 |
|  options_string_length, including null (2 bytes).   |
|  Options string (n bytes, null terminated,          |
|          client/server options string must match).  |
|                                                     |
|  [The username/password data below is optional]     |
|  username_string_length, including null (2 bytes).  |
|  Username string (n bytes, null terminated).        |
|  password_string_length, including null (2 bytes)   |
|  Password string (n bytes, null terminated).        |
+-----------------------------------------------------+
```

```
+-----------------------------------------------------+
|    TLS plaintext packet (if key_method == 2):       |
+-----------------------------------------------------+
|  Literal 0 (4 bytes).                               |
|  key_method type (1 byte).                          |
|  key_source structure (pre_master only              |
|      defined for client -> server).                 |
|  options_string_length, including null (2 bytes).   |
|  Options string (n bytes, null terminated,          |
|          client/server options string must match).  |
|                                                     |
|  [The username/password data below is optional]     |
|  username_string_length, including null (2 bytes).  |
|  Username string (n bytes, null terminated).        |
|  password_string_length, including null (2 bytes)   |
|  Password string (n bytes, null terminated).        |
|                                                     |
|  [The MFA data below is optional]                   |
|  mfa_username_string_length, including null (2 bytes).|
|  MFA Username string (n bytes, null terminated).    |
|  mfa_password_string_length, including null (2 bytes) |
|  MFA Password string (n bytes, null terminated).    |
|                                                     |
|  [Session resumption data (optional)]               |
|  session_timestamp_length, including null (2 bytes). |
|  Session token timestamp (n bytes, null terminated). |
|  Session token length, including null (2 bytes)     |
|  Session token (n bytes, null terminated).          |
+-----------------------------------------------------+
```

Configuration

Client:

```
mfa-session-file <filename>
```

In the absence the above configuration parameter, the user is warned and session resumption is disabled.

Server:

```
mfa-session-expiration session-validity (in hours)
```

Future Work

- Get our work upstreamed.

MFA Demo

Questions