Secure shell over HTTP/3 connections

draft-michel-ssh3

François Michel
Secure Shell (SSH) services

SSH(v2) provides secure services on a remote host over an insecure network.

- Remote interactive shell access
- Remote commands execution
- TCP port forwarding
- X11 forwarding
SSHv2 architecture

SSHv2

SSH Transport
RFC4253
Secure channel establishment

SSH Authentication
RFC4252
User authentication

SSH Connection
RFC4254
SSH services

TCP
SSHv2 architecture

Implementing SSH means implementing RFC4253, RFC5252 and RFC4254.
Implementing SSH means implementing RFC4253, RFC5252 and RFC4254. Adding new crypto algorithms (e.g. PQC) means updating RFC4253 and every implementation.
SSHv2 limitations

- Cannot forward UDP
  - QUIC cannot be forwarded through SSH
  - Can't provide access to real-time resources
- Can easily be blocked or detected
  - blocking port 22
  - Blocking TCP connections exchanging SSHv2 version strings
- TCP is subject to RST/seqnum manipulation attacks
  - The TERRAPIN attack was about tampering with the TCP sequence number
  - TCP-AO addresses it for BGP, but not for SSH.
- Not well integrated with modern web infrastructures
  - These infrastructure propose their own HTTP-based authentication methods (SAML/OIDC/WebAuthn/…)
  - Often requires middlewares and tunnelling to integrate remote shells
- Evolves in parallel to TLS and HTTP, which also propose security and authentication
  - SSH certs are not well-defined and still not widely used. Certs used everywhere in HTTP/TLS.
  - Many individuals own HTTPS servers but still ssh to it using host keys and Trust-On-First-Use.
SSH over HTTP/3 architecture

SSH3

SSH Connection
~RFC4254

SSH services
- user authentication
- URL multiplexing

HTTP/3

QUIC / TLS
- reliable transport
- secure channel establishment
- stream multiplexing
- datagrams
- connection migration
SSH over HTTP/3 architecture

Security and transport can evolve independently from SSH, implementation focuses on SSH services.
SSH over HTTP/3 architecture

Security and transport can evolve independently from SSH, implementation focuses on SSH services. New features come along these modern protocols.
SSH3 conversation establishment

Client

QUIC(/TLS) handshake

Server
SSH3 conversation establishment

Client → QUIC(TLS) handshake → Server

HTTP/3

:method  = "CONNECT"
:protocol = "ssh3"
:path     = <any path>
Authorization = <Any HTTP auth>
SSH3 conversation establishment

Client

QUIC(TLS) handshake

Server

HTTP/3

:method = "CONNECT"
:protocol = "ssh3"
:path = <any path>
Authorization = <Any HTTP auth>

:status = 200 (OK)

SSH3 conversation established
SSH3 conversation establishment

**Client**

**Server**

**QUIC(TLS) handshake**

**HTTP/3**

:method = "CONNECT"
:protocol = "ssh3"
:path = <any path>
Authorization = <Any HTTP auth>

:status = 401 (Unauthorized)
Opportunities: Interesting features of HTTP/3

- Accessing the QUIC API
  - Stream multiplexing
Opportunities: Interesting features of HTTP/3

- Accessing the QUIC API
  - Stream multiplexing
Opportunities: Interesting features of HTTP/3

- Accessing the QUIC API
  - Stream multiplexing
  - Datagrams
Opportunities: Interesting features of HTTP/3

- Accessing the QUIC API
  - Stream multiplexing
  - Datagrams
  - Connection migration

```
SSH3 Server
```

```
2001:db8:a::1
```

```
2001:db8:b::1
```

```
migrate QUIC connection
```

```
new IP address
```
Opportunities: Interesting features of HTTP/3

- Accessing the QUIC API
- Encrypted/authenticated QUIC transport fields
Opportunities: Interesting features of HTTP/3

- Accessing the QUIC API
- Encrypted/authenticated QUIC transport fields
- Access to the X.509 ecosystem with low effort

```
root@ssh3:/home/azureuser# ssh3-server -generate-public-cert ssh3.eastus.cloudapp.azure.com -url-path /my-secret-url-path
password login is disabled
Generate public certificates...
1.705435568521185e+09 info waiting on internal rate limiter  
   "identifiers": ["ssh3.eastus.cloudapp.azure.com"], "ca": "https://acme-v02.api.letsencrypt.org/directory", "account": ""
1.705435568529472e+09 info done waiting on internal rate limiter  
   "identifiers": ["ssh3.eastus.cloudapp.azure.com"], "ca": "https://acme-v02.api.letsencrypt.org/directory", "account": ""
1.705435561253034e+09 info acme_client authorization finalized  
   "identifier": "ssh3.eastus.cloudapp.azure.com", "authz_status": 1
1.705435561253451e+08 info acme_client validations succeeded; finalizing order  
   "order": "https://acme-v02.api.letsencrypt.org/acme/
1.7054355618398514e+09 info acme_client successfully downloaded available certificate chains  
   "count": 2, "first_url": "https://acme-v02.api.letsencrypt.org/acme/
Successfully generated public certificates
Server started, listening on [::]:443/my-secret-url-path
```

Starting an SSH3 server for the 1st time and generating a public certificate for the domain name automatically
Opportunities: Interesting features of HTTP/3

- Accessing the QUIC API
- Encrypted/authenticated QUIC transport fields
- Access to the X.509 ecosystem with low effort
- URL multiplexing
Opportunities: Interesting features of HTTP/3

- Accessing the QUIC API
- Encrypted/authenticated QUIC transport fields
- Access to the X.509 ecosystem with low effort
- URL multiplexing
- HTTP Authentication
Opportunities: Interesting features of HTTP/3

- Accessing the QUIC API
- Encrypted/authenticated QUIC transport fields
- Access to the X.509 ecosystem with low effort
- URL multiplexing
- HTTP Authentication

By default in SSHv2
Opportunities: Interesting features of HTTP/3

- Accessing the QUIC API
- Encrypted/authenticated QUIC transport fields
- Access to the X.509 ecosystem with low effort
- URL multiplexing
- HTTP Authentication

By default in SSHv2
Opportunities: Interesting features of HTTP/3

- Accessing the QUIC API
- Encrypted/authenticated QUIC transport fields
- Access to the X.509 ecosystem with low effort
- URL multiplexing
- HTTP Authentication
- It fits well how the Internet is evolving
Threat: TCP-only networks

SSHv2 runs over TCP port 22 that is well supported on the Internet.

HTTP/3 currently runs over UDP port 443.

- It may be blocked by default in many networks.

Possible solutions:

- Run SSH over HTTP/2 as well
- QUIC on Streams?
- Run SSH over WebTransport that supports both QUIC and TCP
What’s next?

Is it interesting?

What would be the best outcome for this proposal?

- An actual SSHv3 candidate?
- Integrating SSH3 to MASQUE? (e.g. CONNECT-SHELL, CONNECT-PROCESS)
- Other design? (e.g. SSH over TLS, over QUIC, …)

Anyone interested to collaborate on the draft?

Is it interesting enough to integrate an existing wg or start discussions on a list?

Anyone wants to implement and interoperate?

Current design: draft-michel-ssh3-00.
Implem on Github: https://github.com/francoismichel/ssh3
Additional slides
Compatibility with SSHv2

Shipping SSH3 in newly deployed VMs should be easy.

However, SSHv2 is ubiquitous and will probably be around for years to come

- Like HTTP/1.1 and HTTP/2 coexist with HTTP/3

“upgrade” mechanisms (SSHv2->SSH3) could be defined & cached, using e.g. SSHv2’s version string:

```
SSH protoversion-softwareversion SP comments CR LF
```
Modern transports

Modern protocols developed after SSHv2 bring several benefits.

- TLS 1.3: reduced handshake, early data, SNI multiplexing, X.509
- QUIC: encrypted control information, streams multiplexing, datagrams, connection migration
- HTTP/3: URL multiplexing, authentication methods, Extended CONNECT

The IETF work of SSH is still done in parallel of TLS/QUIC/HTTPS.
Opportunities: Interesting features of HTTP/3

- Accessing the QUIC API
- Encrypted/authenticated QUIC transport fields
- Access to the X.509 ecosystem with low effort
- URL multiplexing
- Undiscoverability if demanded
Opportunities: Interesting features of HTTP/3

- Accessing the QUIC API
- Encrypted/authenticated QUIC transport fields
- Access to the X.509 ecosystem with low effort
- URL multiplexing
- Undiscoverability if demanded

Hidden SSH3 endpoint: drop packet on wrong port, respond 404 on wrong request
SSHv2 Connection establishment

Client

TCP connection established

Server

SSHv2

SSH Transport

SSH Authentication

SSH Connection

TCP
SSHv2 Connection establishment

TCP connection established

SYN
SYN + ACK
ACK

KEINIT (Server algorithms)
KEINIT (Client algorithms)
KEXDH_INIT (Diffie-Hellman)
KEXDH_REPLY (Diffie-Hellman, H, Host Key)

End of key exchange

SSHv2

SSH Transport

SSH Authentication

SSH Connection

TCP
SSHv2 Connection establishment

TCP connection established

SYN
SYN + ACK
ACK

TCP connection established

KEXINIT (Server algorithms)
KEXINIT (Client algorithms)
KEXDH_INIT (Diffie-Hellman)

KEXDH_REPLY (Diffie-Hellman, H, Host Key)

End of key exchange

SERVICE_REQUEST (userauth)
SERVICE_ACCEPT

SSHv2

SSH Transport

SSH Authentication

SSH Connection

TCP
SSHv2 Connection establishment

Client

SYN
SYN + ACK
ACK
TCP connection established

Server

KEINIT (Server algorithms)
KEINIT (Client algorithms)
KEHXDH_INIT (Diffie-Hellman)
KEHXDH_REPLY (Diffie-Hellman, H, Host Key)
End of key exchange
SERVICE_REQUEST (userauth)
SERVICE_ACCEPT
USERAUTH_REQUEST(u, pk, sid, sign{u|pk|sid})
USERAUTH_SUCCESS

SSHv2

SSH Transport
SSH Authentication
SSH Connection

TCP
SSHv2 Connection establishment

1. **Client**
   - SYN
   - SYN + ACK
   - ACK
   - TCP connection established

2. **Server**
   - KEXINIT (Server algorithms)
   - KEXINIT (Client algorithms)
   - KEXDH_INIT (Diffie-Hellman)
   - KEXDH_REPLY (Diffie-Hellman, H, Host Key)

3. **End of key exchange**
   - SERVICE_REQUEST (userauth)
   - SERVICE_ACCEPT

4. **Username authentication**
   - USERAUTH_REQUEST (u, pk, sid, sign{u|pk|sid})
   - USERAUTH_SUCCESS

5. **Service request**
   - SERVICE_REQUEST (session)
   - SERVICE_ACCEPT

6. **SSHv2 session established**

**SSHv2**
- SSH Transport
- SSH Authentication
- SSH Connection

**TCP**