SSH, SSL, and IPsec: wtf?

Eric Rescorla
RTFM, Inc.
ekr@rtfm.com
What are we trying to accomplish?

• Alice, Bob want to talk to each other
• But they’re worried about attack
  – How do you know you’re talking to the right person?
  – How do you know people can’t listen to your conversation
  – How do you know people can’t change your conversation?
• We want to build a system that protects against these attacks
Terminology Dump 1: Attacker Capabilities

**Passive** Attacker doesn’t send anything.

**Active** Attacker is allowed to send traffic.

**On-path** Attacker is on the communications path between A and B.
  - Sees all traffic
  - Can seamlessly impersonate either side

**Off-path** Attacker is not on communications path between A and B
  - Can't see traffic between A and B.
  - Can sometimes send traffic as either (subject to address filtering).
Terminology Dump 2: Security Properties

**Confidentiality**  Information being transmitted is kept secret from attackers

**Data Origin Authentication**  Receivers can determine the origin of traffic.

**Message Integrity**  Tampering of traffic can be detected.

**Third-party Verifiability**  A party not involved in the initial communication can verify what happened. (Often misleadingly called *non-repudiation*)
A simple problem: remote authentication

- You’re a Web server
  - X connects to you claiming to be Alice
  - How can you tell?
- Assumptions:
  - All you have is the network traffic
    - Can send messages to X
    - Receive X’s response
  - Attackers can forge but not view, intercept, or modify traffic
  - You have some prior relationship with Alice
Remote authentication: basic ideas

- Alice needs to be able to do something others can’t do
  - Generally, compute some function
    - But why can’t X do that?
- How do we break the symmetry?
  - Give Alice more resources
  - Give Alice some secret
One-sided authentication with shared secrets

• Assume Alice and Bob share a secret $S_{ab}$
  – Alice needs to prove possession of $S_{ab}$
  – (Assume Alice authenticates Bob some other way)

• Simple approach:
  – Bob and Alice both store $S_{ab}$
  – Alice sends Bob $S_{ab}$
  – Bob does memcmp().
Problems with the previous scheme

**Snooping.** an attacker who is on-path can capture the password and *replay* it

**Hijacking.** an attacker can wait for you to exchange the password and then take over the connection

**One-way authentication.** how does Alice authenticate Bob?
Fixing snooping

• Alice doesn’t send $S_{ab}$ over the wire
  – Instead she computes some function $f$
  – And sends $f(S_{ab})$

• What properties does $f$ need?
  1st Preimage Resistant hard to compute $S_{ab}$ from $f(S_{ab})$
  2nd Preimage Resistant hard to find $S'$ st $f(S') = f(S_{ab})$

• Luckily, we have such functions
Cryptographic hash functions

- Basic idea: one-way function (also called message digests)
  - Take an arbitrary length bit string \( m \) and reduce it to 100-200 (\( b \)) bits
  - \( H(m) = h \)
- Hash functions are preimage resistant
  - Takes approximately \( 2^b \) operations to find \( m \) given \( h \)
- Hash functions are collision resistant
  - Takes approximately \( 2^{b/2} \) operations to find \( m, m' \) st. \( H(m) = H(m') \)
- Popular algorithms: MD5, SHA-1, SHA-256
Challenge-Response

• So, Alice just sends $H(S_{ab})$, right?
  – Wrong
  – This becomes the new secret
  – So we still have a replay attack problem

• Bob needs to force Alice to compute a new function each time

\[
\begin{align*}
  Alice & \quad Bob \\
  Challenge & \\
  \leftarrow \quad H(S_{ab} + Challenge) & \rightarrow \\
\end{align*}
\]

• Challenge needs to be unique for every exchange
  – Does not need to be unpredictable
Why mutual authentication?

- We assumed that Alice was talking to Bob
  - But how does Alice know that?
  - She can’t trust the network
  - What if she’s connecting to the attacker

\[ \begin{align*}
  Alice & \quad & \text{Attacker} & \quad & Bob \\
  \text{Challenge} & \quad & \text{Challenge} & \quad & \text{Challenge} \\
  H(S_{ab} + \text{Challenge}) & \quad & H(S_{ab} + \text{Challenge}) & \quad & \text{Attack Commands}
\end{align*} \]

- Alice has just logged in for the attacker
  - He can issue any commands he wants (oops!)
Adding mutual authentication

- We already know how to authenticate Alice
  - Now we need to authenticate Bob
  - Just reverse the procedure

Alice  

\[ \text{Challenge}_1 \]

\[ \rightarrow \]

Bob

\[ \text{Challenge}_2 \]

\[ H(S_{ab} + \text{Challenge}_1 + \text{Challenge}_2) \]

\[ \leftarrow \]

\[ H(S_{ab} + \text{Challenge}_2 + \text{Challenge}_1) \]

- Each side needs to control its own challenges
  - Otherwise we have replay issues again
Hijacking

- This protocol still has a hijacking problem

Alice   Attacker   Bob

\[ H(S_{ab} + \text{Challenge}_1 + \text{Challenge}_1) \]

\[ H(S_{ab} + \text{Challenge}_2 + \text{Challenge}_1) \]

\[ H(S_{ab} + \text{Challenge}_2 + \text{Challenge}_1) \]

\[ H(S_{ab} + \text{Challenge}_1 + \text{Challenge}_1) \]

- We need to authenticate the data
  - Not just the initial handshake
Authenticating data

- Break the data into records
  - Attach a *message authentication code* (MAC) to each record
  - Receiver verifies MACs on record

<table>
<thead>
<tr>
<th>Length</th>
<th>Data</th>
<th>MAC</th>
</tr>
</thead>
</table>
A message authentication code? Dude, wait, what?

- What’s a MAC?
  - A one-way function of the key and some data
  - \( F(k, data) = x \)
    * \( x \) is short (80-200 bits)
    * Hard to compute \( x \) without \( k \)
    * Hard to compute \( data \) even with \( k, x \)

- This sounds kinda like a hash
  - MACs are usually built from hashes
    * World’s simplest MAC: \( H(k + data) \) (this has problems)

- Popular MACs: HMAC
Where does the key come from?

- We want a key that’s unique to this connection
  - And tied to both sides
  - Get it from the challenge-response handshake
- First attempt: \( K = H(S_{ab} + Challenge_1 + Challenge_2) \)
  - But now the key is the same in both directions
  - And the same as the challenge response!
  - Allows reflection attacks
- Second attempt
  - \( K_{a\to b} = H(S_{ab} + ”AB” + Challenge_1 + Challenge_2) \)
  - \( K_{b\to a} = H(S_{ab} + ”BA” + Challenge_1 + Challenge_2) \)
World’s simplest security protocol

\[ \text{Alice} \quad \text{Bob} \]

\[ \text{Challenge}1 \]

\[ \text{Challenge}2 \quad H(S_{ab} + \text{Challenge}1 + \text{Challenge}2) \quad H(S_{ab} + \text{Challenge}2 + \text{Challenge}1) \]

\[ \text{Message}1, \text{MAC} \]

\[ \text{Message}2, \text{MAC} \]

- Each side knows who the other is
- All messages are authenticated
  - But they’re not confidential
  - So don’t send any secret information
Symmetric Encryption

- We have two functions $E, D$ st.
  - $E(k, \text{Plaintext}) = \text{Ciphertext}$
  - $D(k, \text{Ciphertext}) = \text{Plaintext}$
  - These are easy to compute
  - Either function is hard to compute without $k$

- Popular encryption algorithms: DES, 3DES, AES, RC4
A (mostly) complete channel security protocol

Alice  Bob

\[ \text{Challenge}_1 \]

\[ \text{Challenge}_2 \]

\[ H(S_{ab} + \text{Challenge}_1 + \text{Challenge}_2) \]

\[ H(S_{ab} + \text{Challenge}_2 + \text{Challenge}_1) \]

\[ E(k_{a \rightarrow b}, (\text{Message}_1, \text{MAC})) \]

\[ E(k_{b \rightarrow a}, (\text{Message}_2, \text{MAC})) \]

- Each side knows who the other is
- All messages are authenticated
- All messages are confidential
So, we’re done, right?

- How do Alice and Bob get $S_{ab}$?
- Some out of band channel
  - Send a letter—do you trust USPS?
  - Meet in person—airplane tickets are expensive
  - Guys with briefcases handcuffed to their wrists?
- All of these are pretty inconvenient
  - We can do better
Diffie-Hellman Key Agreement

• Each side has two keys ("public" and "private")
  – You publish the public key but the private key is secret
  – $F(K^a_{pub}, K^b_{priv}) = F(K^b_{pub}, K^a_{pub}) = ZZ$
    – You need at least one private key to compute $ZZ$

• This is crypto rocket science—but you don’t need to understand how it works
Using Diffie-Hellman

Alice

\[ \text{Random1, } K^a_{\text{pub}} \]

Bob

\[ \text{Random2, } K^b_{\text{pub}} \]

- Each side sends its public key
- The other side combines its private key with the other side’s public key to compute \( ZZ \)
- The traffic keys are generated from \( ZZ \)
Man-in-the-middle attack

- Each side thinks it’s talking to the other
  - This is what happens when you don’t authenticate
- Alice and Bob need some way to authenticate each other’s public keys
Digital Signatures

• Remember MACs?

• There’s a public key version of this
  – “Sign” with $K_{priv}$
  – “Verify” with $K_{pub}$

• A signed message can only be generated by someone who has the private key

• Popular algorithms: RSA, DSA, ECDSA
Public key distribution

• Public key cryptography is one piece of the puzzle
  – But only one piece

• I can verify a signature came from a given key
  – But where do I get that key from?

• We could have a global directory
  – Obvious scaling problems here

• What if I could give you a credential vouching for your public key?
Certificates

• Digital signatures let us do exactly that
• Create a central certificate authority (CA)
  – Alice proves her identity to the CA
  – The CA gives her a signed message “Alice’s public key is X” (a certificate)
• Anyone can verify this certificate
  – As long as they have the public key of the CA
  – This key is compiled into the software
• Popular CAs: VeriSign, Thawte, GoDaddy
Diffie-Hellman with certificates

Alice     Bob

\[ \text{Random1, Cert}^a \]
\[ \text{Random2, Cert}^b \]
\[ E(k_{a \rightarrow b}, (Message1, MAC)) \]
\[ E(k_{b \rightarrow a}, (Message2, MAC)) \]

- Certificates contain DH public keys
- Each side can authenticate the other
  - This is actually a bug
  - Certificates are too inconvenient for users to get
  - And the user doesn’t always need to be authenticated
  - Or is authenticated some other way
One-way authentication with PKC

- One side (server) has a certificate
- The other side (client) makes up a random key pair

\[
\begin{align*}
\text{Client} & \quad \text{Server} \\
\text{Random1, Cert}^s & \quad \text{Random2, } K^c_{pub} \\
E(k_{c\rightarrow s}, (\text{Credit card #, MAC})) & \quad E(k_{s\rightarrow c}, (\text{OK, MAC}))
\end{align*}
\]

- This authenticates the server but not the client
- We can do a similar trick with RSA
  - Encrypt with public key, decrypt with private key
- This is the main operational mode for SSL/TLS
Perfect Forward Secrecy

- What happens if one side's computer is compromised?
  - Attacker gets private key
  - Can decode all communications by that side
- Fix: have certificates with signature keys (RSA, DSA)
  - Generate a random DH key for each handshake
  - Sign it with your signature key
- Compromise of private key doesn’t affect past traffic
  - But you can MITM future connections
- This is the main operational mode for IPsec
Algorithm negotiation

- There are a lot of choices here
  - Who authenticates,
  - Public key algorithm
  - Digest algorithm
  - Encryption algorithm
- Each make sense in some scenarios
  - A good protocol is adaptable
- This means some kind of negotiation
  - This needs to be protected to prevent downgrade attacks
A complete channel security protocol

\[\text{Alice} \quad \text{Bob}\]

\[\text{Random1,Algorithms} \quad \text{Random2,Algorithm,Cert}^b\]

\[\text{Cert}^a,\text{MAC}(ZZ,\text{HandshakeMsgs}) \quad \text{MAC}(ZZ,\text{HandshakeMsgs})\]

\[E(k_{a\rightarrow b},(Message_1,MAC)) \quad E(k_{b\rightarrow a},(Message_2,MAC))\]
Secure Sockets Layer (SSL)

- Originally a Netscape proprietary protocol
- Target application: e-commerce
  - What people thought the Web was for in 1994
  - Objective: send my credit card to Amazon securely
- Basic principles (ca. 1994)
  - The server is authenticated (via certificate)
  - The client is unauthenticated
  - This should be easy to plug in to both sides
SSL/TLS History (1)

- SSLv1 (never released)
  - Designed by Kipp Hickman
  - Severe security flaws (immediately obvious to anyone who knew crypto)
- SSLv2
  - Hickman again (after being beaten up by others)
  - Modest security flaws (truncation attacks, downgrade)
  - Very widely deployed
- SSLv3
  - Freier, Karlton, Kocher
  - Fixes the above problems
SSL/TLS History (2)

- Transport Layer Security (TLS) 1.0 (RFC 2246)
  - First standardized version of SSL
  - Modest improvements to key derivation
- TLS 1.1 (RFC 4346)
  - Fixes for modest security flaws
- TLS 1.2 (RFC 5246)
  - Flexibility for hash functions (thanks Dr. Wang!)
- As you can see, this is in maintenance mode
HTTP over SSL (HTTPS)

- The client *knows* that the server expects HTTPS
  - It’s in the URL `https://www.example.com/`
  - It’s on a separate port
- The server’s certificate has its domain name (`www.example.com`)
SSL Session Resumption

- Asymmetric (private key) operations are expensive
  - And HTTPS tends to involve a lot of SSL/TCP connections

- Caching pays off here
  - Each handshake establishes a *session*
  - Clients can *resume* the session with the same keying material
  - Thus skipping the key exchange
Upward Negotiation

• What if the client and server don’t know each other’s capabilities
  – Would be nice to discover them
  – And automatically upgrade to TLS

• Example: SMTP

\[ \text{Client} \quad \text{Server} \]

\[ \text{HELO + TLS} \]

\[ \text{OK do TLS} \]

\[ \text{SSL Handshake} \]

\[ \text{SMTP transaction} \]

• Of course, this allows downgrade attacks
DoS Attacks on SSL/TLS

- Resource consumption
  - Public key operations are expensive
    * Client can force the server to do a lot of them
    * But not blindly (TCP handshake)
  - State on the server side

- SSL/TLS connection runs over TCP
  - TCP connections are easy to DoS
  - SSL/TLS can’t protect you from this
  - Needs to be at a lower layer
Datagram TLS (RFC 4347)

- TLS requires a reliable channel
  - The handshake is in sequence
  - The data records depend on each other
  - In practice this means TCP

- What about unreliable channels?
  - DTLS is a slight modification of TLS
  - Reliability for the handshake
  - Record independence

- More DoS resistance (more on this later)
Secure Shell (SSH)

- Originally designed by Tatu Ylonen
  - Replacement for rsh
  - Now the standard tool for secure remote login
  - A lot of authentication mechanisms

- Other features
  - Remote X
  - File transfer
  - Port forwarding

- Original version was seriously broken
  - Later standardized versions are better
  - Transport protocol looks a lot like TLS
SSH leap of faith authentication

• No certificates—server just has a raw public key
  – The server provides the key when the client connects
  – The client stores the server’s key on first connection
  – Any changes in the key are an error

• The key can be authenticated out of band
  – The server operator tells the client the key fingerprint (hash) over the phone
  – But only the most paranoid people do this

• This was considered insanity at the time
  – Now it’s considered clever
SSL Key Exchange Protocol

Client

Protocol=SSH−2.0...

Server

Protocol=SSL−2.0...

KeyExInit(algorithms...)

KeyExInit(algorithms....)

DH(group size)

p,g

DH_{pub}^{c}

DH_{pub}^{s}, Sign(K_{priv}^{s}, DH_{pub}^{s})
SSH Client Authentication

- Server is authenticated first
- Client is then authenticated
  - Raw password
  - Challenge-response
  - Public key
  - GSS-API
  - Kerberos
- Mechanisms are negotiated
SSL Client Authentication Protocol

Client

Auth: None

Auth: publickey,password,...

publickey=XXX

No

publickey=YYYY

No

signature

OK

Server
Port Forwarding

- SSH provides a port forwarding feature
- Example: X11 remote

![Diagram of SSH port forwarding](image)

- SSH server does `setenv DISPLAY localhost:XXXX`
- Apps just automatically work
Secure Remote Shell

- SSH is backward compatible with rsh
  - So other applications can be securely remoted
  - Even without port forwarding
- Examples
  - CVS
  - rsync
  - dump/restore
- Apps don’t need security, just remote access
IPsec: IP Security

- Basic idea: secure IP datagrams
  - Instead of at application layer like TLS or SSH
- Why was this considered a good idea?
  - Secure all traffic, not just TCP/UDP
  - Automatically secure applications
    * Without any change to the application
  - Built-in-firewalling/access control
IPsec history

- Work started in 1992-1993
- General agreement on packet formats early on
  - Though confusion about integrity vs. authentication
- Key agreement was very controversial
  - Design issues
  - IPR issues
- First “proposed standards” published in 1998
  - Mishmash of IKE, ISAKMP, OAKLEY
- Complaints about clarity and complexity
  - IKEv2 approved in 2005
IPsec architecture
### IPsec Packet Formats

**Transport Mode**

<table>
<thead>
<tr>
<th>IP Hdr</th>
<th>IPsec Hdr</th>
<th>TCP Hdr</th>
<th>Data</th>
<th>Transport Mode</th>
</tr>
</thead>
</table>

**Tunnel Mode**

<table>
<thead>
<tr>
<th>IP Hdr</th>
<th>IPsec Hdr</th>
<th>IP Hdr</th>
<th>TCP Hdr</th>
<th>Data</th>
<th>Tunnel Mode</th>
</tr>
</thead>
</table>
IKE “Anonymity”

- The handshakes we’ve seen leak your identity to passive attackers
  - Arguably this is bad
  - IKE tries to stop this

![IKE Handshake Diagram]

- An active attacker can get the initiator’s identity
IKE DoS prevention

- Objective: prevent blind DoS attacks

Diagram:

**Initiator**

\[ DH^i_{pub} \]

\[ Ticket \]

\[ DH^i_{pub}, Ticket \]

\[ DH^r_{pub} \]

\[ \{ CERT^i \} \]

**Responder**

\[ \{ CERT^r \} \]

- Ticket has to be stateless
IPsec Status

- Many implementations
  - Windows, OS/X, Linux, FreeBSD, IOS...
- Nearly all deployments are in VPN settings
- And people are cutting over to SSL/VPN
  - Semi-manual configuration
- This is not what was intended
- Widely regarded as a semi-failure
What was wrong with IPsec?

- Complexity
- Time to market
- Wrong design goals
- Hard to use
Final thoughts

• All of these protocols look strikingly alike
  – To some extent they were designed by the same people
  – But also there appear to only be so many ways to do this

• All have gone through multiple revisions
  – This is really hard to get right
  – Even when you have experienced people
  – Don’t invent your own

• Usage models matter
  – SSL/TLS and SSH got this right
  – IPsec did not