Lightweight Authentication and Key Exchange

John Viega
viega@securesoftware.com
• Drive discussion and understanding about requirements
• The crypto world has produced good solutions that lead to operational nightmares (SSL)
• Lots of off-the-shelf solutions
• Solutions tend not to map well to implicit requirements
Authentication

- **Entity authentication**
  - Who’s on the other side
  - Connections themselves are assumed virtual
  - All messages must be authenticated as coming from a set of entities
  - Non-repudiation usually isn’t a goal and is expensive

- **A goal for both parties: message integrity**

- **Another goal: Temporal consistency**
  - Attackers shouldn’t replay messages
  - Missing messages should be detectable

- **Another goal: connection confidentiality**

- All can be provided with layered services
What can go wrong?

• One entity can pretend to be another
  - False login
  - Connect to a fake server
  - "Man-in-the-middle": attacker as relay

• Single-entity authentication is rarely enough
  - Only when no notion of access control

• Spectacular failures result
  - Do you click the lock on your browser?
  - Would my mom know what to look for if she did?
  - This is true even in non-web applications

• Password authentication is notably suspect
  - Particularly, dictionary attacks
Authentication requires secrets

**Efficient** communication needs shared secrets
- Though not necessarily long-term

Key management is...
- Necessary
- A source of tremendous risk

Should server admins have user passwords?
Should low-entropy passwords persist?
Should we lock out possible attackers?

If insecure channels are necessary, only for account setup
Key Exchange

• With a shared secret, who needs it?
• There’s already a virtual “established connection”

• Might not want to save state
  - Managing sequential nonces is a pain
• Avoid exposing our “good” secrets
  - Many messages encrypted under same key
  - Good design: single key for single purpose
• Forward secrecy: damage control
  - Compromise of some secrets won’t compromise all
Usability should be priority #1

- A hard balance to strike
- Defense-in-depth theoretically helps...
- Physical solutions are slow to adopt
  - Cost
  - Operational problems (newest I’ve heard: germs)
- Passwords are “usable”...
- … but not when they’re secure!
- Best bet?
  - A range of solutions to meet various needs
  - Defaults should be a good compromise
  - We’ll revisit later
Efficiency

- Public key crypto is expensive
- ECC may not help enough for small devices
- AKE takes significant time on a CryptoPhone
- More an issue on server side

- Terse protocols with minimal messages?
Security Assurance

- **Traditional approach: lack of attacks**
  - Assurance requires extensive review

- **Model checking: prove resistance to attacks**
  - Can only do this for known attacks
  - Large state spaces can require approximations
  - In practice, all checkers have limitations

- **Provable security: prove secure**
  - In the sense of an attack implying an attack on a vetted algorithm (e.g., AES, RSA, Diffie-Hellman)
  - Requires concrete security models and *some* review
  - E.g., Bellare-Rogaway: all network-only attacks
Interoperability

- **802.1X (EAP)**
  - Bad bindings abound
  - Usually assumes trusted (physical) path

- **Radius**
  - Central management
  - Hard to do securely

- **Kerberos**
  - Central management
  - Widely supported, rarely deployed

- **IKE: Internet Key Exchange**

- Supporting existing infrastructure compelling
- Otherwise, why?
Other Requirements

- Multi-party problem
- Protection against bad random numbers
- Support for password resets / changes
- Server compromise forbids spoofing?

- In general, assume worst feasible threat model
- Should $10/hr tech support be able to reset a password?
- People should be leery of bringing a password to someone else’s machine
• Look at classes of solutions
• Plus some commentary
• I might be wrong, based on assumptions
• Mostly, I’ve tried to leave it open
• Assumptions:
  - Mutual authentication
  - Usability is a priority
  - Key exchange needs to happen
  - Both parties should contribute random data
• Ignoring (for now):
  - Multi-party problem
  - Key servers
Symmetric Protocols

- crypt, MD5-MCF, S/KEY, HTTP Digest Auth, ...
  - None provide mutual authentication
  - All require existing client-trusted (secure) channel
- Not much, but easy, given requirements
- Forward secrecy requires synchronization
  - But, easy to do
- Password-based protocols are susceptible to dictionary attacks
- Two messages possible using a nonce
  - $A \rightarrow \text{GCM}_k(N, X, B) \rightarrow B \rightarrow \text{GCM}_k(N+1, Y, A) \rightarrow A$
  - $S = X \oplus Y$
- Otherwise, three messages
Public Key Protocols

- We’ll skip the math
- Forward secrecy easier (use ephemeral keys)
- Implementation more complex and slower
- Provably secure protocols, such as modified “Station to Station” (StS).
- Relying on even ad-hoc PKI seems unrealistic
- Password-based possible
- Simple modification to modified StS
- Also, EKE family of protocols
Initial Thoughts

- Authentication alone shouldn’t be enough
  - Secure channel needs to result
  - Bindings for SecurID would need some work
- Shared secrets and passwords
- Allow devices to cache credentials
  - Encourage more efficient transfers
  - Discourage day-to-day passwords
- Support one-time setup for passwords
- Bindings for one-time passwords?
- Provide guidelines for deployment
  - Password expiration recommendations
- Forward secrecy, etc.
Questions?

viega@securesoftware.com