CS 645 : Lecture 6
Hashes, HMAC, and Authentication

Rachel Greenstadt
May 16, 2012
Reminders

• Graded project 2, midterm, available on bbvista

• Project 3 out (crypto)

• Next two weeks entirely online. I will send out links to lectures.

• Short simple quiz on lectures will be on bbvista week of June 4 (folded into participation grade)
Authenticating Messages

- Hash Functions
- MAC
- HMAC
- SSL Man-in-the-middle / Project 4
Introduction to Hash Functions

- If $H$ is a hash function, $m$ is an input bit string, and $h$ is the output of $H$ applied to the input $m$, then we write $h = H(m)$. Some common and useful terminology:

- If $h = H(m)$ then
  - $h$ is called the "hash" of $m$,
  - $m$ is called a "preimage" of $h$,

- for a given input $m$, a "second preimage" of $m$ is a different input $m'$ such that $H(m) = H(m')$,

- if $m$ and $m'$ are different inputs such that $H(m) = H(m')$ then the pair $\{m, m'\}$ is called a "collision" for $H$. 

Wednesday, May 16, 2012
Collision Resistance

• Strong collision resistance
  • Hard to find *any* x,y such that h(x)=h(y)

• Weak collision resistance / preimage attacks
  • First preimage attack: given hash h₁, find m such that h(m) = h₁
  • Second preimage attack : Given message m₁ find message m₂ such that h(m₁) = h(m₂)
Exercise

• Explain the difference between breaking strong collision resistance and a second pre-image attack?

• Quantify this difference in how long it would take to brute-force the attack for a perfect hash function of n bits.
• First case
  • $H(\text{"hello world"}) = x$ find $y$ such that $H(y) = x$

• Second case
  • $m$ and $m'$ s.t. $H(m) = H(m')$
Birthday Attacks on Collision Resistance

- Given function $h$, goal is find two inputs $x, y$ such that $h(x) = h(y)$

- Based on the birthday paradox: A group of 23 or more people will have the same probability > 50%

- $H$ different outputs, then expected $1.25 \times \sqrt{H}$ to find a match

- $2^{160}$ outputs for SHA-1, leads to approx $2^{80}$ tries
Types of Hash Functions

- MD5
  - 128-bit output
  - Designed by Ron Rivest, used very widely
  - Collision-resistance broken (summer of 2004 and it keeps getting worse)
- RIPEMD-160
  - 160-bit variant of MD5
- SHA-1 (Secure Hash Algorithm)
  - 160-bit output
  - US government (NIST) standard as of 1993-95
    - Also the hash algorithm for Digital Signature Standard (DSS)
Group exercise

- Hash functions are reasonably fast, but here's a much faster function to compute. Take your message, divide it into 128-bit chunks, and xor all the chunks together to get a 128-bit result. Do the standard hash function on the result. Is this a good hash function? Why or why not?
SHA-1

160-bit buffer (5 registers) initialized with magic values

Split message into 512-bit blocks

Padding (1 to 512 bits)

Message length ($K \mod 2^{64}$)

Against padding attacks

$L \times 512$ bits = $N \times 32$ bits

Compression function

- Applied to each 512-bit block and current 160-bit buffer
- This is the heart of SHA-1
SHA-1 Compression Function

- Current message block
- Current buffer (five 32-bit registers A, B, C, D, E)
- Four rounds, 20 steps in each
- Let’s look at each step in more detail...
- Very similar to a block cipher, with message itself used as the key for each round
- Fifth round adds the original buffer to the result of 4 rounds
- Note: addition (+) is mod 2^32
- Buffer contains final hash value

Wednesday, May 16, 2012
How Strong is SHA-1?

• Every bit of output depends on every bit of input
  • Very important for collision resistance

• Brute-force inversion requires $2^{160}$ ops, birthday attack on collision resistance requires $2^{80}$

• Recent weaknesses (2005)
  • Collisions can be found in $2^{63}$ ops
Authentication without Encryption

Integrity and Authentication: only someone who knows key can compute MAC for a given message
How to hash the key and message?

• Seems easy, just compute h(key|message)

• Problems?

  • Assume h is SHA-1

  • Recall that in SHA-1, the message is hashed from left to right in 512 bit chunks
Enter Carol

- Bob is Carol’s boss, and Alice is Bob’s boss
- Carol appends “P.S. Give Carol a promotion and triple her salary” to Alice’s message to Bob

Carol can take the original message, add some padding, then add her postscript and pass it into SHA-1
HMAC

- MAC that is “as secure as underlying hash”
- Strong collision resistance
- Attacker that doesn’t know key K cannot compute digest(K,x) for data x even if the attacker can see digest(K,y) for arbitrary y not equal to x
- Result slow but provable
HMAC

- Construct MAC by applying cryptographic hash function to message and key
  - Could also use encryption instead of hashing, but…
  - Hashing is faster than encryption in software
  - Library code for hash functions widely available
  - Can easily replace one hash function with another
  - There used to be US export restrictions on encryption
- Invented by Bellare, Canetti, and Krawczyk (1996)
  - HMAC strength established by cryptographic analysis
  - Mandatory for IP security, also used in SSL/TLS
How HMAC Works

- If key > 512 bits, digest(key) and pad to 512 else if key < 512 bits, pad to 512
- result1 = digest ((Const1 XOR padded key) . message)
- result2 = digest((Const2 XOR padded key) . result1)
- HMAC(message, key) = result2
Combine encryption and MAC for confidentiality and integrity
Encrypt-and-MAC

Natural approach for authenticated encryption: Combine an encryption scheme and a MAC.

$E_{Ke,Km}$
- $M$ as input
- $C'$ and $T$ as output
- $K_e$ for encryption
- $K_m$ for MAC

$D_{Ke,Km}$
- $M$ as input
- $C'$ and $T$ as output
- $K_e$ for decryption
- $K_m$ for verification

Return $M$ if valid/invalid
Insecure!

Assume Alice sends messages:

If $T_i = T_j$ then $M_i = M_j$

Adversary learns whether two plaintexts are equal.

Especially problematic when $M_1, M_2, ...$ take on only a small number of possible values.
Attacks

• Confidentiality considers indistinguishability under...
  • Chosen Plaintext Attack (CPA) An attacker can obtain the ciphertext for any provided plaintext (but does not have the key).
  • Chosen Ciphertext Attack (CCA) An attacker can obtain the plaintext for any provided ciphertext (but does not have the key).

• Integrity
  • PTXT - Integrity of Plaintext - computationally infeasible to produce a ciphertext decrypting to a message that the sender had never encrypted.
  • CTXT - Integrity of Ciphertext To be computationally infeasible to produce a ciphertext not previously produced by the sender.
Results of [BN00,Kra01]

<table>
<thead>
<tr>
<th>Method</th>
<th>Privacy</th>
<th>Integrity</th>
</tr>
</thead>
<tbody>
<tr>
<td>Encrypt-then-MAC</td>
<td>Strong (CCA)</td>
<td>Strong (CTXT)</td>
</tr>
<tr>
<td>MAC-then-Encrypt</td>
<td>Weak (CPA)</td>
<td>Weak (PTXT)</td>
</tr>
<tr>
<td>Encrypt-and-MAC</td>
<td>Insecure</td>
<td>Weak (PTXT)</td>
</tr>
</tbody>
</table>
Authenticating Users

“Something you forget, something you lose, and something you used to be”

• Passwords
• Alternatives
• Multi-factor Authentication
Password Security Review

- Summarize system
  - Identify assets: What do you wish to protect
  - Identify adversaries and threats
  - Identify vulnerabilities
  - Calculate the risks
  - Evaluate controls/mitigation strategies
- Iterate
Assets
Adversaries
Vulnerabilities
Vulnerabilities

• Online guessing/dictionary attack
• Offline guessing/dictionary attack
• Shared passwords
• Password fallback schemes
Risks
Mitigation Strategies
Mitigation Strategies

- Salts
- Encrypted Storage
- Challenge/Response
Alternatives to Passwords

- Graphical passwords, phrases
- Tokens/dongles
- Biometrics
Multifactor Authentication
Public Key Authentication
Authenticity of Public Keys

**Problem**: How does Alice know that the public key she received is really Bob’s public key?
Distribution of Public Keys

- Public announcement or public directory
  - Risks: forgery and tampering
- Public-key certificate
  - Signed statement specifying the key and identity
    - $\text{sig}_{\text{Alice}}(\text{"Bob"}, \text{PK}_B)$
- Common approach: certificate authority (CA)
  - Single agency responsible for certifying public keys
  - After generating a private/public key pair, user proves his identity and knowledge of the private key to obtain CA’s certificate for the public key (offline)
  - Every computer is pre-configured with CA’s public key
Hierarchical Approach

◆ Single CA certifying every public key is impractical
◆ Instead, use a trusted root authority
  • For example, Verisign
  • Everybody must know the public key for verifying root authority’s signatures
◆ Root authority signs certificates for lower-level authorities, lower-level authorities sign certificates for individual networks, and so on
  • Instead of a single certificate, use a certificate chain
    – sig_{Verisign} (“UW”, PK_{UW}), sig_{UW} (“Alice”, PK_{A})
  • What happens if root authority is ever compromised?
X509 Certificates

- Version
- Certificate Serial Number
- Signature algorithm identifier
- Signature
- Issuer Name
- Subject Name
- Subject's public key info
- Issuer Unique Identifier
- Subject Unique Identifier
- Extensions

Added in X.509 versions 2 and 3 to address usability and security problems
Bad Certificates

- What to do if a bad certificate is issued?
- In practice...wait for it to expire
- In theory
  - Revocation Services
  - Revocation Lists
Programming Project 4

- Out May 12
- Due May 26 11:59pm
- Teams of up to three people
  - New teams OK (old teams also OK)
- Basic idea: Implement a “Man-in-the-Middle” attack against SSL
- Recall Security and Privacy Ethics
- Based on Dan Boneh’s CS255 project (Stanford)/Yoshi Kohno’s CSE 484 project (U Washington)
Overview

• Implement a simple Man In The Middle (MITM) attack on SSL
• Use Java’s networking, SSL and Certificate implementations
  – No need for low level packet manipulation
• Also implement a password based authentication system for the MITM server
  – Allows hacker to issue commands to server
Overview

- Normal SSL
  - SSL encrypted data routed like normal TCP/IP data over the internet
Proxy Server

- Browser connects to proxy
- Proxy connects to web server and forwards between the two
Man in the Middle

• Instead of forwarding encrypted data between the two hosts, our proxy will set up two DIFFERENT SSL connections between the two.
  • Proxy<->Remote Server
    – Sets up a normal SSL client connection to requested remote site
  • Proxy<->Browser
    – Sets up a SSL server connection to the browser, using its own certificate, generated as a copy of the remote host’s cert
• If the browser accepts this fake cert, the proxy has access to the data in the clear!
Security Features

• Secure connection between admin client and proxy server using SSL
• Password based authentication
  • Secure storage
  • Passwords stored hashed using public and private salts
Proxy Server

• Already listens for the browser CONNECT request and sets up the needed SSL connections
• You need to
  – Understand the connections being made
  – Obtain the remote server cert from the remote SSL conn
  – Copy the relevant fields and sign the forged cert using your CA cert (from your keystore) (use IAIK)
  – Modify the code creating the client SSL conn to use the newly forged cert
Signing Certificate

• Build a self signed cert for the proxy server using keytool
  – keytool -genkey -keyalg RSA
  – Store this in a JKS keystore for use by your proxy server
  – Use it for signing your programmatically generated certs
  – You pretend to be a CA e.g. Verisign

• Submit a keystore with your project
Generating Certs “On the Fly”

• Not easy to generate certs programmatically using standard Java libs
• Use the IAIK-JCE library
  – iaik.x509.X509Certificate
iaik.x509.X509Certificate

• To convert from a java cert:
  – new X509Certificate(javaCert.getEncoded());

• Signing
  – cert.sign(
    AlgorithmID.sha256withRSAEncryption,
    issuerPk);

• See iaik.asn1.structures.Name
  – For extracting info (e.g. common name) from
    the cert’s DN (cert.getSubjectDN())
Managing Certs and SSL Sockets

• Use the KeyStore class for
  – Loading certs from file (e.g. your CA cert)
  – Storing programmatically generated certs
• Use SSLContext class for setting up certs to be used with an SSLServerSocket
  – Create a cert
  – Load into new KeyStore
  – Init a KeyFactoryManager with new KeyStore
  – Init SSLContext with new KeyFactoryManager and provided “TrustEveryone” TrustManager
• Use SSLContext for creating SSLSocketFactories
Admin Server

- Already listens for client connections and parses the data sent, using plain sockets
- You need to
  - Modify the code to use SSL sockets (see the proxy server code for examples)
  - Implement authentication for the transmitted username and password
  - Implement the required admin commands
    - Shutdown – the proxy server to stops accepting connections and exit
    - Stats – the proxy server returns a summary of the number of connections it has processed. Add code to record these
Password Authentication

- Proxy server listens for SSL connections from admin client too
- On connection client transmits a username and password
- Server verifies these from its local password file, and executes command if the client is authenticated
Password File

• Need to store a file containing usernames, salts, and hashed passwords

• Use BOTH public and secret salts (AKA pepper)

• Should be stored encrypted/MACed

<table>
<thead>
<tr>
<th>Username</th>
<th>Salt</th>
<th>Password</th>
</tr>
</thead>
<tbody>
<tr>
<td>ibaker</td>
<td>S</td>
<td>H(Pwd</td>
</tr>
<tr>
<td>singuva</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>dabo</td>
<td></td>
<td></td>
</tr>
<tr>
<td>...</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Password File Utility

- You need to add a utility for creating these password files
- Simple method:
  - Make a class to take a file with a list of usernames and passwords and convert it to a password file
Configuring Mozilla
Getting the Proxy to Work

• Try it out on (non-sensitive) SSL sites

• You should see one warning, be able to click past it (continue) and go to site

• (Not Firefox 3)

• Click View Certificate (or see details) to see your cert and verify that its fields are correct
Possible Problems

• You should be able to start up the proxy server and connect to it “out of the box”
• If you are having problems
  – Is someone else using the port? (default 8001)
    • Try a different port on the command line
  – Firewall problems?
    • Try opening the needed ports 8001/8002 (or whatever)
  – Try running your browser on the same machine and setting the proxy as localhost
  – We can’t debug your local network setup
Grading

• Security comes first
  – Design choices
  – Correctness of the implementation
• Did you implement all required parts?
• Secondary
  – Cosmetics
  – Coding style
  – Efficiency
Submitting
(Email a tarball)

- README file
- Names
- Describe your design choices
- How to run your system (e.g. create passwords)
- Answer to discussion question
- Your sources
- A sample of data recorded from your proxy