Security, part 1

15-440

```
quine = 'quine = %r\r\nprint quine %% quine'

print quine % quine
```

A quine is a self-reproducing program.

What does this have to do with security?

It lets you write a self-propagating trojan horse compiler

Ken Thompson's compiler hack from "Reflections on Trusting Trust."

- Modified C compiler does two things:
  - If compiling a compiler, inserts the self-replicating code into the executable of the new compiler.
  - If compiling `login`, inserts code to allow a backdoor password
- After recompiling and installing old C compiler:
  - Source code for Trojan horse does not appear anywhere in login or C compiler
  - Only method of finding Trojan is analyzing binary
    - So, of course, you make the compiler also trojan the debugger, and so on, and so on...

slide derived from original by Nick Feamster

Lesson and a warning

- Attacks can come from anywhere...
  - Trojaned compilers, software;
  - Outside attack over the network;
  - USB keys left outside your office with malware;
  - Insider jobs at banks,
  - ...
- Remember today and tomorrow: The adversary will attack the weakest point of the system. If you use weak crypto, they'll break the crypto. Use strong crypto? Attack the crypto protocol. Good protocol? Attack the implementation... etc., etc.
- No silver bullet: *security is really hard* and requires enormous attention to detail at all levels.

What do we want to secure?

(What's our model for the good & bad guys and the environment?)

A wants to talk to B...

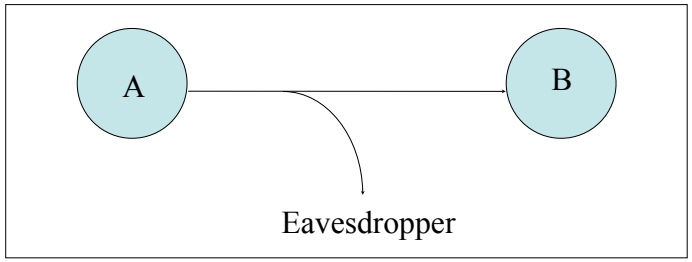
Basic Components

- **Confidentiality** is the concealment of information or resources
- **Authenticity** is the identification and assurance of the origin of information
- **Integrity** refers to the trustworthiness of data or resources in terms of preventing improper and unauthorized changes
- **Availability** refers to the ability to use the information or resource desired

slide derived from original by Nick Feamster

Example: Eavesdropping - Message Interception (Attack on Confidentiality)

- Unauthorized access to information
- Packet sniffers and wiretappers
- Illicit copying of files and programs



slide derived from original by Nick Feamster

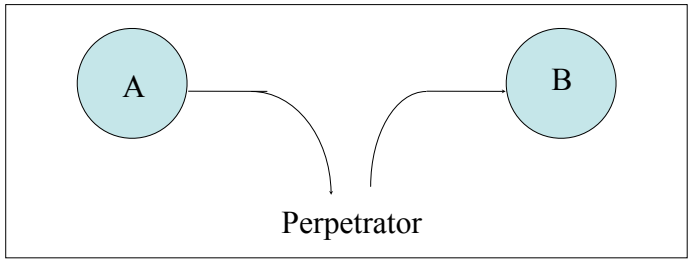
Eavesdropping Attack: Example

- tcpdump with promiscuous network interface
  - On a switched network, what can you see?
- What might the following traffic types reveal about communications?
  - Full IP packets with unencrypted data
  - Full IP packets with encrypted payloads
  - Just DNS lookups (and replies)

slide derived from original by Nick Feamster

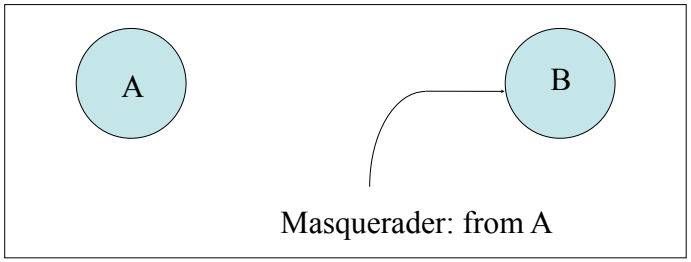
Integrity Attack - Tampering

- Stop the flow of the message
- Delay and optionally modify the message
- Release the message again



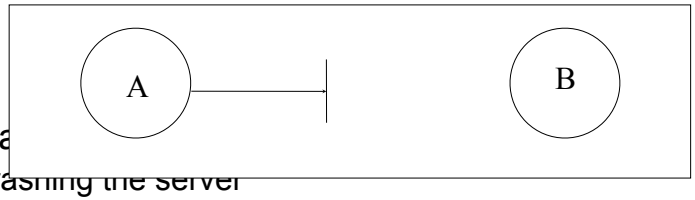
Authenticity Attack - Fabrication

- Unauthorized assumption of other's identity
- Generate and distribute objects under this identity



Attack on Availability

- Destroy hardware (cutting fiber) or software
- Modify software in a subtle way (alias commands)
- Corrupt packets in transit



- Blata
  - Crashing the server

Impact of Attacks

- Theft of confidential information
- Unauthorized use of
  - Network bandwidth
  - Computing resource
- Spread of false information
- Disruption of legitimate services

All attacks can be related and are dangerous!

Security Policy and Mechanism

- **Policy:** a statement of what is, and is not allowed
- **Mechanism:** a procedure, tool, or method of enforcing a policy

- Security **mechanisms** implement functions that help *prevent, detect, and respond to recovery from* security attacks
- Security functions are typically made available to users as a set of *security services* through APIs or integrated interfaces
- Cryptography underlies many security mechanisms.

Security Services

- **Confidentiality:** protection of any information from being exposed to unintended entities
  - Information content
  - Parties involved
  - Where they are, how they communicate, how often, etc.

Security Services - Cont'd

- **Authentication:** assurance that an entity of concern or the origin of a communication is authentic - it's what it claims to be or from
- **Integrity:** assurance that the information has not been tampered with
- **Non-repudiation:** offer of evidence that a party indeed is the sender or a receiver of certain information

Security Services - Cont'd

- **Access control:** facilities to determine and enforce who is allowed access to what resources, hosts, software, network connections
- **Monitor & response:** facilities for monitoring security attacks, generating indications, surviving (tolerating) and recovering from attacks

Example: Web access

- Alice wants to connect to her bank to transfer some money...
- Alice wants to know ...
  - that she's really connected to her bank. **Authentication**
  - That nobody can observe her financial data **Confidentiality**
  - That nobody can modify her request **Integrity**
  - That nobody can steal her money! **(A mix)**
- The bank wants to know ...
  - That Alice is really Alice (or is authorized to act for Alice)
  - The same privacy things that Alice wants so they don't get sued or fined by the government.

Q1: How does Alice know her bank is her bank? Authentication

If Alice accepts K, Mallory can snoop and modify all traffic!

download code at: http://www.cs.cmu.edu/~perspectives/

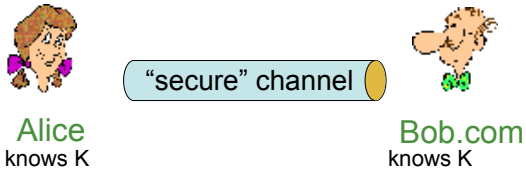
Well...

- What tools do we have at hand?
  - Hashing (e.g., SHA-1)
  - Secret-key cryptography, aka symmetric key.
    - e.g., AES
  - Public-key cryptography
    - e.g., RSA

download code at: http://www.cs.cmu.edu/~perspectives/

Secret Key Cryptography

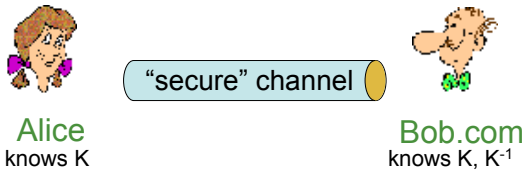
- Given a key *k* and a message *m*
  - Two functions: Encryption (E), decryption (D)
  - ciphertext *c = E(k, m)*
  - plaintext m = D(k, c)
  - Both use the *same* key *k*.



But... how does that help with authentication? They both have to know a pre-shared key K before they start!

Public Key Cryptography

- Given a key *k* and a message *m*
  - Two functions: Encryption (E), decryption (D)
  - ciphertext $c = E(k, m)$
  - plaintext $m = D(k^{-1}, c)$
  - Encryption and decryption use *different* keys!



But how does Alice know that K means "Bob"?

Signatures

- Assume Alice *does* know that Bob's key is *K*...
  - Let's build a more powerful primitive: A digital signature
  - s = signature(K, M)
    - *s* is ideally small, while M might be huge
    - Only the holder of key K can create s
      - In other words, K is proving that it "said" M
- Using secret key crypto, pre-shared key K:
  - HMAC(K, m) ("Hash-based Message Authentication Code")
    - H((K xor opad) | H((K xor ipad) | m))
    - Where "opad" and "ipad" are globally known constants that just mix the bits up
- Why so complex? Why not just...
  - H(key | message) for example?
    - Concatenation attack! Many hash functions can be iterated...
    - H(m1 | m2) = f(H(m1), m2)
    - So if you sent me a MAC for "hi!" I could turn it into "hi! I want to drop the class"
  - H(message, key) is better, but suffers some weaknesses for collision resistance.

Uses of HMAC

- Drawback to previous: Had to have a pre-shared key.
  - HMAC is used all over the place; hugely useful! (Don't implement it yourself, lots of libraries).
- A common use:
  - I create a message
  - I give it to you
  - You give it back to me later
  - I want to verify that it's what I originally gave you
- Why would I want to do this?

Web page authentication

- Low-security example:
  - User logs into the NY Times website using username + password.
    - That login is protected using SSL
    - SSL is expensive! 10-100x more CPU to use SSL than unencrypted HTTP
  - Want to let them return and browse articles without logging in and without SSL
    - (But only browse articles - low security requirement)
- How can we accomplish this?
  - Cookies!