

Introduction to Security and User Authentication

Brad Karp
UCL Computer Science



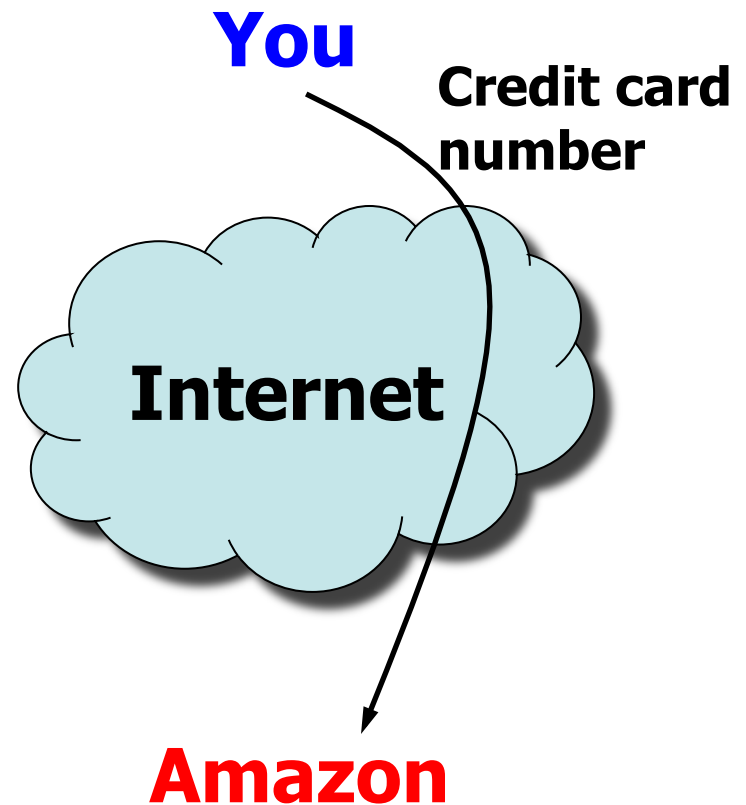
CS GZ03 / M030
10th November 2014

Topics We'll Cover

- User login authentication (local and remote)
- Cryptographic primitives, how to use them, and how **not** to use them
- Kerberos distributed authentication system
- Secure Sockets Layer (SSL)/Transport Layer Security (TLS) authentication and encryption system
- TAOS: logic for reasoning formally about authentication
- Software vulnerabilities and exploits
- Exploit Defenses
- Software Fault Isolation (SFI): containing untrusted code
- OKWS: a least-privilege isolated web server for UNIX

A Simple Example

- Suppose you place an order with Amazon
- Goals:
 - You get the item you ordered
 - Amazon gets payment in the amount you agreed to pay on the payment page

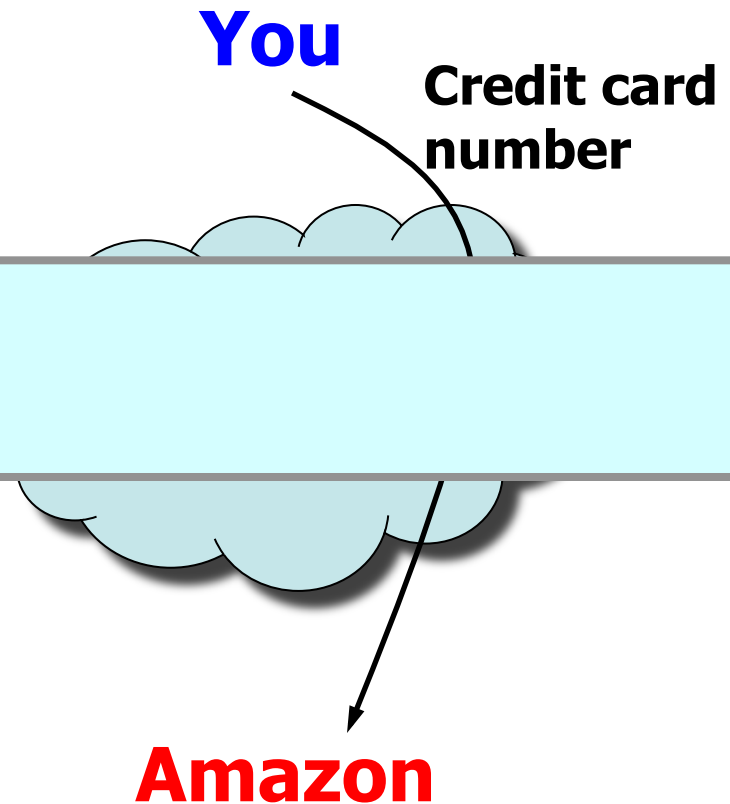


A Simple Example

- Suppose you place an order with Amazon

**How might this go wrong?
Let us count the ways...**

- you ordered
- Amazon gets payment in the amount you agreed to pay on the payment page



Worries for Amazon Order

- What if an **eavesdropper taps Internet link?**
 - Network cables usually not physically secure
- What if someone has **broken into Internet router?** (They're just computers...)
- How do you know you're communicating with **Amazon?**
- How does Amazon know you are **authorized** to use the credit card number you provide?
- What if a **dishonest Amazon employee** learns my credit card number?
- What if Amazon sends me **wrong book, in error**

Worries for Amazon Order (2)

- What if someone has broken into my desktop PC? Or my file server?
- Where did my web browser come from? How about my OS?
- What if my display or keyboard radiates a signal that can be detected at some distance?

Worries for Amazon Order (2)

- What if someone has broken into my

Fundamental security question:

“Whom or what am I trusting?”

Weakest item on list of answers determines system security!

- what if my display or keyboard radiates a signal that can be detected at some distance?

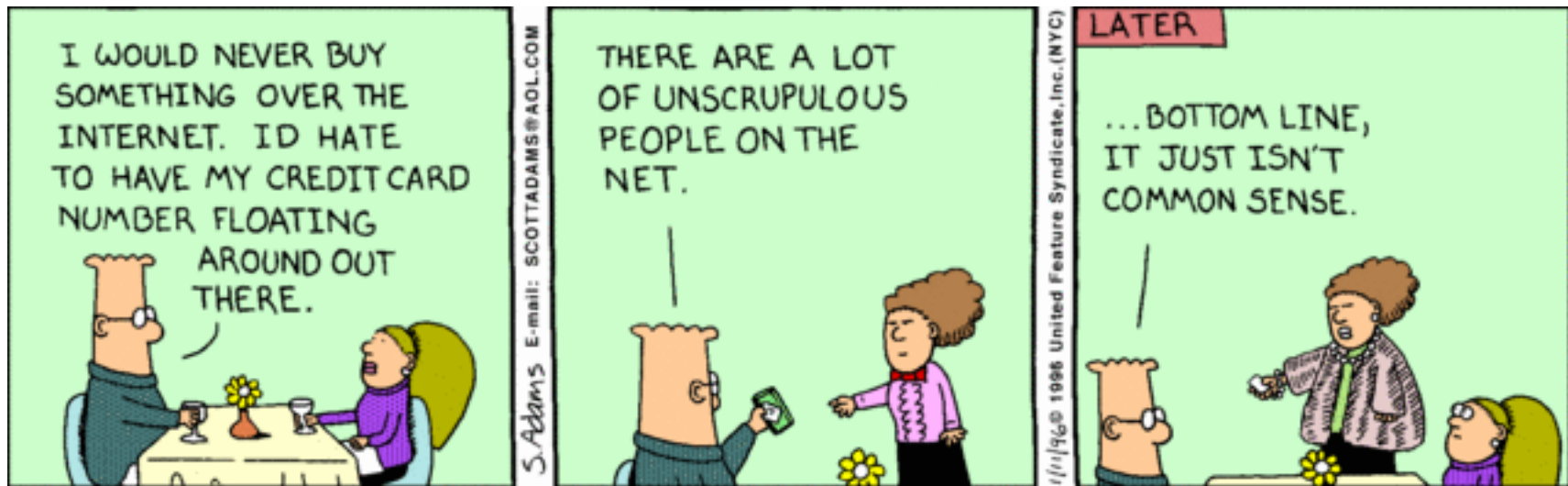
Whom or What Am I Trusting?

“They showed me a telephone, and said they were worried about ‘the microphone.’ When I look at a telephone, I see one high-fidelity microphone and one ‘low-fidelity microphone.’”

“Most people call this a telephone cord. I call it an antenna.”

– Bob Morris, Sr., former Chief Scientist of the National Computer Security Center, NSA

Whom or What Am I Trusting? (2)



Example Secure System Design

- Secure telephone line between FBI and CIA
- Goal: **only people in FBI and CIA buildings can learn what's said in calls**
- Plan:
 - Radiation-proof buildings
 - One entrance/exit per building
 - Armed guards at entrances
 - Guards check ID cards, record all people in/out
 - Pressurized, shielded cable between two buildings
 - No other cables allowed to leave buildings
 - Pass laws to punish people who reveal government secrets
 - Invite NSA to try to steal content of calls
 - Send dummy information, spy on KGB, see if they learn it

Perfect Security: An Unattainable Goal

- Merely a question of how **motivated** adversary is, and how much **money** he has
- No individual technique perfect
 - Pressurized cable only raises cost for attacker
 - Can't completely shield a building
 - People can be bribed, blackmailed
- Could meet stated goal, but it could be inappropriate
 - What if FBI, CIA allow in uncleared visitors?
 - What if employees go home and talk in sleep?
 - **Solution: forbid employees from leaving the building...**

Definitions

- **Security:** techniques to control who can access/modify system
- **Principal:** unit of accountability in a system (e.g., user)
- **Access control:** techniques to restrict operations to particular principals
- **Authentication:** verification of identity of principal making request
- **Authorization:** granting of request to principal

Attacks on Security

- Violation of **secrecy**
 - Attacker **reads data without authorization**
- Violation of **integrity**
 - Attacker **modifies data without authorization**
 - e.g., attacker modifies data on disk
 - e.g., attacker modifies network reply to “read file” request
- **Denial of service**
 - Attacker makes system **unavailable to legitimate users**
 - e.g., overload the system, or cause a deadlock
 - e.g., trigger security mechanism (wrong ATM PIN 3 times)

Building Secure Systems: General Approach

- Figure out what you want to protect, what it's worth
- Figure out which attacks you want to defend against
- State goals and desired properties clearly
 - Not “impossible to break”
 - Better: “attack X on resource Y should cost \$Z”
- Structure system with two types of components:
 - **Trusted**: must operate as expected, or breach
 - **Untrusted**: subverted operation doesn't lead to breach
- Minimize size of trusted components
 - Maybe we should have built secure **room**, not **building...**
- Analyze resulting system, monitor success

Security Is a Negative Goal

- Ensure nothing happens without authorization
 - How do you reason about what a system will **not** do?
- First step: specify who authorized to do what
 - In other words, specify a **policy**

Policy

- Policy: goal security must achieve
 - Human intent—originates from outside system
- Often talked about in terms of subjects and objects
 - Subject: principal
 - Object: abstraction to which access requested (e.g., file, memory page, serial port)
 - Each object supports different kinds of access (e.g., read or write file, change permissions, ...)
- Access control: should operation be allowed?
 - What principal making request? (Authentication)
 - Is operation permitted to principal? (Authorization)

Access Control: Examples

- Machine in locked room, not on network
 - Policy: only users with keys can access computer
- Bank ATM card
 - Policy: only allowed to withdraw money present in your account
 - Authentication: **must have card and know PIN**
 - Authorization: **database tracks account balances**
- Private UNIX file (only owner can read)
 - Authentication: **password to run software as user**
 - Authorization: **kernel checks file's permission bits**
- Military classified data
 - **If process reads "top-secret" data, cannot write "secret" data**

Next: User Authentication

- How to use passwords to authenticate users: at the console, and remotely, over a network
- Attacks against password-based authentication schemes
- Designing robust password-based authentication schemes

Authentication of Local Users

- Goal: only file's owner can access file
- UNIX authentication policy:
 - Each file has an owner principal: an integer **user ID**
 - Each file has associated **owner permissions** (read, write, execute, &c.)
 - Each **process** runs with integer user ID; only can access file as owner if matches file's owner user ID
 - OS assigns user ID to user's **shell process** at login time, authenticated by **username and password**
 - Shell process creates new child processes with **same user ID**
- How does UNIX know the correspondence among **<username, user ID, password>**, for all users?

Straw Man: Plaintext Password Database

- Keep password database in a file, e.g.:
bkarp:3715:secretpw
mjh:4212:multicast
- Passwords stored in file in **plaintext**
- Make file readable only by privileged **superuser** (root)
- `/bin/login` program prompts for usernames and passwords on console; runs as root, so can read password database
- **How well does this scheme meet original goal?**

Cryptographic Primitive: Cryptographic Hash Function

- Don't want someone who sees the password database to **learn users' passwords**
- Cryptographic hash function, $y=H(x)$ such that:
 - $H()$ is **preimage-resistant**: given y , and with knowledge of $H()$, computationally infeasible to recover x
 - $H()$ is **second-preimage-resistant**: given y , computationally infeasible to find $x' \neq x$ s.t. $H(x)=H(x')=y$
- Widely used cryptographic hash functions:
 - MD-5: output is 128 bits, **broken**
 - SHA-1: output is 160 bits; **on verge of being broken**
 - SHA-256: output is 256 bits, **best current practice**

Better Plan: Hashed Password Database

- Keep password database in a file:

```
bkar p : 3715 : Xc8zOP0ZHJkp
```

```
mjh : 4212 : p6FsAtQl4cwi
```

- Instead of password plaintext x , store $H(x)$
- Make file readable by all (!)
- One-wayness of $H()$ means no one can recover x from $H(x)$, right?
 - **WRONG! Users choose memorable passwords...**

Insight: Counting Possible Passwords

- If users pick random n-character passwords using c possible characters, how many guesses expected to guess one password?

$$c^n/2$$

e.g., 8 characters, each ~90 possibilities, **2.15×10^{15}**

- **Do users pick random passwords?**
 - Of course not; very hard to remember
 - Common choice: word in native language
- How many words in common use in modern English?
 - **50,000-70,000** (or far fewer, if you read Metro)

Dictionary Attack on Hashed Password Databases

- Suppose hacker obtains copy of password file (until recently, world-readable on UNIX)
- Compute $H(x)$ for 50K common words
- String compare resulting hashed words against passwords in file
- **Learn all users' passwords that are common English words after only 50K computations of $H(x)$!**
- **Same hashed dictionary works on all password files in world!**