Security

CSE 454

Security

- **Authentication & Authorization**
- **Pests**
  - Spam & Link Spam
  - Robots
  - DoS
  - Viruses & Worms
- **Spyware**
- **Worm Epidemic**

The Two "Auth-" Operations

- **Authentication**
  - "Process of accepting *credentials* from a user and *validating* those credentials against some *authority*"
  - The result is an authenticated identity
- **Authorization**
  - "Process of *determining* whether the authenticated *identity* has *access* to a given resource"
- **Both steps follow this order and both are essential!**

What Can Go Wrong?

- **Authentication breaks if:**
  - Credentials are forged
  - Authority is subverted
  - Validating function is replaced
- **Authorization breaks if:**
  - Authentication identity is forged
  - Access matrix is tampered with
  - Matrix lookup function is replaced
- Lesson: **Security needs to be provisioned on each step!**

Types of Authentication

- **Server authentication**
  - Necessary in e-commerce
  - Achieved via:
    - X.509 certificates, signed by known certificate authorities (CA)
    - Digital signatures using public/private key encryption
- **Client authentication**
  - Necessary in e-commerce
  - Majority of clients typically do not use X.509 certificates, or public/private key pairs
    - How many of you use one of these methods for authentication?

How to Evaluate Proposed Approaches?

Ask:

1. **What problem is the approach trying to solve?**
2. **What are the ways in which the approach can fail (including, be deliberately made to fail)?**
3. **Given the ways the approach can fail, does it really solve the problem at hand?**
4. **What are the costs (financial and otherwise) of deploying a real implementation of the approach?**
5. **Given the failure conditions and costs, is it worthwhile?**

Client Authentication Methods

- **Client certificates**
  - No incentive for clients to have one ⇒ not widely deployed
- **Digital signatures**
  - No PKI yet ⇒ hard to safely distribute public keys
- **Passwords**
  - Most primitive, pervasive method
  - Easy to use, easy to crack: passwords are guessable or users do not remember them
    - Copy-and-store-in-wallet - works well in practice with random passwords
    - Visual passwords - random art; a drawing in lieu of a word
    - S/Key protocol - changing passwords on every communication
    - Smart cards - store random password safely; PIN for theft protection; activated only by a special card reader; European invention

Client Authentication Methods

- **Biometrics**
  - Unique, inherently tied to the individual
  - But:
    - Fingerprinting - non-permanent, could be tampered with
    - Retina scans - non-permanent, invasive, even dangerous
    - Face recognition - high false positives rate, could be easily fooled
    - Voice recognition - high false positive and false negative rate, recordable
    - DNA analysis - slow, extremely invasive, may be non-permanent
    - (Normal) Signature - varies widely (high false negative rate), more appropriate for non-repudiation that authentication

Client Authentication on the Web

- **What assumptions / constraints does the Web environment imply?**
- **Which of the above methods are unsuitable for authentication on the Web?**
- **What remains?**

Motivation

- **Growing need for *personalized, access-controlled* Web-based services**
  - E.g.: nytimes.com, myuw.washington.edu, hotmail.com
- **Some popular authentication mechanisms not suitable for the Web environment**
  - Designed for long-running connections
  - Involve expensive computations - public/private key crypto
  - Authentication identities can be replayed - biometrics
- **Developers lack proper background in security**
- Result: **Proliferation of home-grown weak authentication schemes**

Limitations on Web Authentication Schemes

- **Must use only *widely deployed*, *portable* and *lightweight* technologies**
  - No smart cards or client certificates; JavaScript may be ok
- **Must require minimum user involvement**
  - No password re-typing or perpetual dialog boxes
- **Must not unduly overload servers with expensive computations**
  - No public-key crypto; cryptographic hashes are fine
- **Must store client state in a very limited space**
  - E.g.: cookies on the client, (maybe) a database on server

Not All Web Authentication Schemes Are Created Equal

Designs differ depending on:

- **Type of service**
  - General subscription
    - Online newspapers and libraries
  - User customization
    - Online identities, per-user content filtering
- **Security needs**
  - Sensitivity of the client data
    - Store data on server and put an index to it in a client cookie
  - Load tolerance on the server
    - Delicate tradeoff with clients' need for strong protection

Threat Model or What Attacks Do We Fear?

- **Forging* an authentication token for**
  - A *random* user (a.k.a. existential forgery)
    - Useful for free access to subscription services
  - A *chosen* user (a.k.a. selective forgery)
    - Allows access to data for any selected user
  - *All* users (a.k.a. total break)
    - Allows forging tokens for all users at any time

*** forging ≠ replay attack**

Threat Model or What Adversaries Do We Fear?

Active Adversary

Eavesdropping Adversary

Interrogative Adversary
- Queries the server (adaptively, based on previously seen data)
- Creates new accounts (assuming no out-of-bound throttling)
- Uses publicly known information

- Records traffic between users and the server
- Replays selected captured messages

- Modifies / injects traffic between users and the server
- Mounts man-in-the-middle attack

- Resolution: **Viable schemes must at least protect against interrogative adversaries!**

Hints for Designing Client Authentication Schemes

Disclaimer:

Hints are useful, but following them is neither necessary, nor sufficient for security

Hints: Use Cryptography Appropriately

- **Using crypto is inescapable if you want to protect from adversaries!**
- **Hint #1: Assess your needs for protection**
  - Tradeoffs between usability and complexity
- **Hint #2: Choose a "tried and true" existing scheme**
  - Home-grown schemes are almost always trivial to break

Hints: Use Cryptography Appropriately

If you *absolutely must* design your own scheme:
- **Hint #3: Think twice! Ask those who know better!**
- **Hint #4: Have it reviewed by security experts**
  - Announcing it loudly is good but not sufficient
- **Hint #5: Keep the scheme simple**
  - Makes it easier to analyze for security
- **Hint #6a: Do not rely on the secrecy of the protocol**
  - Gives you false sense of security until someone figures it out
- **Hint #6b: Instead, rely on the secrecy of *keys***

Hints: Use Cryptography Appropriately

- **Hint #7: Understand the properties and details of crypto primitives you use**
  - Many provide some assurances, but not other (e.g., SSL)
  - Many make fine-print assumptions
    - UNIX crypt() hash function truncates input beyond 8 characters
- **Hint #8: Avoid composing security schemes**
  - May weaken the composite, even if secure in isolation
    - E.g., using the same secret key for multiple purposes

Status on Using Passwords

- **Users don't want passwords**
  - Tradeoff between usability and security
  - Users tend to pick poor (easy) passwords
    - Do not suggest ideas - they will blindly follow it
- **Users tend to reuse passwords across many sites**
  - How many different passwords do you use?
  - How many of them do you commit to memory?
  - How many of them do you have written somewhere (as a backup)?
- **Compromising a password leads to impersonation**

Hints: Protect Passwords

- **Hint #9: Prohibit easy-to-guess passwords**
  - Otherwise: an easy prey for dictionary attacks
  - Change periodically, enforce non-similarity, minimum password length, special characters
  - Giving out (random) passwords may turn off users
- **Hint #10: Never reveal a user's password**
  - User knows it, everyone else has no reason to ask for it
  - Keep passwords always encrypted in transfer
    - Login over SSL for confidentiality of password exchange
  - Avoid unnecessary password transfers
    - Give out and use (temporary) client authentication tokens instead

Hints: Protect Passwords

- **Hint #11: Redo authentication before security-sensitive operations**
  - E.g.: changing passwords
  - Avoids attacks through replayed authentication tokens

Hints: Handle Authentication Tokens Wisely

- **Hint #12: Avoid predictable authentication tokens**
  - E.g.: publicly available info, sequential ID numbers, etc.
- **Hint #13: Protect tokens from tampering**
  - Tokens may contain sensitive user info
  - Use only strong cryptographic hash functions (e.g., no CRC)
  - Use a keyed message digest (e.g., MAC, no MD5)
- **Hint #14: If combining multiple data into a token, separate components unambiguously**
  - Avoids a splicing attack:
    - "Alice" • "213" • "Bob" == "Alice2" • "13" • "Bob"

Hints: Handle Authentication Tokens Wisely

- **Hint #15: Encrypt tokens**
  - For tokens stored in cookies and sent over SSL, set Secure flag
  - Prevents eavesdroppers from capturing and replaying tokens
- **Hint #16: Do not include a token as part of a URL**
  - Otherwise, token may leak through plaintext channels
    - E.g.: cross-site scripting attack using the HTTP Referer field
- **Hint #17: Avoid using persistent cookies**
  - If cookie (file) is leaked, attacker can impersonate user
  - Can users defend against this threat (the authentication scheme designer may have been negligent)?

Hints: Handle Authentication Tokens Wisely

- **Hint #18: Make authentication tokens expire:**
  - Store a tamper-resistant timestamp in cookie, or keep token expiration time on the server
    - Limits the potential damage in case a token leaks out
- **Hint #19: Do not trust the client…**
  - … to enforce token expiration (manipulating a cookie is easy)
  - … (in general) for anything that the client can possibly forge
- **Hint #20: To prevent replays of leaked tokens:**
  - Keep tokens confidential and mint new ones after each use
  - Bind tokens to network addresses
    - But DHCP users' tokens may expire prematurely

Sample Authentication Scheme

- **Goals**
  - Statelessly verify authenticity of request and its contents
  - Explicitly control lifetime of token
  - Portability
- **Design choice**
  - Authentication cookies
    - Anyone with a valid cookie has access to protected server content
- **Claim**
  - Secure against an interrogative adversary
  - If layered over SSL with server authentication, secure against an active adversary

Cookie Basics

- **HTTP is a stateless protocol**
- **Client IDs generated by server, stored on client**
- **Sent back to server with subsequent requests**
- **Cookie attributes:**
  - Data - used to uniquely identify client
  - Domain - cookie only applies to this server domain
  - Path - server path
  - Secure flag - should cookie data be encrypted?
  - Expiration - current session or physical time

Suggested Cookie Structure

$exp=t\&data=s\&digest=MAC_k(exp=t\&data=s)$

t → expiration time (seconds past 1970 GMT)
s → data, associated with the client
k → server secret key
MAC → strong cryptographic hash function

$HMAC_k(M) ::= H(k \oplus 0x5c \bullet H(k \oplus 0x36 \bullet M))$

where H ∈ {SHA1, MD5}, M is the message

Disecting the Scheme

- **Expiration time:**
  - Avoids keeping server state
  - Tradeoff between potential damage and frequent reauthentication (security vs. usability)
  - Should users be allowed to control it?
- **Data:**
  - Sensitive data should not be stored here
    - If needed, store cryptographically random session ID, while keeping important data on server
  - Balance between respecting users' privacy and saving server resources
    - Likely to be biased in favor of the latter

Disecting the Scheme

- **Key:**
  - Recommended length is twice that of block encryption ciphers (~160 bits or more)
    - Fends off birthday attacks

Disecting the Scheme

Strengths:
- **Simplicity**
- **Authenticating clients:**
  - Requires O(1) server state (for the key)
  - Takes O(1) time
  - Would depend on number of clients if server state were kept
- **Easier to deploy multiserver systems**
  - No need for dynamically shared data between servers

Disecting the Scheme

Weaknesses:

- **Server is vulnerable against colluding clients**
  - Clients more likely to share temporary tokens than passwords
  - How many other people's passwords do you know?
- **No mechanism for selective secure token revocation**
  - Unnecessary for short sessions
    - Separation of policy and mechanism?
  - If needed, keep session status on server
    - Yahoo does it
  - But, allows simultaneous revocation of all tokens
    - By changing the secret server key

Security Analysis

Strength of authentication scheme depends on:

- **Strength of MAC function**
- **Secrecy of server key**
- **Strength of server key and frequency of changing it**
  - Longer keys adversely affect performance of hash functions
- **Strength of client passwords against guessing and dictionary attacks**

Performance Factors

- **HMAC-SHA1**
  - 1.2 ms / request
  - Runs on small chunks of data
- **SSL**
  - 90 ms / request
  - Runs on the entire HTTP stream
  - New connections are costly to setup, session resumption helps

Other Authentication Schemes

- **HTTP Basic Authentication**
  - Sends username and password repeatedly in cleartext
  - Falls prey to eavesdropping adversaries
    - `dsniff` - automated tool for sniffing authentication exchanges
- **HTTP Digest Authentication**
  - Encrypts username and password before transmitting
  - Little client support yet
- **SSL**
  - Requires public-key crypto in X.509 certificates
  - No global PKI → no wide support for client certificates
  - Involves heavyweight operations

Conclusions

- **No single authentication scheme can effectively and efficiently meet the requirements of all Web sites and Web clients**
- **There are clear guidelines (but no standards yet) for designing secure authentication schemes**

Open Issues

- **What can end users do to protect themselves?**
  - Those who can provide a solution (i.e., vendors) have no incentive to do so.
  - Those who really care about finding a solution (i.e., clients) cannot create one.
- **Should there be a standard for authentication protocols? What factors play against establishing such a standard?**
- **Would you trust a centralized authentication service (such as Microsoft Passport) with your data? A step in which direction is this - forward or backward?**

SPAM

- **Problem**
  - Zero marginal cost of sending an email
- **Solutions**
  - Machine learning client to detect spam
  - Brightmail
    - Dummy accounts
    - Correlate SPAM messages
    - Supply fingerprint to enterprise customers
  - Client refuses messages from unknown senders, until
    - They respond to a Turing test query
    - They execute a computationally expensive applet
    - Micropayment

Link Spam

- **Keyword / Meta tag stuffing**
  - Linguistic spoofing
- **Multiple titles**
- **Tiny fonts**
- **Invisible text**
  - <body bgcolor="FFFFFF">
  - Your text here
  - Problem: takes up space. Size=1? Bottom?
- **Doorway / jump pages**
  - Fast meta refresh
- **Cloaking ~ Code swapping**
- **Pagerank spoofing (Link newtworks)**

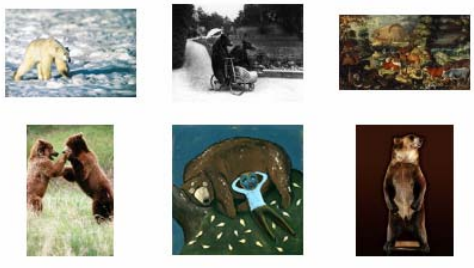
Robots

- **Threat: automatic creation of accounts**
  - Paypal
  - Storage associated: Hotmail, Yahoo communities…
  - Adbots in chat rooms
  - Online polls
- **Solutions**
  - Turing tests
    - Distorted speech recognition
    - Overlayed distorted text recognition
    - CAPTCHA
      - Completely automated public turing test to tell computers and humans apart
      - http://www.captcha.net/

Gimpy: Type 3 words

Mori & Malik (UCB) program solving ez-gimpy with accuracy 83%

Semantic Tests



ESP Game

http://www.espgame.org/

Viruses

- **Defn**
  - Requires human action to spread
  - Infects most files on local computer
  - Doesn't automatically spread across network
  - Carries payload (destructive or annoying messages)
- **Common MO**
  - Macro attached to office document
- **Solutions**
  - Fingerprint based (to detect viruses)
  - Application checksums (to detect tampering)

Worms

- **Defn**
  - Automatically spreads to other systems
- **MO**
  - Protocol worms
  - Hybrid virus / worms
- **Solutions**

Spyware: analysis and mitigation

Steven Gribble

Department of Computer Science and Engineering
University of Washington

kingsofchaos.com

- **A benign web site for an online game**
  - earns revenue from ad networks by showing banners
  - but, it relinquishes control of the ad content

banner ad from adworldnetwork.com (a legitimate ad network)

inline javascript loads HTML from ad provider

Incident: January 2004

- **kingsofchaos.com was given this "ad content"**

```
<script type="text/javascript">document.write('
\u003c\u0062\u006f\u0064\u0079\u0020\u006f\u006e\u0055\u
006f\u0077\u0050\u006f\u0070\u0075\u0070\u0028\u0029\u0
03b\u0073\u0068\u006f\u0077\u0048\u0069 ...etc.
```

- **This "ad" initiated a cascade of redirections through many sites, and ultimately:**
  - bombarded the user with pop-up ads
  - hijacked the user's homepage
  - exploited an IE vulnerability to install spyware

What's going on?

- **The advertiser was really an ad-spammer**
  - his goal: **force** users to see ads from his servers
    - revenue from ad "affiliate programs"
    - paid to show ads for bogus anti-spyware software
- **Why install spyware?**
  - to show ads whether or not the victim is on the Web
  - to make escape hard
    - his spyware shows his ads
    - the hijacked home page shows his ads
    - some of his ads re-install spyware and re-hijack

Take-away lessons

- **Your PC has value to third parties**
  - spyware tries to steal this value from you
    - adware: eyeballs and demographic information
    - spyware: sensitive data, PC resources
- **Web content should never be trusted**
  - even if its direct provider is
- **Consumer software and OSs are weak**
  - browsers are bug-ridden
  - OSs cannot deal with malicious software

What is Spyware?

- **Incredibly difficult to define precisely**
  - no clean line between good and bad behavior
  - hard to define 'informed consent'
- **Spyware is a *software parasite* that:**
  - collects info of value and relays it to a third party
  - hijacks functions or resources of PC
  - installs without consent of user, resists de-installation
- **Spyware provides value to others, but not to you**

Types of spyware

Class	# signatures
Cookies and web bugs	47
Browser hijackers	272
Adware	210
Keyloggers	75
Dialers	201
Backdoors / monitors	279

From the "Spybot S&D" database, Feb. 2005

Infection methods

- **Piggybacking on legitimate software**
  - provides revenue stream for free software vendors
- **Drive-by downloads**
  - malicious Web content exploits browser vulnerability
  - software is installed and run silently
- **Installed during remote attack**
  - some worms now carry spyware payload
- **Snowball effect from existing spyware**
  - trojan downloaders

Spyware trends

- **Most Internet PCs have it**
  - June '03: 80% of Internet-connected PCs are infected
  - *[AOL/NCSA online safety study]*
- **It's getting more vicious**
  - December '04: 14% of enterprise PCs have backdoor or monitor spyware
  - doubled between October '04 and December '04
  - *[Webroot reported scan statistics]*
- **Convergence of threats**
  - worms, viruses, spyware, botnets are fusing

Two research studies

- **November 2003 study of adware within UW**
  - passive network measurement of entire campus
  - measured spread of four adware programs
- **Sneak preview of crawler-based study**
  - active retrieval of content from the Web
  - how much is "out there," and who is spreading it

UW adware study

- **Examined four programs**
  - Gator, Cydoor, SaveNow, and eZula
  - piggyback installation, adware and HTML rewriting
- **Derived network signatures**
  - look for the spyware "phoning home"
    e.g., Gator traffic contains Gator/x.xx UserAgent and is sent to a *.gator.com host
  - signatures permit passive network monitoring

Method

- **Network monitor deployed at UW**
  - sniffs packets sent between UW hosts and Internet
  - gathered a 7 day Web trace
    - Aug. 26th – Sept. 2nd, 2003
  - looked for packets that match signatures
    - traffic matches signature ⇒ sender has spyware

The major result

	WWW	Gator	Cydoor	SaveNow	eZula
# clients (% clients)	**31,303 (100%)**	**1,077 (3.4%)**	**399 (1.3%)**	**406 (1.3%)**	**63 (0.2%)**

- **5.1% of UW hosts have ≥ 1 of these programs**
- **This may appear small, but:**
  - Only considers 4 spyware programs out of thousands
  - University may be non-representative
    - modem pool has 2.5x higher infection rate
  - a Gator vulnerability => 1000+ UW hosts at risk

Security flaws

- **Gator & eZula "auto-update" their code, data**
  - periodically download ZIP file, unzip into filesystem
- **No integrity / authenticity checks on updates**
  - Could attack with DNS spoofing or TCP hijacking
  - We could install an executable in "Startup" directory
  - Tens of millions of hosts susceptible
- **They communicated flaws to both companies**
  - Gator flaw was quickly repaired

Is there "at-risk" behavior?

- **# of web objects downloaded**
  - fewer than 1000 requests per week: 1.8% have Gator
  - more than 12,000 requests per week: 8.9% have Gator
- **# of executables downloaded**
  - none downloaded over week: 0.9% have Gator
  - one or more over week: 8.4% have Gator
- **using the Kazaa P2P client**
  - issued one or more Kazaa request: 38% have Gator
    but…62% of spyware infections are in hosts that didn't issue a Kazaa request

Two research studies

- **November 2003 study of adware within UW**
  - passive network measurement of entire campus
  - measured spread of four adware programs
- **Sneak preview of crawler-based study**
  - active retrieval of content from the Web
  - how much is "out there," and who is spreading it

Two studies

- **November 2003 study of adware within UW**
  - passive network measurement of entire campus
  - measured spread of four adware programs
- **Sneak preview of crawler-based study**
  - active retrieval and analysis of Web content
  - how much is "out there," and where is it coming from?

Method

- **Crawl subsets of Internet to find spyware**
  - used "heritrix" public domain crawler
  - downloaded .zip, .exe, .cab, etc. (programs)
- **Cluster of virtual machines to analyze programs**
  - "forked" a clean Windows VM per program
    - installed program, ran anti-spyware tool to analyze
  - O(1 min) per program
    - on 10-node cluster, O(15,000) programs per day
    - many performance optimizations possible

Major result

- **Web sites crawled: 12,000**
  - URLs retrieved: 23,714,927
- **# of executable files downloaded: 9,330**
  - # infected with spyware: 766 (8.21%)
  - unique spyware programs found: 137

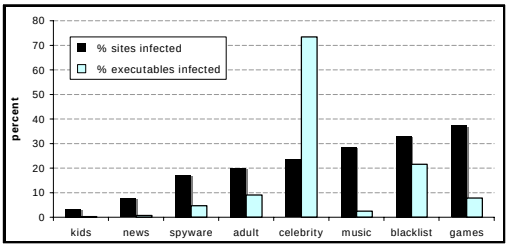
1 in 12 **executables on the Internet have spyware!**

What kind of spyware is out there?

Behavior	% spyware
Adware	57%
Browser hijackers	56%
Keyloggers	0.06%
Dialers	0.1%
Backdoors / monitors	15%

- **Other stats:**
  - 58% try to evade discovery or removal
  - 32% monitor Web browsing behavior
  - most popular: eZula, 180 solutions, SaveNow

Where does it come from?



- **spyware purveyors "troll" popular destinations**
- **blacklists aren't terribly useful**

Spyware Wrap-up

- **Spyware affects many people**
  - 5% of UW computers have adware
    - substantially underestimates all spyware
  - 1 in 12 executables on the Internet have spyware
- **Most spyware appears benign**
  - adware is the most rampant
  - but, trojans and monitors are on the rise
- **Even "benign" spyware can harm you**
  - hidden risk of security flaws, instability
  - no opportunity to mitigate or isolate