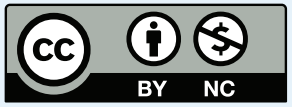
Web and Email Security



Continuing Authentication

Continuing Authentication

- Assume initial authentication is by password
- How is continuing authentication done?
- Two principal ways: cookies and hidden values
- Both have their limits
- Fundamental issue: both are sent by *untrusted clients*

Untrusted Clients

- The web site is interested in identifying users
- (Some) users have incentive to cheat
- The goal of the web site is to make cheating impossible
- But the web site doesn't control the client software or behavior

Repeat: Untrusted Clients

- You can *never* trust the client
- All input *must* be sanitized, scrutinized, etc.
- Solutions: server-side storage or *cryptographic sealing*

Server-Side Storage

- Provide the client with an index to some server-side file
- Client sends back the nonce; server looks up the identity (or other session details)
- Caution: make certain that nonces are not guessable or findable by exhaustive search. Also make sure they're not (easily) stealable from other users
- Problem: server-side storage can be exhausted; sessions must be of finite duration

Cryptographic Sealing

- After the user logs in (somehow), create a string that contains the userid
- Encrypt (optional) and MAC this string, using keys known only to the server; pass the string to the client
- When the string is sent to the server, validate the MAC and decrypt, to see who it is
- Only the server knows those keys, so only the server could have created those protected strings (similar to Keberos TGT)
- Optional: include (especially) timestamp, IP address, etc.

Hidden Values

- Protected userid string can be embedded in the web page, and returned on clicks
- Embed in URLs — but then they're visible in log files
- Make them hidden variables passed back in forms:

```
<INPUT TYPE=HIDDEN NAME=REQRENEW>
<INPUT TYPE=HIDDEN NAME=PID VALUE="2378">
<INPUT TYPE=HIDDEN NAME=SEQ VALUE="200609280023
<P><INPUT TYPE=SUBMIT VALUE="Renew Items"><INPUT
</FORM>
```

Cookies

- More commonly used
- Allow you to re-enter site
- Are sometimes stored on user's disks

Protecting Authentication Data

- Continuing authentication data is frequently transmitted unencrypted!
- Most sites don't want the overhead of SSL for everything
- Credentials are easily stolen, especially in wireless hotpots (always use HTTPS with gmail)
- Usual defenses: lifetime; reauthenticate before doing really sensitive stuff

Sidebar: Cookies and JavaScript

- IE trusts local content more than it trusts downloaded files
- Content is "local" if it's coming from a file on the user's disk
- Each cookie is stored as a separate file
- Suppose you put a script in a cookie, and then referenced it by filename?
- Now you know why browsers use random characters in some of their filenames...
- (Partially changed by Windows XP SP2)

Cross-Site Scripting (XSS)

- Problem usually occurs when sites don't sanitize user input to strip HTML
- Example: chat room (or MySpace or blog sites) that let users enter comments
- The "comments" can include JavaScript code
- This JavaScript code can transmit the user's authentication cookies to some other site

Why It Works

- A JavaScript program can only access data for the current web site
- But JavaScript from a site can access that site's cookies
- Because of the XSS bug, the JavaScript *from that site* contains malicious code
- It can therefore steal cookies and send them to some other site, via (say) an IMG URL

CS @CU

Sanitizing Input

- Very hard to do properly
- Whitelist instead of blacklist — accept `<I>` instead of blocking `<SCRIPT>`
- Watch for encoding: `%3C`
- Watch for Unicode: `&#x3C;` or `&#x003c;` or `&#x00003c;` or `&#60;` or ...
- Probably a way to write it in octal, too
- Unicode is tricky — see RFC 3454. What do *all* of your users' browsers understand?

GET versus POST

- Web requests can use `GET` or `POST` commands
- `GET` puts parameters in the URL; `POST` sends it as data after the command
- The nice thing about `GET`: you can reuse the URL
- But — URLs are logged, by servers and sometimes proxies
- *Don't* put anything sensitive in the parameters!
- `POST` can't be reused, but doesn't cause sensitive data to be logged

Server-Side Security

Protecting the Server

- Servers are very tempting targets
- Defacement
- Steal data (i.e., credit card numbers)
- Distribute malware to unsuspecting clients

Standard Defenses

- Check all inputs
- Remember that *nothing* the client sends can be trusted
- Scrub your site

Server-Side Scripts

- Most interesting web sites use server-side scripts: CGI, ASP, PHP, server-side include, etc.
- Each such script is a separate network service
- For a web site to be secure, *all* of its scripts must be secure
- What security context do scripts run in? The web server's? How does the server protect its sensitive files against malfunctioing scripts?
- This latter is a particular problem with server plug-ins, such as PHP
- Partial defense: use things like `suexec`

Injection Attacks

- Often, user-supplied input is used to construct a file name or SQL query
- Bad guys can send bogus data
- Example: a script that sends email collects a username and executes `/usr/bin/sendmail username`
- The bad guy supplies `foo; rm -rf /` as the username
- The actual code executed is `/usr/bin/sendmail foo; rm -rf /`
- Oops...

Scrubbing Your Site

- What is *really* being served?
- Web servers often come with default scripts — some of these are insecure
- Example: `nph-test-cgi` that used to come with Apache
- Example: proprietary documents; Google for them:

```
filetype:pdf "company confidential"
```

- (By the way, many document have other, hidden data)
- Can Google for some other vulnerabilities, too

Users

- If your site permits user web pages — this deparment? — you have serious threats
- Are the user CGI scripts secure?
- Can users run PHP scripts in the browser's security context?
- Are all of these secure?

ACLs

- Web sites can block or permit certain IP addresses:

```
Deny from 192.168.205
Deny from phishers.example.com
Deny from moreidiots.example
Deny from ke
```

- Possible to allow just a few addresses, too:

```
Order deny,allow
Deny from all
Allow from dev.example.com
```

- Caution: do *not* trust domain names

SSL and Proxies

- On proxied web connections, the proxy does not see the full URL
- It only sees the hostname and port:

```
CONNECT www.example.com:443 HTTP/1.1
User-Agent: Mozilla/5.0 ...
Proxy-Connection: keep-alive
Host: www.example.com

(SSL starts here)
```

- But — the hostname (such as www.reallynastystuff.com) is seen and logged
- Performance note: SSL pages are not cached
- SSL-only site can be 10–15× more expensive

Email Security

The Usual Questions

- What are we trying to protect?
- Against whom?

Assets

- Confidentiality — people often discuss sensitive things via email
- Authenticity — who really sent the email?
- Anti-spam?
- Phishing?
- Authenticity has many motivations here

Security at Rest

- TLS protects data *in motion* — during communication
- For email, do we want that or do we want security *at rest* — while the email is stored somewhere
- Other terminology: *transmission security* versus *object security*
- Usual answer: both
- Security at rest is much harder

In Motion versus at Rest

In Motion	*At Rest*
Authentication keys are transient	Authentication keys must last as long as data must be provably valid
Reject expired certificates	Must store and accept old ones them for later use
Negotiate algorithms	Assume known algorithms
Decryption failures noticed immediately	Decryption failures noticed much later
Restart communications on decryption failure	Decryption failure unknown to sender

Components

- Mail User Agents (MUA): Outlook, Thunderbird, webmail sites, etc.
- Mail submission servers
- Mail Transfer Agents (MTAs)
- Mail Receivers
- Mail storage and retrieval systems (IMAP, POP, etc.)

Threats

Eavesdropping

- Most obvious way to read email: eavesdropping
- The bad guy "simply" listens to the network
- Harder than it sounds, except for some wireless nets
- Frequently used by police and intelligence agencies, i.e., the FBI's *Carnivore* device

CS@CU

Password Theft

- Most email is retrieved by login and password
- Anyone who gets your password can read your email
- It's much easier for an eavesdropper to pick those up — passwords are usually sent each time someone polls for new email

Hacking

- The real threat to email is while it's in storage
- This can be temporary storage, waiting for you to pick it up
- It can also be your personal machine, for email you've sent or received
- What if your laptop is stolen? Does it have plaintext copies of all the secure email you've sent and received?

Screen Dumps

- Connect via X11
- Use some other Trojan horse software to dump user's screen periodically
- Reflection off the back wall...

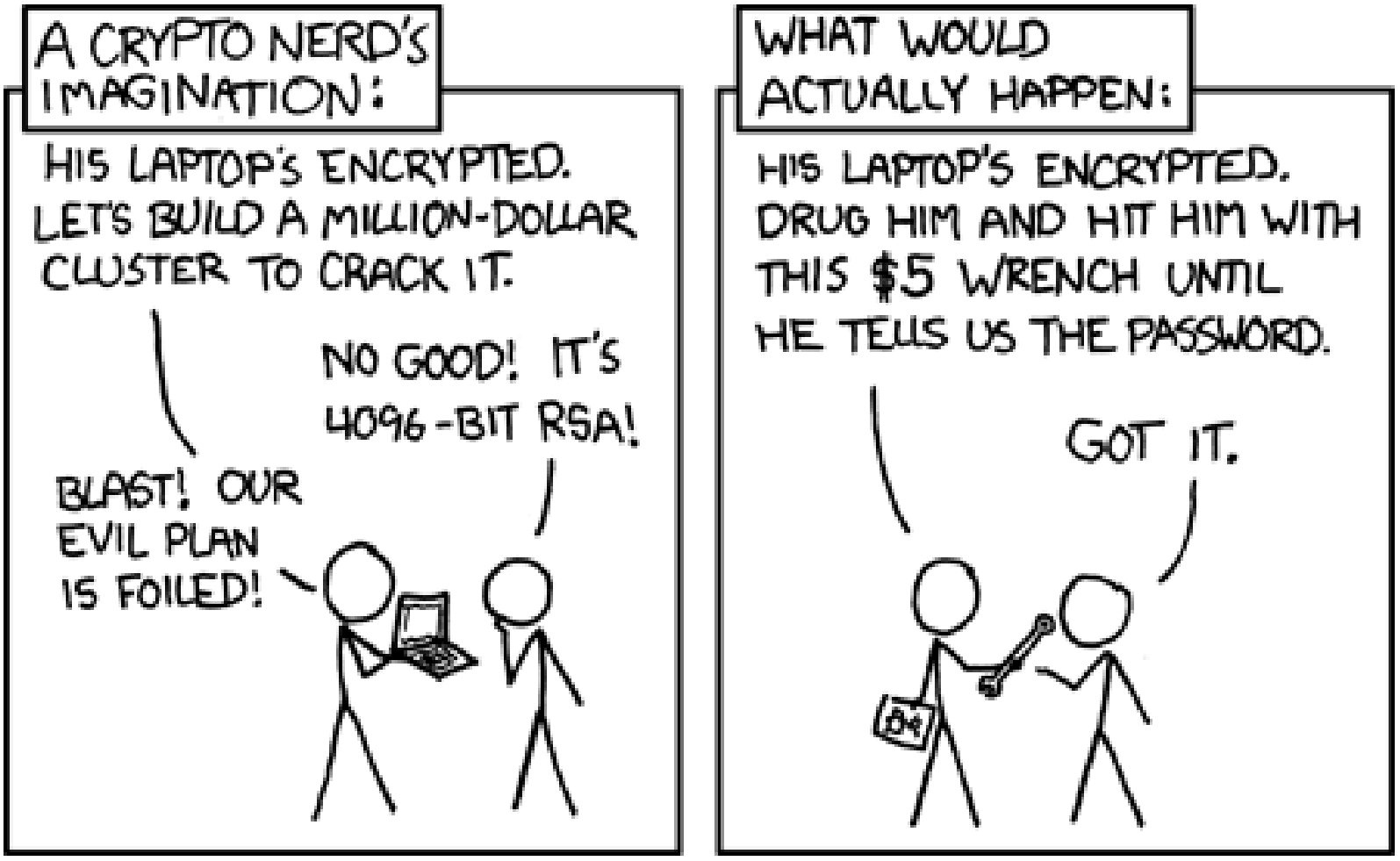
Subpoena Attacks

- What if your records are subpoenaed?
- This is a legal issue; technical wiggling won't help!
- Even a search warrant is very disruptive

Rubber Hose Cryptanalysis

- What if the local secret police want to know what some intercepted email says?
- Protecting human rights workers was one of the original goals for PGP!
- It's public key-encrypted — you *can't* read it
- If the signature is encrypted, they can't even prove you sent it
- Of course, people like that don't care much about proof, and they don't like to take "no" for an answer...

From http://xkcd.com/538/



Spoofing

- Ordinary email is trivial to spoof
- On timesharing machines and web mailers, the systems can tack on the userid
- On PCs, individuals set their own addresses
- *No* security — if you need to authenticate email, you have to use crypto

Systems Issues

- Only read email on secure machines
- Only connect to them securely
- Watch out for buggy mailers and systems
- But if the process of reading secure email is too cumbersome, your email will be insecure, because you'll never use the secure version
- Finding the right tradeoff is a difficult engineering choice