Network Security - ISA 656
Web Security

Angelos Stavrou

September 28, 2008

Web Security

- Crypto (SSL)
- Client security
- Server security

SSL

- Mostly covered last time
- Crypto is insufficient for Web security
- One issue: linkage between crypto layer and applications

Trusting SSL

- What does the server *really* know about the client?
- What does the client *really* know about the server?

The Server's Knowledge of the Client

- What has SSL told the server?
- Unless client-side certificates are used, *absolutely nothing*
- SSL provides a secure pipe. *Someone* is at the other end; you don't know whom

How Did That Happen?

- In theory, we could have had digitally-signed purchase orders linked to credit card accounts
- That would have required that Netscape, when it invented SSL, have some way to issue client-side certificates that were linked to credit card accounts *and* didn't have the credit card number in the certificate
- Netscape couldn't have done that; only the banks could have
- Back in 1994, banks didn't believe in this new-fangled Internet thing (remember that until Windows 95, TCP/IP wasn't included in Windows

SET

- A few years later, Visa and Master-card (and eventually Amex) tried
- They developed a protocol called SET (Secure Electronic Transactions)
- It provided client-side certificates linked to credit cards
- In theory, merchants wouldn't need to know (and store) credit card numbers
- Virtually no one used it
- The reasons were both technical and financial

The Failure of SET

- It required client-side software

⇒ Very few people install extra software

- Client-side certificates are hard to use — what if you use several computers?
- There was too little financial incentive for merchants, so they couldn't give customers a discount for using SET
- It *still* permitted merchants to store credit card numbers; in fact, they were present, albeit encrypted, in the certificate

⇒ Merchants use credit card numbers as customer tracking keys for databases

- Good crypto alone isn't sufficient!

Aside: The SET Root Certificate

- Who should control the SET root certificate, used to sign the Visa, Master-card, etc., top-level certificates?
- (SET certified Visa et al.; they certified banks, who in turn issued customer certificates)
- It would be catastrophic if the root's private key were compromised
- Visa didn't trust Master-card, or vice-versa
- Solution: a sacrificial PC signed all of the second-level certificates, at which point it was physically *smashed*. Different organizations took home different pieces. . .

The Client's Knowledge of the Server

- The client receives the server's certificate. Does that help?
- A certificate means that *someone* has attested to the binding of *some* name to a public key.
- Who has done the certification? Is it the right name?

Who Issues Web Certificates?

- Every browser has a list of built-in certificate authorities
- The latest version of Firefox has 138 certificate authorities!
- Do you trust them all to be honest and competent?
- Do you even know them all?
- (Baltimore Cyber-trust is listed. It *sold* its PKI business in 2003. Are the new owners trustworthy?)

Mountain America Credit Union

- Early this year, someone persuaded a reputable CA to issue them a certificate for Mountain America, a credit union
- The DNS name was `www.mountain-america.net`
- It looks legitimate, but the *real* credit union site is at `www.mtnamerica.org`.
- (There's also `www.mountainamerica.com`, a Las Vegas travel site)
- Which site was *intended* by the user?

A Fake Certificate

A Technical Attack

- Usually, you shop via unencrypted pages
- You click "Checkout" (or "Login" on a bank web site)
- The *next page* — downloaded without SSL protection — has the login link, which will use SSL
- What if an attacker tampers with that page, and changes the link to something different? Will you notice?
- Note that some small sites out-source payment processing. . .

Conclusions on SSL

- The cryptography itself seems correct
- The human factors are dubious
- Most users don't know what a certificate is, or how to verify one
- Even when they do know, it's hard to know what it should say in any given situation
- There is no rational basis for deciding whether or not to trust a given CA

Protecting the Client

Web Browser Security

- User interface
- Buggy code
- Active content

The Attackers' Goals

- Steal personal information, especially financial site passwords
- Turn computers into "bots"
- Bots can be used for denial of service attacks, sending spam, hosting phishing web sites, etc.

Buggy Code

- *All* browsers are vulnerable, and getting worse
- Browser bugs (Symantec):

Browser	1H2005	2H2005	1H2006
IE	25	25	38
Firefox	32	17	47
Opera	7	9	7
Safari	4	6	12

- Exposure period (Symantec):

Browser	2H2005	1H2006
IE	25	9
Firefox	-2	1
Safari		5
Opera	18	2

Why Are Browsers So Insecure?

- Their task is complex
- They are dealing with many untrusted sites
- By definition, browser inputs cross *protection domains*
- It is likely that no browser is significantly better than any other in this regard — they're *all* bad

Active Content

Active Content

- There's worse yet for web users: active content
- Typical active content: Javascript, Java, Flash, ActiveX
- Web pages can contain more-or-less arbitrary programs or references to programs
- To view certain web pages, users are told "please install this plug-in", i.e., a program
- "Given a choice between dancing pigs and security, users will pick dancing pigs every time." (Ed Felten)

Javascript

- No relationship to Java — originally called LiveScript (EvilScript?)
- Source of most recent security holes, in Firefox and IE
- No clear security model
- Crucial link in *cross-site scripting* attacks

AJAX

- AJAX — Asynchronous Javascript and XHTML
- Permits highly interactive web pages, i.e., Google Maps
- Security implications for client and server are still quite unclear (but are likely to be bad. . .)

ActiveX

- *The* biggest active content design error
- Over 1,000 ActiveX controls on a typical new, out-of-the box, machine
- Translation: over 1,000 different pieces of code that can be run by almost any web page
- But wait, there's more!

Downloading ActiveX Controls

- Any web page can download other controls
- Translation: any web page can download an arbitrary piece of code to run on a user's machine
- The only protection is a digital signature on the downloaded code
- But at best that identifies the author — see the previous discussion of certificates!
- There is *no* restriction on what the code can do

Why ActiveX?

- It can be used for some very beneficial things, such as Windows Update
- It can be used to "enhance" the user's web experience, i.e., provide dancing pigs
- Business reasons? Tie web sites to Windows and IE?
- Only IE has ActiveX. This is the single biggest security difference between IE and Firefox

Continuing Authentication

Continuing Authentication

- Initial authentication is usually 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

Protecting Identification Information

- After the user logs in (somehow), create a string that contains the user-id
- 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 timestamp, IP address, etc.

Hidden Values

- Protected user-id 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="2006092800235
<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 unencrypted!
- Most sites don't want the overhead of SSL for everything
- Credentials are easily stolen
- Usual defenses: lifetime; re-authenticate 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

Sanitizing Input

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

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 malfunctioning 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 department? — 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?