Architecture

cc BY NC

Web Servers and Security

- The Web is the most visible part of the net
- Two web servers—Apache (open source) and Microsoft's IIS—dominate the market

  ☞ Apache has 24%; IIS has 45%. (Nginx, another open source web server, is in third place with 14%.) (source: `http://news.netcraft.com/archives/web_server_survey.html`)
- Both major servers have lots of functionality
- Are they secure? Let's look at Apache.

Metanote on Program Complexity

- Both Apache and IIS are very large, complex programs
- Large, complex programs are often buggy; these are no exception
- Both have had security problems
- IIS used to be *very* insecure:

  > *Using Internet-exposed IIS Web servers securely has a high cost of ownership. Nimda has again shown the high risk of using IIS and the effort involved in keeping up with Microsoft's frequent security patches.*
  >
  > —The Gartner Group, 2001

- (They canceled that warning in 2004)
- Web servers are still large and complex. . .

Important Web Server Features

- Access control
- User behavior
- CGI (Apache) or ASP (IIS) scripts (often via special scripting languages)
- Plug-ins
- Back-end databases
- Cryptography
- (Does this remind you of an operating system?)

Access Control

- Many different forms
- Many different types of authentication
- Many interactions

Document Root

- All files served must reside under a certain directory
- Watch out for “`..`” in URLs (gee, we’ve seen that before)
- For convenience, some “subtrees” can reside somewhere else:

```
ScriptAlias /mailman/ "/usr/pkg/lib/mailman/cgi-bin/"
Alias /pipermail/ "/var/db/mailman/archives/public/"
Alias /mailman-icons/ "/usr/pkg/lib/mailman/icons/"
```

- If the Web server supports “virtual hosting”, each “host” gets its own subtree

  ☞ With virtual hosting, a single machine and web server can offer up several different web sites

Explicit Access Control

- Access control lists settable by the webmaster for any directory tree
- Passwords or certificates can be configured as well
- Permission can be granted or withheld based on client IP address
- If a directory has no **`index.html`** file, should the web server just list its contents?
- Applications can do their own authentication and access control
- All of these interact; combinations can be used

A Sample Configuration

Here is a **.htaccess** file for a directory:

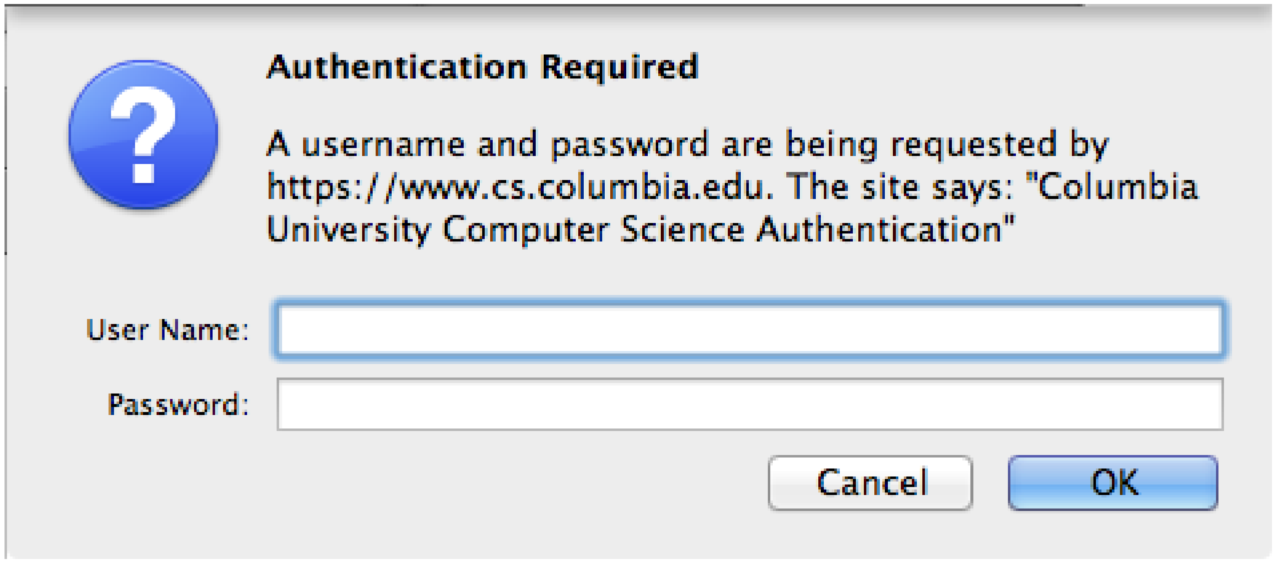
```
<Files *>
    AuthUserFile /home/smb/pwdir/.htpasswd
    AuthGroupFile /dev/null
    AuthName "File Access"
    AuthType Basic
    Require valid-user
</Files>
```

The string **File Access** is displayed to the user. Logins and passwords are stored in **/home/smb/pwdir/.htpasswd**.

Web Authentication

A web password file:

```
user1:eO3rzWPNjjZFo
user2:CqkaeLJSVcRpI
```



Authentication Required

A username and password are being requested by https://www.cs.columbia.edu. The site says: "Columbia University Computer Science Authentication"

That’s Rarely Used—Why?

- No site-specific display
- No error recovery, e.g., a link for “I forgot my password”
- Too restrictive—no good option for partial display, e.g., of a news article
- A simple linear file doesn’t scale up very well

Which is Implemented Better?

- Who implemented the application’s password logic?
- Did they do it correctly?
- What about related information, such as email address?

Operating System Access Control

- Can the web server benefit from OS access control?
- What UIDs does the server run under?
- What permissions can/should be used for the files being served?

“Privileged” Ports versus Security

- Most Unix systems reserve ports < 1024 for root
- Web servers listen on port 80; therefore, they have to run as root
- Do we really want such a large, complex program running as root? Not if we can help it…

Shedding Privileges

- Apache starts as root
- Note: it is *not* setuid; it must be invoked by root. (Why is that the right choice?)
- It opens the socket and some log files, then forks and sheds privileges
- Serving web pages is done as non-privileged user “www”

File Permissions

- If the web server isn't root, it can't open protected files
- All pages served must be readable by the web server, its group, or "other"
- *Don't* make them owned by www; that way, a compromised web server can't overwrite them
- In other words, the web server itself has as few privileges as possible

Design Philosphy

- Use the OS to protect the system against the web server
- Assume the web server can enforce its own access control mechanisms

Can We Lock Things Away?

- We don’t want content owned by user www
- We could try putting user content under some lock directory, with a setuid helper program to let people publish web pages
- We can protect a few resources by using group read permissions—make the content group-readable but not other-readable, and let the web server run with several groups’ permissions

☞ Unfortunately, Apache doesn’t seem to support that

- There’s still a problem with scripts

Scripts

- Retrieving static files is ok, but scripts make life interesting
- Scripts are *programs*
- Each script is a separate network service
- Is each one correct?
- From the Apache Security Guide: “Always remember that you must trust the writers of the CGI script/programs or your ability to spot potential security holes in CGI, whether they were deliberate or accidental.”

Script Permissions

- In general, all scripts run with the same permissions
- This uid shouldn't own any files; see above for OS access controls
- Scripts can interfere with each other: "All the CGI scripts will run as the same user, so they have potential to conflict (accidentally or deliberately) with other scripts *e.g.* User A hates User B, so he writes a script to trash User B's CGI database."

Design Philosophy

- Use Apache access controls to isolate the dangerous stuff
- Use OS permission mechanisms—as invoked by Apache—to isolate CGI scripts from each other
- Separation isn't as strong as for the base Apache system, because of the overwrite scenario

Plug-Ins

- Scripting languages are often available as Apache *modules*
- This means that they run as part of the Apache process
- Modules are an efficiency hack: save the expense of fork()/exec()
- Modules run with the full permissions (and address space) of Apache
- Very dangerous!

PHP's Safe_Mode

- PHP, if `safe_mode` is turned on, restricts scripts to opening files owned by the script owner
- This in an application—PHP—enforcing something resembling OS permissions
- Did they get it right? Are there race condition attacks?
- Still does not protect against attacks from on-machine

Other Script Languages

- Java can be configured to be secure
- To my knowleged, neither Perl nor TCL—two other languages that can run as plug-ins—have such a feature
- There is *no* way to confine C or C++

Invoking Scripts

- Scripts are often invoked with client-supplied parameters
- Magic shell characters aren't as big a problem for parameters, because they're passed to scripts via an environment variable, not on the command line
- But—what about magic shell characters *in the script name*?
- Example: `http://www.example.com/cgi-bin/`rm-rf/``
- After all, if it's in **`cgi-bin`** it's executable…

Administrator Strategy

- Use a complex local scheme
- Provide a setuid program to copy user content to the web server
- Do not allow user programs to execute on that server
- Permit only “safe” scripting languages with their own access control
- Do *not* permit execution of C or C++ programs!
- Use web server access controls to restrict other access

Uploading Files

- If all scripts run with the same permissions, and if local users have read-access to user content, how can you do safe upload?
- Example: suppose I wanted to write a PHP script for homework submission
- Create an upload directory owned by me that is mode **`rwx,-wx,-wx`**: anyone can write to it or trace a search path, but not read it
- Use a true-random string for part or all of the filename
- For instance, store smb2132.0.tar as 158cb5864f2c7662b-smb2132.0.tar (generated from **`/dev/urandom`**)
- No one will guess that to retrieve it or overwrite it
- Note: I'll be able to list the directory and read the files (if I set the file permissions correctly), but I won't own the files; www will

Back-End Databases

- Scripts are often front-ends to databases
- Does the database have its own access control? Where is the password stored?
- How does the script supply the password?
- Remember that any file on the server is readable by all other users or script writers. . .

Protecting the Database

- If the web server is hacked, how do you protect the database?
- One answer: put it on a separate machine
- But—a hacked web server can issue bogus queries
- Why move it?
- Restrictions and logging

Restrictions

- The web server does not need full access to the database
- Example: it doesn't need to be able to read credit card numbers, only write them
- (It may need to read a "display" version, e.g., the last four digits—but that can be a separate column in the database)
- More details on this later in the term

Logging

- If the web server is compromised, its logs can't be trusted
- If the database server is on another machine, it can log all changes made to it
- Again, more later

Ignoring the Database Server

- A database is just a (very large) file with a particular internal structure
- A sophisticated attacker who has full privileges can read that file *without* going through the server and its permissions and logs
- Putting the database elsewhere protects against that

Hacking the Database Anyway

- This is good, but. . .
- One common attack is SQL injection—and that's an attack on the database directly, not the web server
- Putting it on a different computer doesn't help

Design Issues

- Neither the OS nor Apache's access controls can help us much
- We have to rely on the script language's access controls
- Even that may not protect us from subverted scripts

Cryptography

- TLS encryption used for most e-commerce
- TLS uses hybrid public key/symmetric crypto
- Where does the web server get is private key?
- Again, how do we store a key on a computer?

Key Storage

- Ideally, it's stored in encrypted form, or in some tamper-resistant device
- We can't store it encrypted—how is the decryption key supplied at Apache startup?
- A few large sites use TLS front-end/load-balancer devices, but these aren't common
- We must store the key in the clear, on the web server machine

Protecting the Key

- Of course, it's stored mode `r--,---,---`
- It's also owned by root, and read in at startup before changing UIDs
- Why? To provide maximum OS protection against subversion

Authentication

- Two basic types: passwords and client-side certificates
- Passwords can be for the built-in Web browser authentication or for application-specific authentication
- Passwords should *never* be used without encrypting the network connection
- Client-side certificates are more secure, but they're rare
- They're also less convenient: how does the user carry around a private key to multiple machines?
- Ultimately, the client's identity feeds into Apache's access control mechanisms

Phishing

- Trick people into sending their passwords to the wrong site
- People *could* check the site's certificate—but very few people do that
- Legitimate site should *never* email clickable links—but many do
- What good is a strong web password?

Password Managers

- Store passwords for many web sites
- Protect them with a master password
- The code matches the web site name against the stored password; it won't be fooled by phishing email

Lessons

- Web servers are *very* hard to secure
- We need all of our tools: OS permissions, application ACLs, script language security, cryptography, and more
- There are often residual issues even then