SOPHOS

ApacheCon

Apache As A Malware-Scanning Proxy

Jeremy Stashewsky, Sophos Plc.

http://www.sophos.com/

jeremys@ca.sophos.com

US 2006

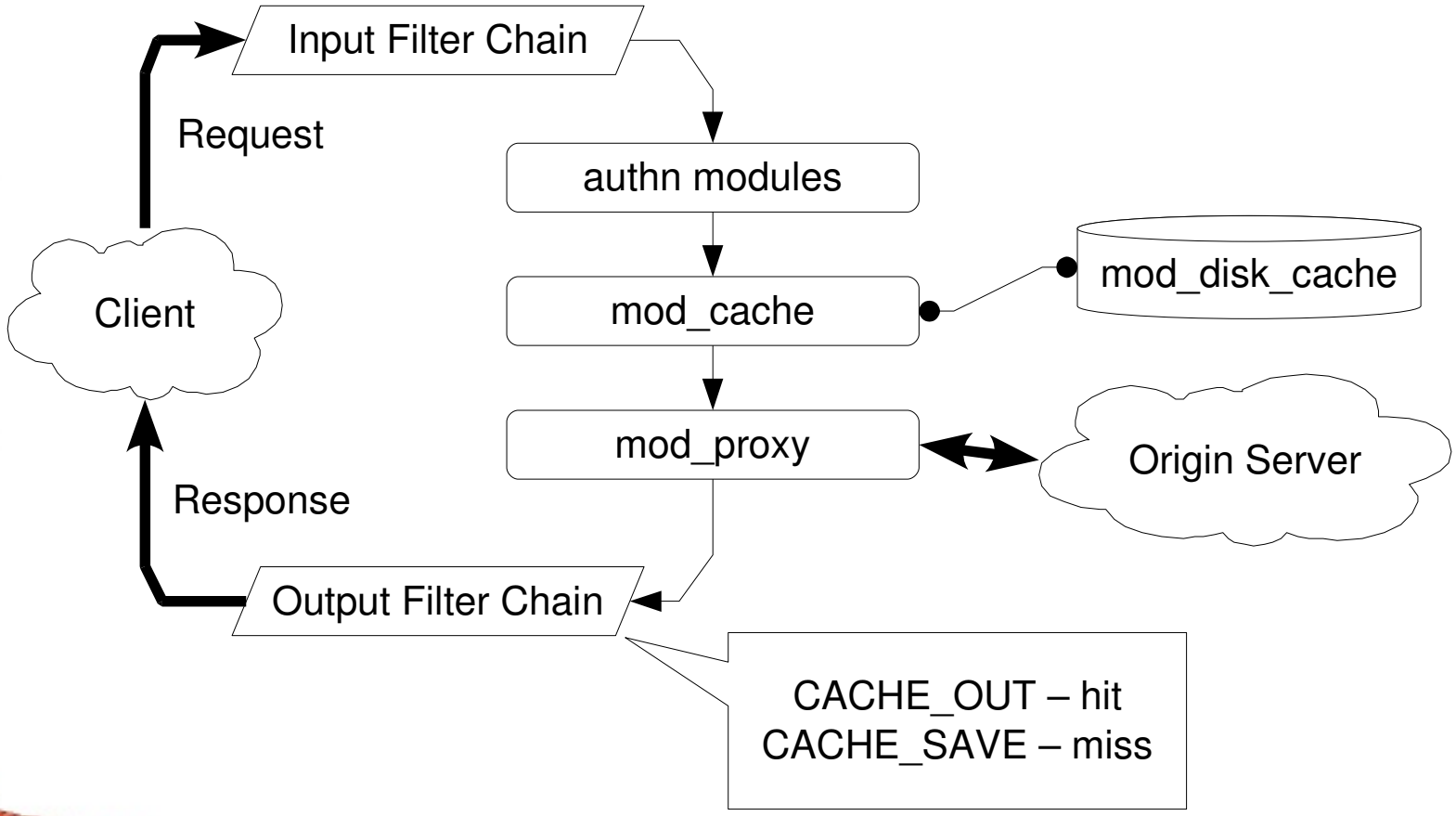
Overview

- The case: building an appliance product
- Apache HTTPD proxy architecture
- Malware scanning: challenges and solutions
- Where do we go from here: improving Apache.

Apache as a Proxy

- Solid reverse **and** forward proxy
- Decent performance
- Variety of AAA modules
- 2.2.x: cache modules now stable

Basic Apache Architecture



Basic Scanning

- New output filter captures bytes
- Spools to temporary storage
  - If not cached
- Scans with an external program
- Safe? Let it through
- Unsafe? Show a block page

Problems with Basic Scanning

- Launches an external program
- Stopping-up latency
  - Client time-outs
  - Indefinite content-length
  - Unhappy users

Alternatives to Launching A Scanner

- Worker MPM?
  - Load engine in child process, scanner threads
  - Bad: thread crash kills process
- ICAP (RFC 3507) scanner?
- Custom external scanner
  - Unix/TCP daemon accepts scan commands

Custom External Scanner

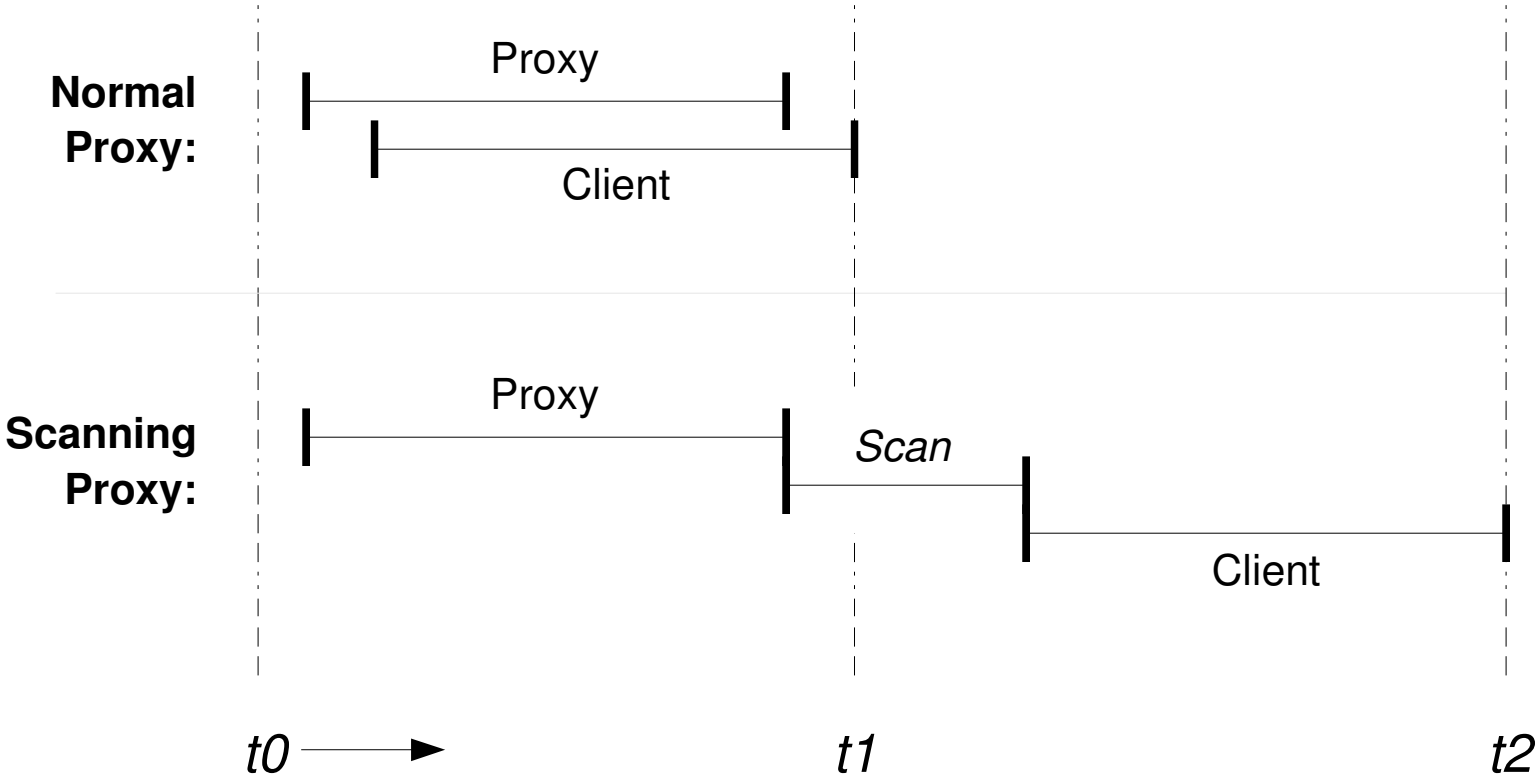
- Safety from problem files
- Local IPC traffic
  - No body transfer overhead
- Global fairness
- Wrapping a Library
  - Apache w/ protocol filters
  - Stand-alone daemon with APR

Stream Scanning?

- Interesting stuff at EOF
  - Viruses often append themselves
  - Many file formats put “Index” at EOF
- Just don't send the bad part?
  - Interpreted
  - Auto-repair
- Disinfection

SOPHOS

"Stopping-up" Effect



Time-Outs

- Client: 60-300 seconds
  - Highly browser/user-agent dependent
- Users: 4-7 seconds
  - Depends on content; HTML is a bit longer
  - Speed of Internet pipe important

Keep the Client Happy

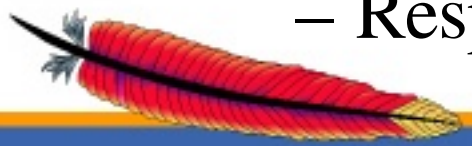
- Trickle “H... T... T... P... /... 1... ”
  - Some Clients more willing to wait if data flowing
  - Tricky: protocol filter
- Trickle headers
- Pause before body
- Trickle body? Dangerous

Keep the User Happy

- "Patience Page"
- Download, scan and store
- Provide link to stored file
  - E-mail notification?

Patience Page Problems

- Right-click, “Save As...”
  - User: “Corrupt files! Argh!!”
  - IE: no Referer header
  - All: no Referer header when entering URL in Address bar
  - No good workaround
- Non-visual Clients (e.g. wget)
  - Response codes help



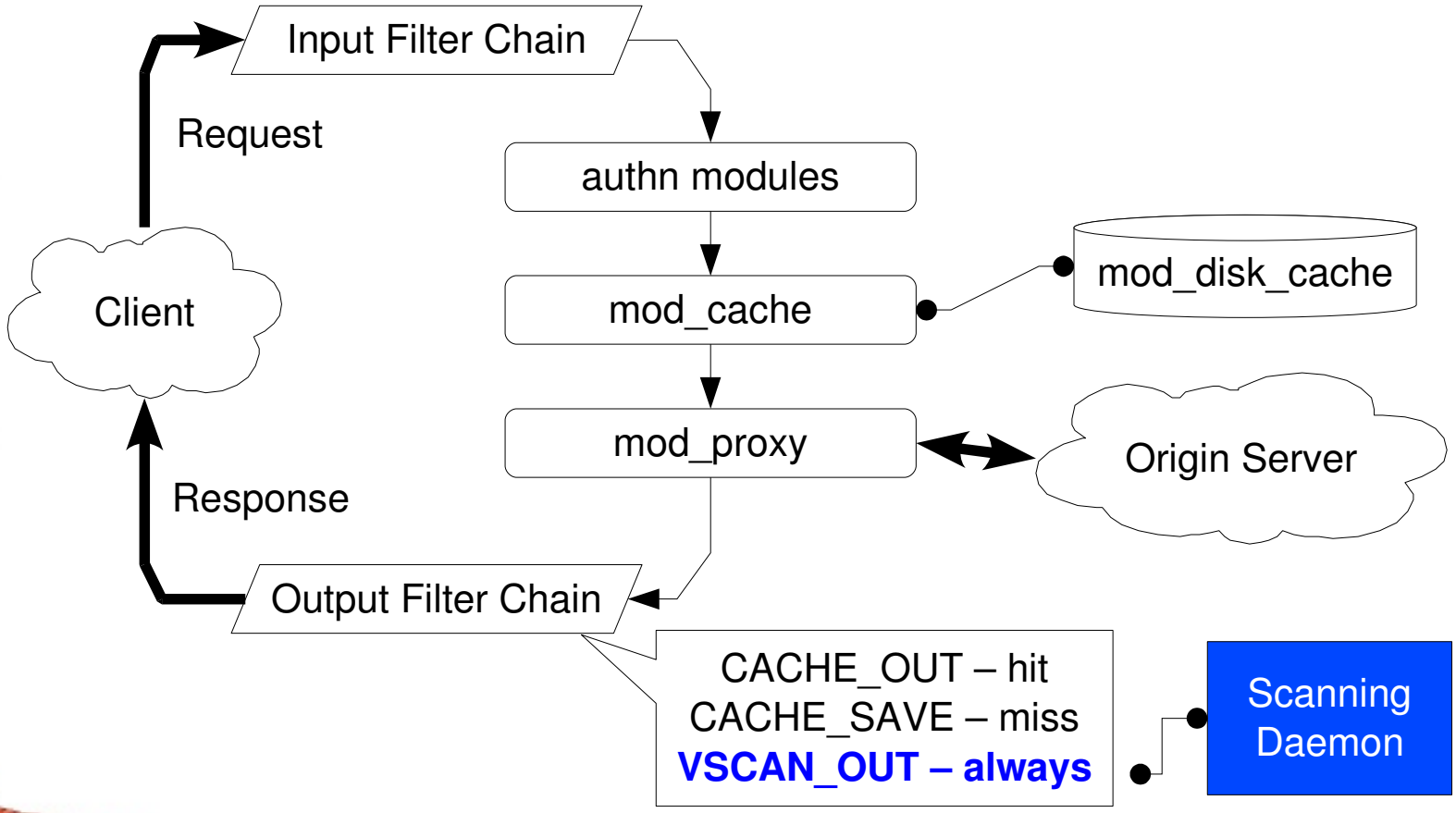
A Patience Page in Apache

- Send 403, some content
- Keep both Client and Origin sockets open
- JavaScript sent to Client
- Provide download link when done!
- Maintain caching? Make an output filter after CACHE_OUT.

Advanced Scanning

- "Safe" file type bypass
  - Can also increase TPS at cost of security
- Stream scanning for media
  - Detect exploits, embedded scripts
  - Users can "tolerate" streaming media stopping
- Incremental scanning
  - Archives/containers

Architecture Recap



Moving Beyond Scanning

- Why waste time scanning if you know it's infected?
- Interoperability Bugs?

Add URI-based Policy

- Blocking an unsafe URI
  - Save CPU -> more TPS
  - Combat 0-day & suspected Malwares
- Bypass local or trusted sites
  - Workarounds
  - Improve Performance
- Apache “auth checker” module

A First Step: URI Text File

- Linear search
  - Load into `apr_hash`?
- Text is easy to patch
- Doesn't scale well past a few thousand entries
- Key Problem: URIs have structure
  - string searching doesn't map well

URIs: Relational Database

- Good idea if a central database is required
- Findings:
  - Good: apr-util DBI
  - Good: Reasonable update speed
  - Bad: Slow lookup time hurts TPS
  - Bad: Heavyweight

URIs: Simple Database

- Data is hierarchical -> search trie
- DBM files
  - apr_dbm in apr-util
- Pre-Compiled Hash
- Findings:
  - DBM: faster than relational, small updates
  - Hash: faster still, but big/slow updates

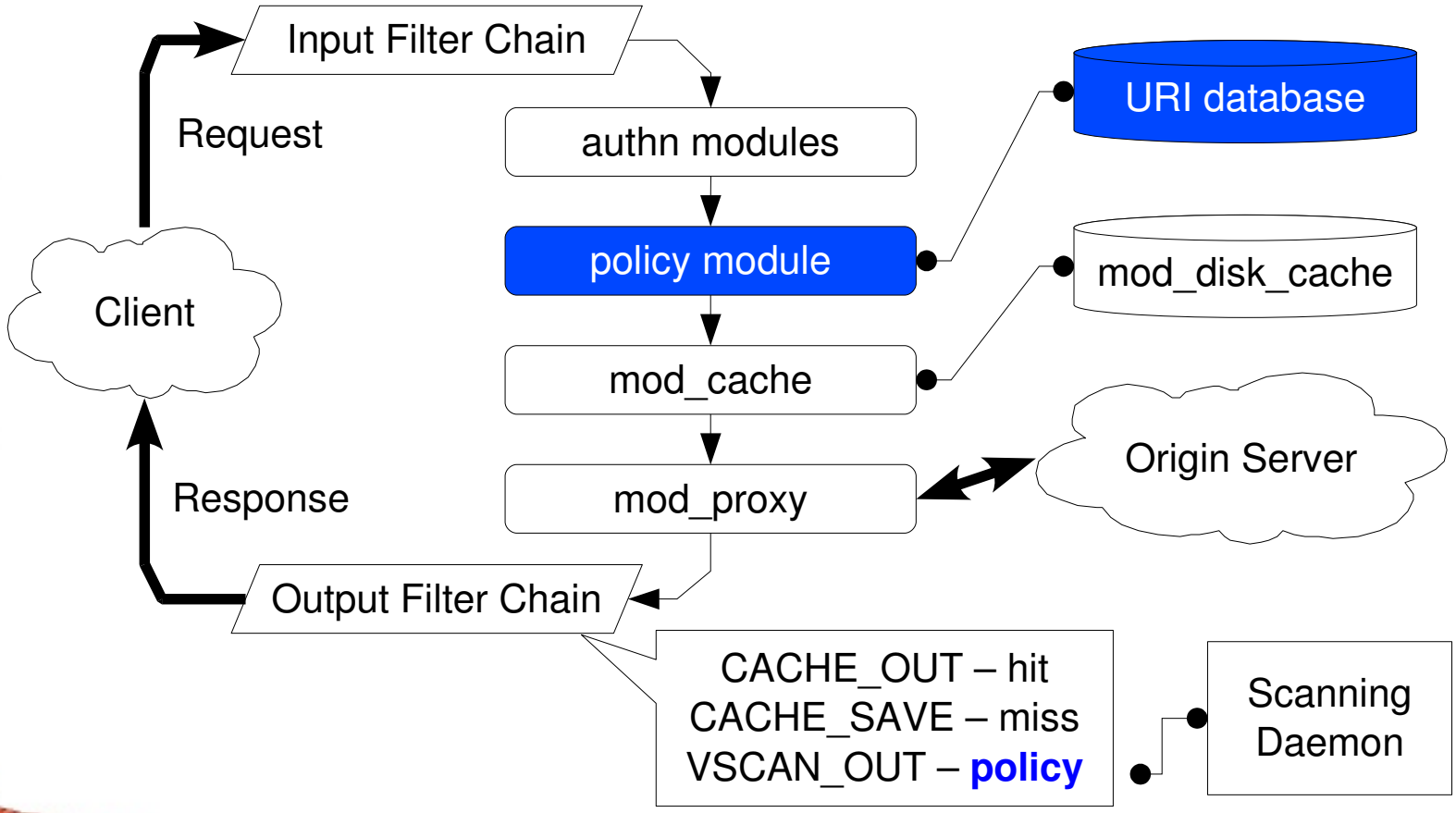
URIs: Domain Hashing

- `bucket = substr(hash(domain),...)`
  - Similar to mod_disk_cache's implementation
- Splits up database
- 12 bits are sufficient for 10^6 domains
  - 4096 buckets

Hash Domains & Simple DBs

- Fast; kept under O(log(n))
- Bucketing keeps indexes small
- Binary-diff for distribution
- Scales to at least 10^6 entries from experience

Architecture Recap



SOPHOS

User Interface

- Apache, mod_ssl, mod_php
  - Administrative and End-user UI
- Block and Error pages
  - Internal redirect to PHP
- Patience Page
  - PHP generates the content to disk one-time
  - Make file `apr_bucket`

Where do we go from here?

- Transparent Proxy
- HTTPS Scanning
- mod_cache, mod_disk_cache improvements
- mod_proxy improvements

Transparent Proxy

- OS redirects traffic
- Key: Provide missing info to apache
  - Fixup-phase module?
- Hostname?
  - Reverse-lookup: unreliable
  - HTTP/1.1 “Host” header
  - Resolve, check against destination IP

HTTP over TLS/SSL

- Certificate checking
  - List of trusted CAs
- Dynamic Cert generation
  - Keep Subject, replace Issuer, sign
  - User must trust Issuer
- Transparent?
  - Grab cert to get hostname!

HTTPS: Social Issues

- HTTPS sites can get hacked
- Have cert != legitimate
- Don't trust proxy to scan?
  - Policy bypass for individuals
- Don't trust admin?
  - Access your bank from home

Improving mod_cache

- Disk cache expiry: needs improvement
  - Disk cache can grow too large
- Cacheability correctness bugs
  - Apache-Test suite would be handy

Improving mod_cache

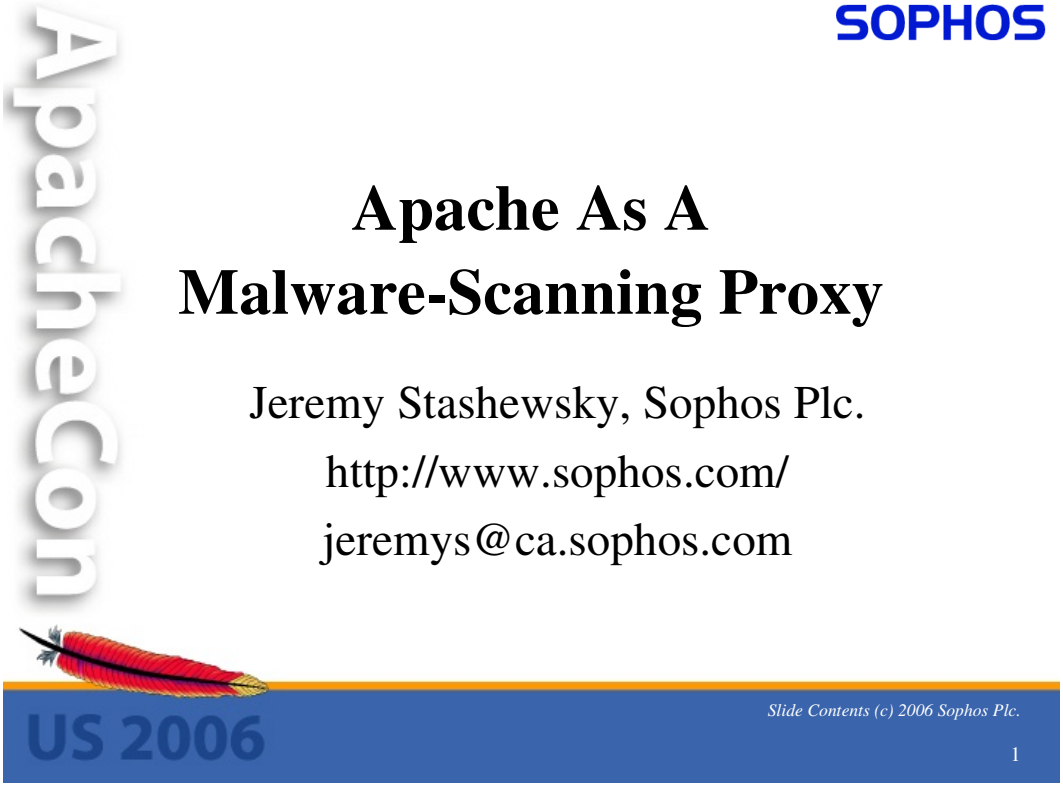
- Store meta-data with objects
  - Expiry meta-data
  - Scan caching & revalidation
- Multi-layer cache providers
  - Scan revalidation as a top-level cache provider
  - Performance

Improving mod_proxy

- Persistent connections!
- Limiting connections to an Origin
- Overall throughput
  - Maybe best handled by OS' QoS

Conclusions

- Apache: Not just a good web server!
  - Clear, modular design
- Key Challenges Covered:
  - Stopping-up
  - Keeping the User Happy
  - URI-based Policy
  - Apache improvements



A bit of background:

Sophos develops integrated threat management solutions to protect against malware, spam, and policy abuse. I'm a technical lead developer on a project to build a malware-scanning web gateway appliance (using Apache).

This presentation is about some of the core challenges we faced and solutions we tried when building the appliance.

SOPHOS

Overview

- The case: building an appliance product
- Apache HTTPD proxy architecture
- Malware scanning: challenges and solutions
- Where do we go from here: improving Apache.

ApacheCon US 2006

Our Rough Appliance specs:

- 2 to 4 GB Ram
- 1 CPU (possibly dual-core) ~ 3GHz
- small SATA disks

Had to choose between Linux and FreeBSD.

Decided to use Linux as it showed better performance with mod_disk_cache.

SOPHOS

Apache as a Proxy

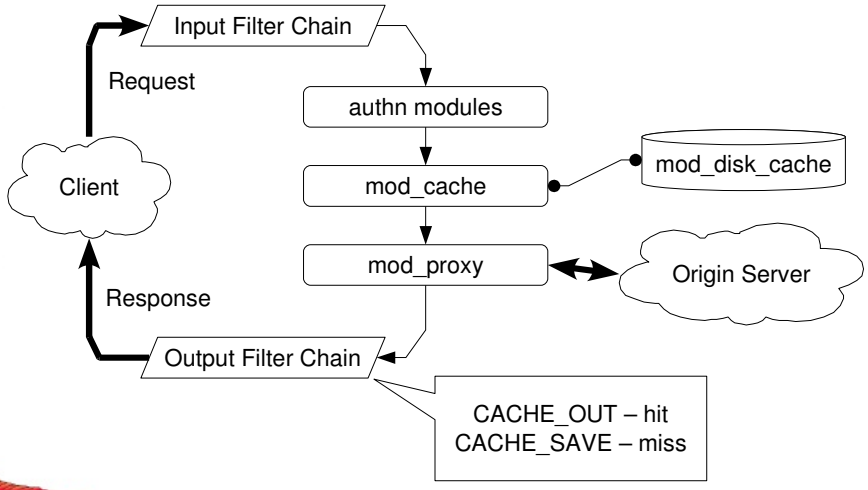
- Solid reverse **and** forward proxy
- Decent performance
- Variety of AAA modules
- 2.2.x: cache modules now stable

ApacheCon US 2006

The availability of AAA modules and the modularity of Apache 2's design is what attracted us to use it in our product.

Secondarily, we've also got a tradition of using Open Source Software wherever possible – an influence from when the Vancouver office was ActiveState.

Basic Apache Architecture

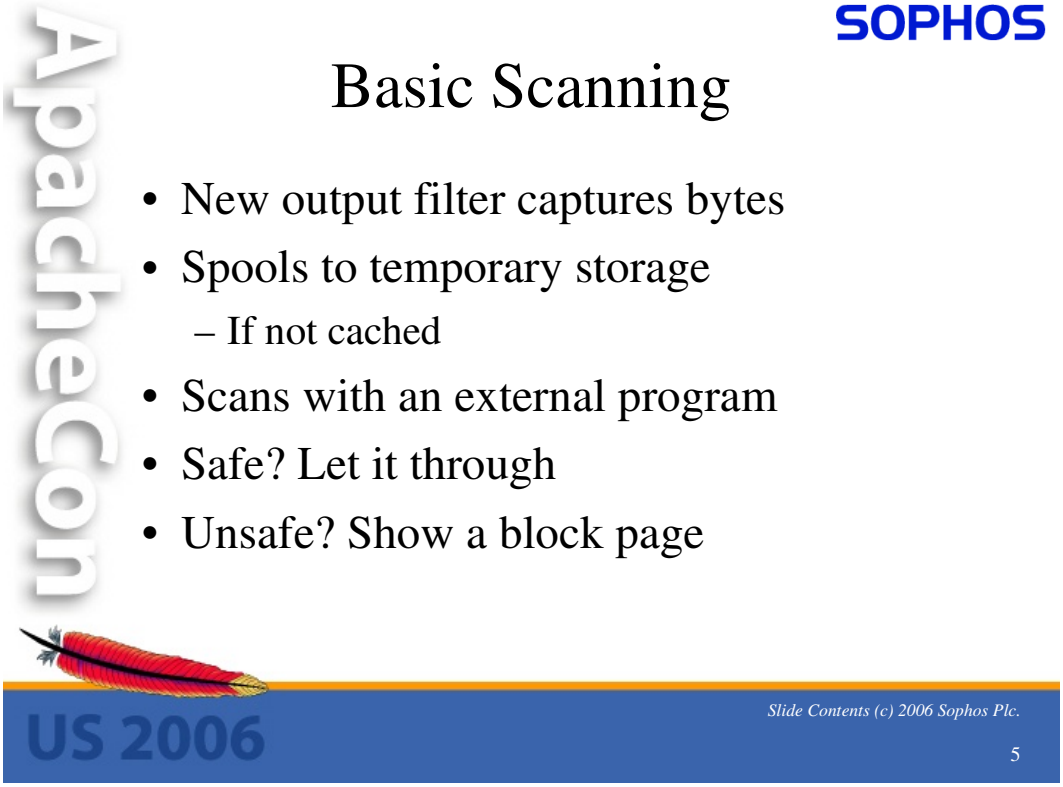


Thick lines: HTTP traffic

Thin lines: Apache Communication

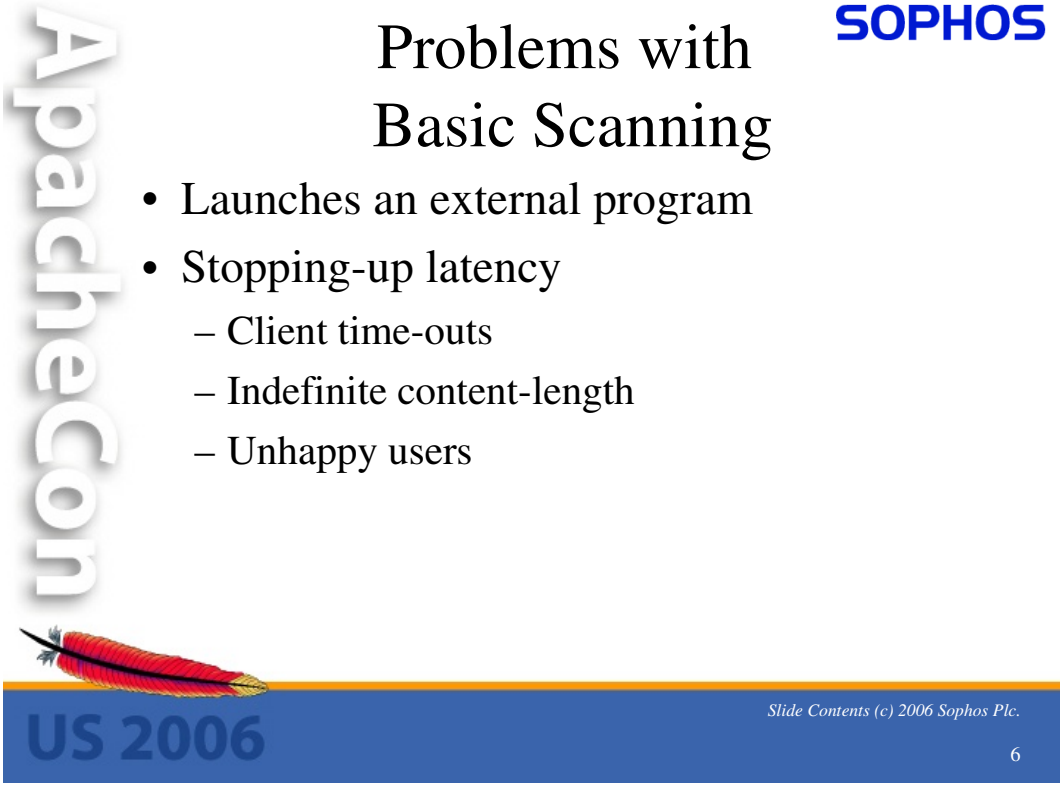
mod_cache can bypass the proxy if it's provider (mod_disk_cache) has the requested object. CACHE_OUT is enabled in this case.

On a cache miss, and if mod_cache determines the object is cacheable, CACHE_SAVE is enabled.



Bucket brigades – facilitates stream scanning (to be discussed later)

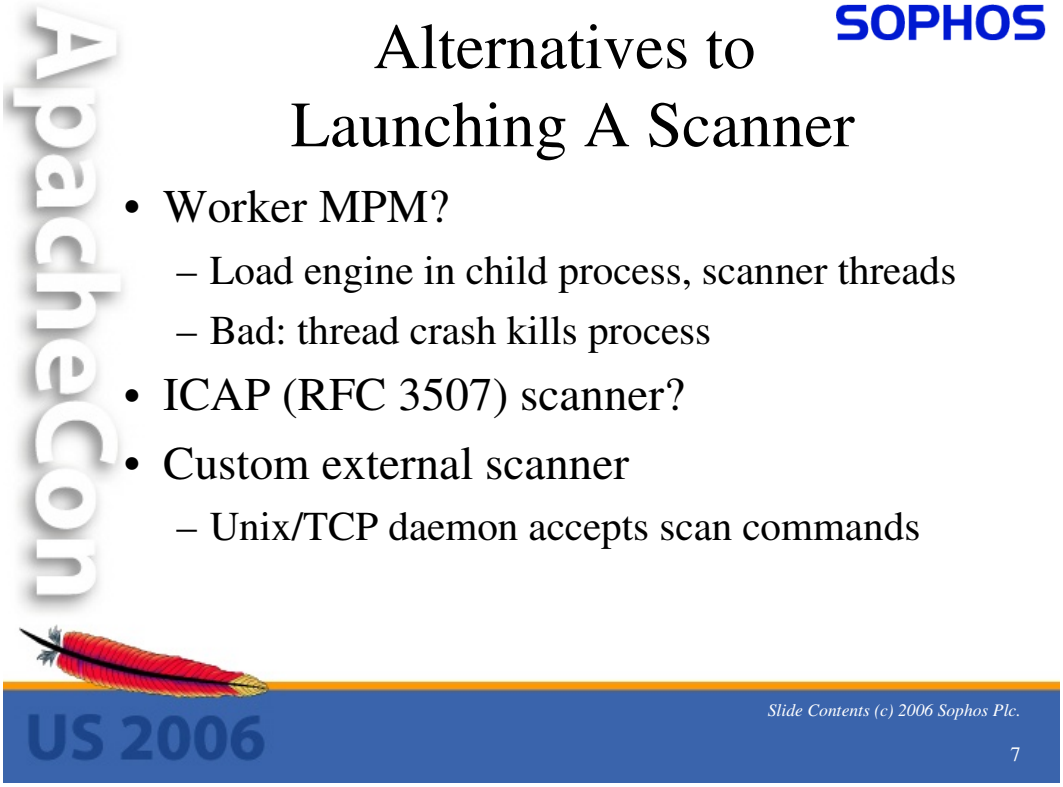
If mod_disk_cache is caching the content, we can use that to scan rather than duplicating a spooled copy.



The problems have one thing in common: bad performance.

Launching a process is slow because of:

- OS fork/exec overhead
- Loading malware signatures, etc. every time is wasteful



Internet Content Adaptation Protocol

Has provisions for malware scanning, like 4kB previews, streaming, etc.

Writing an ICAP server? An Apache **or** APR implementation should be straightforward.

Custom External Scanner

- Safety from problem files
- Local IPC traffic
  - No body transfer overhead
- Global fairness
- Wrapping a Library
  - Apache w/ protocol filters
  - Stand-alone daemon with APR

Apache wrapper: less implementation to worry about

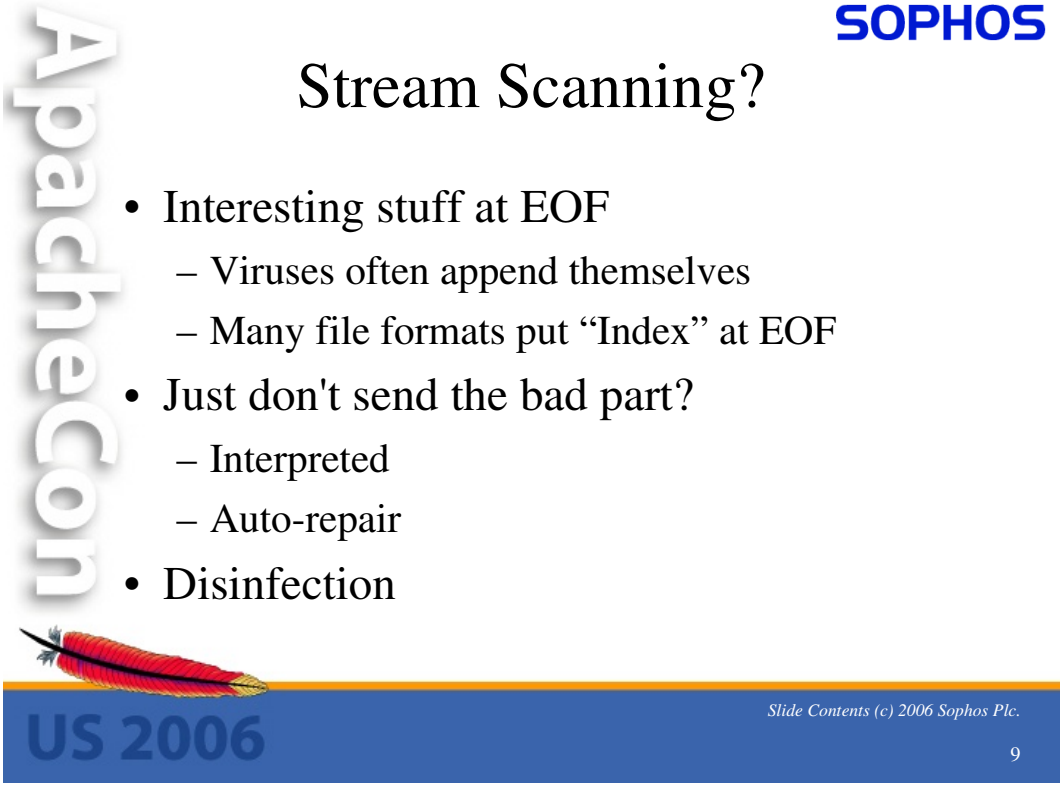
Standalone:

Can make better service guarantees, such as timeouts and Advanced queuing

Implementing:

- APR pools, threads, etc. for portability

- apr_poll and sockets – scalable & event-driven



Stop-up

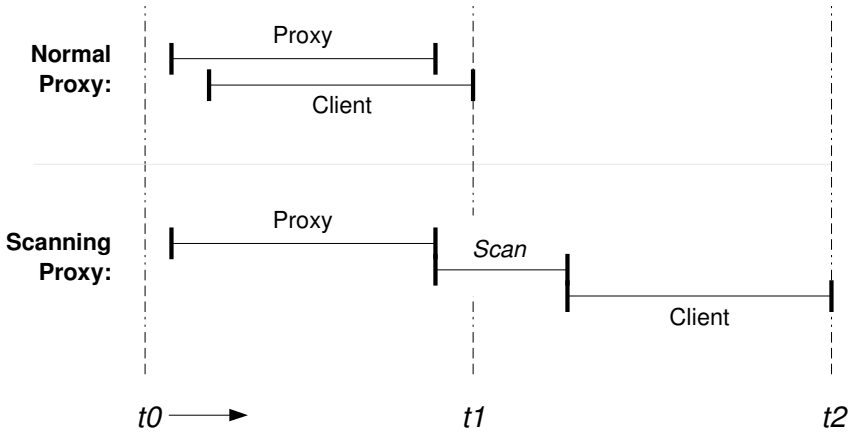
Some file formats can still execute the malware content with just part of the file.

e.g. Word processor documents and embedded Macros. Interpreters: partial source code (like a shell or Perl script) can still execute and cause damage.

RAR files: auto-repair capabilities. We can't guarantee that what we've sent so far can't be repaired, extracted, and executed.

Disinfection typically requires the whole file to repair; can't stream this. Also, not 100% safe for the same reasons we can't detect all Malwares.

"Stopping-up" Effect



Proxy bar: bytes received

Client bar: perceived download

Total: time-to-completion

- perceptual? start of bars.

Small Files: add download overhead, negligible scan overhead

Large Files: download overhead is the biggest contribution to perceived latency

Archives and complex file types take a long time to scan too

Infinite files! Need specialized scanning treatment.

SOPHOS

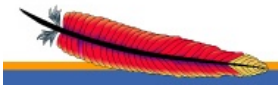
Time-Outs

- Client: 60-300 seconds
  - Highly browser/user-agent dependent
- Users: 4-7 seconds
  - Depends on content; HTML is a bit longer
  - Speed of Internet pipe important

Side-effect of “Stopping-up”: Time-outs.

Keep the Client Happy

- Trickle “H... T... T... P... /... 1... ”
  - Some Clients more willing to wait if data flowing
  - Tricky: protocol filter
- Trickle headers
- Pause before body
- Trickle body? Dangerous



Interesting ideas, but didn't have time to implement this.

Actual effectiveness of trickling is unproven... likely Client dependent and thus a bit fragile.

SOPHOS

Keep the User Happy

- "Patience Page"
- Download, scan and store
- Provide link to stored file
  - E-mail notification?

"Please wait while your file is being downloaded".

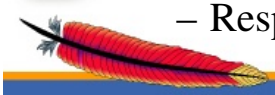
"Please wait while we scan your file"

Use standard HTTP result codes to give non-visual user-agents a hint (e.g. 403 forbidden)

Good idea to use an "inclusion" list of User-Agents that are known to be visual (e.g. "Mozilla-compatible" in U-A header)

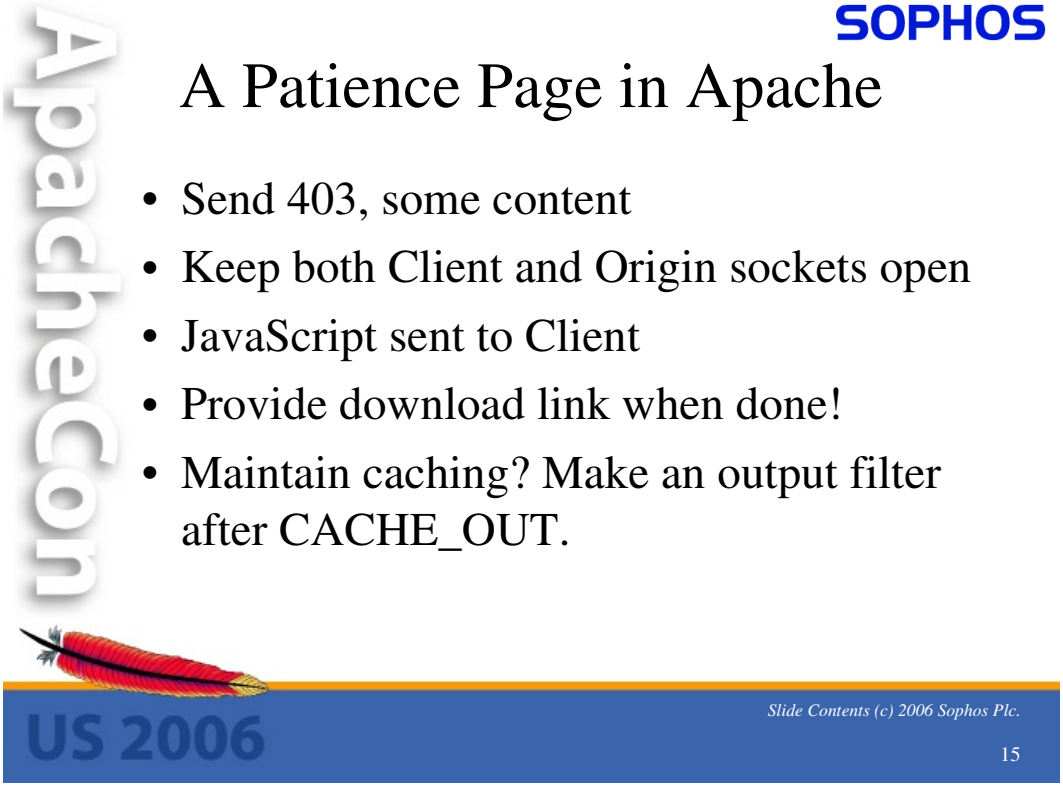
Patience Page Problems

- Right-click, “Save As...”
  - User: “Corrupt files! Argh!!”
  - IE: no Referer header
  - All: no Referer header when entering URL in Address bar
  - No good workaround
- Non-visual Clients (e.g. wget)
  - Response codes help



The right-click problem is **really** frustrating!

Wget at least seems to Do The Right Thing when presented a 403 Forbidden.



VirtualHost serves saved content.

We used UUIDs to save the content.

Small CGI script to serve the content and delete it.

Optimization: hard-link files from cache and delete hard-link after user downloads?

Uses “Content-Disposition: attachment; filename=blah” to trigger download.

- non-standard header, but IE introduced it and most browsers “get-it”

Advanced Scanning

- "Safe" file type bypass
  - Can also increase TPS at cost of security
- Stream scanning for media
  - Detect exploits, embedded scripts
  - Users can "tolerate" streaming media stopping
- Incremental scanning
  - Archives/containers

Incremental:

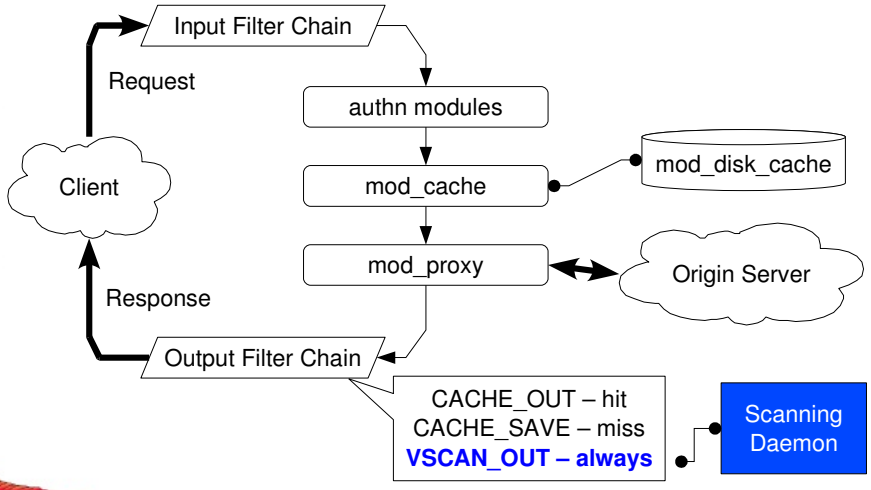
- Validate archive header is valid (known-exploit free)
- Send archive header to client
- For each file, send the corresponding archive bytes to the client.
- If an infected file is detected, terminate the connection.

Pro: less latency

Con: less user-friendly

Similar approach for media scanning.

Architecture Recap



"VSCAN_OUT" filter captures bytes

commands sent to scanning daemon

- file-type/stream check
- incremental scanning
- full file scan w/ patience page

Advantage of VSCAN_OUT always on: can catch false-negatives and false-positives

Disadvantage: CPU – scans **everything**

SOPHOS

Moving Beyond Scanning

- Why waste time scanning if you know it's infected?
- Interoperability Bugs?

Reputation of sites:

Feed back URIs of infected files to a virus lab?

URI listed in an outbreak bulletin?

Interoperability:

Product update sites, e.g MS Windows Update

“Non-compliant” web servers.

Etc.

Add URI-based Policy

- Blocking an unsafe URI
  - Save CPU -> more TPS
  - Combat 0-day & suspected Malwares
- Bypass local or trusted sites
  - Workarounds
  - Improve Performance
- Apache “auth checker” module

More efficient to straight-up block a site rather than scan it's content.

Usability improvements?

- Soft block -> user could “click through” a block page
- URI publisher “suggests”, administrator “implements”
  - maybe not too scalable

Many vendors provide site categorization lists, block lists.

SOPHOS

A First Step: URI Text File

- Linear search
  - Load into `apr_hash`?
- Text is easy to patch
- Doesn't scale well past a few thousand entries
- Key Problem: URIs have structure
  - string searching doesn't map well

apr_hash will speed-up exact matching on parts like the domain name.

Scaling issues are related to load/re-load times with hashes and the nature of the data.

Search time becomes an issue when linear searching a plain list. You could binary-search on domain name, but it gets harder for other parts of the URI, especially when you want to do prefix or suffix matching.

Hooks auth_checker.

URIs: Relational Database

- Good idea if a central database is required
- Findings:
  - Good: apr-util DBI
  - Good: Reasonable update speed
  - Bad: Slow lookup time hurts TPS
  - Bad: Heavyweight

Not really viable for a small Appliance such as the one we're making, more relevant for large or clustered appliances, perhaps.

Bad mapping: the problem domain doesn't lend well to normalization.

Prepared statements can reduce the SQL overhead.

Network/remote access overhead is costly.

SQLite (http://sqlite.org) was explored. Performance was good, but the DB schema (indexes and tables) was fairly complex.

URIs: Simple Database

- Data is hierarchical -> search trie
- DBM files
  - apr_dbm in apr-util
- Pre-Compiled Hash
- Findings:
  - DBM: faster than relational, small updates
  - Hash: faster still, but big/slow updates

Good fit for the small appliance since fast lookup = less CPU

Hash: O(1) look-ups on average

Trie: also O(1) on average for fixed-depth trees (there's only so-many parts of a URI)

Alternatives:

- CDB
- BerkeleyDB

URIs: Domain Hashing

- `bucket = substr(hash(domain),...)`
  - Similar to mod_disk_cache's implementation
- Splits up database
- 12 bits are sufficient for 10^6 domains
  - 4096 buckets

Familiar with mod_disk_cache? It uses directory buckets to keep directories from filling up. The approach described here is similar, but maybe doesn't have to use a tree of directories.

Bucketing can apply to either the Relational or Simple database approaches.

Relational: split up tables for smaller indexes. Might get a small perf improvement, definitely helps with insert/updates.

Simple: smaller files with smaller and faster indexes.

Experiments showed that as little as 12 bits from a good hash (e.g. MD5) are enough to get a good spread.

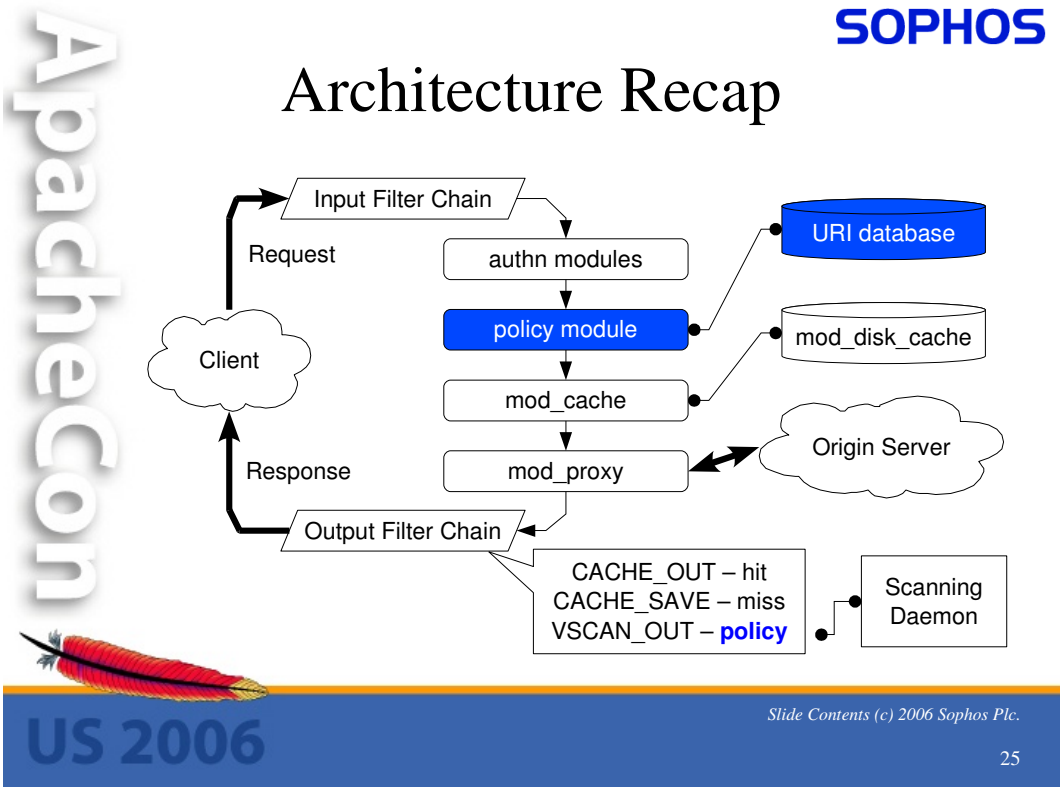
Hash Domains & Simple DBs

- Fast; kept under O(log(n))
- Bucketing keeps indexes small
- Binary-diff for distribution
- Scales to at least 10^6 entries from experience

n = average size of a bucket

1.5 x 10^6 entries in this form was roughly 30MB compressed. Used our own, proprietary database-file format (similar to CDB, uses Hash tables).

binary diffs done using using bsdiff (http://www.daemonology.net/bsdiff/) were consistently under 100KB every 10-15 minutes (5-100 URI changes).



The VSCAN_OUT filter is now driven by our policy module.

The policy module can also take into account more than the destination URI. policy module hooks “auth_checker”.

Options:

- source IP
- username (From apache AAA modules!) or group
- Time of Day + category
  - e.g. no news sites until lunch time

FWIW, We used mod_ntlm_winbind and winbindd from the Samba “lorikeet” and core projects (respectively). http://download.samba.org/ftp/unpacked/lorikeet/mod_ntlm

or http://tinyurl.com/mxt7h

User Interface

- Apache, mod_ssl, mod_php
  - Administrative and End-user UI
- Block and Error pages
  - Internal redirect to PHP
- Patience Page
  - PHP generates the content to disk one-time
  - Make file `apr_bucket`

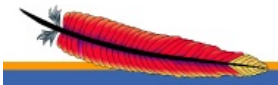
Or CGI instead of PHP, perhaps?

Rather than a PHP VirtualHost, you could use a reverse proxy to back onto a separate Apache process or server.

We use Selenium (http://www.openqa.org/selenium/) to auto-test our UI. Selenium uses real web browsers to do automated unit and system testing.

Where do we go from here?

- Transparent Proxy
- HTTPS Scanning
- mod_cache, mod_disk_cache improvements
- mod_proxy improvements



This section of the presentation is about things we've worked on, but haven't implemented solutions yet (or, we're looking for better ones at the time of writing).

Basically, some things to think about in the Apache HTTPD community.

SOPHOS

Transparent Proxy

- OS redirects traffic
- Key: Provide missing info to apache
  - Fixup-phase module?
- Hostname?
  - Reverse-lookup: unreliable
  - HTTP/1.1 “Host” header
  - Resolve, check against destination IP

Primary deployment of our product is in “opaque” mode, but added transparency to facilitate evaluations. Sophos should be able contribute back some patches here

Linux: IPTables and BRTables for traffic redirection.

HTTP over TLS/SSL

- Certificate checking
  - List of trusted CAs
- Dynamic Cert generation
  - Keep Subject, replace Issuer, sign
  - User must trust Issuer
- Transparent?
  - Grab cert to get hostname!

CA = Certificate Authority.

Cert checking:

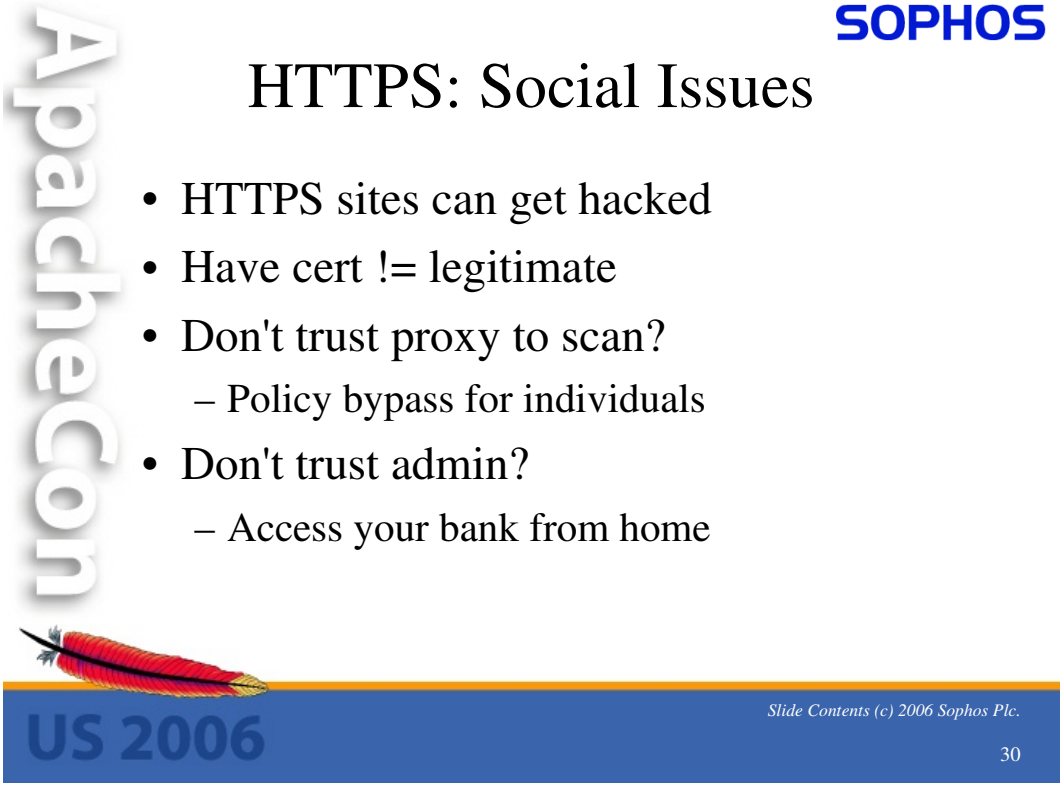
- dates (not before, not after)
- issuer signature (verify with pubkey in CA list)

Cert generation:

- challenge: getting mod_ssl (the input/output filter) working independently of the mod_proxy 'https' worker.
- sadly, most users will trust an unrecognized CA.

Transparent:

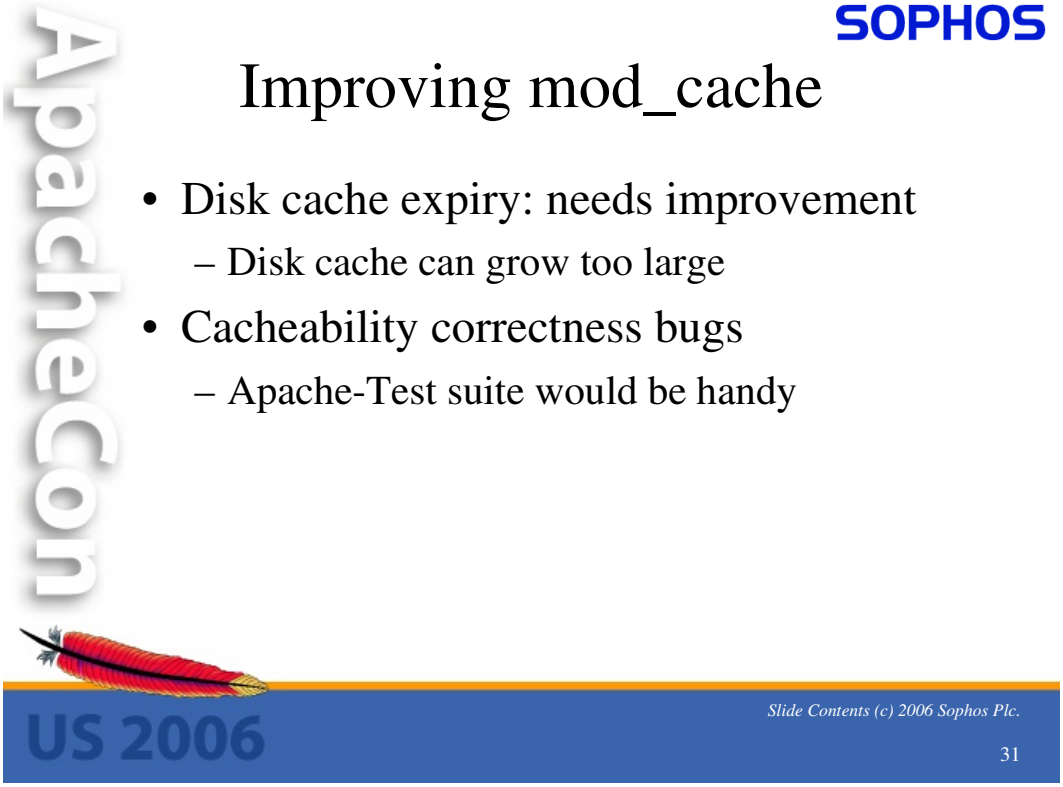
- a bit easier with HTTPS: the cert contains the hostname!



Guarantees that Certificate Authorities installed in your browser are legitimate? Issuers aren't perfect.

Serious privacy implications, but malware **can** appear on secure sites.

Research shows that often users will just ignore certificate warnings. Admin/IT policy for which CAs to trust can be enforced at the gateway.



htcacheclean too slow...

Our initial naive approach: delete objects at random =). Achieves disk-size limits for sure!

Better: expire objects intelligently LRU, LFU, etc. Meta-data stored in one place on the filesystem?

Summer of Code project work could improve the situation – I'm interested in seeing the results. Otherwise, this should be something Sophos will try to work on and share.

We developed an Apache-Test to uncover a few caching bugs. With a bit more work, this could be used to show RFC compliance.

SOPHOS

Improving mod_cache

- Store meta-data with objects
  - Expiry meta-data
  - Scan caching & revalidation
- Multi-layer cache providers
  - Scan revalidation as a top-level cache provider
  - Performance

Performance improvement: If we can store the version of the malware-scanning engine and it's signature data, we can skip re-scanning files “tagged” with the same revisions.

Multi-layer cache? Should improve performance, in theory.

Again, Summer of Code results haven't been released at the time of writing this.

Improving mod_proxy

- Persistent connections!
- Limiting connections to an Origin
- Overall throughput
  - Maybe best handled by OS' QoS

Persistent Client-proxy connections aren't **really** supported in Apache, unlike it's excellent support for when it's a web server.

NTLM, which is connection-oriented, can benefit from this.

Persistent proxy-origin connections would require a pool of “back-end” sockets, which is maybe possible with something like how the Event MPM handles keepalive connections. This approach would allow limiting of connections to origin servers, pipelining, etc.

Conclusions

- Apache: Not just a good web server!
  - Clear, modular design
- Key Challenges Covered:
  - Stopping-up
  - Keeping the User Happy
  - URI-based Policy
  - Apache improvements