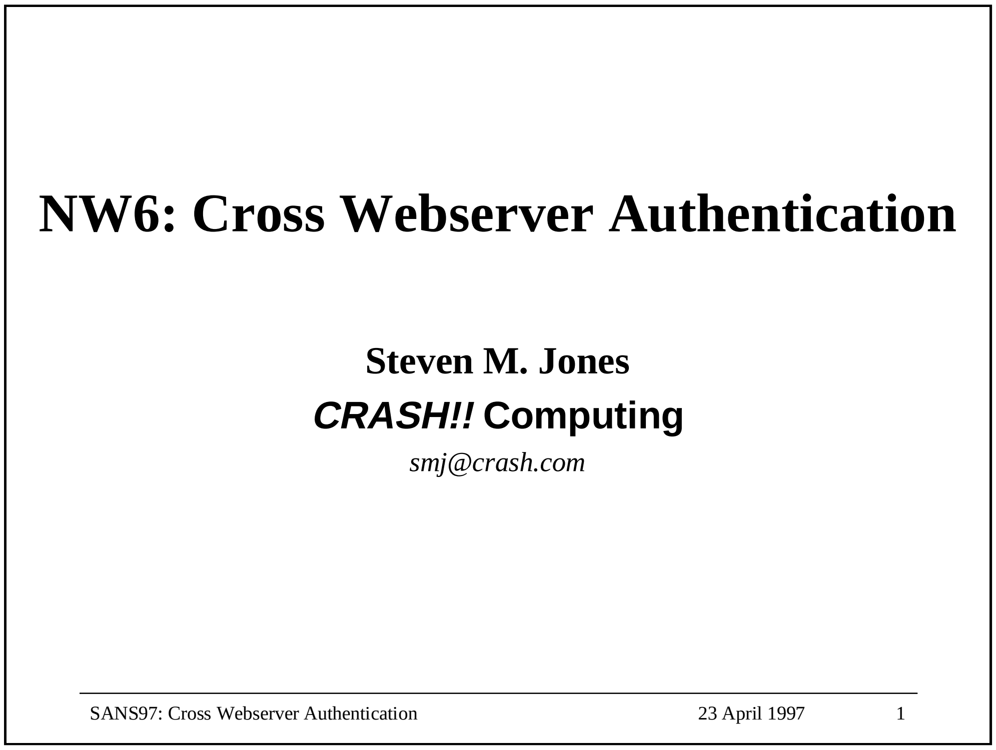


# NW6: Cross Webserver Authentication

**Steven M. Jones**

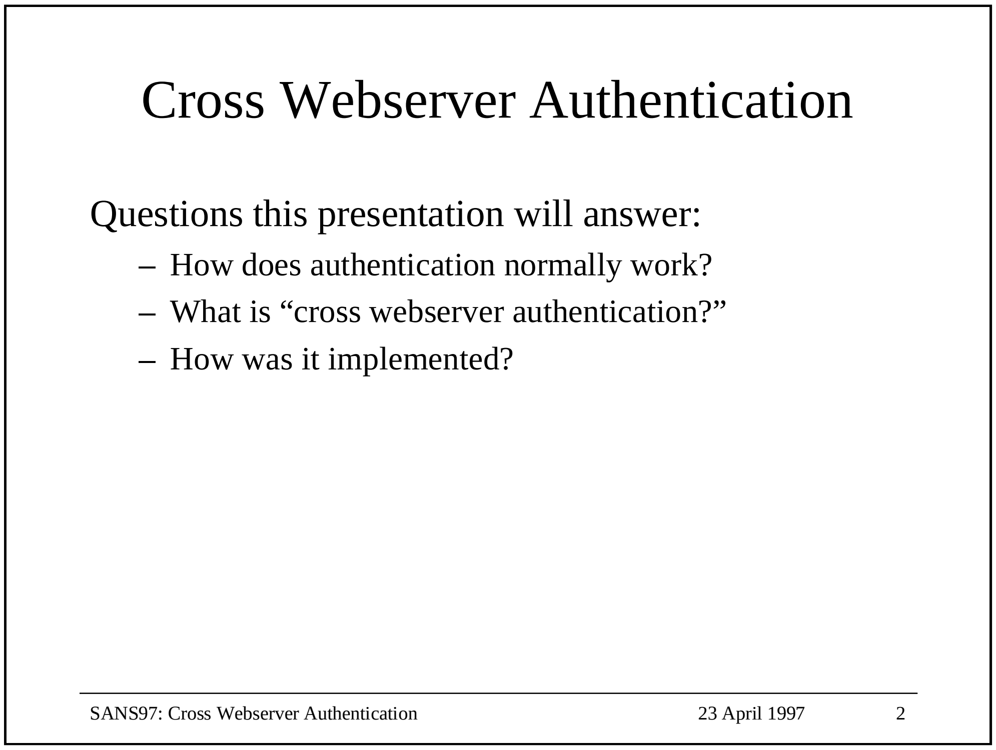
***CRASH!!* Computing**

*smj@crash.com*

SANS97: Cross Webserver Authentication 23 April 1997 1

Crash Computing Inc. is a small consulting firm operating in the areas of systems and network administration, systems and application programming, information systems, and network security.

This presentation describes work that was done for a client. It will not provide complete code examples, nor will it provide any more information about the client than is absolutely necessary. However it will review the concepts and mechanisms that were used to solve the problem indicated in the title of this talk, and it would be feasible to create a working solution using the techniques discussed herein.



## Cross Webserver Authentication

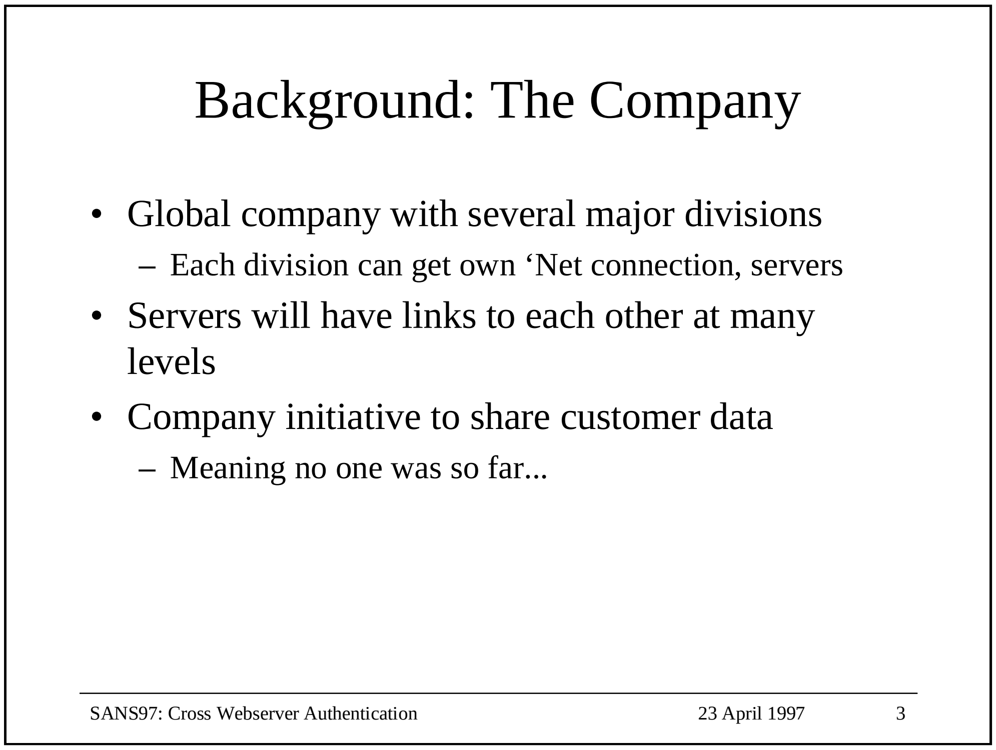
Questions this presentation will answer:

- How does authentication normally work?
- What is “cross webserver authentication?”
- How was it implemented?

SANS97: Cross Webserver Authentication 23 April 1997 2

The organization of this presentation is roughly as follows:

1. Introduction
2. Background
3. Normal Authentication
4. More Background
5. CSA: How its done
6. Summary



## Background: The Company

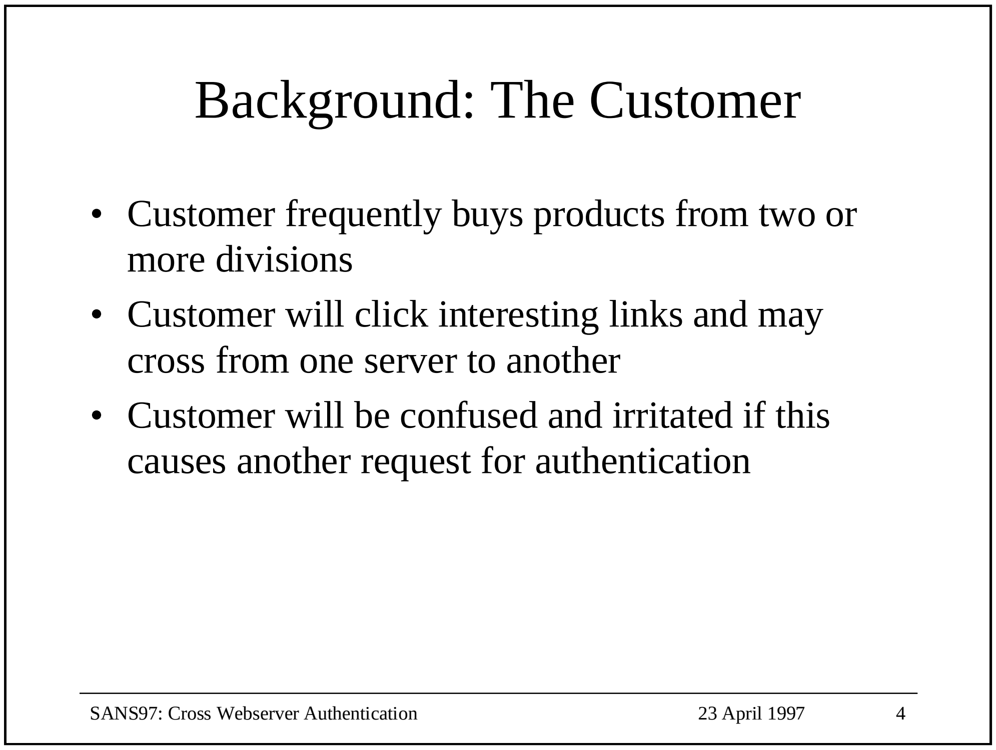
- Global company with several major divisions
  - Each division can get own ‘Net connection, servers
- Servers will have links to each other at many levels
- Company initiative to share customer data
  - Meaning no one was so far...

SANS97: Cross Webserver Authentication 23 April 1997 3

It will help to know that one of the company’s products was information, in the form of reports and advisories. Thus many documents on the Webservers were considered to have intrinsic value, as well as any services or applications that were available.

Indeed the divisions could and did deploy their own ‘Net connections and servers, as they had developed their own IT organizations in the past. Within the division Crash assisted there was also duplication of groups within IT at different locations around the world, including the Web group for that division. Fortunately only one of those groups was producing a public Web site at the time...

There was one customer database maintained by a different division, which had started life as a mailing list for customers of that division who received printed publications. Over time some groups within other divisions had been allowed to use it for their customers. While it was the logical place to store user authentication data for this project, there were many issues of control and budget that only began to be discussed during the project discussed in this presentation. So while everyone agreed it would be good to use this database for Web authentication data, its unknown whether or not those issues were ever addressed and resolved.



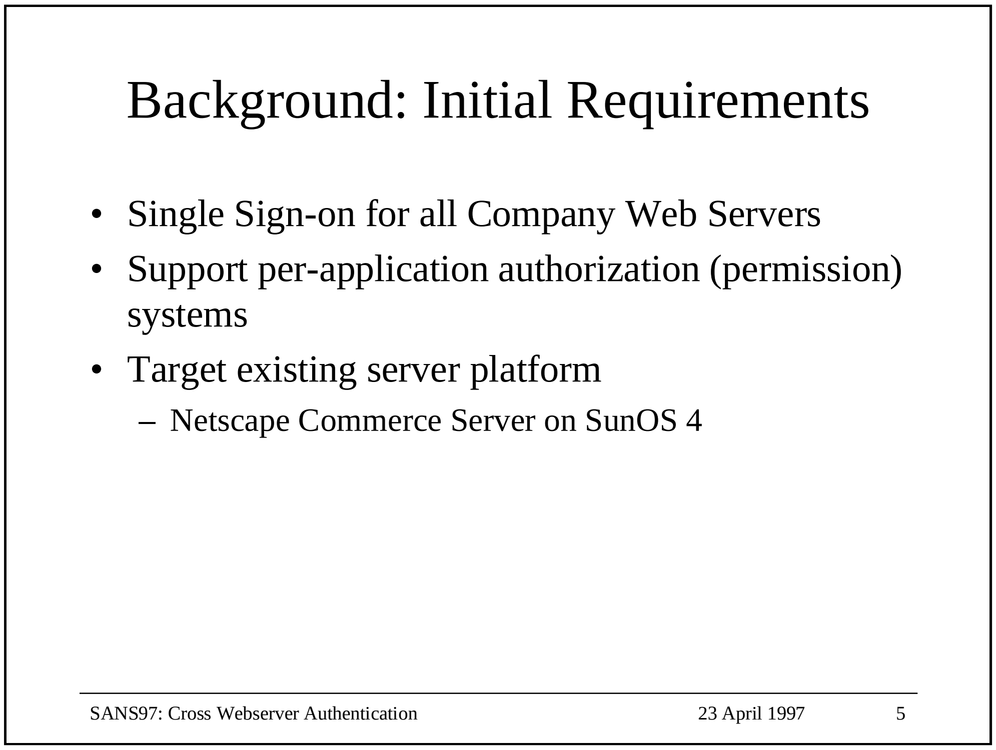
## Background: The Customer

- Customer frequently buys products from two or more divisions
- Customer will click interesting links and may cross from one server to another
- Customer will be confused and irritated if this causes another request for authentication

SANS97: Cross Webserver Authentication 23 April 1997 4

There were many synergy's between the information products produced by the different divisions. It was not at all uncommon for a report from one division to refer the reader to a particular advisory or service offered by another division.

The ability of the Web to support a highly interactive and immediate integration of these products was one of the reasons management viewed it as such a crucial initiative. Thus the customer was going to be skipping between the Webservers of different divisions sooner rather than later.



## Background: Initial Requirements

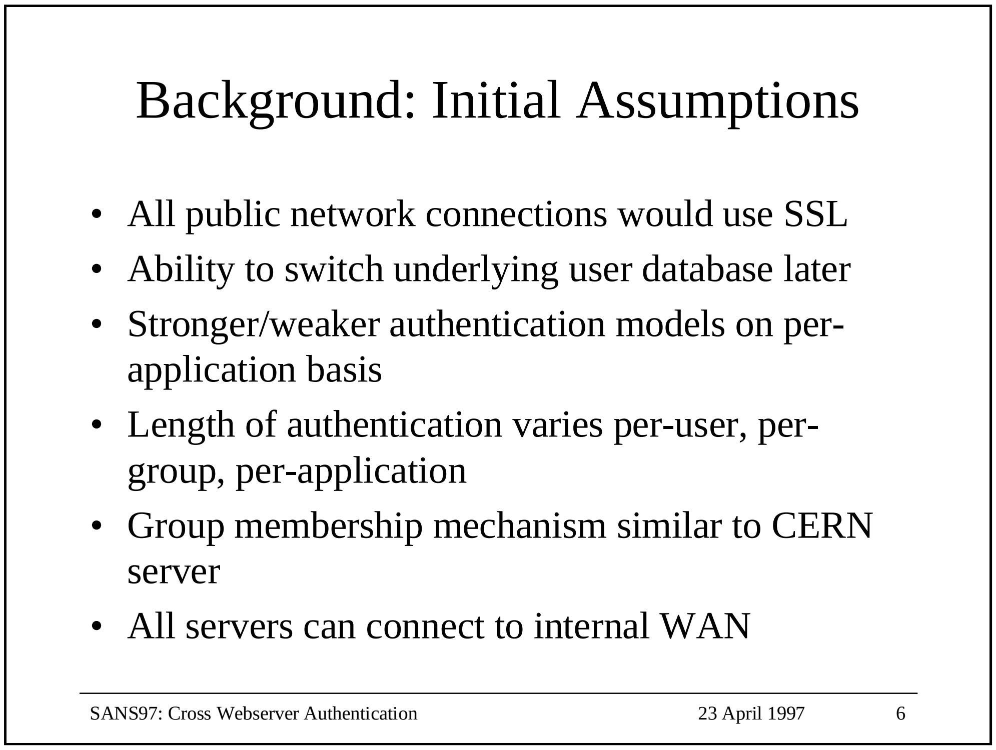
- Single Sign-on for all Company Web Servers
- Support per-application authorization (permission) systems
- Target existing server platform
  - Netscape Commerce Server on SunOS 4

SANS97: Cross Webserver Authentication 23 April 1997 5

“Single Sign-on” in this context means that if the user would be required to authenticate to one of the company’s Webservers, that authentication would be valid and recognized by all of the company’s other Webservers.

Several large applications were being developed by one division’s IT organization for internal and external deployment via the Web at this time. At least one of them already used a permission or “entitlements” system to determine which users had access to different parts of the application. It was unclear how this model was going to be extended in the next release of the application, but there was a clear desire that it still be part of the application and not moved out of that development group.

There was some discussion of migrating the internal and external Webserver to Solaris. In the early stages of the project this was declared an unlikely possibility, so little effort was spent planning for cross-platform development and testing of the solution. Given a second chance, this would not have been dropped so readily.



## Background: Initial Assumptions

- All public network connections would use SSL
- Ability to switch underlying user database later
- Stronger/weaker authentication models on per-application basis
- Length of authentication varies per-user, per-group, per-application
- Group membership mechanism similar to CERN server
- All servers can connect to internal WAN

SANS97: Cross Webserver Authentication 23 April 1997 6

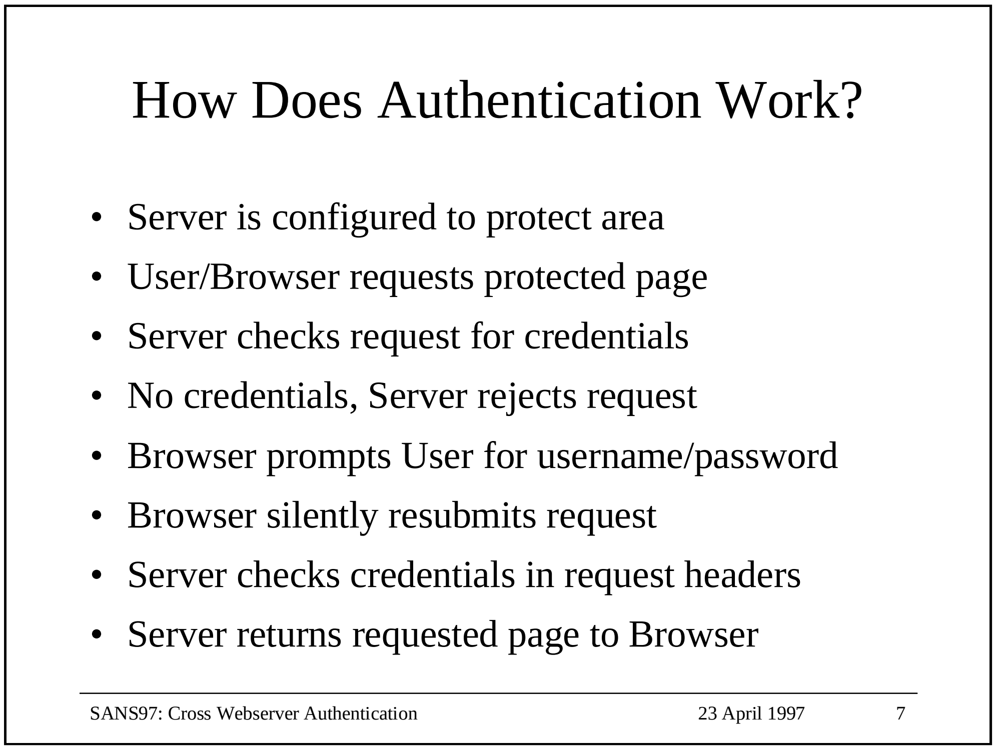
As the Web group examined this concept and the requirements from management, there seemed to be some things that could be assumed would be true for all applications, and some features that would greatly ease projects just appearing on the horizon.

For instance it seemed obvious that some apps or reports would be more sensitive than others, and therefore would require stronger authentication than a simple username and password. In addition, the authentication data would be vulnerable to interception unless SSL or a similar technique were used.

Since the business groups had not really begun to develop applications for broad deployment to the external Web server it was necessary to anticipate what mechanisms might be called for later, so that when called on they would be able to support whatever policy was deemed necessary.

One capability that was in use on the internal Web servers was the CERN server's ability to support Unix-like group membership. At the time there was no good way to implement this with the Commerce Server, so it was added to the list of desired features.

Access to the internal WAN will of course be mediated and controlled.



## How Does Authentication Work?

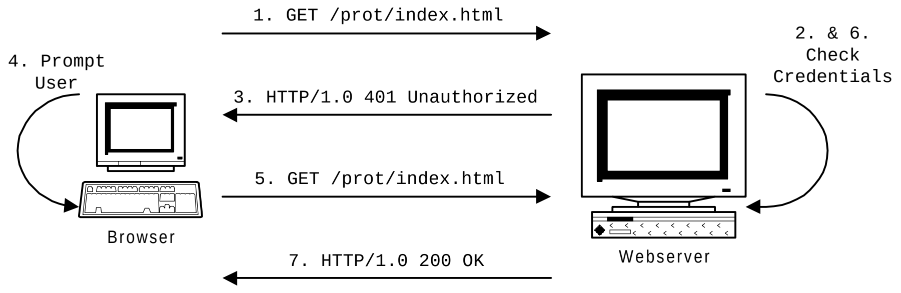
- Server is configured to protect area
- User/Browser requests protected page
- Server checks request for credentials
- No credentials, Server rejects request
- Browser prompts User for username/password
- Browser silently resubmits request
- Server checks credentials in request headers
- Server returns requested page to Browser

SANS97: Cross Webserver Authentication 23 April 1997 7

To get a feel for the protocol, try connecting to port 80 of a local web server using Telnet. Type "`GET / HTTP/1.0`" hit Enter twice, and examine the output. Notice that the first few lines look like the headers of a MIME mail message.

The specification for the HTTP/1.0 protocol currently in use by most of the Web is not an Internet standard per the IETF, but is completely documented in RFC1945. A replacement protocol, HTTP/1.1, is being developed as an IETF standard and has been assigned RFC2068. Both of these documents are available from the RFC repository cited at the end of this presentation.

## How Does Authentication Work?

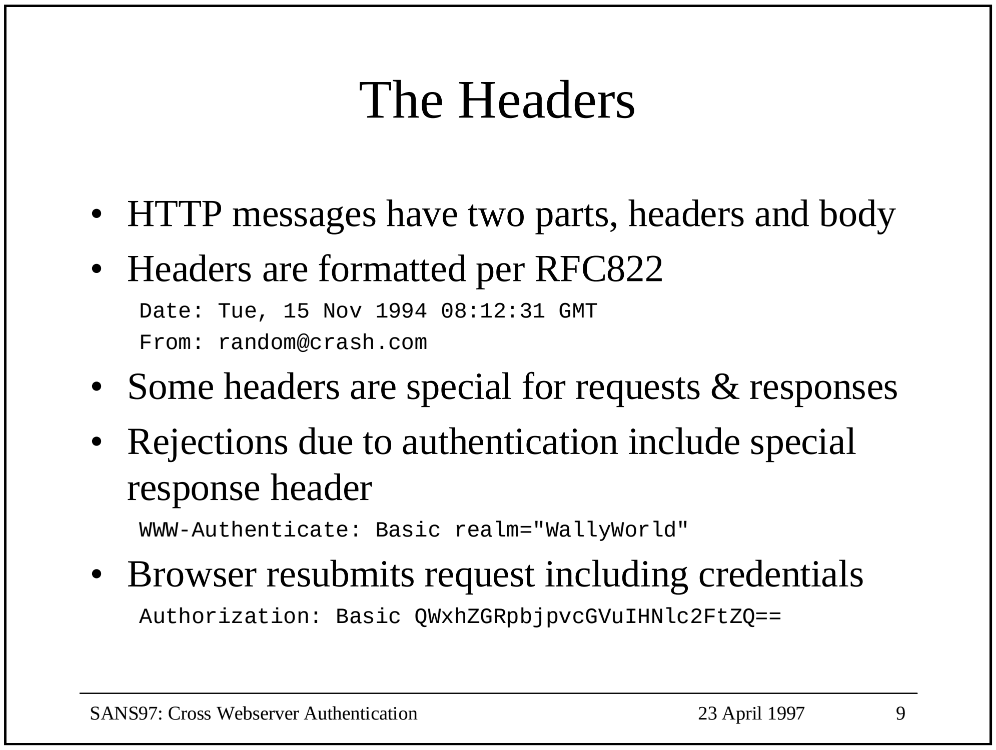


SANS97: Cross Webserver Authentication 23 April 1997 8

At some point assume that the Webserver has been configured to protect the document tree under the path /prot. This exchange would then occur:

1. User/Browser requests protected page.
2. Server checks request for credentials.
3. No credentials were included, Server rejects request.
4. Browser prompts User for username/password.
5. Browser silently resubmits request with credentials User supplied.
6. Server checks credentials in request headers. Assume they are valid.
7. Server returns requested page to Browser.

All common Browsers will cache the credentials a user supplies for a given server based on the hostname. As far as I know none of them seem to properly support storing different credentials for different parts of a given server’s document tree.



## The Headers

- HTTP messages have two parts, headers and body
- Headers are formatted per RFC822

```
Date: Tue, 15 Nov 1994 08:12:31 GMT
From: random@crash.com
```

- Some headers are special for requests & responses
- Rejections due to authentication include special response header

```
WWW-Authenticate: Basic realm="WallyWorld"
```

- Browser resubmits request including credentials

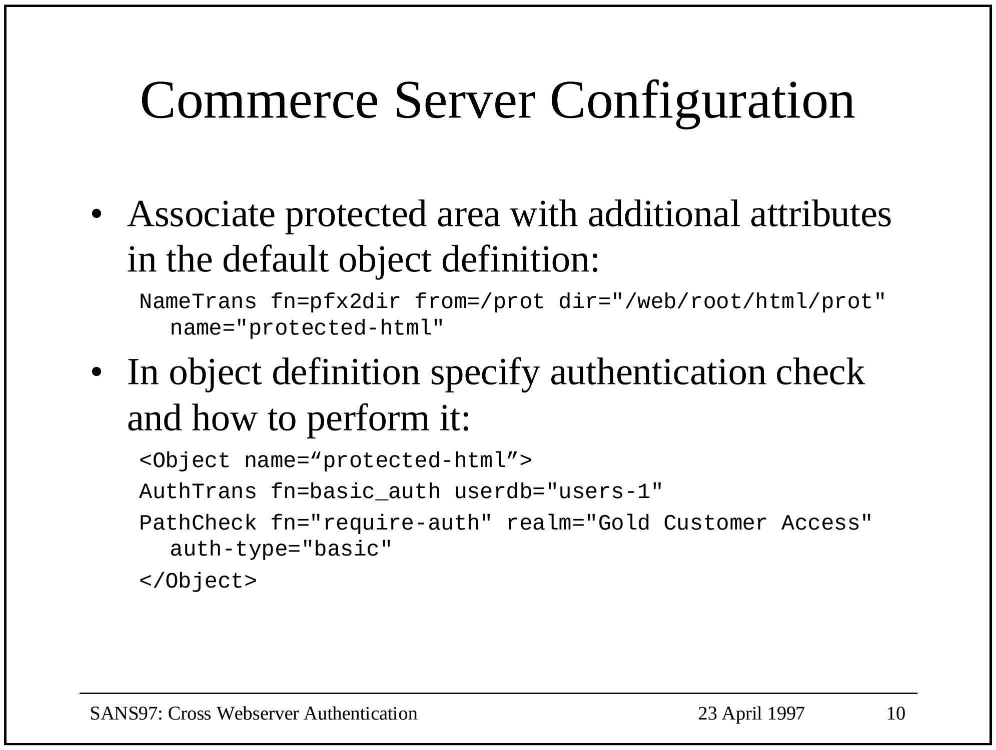
```
Authorization: Basic QWxhZGRpbjpvcGVuIHNlc2FtZQ==
```

SANS97: Cross Webserver Authentication 23 April 1997 9

HTTP/1.0 only details one authentication scheme, Basic or Basic-auth. While the spec doesn't preclude other methods, this is the only one widely implemented. Basic is therefore the only method that was considered for this project. Recent work has produced RFC2069, which seeks to introduce an authentication scheme that will remedy some of Basic's more obvious shortcomings. It is likely to be incorporated into HTTP/1.1.

The `WWW-Authenticate:` response header indicates which authentication mechanism the server requires for the protected object, and includes parameters that the browser should use to obtain credentials (presumably from the user). Common practice with Basic is that the "realm" string is used in a dialog box that prompts the user for a username/password combination.

The `Authentication:` request header is a simple base64 encoding of the username and password formatted as "username:password" The example above decodes to "Aladdin:open sesame" The base64 encoding does nothing to enhance the security of the protocol, but it does allow special characters to be used in the username/password that would violate the RFC822 header specification if they were included in the clear.



## Commerce Server Configuration

- Associate protected area with additional attributes in the default object definition:

```
NameTrans fn=pfx2dir from=/prot dir="/web/root/html/prot"
  name="protected-html"
```

- In object definition specify authentication check and how to perform it:

```
<Object name=“protected-html”>
AuthTrans fn=basic_auth userdb="users-1"
PathCheck fn="require-auth" realm="Gold Customer Access"
  auth-type="basic"
</Object>
```

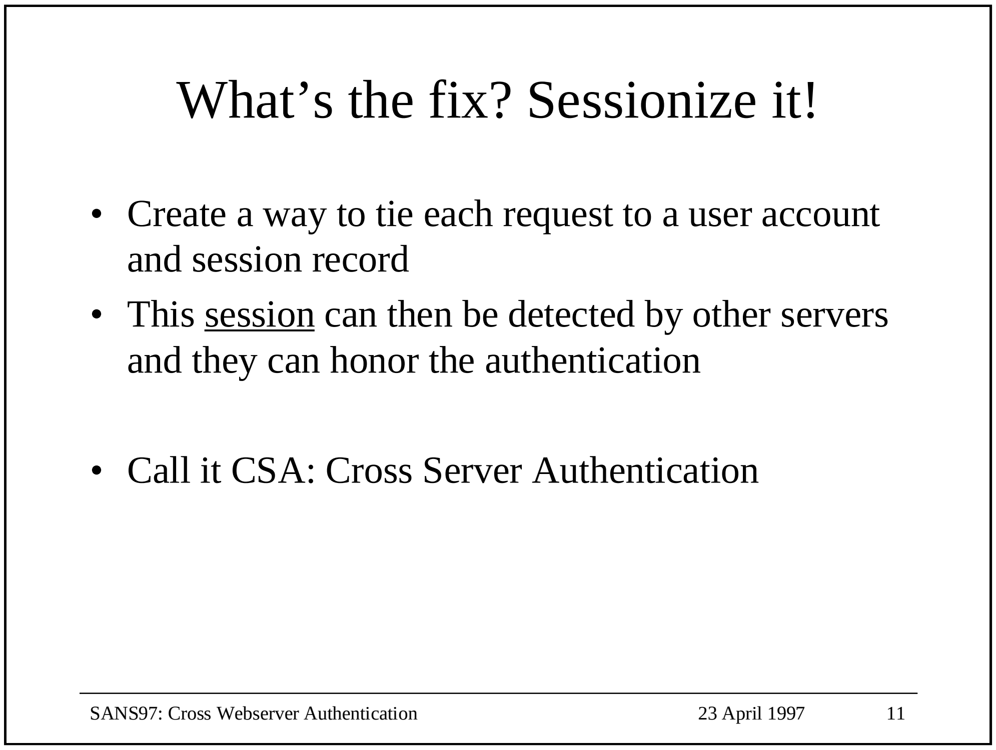
SANS97: Cross Webserver Authentication 23 April 1997 10

Here is a more complete example of the `obj.conf` file:

```
<Object name=default>
NameTrans fn=pfx2dir from=/prot dir="/web/root/html/prot" name="protected-html"
NameTrans fn=pfx2dir from=/mc-icons dir="/web/root/mc-icons"
NameTrans fn=pfx2dir from=/images dir="/web/root/images"
NameTrans fn=pfx2dir from=/cgi-bin dir="/web/root/cgi-bin" name="cgi"
NameTrans fn=document-root root="/web/root/html"
PathCheck fn=unix-uri-clean
PathCheck fn=find-pathinfo
PathCheck fn=find-index index-names="index.html,home.html,welcome.html"
ObjectType fn=type-by-extension
ObjectType fn=force-type type=text/plain
Service method=(GET|HEAD) type=magnus-internal/imagemap fn=imagemap
Service method=(GET|HEAD) type=magnus-internal/directory fn=index-simple
Service method=(GET|HEAD) type=*~magnus-internal/* fn=send-file
AddLog fn=common-log
</Object>

<Object name=cgi>
ObjectType fn=force-type type=magnus-internal/cgi
Service fn=send-cgi
</Object>

<Object name=protected-html>
AuthTrans fn=basic_auth userdb="users-1"
PathCheck fn="require-auth" realm="Gold Customer Access" auth-type="basic"
</Object>
```

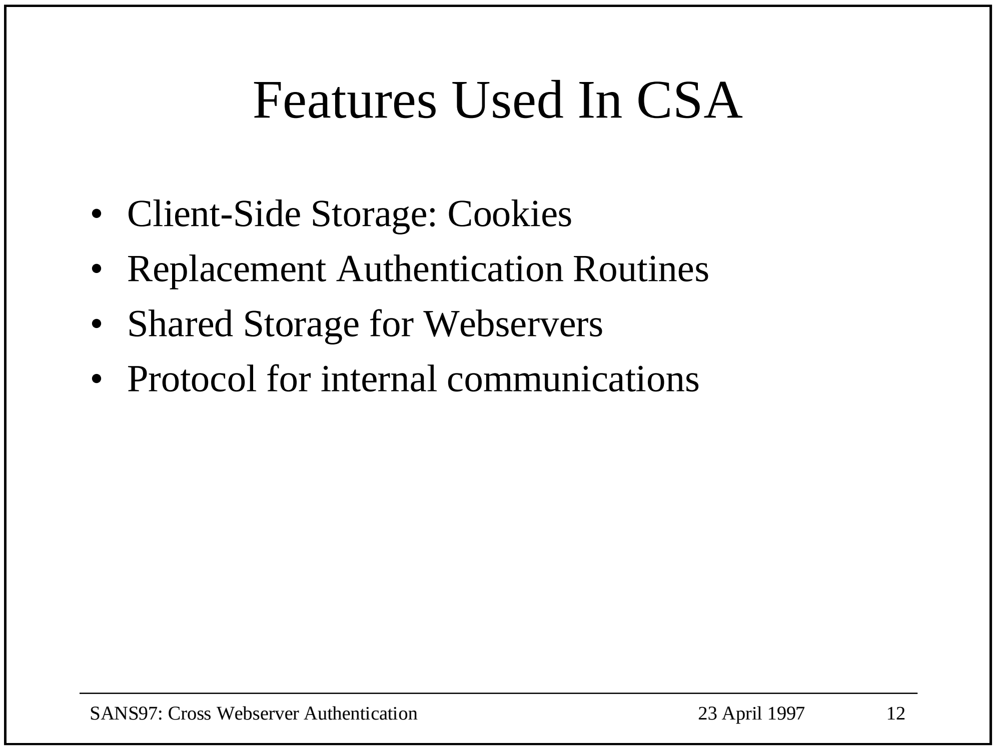


## What’s the fix? Sessionize it!

- Create a way to tie each request to a user account and session record
- This <u>session</u> can then be detected by other servers and they can honor the authentication

- Call it CSA: Cross Server Authentication

SANS97: Cross Webserver Authentication 23 April 1997 11

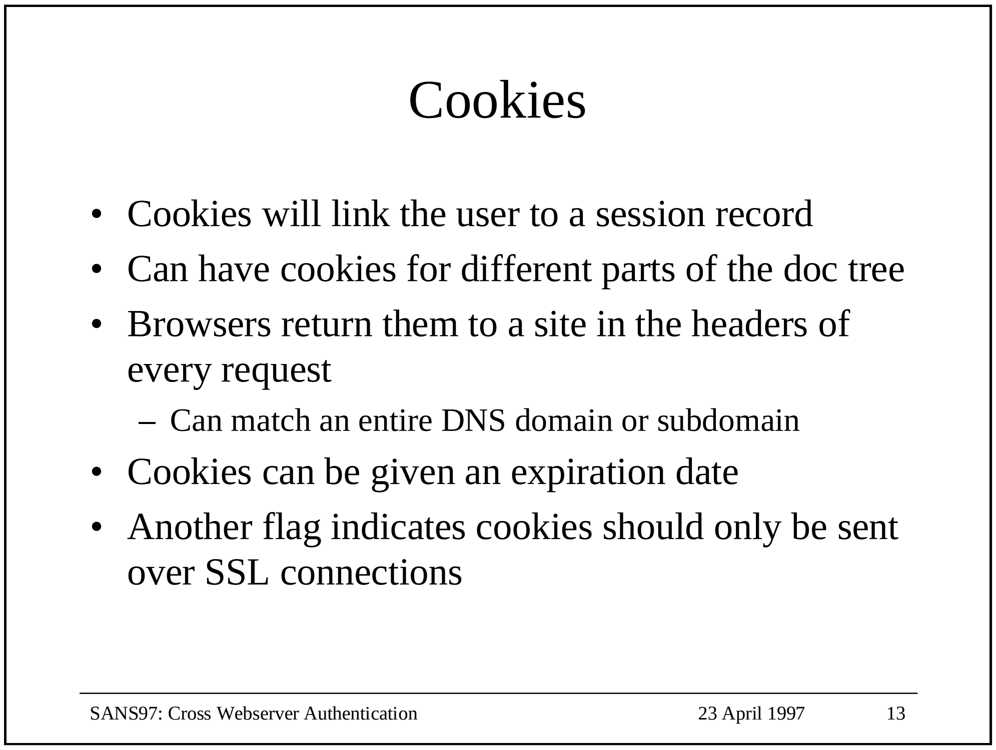


## Features Used In CSA

- Client-Side Storage: Cookies
- Replacement Authentication Routines
- Shared Storage for Webservers
- Protocol for internal communications

SANS97: Cross Webserver Authentication 23 April 1997 12

Each of these items will be covered in detail in the next few slides.



## Cookies

- Cookies will link the user to a session record
- Can have cookies for different parts of the doc tree
- Browsers return them to a site in the headers of every request
  - Can match an entire DNS domain or subdomain
- Cookies can be given an expiration date
- Another flag indicates cookies should only be sent over SSL connections

SANS97: Cross Webserver Authentication 23 April 1997 13

Browsers that conform to the Netscape preliminary spec for cookies will have the following minimum capacities:

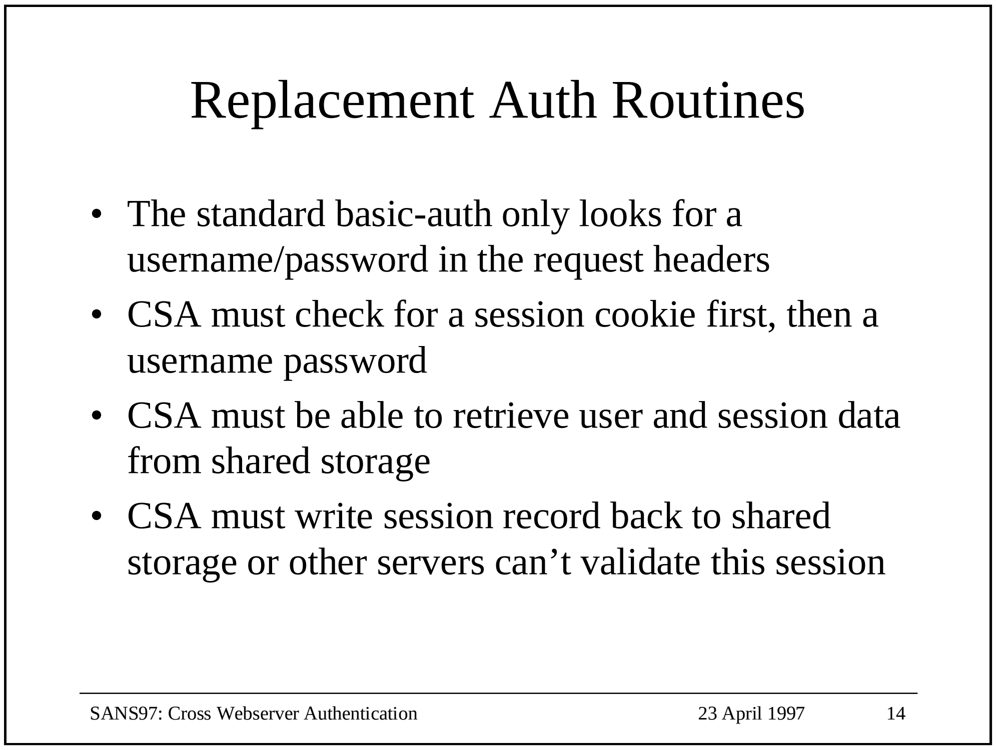
300 cookies overall

4KBytes per cookie

20 cookies per server/hostname

20 cookies per domain

When these limits are exceeded an LRU algorithm will be used to decide which objects to delete.



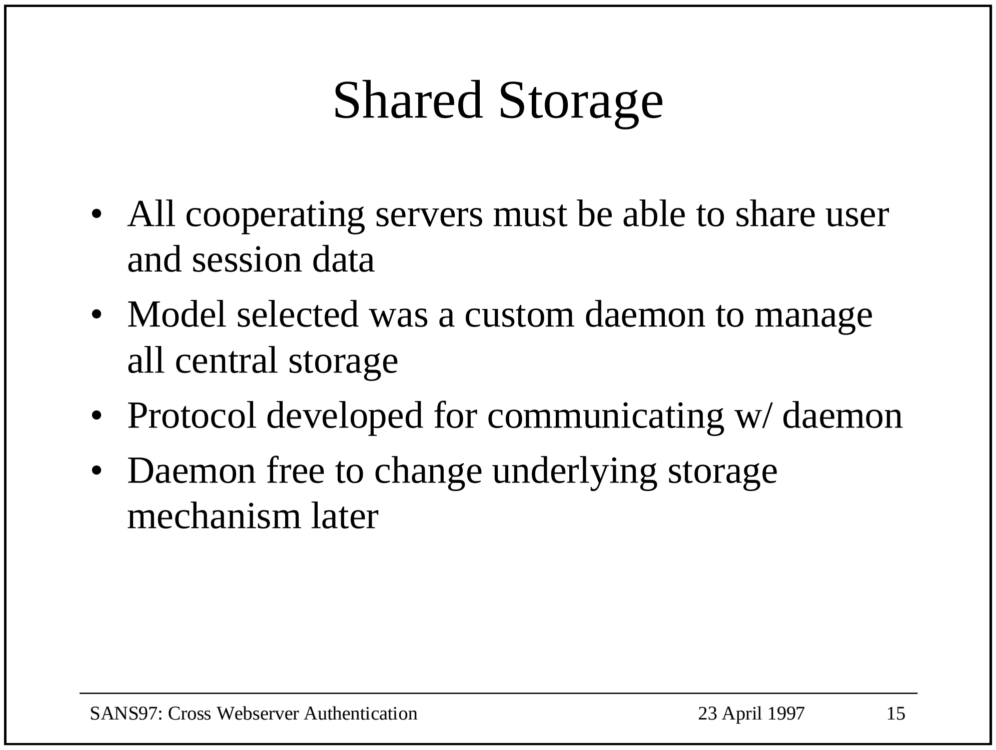
## Replacement Auth Routines

- The standard basic-auth only looks for a username/password in the request headers
- CSA must check for a session cookie first, then a username password
- CSA must be able to retrieve user and session data from shared storage
- CSA must write session record back to shared storage or other servers can't validate this session

SANS97: Cross Webserver Authentication 23 April 1997 14

In the incredibly slim developer's documentation that accompanied the Commerce Server, Netscape specifies how AuthTrans functions are supposed to return status using the `REQ_NOACTION` and `REQ_ABORTED` constants. It's worth noting here that the documented methods of returning from AuthTrans functions did not work as advertised.

I tried to get answers out of tech support, but they upheld the documentation and in my environment that just didn't work. As I recall I had to basically switch what the documentation said for these two return codes before it behaved properly.



## Shared Storage

- All cooperating servers must be able to share user and session data
- Model selected was a custom daemon to manage all central storage
- Protocol developed for communicating w/ daemon
- Daemon free to change underlying storage mechanism later

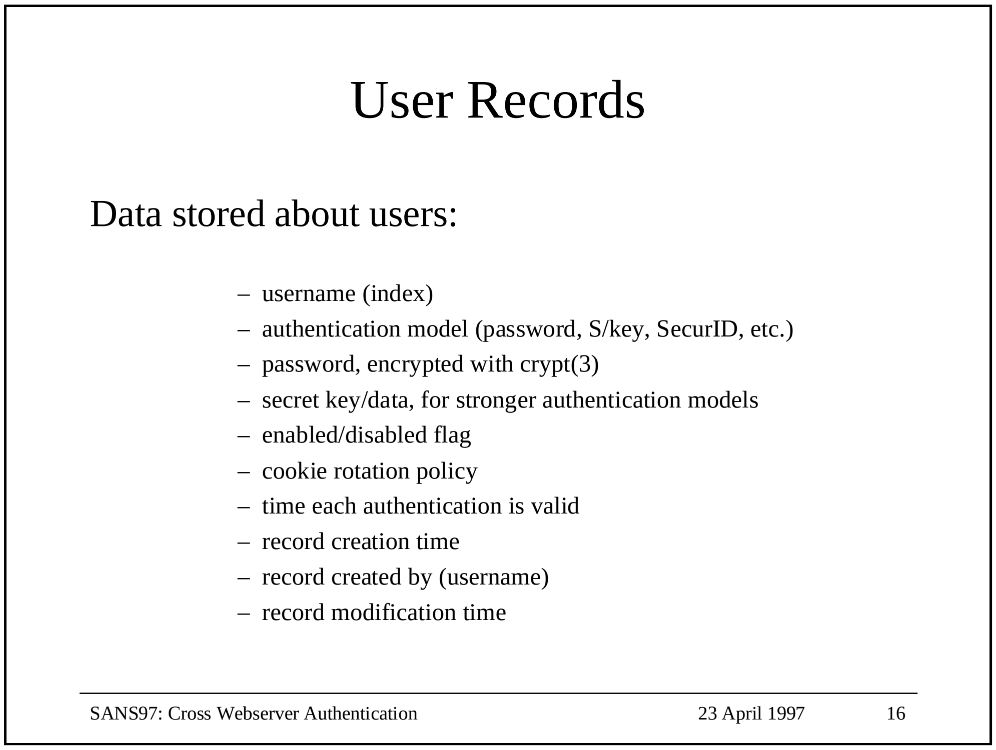
SANS97: Cross Webserver Authentication 23 April 1997 15

One of the general initiatives at the company was to develop a centralized user database for both internal and external users of the Web and other information systems. Clearly this should at some point become the source of all user information for CSA, hence the desirability of being able to mask the storage mechanism used by the daemon.

Other models considered included rdist'ed flat files, direct calls to Sybase, and unmediated server-to-server communications. Flat files shared user data effectively, but didn't help with session data at high hit-rates. Linking any other vendor's code into the Commerce Server was considered a bad idea because of the potential for endless vendor finger-pointing. Many-to-many connections would create a registration or configuration problem: either each server's configuration files would have to be updated whenever another server came online, or there would need to be some sort of registration entity. Plus then each server is trying to keep a local database of duplicate data… Yech.

It was decided the perils of amateur protocol design were preferable to the alternatives.

I was able to tag the centralized user database the Global User Registry and Profile System, or GURPS (apologies to Steve Jackson) in a few memos, but I'm not sure if it stuck after the project was delayed.



## User Records

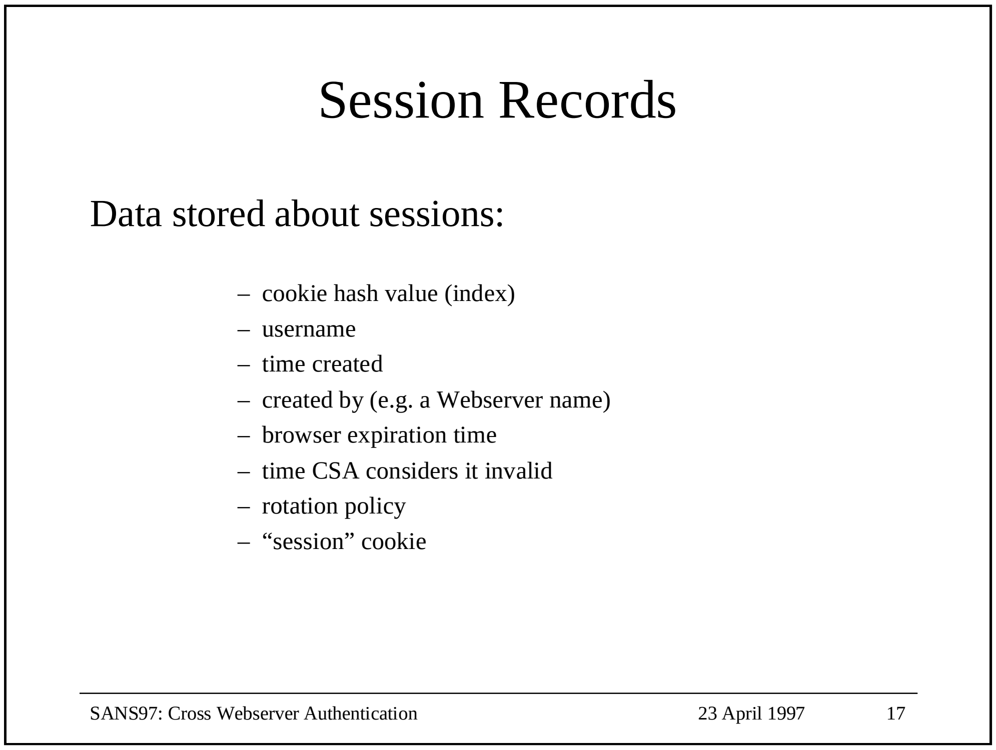
Data stored about users:

- username (index)
- authentication model (password, S/key, SecurID, etc.)
- password, encrypted with crypt(3)
- secret key/data, for stronger authentication models
- enabled/disabled flag
- cookie rotation policy
- time each authentication is valid
- record creation time
- record created by (username)
- record modification time

SANS97: Cross Webserver Authentication 23 April 1997 16

No effort was spent exploring stronger authentication methods in the period covered in this presentation. The fields and data-structures theoretically allowed for it, but that was never put to the test. For starters the method of challenging the user for authentication credentials would have to be reconsidered. Secondly there were no business groups asking for better authentication. Thirdly, the group functionality had been postponed until after the basic system had been delivered, so it would have been the next item slated for implementation.

The length of time an authentication was good for was set on a per-user basis. This was expected to serve as a default, in case no value were specified for a given document being requested or group that the user was a member of. The same intention held for the “cookie rotation policy” field.



## Session Records

Data stored about sessions:

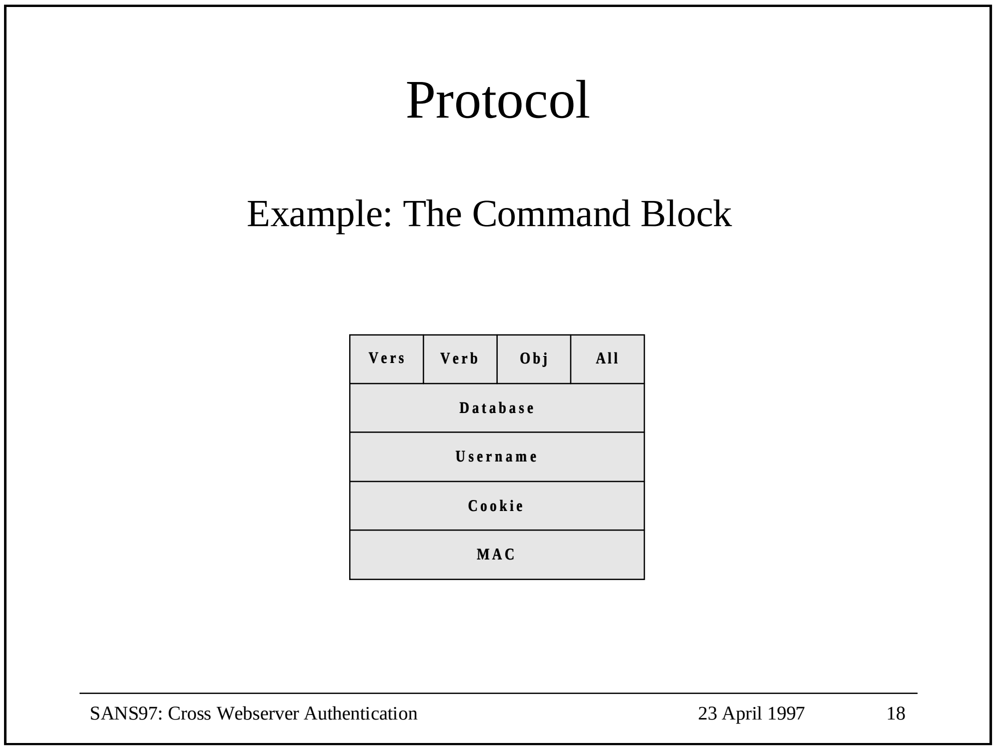
- cookie hash value (index)
- username
- time created
- created by (e.g. a Webserver name)
- browser expiration time
- time CSA considers it invalid
- rotation policy
- “session” cookie

SANS97: Cross Webserver Authentication 23 April 1997 17

The session cookie requires some explanation. Cookies always had expiration times set because in early discussions it had been deemed desirable to have a user be able to exit their browser, have lunch, start their browser, and resume their session without re-authenticating. This meant that the browser expiration time would have to be set for each cookie, which is the only way a browser will store a cookie in a local file that survives across program startups.

A short while later it became apparent that it would be trivial for a malicious user to copy this file and its associated cookie, and thereby gain unauthorized access. Without questioning the original goal of the expiration time, it was decided that a second cookie without an expiration time should also be used. If the main cookie were sent without the “session” cookie it would mean that the user might be legitimate and should be re-authenticated, perhaps using a more friendly prompt.

So the session cookie protects against abuse of the cookie cache file, but it also eliminates the real benefit of using it in the first place - the user can no longer continue an authenticated session across restarts of the browser. As soon as that was realized the session cookie and the requirement should have been dropped, or at least reconsidered, but instead the code just grew a little cruftier...



## Protocol

Example: The Command Block

| | | | |
|---|---|---|---|
| Vers | Verb | Obj | All |
| Database | | | |
| Username | | | |
| Cookie | | | |
| MAC | | | |

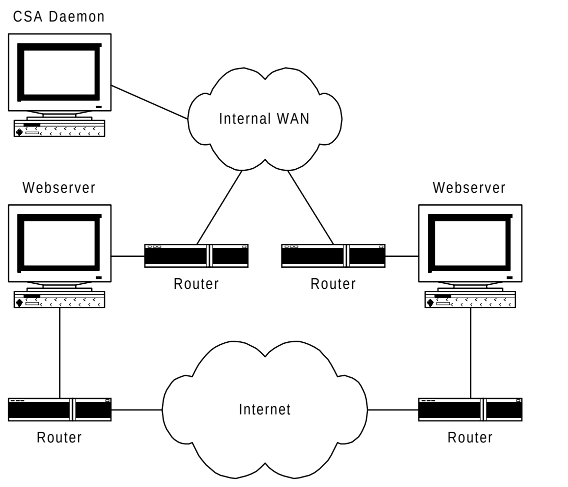
SANS97: Cross Webserver Authentication 23 April 1997 18

The Message Authentication Code (MAC) was used for all packets exchanged between Webservers and the CSA daemon. It was intended mostly to make it difficult to interfere with protocol traffic over the internal WAN. The MD4 algorithm was used again, this time it was fed all packet data plus a communications password that was known to all CSA webservers and the daemon.

The diagram indicates the structure of a Command Block (CBlock) in the protocol. This standard, fixed-size structure specified an action to take on a single user or session record, all such records in a given database, or on all such records under the daemon's management.

In response to a CBlock, the recipient would send a Response Block (RBlock). That RBlock, a fixed-size structure not dissimilar to the CBlock, would indicate whether or not it was being followed with a sequence of data records.

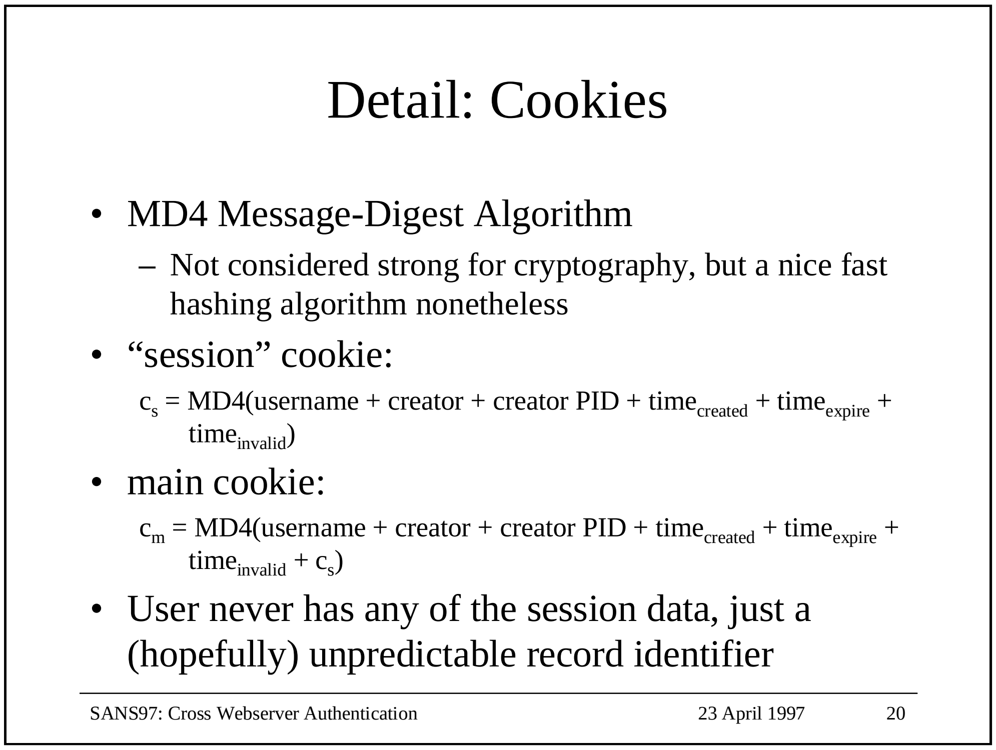
## Webserver Network Diagram



SANS97: Cross Webserver Authentication 23 April 1997 19

The configuration and network topology for Webservers was a subject of much debate and change during this project. Of course each division wound up deploying very different network topologies, and almost very different server OS platforms. In essence, they were configured the same as firewalls in the division that commissioned the CSA work, and they were set up and administered by the same group as handled the "real" firewalls.

The routers in this diagram are all configured to act as packet filters. The relative merits of routers versus general purpose CPUs as firewall components is far outside the scope of this talk; suffice it to say that the interior routers were configured to allow a single TCP connection from the Webserver to the host labeled, "CSA Daemon."



## Detail: Cookies

- MD4 Message-Digest Algorithm
  - Not considered strong for cryptography, but a nice fast hashing algorithm nonetheless
- “session” cookie:

  $c_s$ = MD4(username + creator + creator PID + $time_{created}$ + $time_{expire}$ + $time_{invalid}$)
- main cookie:

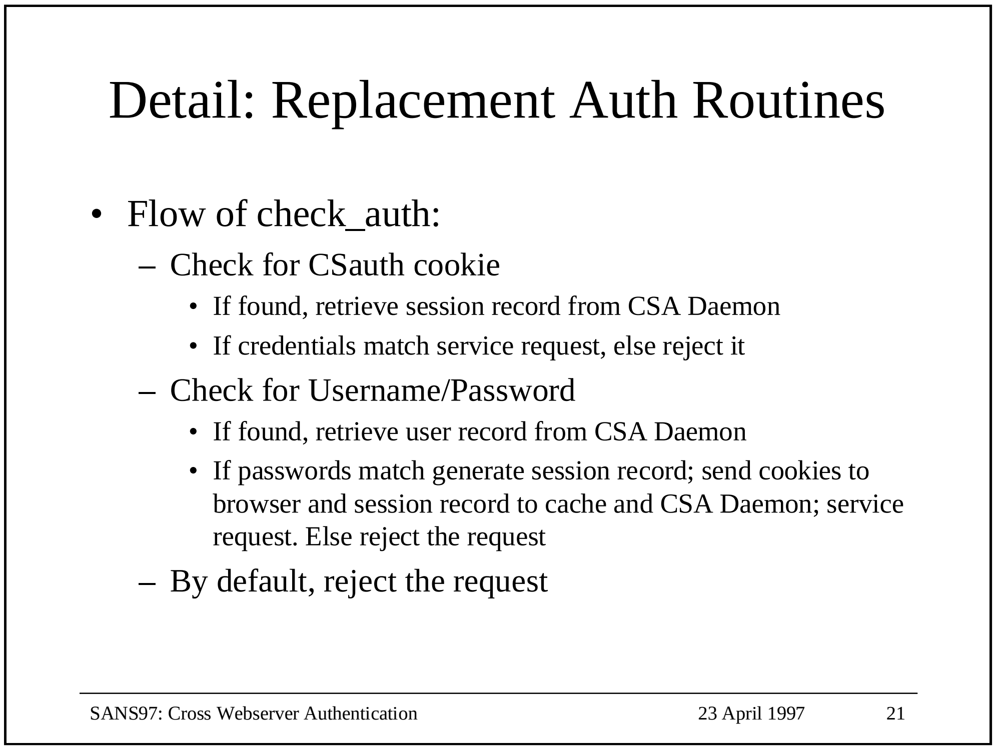
  $c_m$ = MD4(username + creator + creator PID + $time_{created}$ + $time_{expire}$ + $time_{invalid}$ + $c_s$)
- User never has any of the session data, just a (hopefully) unpredictable record identifier

SANS97: Cross Webserver Authentication 23 April 1997 20

MD4 was selected for two reasons: I remembered it was listed as being fast in Bruce Schneier’s book *Applied Cryptography*, and the code was readily available including a routine to render a printable string from the hash value.

The main cookie actually included the entire session record structure, including the “session” hash value, but since the other fields rarely varied this is essentially correct. The name of the main cookie was CSauth, and the other was CSsession.

There was some desire to include more data about the client such as an IP or Email address. However IP addresses were threatened by various proxy schemes being discussed at the time, and Email addresses are unreliable at best, and at worst provide a malicious user with some ability to run known plaintext through our hashing algorithm. One assumption about the system, and a reason for choosing a cryptographic hashing algorithm, is that it should be made difficult for malingerers to predict cookie hash values.



## Detail: Replacement Auth Routines

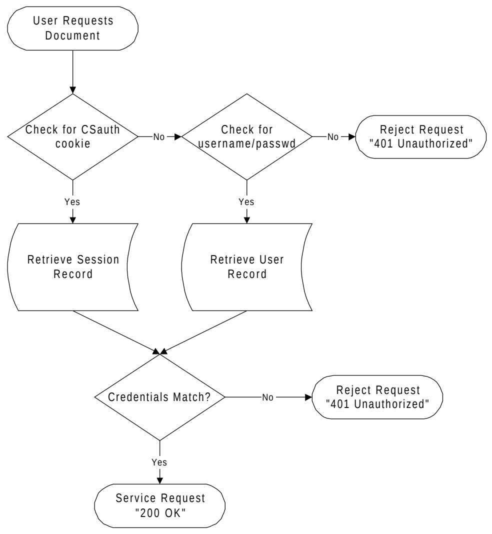
- Flow of check_auth:
  - Check for CSauth cookie
    - If found, retrieve session record from CSA Daemon
    - If credentials match service request, else reject it
  - Check for Username/Password
    - If found, retrieve user record from CSA Daemon
    - If passwords match generate session record; send cookies to browser and session record to cache and CSA Daemon; service request. Else reject the request
  - By default, reject the request

SANS97: Cross Webserver Authentication 23 April 1997 21

To put it a different way, here’s some pseudo-code:

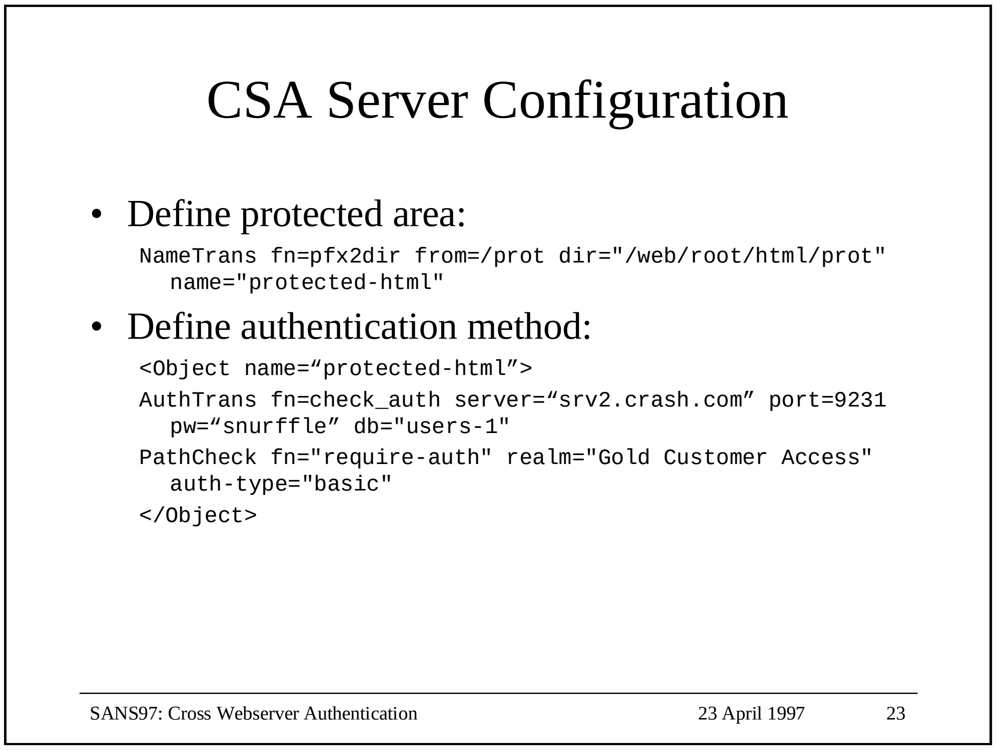
```
int check_auth(pblock *pb, Session *sn, Request *rq)
{
  /* 1. See if they have a valid pair of cookies */
  if (check_cookie(pb, sn, rq) == 0)
    return REQ_PROCEED;         /* They check out */
  /* 2. Check to see if they've included a username/password */
  if (check_decode_creds(pb, sn, rq)!= 0)
    return REQ_NOACTION;       /* no Authorization: header */
  else
    if (check_user(pb, sn, rq, &urec) != 0)
      return REQ_NOACTION;     /* user/pass didn’t match record */
  /* 3. Passwords matched, generate a new session record */
  srec = session_create(pblock_findval("user", pb), urec.cookie_cycle,
                        (now + urec.authlifetime), (now + urec.authlifetime),
                        now, creator);
  /* 4. Generate hash to send in browser cookie */
  hash(cookie, (char *)srec, sizeof(SessionRec));
  /* 5. Send new session record to CSA Daemon and store it in local cache */
  store_cookie(cookie, srec, pb, sn, rq);
  /* 6. Set response headers and return */
  set_response_headers(cookie, srec, pb, sn, rq);
  return REQ_PROCEED;
}
```

## Detail: Replacement Auth Routines



SANS97: Cross Webserver Authentication 23 April 1997 22

All right I got a little carried away with the drawing package. But in the rather unlikely event that it could make something clearer, here’s the amateur flowchart version.



## CSA Server Configuration

- Define protected area:

```
NameTrans fn=pfx2dir from=/prot dir="/web/root/html/prot"
  name="protected-html"
```

- Define authentication method:

```
<Object name=“protected-html”>
AuthTrans fn=check_auth server=“srv2.crash.com” port=9231
  pw=“snurffle” db="users-1"
PathCheck fn="require-auth" realm="Gold Customer Access"
  auth-type="basic"
</Object>
```

SANS97: Cross Webserver Authentication 23 April 1997 23

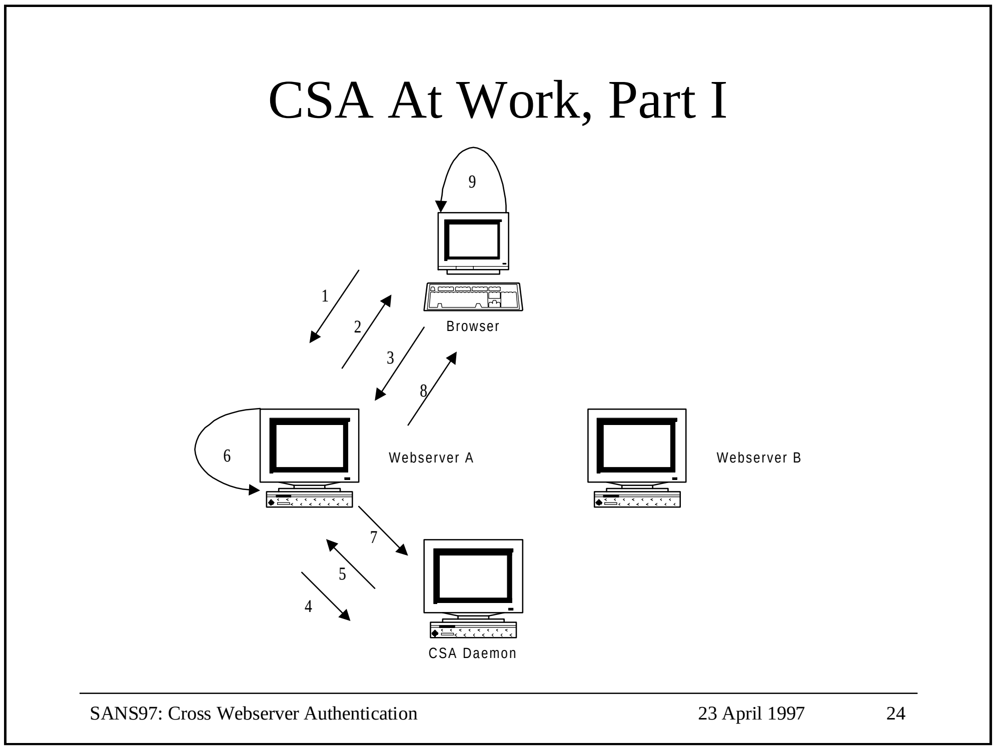
Here is an almost-complete example of the `obj.conf` file:

```
<Object name=default>
NameTrans fn=pfx2dir from=/prot dir="/web/root/html/prot" name="protected-html"
NameTrans fn=pfx2dir from=/cgi-bin dir="/web/root/cgi-bin" name="cgi"
NameTrans fn=document-root root="/web/root/html"
PathCheck fn=unix-uri-clean
PathCheck fn=find-pathinfo
ObjectType fn=type-by-extension
ObjectType fn=force-type type=text/plain
Service method=(GET|HEAD) type=magnus-internal/directory fn=index-simple
Service method=(GET|HEAD) type=*~magnus-internal/* fn=send-file
AddLog fn=common-log
</Object>

<Object name=“protected-html”>
AuthTrans fn=check_auth server=“srv2.crash.com” port=9231 pw=“snurffle”
db="users-1"
PathCheck fn="require-auth" realm="Gold Customer Access" auth-type="basic"
</Object>
```

And from the end of the `magnus.conf` file:

```
Init fn=load-modules shlib=/web/root/lib/webauth_mod.so
funcs="cache_init,check_auth,check_cookie,check_user,check_decode_creds,hash,se
ssion_create"
Init fn=cache_init cachetime=5 cachefile=/web/root/tmp/cookie_cache
```



## CSA At Work, Part I

SANS97: Cross Webserver Authentication 23 April 1997 24

Here’s a typical Authenticated Web session using CSA. Assume that the Webservers have been configured to protect every page in their document trees using CSA:

1. User generates normal HTTP request by opening a URL or bookmark.

2. Webserver A examines the request for credentials - first cookies, then username and password. It finds none, so it rejects the request.

3. Browser prompts the user for a username/password and resubmits.

4. Webserver A examines the request for credentials - first cookies, then username and password. It finds a username/password combination, so it requests that user record from the CSA Daemon.

5. The CSA Daemon returns the appropriate user record.

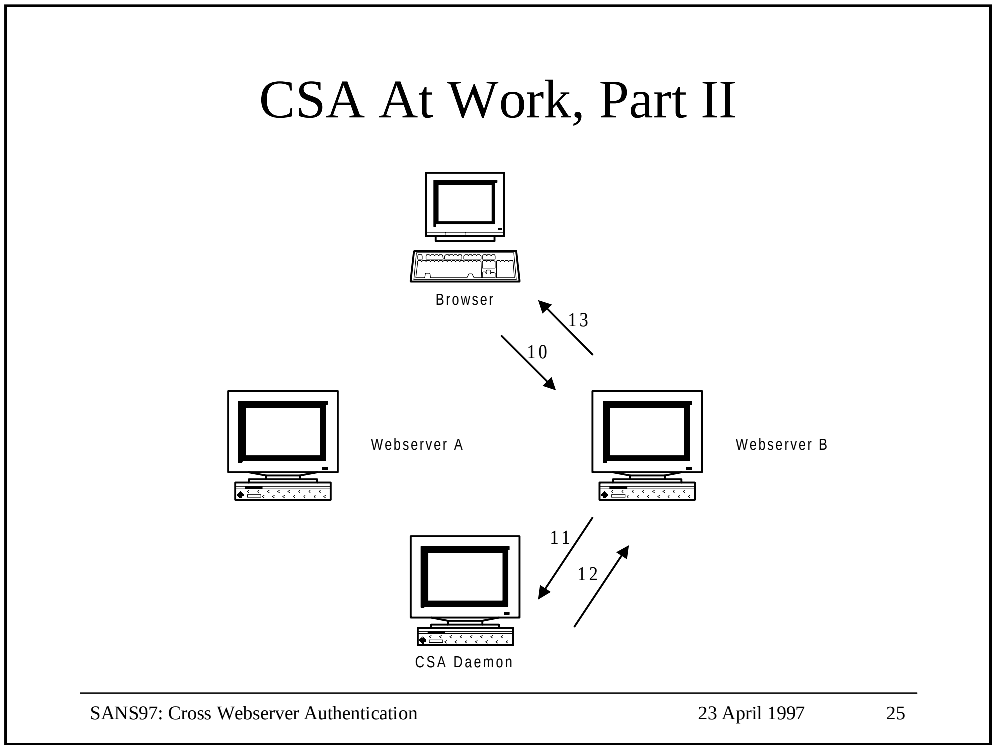
6. Webserver A encrypts the password it received with the request and compares it to the one in the user record. Assuming that it matches, Webserver A now generates a session record and cookies for this session.

7. Webserver A stores the session record in a local cache and sends a copy to the CSA Daemon.

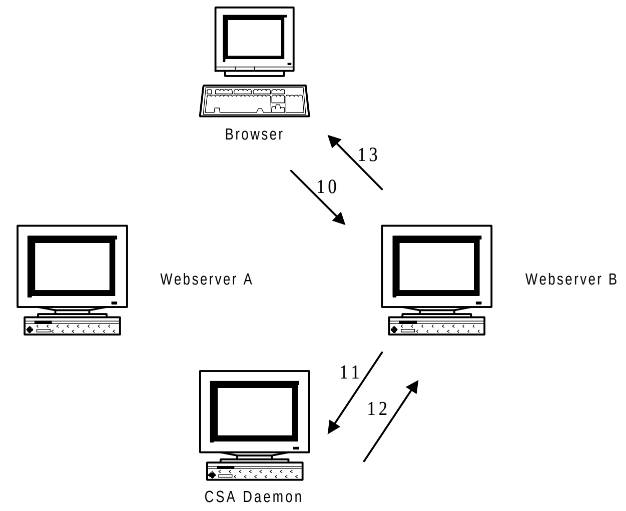
8. The auth routines setup special response headers, and the request is serviced normally with data returned to the browser.

9. The browser records the cookie data to submit with future requests.

User now browses around Webserver A for a while.



## CSA At Work, Part II



SANS97: Cross Webserver Authentication 23 April 1997 25

Now let’s see what happens when the user follows a link on Webserver A that takes them to a document on Webserver B. Assume that the user has not authenticated to Webserver B recently.

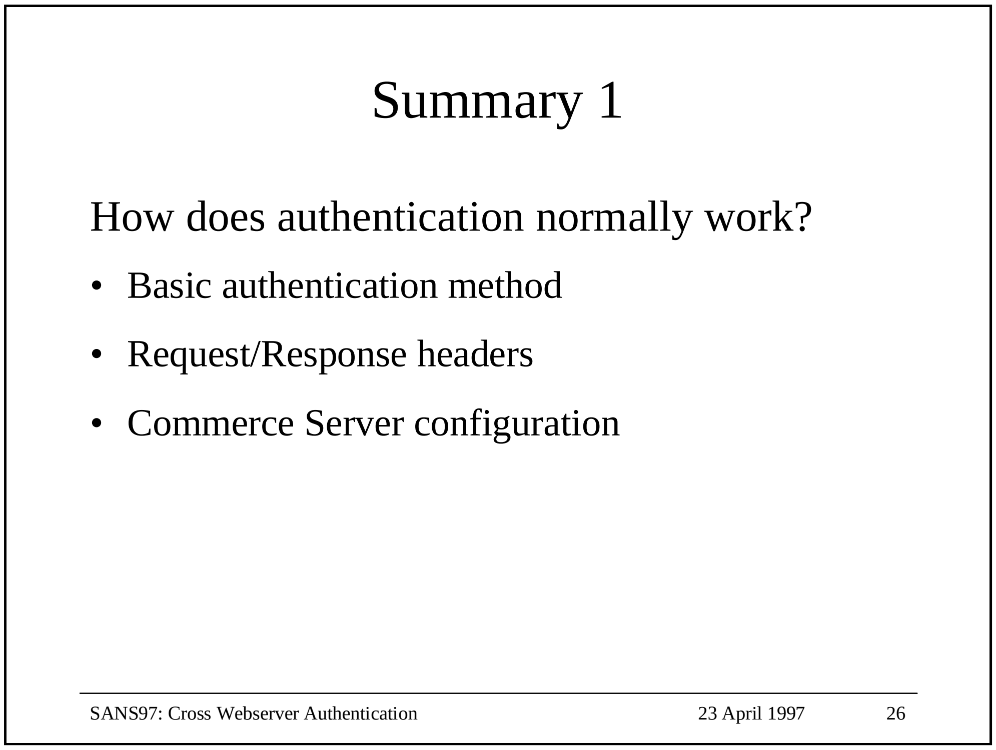
10. Browser submits a normal GET request. Since Webserver B is in the same DNS domain as Webserver A, the cookie that A set in the browser is sent to B in the request headers.

11. Webserver B examines the request for credentials - first cookies, then username and password. It finds a cookie, so it requests the corresponding session record from the CSA Daemon.

12. CSA Daemon returns the record.

13. Webserver B compares the “session” cookie it received in the request headers with that in the session record. If they do not match, an error is logged and the request is denied. If they do match, the session record is written to a local cache and the request is serviced normally.

User now browses around Webserver B without interference.

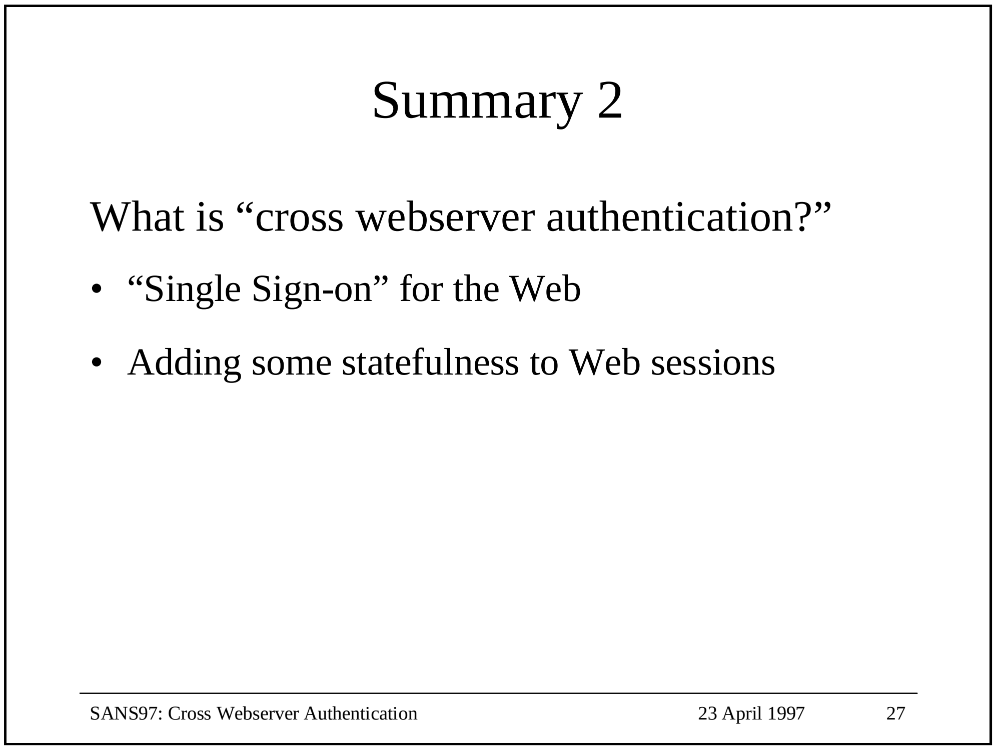


## Summary 1

How does authentication normally work?

- Basic authentication method
- Request/Response headers
- Commerce Server configuration

SANS97: Cross Webserver Authentication 23 April 1997 26

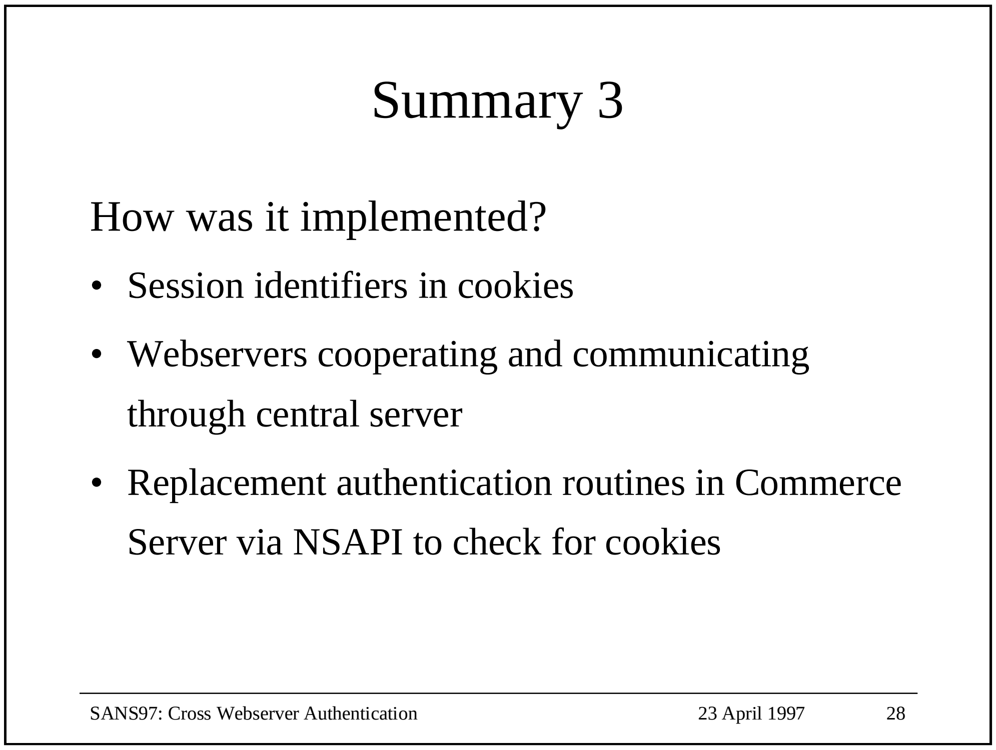


## Summary 2

What is “cross webserver authentication?”

- “Single Sign-on” for the Web
- Adding some statefulness to Web sessions

SANS97: Cross Webserver Authentication 23 April 1997 27

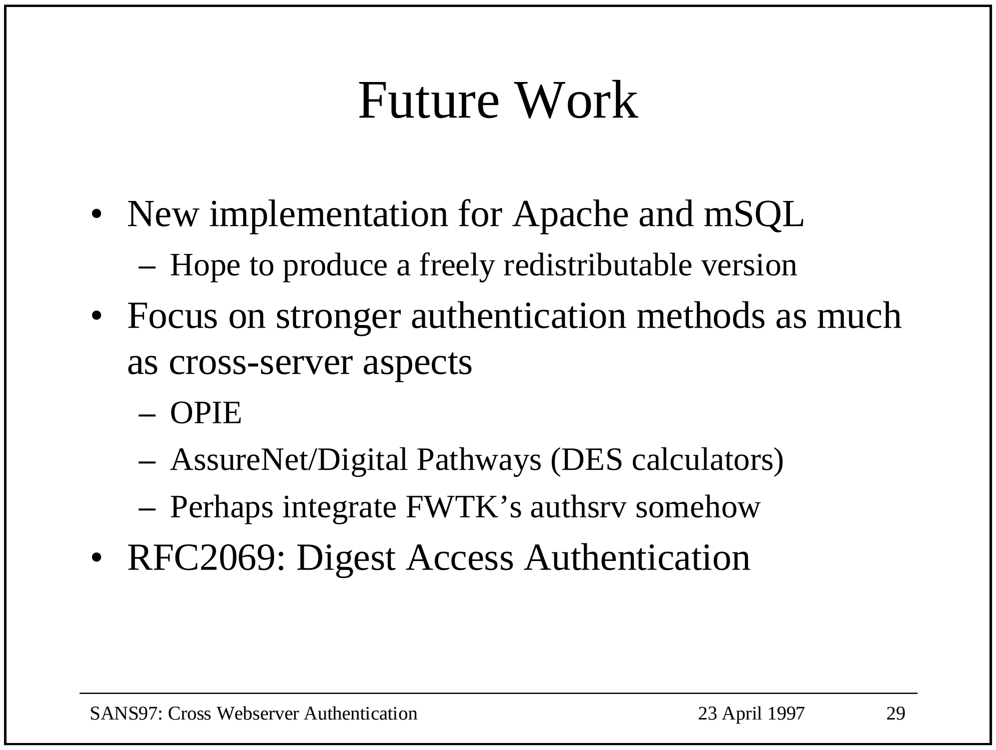


## Summary 3

How was it implemented?

- Session identifiers in cookies
- Webservers cooperating and communicating through central server
- Replacement authentication routines in Commerce Server via NSAPI to check for cookies

SANS97: Cross Webserver Authentication 23 April 1997 28

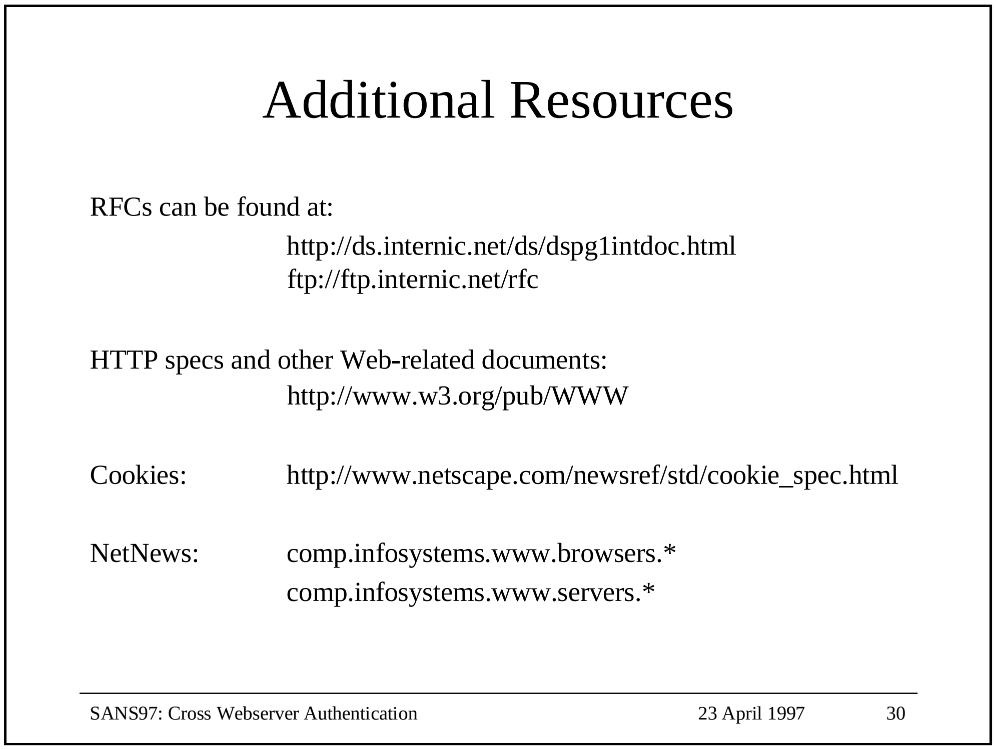


## Future Work

- New implementation for Apache and mSQL
  - Hope to produce a freely redistributable version
- Focus on stronger authentication methods as much as cross-server aspects
  - OPIE
  - AssureNet/Digital Pathways (DES calculators)
  - Perhaps integrate FWTK’s authsrv somehow
- RFC2069: Digest Access Authentication

SANS97: Cross Webserver Authentication 23 April 1997 29

As mentioned briefly in the note on slide 9, the digest authentication method is being developed as a remedy to the more obvious flaws in basic-auth. The authors describe it as a “weak access authentication method,” and it’s main difference is that the browser computes an hash or checksum of the password to send to the server instead of the plaintext. As the authors note, while not a vast improvement this is certainly better than off-the-shelf Telnet and FTP…



## Additional Resources

RFCs can be found at:

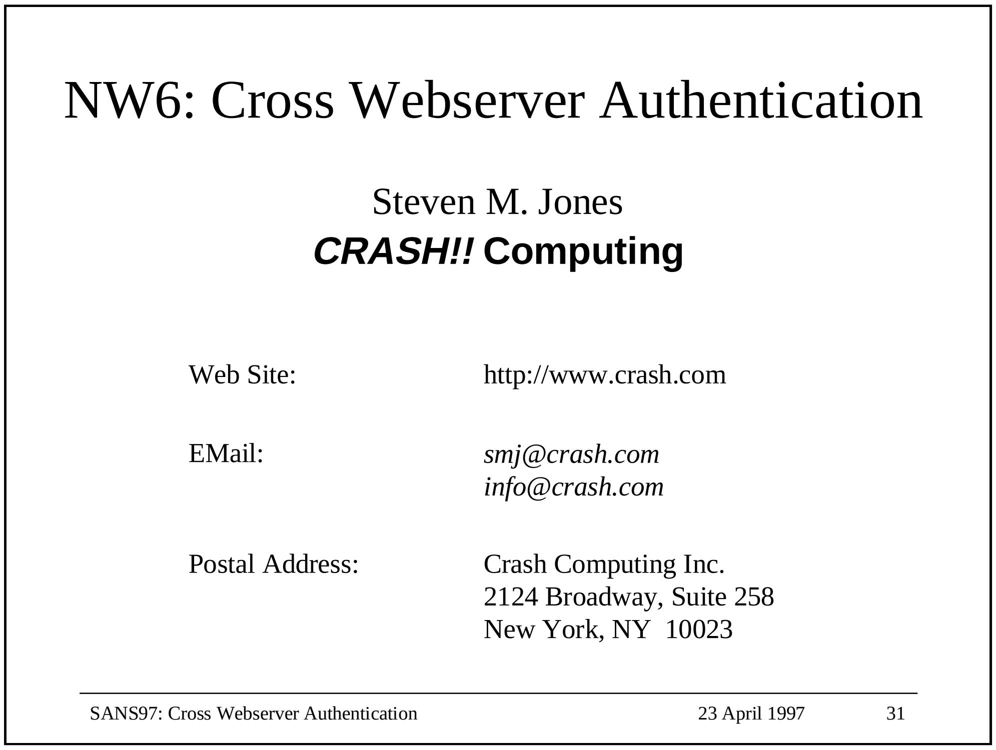
http://ds.internic.net/ds/dspg1intdoc.html
ftp://ftp.internic.net/rfc

HTTP specs and other Web-related documents:

http://www.w3.org/pub/WWW

Cookies: http://www.netscape.com/newsref/std/cookie_spec.html

NetNews: comp.infosystems.www.browsers.*
comp.infosystems.www.servers.*



# NW6: Cross Webserver Authentication

Steven M. Jones
***CRASH!!* Computing**

| | |
|---|---|
| Web Site: | http://www.crash.com |
| EMail: | *smj@crash.com*<br>*info@crash.com* |
| Postal Address: | Crash Computing Inc.<br>2124 Broadway, Suite 258<br>New York, NY 10023 |

SANS97: Cross Webserver Authentication 23 April 1997 31