Setting Up Cosign Clients

March 4, 2003

This presentation was originally given at an Academic Computing Support Forum (ACSF) meeting at the University of Michigan.

Agenda

- Cosign overview
- Configuring cosign on department web servers.
- Adapting commercial web-based applications to work with cosign.
- Questions and answer session.

[Most slides in this presentation include explanatory notes; be sure to make these notes visible when printing or viewing the slides.]

Cosign Overview

- Cosign is a web single sign on system written by the University of Michigan Web Services team.
- " Cosign" is short for for cookie signer.
- Cosign is the successor to cookieserver (1996 – 2003).
- Sessions have both idle and hard timeouts.
- Users can log out of all cosign-enabled web services by visiting a single URL.

•The cookies that cosign uses do not contain any sensitive information – they are just random strings. Thus the cookies are not at risk of being "cracked" or "broken".

•The cookies are session cookies which are only valid for the length of the browser session – they are not stored on the disk of the machine running the web browser.

Cosign Overview

- Cosign client web servers do not need to run SSL; sniffed cookies will compromise only the non-SSL-protected service, not the entire cosign infrastructure.
- Cosign is compatible with common SSL accelerators and clustering load balancers.

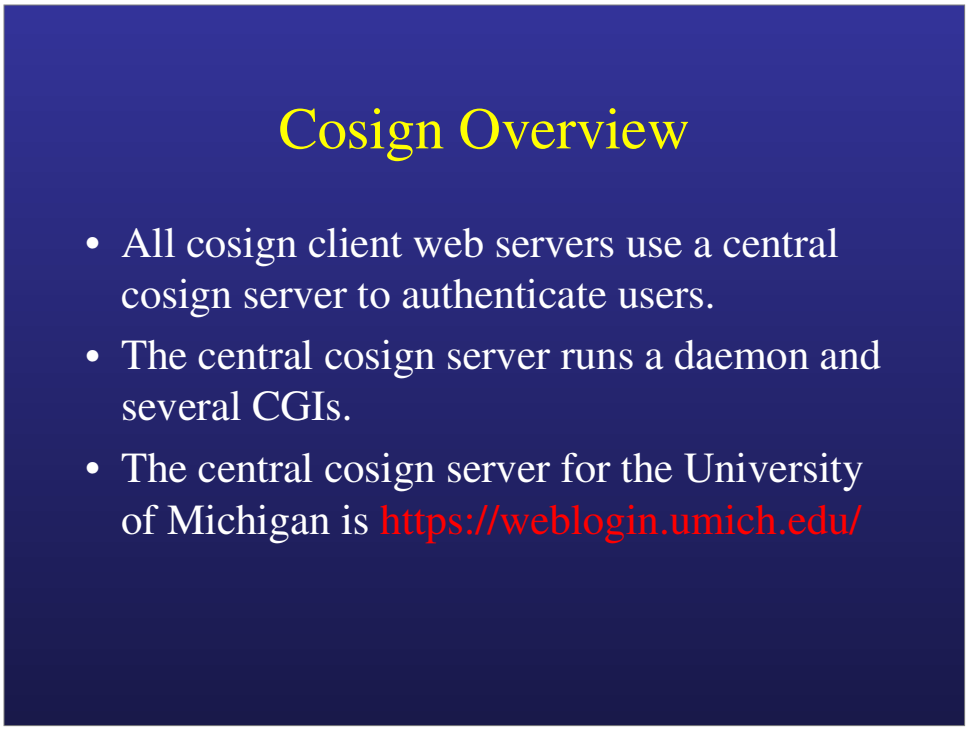
•The cookies used by cosign are not domain cookies. Thus, any web server can be set up as a cosign client regardless of DNS domain name.

•Wayne Wilson in the UofM Medical School is using cosign with SSL accelerators and load balancers.

Cosign Overview

- Each cosign client web server runs an authentication filter module. Currently, Apache 1.3.x and IIS web servers are supported. Development for Apache 2.x and J2EE is underway.
- A compromised cosign client web server does not represent a compromise of the cosign system as a whole.

•Note that a cosign client web server can optionally be configured to receive Kerberos tickets for end users from the central cosign server. If a cosign client web server that stores user Kerberos tickets is compromised, this is more severe than a cosign client web server which does not store user Kerberos tickets being compromised. Thus, cosign client web servers which receive user Kerberos tickets from the central cosign server should be configured to meet certain minimum security configuration criteria.



•Currently, a campus will have only a single central cosign server. The cosign design permits multiple central cosign servers which share and replicate state information for fault tolerance and load balancing purposes, however. Support for replicated central cosign servers will be added in a future release of cosign.

•The cosign daemon is used both by the CGIs on the central server as well as by the authentication filter module on the cosign client web servers.

Cosign Overview

- The central cosign server in turn authenticates users against Kerberos 5.
- Kerberos tickets can be passed back to the cosign client web servers. Kerberos 4 compatibility and GSSAPI are also supported.
- “ Friend” support is under development to permit self-created guest accounts.

Cosign Overview

- Users will be able to authenticate without using passwords via X.509 / KX.509 (note: this is not currently enabled on UofM's production central cosign server).
- Cosign is ultimately targeted as a component of the *Shibboleth* Internet 2 Middleware project.
- More information is available on the cosign home page: http://www.weblogin.org/

•Users can authenticate to cosign via X.509.

•Users can also do KX.509 retrieval of junk certificates, although this runs entirely on the user's machine.

•Cosign will also support KCT proxy retrieval of kerberos tickets using an SSL handshake.

•Note that X.509, KX.509, and KCT proxy retrieval are all independent and separable.

•Bill Doster of ITCS at the University of Michigan has successfully used cosign as a Shibolleth origin.

Configuring Cosign on Department Web Servers

- We'll demonstrate how to install and configure cosign on a client web server to protect stand-alone CGIs, static web pages, ASPs, and so on.
- The demonstration is on an LSA web server running Apache 1.3.26. The server is a Sun Netra T1 AC200 running Solaris 8 in 64-bit mode.

•The Solaris web servers for the College of Literature, Science, and the Arts (LSA) at the University of Michigan are not set up in a standard way. Thus, some of the commands that follow are specific to the LSA environment, and installing cosign is made more difficult by the "non-vanillia" nature of the web server machine. Any LSA-specific commands will be noted and the normal, standard commands will also be given.

Configuring Cosign

- Get and unpack the software (check http://www.weblogin.org/ for the latest version):

```
mkdir /var/tmp/cosign-build
cd /var/tmp/cosign-build

wget http://www.umich.edu/~umweb/downloads/
  cosign-0.9.3.tar.gz

md5 cosign-0.9.3.tar.gz

tar zxf cosign-0.9.3.tar.gz

cd cosign-0.9.3
```

•Check http://www.weblogin.org/ to be sure that 0.9.3 is still the latest version of cosign before running the commands above.

•Be sure to verify that the MD5 fingerprint of the file you download matches the MD5 fingerprint given on the web page.

Configuring Cosign

- Run the “configure” program. “configure” will examine your system to figure out how to compile cosign. On most systems, we’d just be able to type:

```
./configure
```

- For our system, “configure” needs some help to figure out some non-standard things about the environment.

•“ ./configure” should just work on most systems. It’s a bug in cosign if “configure” fails on ordinary, vanillia, systems.

•Typing “./configure –help” will show you all of the possible command line options to configure. Additional information is also available with the documentation that comes as a part of the cosign source code distribution (particularly the README file).

Configuring Cosign

- Run the “configure” program:

```
env CC="cc -fast -xtarget=generic64" CFLAGS="-v" \
 LDFLAGS="-L/opt/lib/sparcv9 -R/opt/lib/sparcv9" \
 ac_cv_path_install="/usr/ucb/install" \
 ./configure \
 --prefix="/opt/packages/cosign-0.9.3" \
 --libdir="/opt/packages/cosign-0.9.3/lib/sparcv9" \
 --enable-shared --disable-static \
 --with-ssl="/opt" --with-apache-
apxs="/opt/bin/apxs"
```

•The configure command shown does not enable Kerberos; if a cosign protected service needs to receive a Kerberos ticket granting ticket for the user from the central cosign server, add the option “—enable-krb”.

•The environment variables in the “env” command are to get configure to use the Sun compiler in preference to gcc, to produce 64-bit executables, and to look in the right places for 64-bit libraries (64 and 32 bit libraries are stored in different directories under Solaris).

•ac_cv_path_install is because configure finds /usr/vice/bin/install in our path before /usr/ucb/install. /usr/vice/bin/install is a part of the AFS client software and is not a BSD-compatible install program.

•“— prefix” tells configure where to install cosign (it installs by default in /usr/local, which is used for other purposes in LSA); “--libdir” is because we're producing 64-bit libraries which need to be installed separately from the 32-bit ones.

•“ --enable-shared” and “--disable-static” produce a shared-only version of libsnet. LSA's Apache doesn't like linking the static version of libsnet into the filter unless it's compiled with -KPIC. Using the shared libsnet works around this problem. (libsnet is a high-level networking library for implementing networked services; it includes TLS support. Besides cosign, libsnet is also used by other software packages, including radmind).

•“ --with-ssl” and “--with-apache” help configure find things that we've got installed in non-standard places in the LSA environment.

Configuring Cosign

- Build the cosign authentication filter:

```
make
```

Configuring Cosign

- Normally, installing cosign is simple:

```
make install
```

- However, due to the specialized environment on the LSA webservers, we'll need to install cosign by hand.

Configuring Cosign

- Install the libsnet library (normally libsnet is created as a static library that does not need to be installed separately, but we're using a special Apache DSO setup on this system, so…)

```
cd libsnet
make install
cd /opt
makelinks --prefix="/opt" packages/cosign-0.9.3
chown -R bin:bin /opt/packages/cosign-0.9.3
chmod -R ugo+rX /opt/packages/cosign-0.9.3
```

* "makelinks" is a symbolic link management script used in LSA which is similar to GNU Stow. It will create symlinks in /opt/lib/sparcv9 for all files in /opt/packages/cosign-0.9.3/lib/sparcv9. LSA's Apache then looks for shared libraries under /opt/lib/sparcv9.

Configuring Cosign

- Install the cosign authentication filter (normally "make install" would run "apxs" to install this, but installing via apxs does not work in our environment because our httpd.conf file is in a special location):

```
cd /var/tmp/cosign-build
cd cosign-0.9.3/filters/apache
cp mod_cosign.so /opt/www/apache/libexec
chown root:other /opt/www/apache/libexec/*
chmod 755 /opt/www/apache/libexec/*
```

Configuring Cosign

- This completes the work normally done by “make install”.
- The steps on the following slides are in addition to what is done by “make install” and always need to be done by hand.

Configuring Cosign

- Cosign expects to store cookies in the directory "/var/cosign/filter". This directory must be owned by the user the web server runs as – in our case, user "**http**".

```
mkdir /var/cosign
chown root:other /var/cosign
chmod 755 /var/cosign

mkdir /var/cosign/filter
chown http:other /var/cosign/filter
chmod 700 /var/cosign/filter
```

•Be sure to change "http" above to be the user as which the web server runs.

Configuring Cosign

- Cosign needs to know about additional Certificate Authorities (CAs), in particular the UMWeb CA. If you do not already have a directory to store CA information, create one:

```
mkdir /opt/www/etc/ssl.ca
chown root:other /opt/www/etc/ssl.ca
chmod 755 /opt/www/etc/ssl.ca
```

•The directory does not need to be called “ssl.ca”, you can name it anything you wish.

Configuring Cosign

- Install the CA certificates that come as a part of the cosign source code distribution (UMWeb, Entrust, Verisign, and RSA Data Security certs):

```
cd /var/tmp/cosign-build/cosign-0.9.3
cp CAcerts/*.pem /opt/www/etc/ssl.ca
cd /opt/www/etc/ssl.ca
chown root:other *.pem
chmod 644 *.pem
c_rehash /opt/www/etc/ssl.ca
```

•The UMWeb CA information is necessary to verify the identity of the central cosign web server; the other CAs are not as necessary.

•The “c_rehash” command is a part of the OpenSSL package and is used to create symbolic links with hashed values of the CA certificates.

Configuring Cosign

- The cosign client web server and the central cosign server need to be able to verify each other's identity. Send a Certificate Signing Request (CSR) for your web server to the UMWeb team and they will send you back a cosign certificate.
- In this example, we send the /opt/www/etc/ssl.csr/server.csr file.

•The CSR you send can be the same one that you used to get a commercially signed SSL certificate from Versign, Entrust, or another commercial company. If you use OpenSSL, you can generate a CSR by running

```
openssl genrsa –out server.key 1024
openssl req –new –key server.key –out server.csr
```

(you'll need to install the private key file server.key if you generate a CSR this way, of course).

Configuring Cosign

- The UMWeb team also needs a *service name* for the service provided by your cosign client web server. You should email a requested service name along with the CSR.
- The service name might be shared by multiple cosign client web servers all providing the same web based service.

Configuring Cosign

- Note that the service name you choose will appear as a part of the names of the cookies that cosign sets in users' browsers.
- In this example, we'll request "lsa-test" as a service name. Cosign service cookies for this service will therefore be named "cosign-lsa-test".

Configuring Cosign

- Install the cosign client certificate sent to you by the UMWeb team. This cert is normally installed in the same directory as any other certificates your web server has:

```
cd /opt/www/etc/ssl.crt
cp /path/to/saved/cert cosign.crt
chown root:other cosign.crt
chmod 600 cosign.crt
make clean ; make     # or run c_rehash
```

•The makefile in this directory is one that is installed by mod_ssl for Apache in order to manage certificates. Any other way of recreating the certificate hashes, such as the OpenSSL c_rehash program, can also be used.

Configuring Cosign

- Optionally, you can check to verify that the UMWeb CA and cosign certificate are both set up correctly. This can avoid problems later; if the command below does not work, cosign will not work either.

```
openssl verify -verbose \
  -CApath /opt/www/etc/ssl.ca \
  /opt/www/etc/ssl.crt/cosign.crt
```

Configuring Cosign

- Add the necessary cosign configuration directives to your httpd.conf file.
- Verify that the following two lines were added to httpd.conf by apxs when you ran “make install”; add them if they are not there.

```
LoadModule cosign_module libexec/mod_cosign.so
AddModule mod_cosign.c
```

•Since in LSA we did not run “make install” for cosign, these lines have to be added manually.

Configuring Cosign

- Add the following lines to the section defining your SSL-protected virtual host configuration:

```
CosignHostname weblogin.umich.edu
CosignRedirect https://weblogin.umich.edu/
CosignPostErrorRedirect https:
//weblogin.umich.edu/post_error.html
CosignService lsa-test
CosignCrypto /opt/www/etc/ssl.key/server.key
/opt/www/etc/ssl.crt/cosign.crt
/opt/www/etc/ssl.ca
CosignProtected On
```

•Change the service name above to be the service name you chose or were assigned.

•Also adjust the path to the SSL server key, the path to the cosign certificate, and the path to the CA info directory as necessary in the CosignCrypto directive.

•The cosign configuration directives should be in the main part of the virtual host configuration, not in a Directory or Location context.

Configuring Cosign

- Add the following line to the Directory stanza(s), Location stanza(s), and/or .htaccess file(s) for the resources you want to protect with cosign:

```
AuthType Cosign
```

•The AuthType directive may not be necessary depending on how you have your web server configured, but it shouldn't hurt. In LSA we use multiple authentication filters and so this directive is necessary to distinguish between them.

Configuring Cosign

- Restart the web server to get the changes to httpd.conf to take effect:

```
/etc/init.d/httpd stop
/etc/init.d/httpd start
```

Configuring Cosign

- For this example, we just protected a simple stand-alone CGI:

  https://latimer.lsait.lsa.umich.edu/test/printenv

•Note that the above URL was set up for the presentation and may no longer work. It was a script that printed the date and time, provided a link to the logout CGI on the cosign server (https://weblogin.umich.edu/cgi-bin/logout), and then dumped its environment variables.

•When going to the URL above, the cosign authentication filter on the client web server (latimer) will notice that you are not authenticated and redirect you to the login CGI on the central cosign server. The login CGI will see that you don't have a cosign cookie and display a "splash screen" saying that you need to log in by going to https://weblogin.umich.edu; it will also provide a link to this URL. When you follow the link, you will receive a form asking for your username and password. After you submit the form, cosign will then authenticate you and redirect you back to the URL on the client web server from which you originally came.

•In the output of the CGI above, the AUTH_TYPE environment variable will be set to "Cosign". Scripts can use this to verify that cosign authentication ocurred. The REMOTE_USER variable gives the identity of the authenticated user, and the REMOTE_REALM variable indicates which Kerberos realm the central cosign server authenticated the user against.

Adapting Commercial Web-Based Applications to Work with Cosign

- The difficulty of getting a commercial web-based application to work with cosign can vary greatly depending upon what assumptions are made by the commercial application.
- Educating the software vendor about cosign and getting their “buy in” is very important and can greatly increase the chances of successfully adapting the commercial application to work with cosign.

Commercial Applications

- The simplest scenario is if the commercial web-based application already relies upon the web server for authentication. For example, an application might use Apache's mod_auth authentication filter, .htpasswd files, and use the REMOTE_USER environment variable to determine the identity of the authenticated user.
- In this case, no changes need to be made to the application; just change the web server configuration to replace the existing authentication directives (e.g., mod_auth) with the corresponding cosign authentication directives.

Commercial Applications

- Unfortunately, many web based applications will provide their own authentication code rather than leveraging the authentication capabilities of the underlying web server.
- Although cosign will not conflict with this type of application, users would be asked to authenticate twice, once to cosign and once to the application, defeating the purpose of cosign as a web single sign on mechanism.
- Thus, these applications should be modified to disable their internal authentication mechanism.
- The following strategy should work with both Apache as well as IIS.

Commercial Applications

- Basic strategy:
  - SSL-protect the entire application (optional but recommended for higher security).
  - Set up and configure cosign normally to protect all URLs used by the application as we did in the demo.
  - Modify the application's login web page to get the user's identity from the REMOTE_USER environment variable rather than from an HTML form field. After "injecting" our own value at this point, we allow the application to keep track of it from then on via its own internal mechanisms.

Commercial Applications

- Basic strategy:
  - Anywhere the application uses an HTML form field to prompt for a password (the application's login page and elsewhere), change the type of the password form field to "hidden" and give it a dummy value. This will prevent the user from being prompted for their password by the application. Any authentication related text displayed by the application (e.g., "enter your password below") should also be modified or deleted.

•This is just a suggested strategy. It has the advantage of minimizing the number of changes to the commercial application's code. Other strategies are possible – for example, eliminating the HTML form field, or getting rid of the application's login page entirely.

Commercial Applications

- Basic strategy (continued):
  - Modify any authentication checks performed by the application so that they unconditionally succeed. Very often, there will be a single authentication function called by all of the application's web pages which is easy to modify. This function will continue to receive a dummy value for the user's password from the hidden form fields, but rather than checking the "password" the commercial application's authentication function will be modified to simply ignore it.

Commercial Applications

- Basic strategy (continued):
  - Leave any *authorization* checks alone – for example, in addition to verifying the user's password, the application's central "authentication" function may also check to see if the user is in a local database, and it should continue to deny access to any user who is not in the database.

Commercial Applications

- Basic strategy (continued):
  - Optionally, modify the application's logout web page to redirect to the cosign logout CGI (https://weblogin.umich.edu/cgi-bin/logout) after doing any application-specific cleanup. Or you may want to have the application's logout button log the user out of the application only, and provide a second logout button to log the user out of both the application and cosign together.

* It's not possible (or meaningful) to have cosign log the user out of a single cosign protected service – since the user still has the cosign cookie used for all services (named "cosign"), if they re-visit any page of the service, they will automatically be granted another cosign service cookie (e.g., "cosign-lsa-test") without being asked to re-authenticate again. This is the whole point of cosign as a web single sign on technology. Therefore it's useful to provide two logout buttons – one which just performs any actions the commercial application needs (checking in licenses, deleting temporary files on the web server, etc.) and a second button which performs the same action but also redirects the user to the central cosign server logout URL which will destroy their cosign credentials for all cosign-enabled services (note that cosign won't actually perform service-specific logout actions such as checking in licenses for these services – it merely destroys the cosign credentials used to access the service).

Commercial Applications

- If a user accesses a cosign protected web page before authenticating to cosign, cosign will send them to a “splash screen” which will inform them that they need to authenticate. The purpose of this is to allow the user to more easily verify that they are providing their password to the central cosign server (as opposed to some other server) and prevent spoofing attacks. This is called “entering from the side”.

Commercial Applications

- A user can avoid the splash screen by authenticating to cosign before attempting to access a cosign protected service. This is known as “entering from the top”.
- Optionally, to provide a smoother user experience, you may want to use URL rewriting to send the user to the cosign login CGI when they access the commercial web based application. If they are already authenticated, they will receive an “authentication succeeded” message; if not, they will get the cosign login screen directly, avoiding the splash screen. Your URL rewriting rule can provide the cosign login CGI with a URL, such as the URL for the application’s login page, and cosign will redirect the user to this URL after authenticating them.

Case Study

- FootPrints is a commercial web based help desk / trouble ticket application from UniPress software (http://www.unipress.com).
- FootPrints version 5.5 does its own authentication via CGIs. The CGIs check user names and passwords against either an internal database, the Unix /etc/passwd and /etc/shadow files, a Windows NT domain controller, or an LDAP / Active Directory server.

Case Study

- FootPrints is written mostly in Perl and hence was easy to modify – it is not a “black box”; this was a factor in LSA’s purchasing descision since LSA wants all web based applications to integrate into the University’s cosign environment.
- The FootPrints vendor, UniPress software, is open to new ideas and responsive to customer needs, which made adding support for cosign to FootPrints much easier.

Case Study

- Working with UniPress, LSA added a fifth authentication type to FootPrints: web server authentication, where FootPrints relies upon the web server to correctly authenticate the user before any page is served or CGI is invoked. This not only enables FootPrints to use cosign, but also enables it to use any other web server authentication filter which other customers may want to use.

* Pitching the proposed change to the vendor as “adding support for web server based authentication” rather than “adding support for cosign” helped get the vendor to agree to the change since it is something that could then be useful to a broder customer base.

Case Study

- We followed the basic strategy outlined earlier for adapting commercial web based applications to work with cosign.
- Fully integrating FootPrints with cosign required changes to only 139 lines of Perl code (the total size of the patch to make these changes is 481 lines). This is a minor change, considering that the total size of FootPrints is over 327,000 lines. Aproximately half of the changes were purely cosmetic – hiding username and password form fields from the user and changing explanatory text.

•Of course, it took a while to learn enough about the internals of FootPrints to determine *which* lines needed to be changed.

Case Study

- UniPress software has accepted the “web server based authentication” patch, and this functionality will be included as a standard feature in the next major release of FootPrints later this year.

Questions?