scripts.mit.edu

Quentin Smith Geoffrey Thomas
scripts@mit.edu

Student Information Processing Board

October 28, 2008

Outline

Outline

Outline

Outline

Apache

- Everyone wants Apache
- Apache's default configuration isn't safe for scripting
- Scripting *requires* code execution—mod_php, mod_perl, mod_python
- Apache normally runs everything as apache/nobody
- How to secure?

Apache

- Everyone wants Apache
- Apache's default configuration isn't safe for scripting
- Scripting *requires* code execution—mod_php, mod_perl, mod_python
- Apache normally runs everything as apache/nobody
- How to secure?
- suEXEC—allows Apache to spawn a process as the user...
- ...even for static content!

suEXEC

- setuid program
- Passed the request by Apache
- Verifies that the script is in the `web_scripts` directory
- Switches to the uid of the file and executes
- Even for static files!

Postfix

- Standard Postfix server
- No local mailboxes
- All mail is passed to procmail

```
mailbox_command = /usr/bin/procmail -t \
-a "${EXTENSION}" ~/mail_scripts/procmailrc
```

procmail

- Reads `~/mail_scripts/procmailrc` from user's home directory
- Users can do whatever they want with messages
- AFS causes problems—No way to know if failure is temporary (file server is down) or permanent (user isn't signed up for mail scripts)
- All procmail failures are treated as temporary, so mail is queued

Cron (cronie)

- Crontabs are currently stored locally on scripts servers
- `cronload` command loads the crontabs from `~/cron_scripts/crontab`

Cron (cronie)

- Crontabs are currently stored locally on scripts servers
- `cronload` command loads the crontabs from `~/cron_scripts/crontab`
- Needs improvement
- Cron does not fail over with Web and Mail
- Plan to move crontabs into AFS and do hot failover

sql.mit.edu

Though scripts.mit.edu makes use of sql.mit.edu, it's a separate SIPB service with different maintainers.

- sql.mit.edu provides MySQL databases to scripts users and anyone else
- SQL data is stored locally, replicated across multiple servers
- Nightly backups go into AFS

SVN and Git hosting

- New service (September 2008), not well documented
- svn://*username*.scripts.mit.edu/ and git://*username*.scripts.mit.edu/
- Uses suEXEC to run a svnserve / git-daemon as the user
- /mit/*username*/Scripts/{svn,git}
- git:// is read-only, so future plans for svn+ssh:// and git+ssh://

Outline

AFS access controls

- AFS enforces server side access controls.
- On Athena systems: user's password → Kerberos tickets → AFS tokens, which authenticate the client to the AFS server.
- On scripts, we don't have the user's password or tickets.
- User's scripts are not publicly readable.
- Access is controlled through a single `daemon.scripts` AFS user.

Isolating users on scripts

- If all users share `daemon.scripts` AFS tokens, how are they prevented from accessing each other's `web_scripts`?
- On scripts, we enforce additional restrictions in the AFS kernel module.
  - `afsAccessOK()` in `openafs/src/afs/VNOPS/afs_vnop_access.c`

You can only use `daemon.scripts` credentials to access files in a volume with volume ID equal to your UID,

```
 int
 afs_AccessOK(struct vcache *avc, afs_int32 arights,
              struct vrequest *areq, afs_int32 check_mode_bits)
 {
     ...
+    if (!(areq->realuid == avc->fid.Fid.Volume) &&
+        !((avc->anyAccess | arights) == avc->anyAccess) &&
+        !(arights == PRSFS_LOOKUP && areq->realuid == HTTPD_UID) &&
+        !(arights == PRSFS_LOOKUP && areq->realuid == POSTFIX_UID) &&
+        !(arights == PRSFS_READ && areq->realuid == HTTPD_UID &&
+          avc->m.Mode == 0100777) &&
+        !(PRSFS_USR3 == afs_GetAccessBits(avc, PRSFS_USR3, areq) &&
+          areq->realuid == 0) &&
+        !(PRSFS_USR4 == afs_GetAccessBits(avc, PRSFS_USR4, areq) &&
+          (areq->realuid == 0 || areq->realuid == SIGNUP_UID))) {
+       return 0;
+    }
```

or the file is `system:anyuser` readable anyway,

```
 int
 afs_AccessOK(struct vcache *avc, afs_int32 arights,
              struct vrequest *areq, afs_int32 check_mode_bits)
 {
     ...
+    if (!(areq->realuid == avc->fid.Fid.Volume) &&
+        !((avc->anyAccess | arights) == avc->anyAccess) &&
+        !(arights == PRSFS_LOOKUP && areq->realuid == HTTPD_UID) &&
+        !(arights == PRSFS_LOOKUP && areq->realuid == POSTFIX_UID) &&
+        !(arights == PRSFS_READ && areq->realuid == HTTPD_UID &&
+          avc->m.Mode == 0100777) &&
+        !(PRSFS_USR3 == afs_GetAccessBits(avc, PRSFS_USR3, areq) &&
+          areq->realuid == 0) &&
+        !(PRSFS_USR4 == afs_GetAccessBits(avc, PRSFS_USR4, areq) &&
+          (areq->realuid == 0 || areq->realuid == SIGNUP_UID))) {
+       return 0;
+    }
```

or the apache or postfix users are doing a stat(),

```
 int
 afs_AccessOK(struct vcache *avc, afs_int32 arights,
              struct vrequest *areq, afs_int32 check_mode_bits)
 {
     ...
+    if (!(areq->realuid == avc->fid.Fid.Volume) &&
+        !((avc->anyAccess | arights) == avc->anyAccess) &&
+        !(arights == PRSFS_LOOKUP && areq->realuid == HTTPD_UID) &&
+        !(arights == PRSFS_LOOKUP && areq->realuid == POSTFIX_UID) &&
+        !(arights == PRSFS_READ && areq->realuid == HTTPD_UID &&
+          avc->m.Mode == 0100777) &&
+        !(PRSFS_USR3 == afs_GetAccessBits(avc, PRSFS_USR3, areq) &&
+          areq->realuid == 0) &&
+        !(PRSFS_USR4 == afs_GetAccessBits(avc, PRSFS_USR4, areq) &&
+          (areq->realuid == 0 || areq->realuid == SIGNUP_UID))) {
+       return 0;
+    }
```

or the apache user is trying to read a file with mode 777,

```
 int
 afs_AccessOK(struct vcache *avc, afs_int32 arights,
              struct vrequest *areq, afs_int32 check_mode_bits)
 {
     ...
+    if (!(areq->realuid == avc->fid.Fid.Volume) &&
+        !((avc->anyAccess | arights) == avc->anyAccess) &&
+        !(arights == PRSFS_LOOKUP && areq->realuid == HTTPD_UID) &&
+        !(arights == PRSFS_LOOKUP && areq->realuid == POSTFIX_UID) &&
+        !(arights == PRSFS_READ && areq->realuid == HTTPD_UID &&
+          avc->m.Mode == 0100777) &&
+        !(PRSFS_USR3 == afs_GetAccessBits(avc, PRSFS_USR3, areq) &&
+          areq->realuid == 0) &&
+        !(PRSFS_USR4 == afs_GetAccessBits(avc, PRSFS_USR4, areq) &&
+          (areq->realuid == 0 || areq->realuid == SIGNUP_UID))) {
+       return 0;
+    }
```

or the root or signup users are accessing file with the special D or E bits.

```
 int
 afs_AccessOK(struct vcache *avc, afs_int32 arights,
              struct vrequest *areq, afs_int32 check_mode_bits)
 {
     ...
+    if (!(areq->realuid == avc->fid.Fid.Volume) &&
+        !((avc->anyAccess | arights) == avc->anyAccess) &&
+        !(arights == PRSFS_LOOKUP && areq->realuid == HTTPD_UID) &&
+        !(arights == PRSFS_LOOKUP && areq->realuid == POSTFIX_UID) &&
+        !(arights == PRSFS_READ && areq->realuid == HTTPD_UID &&
+          avc->m.Mode == 0100777) &&
+        !(PRSFS_USR3 == afs_GetAccessBits(avc, PRSFS_USR3, areq) &&
+          areq->realuid == 0) &&
+        !(PRSFS_USR4 == afs_GetAccessBits(avc, PRSFS_USR4, areq) &&
+          (areq->realuid == 0 || areq->realuid == SIGNUP_UID))) {
+       return 0;
+    }
```

Serving static content

- The `apache` user does not have permission to read the user's files directly.
- Both static and dynamic content is served through suEXEC.

1. `/etc/httpd/conf.d/execsys.conf` is configured to serve static content with the `cgi-script` handler.

```
<Files *.pl>
        SetHandler cgi-script
        Options +ExecCGI
</Files>
<Files *.php>
        SetHandler cgi-script
        Options +ExecCGI
</Files>
...
<Files *.html>
        SetHandler cgi-script
        Options +ExecCGI
</Files>
<Files *.css>
        SetHandler cgi-script
        Options +ExecCGI
```

2. openafs/src/afs/VNOPS/afs_vnop_access.c is modified to mark *all* files as executable (!).

```
int
afs_access(OSI_VC_DECL(avc), register afs_int32 amode,
           struct AFS_UCRED *acred)
{
    register afs_int32 code;
    struct vrequest treq;
    struct afs_fakestat_state fakestate;
    OSI_VC_CONVERT(avc);

    AFS_STATCNT(afs_access);
+   amode = amode & ~VEXEC;
    afs_Trace3(afs_iclSetp, CM_TRACE_ACCESS, ICL_TYPE_POINTER, avc,
               ICL_TYPE_INT32, amode, ICL_TYPE_OFFSET,
               ICL_HANDLE_OFFSET(avc->m.Length));
    ...
}
```

3. `httpd/support/suexec.c` is modified to dispatch static content to `/usr/local/bin/static-cat`.

```
+#define STATIC_CAT_PATH "/usr/local/bin/static-cat"
+static const char *static_extensions[] = {
+    "html",
+    "css",
+    ...
+}
+
 int main(int argc, char *argv[])
 {
     ...
+    if (is_static_extension(cmd)) {
+        argv[2] = STATIC_CAT_PATH;
+        execv(STATIC_CAT_PATH, &argv[2]);
+        log_err("(%d)%s: static_cat exec failed (%s)\n", errno,
+                strerror(errno), argv[2]);
+        exit(255);
```

Group locker support

- "Users" on scripts are actually lockers.
- User IDs are actually locker volume IDs.

Group locker support

- "Users" on scripts are actually lockers.
- User IDs are actually locker volume IDs.
- Kerberos is modified to let users SSH in as any locker they administrate.
  - Replaced the `.k5login` mechanism: `krb5_kuserok()` in `krb5/src/lib/krb5/os/kuserok.c`
  - Calls a Perl script `/usr/local/sbin/admof` to do the actual check.

```
krb5_boolean KRB5_CALLCONV
krb5_kuserok(krb5_context context, krb5_principal principal,
             const char *luser)
{
     ...
+    if ((pid = fork()) == -1) {
+       free(princname);
+       return(FALSE);
+    }
+    if (pid == 0) {
+#define ADMOF_PATH "/usr/local/sbin/ssh-admof"
+        exec(ADMOF_PATH, ADMOF_PATH, (char *) luser, princname, NULL);
+        exit(1);
+    }
+    if (waitpid(pid, &status, 0) > 0 && WIFEXITED(status) &&
+        WEXITSTATUS(status) == 33) {
+        isok = TRUE;
+    }
     ...
```

LDAP data

- All user-specific information is stored in LDAP records
- Each scripts server runs a local LDAP daemon with multi-master replication
- Each user has a `posixAccount` and at least one `apacheConfig` and `scriptsVhost`
- Users can request additional virtual hosts
- We hope to create a web interface (phase 1 of "scripts-pony") for users to create virtual hosts in the `*.user.scripts.mit.edu` namespace

Apache modules

- We make it easy to do authentication against MIT certificates.
- Both `https://scripts-cert.mit.edu`, and port `444` on any scripts hostname, are configured to request client certificates.
- `mod_ssl` provides the `SSL_CLIENT_S_DN_Email` environment variable, but does not integrate with the Apache authentication and authorization framework.
- Wrote a collection of Apache modules to make this cleaner.

mod_auth_sslcert

- mod_auth_sslcert passes the SSL_CLIENT_S_DN_Email variable to the Apache authorization handlers.

```
AuthType SSLCert
AuthSSLCertVar SSL_CLIENT_S_DN_Email
AuthSSLCertStripSuffix "@MIT.EDU"
```

mod_authz_afsgroup

- mod_authz_afsgroup does Apache authorization based on AFS groups.

```
Require afsgroup system:scripts-team
```

mod_auth_optional

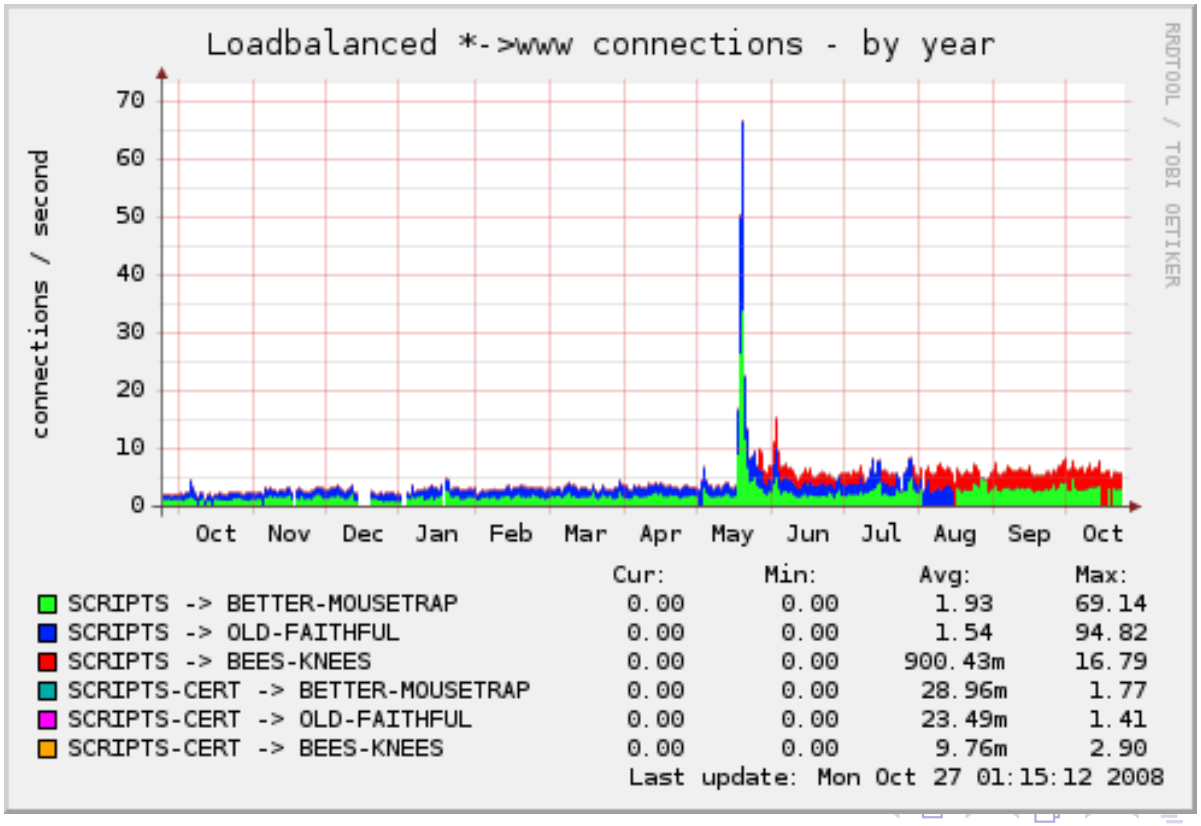
- mod_auth_optional subverts the authorization process to allow you to serve different pages to users with certificates and users without certificates.

Linux Virtual Server

- Provides high availability and load balancing
- `heartbeat` provides failover between LVS "directors"
- `ldirectord` keeps track of online scripts servers and chooses destination server for each request

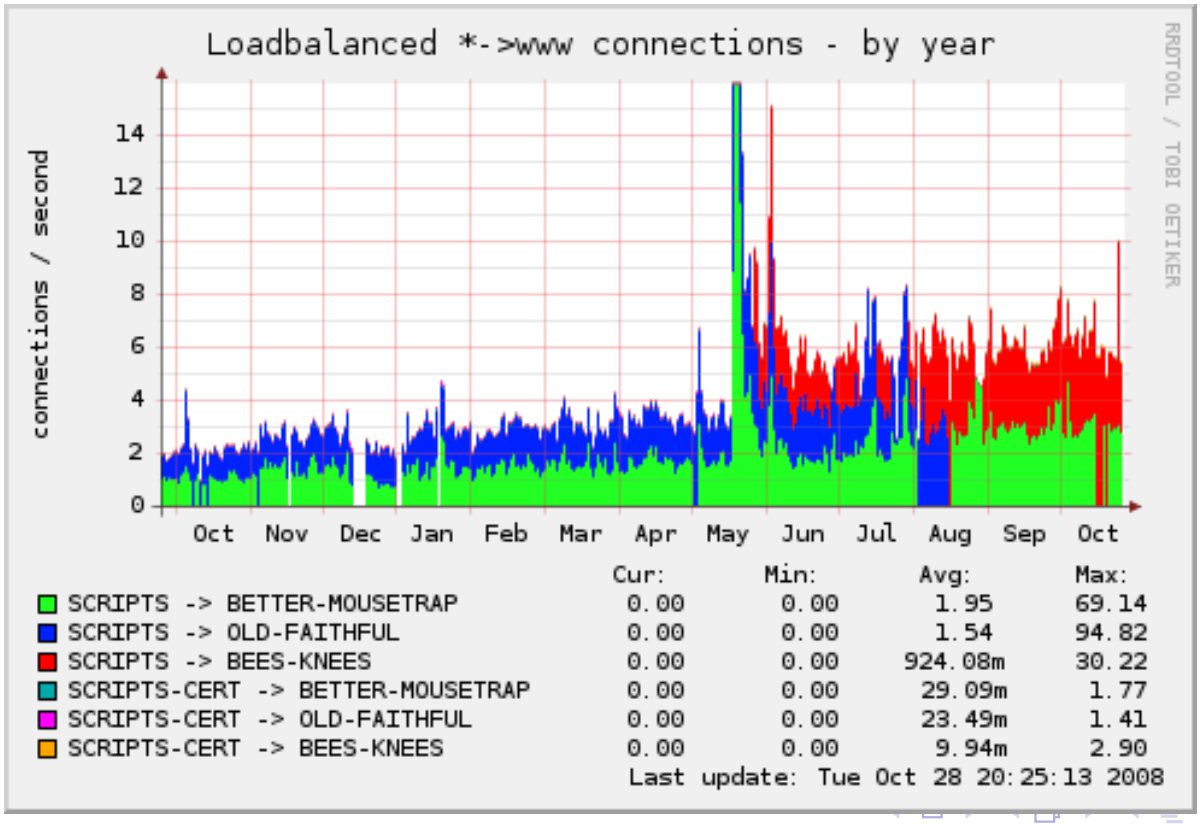
Load Balancing

- Users are assigned to scripts servers based on IP
- Works around bugs in scripts that assume a single web server



Load Balancing

- Users are assigned to scripts servers based on IP
- Works around bugs in scripts that assume a single web server



Outline

Further Info

Subversion: `svn://scripts.mit.edu/`
Scripts Hackathon
Saturday, 2 PM, W20-557