scripts.mit.edu

Quentin Smith
scripts@mit.edu

Student Information Processing Board

October 29, 2019
Outline

1 Services
- Web
- Mail
- Cron ("Shortjobs")
- SQL
- Version control
Outline

1 Services
   - Web
   - Mail
   - Cron ("Shortjobs")
   - SQL
   - Version control

2 Backend
   - AFS
   - suEXEC
   - Kerberos
   - LDAP
   - Apache modules
   - LVS
   - Ansible
Outline

1 Services
   - Web
   - Mail
   - Cron ("Shortjobs")
   - SQL
   - Version control

2 Backend
   - AFS
   - suEXEC
   - Kerberos
   - LDAP
   - Apache modules
   - LVS
   - Ansible

3 Further Info
   Quentin Smith  scripts@mit.edu  scripts.mit.edu
Outline

1 Services
   - Web
   - Mail
   - Cron ("Shortjobs")
   - SQL
   - Version control

2 Backend
   - AFS
   - suEXEC
   - Kerberos
   - LDAP
   - Apache modules
   - LVS
   - Ansible

3 Further Info
Apache

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

Apache’s default configuration isn’t safe for scripting

Scripting requires code execution—mod_php, mod_perl, mod_python, mod_wsgi

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!

Quentin Smith  scripts@mit.edu  scripts.mit.edu
Postfix

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

```
mailbox_command = /usr/bin/procmail -t -p \ -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
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

- Not well documented
- svn://username.scripts.mit.edu/ and git://username.scripts.mit.edu/
- Uses suEXEC to run a svnservice / git-daemon as the user
- /mit/username/Scripts/{svn,git}
- git:// is read-only, so future plans for svn+ssh:// and git+ssh://
Outline

1 Services
   - Web
   - Mail
   - Cron ("Shortjobs")
   - SQL
   - Version control

2 Backend
   - AFS
   - suEXEC
   - Kerberos
   - LDAP
   - Apache modules
   - LVS
   - Ansible

3 Further Info

Quentin Smith  scripts@mit.edu  scripts.mit.edu
AFS enforces server side access controls.

On Athena systems: user’s password $\rightarrow$ Kerberos tickets $\rightarrow$ 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,

```c
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 || avc->apache_access) &&
+        !(PRSFS_USR2 == afs_GetAccessBits(avc, PRSFS_USR2, areq)) &&
+        !(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))) {
    ...
```
or the file is `system:anyuser` readable anyway,

```c
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 || avc->apache_access) &&
    +     !(PRSFS_USR2 == afs_GetAccessBits(avc, PRSFS_USR2, areq)) &&
    +     !(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()`,

```c
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 || avc->apache_access) &&
    +     !(PRSFS_USR2 == afs_GetAccessBits(avc, PRSFS_USR2, areq)) &&
    +     !(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))) {
        ...
```
or apache is trying to read a file with mode 777 or named “.ht*”,

```c
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 || avc->apache_access) &&
    +   !(PRSFS_USR2 == afs_GetAccessBits(avc, PRSFS_USR2, areq)) &&
    +   !(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)) {}}
```
or the root or signup users are accessing file with the special C, D, or E bits.

```c
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 || avc->apache_access) &&
    +     !(PRSFS_USR2 == afs_GetAccessBits(avc, PRSFS_USR2, areq)) &&
    +     !(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))) { 
```
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.
/etc/httpd/conf.d/execsys.conf is configured to serve static content with the cgi-script handler.

<FilesMatch '``(?i)\.(cgi|exe|php|pl|py|scm)$``'>
    SetHandler cgi-script
    Options +ExecCGI
</FilesMatch>
<FilesMatch '``(?i)\.(avi|css|doc|docm|docx|...|zip)$``'>
    SetHandler cgi-script
    Options +ExecCGI
</FilesMatch>
...
httpd/support/suexec.c is modified to dispatch static content to /usr/local/bin/static-cat.

```c
#define STATIC_CAT_PATH "/usr/bin/static-cat"
static const char *static_extensions[] = {
    "html",
    "css",
    ...
};

int main(int argc, char *argv[]) {
    ...
    if (is_static_extension(cmd)) {
        if (setenv("PATH_TRANSLATED", cmd, 1) != 0) {
            log_err("setenv failed\n");
            exit(255);
        }
        execl(STATIC_CAT_PATH, STATIC_CAT_PATH, (const char *)NULL);
        log_err("(%d)%s: static-cat exec failed (%s)
", errno, strerror(errno), STATIC_CAT_PATH);
        exit(255);
    }
    ...
}
```

Quentin Smith
scripts@mit.edu
scripts.mit.edu
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 \texttt{.k5login} mechanism: \texttt{krb5\_kuserok()} in \texttt{krb5/src/lib/krb5/os/kuserok.c}
  - Calls a Perl script \texttt{/usr/local/sbin/admof} to do the actual check.
```c
krb5_boolean KRB5_CALLCONV
krb5_kuserok(krb5_context context, krb5_principal principal,
    const char *luser)
{
    ...
    + if ((pid = fork()) == -1)
    +   goto cleanup;
    + 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) {
    +   result = ACCEPT;
    + }
    ...
    }
```
LDAP architecture

- All user-specific information is stored in LDAP records
- `scripts-ldap-1` through `scripts-ldap-3` run LDAP daemons with multi-master replication
- Each realserver runs a read-only local caching LDAP daemon
LDAP data

- Each user has a scriptsAccount and at least one scriptsVhost
- Users can request additional virtual hosts using “pony”
- scriptsAccount is consulted by Postfix for mail routing (so accounts can be blocked)
- scriptsVhost is consulted by a cron job for SSL certificates
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
- Pacemaker 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 status

- http://scripts.mit.edu:78/ shows the current load
- Or you can finger @scripts.mit.edu for more detail

```
$ finger @scripts
[scripts.mit.edu]
IP Virtual Server version 1.2.1 (size=4096)
Prot LocalAddress:Port Scheduler Flags
   -> RemoteAddress:Port     Forward  Weight  ActiveConn  InActConn
FWM 1 wrr
   -> CATS-WHISKERS.MIT.EDU:0  Route 1  13   6
FWM 2 wlc persistent 600
   -> SHINING-ARMOR.MIT.EDU:0  Route 4096 53   855
   -> BEES-KNEES.MIT.EDU:0    Route 4096 50   2140
   -> CATS-WHISKERS.MIT.EDU:0  Route 1024 17   53
   -> BUSY-BEAVER.MIT.EDU:0   Route 4096 54   641
   -> PANCAKE-BUNNY.MIT.EDU:0 Route 4096 52   1039
FWM 3 wlc persistent 600
   -> SHINING-ARMOR.MIT.EDU:25 Route 4096 0    0
   -> BEES-KNEES.MIT.EDU:25   Route 4096 0    1
   -> CATS-WHISKERS.MIT.EDU:25 Route 1024 0    1
   -> BUSY-BEAVER.MIT.EDU:25  Route 4096 0    1
   -> PANCAKE-BUNNY.MIT.EDU:25 Route 4096 0    2
```
Ansible is a tool for declarative configuration management

We can install LVS, syslog, and real servers using Ansible

Server configuration is modular, so each feature can be separately developed

Hopefully will make future version upgrades easier
Example Role - real-fuse

- name: Install fuse.conf
  copy:
    dest: /etc/fuse.conf
    content: |
      user_allow_other
- name: Load fuse kernel module
  copy:
    dest: /etc/modules-load.d/fuse.conf
    content: |
      fuse
    notify: load modules
- name: Immediately load new modules
  meta: flush_handlers
Outline

1 Services
   • Web
   • Mail
   • Cron ("Shortjobs")
   • SQL
   • Version control

2 Backend
   • AFS
   • suEXEC
   • Kerberos
   • LDAP
   • Apache modules
   • LVS
   • Ansible

3 Further Info

Quentin Smith  scripts@mit.edu
Further Info

- Trac: https://scripts.mit.edu/trac/wiki/StarterTickets
- GitHub: https://github.com/mit-scripts/
- Zephyr: -c scripts (http://sipb.mit.edu/doc/zephyr/)