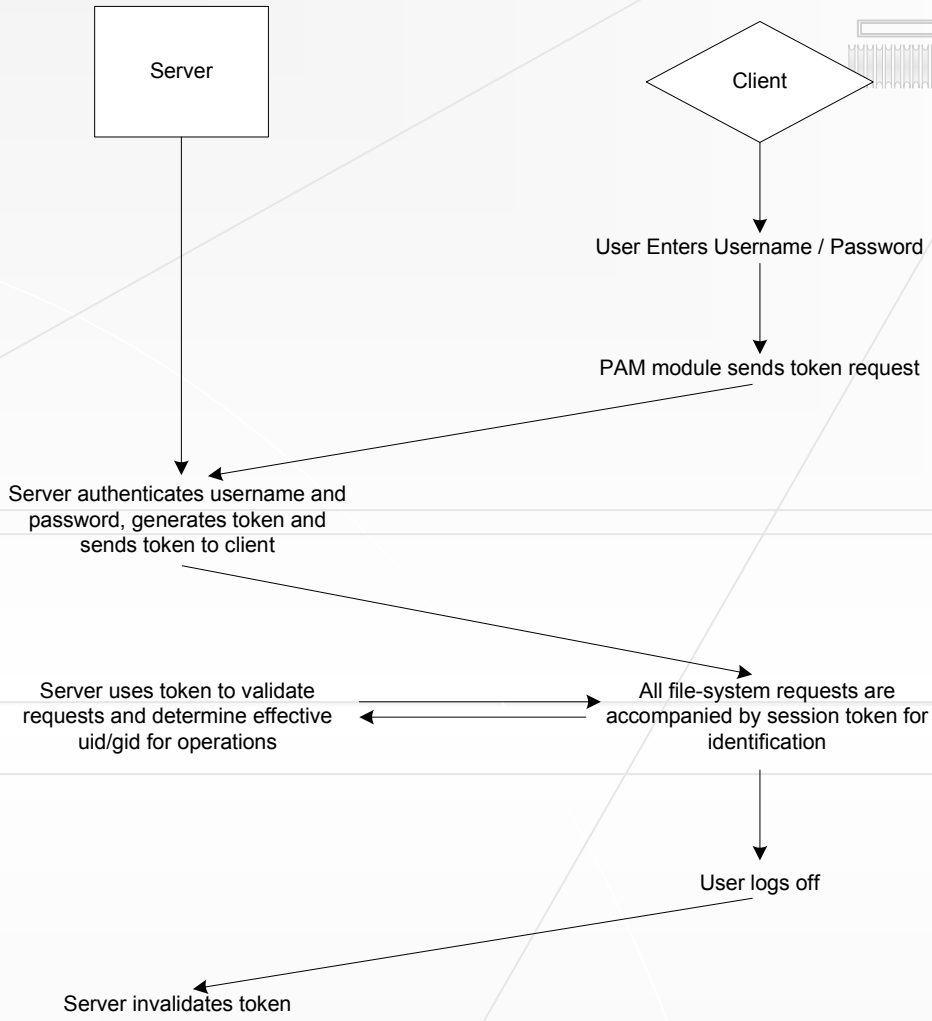
Session Authentication Overview

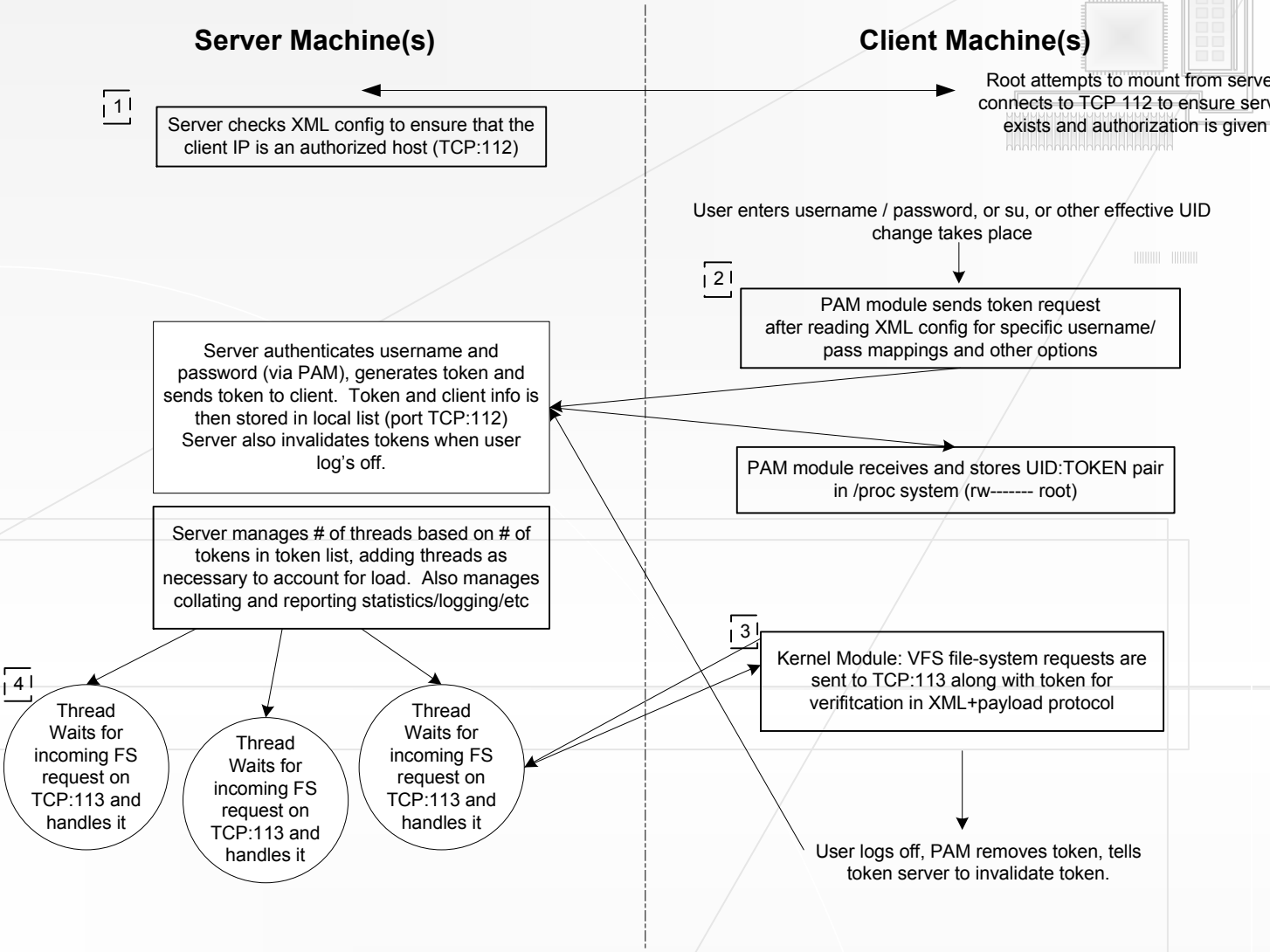
Thursday, September 26, 2002



Protection Against Common Attacks:
Client cannot “impersonate” tokens by generating them as server must generate tokens unique to each session. Tokens that accompany requests but are NOT in the list of previously generated and valid tokens result in error and indicate either an attempt at fraudulent connection, or a session who’s TTL has passed.

Client-Server network protocol should take place over an encrypted channel, preventing file-system traffic from being monitored.

The server should be able to associate a given host X with a unique token Y - to the exclusion of the same token Y from another host. This prevents a rogue machine from acquiring a token through monitoring a session and using it to “act” as the client. Such identification will have to be at the TCP/IP packet level or lower.



1) Server: This is a user mode (run as root) daemon that:

- Listens for and handles token requests/invalidations
- Manages threads that are responsible for parsing and acting on the file-system requests made by the VFS client(s). Network protocol will be XML, with payload packets separate (XML = control language). All file-system action will be taken only if:

A. Request packet token == server's token
B. Request packet IP == stored IP
C. Effective UID/GID has appropriate permission

2) PAM Module: PAM module will be responsible for:

A: Obtaining and using servers SSL cert for secure token communication
B: Contacting Token server (1) and requesting a token
C: Receiving token request response and storing valid UID:Token pairs in the /proc filesystem (/proc/??? Created by (3))
D: Sending token invalidation information when a user logs out or a session ends

3) Kernel mode VFS module: This module will be responsible for translating all filesystem requests into XML control + payload packets to be sent to the SNFS server (and receiving responses from same server). SSL certificate retrieved by PAM module should be used to encrypt each control + payload packet.

1) VFS Kernel module to SNFS server communication:

A file descriptor is created and used in the PAM->Server communication process - and ideally the kernel module could then use the same descriptor for further communication with that “session’s” daemon, however, I am unsure as to how a file descriptor can be passed to the kernel module at runtime on a per-session basis. Is it possible to somehow write the descriptor out to the /proc system in the same place/manner as the token?

SOLVED - Kernel will initiate a new connection with TCP:113 for every request

2) Client permission’s listing:

On the client side, when a listing or other examination of a snfs mount is done, how do we best display permissions? i.e. User permissions can be easily displayed/provided by using the local users uid in the response, however, gid’s may not be consistent between client and server. Perhaps this is best left as an option, allowing the user to:

- Map gid’s (much like mapping users) similar to local=remote so:
  100=200
  would display remote gid 200 as local gid 100
- Map all gid’s to a single local gid (nobody, or some other) - DEFAULT
- etc.

This may simply be a matter of providing as much flexibility as possible in configuration - followed by extremely careful and explicit documentation.

3) Cross-platform issues:

Less of an “issue” but something that should be noted from the start, the more x-platform we can make this the better for obvious reasons, what steps can we take to best ensure such portability? Obvious answers:

- Byte ordering should either be neutral (ala network order) or #ifdef’d in the source according to various platforms.
- Server and PAM module code should be as modular as possible - made possible mostly by the use of OOP

The server and pam sections of code may be the easiest to port, while the vfs modules the hardest due to its interdependency with the OS kernel. There may simply not be a way around this beyond coding a single vfs module at a time, then hand-porting the code to new architectures.