CMU-ITC-84-015

Authentication Server Internals

David King

File System Group

1. This document describes, in not much detail, the internal organization and operation of the VICE Authentication Server. This information is not necessary to write VICE server software or user workstation software to use the Authentication Server.

Definitions for all the internal data bases, arbitrary codes, and messages are contained in a header file, "authserver.h."

1.1. Definitions

central machineSome cluster machine on the network which is distinguished as far as accounting is concerned, running a copy of the Authentication Server with additional routines.

central serverThe Authentication Server running on the central machine. The central server is used as the final deposit for accounting transactions. It may be that the reliability of VICE machines will be so low that a more dynamic approach will be required to perform this function; if this proves to be the case, a different design can be developed.

internal keyAn encryption key known only by the authentication system. At present it is not easily changed, and there is no advanced key distribution system internal to the authentication system. It will become completely unnecessary if there is a separate, secure network linking the cluster machines together.

master keyThe key under which passwords are encrypted when stored in the User Authorization File.

password encryption algorithmThere may be more than one encryption algorithm or master key used for storing passwords over the life of the product.

1.2. Data Bases

1.2.1. User Authorization File

The UAF contains one record for each user, containing:

```
struct uaf {
    struct uaf_head {
        char username[8];       /* Username, or zeroes for null file slot */
        char password[16];      /* Password, encrypted with master key */
        char defacct[4];        /* Default account, or zero */
        int pchange;            /* Password change time */
        char algorithm;         /* Password encryption algorithm */
        char acctcnt;           /* The number of account blocks to follow */
    } head;
/* Maximum number of account blocks allowed */
#define UAF_MAXACCT 10
    struct uaf_acct {           /* As many of these as necessary */
        char account[4];        /* Account number */
        char flag;              /* Account flag */
#define UAF_OVERALLOC 1
        char dummy;             /* Entries must be even-bytes in length */
    } acctblk;
};
```

It would be good to use a salted password storage hashing system, as Unix does, but our use of passwords as encryption keys makes this impossible.

1.2.2. Accounting Data Base

There will be an Accounting Data Base (ADB) file on the central machine, containing the allocation status of each user. This information is not kept in the UAF because it is not needed in all of the cluster machines.

The ADB contains one record for each username/account code pair:

```
struct adb {
    char username[8];
    char account[4];
    int balance;
};
```

1.2.3. Rate Table File

There will be a Rate Table File, giving the current billing rate for each system-provided commodity. The file starts with an integer containing the number of entries to follow; the entries contain the commodity code and the rate in ergs (or perhaps tenths or hundredths of ergs; this will depend on the scale involved). It must exist on every machine, and changes very rarely. This can be read in once at system initialization; there may be a maintenance function to force the server to read a new table.

```
        struct rate_entry {
            int commodity;
            int rate;
        };
```

1.2.4. Cluster Machine File

There will be a file containing the names of each of the cluster machines, suitable for use by the RPC system. Each line of the ASCII file will contain the name of one machine; the first line will designate the "central" machine.

1.2.5. Active User Block

This will exist in the virtual memory of each Authentication Server, and will contain the data base of who is Connected and what their keys are. Each block will contain

```
        struct aub {
            int job;                    /* The job number */
            char username[8];           /* The username */
            char account[4];            /* The account number */
            char key[8];                /* Encryption key */
            int logtime;                /* Time of login */
        };
```

1.2.6. Usage Update Block

Each Authentication Server will have in the virtual memory of its main process a list of username/account number pairs, and a list of transactions and a total amount for each one, to remember how much usage should be charged against the user's account. As servers call *Bill User* these entries are created or augmented; as the server wishes, it will batch the entries to the central server to be applied against the data base, and write the transactions to the transaction file.

```
struct uub {
    struct uub *link;
    char username[8];
    char account[4];
    int total;
    struct uub_trans {
        struct uub_trans *link;
        int count;
        int subtotal;
        int code;
        int repeat;
    } *tranlist;
};
```

1.2.7. Accounting Transaction File

Each Authentication Server will have its own transaction file. The Accounting Transaction File will contain records of:

```
struct trans_head {
    int stamp;             /* Timestamp */
    char username[8];      /* Username */
    char account[4];       /* Account number */
    char cluster[20];      /* Cluster machine providing service */
    int charge;            /* Total ergs in this transaction */
    int count;             /* Number of commodity blocks to follow */
};
struct trans_comm {
    int code;              /* Commodity code */
    int repeat;            /* Number of independent transactions */
    int sum;               /* Number of "things" done */
    int subtotal;          /* Total ergs expended on "things" */
};
```

The "cluster number" in the transaction header is the index of the cluster machine's entry in the cluster name table, for eventual traffic analysis. It could be made an Internet adapter address, or the ASCII name of the cluster machine, if they turn out to be more reliable.

There is a redundancy here, since the transaction contains both the original information and the final charge. This is desirable to allow user's allocations to be corrected if retroactive changes are made to the rate tables.

1.2.8. Broadcast Queue

There will be times that the other servers must be informed of things, such as password changes and balance updates, but when they cannot be reached (one is down or the network is partitioned). The Broadcast Queue, stored in a disk file, will contain such transactions; a process will be responsible for accepting such messages, queueing them, and sending them when the recipient becomes available.

The file contains entries of the form

```
struct broadmsg_h {
    int length;              /* Length of entire entry */
    int ccnt;                /* Number of cluster addresses in list */
    int mlen;                /* Length of message */
    int cleft;               /* Number of nonzero clusters in list */
    int opcode;              /* Opcode of message */
};
```

The formats of the messages are described later, in the description of the utilities they call.

1.3. Authentication Server organization

The server will exist as two processes communicating over pipes.

o The Main Process will create the broadcast subprocess, do other initialization, and then be the focal point for centralized processing, receiving connections from clients, reading and processing their requests, and closing the connections at the end of the conversations. It will activate other work routines based on the timer. (This used to be a multi-process application, using the RPC's automatic forking mechanism, but persuasive arguments concerning the efficiency of forking, and the amount of authentication traffic to expect, have caused everything to be collapsed into one process again.)

o One child process will handle the Broadcast Queue.

1.4. Main process

The main process initializes the broadcast queue process, initializes for remote procedure calls, and then loops waiting for connections. When a connection is received, it loops reading requests and replying to them, until the conversation is over and the connection is closed. It also calls demons when required by the clock.

1.4.1. Initialization

The Internal Key is initialized, at present from a constant. The Rate Table must be read in.

The main process must create each the Broadcast Queue Process, with a pipe to feed it.

1.4.2. Usage Update Spill demon

Every hour, the master process will run through the Usage Update Blocks, write transactions to the transaction file, and send an Update ADB message to the Central via the Broadcast Queue Process containing the balance-update amounts for each of the users. (If this is the Central server, it will just call the *Update ADB* routine directly.) It will wipe out the data in the blocks, and wipe out the blocks themselves if they haven't been used in a while.

1.4.3. Flush AUB demon

This routine will be activated every few hours, to go through the AUB and erase the entry for any job which has been logged in for more than a certain amount of time (e.g., two days), after which it is assumed that the workstation never bothered to log out. If there are legitimate applications which do this, it may be necessary to develop a more elaborate scheme to keep track of still-active users.

1.4.4. Transaction File Spill demon

Whenever it seems appropriate (based on time or file size, perhaps) the master process will rename the transaction file to a new name and start a new transaction file. It can then fork a temporary process to copy the completed transaction file to a directory in the Central machine (or just rename it there if this is the Central), and delete the file. For now it can just rename it to a standard name, like "oldfile."

1.4.5. Transaction File Dump

On the Central system, every week, or whenever it is otherwise needed, the main process should fork a temporary process to do whatever is appropriate for the accumulated transaction files. If nothing else, it should erase them; perhaps it should write them to tape first.

1.4.6. Disk Billing

Every day, the main process will fork a temporary process to do disk billing. For all user directories (those starting with "/user") it is sufficient to examine each file in the directory tree (including the directory directory files themselves) and to bill its storage to the owning user's default account (or, if he has not defined one, to an account picked at random).

1.4.7. UAF consolidation

Every day the Central server should fetch the UAF from every other server and compare it with its own UAF to check for inconsistencies. If any are found they can be reported, and the other server corrected from the central's UAF.

1.4.8. Connect

The Connect service will always be called immediately after an initial connection for a secure connection. It will do the other things described above.

1.4.9. Disconnect

Upon receiving the DISCONNECT message, the Authentication Server will first see that the user is who he says he is. It will bill the user, remove the AUB entry, and send a success reply.

1.4.10. Change Authorization (CHGAUTH)

The server will validate the user's authentication information, then validate the user's request. For a CHGAPWD request, it will perform whatever checks it wishes on the new password (for instance, whether it is long enough). For a CHGAACT request, it will make sure that the specified account is on the user's account list. It will then perform the change on the local UAF; if the password is changed the password change time will be updated. When this is done, it will then distribute the change by sending an Update UAF (UPDUAF) message to every other server via the Broadcast queue process. Once this has been done, it can send the reply.

1.4.11. Assign Key (ASSIGNKEY)

The server will look up the user' information, assign the conversation key, build the message, and reply. It may make an Assign Partner RPC request of the partner's Authentication Server if necessary.

1.4.12. Get User Data

On receipt of the GETUSER message, the Authentication Server will look up the supposed username and job number in the list of Active User Blocks; if found, it will return the information desired.

1.4.13. Bill User

For a BILLUSER message, the server computes the price for the commodities, as listed in the Rate Table, and creates or updates User Update Blocks to contain the new transaction.

1.4.14. User Maintenance

It will make the change to the local UAF, and then will distribute the change via Update UAF (UPDUAF).

1.4.15. Assign Partner (ASSIGNPART)

This will be called during Assign Key when the two users are logged in to different Authentication Servers. It will do the work of assigning the key and building the reply messages, encrypting one part with the proper session key.

1.4.16. Update UAF (UPDUAF)

This will be called on servers to distribute UAF changes caused by users or by maintenance functions. It will make the change to the UAF.

1.4.17. Update ADB

The Central Server receives an UPDADB message to record permanent changes in usage allocations. In performing the changes, the server will keep track of those accounts which have now gone over their allocations. It will call *Update UAF* to set the UAF_OVERALLOC flags in their UAF entries, and pass an Update UAF (UPDUAF) message to the Broadcast Process to tell all the other Authentication Servers.

1.5. Broadcast Queue Process

This exists in every Authentication Server, and receives messages from the main process via a pipe. It will initialize by initializing the RPC system to communicate with other Authentication Servers, and by opening and parsing the Broadcast Queue. It will try to flush the queue, and then wait for messages from the main process.

Messages presented on the pipe from the main process, and messages saved in the Broadcast Queue, will look the same. When the process receives a message, it will create a header containing a list of the network addresses of all the cluster machines which have yet to receive the message (the caller has the option of targeting the message to only one machine). It will write the message to the queue file. Then it will attempt to send it, by establishing an RPC connection to the receiving cluster, making the request, getting a reply, and closing the connection; if this all happens without error the entry can be removed from the queue (or the cluster address can be removed to nullify the recipient).

Every once in a while it will time out of the receive wait and run through the queue file, attempting to send each unsent message. Once an hour seems fine for the UAF changes.

An optimization: it would be a fine thing for this to send out several messages in a burst, to overlap processing on the various clusters. But using stream sockets, which can't be overlapped as easily, seems safest for now.

1.6. Maintenance Programs

There will have to be a program to maintain the authorization file, which will add and remove users and assign passwords. There must be a program to add and remove user/account entries in the data base, to update the allocations, and to adjust or reset the balances. They should have both interactive and batch usage modes. They should *not* be major data base management systems, but simply the last interface to the operating system level, taking inputs from the large user administration system. They should use *User Maintenance and Update ADB* as necessary, which should queue any interactions it needs for clusters which are partitioned.

Initially, it will be simplest to have these programs run in the VICE machine, distributing the changes with the Broadcast mechanism. The security system will be, then, that if you can log in to a VICE machine, you can do maintenance. We can do better when we want to.