To: Distribution

From: Stan Vestal

Date: December 12, 1974

Subject: The Multics Card Input Facility

I. Introduction

The content of this MTB describes some proposed changes to the punched card input facility. The primary objective of this design change is to eliminate the security hole present in the existing card input mechanism. Issues which are addressed include the controls necessary to maintain consistency with the Access Isolation Mechanism, the approach to card data pool management, and the necessary operational procedures to support the described software changes. The general approach taken is to perform all card input from a system process and deny user attachments to the card reader.

II. Background

Currently a user of the Multics punched card input facility is required to construct a control card containing a <dim_name>, <directory_name>, <entry_name>, and an optional<access_name>. This control card is placed in front of the card deck to be read and an end-of-file card (a card containing a 5-7 multi-punch in column 1) is appended to the end of the deck. The deck and control cards are now submitted to operations via dispatch , where the read_cards command of the IO daemon is used to read the deck. Card images are placed in a segment in >ddd>cards and a link, named <entry_name>, is placed in <directory_name> pointing to the card image segment. Read, "r", access is set on the card image segment for <access_name> or to *.*.* if not specified.

This method is unsatisfactory in the context of the Multics Access Isolation Mechanism. First, a system process should not be allowed to create a link in a directory with no validation of its authority to do so. The placement of this link might allow the installation of a Trojan Horse program in an unsuspecting user's directory. Second, there is no mechanism for setting the <access_class> of the card image segment, or verifying the ownership of a card deck, other than via <directory_name> or the optional <access_name>.

III. Proposed Card Input Facility

A. Requirements

1. Each card deck (file) must be uniquely identified to ensure that multiple decks in the input hopper are not grouped in the same card image segment.

2. Card data must be assigned the correct <access_class> when placed in a card image segment.

3. The card image segment must not be placed in the search path of a process without explicit action by the user (no automatic link creation).

4. Only the owner (submitter) of the deck should be able to read the card image segment. No personid of "*" should be placed on the ACL.

5. The user should be able to easily copy the card image segment from the card pool into his working directory, using a segment name supplied by him at the time he submitted the deck read request.

B. New Deck Format

The user will supply his card deck with two control cards in front. The control card used in the present facility will be modified to include only <dim_name> and a new field <deck_name>. The <deck_name> is the name the user wishes to call the deck and is the entry name of the card image segment to be used in the move_cards operation described below. This card will be referred to as the deck_id card. A new card, the access_id card, will preceed the deck_id card. It will be used for access control purposes and will contain the <personid>, <projectid>, and <access_class> to be used for the card image segment. If the <access_class> field is omitted, system_low authorization is assumed. The user is responsible for the correctness of the <access_class> and for including the access_id and deck_id control cards with each deck. When the deck is submitted to dispatch, site personnel should validate the access_id card as a procedural check, although no security violation will occur if this check is omitted.

Operations will supply two identical card pairs for the front and back of the deck. The first card of the pair is an end-of-file (EOF) card. The function of the EOF card (5-7 multi-punch) will not change. This card will still be detected and interpreted by crz. The second card is a unique id (12 characters) in mcc format (See Appendix I for a sample deck). The EOF card is used as control information to mark the end of user supplied data. The unique id card is used to verify that the deck is correctly identified and that the user has not put an EOF card in his deck.

C. Card Pool Management

All card image segments created by the card input facility will reside in system pool storage. The location in the pool hierarchy is determined by <personid> and <access_class>. That is, the path name will be:

```
card_pool_root>access_class_dir>personid>deck_name
```

where access_class_dir is a unique name derived from the authorization of the card reading process. The user will have "sm" access to the personid directory at the authorization of access_class so that his process may remove segments which have been successfully copied into his working area.

Terminal quota will initially reside at the access_class directory level during card reading. After the successful completion of a read, quota equal to pages used will be moved to the personid level. In this way a user's use of the system card pool space is limited.

Garbage collection in the card pool will be based upon the dtem of the personid directory branch as well as the dtm of individual segments in the directory. It will be performed by a single subroutine which invokes system privilege to effect its deletion and quota moves, based on the card_pool_root and the number of days elapsed since last collection. This garbage collection subroutine may optionally be invoked from the card reading process or from command level of a privileged process, but always at the explicit request of operations personnel. Once the expired card image segments have been removed from the pool area, the quota will be restored back to the highest level possible.

Name duplications in the personid directory indicate a possible attempt to circumvent access validation and therefore will cause the operator to be notified and reading to cease. The segment bearing the offending name will be deleted to eliminate the possibility of a trick deck being slipped in on some unsuspecting user. Therefore, the user must ensure that deck_name is unique within the personid directory when it is entered on the deck_id card.

D. New Operation of read_cards Command

1. Read first card in hopper using mcc dim.
2. If no cards present, return to process command level.
3. If not an EOF card, go to step 1.
4. Read unique id from next card and save for later checking.
5. Read <personid>, <projectid>, and <access_class> from next card and save for later use. A "*" in any field except the projectid field is not legal.
6. If <access_class> not equal to current authorization of card reading process, abort and notify operator.
7. Read <deck_name> and <dim_name> from next card.
8. Create the <personid> subdirectory in card_pool_root>access_class_dir if it does not exist. Create the segment <deck_name> in personid with no access to any other process at this time. On name duplication, delete the old segment, abort and notify the operator.
9. Attach input stream through <dim_name> to crz card input stream.
10. Read cards into <deck_name> until EOF is found.
11. Detach input stream and reattach through mcc dim to read in an intelligible format.
12. Read next card to verify the unique id from above.
13. If unique id just read is not equal to unique id from step 4, delete deck_name, abort and notify the operator.

14. Terminate deck_name segment, set the bit count, remove access of the card reading process and set access for personid.projectid.* to r on the segment and personid.*.* to sm on the directory.

15. Go to step 1

E. User Access to Card Image Segment

1. User logs in as <personid>.<projectid> at the authorization equal to <access_class>. If the user is unable to login with the correct <personid>, <projectid>, and <access_class> the card image segment will not be accessible to him.

2. User executes a new command, "move_cards <deck_name> [target_seg]", where <deck_name> may use the star convention, and the optional argument target_seg is the pathname of the segment in which to place the card image segment matching deck_name (equals convention may be used). If target_dir is omitted, the working directory is assumed.

3. The move_cards module will locate and copy segments in the pool directory belonging to <personid> for which <access_class> equals the authorization of the user. After the copy is successfully performed, the card image segment will be deleted from the pool directory.

F. General Comments

1. Any segments remaining in the pool directories more than n days without being deleted by the user, will be deleted by operations, as will any empty pool directories during garbage collection.

2. The card reading process will ensure that a pool directory corresponding to its level exists by using system process privilege while in cards_overseer_. After that, the only privilege used is attachment of the card reader, use of the message routing dim and pool management.

3. There appears to be no good reason for the card reading process to be IO.SysDaemon now that the process no longer modifies the user's directory. Therefore, a process group id of Cards.SysDaemon is proposed to make the name more descriptive of its function and to simplify the code in the regular IO Daemon.

Appendix I. Deck Setup Description

```
EOF
unique character id card
access_id card (personid, projectid, access_class)
access_id card continued as necessary, terminated by ";"
deck_id card (deck_name, dim_name)
(user data inserted here)
               .
               .
               .
EOF
unique character id card
               .
               .
(continue sequence for multiple decks)
               .
               .
```

SAMPLE

```
EOF
XGHSKLTPCWQT
\R\JONES \COM_PROJ SENSITIVE,C1,C2;
LINPROG.PL1 MCC
LINPROG:   PROC;
/*  WRITTEN BY R. JONES 10/3/74 */
           .
           .
( remainder of program text )
           .
           .
     END;
EOF
XGHSKLTPCWQT
EOF
TRPLWNQMTBXL
           .
           .
(next user supplied deck)
           .
           .
EOF
TRPLWNQMTBXL
```

Name: move_cards

The move_cards command moves specified card image segments from system pool storage into a user's directory. The segments to be moved must have been created using the Multics Card Input Facility. The user process executing this command must have the proper access to the card image segment in order to perform the move. After a successful move, the card image segment in pool storage will be deleted.

Usage

```
     move_cards deck_name [new_deck_name]
```

1) deck_name	is the name which was entered on the deck_id card when the card deck was submitted for reading.
2) new_deck_name	is the pathname of the segment in which the matching card image segment is to be placed. If omitted, the working directory and deck_name is assumed (optional).

Notes

The deck_name may follow the star convention and all matching card image segments in pool storage to which the user has access will be moved. Similarly, new_deck_name may use the equal convention. It is the user's responsibility to resolve name duplication difficulties. The original segment will not be deleted until the move has been performed successfully.

See the description of the card input facility in the MPM for the format of the control cards needed when submitting a card deck to be read by system operations.

Example

```
          move_cards my_deck
```

would move the user's card image segment named my_deck from the card pool storage into the user's current working directory.