Security Controls of Multics

Michal Paulovic
STU FIIT 2020

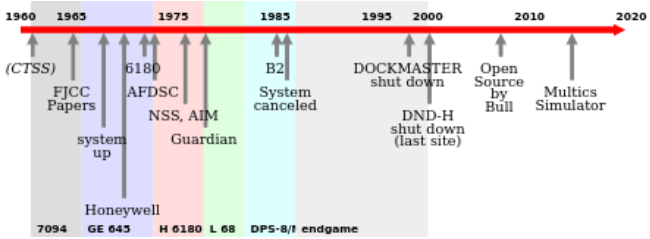

Figure 1: Timeline of Multics [10]

1 HISTORY OF MULTICS

In the early 1960s, large scale-computing was dominated by *mainframes*. Controlled by *batch processing* supervisor program. Users submitted jobs on *5 hole paper tape*, and received printed output. Timesharing, in which a large machine switched between multiple users, giving each the impression of virtual computer, had been proposed by MIT Professor *John McCarthy* in 1959.

Multics (Multiplexed Information and Computing Service) is a timesharing operating system which begun in 1965 and been used until 2000. The system was started as a co-project of MIT's *Project MAC*, *Bell Labs* (withdrew from development in 1969) and *General Electric Company*.

Chief of project was **Professor Fernando J.Corbató** (MIT). MIT operated a Multics timesharing service with over a thousand users through the 1970s. Major customer of this service was Honeywell, which also helped a lot during the *security enhancement era* of Multics operating system.

Multics was invented to provide access to many remote terminal users simultaneously, in a manner similar to MIT's **CTSS** system. Multics combined ideas from other operating systems with new innovations. Security was a fundamental design requirement. Multics ran on specialized expensive CPU hardware that provided a segmented, paged, ring-structured virtual memory.

Multics was introduced in a series of papers at the 1965 Fall Joint Computer Conference:

- Introduction and Overview of Multics System - F.J. Corbató, V.A. Vyssotsky
- System Design of a Computer for Time-Sharing Applications - E.L. Glaser, J.F. Couleur, G.A. Oliver
- Structure of the Multics Supervisor V.A. Vyssotsky, F.J. Corbató, R.M. Graham
- Communications and Input/Output Switching in a Multiplex Computing System - J.F. Ossanna, L. Mikus, S.D. Dunten
- Some Thoughts About the Social Implications of Accessible Computing - E.E. David Jr., R.M. Fano

1.1 Major Goals for Multics

Goals for Multics are described by Corbato and Vyssotsky in *Introduction and Overview of Multics System* [2].

- Convenient remote terminal use.
- Continuous operation analogous to power and telephone services.
- A wide range of system configurations, changeable without system or user program reorganization.
- A high reliability internal file system.
- Support for selective information sharing.
- Hierarchical structures of information for system administration and decentralization of user activities.
- Support for a wide range of applications.
- Support for multiple programming environments and human interfaces.
- The ability to evolve the system with changes in technology and in user aspirations.

The team's ambition was that Multics would change the way people used and programmed computers.

> "One of the great unanticipated discoveries of Project MAC was that when more than one person can use a computer at the same time, those people will use the computer to communicate with each other." **Simson Garfinkel, Architects of the Information Society**

2 DESIGN PRINCIPLES

Design principles and overview of Security controls (section 3) are described in *Protection and the control of information sharing in multics* by J.H. Saltzer [7] **and briefly by E.L. Glaser** [3].

(1) Every designer should know and understand the protection objectives of the system. There are many points at which *"I don't care"* decision can be a breaking point of internal security measurements. By keeping all designers aware of the protection objectives, the decisions are more likely to be made correctly.
(2) Keep the design as *simple* and as *small* as possible. Design and implementation errors which result in unwanted access paths will not be immediately noticed during routine use. Therefore, techniques such as complete, *line-by-line* auditing of the protection mechanisms are necessary, that's the reason why small and simple design is essential.
(3) Protection mechanism should be based on *permission* rather than exclusion. The principle means that the default situation is *lack of access*, and the protection scheme provides selective permission for specific purpose.
(4) Every access to every object must be checked for authority.
(5) The design is not secret. The mechanism do not depend on the *ignorance of potential attackers*, but rather on possession of specific, more easily protected, protection keys or passwords.
(6) The principle of least privilege. Every program and every privileged user of the system should operate using the least amount of privilege necessary to complete the job.

(7) Make sure that the design encourages correct behaviour in the user, operators and administrators of the system. It is essential that human interface be designed for naturalness, ease of use and simplicity, so that user will routinely and automatically *apply the protection mechanisms*.
(8) Provide the option of complete decentralization of the administration of protection specifications. If the system design forces all administrative decisions to be set by a single administrator, that administrator quickly becomes a *bottleneck*. which could result in users begin adopting habits which bypass the administrator.
(9) The system provides a complete set of tools, so that a user without exercising special privileges, may construct a *protected subsystem*, which is a collection of programs and data with the property that the data may be accessed only by programs in the subsystem, and the programs may be entered only at designated entry points.

3 SECURITY CONTROLS

3.1 Access Control List

The central fixture of Multics is an organized information storage system which provides reliability and protection from unauthorized information release. All use of information in the storage system is implemented by mapping the information into the virtual memory of some process.

Storage is logically organized in separately named data storage segments, each which contains up to 262144 36-bit words. A segment is the cataloguing unit of the storage system.

Segments in Multics are stored in a hierarchy of director(Because of endless and penetrable checking of input arguments).*Directory* is a special type of of segment that is not directly accessible to the users. Directory stores names and other information about underlying segments and directories. Each segment and directory has an *Access control list* (ACL) in its parent directory entry *who* may *read*, *write* or *execute* (also known as 'r-w-x') the segment or obtain *status* of, *modify* entries line or *append* entries on directory (also known as 's-m-a'). The ACL consists of three partitions (can be seen in *Figure 2*):

(1) Every individual user of system in a separate access control group by himself with his *personal name*.
(2) Sets of users in groups who cooperate in some activity, called *projects*. One person may be a member of several projects.
(3) The third partition allows an individual user to create his own, named protection compartments (directories).

For example: User Mike in Project Compiler has all rights to read-write-execute

```
Mike.Compiler.x      (rwx)
```

Also *asterix (*)* to grant access to *all*. User Mike in all associated project, thus directories has read-write privilege.

```
Mike.*.*             (rw)
```

In order to access file/directory, the ring 0 (rings mechanism is in greater details described in section cislo.cislo) software compares the ID of user with access control list entries.

User identification is established when user sings on to the Multics and it is stored in the Process Data Segment (PDS), which is

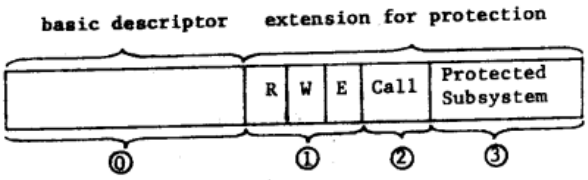

Figure 2: Multics descriptor [7]

accessible only in ring 0 or in Master mode - *so that user cannot modify the process data segment* and grant access anywhere on the system.

3.2 Authentication of users

Whole process of access control lists, access modes, protected subsystems and hierarchical control depends on an accurate principal identifier being associated with every process. Accuracy of the identification depends on authentication of the user's claimed identity.

Every user is *registered* with *unique name* (convention: last name, plus one or two initials) associated with a *password* of up to eight ASCII characters. Also as another measurement, the user may (after authenticating) change his password at any time.

When user types a password it is *enciphered* and compared with the stored enciphered version for validity. All passwords on system are stored in non-invertible cipher (one-way encryption) form, which also acts as additional security in case of system dump.

Plain text passwords are stored *nowhere* on the system. As a result, passwords are not routinely know by any system administrator or project administrators.

Additional feature of Multics system password mechanism is that, when user is requested to give his password, the printer on his terminal is *turned off*.(Because of endless and penetrable checking of input arguments).

Each login and logout is carefully audited to check for attempts to guess a valid password for a certain user. If a user provides incorrect password, the event of an *incorrect login attempt* is noted in *threat-monitoring log*, and user is permitted to try again, up to ten-times at which point the telephone or network connection is broken by the system.

In addition each user is informed of the *date*, *time* and *terminal identification* of last login to detect past compromises of the users *access rights*, also user is told the number of times his password has been given incorrectly since its last correct use in his greeting message.

3.3 Primary Memory Protection

The virtual address space of a Multics process is implemented with an *array of descriptors*, called a *descriptor segment*. Every reference to the virtual memory specifies both, segment number (index) and a word number within the segment.

The reason why the protection information is associated with the addressing descriptor rather than data itself is, in *multiprocessor system* such as Multics, two same processes may be executing at a same time, so single protection specification associated with the data is not sufficient.

(1) Physical address and size of the segment based on this descriptor.
(2) Bits separately controlling permission (read-write-execute).
(3) Control of permission to enter a protected subsystem which has entry points in the segment.
(4) Controls on which protected subsystems may use this descriptor.

Another feature of the descriptors used in Multics is implemented inside the *second* and *third* part of descriptor, which can be seen on *Figure* 2, together they allow hardware enforcement of protected subsystems, which are intended to be used only by calls to *designated entry points*, know as *gates*. As a result of this, it is possible to construct proprietary programs which cannot be read. They return only statistics rather than raw data.

Protected subsystems are formed by using the *third field* of the descriptor extension. Multics force a *hierarchical* constraint on all subsystems within a single process. Each subsystem is assigned a number between **0** and **7**. This scheme is also called *the ring mechanism* or *the rings of protection*. The protection rings are defined in the way: *The higher numbered rings having less privilege than lower numbered rings, and ring 0 containing 'hardcore' supervisor.* Therefore, each subsystem is permitted to use all of those descriptors containing protected subsystem numbers greater or equal to its own.

The descriptors are adjusted to provide *only* the amount of access required by the *supervisor* in direct harmony of *Design principal number 6*. These mechanism do not prohibit the supervisor from making full use of hardware, rather than protection against accidental/exploitation overuse of supervisor privileges.

3.4 Master Mode

The convention around protecting *master mode software*, was specified as the master mode procedures were not to be used outside ring 0. Convention had been defined as of "*each master mode procedure must have a master mode pseudo-operation code assembled into location 0*". The master mode pseudo-operation generates code to test an index register for a value corresponding to an entry point in the segment. If the index register is invalid, the master mode pseudo-operation saves the registers for debugging and shuts the system down.

3.5 Software Maintenance Procedure

The system is initialized from a magnetic tape which contains copies of every program residing in the *most protected area*. The maintenance of the *Multics software* had been carried out **online** on a *dial-up Multics facility*. A programmer prepared (and debugs) his software for installation, then submits his software to a *library installer* who copied and recompiled the source code in a *protected directory*. The software prior installing into the system source and object libraries is checked by library installer. Ring 0 software is stored on a *system tape* that is reloaded into the system each time it is brought up. (New system tapes are generated from online copies of the ring 0 software). The system libraries had been protected against modification by access control list mechanism. In addition the library installers had been periodically checked of last modification of all segments in the library to detect unauthorized modifications.

4 RUNNING MULTICS NOWADAYS

Creating a Multics system or machine nowadays is a bit sketchy process. Since Multics is relying on designated hardware such as HIS-645 or 6180 processors (developed by Honeywell) it is not an easy task to do. You can not simply create *Virtual Machine* from from disk image (ISO). But there are still some options how to do it, you can use *simulator/emulator* for such a task.

MIT in cooperation with BULL runs Multics Internet Server [5] containing edition of Multics software and documentation, which is useful if you want to try PL/I and rest locally.

4.1 DPS8-Simulator

I came across project which still maintain love and desire for running Multics system. **RingZero - Multics Reborn** [6] project, created by *Harry Reed* and *Charles Anthony* which is available for public download. Development takes place on GitLab. They provide *DPS8-M simulator*. Still with no much capabilities, but as stated milestone from 11. August 2014:

> *Multics reborn! The dps-8/m SIMH-based simulator booted Multics MR 12.5, came to operator command level, entered admin mode, created a small PL/I program, compiled and executed it, and shut down.* [6]

4.2 How to run Multics on GNU\Linux

Detailed process can be found Swenson's Multics Wiki [8]

(1) Download *Pre-built executable* (so we do not need to build simulator from source).
(2) Download "QuickStar RLV", which provides and already Multics system disk image that is ready to run.
(3) Unzip both archives and put in the same directory in system. Go to directory in terminal and:

```
$> ./dps8 MR12.6f_boot.ini

DPS8/M emulator (git 2a56f38d)
Production build
#### M_SHARED BUILD ####
System state restored
Please register your system at https://ringzero.wikidot.com/wiki:register
or create the file 'serial.txt' containing the line 'sn:0'.
FNP telnet server port set to 6180

DPS8/M simulator V4.0-0 Beta        git commit id: c420925a
TAPE: unit is read only
MR12.6f_boot.ini-30> set opcon config=attn_hack=1
Non-existent device
[FNP emulation: listening to 127.0.0.1 6180]
CONSOLE: ALERT
bootload_0: Booting system MR12.6f generated 01/09/17 1119.1 pst Mon.
1500.2  announce_chwm: 428. pages used of 512. in wired environment.
1500.2  announce_chwm: 706. words used of 1024. in int_unpaged_page_tables.
```

```
find_rpv_subsystem: Enter RPV data: M-> [auto-input] rpv a11 ipc 3381 0a

1500.2  load_mst: 946. out of 1048. pages used in disk mst area.
bce (early) 1500.2: M-> [auto-input] bce

Multics Y2K.  System was last shudown/ESD at:
Saturday, April 1, 2017 20:37:12 pst
Current system time is: Sunday, April 19, 2020 7:00:16 pst.
Is this correct? M-> [auto-input] yes

The current time is more than the supplied boot_delta hours beyond the
unmounted time recorded in the RPV label.  Is this correct? M-> [auto-input] yes

The current time I'm using is more than 12 hours
after the last shutdown time recorded in the RPV label.
Are you sure this is correct? M-> [auto-input] yes

bce (boot) 0700.2: M-> [auto-input] boot star

Multics MR12.6f - 04/19/20  0700.4 pst Sun
0700.4  Loading FNP d, >user_dir_dir>SysAdmin>a>mcs.7.6c>site_mcs 7.6c
Received BOOTLOAD command...
0700.4  FNP d loaded successfully


Volumes to be Scavenged:

rpv    (root)
root2  (root)
root3  (root)

Ready
0700  as   as_init_: Multics MR12.6f; Answering Service 17.0
0700  as   LOGIN              IO.SysDaemon dmn cord (create)
0700  as   LOGIN              Backup.SysDaemon dmn bk (create)
0700  as   LOGIN              IO.SysDaemon dmn prta (create)
0700  as   LOGIN              Utility.SysDaemon dmn ut (create)
0700  as   LOGIN              Volume_Dumper.Daemon dmn vinc (create)
0700  as   LOGIN              Scavenger.SysDaemon dmn scav1 (create)
0700  as   as_mcs_mpx_: Load signalled for FNP d.
0700  bk
0700  cord Enter command:  coordinator, driver, or logout:
-->  cord
0700  bk   r 07:00 0.194 30
0700  bk
-->  bk
0700  prta Enter command:  coordinator, driver, or logout:
-->  prta
0700  ut   copy_dump: Attempt to re-copy an invalid dump.
0700  ut   delete_old_pdds: Some directory or segment in the pathname is not listed in the VTOC. Unable to salvage >pdd.!BBBKPBBzF
0700  ut   jWCfq. Will attempt to delete it.
0700  ut   delete_old_pdds: Some directory or segment in the pathname is not listed in the VTOC. Unable to salvage >sl1.!BBBKPBBzF
0700  ut   jWCfq. Will attempt to delete it.
0700  vinc
0700  vinc r 07:00 0.197 25
0700  vinc
-->  vinc
0700  scav1 Created >system_control_1>scav1.message
0700.5  scavenger: Begin scavenge of dska_00a by Scavenger.SysDaemon.z
0700  scav1 Scavenging volume rpv of logical volume root
0700  as   sc_admin_command_: Utility.SysDaemon.z: delete_old_pdds
0700  ut   send_admin_command: Execution started ...
0700.7  vtoce_stock_man: Attempt to deposit free vtocx 4037 on dska_00
0700.7  vtoce_stock_man: Attempt to deposit free vtocx 4036 on dska_00
0700  ut   completed.
0700.7  scavenger: Scavenge of dska_00a by Scavenger.SysDaemon.z completed.
0700.7  scavenger: Begin scavenge of dska_00b by Scavenger.SysDaemon.z
0700  scav1 Scavenging volume root2 of logical volume root
0700  ut   monitor_quota: The requested action was not performed.
0700  ut   The quota of >dumps is 0, a record limit needs to be specified.
0700.7  RCP: Attached tapa_00 for Utility.SysDaemon.z
0700  ut
0700  ut   Records   Left  %    VTOCEs   Left  %   PB/PD LV Name
0700  ut
0700  ut   166172   99180  60   42218   33972  80  pb root
0700  ut
0700  ut   r 07:00 0.505 477
0700  ut
-->  ut
0700.7  RCP: Detached tapa_00 from Utility.SysDaemon.z
0700.7  RCP: Attached rdra for Utility.SysDaemon.z
0700.7  RCP: Detached rdra from Utility.SysDaemon.z
0700.7  RCP: Attached puna for Utility.SysDaemon.z
0700.7  RCP: Detached puna from Utility.SysDaemon.z
0700.7  RCP: Attached prta for Utility.SysDaemon.z
0700.7  RCP: Detached prta from Utility.SysDaemon.z
0700.9  scavenger: Scavenge of dska_00b by Scavenger.SysDaemon.z completed.
0700  scav1 Scavenging volume root3 of logical volume root
0700.9  scavenger: Begin scavenge of dska_00c by Scavenger.SysDaemon.z
0701.1  scavenger: Scavenge of dska_00c by Scavenger.SysDaemon.z completed.
```

Your Multics system is up and running. The terminal shell in which you started the simulator and Multics is now the operator console.

The simulator should be listing on TCP port 6180 and if you connect to that port using telnet, you should be able to login.

```
$> telnet localhost 6180

Trying ::1...
telnet: connect to address ::1: Connection refused
Trying 127.0.0.1...
Connected to localhost.
```

```
Escape character is '^]'.
HSLA Port (d.h000,d.h001,d.h002,d.h003,d.h004,d.h005,d.
    h006,d.h007,d.h008,d.h009,d.h010,d.h011,d.h012,d.
    h013,d.h014,d.h015,d.h016,d.h017,d.h018,d.h019,d.
    h020,d.h021,d.h022,d.h023,d.h024,d.h025,d.h026,d.
    h027,d.h028,d.h029,d.h030,d.h031)?
Attached to line d.h000

Multics MR12.6f: Installation and location (Channel d.
    h000)
Load = 5.0 out of 90.0 units: users = 5, 04/19/20  0706.0
     pst Sun
```

Now you can login as SysAdmin

```
$> login Repair -cpw    # flag to change password

Password:
New Password:
New Password Again:

Your password was given incorrectly at 04/15/20  0706.4
    pst Sun from ASCII term
\cinal "none".
# Audit Trial in use, I've put password once incorrectly

Password changed.
You are protected from preemption.
Repair.SysAdmin logged in 04/19/20  0706.7 pst Sun from
    ASCII terminal "none".

New messages in message_of_the_day:

Welcome to the Multics System.

print_motd:  Created >user_dir_dir>SysAdmin>Repair>Repair
    .value.
r 07:06 0.208 31

M->         # you can start using Multics
```

With this running Multics you can do:

- Logging into an Existing Account
- Logging in as an Anonymous User
- Creating Directories
- Listing the Contents of Directories
- Entry Names
- Editing files (Emas, qedx, edm, Teco)
- Printing files
- Setting Time Zone

Quick overview of these capabilities will be demonstrated during my presentation.

5 MULTICS SECURITY EVALUATION BY USAF

Since on running DPS8-M simulator, you can not really breach security controls. I will describe in short well known evaluation of Multics system and how they were able to exploit vulnerabilities in system.

The security evaluation is described in *Multics Security Evaluation: Vulnerability analysis* by P.A. Karger and R.R. Schell [4].

Evaluation of the security of the Multics system had been carried out on the *HIS 645* Multics Systems installed at the *Massachusetts Institute of Technology* and at the *Rome Air Development Center* in span of March 1972 and June 1973.

The main purpose for this evaluation had been the lack of effective *multi-level security controls* in US military. The term multi-level (from US Military point of view) means, in the most general case, those controls needed to process several levels of *classified material*, from unclassified through compartmented top secret in a *multi-processing multi-user computer system with simultaneous access* to the system by users with different levels of clearances.

The lack of such effective systems in those days has led the *US Military* to operate computers in closed environment in which system had been dedicated to the highest level of classified material and all users had been required to be cleared to *that* level. Such a systems resulted in extreme insufficient manpower utilization and high costs of deployed systems, mainly becuase of quantity of dedicated systems.

Requirements of the *Air Force Data Service Center* (AFDSC) were to be providen with *responsive interactive time-shared computer services* to users within the Pentagon at all *classification levels*. From *unclassified* to *top secret* clearance level. AFDSC in particular did not wish to incur the expense of *multiple computer systems* nor the expense of *encryption devices* for remote terminals which would otherwise be processing unclassified materials.

Several more analysis has been done in years after, but this analysis was the key for Multics to gain B2 security evaluation mark.

5.1 State of "current systems" in early 70's

The internal controls of current computers repeatedly had been shown *Insecure* through many *penetration exercises* on system such as *WWMCCS GCOS* and *IBM OS/360/370*. This insecurity of internal controls had been fundamental weakness of *these* operating systems and cannot be corrected by *"patches"*, *"fix-ups"* and *"add-ons"* to these systems. Rather a *fundamental re-implementation* using an integrated hardware and software design which would consider **security as fundamental requirement**.

It is not sufficient to use a team of experts to *test* the security controls of a system. Such a *ZARF "tiger team"* can only show the existence of vulnerabilities but *cannot* prove their *non-existence*.

5.2 Reference Monitor

The concept of *reference monitor* had been introduced by the *ESD Computer Security Technology Panel*. This reference monitor is that hardware and software *combination* which must monitor **all** references by **any program** to **any data anywhere** in the system to ensure that the security rules are followed, which are defined by:

- The monitor must be tamper proof.
- The monitor must be invoked for every reference to data anywhere in the system
- The monitor must be small enogh to be *proven correct*

Systems such as GCOS and OS/360 meet just first requirement. The second requirement was generally not met by systems contemporary in *60s* and *70s*, since they usually include *bypasses* to permit special software top operate. The most important, operating systems in those days had been so *large*, so *complex* and so *monolithic* that one could not begin to attempt a formal proof or certification of their current implementation.

5.3 Approach Plan

An attempt was to be made to operate with same type of ground rules under which a real agent would operate. That is, with each penetration, and attempt would be made to extract or modify sensitive system data without detection by the *system maintenance* or *administrative personnel*. Several exploitation's had been discovered, such as changing access fields in SDW's, changing protected identities in the PDS, inserting trap doors into system libraries and accessing the system password file.

In this paper we are not going to to describe all the vulnerabilities shown by this evaluation, rather just those interesting and with the most significant impact on contemporary development of operating systems and its security's control systems.

5.4 Subverter routine

To attempt a gross measure of the rate of security sensitive component failure, a procedure called the *subverter* was written to sample the security sensitive hardware on frequent basis, testing for component failures which could compromise the security controls. The subverter was run in the background of an interactive process, once each minute. The subverter was run over 1100 hours in a year period on the *MIT 645 System*, during these operating hours no security sensitive component failures were detected. Which points at and gave great result about the concert in a system processing multi-level classified material. If the hardware is prone to error, potential security vulnerabilities become a significant problem.

5.5 Instruction Access Check Bypass

While experimenting with the hardware subverter, a sequence of code was observed which would cause the hardware of the 645 to *bypass access checking*. Specifically the *execute* instruction in certain cases would permit the executed instruction to access a segment for *reading* or *writing* without the corresponding permissions in the SDW. This vulnerability occurred when the execute instruction was in certain restricted locations of a segment with at least *read-execute* permission. The exact layout of instructions and indirect words was crucial, otherwise the *access checks* were done properly. This bug represents a violation of one of the most fundamental rules of the Multics design concept - the checking of **every** reference to a segment by the hardware.

5.6 Insufficint Argument Validation

On the 645 CPU rings mechanism have not been implemented in the way of hardware. (Only after several years on HIS 6180 CPU.) Therefore, the protection rings had been implemented by providing *eight descriptor segments* for each user, one descriptor segment per ring. Because the 645 Multics system must simulate *protection rings* in software, there is no direct hardware validation of arguments passed in a subroutine call from a *less privileged ring* to *more privileged ring*. Some form of validation is required, because a malicious user could call a ring 0 routine that stores information through a user supplied pointer. If the malicious user supplied a pointer to data to which ring 0 had *write* permission but to which the user ring did not, ring 0 could be *"tricked"* into causing a security violation.

To provide validation, the 645 *software ring crossing mechanism* requires all gate segments to declare to the *"gatekeeper"* the following information:

(1) number of arguments expected
(2) data type of each arguments
(3) access requirements for each argument - read only or read-/write

This information is stored by convention in specified locations within the *gate segment*. The gatekeeper invokes an argument validation routine that inspects the argument list being passed to the gate to ensure that the declared requirements are met. If any test fails, the argument validator aborts the call and signals the *" gate_error"* in the calling ring.

In 1973, a vulnerability was identified in the argument validator that would permit the *"fooling"* of ring 0 programs. The argument validator's algorithm to validate read or read/write permission was as this:

(1) Copy the argument list into ring 0 to prevent modification of the argument list by a process running on another CPU in the system while the first process is in ring 0 and has completed argument validation
(2) Force indirection through each argument pointer to obtain the segment number of the target argument
(3) Look up the segment in the calling ring's descriptor segment to check for read or write permission

The vulnerability is as follows:

- An argument pointer supplied by the user is constructed to contain an IDC modfier that causes the first reference through the indirect chain to address a valid argument.
- The first reference is the one made by the argument validator.
- The reference through the IDC modifier increments the address field of the tally word causing it to point to a different ITS pointer which points to an argument which is writable in ring 0 **only**.
- The second reference through this modifier indirect chain is made by the ring 0 program which proceeds to **write data where it should not**.

This vulnerability resulted from violation of a basic rule of the Multics design, that *all arguments to a more privileged ring must be validated*. The problem was no in the fundamental design, but the lack of *ring hardware* on HIS-645 CPU system.

5.7 Master Mode Transfer

The HIS-645 CPU has a *master mode* in which privileged instructions may be executed and in which *access checking is inhibited* although address translation through segment and page tables is retained. The original design of Multics protection rings called for master mode code to be restricted to ring 0 by convention. This convention caused the *fault handling mechanism* to be excessively expensive due to necessity of switching from user ring into ring 0 and out again using the full software ring crossing mechanism.

Therefore it was proposed and implemented that the *signaller* module was permitted to run in the user ring to speed up fault processing.

The analysis shows that master mode procedures in the user ring represents a fundamental violation of the Multics security concept. Violating this concept moves the security controls from the basic hardware and software mechanism to the cleverness of the systems programmer who, *being human, makes mistakes and commits oversights* The master mode procedures become classical "supervisor calls" with no rules for sufficient security checks.

The master mode *pseudo-operation* code was designed only to protect master mode procedures from random calls within ring 0. It was not designed to withstand the attack of a malicious user. By moving the signaller into the user ring, the designers allowed a user leaving a core dump and therefore to crash the system.

5.8 Dump and Patch Utilities

This section shows the exploitation by a remote user online.

To provide support to the system maintenance personnel, the Multics system includes commands to *dump* or *patch* any word in the **entire virtual memory**. These utilities are used to make online repairs while the system continues to run. These commands are dangerous, because they *bypass* all security controls to access otherwise protected information, and if misused, it can cause crash of the entire system.

To protect the system, these commands are implemented by special privileged *gates* into ring 0. The access control lists on these gates restrict their use to system maintenance personnel by name and authenticated by the login procedure.

Dump and patch utilities would be of a great use to a system penetrator, because they can be used to "do his job". These utilities are prevented from misuse by *ACL* and *audit trial*, but using *insufficient argument validation*, *master mode transfer* and *unlocked stack base* vulnerabilities, these procedural controls may be bypassed and the penetrator can implement his own dump and patch utilities. Can be seen on *Figure 3*.

5.9 Violating The User Identification

In the need for a protected identification of every user, his ID must be compared with access control list entries to determine whether a user may access some segment as stated in Security Controls section. This identification is established when the user logs into Multics and is authenticated by the password.

The user identification in Multics is stored in a per-process segment called the *process data segment* (PDS). PDS resides in ring 0 and contains many constants used in ring 0 and the ring 0 procedure stack. The PDS must be accessible to any ring 0 procedure within a user's process and must be accessible to ring 4 master master mode procedures such as signaller.

Therefore as stated above section about Dump and Patch utilities vulnerability, these can dump and patch portions of the PDS, thus *forging the non-forgeable user identification.* This capability provides the penetrator with an ultimate exploitation weapon. The penetrator can undetechably act as any user of the system, including the *system administrator* or *security officer*, immediately assuming that user's access privileges.

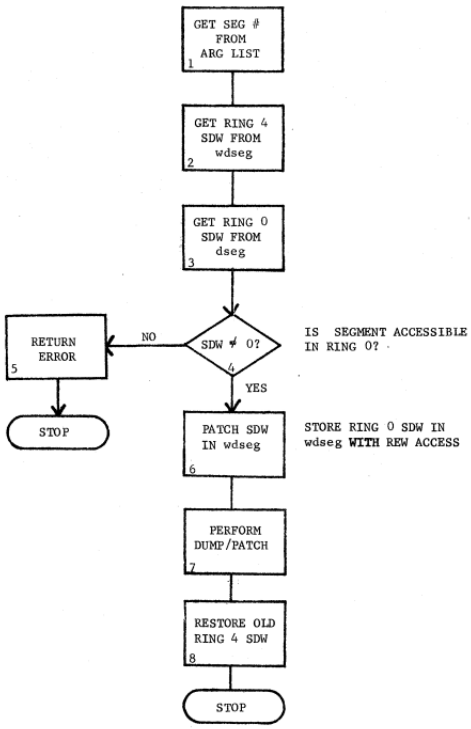

Figure 3: Dump/patch utility using insufficient argument validation

5.10 Value of the Password File

The password file is often kept enciphered. A great deal of effort may be required to invert such a cipher, if the cipher is even invertible at all. The password file is generally the most protected file in a computer system. If the penetrator has succeeded in breaking down internal controls to access the password file, he can also **access every other file in the system**. The login path to a system is generally the most carefully audited to attempt to catch *unauthorized* password use. The penetrator risks *detection* if he uses an unauthorized password.

So in this point we can assume that accessing the system password file is of minimal value to a penetrator for reasons stated above.

5.11 Modifying Audit Trials

Audit trails are frequently put into computer systems for the purpose of detecting breaches of security. For example, a record of last login time printed when a user logged in could detect the unauthorized use of a user's password and identification. Sometimes it is not convenient for the penetrator to bypass an audit, if the audit trial is kept online, it may be easier to allow the audit to take place and then go back and modify the evidence of wrong doing. Every segment in Multics carries with it audit information on the *Date Time last Used* (DTU) and *Date Time last Modified* (DTM). These dates are maintained by an *audit mechanism* at a low level in the system.

A routine called "set_dates" is provided among the various subroutine calls into ring 0, which is used when a segment is retrieved from a *backup tape* to set the segments DTU and DTM to the values at the time the segment was backed up. This routine is supposed to be callable only by system maintenance personnel (ring 0). However, since the penetrator can change his user identification, this restriction proves to be no barrier.

To access a segment penetrator simply:

(1) Change user ID to access segment.
(2) Remember old DTU and DTM.
(3) Use or modify the segment.
(4) Change user ID to system maintenance.
(5) –do whatever you want–
(6) Reset DTU and DTM to old values.
(7) Change user ID back to old value.

5.12 Trap Door Insertion

In short, trap door is a a feature or defect of a computer system which allows surreptitious unauthorized access to data and system itself. When trap door is inserted, it must be well hidden to avoid detection by system maintenance personnel. Trap doors can be inserted in many places:

- At the facility which the system is produced.
- During the distribution phase. If updates are sent via insecure communications.
- During the installation and operation of the system at user's site.

Trap doors can be easily hidden in changes to the binary code of compiled routine. Such a change can be detected only by comparing bit by bit the object code and the compiler listing. Which can be easily secured by regularly recompile of all modules of the system to eliminate such a trap doors. However, if the compiler trap door is inserted to permit object code trap doors to survive even a complete recompilation of entire system. In Multics most of the ring 0 supervisor is written in PL/I. Penetrator could insert trap door in the PL/I and easily delist the desired code to be recompiled. Since the PL/I compiler is itself written in PL/I, the trap door can *maintain itself, even if the compiler is recompiled.*

The actual insertion of the trap door was done by following steps:

(1) Change user ID to project SysLib.
(2) Make patch in the object archive copy of "check$device name" in >ldd>hard>object.
(3) Reset DTM on object archive.
(4) Mark patch in bound archive copy of "check$device name" in >ldd>hard>boundcomponents.
(5) Reset DTM on bound archive.
(6) Reset user ID back to old value.

5.13 Conclusion by Schell and Karger

The primary conclusion one can reach from this analysis is that Multics is *not currently a secure system* (1973). But on the other hand, it is **significantly more secure** than any other system.

6 SIGNIFICANCE AND INFLUENCE OF MULTICS

As we could read in previous sections, it is clear that Multics was the first system, which considered *security* a **necessity**, not an additional feature. Multics was designed to be secure from the beginning. This fact is in modern operating systems taken for granted. In the 1985 the system was awarded the B2 security rating by the NCSC, the first (and for years only) system to get a B2 rating.

Multics was written in the *PL/I* language (created by IBM in 1964), only part of the operating system was implemented in *assembly* language. Writing operating system in a *high-level* (not in today's standards) language was a radical idea at the time. In addition to PL/I, Multics supports BCPL, BASIC, FORTRAN, LISP, COBOL, ALGOL68 and additionally C and Pascal.

Multics provided *first commercial relational database* product. The *MRDS*, Multics Relational Data Store.[1]

All systems in early 70's was designed to be able to run 24/7, but only Multics had capability to add or remove CPUs, memory, I/O controllers and disk drives from system configuration while the system is running.

The most notable influence on other operating system, which we can still see today is in GNU/Linux based systems. Part of it is because inventors of **UNIX** *Ken Thompson* and *Dennis Ritchie* (also inventor of C language), worked on Multics until *Bell Labs* dropped out of the Multics development in 1969.

Ken Thompson was "inspired" by *access control list mechanism*, which was implemented in UNIX and was also one of the primary security measurements in the system.

The idea of having the *command processing shell* to be an ordinary slave program came from the Multics design, and a predecessor program on *CTSS* by *Louis Pouzin* called **RUNCOM**.[10]

The value of Multics in UNIX early development ideas could be also seen at the *50th Anniversary* of the Unix creation. One of the articles, by *Richard Jensen* is called *Unix at 50: How the OS that powered smartphones started from failure*, which direct reference to Multics. Such articles often say:

> "Multics failed. The Bell Labs computer guys then invented their own system, Unix, and Unix succeeded.

On the other hand, Multics did not fail, it accomplished almost all of its stated goals. Unix succeeded too, solving a different problem.

Next time, when you "ssh-into" your remote server and see greeting such as this:

```
1    Last login: Fri Apr 17 13:52:12 2020 from
      88.212.40.17}
```

You can think of **Multiplexed Information and Computing Service**.

> *"To what extent should one trust a statement that a program is free of Trojan horses? Perhaps it is more important to trust the people who wrote the software."*
> **Ken Thompson** [9]

7 GLOSSARY

- CTSS - The Compatible Time-Sharing System, IBM 7090 and young prof. Corbato about Timesharing on WGBH-TV (1963)
- Descriptor - Contents of an SDW.
- Gate - Segment that allows transfer of control between rings in a controlled fashion. Each gate segment has a vector of entries at its start.
- IBM OS\360\370 - Family of mainframe computer system developed by IBM.
- IDC - Incerement address, Decrement tally, and Continue
- PDS - Process Data Segment
- PL\I - Programming Language #1, invented by George Radin of IBM in 1964
- RCU - Restore Control Unit (ring 0 instruction)
- Signaller - Module responsible for processing faults (master mode procedure)
- SDW - Segment Descriptor Word. An element of a process's descriptor segment; the hardware-accessible data element that defines a segment and the process's access rights to it.
- Subverter - Program written by the Project ZARF team.
- WWMCCS GCOS - The Worldwide Military Command and Control System General Comprehensive Operating Supervisor
- ZARF - Code name for Air Force/MITRE tiger team project that cracked Multics security in 1973.

REFERENCES

[1] 2020 (last accessed April 19, 2020). *Mindpride Computer Services.* http://www.mindpride.net/root/Extras/History/OS/history_of_multics.htm

[2] F. J. Corbató and V. A. Vyssotsky. 1965. Introduction and Overview of the Multics System. In *Proceedings of the November 30–December 1, 1965, Fall Joint Computer Conference, Part I (AFIPS '65 (Fall, part I)).* Association for Computing Machinery, New York, NY, USA, 185–196. https://doi.org/10.1145/1463891.1463912

[3] Edward L. Glaser. 1967. A Brief Description of Privacy Measures in the Multics Operating System. In *Proceedings of the April 18-20, 1967, Spring Joint Computer Conference (AFIPS '67 (Spring)).* Association for Computing Machinery, New York, NY, USA, 303–304. https://doi.org/10.1145/1465482.1465529

[4] P.A. Karger and R.R. Schell. 2002. Multics security evaluation: Vulnerability analysis. 127 – 146. https://doi.org/10.1109/CSAC.2002.1176286

[5] MIT and BULL. 2020 (last accessed April 19, 2020). *Multics Internet Server.* https://web.mit.edu/multics-history/source/Multics_Internet_Server/Multics_sources.html

[6] Harry Reed and Charles Anthony. 2020 (last accessed April 19, 2020). *RingZero - Multics Reborn.* http://ringzero.wikidot.com/

[7] Jerome H. Saltzer. 1974. Protection and the Control of Information Sharing in Multics. *Commun. ACM* 17, 7 (July 1974), 388–402. https://doi.org/10.1145/361011.361067

[8] Swenson. 2020 (last accessed April 19, 2020). *RingZero Multics Wiki.* http://multics-wiki.swenson.org/index.php/Getting_Started

[9] Ken Thompson. 1984. Reflections on Trusting Trust. *Commun. ACM* 27, 8 (Aug. 1984), 761–763. https://doi.org/10.1145/358198.358210

[10] Tom Van Vleck. 2020 (last accessed April 19, 2020). *Multicians - The Multics Web.* https://multicians.org/multics.html