A Penetration Analysis of a Burroughs Large System

A.L. Wilkinson, D.H. Anderson, D.P. Chang, Lee Hock Hin,
A.J. Mayo, I.T. Viney, R. Williams, W. Wright.

University of Canterbury, Christchurch, New Zealand.

Abstract

A penetration analysis of a Burroughs Large System is described and a fundamental flaw allowing total penetration is discussed. Specific aspects examined include File Security, Resource Limits, Accounting Methods, and Disruptions.

1 Introduction

A recent paper (1) reported a penetration analysis performed by a graduate-level computer science class at the University of Michigan on the Michigan Terminal System (MTS). A penetration analysis attempts to assess, as comprehensively as possible, how secure a system is against deliberate attempts to use it in an illegitimate manner (5).

Penetration analysis is a useful educational exercise as well as an aid to computer management. Students have to work as a team, handle confidential information responsibly, co-operate with the computer system management, and gain a substantial amount of technical insight.

This paper describes a penetration analysis of the Burroughs B6700 computer system at the University of Canterbury. The software system was Burroughs' 3.0 P.R.#1 release, and incorporated minor local modifications. Burroughs' hardware and software architecture is sufficiently different (2) from conventional architectures that some unique problems were encountered. The analysis is also of interest following the rejection of the B6700 architecture on security grounds by the U.S. Department of Defense during its comparative study of machine architectures (3).

This analysis was carried out by an honours-level computer science class at the University of Canterbury, assisted by a Programmer/Analyst from the University computer centre. As with the MTS study, the work was done with the co-operation of the

University computer centre, and full access to system documentation and source listings was provided.

2 Organization

The project was undertaken as part of a Computer Systems course which is one of the fourth-year Computer Science Honours Part 3 courses at the University of Canterbury. Following a series of lectures covering general principles of operating systems and software, twelve lectures were given on specific details of the B6700 system. The project was given as a class assignment and the work was assessed and contributed towards the final grades given for the course.

The class was divided into four groups, each of two students. The instructions given were as follows:

> Each group will produce a report describing their analysis of the susceptibility of the Canterbury University B6700 Computer System to "penetration" by non-privileged users. The reports should include a brief survey of the literature, and be as comprehensive as possible in considering various kinds of penetration which could be attempted. Actual investigation of possible security flaws is limited to software aspects, rather than physical entry or compromising of administrative or operational staff or procedures. Only normal job-submission and terminal-access procedures may be used to obtain system access.
>
> The analysis is expected to be treated as a professional project with due ethical regard for your responsibilities to your client - in this case, the University Computer Centre. In particular, on no account must any revealed security flaws be "leaked" outside the study group until they have been reported to the Computer Centre and corrected. Also, only tests designed to avoid any impact on other users or the system operation should be used to prove penetration to secured files or practicable denial-of-access mechanisms.

At an initial meeting, the class decided to assign one aspect to each group. The aspects chosen for investigation were:

(a) File Security
(b) Resource Limits
(c) Accounting Methods
(d) Disruptions

Each group then worked separately but communicated informally and came together for a combined discussion in the middle of the project. Separate reports covering each aspect were produced and assessed.

3 File Security

3.1 Background

The B6700 file security relies on distinctions between ordinary users and privileged users, privileged programs ("message control systems"), and operating-system tasks ("independent runners"). Ordinary users are subject to strict security controls, but the last three categories have unrestricted access to file data although they are still prevented from compromising the integrity of the disk file subsytem structures.

Unlike other architectures, the B6700 is also dependent on the security of its file-typing and on the reliability of its compilers. The hardware restricts memory access within bounds set by the compilers. The file system allows only valid compilers to create or alter machine-code files. (If a program which is not a valid compiler writes into a code-file its type is changed to that of an ordinary data file and it can no longer be executed.) It also controls the creation of new compilers. All programming is done in high-level languages - there is no assembler language (4).

3.2 Possible Flaws

The ease of the successful penetration described in 3.3 may give a false impression of the system. Other possible mechanisms are not easy to envisage. We suspect that a B6700 system which disallowed the loading of code files from demountable tape or disk packs would be very difficult to penetrate. The software is well-proven and reliable. No flaws were found in the control exercised by system software over user attempts to manipulate the disk file system directly. The standard protection mechanisms provide a powerful, flexible structure for controlling access to disk and tape files. There is a high probability of detecting attempts at unauthorized access.

As with many systems, password stealing (especially if a privileged user can be identified: see 5.3) represents a serious threat (but see 5.1).

The standard software gives the central site operators privileged status for file access. Where this is considered to be inappropriate it could readily be withdrawn.

3.3 Successful Penetration

An ordinary unprivileged user with sufficient knowledge of the system needs only the ability to be able to modify machine-code in order to penetrate the system completely. This ability is provided because the system allows code files to be loaded from storage on magnetic tape. Tape is a standard medium for transfering data between computer systems but has no security structures to protect it. (Figure 1)

```
                               Security
  -----------------           Maintained       -----------------
 | Disk File System |    -------------->   | Tape File System |
 |                  |    <--------------   |                  |
  -----------------      <**************   -----------------
          ^ |               Violation Path ^      ^ |
          | |                              *      | |
 Security | |                              *      | |    Unsecured
Maintained| |            ---------------  ***     | |     Access
          |  -------->  | User Programs |  --------  |
           -----------  |               |  <---------
                         ---------------
```

Fig 1. File System Security Violation Path

Although the Burroughs software which transfers files between tape and disk storage does use complex protective structures, there is nothing to prevent a knowledgeable user from imitating these structures and creating arbitrary code files which the Burroughs system will load and execute.

This security flaw was demonstrated by the following sequence of operations:

1. A program was written and compiled which endeavoured to change the type of an arbitrary file to any desired code-file type. (Because it was not recognized as a compiler, this program could not yet be run successfully.)

2. The code for this program was copied to tape. The system utility which does this creates a header record specifying the file-type.

3. Using direct tape access methods, a second tape was created on which the code file was validated as a compiler by altering the file-type in the header record.

4. The code file was copied from the second tape to disk and the system accepted it as a valid compiler. The security breach illustrated in Figure 2 had now been accomplished.

```
   Disk File System                           Tape File System
----------------------                      ----------------------
|                    |      Security      |                      |
|  ----------------  |     Maintained     |  ----------------    |
| | Illegitimate    |------------------->|  Illegitimate    |*****
| | Compiler        |  |                 |  | Compiler        |  |    *
|  ----------------  |                    |  ----------------    |    *
|                    |                    |                      |    *
|  ----------------  |                    |  ----------------    |    *
| | Illegitimate,   |  |                 |  | Illegitimate,   |  |    *
| | Validated       |<-------------------|  Validated        |  |    *
| | Compiler        |  |   Security      |  | Compiler        |  |    *
|  ----------------  |     Breached       |  ----------------    |    *
|                    |                    |          ^           |    *
----------------------                    ----------*-----------     *
                                                    *                  *
                                                    *  Unsecured       *
                                                    *  Access          *
                                                    *                  V
                                                  ----------------
                                                 | User Program   |
                                                  ----------------
```

Fig 2. Validation of an Illegitimate Compiler

5. A subroutine was written to make privileged any program that called it. This was achieved by compiling using a standard compiler, modifying the resulting code file, and then using the false compiler to change the file-type back to valid code. (See 3.1 and Figure 3.) valid code.

6. This subroutine was used by a program which invoked a protected system intrinsic to make a user permanently privileged.

At this point, complete penetration of the file and run-time system had been achieved. Any desired access to files, accounting data, or the run-time system was possible.

```
          Disk File System
     ---------------------------
    |                           |
    |   ---------------------   |          ---------------------
    |  | Secured Code File   | <----------| Legitimate Compiler |
    |   ---------------------   |          ---------------------
    |                        ********
    |                           |   *
    |                           |   *****> ---------------------
    |   ---------------------   |         | User Program        |
    |  | Modified, illegit., |<*********** ---------------------
    |  | but non-executable  |  |
    |  | Code File           |*******
    |   ---------------------   |   *      ---------------------
    |                           |   *****>| Illegitimate,       |
    |                           |         | Validated Compiler  |
    |                           |   ****** ---------------------
    |   ---------------------   |   *
    |  | Illegitimate,       |  |   *
    |  | Executable Code File|<******
    |   ---------------------   |
     ---------------------------
```

Fig 3. Exploitation of an Illegitimate Validated Compiler

3.4 Remedies

The only complete remedy that we can envisage is to deny unprivileged users the ability to load code files from demountable tape or disk pack. For many configurations, including ours, this is not practicable. It would certainly be practicable to prevent unprivileged users from loading compilers from tape, but that makes penetration only slightly more difficult since modified code must then be validated on tape rather than on disk.

No tape modification detector, such as a checksum, can be proof against a user who discovers the algorithm. Moreover, any format modification must be compatible with the vast number of existing tape files (and those imported from other Burroughs systems) which do not have it. If alternative formats are provided, a well-informed user can simulate the simpler one. Nevertheless, this option seems to be the only possible way to increase the difficulty of penetrating configurations which allow code files to be loaded from tapes.

4 Resource Limits

4.1 Background

Code execution on the B6700 is based on the concept of a task. Each task has a working stack and an associated set of task attribute variables. These structures contain information on resource usage and limits for the task. A task may initiate further (sub)tasks. When it does, appropriate limits are transferred to the new tasks. The standard compilers restrict access to these structures from user programs.

4.2 Possible Flaws

If a task can execute uncontrolled machine-code (produced in the manner described in 3.3, for example), then the system can no longer enforce control over it. There is no hierarchy of privilege or machine states to protect the hardware operators. Protection of the run-time system is completely dependent on control over the code which is executed.

No faults were found in the controlled access permitted to the limit variables. A parent task can impose more severe restraints on its sub-tasks than apply to itself, but not less severe. No task can relax its own restraints.

Burroughs software must be modified to apply resource limits to work initiated through the terminal system (CANDE). Earlier (unauthorized!) investigators at the University of Canterbury had discovered a flaw in the control of independently-running jobs initiated from CANDE tasks. In the case of batch work, the independent job is assigned limits taken from the initiating job (which runs as the highest level task). However, in the case of CANDE work there is no individual job task for each user sharing the system, and appropriate limits are not applied. At our installation, CANDE tasks have now been prevented from initiating this kind of job.

There is also a flaw in the way limits are applied to subtasks. The resources remaining for the initiating task or job are transferred to the new task when it is initiated, but the resources used by each task are debited to the job when the task terminates. By initiating new tasks and holding nearly completed tasks in a wait state to prevent them terminating, it is possible for the total resources used by all the tasks to exceed the intended limits.

This tactic has the disadvantage of being fairly obvious to the computer operators (since a number of tasks will appear in the wait state) and also potentially controllable by limits on the number of tasks which may be initiated. This flaw could be removed by more extensive checking of resource usage at task initiating and task wait points.

5 Accounting Methods

5.0 Background

The University of Canterbury uses a locally-developed real-time accounting package which utilizes the standard Burroughs intrinsics and userdata file structures to maintain its database. The database is updated automatically at job completion, and by account entry and reporting packages. It may also be modified by the standard Burroughs package which accesses the userdata file. All these programs are protected by the normal file security system against unauthorized use.

5.1 Password Security

Passwords are held in the userdata file in a hashed form and never retained in a readable form. The hashing algorithm is sufficiently non-trivial to make the reverse transformation difficult. However, all attempts to obtain even the hashed form of another user's password were foiled by the security system. Only privileged users can read records other than their own from the userdata file. Ordinary users may not alter even their own records by direct access to the file.

Various methods of password testing were studied to try to find one which was suitable for a repeated trial-and-error attack. None were vulnerable, and several produced diagnostic printouts showing an invalid password had been tested. This should warn the owner of the password and possibly help to identify the offender. The terminal system logs unsuccessful attempts immediately and deactivates the terminal if several occur in succession. The run-time system terminates any task which tests an invalid password.

The most vulnerable location of passwords is on the punched cards used for submission of batch jobs. Whereas the increasing use of remote terminals intensifies other security problems, password stealing will become less easy.

5.2 Task Resource Accounting

The accounting system relies on resource usage as logged by the operating system when each task is completed. Although many of these usage variables may be interrogated by user programs (see 4.2) the system successfully resisted attempts to change them directly.

Using a seek command on a card input file caused the number of cards read to be miscounted. Also, use of the direct card-to-disk input mechanism avoided any charge for card-reading. No other flaws were discovered in the resource accounting mechanism.

5.3 Total Penetration

Total penetration of the accounting system was proven in the course of the operations described earlier (see 3.3). Any privileged user has complete access to the file system, including the accounting system. From a security point of view this is unsatisfactory, although the issue of code protection is of over-riding importance. Once code is adequately protected there is a need to bring the privileged users and tasks within the same kind of structured security system as currently controls non-privileged users.

6 Disruption

6.1 Overview

The forms of penetration already discussed could be employed to cause disruption. Files and accounts could be compromised or destroyed and the run-time system sabotaged. This section considers other disruptive mechanisms.

Physical disruption could be caused by external factors (interference or failure of the power supply or communications network) or by internal factors. Generally these depend on the local environment and the robustness of the hardware.

Software disruption may occur if there are any situations in which the system fails to cope. This may manifest itself as a system crash, pre-emption of resources or a deadlock situation.

Psychological disruption may be caused to either users or operators of the system by activities other than disrupting the system itself.

Where any of these methods can be proved to have been employed deliberately, any perpetrator will have to avoid being caught.

6.2 Possible Flaws

The risks of physical disruption are clear and local precautions are generally taken to meet a balance between risk and cost and convenience factors.

The likelihood of opportunity for software disruption diminishes with maturity of the software as "bugs" are encountered and fixed by the systems software staff. New software releases inevitably bring a few new bugs which are duly fixed in their turn.

Resource control was found to be fairly tight (see 4). The only clearly vulnerable resource was disk space. This could be preempted by the user, but the machine operators have utilities which both retrieve it and identify the culprit. At the time this work was done this problem was highlighted by a chronic shortage of disk space on the configuration.

Deadlocks tend to fall into the same category as software disruption and are rarely possible in mature software.

Disruptive attacks on the system operators through excessive task generation, or creating deceptive messages on their display consoles were investigated but were found to be traceable and controllable.

Except by generating code illicitly (see 3.3) all attempts to evade the memory protection system failed.

6.3 Summary

Generally, the system was well-protected against disruption. The major risk was that a password could be stolen and used to disguise the real author of anti-social activities. If this happens, the password may be changed to stop the disruption even if the offender is not apprehended.

7 Conclusion

As with the MTS study, non-total forms of penetration met with very limited success. The security of the controls on code validation and privileged users are interdependent. A breach of either gives total penetration. The system is theoretically wide open to complete penetration through the insecurity of code files stored on magnetic tape. However, exploitation requires access to system-programming details not normally provided. To a lesser extent this also applies to the exploitation of a stolen privileged user's password and probably accounts for the survival of these flaws for many years - they have obviously not caused practical problems. Nevertheless it is time they were removed.

System security requires more than correctly coded software and secure hardware. It cannot be evaluated without considering the people who operate the system. Conversely, a satisfactory level of security is often achieved with less-than-perfect hardware and software barriers to penetration. Threats of detection or intervention by "guards", and concealment of information, often suffice to inhibit security violations which may be theoretically feasible (6). It is foolish to place undue emphasis on theoretical perfection.

References

1. Hebbard, B. et al. 1980. A Penetration Analysis of the Michigan Terminal System. Operating Systems Review, Vol 14, No. 1 (January), p 7-20. Association for Computing Machinery.

2. Organick, E.I. 1973. Computer System Organization - The B5700/B6700 Series. Academic Press (New York and London).

3. Fuller, H., and Burr, W.E. 1977. Measurement and Evaluation of Alternative Computer Architectures. Computer, Vol 10, No. 10 (October), p 24-35. IEEE Computer Society.

4. Bingham, H.W. 1974. Access Controls in Burroughs Large Systems. Privacy and Security in Computer Systems, National Bureau of Standards Special Publication No. 404, p 42-45.

5. Linde, R.R. 1975. Operating System Penetration. Proceedings 1975, National Computer Conference, AFIPS, p 361-368.

6. Gaines, R.S., and Shapiro, N.Z. 1978. Some Security Principles and their Application to Computer Security. Operating Systems Review, Vol 12, No. 3, p 19-28. Association for Computing Machinery.

7. Saltzer, J.H., and Schroeder, M.D. 1975. The Protection of Information in Computer Systems. Proceedings of the IEEE, Vol 63, No. 9, p 1278-1308.