Thirty Years Later: Lessons from the Multics Security Evaluation

Paul A. Karger
IBM Corp., T. J. Watson Research Center
Yorktown Heights, NY 10598
karger@watson.ibm.com

Roger R. Schell
Aesec Corporation
Pacific Grove, CA 93950
schellr@ieee.org

ABSTRACT

Almost thirty years ago a vulnerability assessment of Multics identified significant vulnerabilities, despite the fact that Multics was more secure than other contemporary (and current) computer systems. Considerably more important than any of the individual design and implementation flaws was the demonstration of subversion of the protection mechanism using malicious software (e.g., trap doors and Trojan horses). A series of enhancements were suggested that enabled Multics to serve in a relatively benign environment. These included addition of "Mandatory Access Controls" and these enhancements were greatly enabled by the fact the Multics was designed from the start for security. However, the bottom-line conclusion was that "restructuring is essential" around a verifiable "security kernel" before using Multics (or any other system) in an open environment (as in today's Internet) with the existence of well-motivated professional attackers employing subversion. The lessons learned from the vulnerability assessment are highly applicable today as governments and industry strive (unsuccessfully) to "secure" today's weaker operating systems through add-ons, "hardening", and intrusion detection schemes.

1 INTRODUCTION

In 1974, the US Air Force published [24] the results of a vulnerability analysis of what was then the most secure operating system available in the industry. That analysis of Multics examined not only the nature and significance of the vulnerabilities identified but also the technology and implementation prospects for effective solutions. The results are still interesting today, because many of the concepts of UNIX and other modern operating systems came directly from Multics.

This paper revisits those results and relates them to the widespread security problems that the computer industry is suffering today. The unpleasant conclusion is that although few, if any, fundamentally new vulnerabilities are evident today, today's products generally do not even include many of the Multics security techniques, let alone the enhancement identified as "essential."

2 Multics Security Compared to Now

Multics offered considerably stronger security than most systems commercially available today. What factors contributed to this?

2.1 Security as a Primary Original Goal

Multics had a primary goal of security from the very beginning of its design [16, 18]. Multics was originally conceived in 1965 as a computer utility – a large server system on which many different users might process data. This requirement was based on experience from developing and running the CTSS time sharing system at MIT. The users might well have conflicting interests and therefore a need to be protected from each other – a need quite like that faced today by a typical Application Service Provider (ASP) or Storage Service Provider (SSP). With the growth of individual workstations and personal computers, developers concluded (incorrectly) that security was not as important on minicomputers or single-user computers. As the Internet grew, it became clear that even single-user computers needed security, but those intervening ill-founded decisions continue to haunt us today. Today, the computer utility concept has returned [13], but today's operating systems are not even up to the level of security that Multics offered in the early 1970s, let alone the level needed for a modern computer utility. There has been work on security for Computational Grids [12, 21], but this work has focused almost entirely on authentication and secure communication and not on the security of the host systems that provide the utility.

2.2 Security as Standard Product Feature

One of the results of the US Air Force's work on Multics security was a set of security enhancements [44] that ultimately became the primary worked example when defining the standard for Class B2 security in the Trusted Computer System Evaluation Criteria (TCSEC) [2], better known as the Orange Book. In the years that followed, other similar enhancements [9, 11] were done for other operating systems, generally targeted for the much weaker Class B1 level, but none of these security enhanced operating systems integrated as well with applications as the Multics enhancements had. This was because unlike ALL of the other security enhanced systems, the Multics

Proceedings of the 18th Annual Computer Security Applications Conference (ACSAC'02)
1063-9527/02 $17.00 © 2002 IEEE

IEEE COMPUTER SOCIETY

security enhancements were made part of the standard product, shipped to ALL users, rather than only to those users who specifically requested them. A significant reason why security enhancements to other operating systems were not integrated into the mainstream products was that, unlike Multics, neither the operating systems nor the applications generally were well structured for security, making security additions awkward.

While this difference might not seem significant initially, and indeed might even seem detrimental to the manufacturer's revenue stream (since security enhancements could command high prices), the reality was very different. Because the Multics security enhancements, including mandatory access controls, were shipped to ALL customers, this meant that the designers of applications had to make sure that their applications worked properly with those controls. By contrast, many application developers for other systems with optional security enhancements don't even know that the security enhancement options exist, let alone develop applications that work with them.

2.3 No Buffer Overflows

One of the most common types of security penetrations today is the buffer overflow [6]. However, when you look at the published history of Multics security problems [20, 28-30], you find essentially no buffer overflows. Multics generally did not suffer from buffer overflows, both because of the choice of implementation language and because of the use of several hardware features. These hardware and software features did not make buffer overflows impossible, but they did make such errors much less likely.

2.3.1 Programming in PL/I for Better Security

Multics was one of the first operating systems to be implemented in a higher level language.[1] While the Multics developers considered the use of several languages, including BCPL (an ancestor of C) and AED (Algol Extended for Design), they ultimately settled on PL/I [15].

Although PL/I had some influence on the development of C, the differences in the handling of varying length data structures between the two languages can be seen as a major cause of buffer overflows. In C, the length of all character strings is varying and can only be determined by searching for a null byte. By contrast, PL/I character strings may be either fixed length or varying length, but a maximum length must always be specified, either at compile time or in an argument descriptor or in another variable using the REFER option. When PL/I strings are used or copied, the maximum length specifications are honored by the compiled code, resulting in automatic string truncation or padding, even when full string length checking is not enabled. The net result is that a PL/I programmer would have to work very hard to program a buffer overflow error, while a C programmer has to work very hard to avoid programming a buffer overflow error.

Multics added one additional feature in its runtime support that could detect mismatches between calling and called argument descriptors of separately compiled programs and raise an error.

PL/I also provides richer features for arrays and structures. While these differences are not as immediately visible as the character string differences, an algorithm coded in PL/I will have less need for pointers and pointer arithmetic than the same algorithm coded in C. Again, the compiler will do automatic truncation or padding, even when full array bounds checking is not enabled.

While neither PL/I nor C are strongly typed languages and security errors are possible in both languages, PL/I programs tend to suffer significantly fewer security problems than the corresponding C programs.

2.3.2 Hardware Features for Better Security

Multics also avoided many of the current buffer overflow problems through the use of three hardware features. First and most important, Multics used the hardware execute permission bits to ensure that data could not be directly executed. Since most buffer overflow attacks involve branching to data, denying execute permission to data is a very effective countermeasure. Unfortunately, many contemporary operating systems have not used the execute permission features of the x86 segmentation architecture until quite recently [17].

Second, Multics virtual addresses are segmented, and the layout of the ITS pointers is such that an overflow off the end of a segment does not carry into the segment number portion. The net result is that addressing off the end of a segment will always result in a fault, rather than referencing into some other segment. By contrast, other machines that have only paging (such as the VAX, SPARC, or MIPS processors) or that allow carries from the page offset into the segment number (such as the IBM System/370) do not protect as well against overflows. With only paging, a system has to use no-access guard pages that can catch some, but not all references off the end of the data structure. In the case of the x86 processors, although the necessary segmentation features are present, they are almost never used, except for operating systems specifically designed for security, such as GEMSOS [32].

Third, stacks on the Multics processors grew in the positive direction, rather than the negative direction. This meant that if you actually accomplished a buffer overflow, you would be overwriting unused stack frames, rather than your own return pointer, making exploitation much more difficult.

2.4 Minimizing Complexity

Multics achieved much of its security by structuring of the system and by minimizing complexity. It is interest-

1 Burroughs' use of ALGOL for the B5000 operating system was well known to the original Multics designers.

IEEE COMPUTER SOCIETY

ing to compare the size and complexity of Multics of that time with current systems, such as the NSA's Security Enhanced Linux (SELinux). As documented in [41], the ring 0 supervisor of Multics of 1973 occupied about 628K bytes of executable code and read only data. This was considered to be a very large system. By comparison, the size of the SELinux module with the example policy [37] code and read only data has been estimated [22] to be 1767K bytes. This means that just the example security policy of SELinux is more than 2.5 times bigger than the entire 1973 Multics kernel and that doesn't count the size of the Linux kernel itself. These numbers are quite inexact, but they raise a warning. Given that complexity is the biggest single enemy of security, it is important that the SELinux designers examine whether or not there is a complexity problem to be addressed.

3 Malicious Software

One of the major themes of the Multics Security Evaluation was to demonstrate the feasibility of malicious software attacks. Sadly, we were all too successful, and the predictions of the report have been very prophetic.

At that time, we hypothesized that professional penetrators would find that distribution of malicious software would prove to be the attack of choice. In the 1970s with the Cold War raging, our assumption was that the most immediate professional penetrators would be foreign espionage agents and that the defense establishment would be the target of choice, but we expected commercial penetrators to follow. Of course, malicious software (viruses, worms, Trojan horses, etc.) is very common on the Internet today, targeting commercial and university interests as well as the defense establishment. For example, **The New York Times** has reported [36] on surreptitious add-ons to certain peer-to-peer file sharing utilities that can also divert the revenue stream of commissions paid to affiliates of electronic commerce web sites. While the legal status of such add-ons is unclear, the reaction of affiliates who have had their commissions diverted has been very negative. Anderson [8] shows some additional modern examples.

3.1 Installing Trap Doors

To demonstrate our hypothesis, we not only looked for exploitable flaws in Multics security, but we also attempted to plant trap doors into Multics to see if they could be found during quality assurance and whether they would be distributed to customer sites. Section 3.4.5.2 of the report described a trap door that was installed in the 645 processor, but was not distributed, as the 645 was being phased out. This trap door did demonstrate a property expected of real-world trap doors – its activation was triggered by a password or key that insured the vulnerability was not invoked by accident or discovered by quality assurance or other testing.

Section 3.4.6 gave a brief hint of a trap door that was installed into the 6180 development system at MIT in the routine hcs_$set_ring_brackets_. We verified that the trap door was distributed by Honeywell to the 6180 processor that was installed at the Air Force Data Services Center (AFDSC) in the basement of the Pentagon in an area that is now part of the Metro station. Despite the hint in section 3.4.6, the trap door was not discovered until roughly a year after the report was published. There was an intensive security audit going on at the General Motors Multics site, and the trap door, which was neutered to actually be benign, (it needed a one instruction change to be actively malicious) gave an anomalous error return. The finding of the trap door is described on the Multics web site [45], although the installation of the trap door is incorrectly attributed to The MITRE Corporation.

3.2 Malicious Software Predictions

In addition to demonstrating the feasibility of installing a trap door into commercial software and having the manufacturer then distribute it to every customer in the world, the report also hypothesized a variety of other malicious software attacks that have unfortunately all come true. Section 3.4.5.1 proposed a variety of possible attack points.

3.2.1 Malicious Developers

We suggested that malicious developers might create malicious software. Since that time, we have seen many products with either surreptitious backdoors that allow the developer to gain access to customer machines or with so-called Easter eggs that are simply concealed pieces of code that do humorous things when activated. As expected of malicious trapdoors, activation of most Easter eggs is triggered by a unique key or password that is not likely to be encountered in even extensive quality assurance testing. In most cases, the Easter eggs have NOT been authorized by the development managers, and are good examples of how developers can insert unauthorized code. The primary difference between an Easter egg and a piece of malicious software is the developer's intent. Fortunately for most instances, the developers are not malicious.

3.2.2 Trap Doors During Distribution

The report predicted trap doors inserted into the distribution chain for software. Today, we frequently see bogus e-mail or downloads claiming to contain important updates to widely used software that in fact contain Trojan horses. One very recent incident of this kind is reported in [19].

3.2.3 Boot-Sector Viruses

The report predicted trap doors during installation and booting. Today, we have boot sector viruses that are quite common in the wild.

IEEE COMPUTER SOCIETY

3.2.4 Compiler Trap Doors

The trap doors described in the report were patches to the binary object files of the system. The report suggested a countermeasure to such object code trap doors by having customers recompile the system from source, although the report also notes that this could play directly into the hands of the penetrator who has made changes in the source code. In fact, the AFDSC Multics contract specifically required that Honeywell deliver source code to the Pentagon to permit such recompilations. However, the report pointed out the possibility that a trap door in the PL/I compiler could install trap doors into the Multics operating system when modules were compiled and could maintain its own existence by recognizing when the PL/I compiler was compiling itself. This recognition was the basis several years later for the TCSEC Class A1 requirement for generation of new versions from source using a compiler maintained under strict configuration control. A recent story on CNET news [14] reports that the Chinese government has similar concerns about planted trap doors.

This suggestion proved an inspiration to Ken Thompson who actually implemented the self-inserting compiler trap door into an early version of UNIX. Thompson described his trap door in his 1984 Turing Award paper [40], and attributed the idea to an "unknown Air Force document," and he asked for a better citation. The document in question is in fact the Multics Security Evaluation report, and we gave a new copy of the report to Thompson after his paper was published. Thompson has corrected his citation in a reprint of the paper [39].

3.3 Auditing and Intrusion Detection

Multics had some limited auditing capabilities (basically recording date time modified (DTM) on files) at the time of the original vulnerability analysis, although a great deal more auditing was added as part of the Multics Security Enhancements project [44]. Section 3.4.4 showed how a professional attacker could bypass the auditing capabilities while installing malicious software.

This has major implications for today, because intrusion detection has become a major area of security product development. Unfortunately, many of the intrusion detection systems (IDSs) that have been developed do not deal well with the possibility of a professional attacker going in and erasing the evidence before it can be reported to an administrator to take action. Most IDSs rely on pattern recognition techniques to identify behaviors that are indicative of an attack. However, a professional penetrator would do as we did – debug the penetration programs on a duplicate machine, the Rome Air Defense Center Multics system in our case, before attacking the desired target, the MIT site in our case. This would give the IDS only a single isolated anomaly to detect, and the IDS would have to respond before the professional attacker could erase the evidence as we did with the Multics DTM data.

Most evaluations of IDS systems have focused on how well the IDS responds to a series of pre-recorded traces. A more relevant evaluation of an IDS system would be to see how it responds to a well-prepared and well-funded professional (but ethical) attacker who is prepared to erase the auditing data and to install trap doors immediately for later exploitation, or subvert the operating system on which the IDS is running to selectively bypass the IDS all together. This type of selectively triggered attack of a limited subsystem (i.e., the IDS) through a subverted host operating system is illustrated by the "key string trigger trap door" in section 3.4.5.1 of the report.

4 What Happened Next?

This section discusses the immediate outcomes from the Vulnerability Analysis.

4.1 Multics Security Enhancements

The primary outcome of the vulnerability analysis, and indeed the reason it was done at all was the Multics security enhancements effort [44]. The Air Force had initially approached Honeywell with a proposal for an R&D effort to improve Multics security for the Air Force Data Services Center, but Honeywell initially declined on the basis that Multics was already sufficiently secure. The vulnerability analysis was conducted to provide evidence of the need for significant enhancements. This security enhanced version of Multics ultimately achieved a Class B2 evaluation from the National Computer Security Center in 1985 [3].

4.2 Publicity

The results of the vulnerability received a certain level of press publicity. **Fortune** magazine published [7] an interview with Steve Lipner of the MITRE Corporation that was a bit sensationalized. It focused on the illegal operator's console message described in section 3.3.4.3. That message quoted from a Honeywell press release that appeared in an internal Honeywell newsletter and a Phoenix, AZ newspaper article[2] that incorrectly stated that the Air Force had found some security problems in Multics, but they were all fixed and that the Air Force now "certified" Multics security. Not only was the Honeywell press release inaccurate, but it had not been cleared for public release, as required in the Multics security enhancements contract. While the **Fortune** article was a bit sensational, the story was then retold in a text book [26] in a version that was barely recognizable to us.

A much better description of the work was done by Whiteside[43], based on his earlier articles for **The New Yorker**. Whiteside interviewed both Schell and Lipner and did a much more accurate account of the effort.

2 The authors have been unable to find copies of these articles and would appreciate any information on how to locate them.

IEEE COMPUTER SOCIETY

4.3 Multics Kernel Design Project

As the report suggested in section 4.2.2, a project was started to build a security kernel version of Multics, one intended to ultimately meet what was later defined as the requirements of a Class A1 evaluation of the Orange Book [2]. This effort was conducted jointly by Honeywell, MIT, the MITRE Corporation, and the Air Force with results published in: [10, 33-35, 38].

4.4 Direction to Stop Work

In August 1976, Air Force Systems Command directed termination of the Air Force's computer security work, including the Multics Kernel Design Project, despite the fact that the work was quite successful up to that point, had widespread support from user agencies in the Army, the Navy, and the Air Force, and had been by DoD standards, quite inexpensive. The adverse impact of that termination on the DoD is described in [31]. Another view of those events can be seen in [23, pp. II-74 – II-75] and in the GAO report on the termination [4].

5 Dispelling Mythology

The Multics vulnerability analysis got enough publicity that a certain amount of mythology developed about what happened. We hope this section can dispel at least some of that mythology.

One story that circulated was that the penetrations were only possible, because the authors had special privileged access to Multics. While both authors had been members of the MIT Multics development team (Roger Schell as a Ph.D. student and Paul Karger as an undergraduate), both had left the Multics team months before the vulnerability analysis was even considered. No privileged accounts on either of those systems were used in the analysis. (Privileged accounts were used on RADC-Multics and at the Air Force Data Services Center, but those were only to show that a penetrator could take advantage of a duplicate system to debug penetrations ahead of time and to verify the distribution of the trap door.) The Air Force purchased computer time on the MIT-Multics system and on Honeywell's Multics system in Phoenix, AZ, but both systems routinely sold computer time. Multics source code was readable by any user on the MIT site, much as source code for open source systems is generally available today.

The entire vulnerability analysis was carried out with a ground rule of not causing any real harm to any of the systems under attack. All the documentation and listings of exploit programs was kept in a safe approved for storing TOP SECRET information (although that particular safe actually had no classified material in it). Decrypted password files were also stored in the safe, to ensure that no user accounts would be compromised. Exploit programs were protected with very restrictive access control lists and kept encrypted at all times, except when actually in use. The applicable contracts specified that the purchased computer time at MIT and Phoenix would be used for security testing and evaluation purposes, so there was no question of the legality of the analysis. Finally, publication of [24] was delayed until after all of the penetrations described were repaired.

For several years after the completion of the vulnerability analysis, the authors were periodically contacted when unexplained system crashes occurred on the MIT site, just in case they had been caused by a remaining hidden trap door. However, as part of the basic policy of the analysis to cause no real harm, no trap doors were left behind after the conclusion of the vulnerability analysis.

6 Security has gotten worse, not better

Today, government and commercial interest in achieving processing and connectivity encompassing disparate security interests is driving efforts to provide security enhancements to contemporary operating systems [27]. These efforts include the addition of mandatory access controls and "hardening" (i.e., the removal of unnecessary services.) Such enhancements result in systems less secure than Multics, and Multics, with its security enhancements, was only deemed suitable for processing in a relatively benign "closed" environment [5]. It was concluded that operation of Multics in an "open" environment would require restructuring around a verifiable security kernel to address the threat of professional attacks using malicious software (e.g., trap doors).

6.1 Weak Solutions in Open Environments

Given the understanding of system vulnerabilities that existed nearly thirty years ago, today's "security enhanced" or "trusted" systems would not be considered suitable for processing even in the benign closed environment. Also, considering the extent of network interconnectivity and the amount of commercial off-the-shelf (COTS) and other software without pedigree (e.g., libraries mined from the Web), today's military and commercial environments would be considered very much "open". To make matters worse, everyone is now expected to be a system administrator of their own desktop system, but without any training!

Furthermore, quality assurance testing and subjecting systems to authorized (ethical) hackers are among the most common approaches for concluding that the security of such systems is adequate. There have even been apparently serious proposals [1] for something as important as voting to use student hackers "to test computerized voting systems". Yet it is evident on the face of it that such approaches are utterly useless for detecting the vulnerability most attractive to a professional – malicious software triggered by a unique key or password unknown to those making this assessment.[3]

3 Note that this does not mean that penetration testing or ethical hacking has no use. One of the authors currently works in IBM's Global Security Analysis Laboratory, an ethical hacking organization. Penetration

IEEE COMPUTER SOCIETY

Thus, systems that are weaker than Multics are considered for use in environments in excess of what even Multics could deliver without restructuring around a security kernel. There really seem to be only four possible conclusions from this: either (1) today's systems are really much more secure than we claim; (2) today's potential attackers are much less capable or motivated; (3) the information being processed is much less valuable; or (4) people are unwilling or unable to recognize the compelling need to employ much better technical solutions.

6.2 Technical Solutions

Though Multics was never restructured around a security kernel as recommended, the technical ability to structure a full featured commercial operating system around a kernel designed to meet Class A1 while maintaining good operating system and application compatibility was demonstrated for the VAX [25]. Successful fielding on the open Internet of a highly secure system using the popular x86 processor with the commercial NSA-certified Class A1 GEMSOS [32] security kernel for VPN security management has also been demonstrated [42].

In the nearly thirty years since the report, it has been demonstrated that the technology direction that was speculative at the time can actually be implemented and provides an effective solution to the problem of malicious software employed by well-motivated professionals. Unfortunately, the mainstream products of major vendors largely ignore these demonstrated technologies. In their defense most of the vendors would claim that the marketplace is not prepared to pay for a high assurance of security. And customers have said they have never been offered mainstream commercial products that give them such a choice, so they are left with several ineffective solutions collected under marketing titles like "defense in depth".

In our opinion this is an unstable state of affairs. It is unthinkable that another thirty years will go by without one of two occurrences: either there will be horrific cyber disasters that will deprive society of much of the value computers can provide, or the available technology will be delivered, and hopefully enhanced, in products that provide effective security. We hope it will be the latter.

7 Acknowledgments

We must thank Iris Partlow of the IBM Business Support Center for retyping the 1974 report. Very helpful comments came from Mike Thompson of Aesec Corporation, Jeremy Epstein of webMethods, Wietse Venema and David Safford of IBM Research, Steve Sutton of PatientKeeper, and Tom Van Vleck of NAI Labs. Tom also runs the Multics web site (http://www.multicians.org), where we found some of the information needed for this commentary. Tom was the system administrator and a designer of Multics, so we also thank him (nearly 30 years later) for understanding that our work was to make Multics more secure, even if its presentation was considered overly dramatic at times.

testing can identify some vulnerabilities for repair. However, the failure of any particular penetration team to find vulnerabilities does not imply their absence. It only indicates that the team failed to find any.

REFERENCES

1. *Broward Officials Want Students to Try Hacking Mock Election*, in *Assoc. Press* 16 August 2001: Ft. Lauderdale, FL. URL: http://www.wtsp.com/news/2001_08/16_mock_election.htm

2. *Department of Defense Trusted Computer System Evaluation Criteria*, DOD 5200.28-STD, December 1985: Washington, DC. URL: http://csrc.nist.gov/publications/history/dod85.pdf

3. *Final Evaluation Report of Multics, MR11.0*, CSC-EPL-85/003 (releasable only to US Government and their contractors), 1 September 1985, National Computer Security Center: Ft. George G. Meade, MD. URL: http://www.radium.ncsc.mil/tpep/library/fers/tcsec_fers.html

4. *Multilevel Computer Security Requirements of the World Wide Military Command and Control System (WWMCCS)*, LCD-78-106, 5 April 1978, General Accounting Office: Washington, DC. URL: http://www.gao.gov/docdblite/summary.php?&rptno=LCD-78-106

5. *Technical Rationale Behind CSC-STD-003-85: Computer Security Requirements -- Guidance for Applying the Department of Defense Trusted Computer System Evaluation Criteria in Specific Environments*, CSC-STD-004-85, 25 June 1985, DoD Computer Security Center: Ft. George G. Meade, MD.

6. Aleph One, *Smashing the Stack for Fun and Profit.* **Phrack**, 8 November 1996. **7**(49). URL: http://www.phrack.org/show.php?p=49&a=14

7. Alexander, T., *Waiting for the Great Computer Rip-off.* **Fortune**, 1974. **XC**(1): p. 142-150.

8. Anderson, E.A., III, *A Demonstration of the Subversion Threat: Facing a Critical Responsibility in the Defense of Cyberspace*, M.S. in *Computer Science* 2002, Naval Postgraduate School: Monterrey, CA. URL: http://theses.nps.navy.mil/Thesis_02Mar_Anderson_Emory.pdf

9. Berger, J.L., J. Picciotto*, et al.*, *Compartmented Mode Workstation: Prototype Highlights.* **IEEE Transactions on Software Engineering**, June 1990. **16**(6): p. 608-618.

10. Biba, K.J., S.R. Ames*, et al.*, *A Preliminary Specification of a Multics Security Kernel*, WP-20119, April 1975, The MITRE Corporation: Bedford, MA.

11. Blotcky, S., K. Lynch*, et al. SE/VMS: Implementing Mandatory Security in VAX/VMS*. in **Proceedings of the 9th National Computer Security Conference**. 15-18 September 1986, Gaithersburg, MD National Bureau of Standards. p. 47-54.

12. Butler, R., V. Welch*, et al.*, *A National-Scale Authentication Infrastructure.* **Computer**, December 2000. **33**(12): p. 60-66.

IEEE COMPUTER SOCIETY

13. Carr, D.F., *The Hot Hand in the ASP Game.* **Internet World**, 15 January 2000. **6**(2): p. 53. URL: http://www.internetworld.com/magazine.php?inc=011500/1.15cover_story.html

14. Cooper, C., *Who says paranoia doesn't pay off?* **CNET News.com**, 20 September 2002. URL: http://news.com.com/2010-1071-958721.html

15. Corbató, F.J., *PL/I As a Tool for System Programming.* **Datamation**, May 1969. **15**(5): p. 68-76. URL: http://home.nycap.rr.com/pflass/plisprg.htm

16. Corbató, F.J. and V.A. Vyssotsky. *Introduction and Overview of the Multics System.* in **Fall Joint Computer Conference**. 1965, Washington, DC Vol. AFIPS Conference Proceedings Vol. 27. Spartan Books. p. 185-196. URL: http://www.multicians.org/fjcc1.html

17. Cowan, C., P. Wagle*, et al. Buffer Overflows: Attacks and Defenses for the Vulnerability of the Decade*. in **DARPA Information Survivability Conference and Expo (DISCEX)**. 25-27 January 2000, Hilton Head Island, SC Vol. 2. p. 119-129. URL: http://www.immunix.org/StackGuard/discex00.pdf

18. David, E.E., Jr. and R.M. Fano. *Some Thoughts About the Social Implications of Accessible Computing*. in **Fall Joint Computer Conference**. 1965, Washington, DC Vol. AFIPS Conference Proceedings Vol. 27. Spartan Books. p. 243-247. URL: http://www.multicians.org/fjcc6.html

19. Dougherty, C., *Trojan Horse OpenSSH Distribution*, CERT® Advisory CA-2002-24, 2 August 2002, CERT Coordination Center, Carnegie Mellon University: Pittsburgh, PA. URL: http://www.cert.org/advisories/CA-2002-24.html

20. Forsdick, H.C. and D.P. Reed, *Patterns of Security Violations: Multiple References to Arguments*, in *Ancillary Reports: Kernel Design Project*, D.D. Clark, Editor, June 1977, MIT/LCS/TM-87, Laboratory for Computer Science, Massachusetts Institute of Technology: Cambridge, MA. p. 34-49.

21. Foster, I., C. Kesselman*, et al. A Security Architecture for Computational Grids*. in **Proceedings of the 5th ACM Conference on Computer and Communications Security**. 2-5 November 1998, San Francisco, CA p. 83-92.

22. Jaeger, T., *Personal Communication*, 30 July 2002, IBM Corp., T. J. Watson Research Center: Hawthorne, NY.

23. Jelen, G.F., *Information Security: An Elusive Goal*, P-85-8, June 1985, Program on Information Resources Policy, Harvard University: Cambridge, MA. URL: http://www.pirp.harvard.edu/publications/pdf-blurb.asp?id=238

24. Karger, P.A. and R.R. Schell, *Multics Security Evaluation: Vulnerability Analysis*, ESD-TR-74-193, Vol. II, June 1974, HQ Electronic Systems Division: Hanscom AFB, MA. URL: http://csrc.nist.gov/publications/history/karg74.pdf

25. Karger, P.A., M.E. Zurko*, et al.*, *A Retrospective on the VAX VMM Security Kernel.* **IEEE Transactions on Software Engineering**, November 1991. **17**(11): p. 1147-1165.

26. Logsdon, T., *Computers & Social Controversy*, 1980, Computer Science Press: Potomoc, MD. p. 170-175, 185.

27. Loscocco, P. and S. Smalley. *Integrating Flexible Support for Security Policies into the Linux Operating System*. in **Proceedings of the FREENIX Track: 2001 USENIX Annual Technical Conference (FREENIX '01)**. 2001, Boston, MA. URL: http://www.nsa.gov/selinux/doc/freenix01.pdf

28. Saltzer, J.H., *Repaired Security Bugs in Multics*, in *Ancillary Reports: Kernel Design Project*, D.D. Clark, Editor, June 1977, MIT/LCS/TM-87, Laboratory for Computer Science, Massachusetts Institute of Technology: Cambridge, MA. p. 1-4.

29. Saltzer, J.H. and D. Hunt, *Some Recently Repaired Security Holes of Multics*, in *Ancillary Reports: Kernel Design Project*, D.D. Clark, Editor, June 1977, MIT/LCS/TM-87, Laboratory for Computer Science, Massachusetts Institute of Technology: Cambridge, MA. p. 28-33.

30. Saltzer, J.H., P. Jansen*, et al.*, *Some Multics Security Holes Which Were Closed by 6180 Hardware*, in *Ancillary Reports: Kernel Design Project*, D.D. Clark, Editor, June 1977, MIT/LCS/TM-87, Laboratory for Computer Science, Massachusetts Institute of Technology: Cambridge, MA. p. 22-27.

31. Schell, R.R. *Computer security: the Achilles' heel of the electronic Air Force?* in **Air University Review**. January-February 1979, Vol. 30. p. 16-33. URL: http://www.airpower.maxwell.af.mil/airchronicles/aureview/1979/jan-feb/schell.html

32. Schell, R.R., T.F. Tao*, et al. Designing the GEMSOS Security Kernel for Security and Performance*. in **Proceedings of the 8th National Computer Security Conference**. 30 September - 3 October 1985, Gaithersburg, MD DoD Computer Security Center and National Bureau of Standards. p. 108-119.

33. Schiller, W.L., P.T. Withington*, et al.*, *Top Level Specification of a Multics Security Kernel*, WP-20810, 9 July 1976, The MITRE Corporation: Bedford, MA.

34. Schroeder, M.D., *Engineering a Security Kernel for Multics.* **Proceedings of the Fifth Symposium on Operating Systems Principles, Operating Systems Review**, November 1975. **9**(5): p. 25-32.

35. Schroeder, M.D., D.D. Clark*, et al.*, *The Multics Kernel Design Project.* **Proceedings of the Seventh ACM Symposium on Operating Systems Principles, Operating Systems Review**, November 1977. **11**(5): p. 43-56.

36. Schwartz, J. and B. Tedeschi, *New Software Quietly Diverts Sales Commissions.* **The New York Times**, 27 September 2002: p. C1-C4. URL: http://www.nytimes.com/2002/09/27/technology/27FREE.html

37. Smalley, S., *Configuring the SELinux Policy*, NAI Labs Report #02-007, June 2002, NAI Labs: Glenwood, MD. URL: http://www.nsa.gov/selinux/policy2-abs.html

38. Stern, J., *Multics Security Kernel Top Level Specification*, ESD-TR-76-368, November 1976, Honeywell Information Sys-

IEEE COMPUTER SOCIETY

tems, Inc., McLean, VA, HQ Electronic Systems Division: Hanscom AFB, MA.

39. Thompson, K., *On Trusting Trust.* **Unix Review**, November 1989. **7**(11): p. 70-74.

40. Thompson, K., *Reflections on Trusting Trust.* **Communications of the ACM**, August 1984. **27**(8): p. 761-763.

41. Voydock, V.L., *A Census of Ring 0*, in *Ancillary Reports: Kernel Design Project*, D.D. Clark, Editor, June 1977, MIT/LCS/TM-87, Laboratory for Computer Science, Massachusetts Institute of Technology: Cambridge, MA. p. 5-27.

42. Weissman, C. *BLACKER: Security for the DDN. Examples of A1 Security Engineering Trades*. in **1992 IEEE Computer Society Symposium on Research in Security and Privacy**. 4-6 May 1992, Oakland, CA p. 286-292.

43. Whiteside, T., *Trapdoors and Trojan Horses*, in *Computer Capers*. 1978, Thomas Y. Crowell Co.: New York. p. 115-126.

44. Whitmore, J., A. Bensoussan*, et al.*, *Design for Multics Security Enhancements*, ESD-TR-74-176, December 1973, Honeywell Information Systems, Inc., HQ Electronic Systems Division: Hanscom AFB, MA. URL: http://csrc.nist.gov/publications/history/whit74.pdf

45. Zemmin, F., B. Henk*, et al.*, *Multics Site History: GM*, T.H. Van Vleck, Editor, 20 October 1998. URL: http://www.multicians.org/site-gm.html

IEEE COMPUTER SOCIETY

Multics Security Evaluation: Vulnerability Analysis*

Paul A. Karger, 2Lt, USAF Roger R. Schell, Maj, USAF
Deputy for Command and Management Systems (MCI), HQ Electronic Systems Division
Hanscom AFB, MA 01730

ABSTRACT

A security evaluation of Multics for potential use as a two-level (Secret / Top Secret) system in the Air Force Data Services Center (AFDSC) is presented. An overview is provided of the present implementation of the Multics Security controls. The report then details the results of a penetration exercise of Multics on the HIS 645 computer. In addition, preliminary results of a penetration exercise of Multics on the new HIS 6180 computer are presented. The report concludes that Multics as implemented today is not certifiably secure and cannot be used in an open use multi-level system. However, the Multics security design principles are significantly better than other contemporary systems. Thus, Multics as implemented today, can be used in a benign Secret / Top Secret environment. In addition, Multics forms a base from which a certifiably secure open use multi-level system can be developed.

1 INTRODUCTION

1.1 Status of Multi-Level Security

A major problem with computing systems in the military today is the lack of effective multi-level security controls. The term multi-level security controls 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 differing levels of clearances. The lack of such effective controls in all of today's computer operating systems has led the military to operate computers in a closed environment in which systems are dedicated to the highest level of classified material and all users are required to be cleared to that level. Systems may be changed from level to level, but only after going through very time consuming clearing operations on all devices in the system. Such dedicated systems result in extremely inefficient equipment and manpower utilization and have often resulted in the acquisition of much more hardware than would otherwise be necessary. In addition, many operational requirements cannot be met by dedicated systems because of the lack of information sharing. It has been estimated by the Electronic Systems Division (ESD) sponsored Computer Security Technology Panel [10] that these additional costs may amount to $100,000,000 per year for the Air Force alone.

1.2 Requirement for Multics Security Evaluation

This evaluation of the security of the Multics system was performed under Project 6917, Program Element 64708F to meet requirements of the Air Force Data Services Center (AFDSC). AFDSC must provide responsive interactive time-shared computer services to users within the Pentagon at all classification levels from unclassified to top secret. 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 only unclassified material. In a separate study completed in February 1972, the Information Systems Technology Applications Office, Electronic Systems Division (ESD/MCI) identified the Honeywell Multics system as a candidate to meet both AFDSC's multi-level security requirements and highly responsive advanced interactive time-sharing requirements.

1.3 Technical Requirements for Multi-Level Security

The ESD-sponsored Computer Security Technology Planning Study [10] outlined the security weaknesses of present day computer systems and proposed a development plan to provide solutions base on current technology. A brief summary of the findings of the panel follows.

1.3.1 Insecurity of Current Systems

The internal controls of current computers repeatedly have been shown insecure though numerous penetration exercises on such systems as GCOS [9], WWMCCS GCOS [8, 18], and IBM OS/360/370 [16]. This insecurity is a fundamental weakness of contemporary operating sys-

* This article is a reprint of a technical report [19] published in June 1974. The program listings from the appendices have been omitted, due to space constraints. The text has been retyped and the figures redrawn, but with no substantive changes. The references have been updated, as some were not yet in final form in 1974.

Proceedings of the 18th Annual Computer Security Applications Conference (ACSAC'02)
1063-9527/02 $17.00 © 2002 IEEE

IEEE COMPUTER SOCIETY

tems and cannot be corrected by "patches", "fix-ups", or "add-ons" to those systems. Rather, a fundamental re-implementation using an integrated hardware/software design which considers security as a fundamental requirement is necessary. In particular, steps must be taken to ensure the correctness of the security related portions of the operating system. It is not sufficient to use a team of experts to "test" the security controls of a system. Such a "tiger team" can only show the existence of vulnerabilities but cannot prove their non-existence.

Unfortunately, the managers of successfully penetrated computer systems are very reluctant to permit release of the details of the penetrations. Thus, most reports of penetrations have severe (and often unjustified) distribution restrictions leaving very few documents in the public domain. Concealment of such penetrations does nothing to deter a sophisticated penetrator and can in fact impede technical interchange and delay the development of a proper solution. A system which contains vulnerabilities cannot be protected by keeping those vulnerabilities secret. It can only be protected by the constraining of physical access to the system.

1.3.2 Reference Monitor Concept

The ESD Computer Security Technology Panel introduced the concept of a *reference monitor*. This reference monitor is that hardware/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. Three conditions must be met to ensure the security of the system based on a reference monitor.

a. The monitor must be tamper proof.
b. The monitor must be invoked for *every* reference to data anywhere in the system.
c. The monitor must be small enough to be proven correct.

The stated design goals of contemporary systems such as GCOS or OS/360 are to meet the first requirement (albeit unsuccessfully). The second requirement is generally not met by contemporary systems since they usually include "bypasses" to permit special software to operate or must suspend the reference monitor to provide addressability for the operating system in exercising its service functions. The best known of these is the bypass in OS/360 for the IBM supplied service aid, IMASPZAP (SUPERZAP) [2]. Finally and most important, current operating systems are so large, so complex, and so monolithic that one cannot begin to attempt a formal proof or certification of their correct implementation.

1.3.3 Hypothesis: Multics is "Securable"

The computer security technology panel identified the general class of descriptor driven processors[1] as extremely useful to the implementation of a reference monitor. Multics, as the most sophisticated of the descriptor-driven systems currently available, was hypothesized to be a potentially securable system; that is, the Multics design was sufficiently well-organized and oriented towards security that the concept of a reference monitor could be implemented for Multics without fundamental changes to the facilities seen by Multics users. In particular, the Multics ring mechanism could protect the monitor from malicious or inadvertent tampering, and the Multics segmentation could enforce monitor mediation on *every* reference to data. However, the question of certifiability had not as yet been addressed in Multics. Therefore the Multics vulnerability analysis described herein was undertaken to:

a. Examine Multics for potential vulnerabilities.
b. Identify whether a reference monitor was practical for Multics.
c. Identify potential interim enhancements to Multics to provide security in a benign (restricted access) environment.
d. Determine the scope and dimension of a certification effort.

1.4 Sites Used

The vulnerability analysis described herein was carried out on the HIS 645 Multics Systems installed at the Massachusetts Institute of Technology and at the Rome Air Development Center. As the HIS 6180, the new Multics processor, was not available at the time of the study, this report will describe results of analysis of the HIS 645 only. Since the completion of the analysis, work has started on an evaluation of the security controls of Multics on the HIS 6180. Preliminary results of the work on the HIS 6180 are very briefly summarized in this report, to provide an understanding of the value of the evaluation of the HIS 645 in the context of the new hardware environment.

2 MULTICS SECURITY CONTROLS

This section provides a brief overview of the basic Multics security controls to provide necessary background for the discussion of the vulnerability analysis. However, a rather thorough knowledge of the Multics implementation is assumed throughout the rest of this document. More complete background material may be found in Lipner [21], Saltzer [25], Organick [22], and the **Multics Programmers' Manual** [4].

The basic security controls of Multics fall into three major areas: hardware controls, software controls, and procedural controls. This overview will touch briefly on each of these areas.

1 Descriptor driven processors use some form of address translation though hardware interpretation of descriptor words or registers. Such systems include the Burroughs 6700, the Digital Equipment Corp. PDP-11/45, the Data General Nova 840, the DEC KI-10, the HIS 6180, the IBM 370/158 and 168, and several others not listed here.

IEEE COMPUTER SOCIETY

2.1 Hardware Security Controls

2.1.1 Segmentation Hardware

The most fundamental security controls in the HIS 645 Multics are found in the segmentation hardware. The basic instructions set of the 645 can directly address up to 256K[2] distinct segments[3] at any one time, each segment being up to 256K words long.[4] Segments are broken up into 1K word pages[5] which can be moved between primary and secondary storage by software, creating a very large virtual memory. However, we will not treat paging throughout most of this evaluation as it is transparent to security. Paging must be implemented correctly in a secure system. However, bugs in page control are generally difficult to exploit in a penetration, because the user has little or no control over paging operations.

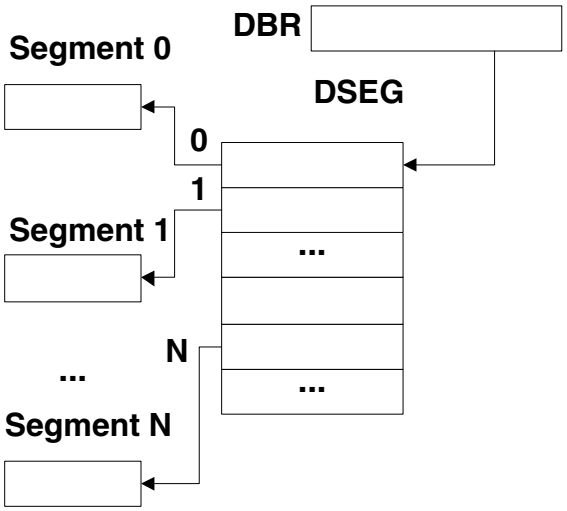

Figure 1. Segmentation Hardware

Segments are accessed by the 645 CPU through segment descriptor words (SDW's) that are stored in the descriptor segment (DSEG). (See Figure 1.) To access segment N, the 645 CPU uses a processor register, the descriptor segment base register (DBR), to find the DSEG. It then accesses the Nth SDW in the DSEG to obtain the address of the segment and the access rights currently in force on that segment for the current user.

2 1K = 1024 units.

3 Current software table sizes restrict a process to about 1000 segments. However, by increasing these table sizes, the full hardware potential may be used.

4 The 645 software restricted segments to 64K words for efficiency reasons.

5 The 645 hardware also support 64 word pages which were not used. The 6180 supports only a single page size which can be varied by field modification from 64 words to 4096 words. Initially, a size of 1024 words is being used. The supervisors on both the 645 and 6180 use unpaged segments of length 0 mod 64.

Each SDW contains the absolute address of the page table for the segment and the access control information. (See Figure 2.) The last 6 bits of the SDW determine the access rights to the segment - read, execute, write, etc.[6] Using these access control bits, the supervisor can protect the descriptor segment from unauthorized modification by denying access in the SDW for the descriptor segment.

0 17	18 29	30	31	32	33 35
ADDR	OTHER	WRITE PERMIT	SLAVE ACC.	OTHER	CLASS

Meaning of CLASS field
0 = FAULT
1 = DATA
2 = SLAVE PROCEDURE
3 = EXECUTE ONLY
4 = MASTER PROCEDURE
5, 6, 7 = ILLEGAL DESCRIPTOR

Figure 2. SDW Format

2.1.2 Master Mode

To protect against unauthorized modification of the DBR, the processor operates in one of two states – master mode and slave mode. In master mode, any instruction may be executed and access control checks are inhibited.[7] In slave mode, certain instructions, including those which modify the DBR, are inhibited. Master mode procedure segments are controlled by the class field in the SDW. Slave mode procedures may transfer to master mode procedures *only* through word zero of the master mode procedure to prevent unrestricted invocation of privileged programs. It is then the responsibility of the master mode software to protect itself from malicious calls by placing suitable protective routines beginning at location zero.

2.2 Software Security Controls

The most outstanding feature of the Multics security controls is that they operate on basis of "form" rather than the classical basis of "content". That is to say, the Multics controls are based on operations on a uniform population of well defined objects, as opposed to the classical controls which rely on anticipating all possible types of accesses and make security essentially a battle of wits.

2.2.1 Protection Rings

The primary software security control on the 645 Multics system is the ring mechanism. It was originally postulated as desirable to extend the traditional master/slave mode relationship of conventional machines to permit layering within the supervisor and within user code (see Gra-

6 A more detailed description of the SDW format may be found in the 645 processor manual [11].

7 The counterpart of master mode on the HIS 6180, called privileged mode, does not inhibit access control checking.

1063-9527/02 $17.00 © 2002 IEEE

IEEE COMPUTER SOCIETY

ham [17]). Eight concentric rings of protection, numbered 0 – 7, are defined with higher numbered rings having less privilege than lower numbered rings, and with ring 0 containing the *hardcore* supervisor.[8] Unfortunately, the 645 CPU does not implement protection rings in hardware.[9] Therefore, the eight protection rings are implemented by providing eight descriptor segments for each process (user), one descriptor segment per ring. Special fault codes are placed in those SDW's which can be used for cross-ring transfers so that ring 0 software can intervene and accomplish the descriptor segment swap between the calling and called rings.

2.2.2 Access Control Lists

Segments in Multics are stored in a hierarchy of directories. A directory is a special type of segment that is not directly accessible to the user and provides a place to store names and other information about subordinate segments and directories. Each segment and directory has an access control list (ACL) in its parent directory entry controlling who may *r*ead (r), *w*rite (w), or *e*xecute (e) the segment or obtain *s*tatus (s) of, *m*odify (m) entries in, or *a*ppend (a) entries to a directory. For example in Figure 3, the user Jones.Druid has read permission to segment ALPHA and has null access to segment BETA. However, Jones.Druid has modify permission to directory DELTA, so he can give himself access to BETA. Jones.Druid cannot give himself write access to segment ALPHA, because he does not have modify permission to directory GAMMA. In turn, the right to modify the access control lists of GAMMA and DELTA is controlled by the access control list of directory EPSILON, stored in the parent of EPSILON. Access control security checks for segments are enforced by the ring 0 software by setting the appropriate bits in the SDW at the time that a user attempts to add a segment to his address space.

2.2.3 Protected Access Identification

In order to do access checking, the ring 0 software must have a protected, non-forgeable identification of a user to compare with the ACL entries. This ID is established when a user signs on to Multics and is stored in the process data segment (PDS) which is accessible only in ring 0 or in master mode, so that the user may not tamper with the data stored in the PDS.

2.2.4 Master Mode Conventions

By convention, to protect master mode software, the original design specified that master mode procedures were not to be used outside ring 0. If the master mode procedure ran in the user ring, the master mode procedure itself would be forced to play the endless game of wits of the classical supervisor call. The master mode procedure would have to include code to check for all possible combinations of input arguments, rather than relying on a fundamental set of argument independent security controls. As an aid (or perhaps hindrance) to playing the game of wits, 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 code saves the registers for debugging and brings the system down.

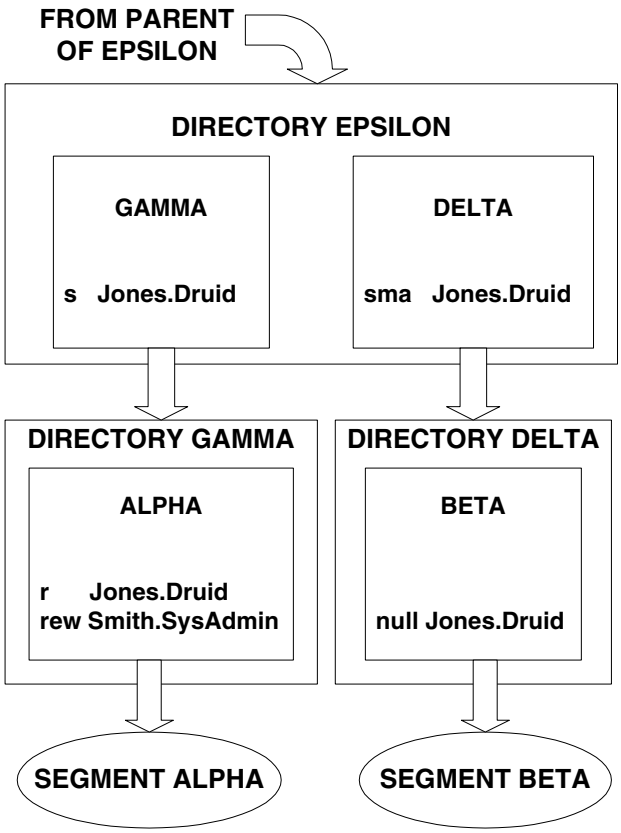

Figure 3. Directory Hierarchy

2.3 Procedural Security Controls

2.3.1 Enciphered Passwords

When a user logs in to Multics, he types a password as his primary authentication. Of course, the access control list of the password file denies access to regular users of the system. In addition, as a protection against loss of a system dump which could contain the password file, all passwords are stored in a *non-invertible* cipher form. When a user types his password, it is enciphered and com-

8 The original design called for 64 rings, but this was reduced to 8 in 1971.

9 One of the primary enhancements of the HIS 6180 is the addition of ring hardware [28] and a consequent elimination of the need for master mode procedures in the user ring.

IEEE COMPUTER SOCIETY

pared with the stored enciphered version for validity. Clear text passwords are stored nowhere in the system.

2.3.2 Login Audit Trail

Each login and logout is carefully audited to check for attempts to guess valid user passwords. In addition, each user is informed of the date, time and terminal identification (if any) of the last login to detect past compromises of the user's access rights. Further, the user is told the number of times his password has been given incorrectly since its last correct use.

2.3.3 Software Maintenance Procedures

The maintenance of the Multics software is carried out online on a dial-up Multics facility. A systems programmer prepares and nominally debugs his software for installation. He then submits his software to a library installer who copies and recompiles the source in a protected directory. The library installer then checks out the new software prior to installing it in the system source and object libraries. Ring 0 software is stored on a system tape that is reloaded into the system each time it is brought up. However, new system tapes are generated from online copies of the ring 0 software. The system libraries are protected against modification by the standard ACL mechanism. In addition, the library installers periodically check the date/time last modified of all segments in the library in an attempt to detect unauthorized modifications.

3 VULNERABLITY ANALYSIS

3.1 Approach Plan

It was hypothesized that although the fundamental design characteristics of Multics were sound, the implementation was carried out on an ad hoc basis and had security weaknesses in each of the three areas of security controls described in Section 2 – hardware, software, and procedures.

The analysis was to be carried out on a very limited basis with less than one-half man month per month level of effort. Due to the manpower restrictions, a goal of one vulnerability per security control area was set. The procedure followed was to postulate a weakness in a general area, verify the weakness in the system, experiment with the weakness on the Rome Air Development Center (RADC) installation, and finally, using the resulting debugged penetration approach, exploit the weakness on the MIT installation.

An attempt was to be made to operate with the same type of ground rules under which a real agent would operate. That is, with each penetration, an attempt would be made to extract or modify sensitive system data without detection by the system maintenance or administrative personnel.

Several exploitations were successfully investigated. These included changing access fields in SDW's, changing protected identities in the PDS, inserting trap doors into the system libraries, and accessing the system password file.

3.2 Hardware Vulnerabilities

3.2.1 Random Failures

One area of significant concern in a system processing multi-level classified material is that of random hardware failures. As described in Section 2.1.1, the fundamental security of the system is dependent on the correct operation of the segmentation hardware. If this hardware is prone to error, potential security vulnerabilities become a significant problem.

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 a 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 received a timer interrupt and performed one test from the list described below. Assuming the test did not successfully violate security rules, the sub-

1100 operating hours		
	Test Name	Number Attempts
1.	Clear Associative Memory	3526
2.	Store Control Unit	3466
3.	Load Timer Unit	3444
4.	Load Descriptor Base Register	3422
5.	Store Descriptor Base Register	3403
6.	Connect I/O Channel	3378
7.	Delay Until Interrupt Signal	3359
8.	Read Memory Controller Mask Register	3344
9.	Set Memory Controller Mask Register	3328
10.	Set Memory Controller Interrupt Cells	3309
11.	Load Alarm Clock	3289
12.	Load Associative Memory	3259
13.	Store Associative Memory	3236
14.	Restore Control Unit	3219
15.	No Read Permission	3148
16.	No Write Permission	3131
17.	XED – No Read Permission	3113
18.	XED – No Write Permission	3098
19.	Tally Word Without Write Permission	3083
20.	Bounds Fault <64K	2398
21.	Bounds Fault >64K	2368
22.	Illegal Opcodes	2108

Table 1. Subverter Test Attempts

IEEE COMPUTER SOCIETY

verter would go to sleep for one minute before trying the next test. A listing of the subverter may be found in Appendix A.

The subverter was run for 1100 hours in a one year period on the MIT 645 system. The number of times each test was attempted is shown in Table 1. During the 1100 operating hours, no security sensitive hardware component failures were detected, indicating good reliability for the 645 security hardware. However, two interesting anomalies were discovered in the tests. First, one undocumented instruction (octal 471) was discovered on the 645. Experimentation indicated that the new instruction had no obvious impact on security, but merely seemed to store some internal register of no particular interest. The second anomaly was a design error resulting in an algorithmic failure of the hardware described in Section 3.2.2.

Tests 1-14 are tests of master mode instructions. Tests 15 and 16 attempt simple violation of read and write permission as set on segment ACL's. Tests 17 and 18 are identical to 15 and 16 except that the faulting instructions are reached from an Execute Double instruction rather than normal instruction flow. Test 19 attempts to increment a tally word that is in a segment without write permission. Tests 20 and 21 take out of bounds faults on segments of zero length, forcing the supervisor to grow new page tables for them. Test 22 attempts execution of all the instructions marked illegal on the 645.

3.2.2 Execute Instruction Access Check Bypass

While experimenting with the hardware subverter, a sequence of code[10] was observed which would cause the hardware of the 645 to bypass access checking. Specifically, the execute instruction in certain cases described below 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 (re) permission. (See Figure 4.) The execute instruction then referenced an object instruction in word zero of a second segment with at least R permission. The object instruction indirected through an ITS pointer in the first segment to access a word for reading or writing in a third segment. The third segment was required to be *active*; that is, to have an SDW pointing to a valid page table for the segment. If all these conditions were met *precisely*, the access control fields in the SDW of the third segment would be ignored and the object instruction permitted to complete without access checks.

The exact layout of instructions and indirect words was crucial. For example, if the object instruction used a base register rather than indirecting through the segment containing the execute instruction (i.e., staq apl0 rather than staq 6,*), then the access checks were done properly. Unfortunately, a complete schematic of the 645 was not available to determine the exact cause of the bypass. In informal communications with Honeywell, it was indicated that the error was introduced in a field modification to the 645 at MIT and was then made to all processors at all other sites.

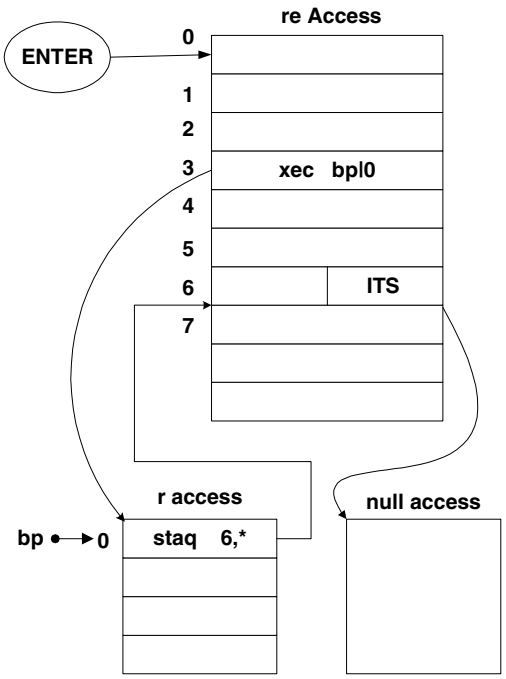

Figure 4. Execute Instruction Bypass

This hardware bug represents a violation of one of the most fundamental rules of the Multics design - the checking of *every* reference to a segment by the hardware. This bug was not caused by fundamental design problems. Rather, it was caused by carelessness by the hardware engineering personnel.

No attempt was made to make a complete search for additional hardware design bugs, as this would have required logic diagrams for the 645. It was sufficient for this effort to demonstrate one vulnerability in this area.

3.2.3 Preview of 6180 Hardware Vulnerabilities

While no detailed look has been taken at the issue of hardware vulnerabilities on the 6180, the very first login of an ESD analyst to the 6180 inadvertently discovered a hardware vulnerability that crashed the system. The vulnerability was found in the Tally Word Without Write Permission test of the subverter. In this test, when the 6180 processor encountered the tally word without write permission, it signalled a *trouble* fault rather than an *access violation* fault. The *trouble* fault is normally signalled only when a fault occurs during the signalling of a fault. Upon

10 The subverter was designed to test sequences of code in which single failures could lead to security problems. Some of these sequences exercised relatively complex and infrequently used instruction modifications, which experience had shown were prone to error.

IEEE COMPUTER SOCIETY

encountering a *trouble* fault, the software normally brings the system down.

It should be noted that the HIS 6180 contains very new and complex hardware that, as of this publication, has not been completely "shaken down". Thus, Honeywell still quite reasonably expects to find hardware problems. However, the inadequacy of "testing" for security vulnerabilities applies equally well to hardware as to software. Simply "shaking down" the hardware cannot find all the possible vulnerabilities.

3.3 Software Vulnerabilities

Although the approach plan for the vulnerability analysis only called for locating one example of each class of vulnerability, three software vulnerabilities were identified as shown below. Again, the search was neither exhaustive nor systematic.

3.3.1 Insufficient Argument Validation

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 a 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[11] to declare to the *gatekeeper* the following information.

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

This information is stored by convention in specified locations within the gate segment.[12] 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 condition *gate error* in the calling ring.

In February 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 follows: First 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. Next, force indirection through each argument pointer to obtain the segment number of the target argument. Then look up the segment in the calling ring's descriptor segment to check for read or write permission.

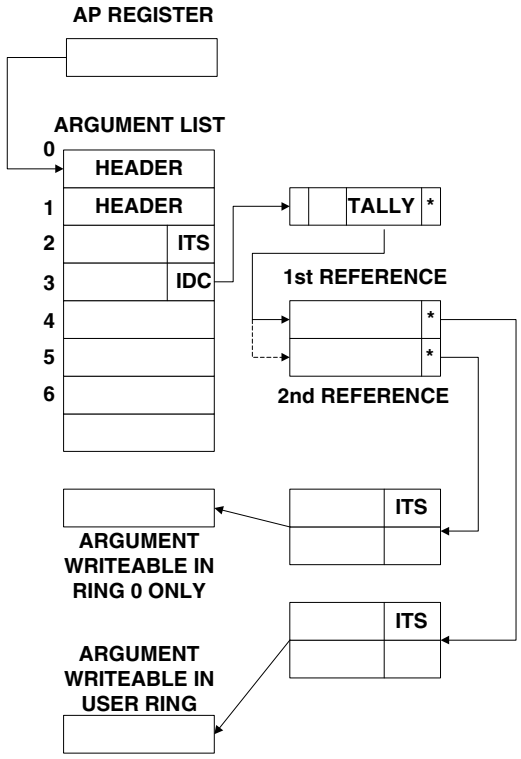

Figure 5. Insufficient Argument Validation

The vulnerability is as follows: (See figure 5.) An argument pointer supplied by the user is constructed to contain an IDC modifier (*i*ncrement address, *d*ecrement tally, and *c*ontinue) that causes the first reference though the indirect chain to address a valid argument. This 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 indirect word which in turn points to a different ITS pointer which points to an argument which is writable in ring 0 only. The second reference through this modified indirect chain is made by the ring 0 program, which proceeds to write data where it shouldn't.[13]

This vulnerability resulted from violation of a basic rule of the Multics design – that all arguments to a more privileged ring be validated. The problem was not in the fun-

11 A gate segment is a segment used to cross rings. It is identified by R2 and R3 of its ring brackets R1, R2, R3 being different. See Organick [22] for a detailed description of ring brackets.

12 For the convenience of authors of gates, a special "gate language" and "gate complier" are provided to generate properly formatted gates. Using this language, the author of the gate can declare the data type and access requirement of each argument.

13 Depending on the actual number of references made, the malicious user need only vary the number of indirect words pointing to legal and illegal arguments. We have assumed for simplicity here that the validator and the ring 0 program make only one reference each.

1063-9527/02 $17.00 © 2002 **IEEE**

IEEE COMPUTER SOCIETY

damental design – the concept of a software argument validator is sound given the lack of ring hardware. The problem was an ad hoc implementation of that argument validator which overlooked a class of argument pointers.

Independently, a change was made to the MIT system, which fixed this vulnerability in February 1973. The presence and exploitability of the vulnerability were verified on the RADC Multics, which had not been updated to the version running at MIT. The method of correction chosen by MIT was rather *brute force*. The argument validator was changed to require the modifier in the second word of each argument pointer always to be zero. This requirement solves the specific problem of the IDC modifier, but not the general problem of argument validation.

3.3.2 Master Mode Transfer

As described in Sections 2.1.2 and 2.2.4, the 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.[14] The original design of the Multics protection rings called for master mode code to be restricted to ring 0 by convention.[15] This convention caused the fault handling mechanism to be excessively expensive due to the necessity of switching from the user ring into ring 0 and out again using the full software ring crossing mechanism. It was therefore proposed and implemented that the *signaller*, the module responsible for processing faults to be signalled to the user,[16] be permitted to run in the user ring to speed up fault processing. The signaller is a master mode procedure, because it must execute the RCU (*R*estore *C*ontrol *U*nit) instruction to restart a process after a fault.

The decision to move the signaller to the user ring was not felt to be a security problem by the system designers, because master mode procedures could only be entered at word zero. The signaller would be assembled with the master mode pseudo-operation code at word zero to protect it from any malicious attempt by a user to execute an arbitrary sequence of instructions within the procedure. It was also proposed, although never implemented, that the code of master mode procedures in the user ring be specially audited. However as we shall see in Section 3.4.4, auditing does not guarantee victory in the *battle of wits* between the implementer and the penetrator. Auditing cannot be used to make up for fundamental security weaknesses.

14 The 645 also has an absolute mode in which all addresses are absolute core addresses rather than being translated by the segmentation hardware. This mode is used only to initialize the system.

15 This convention is enforced on the 6180. Privileged mode (the 6180 analogy to the 645 master mode) only has effect in ring 0. Outside ring 0, the hardware ignores the privileged mode bit.

16 The signaller processed such faults as zerodivide and access violation, which are signalled to the user. Page faults and segment faults, which the user never sees, are processed elsewhere in ring 0.

It was postulated in the ESD/MCI vulnerability analysis that master mode procedures in the user ring represent a fundamental violation of the Multics security concept. Violating this concept moves the security controls from the basic hardware/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. In fact, upon close examination of the signaller, this hypothesis was found to be true.

The master mode pseudo-operation code was designed only to protect master mode procedures from random calls with ring 0. It was not designed to withstand the attack of a malicious user, but only to operate in the relatively benign environment of ring 0.

```
        name            master_test
        mastermode
        entry           a
        entry           b
 a:     code
        …
 b:     code
        …
        end
```

Figure 6. Master Mode Source Code

The master mode program shown in Figure 6 assembles into the interpreted object code shown in Figure 7. The master mode procedure can only be entered at location zero.[17] By convention, the n entry points to the procedure

```
    cmpx0  2,du     "call in bounds?
    tnc    transfer_vector,0
                    "Yes, go to entry
    stb    sp|0     "Illegal call here
    sreg   sp|10    "save registers
    stcd   sp|24
    tra    lp|12,*  "lp|12 points
                    "to mxerror
a:  code
    …
b:  code
    …
transfer_vector:
    tra    a
    tra    b
    end
```

Figure 7. Master Mode Interpreted Object Code

are numbered from 0 to n-1. The number of the desired entry point must be in index register zero at the time of the call. The first two instructions in the master mode sequence check to ensure that index register zero is in bounds. If it is, the transfer on no carry (`tnc`) instruction indirects through the transfer vector to the proper entry. If

17 This restriction is enforced by hardware described in Section 2.1.2.

IEEE COMPUTER SOCIETY

index register zero is out of bounds, the processor registers are saved for debugging and control is transferred to *mxerror*, a routine to crash the system because of an unrecoverable error.

The transfer to *mxerror* is the most obvious vulnerability. By moving the signaller into the user ring, the designers allowed a user to arbitrarily crash the system by transferring to `signaller|0` with a bad value in index register zero. This vulnerability is not too serious, since it does not compromise information and could be repaired by changing *mxerror* to handle the error, rather than crashing the system.

However, there is a much more subtle and dangerous vulnerability here. The `tra lp|12,*` instruction that is used to call *mxerror* believes that the lp register points to the linkage section of the signaller, which it should if the call were legitimate. However, a malicious user may set the lp register to point wherever he wishes, *permitting him to transfer to an arbitrary location while the CPU is still in master mode.* The key is the transfer in master mode, because this permits a transfer to an arbitrary location within another master mode procedure without access checking and without the restriction of entering at word zero. Thus, the penetrator need only find a convenient store instruction to be able to write into his own descriptor segment, for example. Figure 8 shows the use of a `sta bp|0` instruction to change the contents of an SDW illegally.

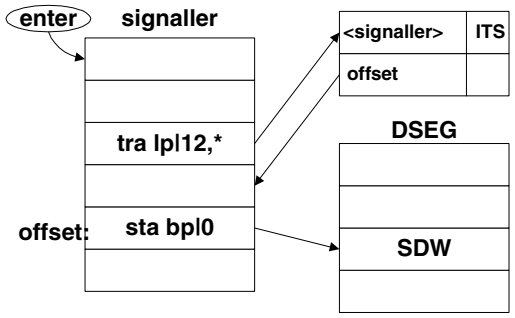

Figure 8. Store with Master Mode Transfer

There is one major difficulty in exploiting this vulnerability. The instruction to which control is transferred must be chosen with extreme care. The instructions immediately following the store must provide some orderly means of returning control to the malicious user without doing uncontrolled damaged to the system. If a crucial data base is garbled, the system will crash leaving a core dump which could incriminate the penetrator.

This vulnerability was identified by ESD/MCI in June 1972. An attempt to use the vulnerability led to a system crash for the following reason: Due to an obsolete listing of the signaller, the transfer was made to an ldbr (*L*oad *D*escriptor *B*ase *R*egister) instruction instead of the expected store instruction. The DBR was loaded with a garbled value, and the system promptly crashed. The system maintenance personnel, being unaware of the presence of an active penetration, attributed the crash to a disk read error.

The Master Mode Transfer vulnerability resulted from a violation of the fundamental rule that master mode code shall not be executed outside ring 0. The violation was not made maliciously by the system implementers. Rather it occurs because of the interaction of two seemingly independent events: the ability to transfer via the lp without the system being able to check the validity of the lp setting, and the ability for that transfer to be master mode code. The separation of these events made the recognition of the problem unlikely during implementation.

3.3.3 Unlocked Stack Base

The 645 CPU has eight 18-bit registers that are used for inter-segment references. Control bits are associated with each register to allow it to be paired with another register as a word number-segment number pair. In addition, each register has a lock bit, settable only in master mode, which protects its contents from modification. By convention, the eight registers are named and paired as shown in Table 2.

Num	Name	Use	Pairing
0	ap	Argument pointer	Paired with ab
1	ab	Argument base	Unpaired
2	bp	Unassigned	Paired with bb
3	bb	Unassigned	Unpaired
4	lp	Linkage pointer	Paired with lb
5	lb	Linkage base	Unpaired
6	sp	Stack pointer	Paired with sb
7	sb	Stack base	unpaired

Table 2. Base Register Pairing

During the early design of the Multics operating system, it was felt that the ring 0 code could be simplified if the stack base (sb) register were locked, that is, could only be modified in master mode. The sb contained the segment number of the user stack which was guaranteed to be writeable. If the sb were locked, then the ring 0 fault and interrupt handlers could have convenient areas in which to store stack frames. After Multics had been released to users at MIT, it was realized that locking the stack base unnecessarily constrained language designers. Some languages would be extremely difficult to implement without the capability of quickly and easily switching between stack segments. Therefore, the system was modified to no longer lock the stack base.

When the stack base was unlocked, it was realized that there was code scattered throughout ring 0 which assumed

IEEE COMPUTER SOCIETY

that the sb always pointed to the stack. Therefore, ring 0 was *audited* for all code which depended on the locked stack base. However, the audit was never completed and the few dependencies identified were in general not repaired until much later.

As part of the vulnerability analysis, it was hypothesized that such an audit for unlocked stack base problems was presumably incomplete. The ring 0 code is so large that a subtle dependency on the sb register could easily slip by an auditor's notice. This, in fact proved to be true as shown below:

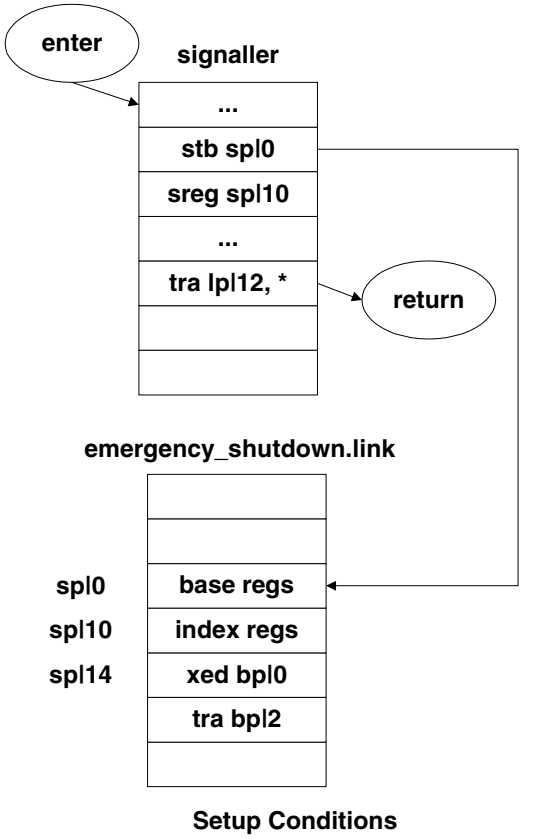

Figure 9. Unlocked Stack Base (Step 1)

Section 3.3.2 showed that the master mode pseudo-operation code believed the value in the lp register and transferred through it. Figure 7 shows that the master mode pseudo-operation code also depends on the sb pointing to a writeable stack segment. When an illegal master mode call is made, the registers are saved on the stack prior to calling *mxerror* to crash the system. This code was designed prior to the unlocking of the stack base and was not detected in the system audit. The malicious user need only set the sp-sb pair to point anywhere to perform an illegal store of the registers with master mode privileges.

The exploitation of the unlocked stack base vulnerability was a two step procedure. The master mode pseudo-operation code stored *all* the processor registers in an area over 20 words long. This area was far too large for use in a system penetration in which at most one or two words are modified to give the agent the privileges he requires. However, storing a large number of words could be very useful to install a *trap door* in the system -- that is a sequence of code which when properly invoked provides the penetrator with the needed tools to subvert the system. Such a *trap door* must be well hidden to avoid accidental discovery by the system maintenance personnel.

It was noted that the linkage segments of several of the ring 0 master mode procedures were preserved as separate segments rather than being combined in a single linkage segment. Further, these linkage segments were themselves master mode procedures. Thus, segments such as signaller, fim, and emergency_shutdown had corresponding master mode linkage segments signaller.link, fim.link, and emergency_shutdown.link. Linkage segments contain a great deal of information used only by the binder and therefore contain a great deal of extraneous information in ring 0. For this reason, a master mode linkage segment is an ideal place to conceal a *trap door*. There is a master mode procedure called emergency_shutdown that is used to place the system in a consistent state in the event of a crash. Since emergency_shutdown is used only at the time of a system crash, its linkage segment, emergency_shutdown.link, was chosen to be used for the *trap door*.

The first step of the exploitation of the unlocked stack base is shown in Figure 9.[18] The signaller is entered at location 0 with an invalid index register 0. The stack pointer is set to point to an area of extraneous storage in emergency_shutdown.link. The AQ register contains a two instruction *trap door* which when executed in master mode can load or store any 36-bit word in the system. The index registers could be used to hold a longer *trap door*; however, in this case the xed bp|0, tra bp|2 sequence is sufficient. The base registers, index registers, and AQ register are stored into emergency_shutdown.link, thus laying the *trap door*. Finally a transfer is made indirect through lp|12 which has been pre-set as a return pointer.[19]

Step two of the exploitation of the unlocked stack base is shown in Figure 10. The calling program sets the bp register to point to the desired instruction pair and transfers to word zero of the signaller with an invalid value in index

18 Listings of the code used to exploit this vulnerability are found in Appendix B.

19 This transfer uses the Master Mode Transfer vulnerability to return. This is done primarily for convenience. The fundamental vulnerability is the storing though the sp register. Without the Master Mode Transfer, exploitation of the Unlocked Stack Base would have been more difficult, although far from impossible.

IEEE COMPUTER SOCIETY

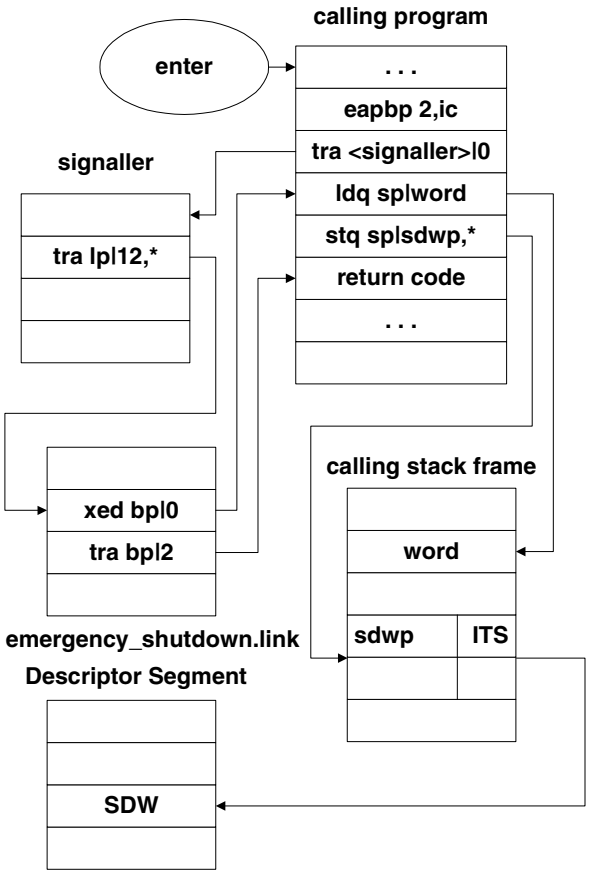

Figure 10. Unlocked Stack Base (Step 2)

register 0. The signaller saves its registers on the user's stack frame since the sp has not been changed. It then transfers indirect through lp|12 which has been set to point to the *trap door* in emergency_shutdown.link. The first instruction of the *trap door* is an execute double (XED) which permits the user (penetration agent) to specify any two arbitrary instructions to be executed *in master mode*. In this example, the instruction pair loads the Q register from a word in the stack frame[20] and then stores indirect through a pointer in the stack to an SDW in the descriptor segment. The second instruction in the *trap door* transfers back to the calling program, and the penetrator may go about his business.

The *trap door* inserted in emergency_shutdown.link remained in the system until the system was reinitialized.[21] At initialization time, a fresh copy of all ring zero segments is read in from the system tape erasing the *trap door*. Since system initializations occur at least once per day, the penetrator must execute step one before each of his working sessions. Step two is then executed each time he wishes to access or modify some word in the system.

The unlocked stack base vulnerability was identified in June 1972 with the Master Mode Transfer Vulnerability. It was developed and used at the RADC site in September 1972 without a single system crash. In October 1972, the code was transferred to the MIT site. Due to lack of good telecommunications between the two sites, the code was manually retyped into the MIT system. A typing mistake was made that caused the word to be stored into the SDW to always be zero (See Figure 10). When an attempt was made to set slave access-data in the SDW of the descriptor segment itself,[22] the SDW of the descriptor segment was set to zero causing the system to crash at the next LDBR instruction or segment initiation. The bug was recognized and corrected immediately, but later in the day, a second crash occurred when the SDW for the ring zero segment fim (the *f*ault *i*ntercept *m*odule) was patched to slave access-write permit-data rather than slave access-write permit-slave procedure. In more straightforward terms, the SDW was set to read-write rather than read-write-execute. Therefore, when the system next attempted to execute the fim it took a no-execute permission fault and tried to execute the fim, thus entering an infinite loop crashing the system.

3.3.4 Preview of 6180 Software Vulnerabilities

The 6180 hardware implementation of rings renders invalid the attacks described here on the 645. This is not to say, however, that the 6180 Multics is free of vulnerabilities. A cursory examination of the 6180 software has revealed the existence of several software vulnerabilities, any one of which can be used to access any information in the system. These vulnerabilities were identified with comparable levels of effort to those shown in Section 3.5.

3.3.4.1 No Call Limiter Vulnerability

The first vulnerability is the No Call Limiter vulnerability. This vulnerability was caused by the call limiter not being set on gate segments, allowing the user to transfer to any instruction within the gate rather than to just an entry transfer vector. This vulnerability gives the penetrator the same capabilities as the Master Mode Transfer vulnerability.

3.3.4.2 SLT-KST Dual SDW Vulnerability

The second vulnerability is the SLT-KST Dual SDW vulnerability. When a user process was created on the 645, separate descriptor segments were created for each ring with the ring 0 SDW's being copied from the segment loading table (SLT). The ring 0 descriptor segment was

20 It should be noted that only step one changed the value of the sp. In step two, it is very useful to leave the sp pointing to a valid stack frame.
21 See Section 3.4.5 for more lasting "trap doors".

22 The attempt here was to dump the contents of the descriptor segment on the terminal. The user does not normally have read permission to his own descriptor segment.

IEEE COMPUTER SOCIETY

essentially a copy of the SLT for ring 0 segments. The ring 4 descriptor segment zeroed out most SDW's for ring 0 segments. Non-ring 0 SDW's were added to both the ring 0 and ring 4 descriptor segments from the Known Segment Table (KST) during segment initiation. Upon conversion to the 6180, the separate descriptor segments for each ring were merged into one descriptor segment containing ring brackets in each SDW [5]. The ring 0 SDW's were still taken from the SLT and the non-ring 0 SDW's from the KST as on the 645.

The system contains several gates from ring 4 into ring 0 of varying levels of privilege. The least privileged gate is called hcs_ and may be used by all users in ring 4. The most privileged gate is called hphcs_ and may only be called by system administration personnel. The gate hphcs_ contains routines to shut the system down, access any segment in the system, and patch the ring 0 data bases. If a user attempts to call hphcs_ in the normal fashion, hphcs_ is entered into the KST, an SDW is assigned, and access rights are determined from the access control list stored in hphcs_'s parent directory. Since most users would not be on the access control list of hphcs_, access would be denied. Ring 0 gates, however, also have a second segment number assigned from the segment loading table (SLT). This duplication posed no problem on the 645, since SLT SDW's were valid only in the ring 0 descriptor segment. However on the 6180, the KST SDW for hphcs_ would be null access ring brackets 0,0,5, but the SLT SDW would read execute (re) access, ring brackets 0,0,5. Therefore, the penetrator need only transfer to the appropriate absolute segment number rather than using dynamic linking to gain access to any hphcs_ capability. This vulnerability was considerably easier to use than any of the others and was carried though identification, confirmation, and exploitation in less than 5 man-hours total (See Section 3.5).

3.3.4.3 Additional Vulnerabilities

The above mentioned 6180 vulnerabilities have been identified and repaired by Honeywell. The capabilities of the SLT-KST Dual SDW vulnerability were demonstrated to Honeywell on 14 September 1973 in the form of an illegal message to the operator's console at the 6180 site in the Honeywell plant in Phoenix, Arizona. Honeywell did not identify the cause of the vulnerability until March 1974 and installed a fix in Multics System 23.6. As of the time of this publication, additional vulnerabilities have been identified but at this time have not been developed into a demonstration.

3.4 Procedural Vulnerabilities

This section describes the exploitation by a remote user of several classes of procedural vulnerabilities. No attempt was made to penetrate physical security, as there were many admitted vulnerabilities in this area. In particular, the machine room was not secure and communications lines were not encrypted. Rather, this section looks at the areas of auditing, system configuration control,[23] passwords, and *privileged* users.

3.4.1 Dump and Patch Utilities

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. Clearly these commands are very dangerous, since they can bypass all security controls to access otherwise protected information, and if misused, can cause the system to crash by garbling critical data critical data bases. To protect the system, these commands are implemented by special privileged gates into ring zero. The access control lists on these gates restrict their use to system maintenance personnel by name as authenticated by the login procedure. Thus an ordinary user nominally cannot access these utilities. To further protect the system, the patch utility records on the system operator's console every patch that is made. Thus, if an unexpected or unauthorized patch is made, the system operator can take immediate action by shutting the system down if necessary.

Clearly dump and patch utilities would be of great use to a system penetrator, since they can be used to facilitate his job. Procedural controls on the system dump and patch routines prevent the penetrator from using them by the ACL restrictions and the audit trail. However by using the software vulnerabilities described in section 3.3, these procedural controls may be bypassed and the penetration agent can implement his own dump and patch utilities as described below.

Dump and patch utilities were implemented on Multics using the Unlocked Stack Base and Insufficient Argument Validation vulnerabilities. These two vulnerabilities demonstrated two basically different strategies for accessing protected segments. These two strategies developed from the fact that the Unlocked Stack Base vulnerability operates in ring 4 master mode while the Insufficient Argument Validation vulnerability operates in ring 0 slave mode. In addition, there was a requirement that a minimal amount of time be spent with the processor in an anomalous state - ring 4 master mode or ring 0 illegal code. When the processor is in an anomalous state, unexpected interrupts or events could cause the penetrator to be exposed in a system crash.

3.4.1.1 Use of Insufficient Argument Validation

As was mentioned above, the HIS 645 implementation of Multics simulates protection rings by providing one descriptor segment for each ring. Patch and dump utilities

23 System configuration control is a term derived from Air Force procurement procedures and refers to the control and management of the hardware and software being used in a system with particular attention to the software update tasks. It is not to be confused with the Multics dynamic reconfiguration capability, which permits the system to add and delete processors and memories while the system is running.

IEEE COMPUTER SOCIETY

can be implemented using the Insufficient Argument Validation vulnerability by realizing that the ring zero descriptor segment will have entries for segments which are not accessible from ring 4. Conceptually, one could copy an SDW for some segment from the ring 0 descriptor segment to the ring 4 descriptor segment and be guaranteed at least as much access as available in ring 0. Since the segment number of a segment is the same in all rings, this approach is very easy to implement.

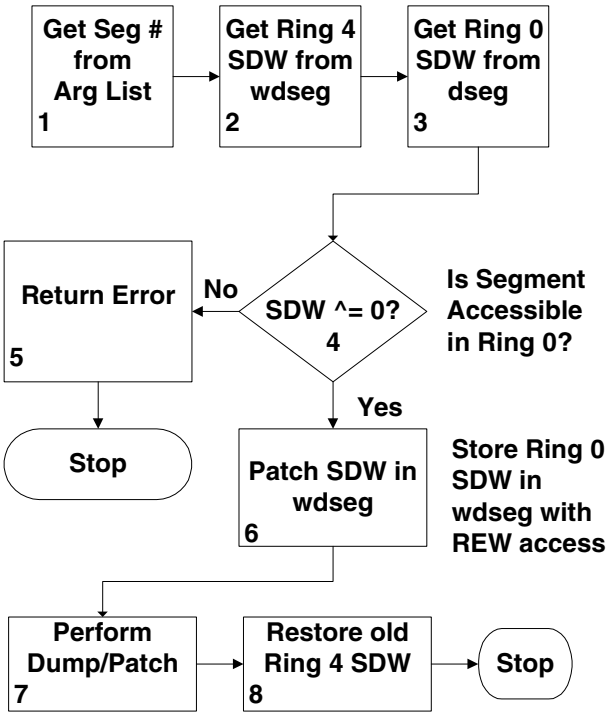

Figure 11. Dump/Patch Utility Using Insufficient Argument Validation

The exact algorithm is shown in flow chart form in Figure 11. In block 2 of the flow chart, the ring 4 SDW is read from the ring 4 descriptor segment (wdseg) using the Insufficient Argument Validation vulnerability. Next the ring 0 SDW is read from the ring 0 descriptor segment (dseg). The ring 0 SDW must now be checked for validity, since the segment may not be accessible even in ring 0.[24] An invalid SDW is represented by all 36 bits being zero. One danger present here is that if the segment in question is deactivated,[25] the SDW being checked may be invalidated while it is being manipulated. This event could conceivably have disastrous results, but as we shall see in Section 3.4.2, the patch routine need only be used on segments which are never deactivated. The dump routine can do no harm if it accidentally uses an invalid SDW, as it always only *reads* using the SDW, conceivably reading garbage but nothing else. Further, deactivation of the segment is highly unlikely since the segment is in *use* by the dump/patch routine.

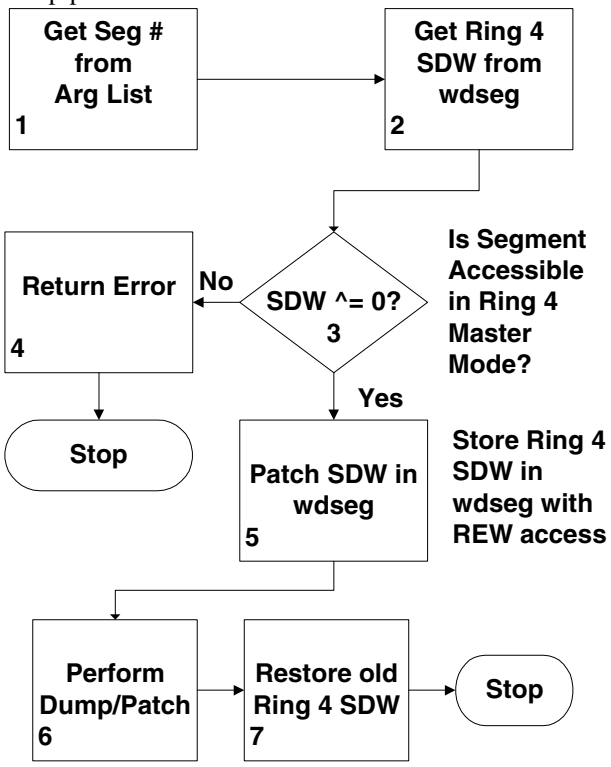

Figure 12. Dump/Patch Utility Using Unlocked Stack Base

If the ring 0 SDW is invalid, an error code is returned in block 5 of the flow chart and the routine terminates. Otherwise, the ring 0 SDW is stored into the ring 4 descriptor segment (wdseg) with read-execute-write access by turning on the SDW bits for slave access, write permission, slave procedure (See Figure 2). Now the dump or patch can be performed without using the vulnerability to load or store each 36 bit word being moved. Finally in block 8, the ring 4 SDW is restored to its original value, so that a later unrelated system crash could not reveal the modified SDW in a dump. It should be noted that while blocks 2, 3, 6, and 8 all use the vulnerability, the bulk of the time is spent in block 7 actually performing the dump or patch in perfectly normal ring 4 slave mode code.

3.4.1.2 Use of Unlocked Stack Base

The Unlocked Stack Base vulnerability operates in a very different environment from the Insufficient Argument Validation vulnerability. Rather than running in ring 0, the Unlocked Stack Base vulnerability runs in ring 4 in master mode. In the ring 0 descriptor segment, the segment dseg is the ring 0 segment, and wdseg is the ring 4 descriptor

24 As an additional precaution, ring 0 slave mode programs run under the same access rules as all other programs. A valid SDW entry is made for a segment in any ring only if the user is on the ACL for the segment. We shall see in Section 3.4.2 how to get around this "security feature".

25 A segment is deactivated when its page table is removed from core. Segment deactivation is performed on a least recently used basis, since not all page tables may be kept in a core at one time.

IEEE COMPUTER SOCIETY

segment.[26] However, in the ring 4 descriptor segment, the segment dseg is the ring 4 descriptor segment and wdseg has a zeroed SDW. Therefore, a slightly different strategy must be used to implement dump and patch utilities as shown in the flow chart in Figure 12.[27] The primary difference here is in blocks 3 and 5 of Figure 12 in which the ring 4 SDW for the segment is used rather than the ring 0 SDW. Thus the number of segments, which can be dumped or patched is reduced from those accessible in ring 0 to those accessible in ring 4 master mode. We shall see in Section 3.4.2 that this reduction is not crucial, since ring 4 master mode has sufficient access to provide *interesting* segments to dump or patch.

3.4.1.3 Forging Generation of New SDW's

Two strategies for implementation of dump and patch utilities were shown above. In addition, a third strategy exists which was rejected due to its inherent dangers. In this third strategy, the penetrator selects an unused segment number and constructs an SDW occupying that segment number in the ring 4 descriptor segment using any of the vulnerabilities. This totally new SDW could then be used to access some part of the Multics hierarchy. However, two major problems are associated with this strategy, which caused its rejection. First the absolute core address of the page table of the segment must be stored in the SDW address field. There is no easy way for a penetrator to obtain the absolute address of the page table for a segment not already in his descriptor segment short of duplicating the entire segment fault mechanism which runs to many hundreds or thousands of lines of code. Second, if the processor took a segment or page fault on this new SDW, the ring 0 software would malfunction, because the segment would not be recorded in the Known Segment Table (KST). This malfunction could easily lead to a system crash and the disclosure of the penetrator's activities. Therefore, the strategy of generating new SDW's was rejected.

3.4.2 Forging the Non-Forgeable User Identification

In section 2.2.3, the need for the protected, non-forgeable identification of each user was identified. This non-forgeable ID must be compared with access control list entries to determine whether a user may access some segment. This identification is established when the user logs into Multics and is authenticated by the user password.[28] If this user identification can be forged in any way, then the entire login audit mechanism can be rendered worthless.

The user identification in Multics is stored in a per-process segment called the process data segment (PDS). The PDS resides in ring 0 and contains many constants used in ring 0 and the ring 0 procedure stack. The user identification is stored in the PDS as a character string representing the user's name and a character string representing the user's project. The PDS must be accessible to any ring 0 procedure within the user's process and must be accessible to ring 4 master mode procedures (such as the signaller). Therefore, as shown in the Sections 3.4.1.1 and 3.4.1.2, the dump and patch utilities can dump and patch portions of the PDS, thus *forging the non-forgeable user identification*. Appendix E shows the actual user command needed to forge the user identification.

This capability provides the penetrator with an *ultimate weapon*. The agent can now undetectably masquerade as any user of the system including the system administrator or security officer, immediately assuming that user's access privileges. The agent has bypassed and rendered ineffective the entire login authentication mechanism with all its attendant auditing machinery. The user whom the agent is impersonating can login and operate without interference. Even the *who table* that lists all the users currently logged into the system records the agent with his correct identification rather than the forgery. Thus to access *any* segment in the system, the agent need only determine who has access and change his user identification as easily as a legitimate user can change his working directory.

It was not obvious at the time of the analysis that changing the user identification would work. Several potential problems were foreseen that could lead to system crashes or could reveal the penetrator's presence. However, none of these proved to be a serious barrier to masquerading.

First, a user process occasionally sends a message to the operator's console from ring 0 to report some type of unusual fault such as a disk parity error. These messages are prefaced by the user's name and project taken from the PDS. It was feared that a random parity error could "blow the cover" of the penetrator by printing his modified identification on the operator's console.[29] However, the PDS in fact contains two copies of the user identification – one formatted for printing and one formatted for comparison with access control list entries. Ring 0 software keeps these strictly separated, so the penetrator need only change the access control identification.

Second, when the penetrator changes his user identification, he may lose access to his own programs, data and directories. The solution here is to assure that the access control lists of the needed segments and directories grant appropriate access to the user as whom the penetrator is masquerading.

26 Actually wdseg is the descriptor segment for whichever ring (1-7) was active at the time of the entry to ring 0. No conflict occurs since wdseg is always the descriptor segment for the ring on behalf of which ring 0 is operating

27 This strategy is also used with the Execute Instruction Access Check Bypass vulnerability, which runs in ring 4.

28 Clearly more sophisticated authentication schemes than a single user chosen password could be used on Multics (see Richardson [24]). However, such schemes are outside the scope of this paper.

29 This danger exists only if the operator or system security officer is carefully correlating parity error messages with the names of currently logged in users.

IEEE COMPUTER SOCIETY

Finally, one finds that although the penetrator can set the access control lists of his ring 4 segments appropriately, he cannot in any easy way modify the access control lists of certain per process supervisor segments including the process data segment (PDS), the process initialization table (PIT), the known segment table (KST), and the stack and combined linkage segments for ring 1, 2, and 3. The stack and combined linkage segments for ring 1, 2, and 3 can be avoided by not calling any ring 1, 2, or 3 programs while masquerading. However the PDS, PIT, and KST are all ring 0 data bases that must be accessible at all times with read and write permission. This requirement could pose the penetrator a very serious problem; but, because of the very fact that these segments must *always* be accessible in ring 0, the system has already solved this problem. While the PIT, PDS and KST are paged segments,[30] they are all used during segment fault handling. In order to avoid recursive segment faults, the PIT, PDS, and KST are never deactivated.[31] Deactivation, as mentioned above, is the process by which a segment's page table is removed from core and a segment fault is placed in its SDW. The access control bits are set in an SDW *only* at segment fault time.[32] Since the system never deactivates the PIT, PDS, and KST, under normal conditions, the SDW's are not modified for the life of the process. Since the process of changing user identification does not change the ring 0 SDW's of the PIT, PDS, and KST either, the penetrator retains access to these critical segments without any special action whatsoever.

3.4.3 Accessing the Password File

One of the classic penetrations of an operating system has been unauthorized access to the password file. This type of attack on a system has become so embedded in the folklore of compute security that it even appears in the definition of a security *breach* in DOD 5200.28-M [7]. In fact, however, accessing the password file internal to the system proves to be of minimal value to a penetrator as shown below. For completeness, the Multics password file was accessed as part of this analysis.

3.4.3.1 Minimal Value of the Password File

It is asserted that accessing the system password file is of minimal value to a penetrator for several reasons. First, the password file is generally the most highly protected file in a computer system. If the penetrator has succeeded in breaking down the internal controls to access the password file, he can almost undoubtedly access *every other file in the system*. Why bother with the password file?

Second, the password file is often kept enciphered. A great deal of effort may be required to invert such a cipher, if indeed the cipher is invertible at all.

Finally, the login path to a system is generally the most carefully audited to attempt to catch unauthorized password use. The penetrator greatly risks detection if he uses an unauthorized password. It should be noted that an unauthorized password obtained outside the system may be very useful to a penetrator, if he does not already have access to the system. However, that is an issue of physical security, which is outside the scope of this paper.

3.4.3.2 The Multics Password File

The Multics password file is stored in a segment called the person name table (PNT). The PNT contains an entry for each user on the system including that user's password and various pieces of auditing information. Passwords are chosen by the user and may be changed at any time.[33] Passwords are scrambled by an allegedly non-invertible enciphering routine for protection in case PNT appears in a system dump. Only enciphered passwords are stored in the system. The password check at login time is accomplished by the equivalent of the following PL/I code:

```
if scramble_(typed_password) = pnt.user.password
      then call ok_to_login;
      else call reject_login;
```

For the rest of this section, it will be assumed that the enciphering routine is non-invertible. In a separate volume [15], Downey demonstrates the invertibility of the Multics password scrambler used at the time of the vulnerability analysis.[34]

The PNT is a ring 4 segment with the following access control list:

```
rw      *.SysAdmin.*
null    *.*.*
```

Thus by modifying one's user identification to the SysAdmin project as in Section 3.4.2, one can immediately gain unrestricted access to the PNT. Since the passwords are enciphered, they cannot be read out of the PNT directly. However, the penetrator can extract a copy of the PNT for cryptanalysis. The penetrator can also change a user's password to the enciphered version of a known password. Of course, this action would lead to almost immediate discovery, since the user would no longer be able to login.

30 In fact the first page of the PDS is wired down so that it may be used by page control. The rest of the PDS, however, is not wired.

31 In Multics jargon, their entry hold switches are set.

32 In fact, a segment fault is also set in a SDW when the access control list of the corresponding segment is changed. This is done to ensure that access changes are reflected immediately, and is effected by setting faults in all descriptor segments that have active SDW's for the segment. This additional case is not a problem, because the access control lists of the PIT, PDS, and KST are never changed.

33 There is a major problem that user chosen passwords are often easy to guess. That problem, however, will not be addressed here. Multics provides a random password generator, but its use is not mandatory.

34 ESD/MCI has provided a 'better" password scrambler that is now used in Multics, since enciphering the password file is useful in case it should appear in a system dump.

IEEE COMPUTER SOCIETY

3.4.4 Modifying Audit Trails

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. However, we have seen that a penetrator using vulnerabilities in the operating system code can access information and bypass many such audits. Sometimes it is not convenient for the penetrator to bypass an audit. If the audit trail is kept online, it may be much easier to allow the audit to take place and then go back and modify the evidence of wrong doing. One simple example of modification of audit trails was selected for this vulnerability demonstration.

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 very low level in the system, and it is almost impossible for a penetrator to bypass this mechanism.[35] An obvious approach would be to attempt to patch the DTU and DTM that are stored in the parent directory of the segment in question. However, directories are implemented as rather complex hash tables and are therefore very difficult to patch.

Once again, however, a solution exists within 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 segment's DTU and DTM to the values at the time the segment was backed up. The routine is supposed to be callable only from a highly privileged gate into ring 0 that is restricted to system maintenance personnel. However, since a penetrator can change his user identification, this restriction proves to be no barrier. To access a segment without updating DTU or DTM:

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. Reset DTU and DTM to old values.
6. Change user ID back to original.

In fact due to yet another system bug, the procedure is even easier. The module set_dates is callable, not only from the highly privileged gate into ring 0, but also from the normal user gate into ring 0.[36] Therefore, step 4 in the above algorithm can be omitted if desired. A listing of the utility that changes DTU and DTM may be found in Appendix F.

It should be noted that one complication exists in step 5 – resetting DTU and DTM. The system does not update the dates in the directory entry immediately, but primarily at segment deactivation time.[37] Therefore, step 5 must be delayed until the segment has been deactivated – a delay of up to several minutes. Otherwise the penetrator could reset the dates, only to have them updated again a moment later.

3.4.5 Trap Door Insertion

Up to this point, we have seen how a penetrator can exploit existing weaknesses in the security controls of an operating system to gain unauthorized access to protected information. However, when the penetrator exploits existing weaknesses, he runs the constant risk that the system maintenance personnel will find and correct the weakness he happens to be using. The penetrator would then have to begin again looking for weaknesses. To avoid such a problem and to perpetuate access into the system, the penetrator can install *trap doors* in the system which permit him access, but are virtually undetectable.

3.4.5.1 Classes of Trap Doors

Trap doors come in many forms and can be inserted in many places throughout the operational life of a system from the time of design up to the time the system is replaced. Trap doors may be inserted at the facility at which the system is produced. Clearly if one of the system programmers is an agent, he can insert a trap door in the code he writes. However, if the production site is a (perhaps online) facility to which the penetrator can gain access, the penetrator can exploit existing vulnerabilities to insert trap doors into system software while the programmer is still working on it or while it is in quality assurance.

As a practical example, it should be noted that the software for WWMCCS is currently developed using uncleared personnel on a relatively open time sharing system at Honeywell's plant in Phoenix, Arizona. The software is monitored and distributed from an open time sharing system at the Joint Technical Support Agency (JTSA) at Reston, Virginia. Both of these sites are potentially vulnerable to penetration and trap door insertion.

Trap doors can be inserted during the distribution phase. If updates are sent via insecure communications – either US Mail or insecure telecommunication, the penetrator can intercept the update and subtly modify it. The penetrator could also generate his own updates and distribute them using forged stationery.

Finally, trap doors can be inserted during the installation and operation of the system at the user's site. Here again, the penetrator uses existing vulnerabilities to gain access to stored copies of the system and make subtle modifications.

Clearly when a trap door is inserted, it must be well hidden to avoid detection by system maintenance personnel. Trap doors can best be hidden in changes to the binary code of a compiled routine. Such a change is completely invisible on system listings and can be detected only by comparing bit by bit the object code and the compiler listing. However, object code trap doors are vulnerable to recompilations of the module in question.

35 Section 3.4.5 shows motivation to bypass DTU and DTM.

36 The user gate into ring 0 contains set_dates, so that users may perform reloads from private backup tapes.

37 Dates may be updated at other times as well.

IEEE COMPUTER SOCIETY

Therefore the system maintenance personnel could regularly recompile all modules of the system to eliminate object code trap doors. However, this precaution could play directly into the hands of the penetrator who has also made changes in the source code of the system. Source code changes are more visible than object code changes, since they appear in system listings. However, subtle changes can be made in relatively complex algorithms that will escape all but the closest scrutiny. Of course, the penetrator must be sure to change *all* extant copies of a module to avoid discovery by a simple comparison program.

Two classes of trap doors which are themselves source or object trap doors are particularly insidious and merit discussion here. These are the teletype key string trigger trap door and the compiler trap door.

It has often been hypothesized that a carefully written closed subsystem such as a query system or limited data management system without programming capabilities may be made invulnerable to security penetration. The teletype key string trigger is just one example of a trap door that provides the penetrator with a vulnerability in even the most limited subsystem. To create such a trap door, the agent modifies the supervisor teletype modules at the development site such that if the user types normally, no anomaly occurs, but if the user types a special key string, a dump/patch utility is triggered into operation to allow the penetrator unlimited access. The key string would of course have to be some very unlikely combination to avoid accidental discovery. The teletype key string trap door is somewhat more complex than the trap door described below in Section 3.4.5.2. However, it is quite straightforward to develop and insert with relatively nominal effort.

It was noted above that while object code trap doors are invisible, they are vulnerable to recompilations. The compiler (or assembler) trap door is inserted to permit object code trap doors to survive even a complete recompilation of the entire system In Multics, most of the ring 0 supervisor is written in PL/I. A penetrator could insert a trap door in the PL/I compiler to note when it is compiling a ring 0 module. Then the compiler would insert an object code trap door in the ring 0 module without listing the code in the listing. Since the PL/I compiler is itself written in PL/I, the trap door can maintain itself, *even when the compiler is recompiled*.[38] Compiler trap doors are significantly more complex than the other trap doors described here, because they require a detailed knowledge of the compiler design. However, they are quite practical to implement at a cost of perhaps five times the level shown in Section 3.5. It should be noted that even costs several hundred times larger than those shown here would be considered nominal to a foreign agent.

38 This type of trap door does not require a higher level language. Entirely analogous trap doors could be placed in an assembler.

There is also a variant on the compiler trap door called the initialization trap door. Here, the system initialization code is modified by the penetrator to insert other trap doors as the system is brought up. Such trap doors can be relatively invulnerable to detection and recompilation, because system initialization is usually a very complex and poorly understood procedure.

3.4.5.2 Example of Trap Door in Multics

```
check$device_name: proc (a, b, code);
dcl   1 code parameter,
        2 err_code fixed bin (35),
        2 key bit (72) aligned,
        2 ptr1 pointer aligned,
        2 ptr2 pointer aligned;
dcl overlay fixed bin (35) based;

/*   Start of regular code  */
      . . .;
/* Here check$device_name would
   normally return  */
if key = bit_string_constant_password
   then ptr2->overlay = ptr1->overlay;

return;
end check$device_name;
```

Figure 13. Trapdoor in check$device_name

To demonstrate the feasibility of inserting and distributing trap doors, the following object code change was made to check$device_name to provide a generalized dump or patch facility. Figure 13 shows a PL/I program that performs essentially the same function as the trap door actually inserted in the object code. The listing of the actual trap door and its calling program is in Appendix C. Check$device_name is normally called with three arguments, the third being a fixed binary error code. The trap door is inserted at the point at which check$device_name would have returned. The trap door looks at the 72 bits immediately following the error code that was passed by the caller. If those 72 bits match a predefined 72 bit password, then the fixed binary word to which ptr1 points is copied into the fixed binary word to which ptr2 points. Since check$device_name is a ring 0 procedure, this copy is carried out using the ring 0 descriptor segment and allows the caller to read or write any word in ring 0. Dump and patch utilities can use this trap door exactly like the Insufficient Argument Validation vulnerability. The 72 bit key is used to ensure that the vulnerability is not invoked by accident by some unsuspecting user.

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

1. Change user identification to project SysLib.
2. Make patch in object archive copy of check$device_name in >ldd>hard>object.
3. Reset DTM on object archive.

IEEE COMPUTER SOCIETY

4. Mark patch in bound archive copy of check$device_name in >ldd>hard>bound_components.
5. Reset DTM on bound archive.
6. Reset user identification.

This procedure ensured that the object patch was in all library copies of the segment. The DTM was reset as in Section 3.4.4, because the dates on library segments are checked regularly for unauthorized modification. These operations did not immediately install the trap door. Actual installation occurred at the time of the next system tape generation.

A trap door of this type was first placed in the Multics system at MIT in the procedure del_dir_tree. However, it was noted that del_dir_tree was going to be modified and recompiled in the installation of Multics system 18.0. Therefore, the trap door described above was inserted in check$device_name just before the installation of 18.0 to avoid the recompilation problem. Honeywell was briefed in the spring of 1973 on the results of this vulnerability analysis. At that time, Honeywell recompiled check$device_name, so that the trap door would not be distributed to other sites.

tem on which to debut penetrations. In this example, the RADC system was used to test penetrations prior to their use at MIT, since a system crash at MIT would reveal the intentions of the penetrations.[39]

Costs are broken down into identification, confirmation, and exploitations. Identification is that part of the effort needed to identify a particular vulnerability. It generally involves examination of system listings, although it sometimes involves computer work. Confirmation is that effort needed to confirm the existence of a vulnerability by using it in some manner, however crude, to access information without authorization. Exploitation is that effort needed to develop and debug command procedures to make use of the vulnerabilities convenient. Wherever possible, these command procedures follow standard Multics command conventions.

All figures in the table are conservative estimates as actual accounting information was not kept during the vulnerability analysis. However, costs did not exceed the figures given and in all probability were somewhat lower.

The costs of implementing the subverter and inverting the password scrambler are not included, because those tasks were not directly related to penetrating the system.

	Identification		Confirmation		Exploitation		Total	
Task	Manhrs	CPU $	Manhrs	CPU $	Manhrs	CPU $	Manhrs	CPU $
Execute Instruction Access Check Bypass	60	$150	5	$30	8	$100	73	$280
Insufficient Argument Validation	1	$0	5	$30	24	$300	30	$330
Master Mode Transfer	0.5	$0	2	$20	--	---	2.5	$20
Unlocked Stack Base	0.5	$0	8	$50	80	$500	88.5	$550
Forging User ID	5	$0	5	$30	5	$90	15	$120
check$device_name Trap door	5	$0	8	$50	5	$30	18	$80
Access Password File (Does not include deciphering.)	1	$0	5	$30	24	$150	30	$180
Total	73	$150	38	$240	146	$1170	257	$1560

Table 3. Cost Estimates

3.4.6 Preview of 6180 Procedural Vulnerabilities

To actually demonstrate the feasibility of trap door distribution, a change which could have included a trap door was inserted in the Multics software that was transferred from the 645 to the 6180 at MIT and from there to all 6180 installations in the field.

3.5 Manpower and Computer Costs

Table 3 outlines the approximate costs in man-hours and computer charges for each vulnerability analysis task. The skill level required to perform the penetrations was that of a recent computer science graduate of any major university with a moderate knowledge of the Multics design documented in the **Multics Programmers' Manual** [4] and Organick [22], plus nine months experience as a Multics programmer. In addition, the penetrator was aided by access to the system listings (which are in the public domain) and access to an operational Multics system on which to debut penetrations. In this example, the RADC system was used to test penetrations prior to their use at MIT, since a system crash at MIT would reveal the intentions of the penetrations.[39]

Costs are broken down into identification, confirmation, and exploitations. Identification is that part of the effort needed to identify a particular vulnerability. It generally involves examination of system listings, although it sometimes involves computer work. Confirmation is that effort needed to confirm the existence of a vulnerability by using it in some manner, however crude, to access information without authorization. Exploitation is that effort needed to develop and debug command procedures to make use of the vulnerabilities convenient. Wherever possible, these command procedures follow standard Multics command conventions.

All figures in the table are conservative estimates as actual accounting information was not kept during the vulnerability analysis. However, costs did not exceed the figures given and in all probability were somewhat lower.

The costs of implementing the subverter and inverting the password scrambler are not included, because those tasks were not directly related to penetrating the system. (See Downey[15].) The Master Mode Transfer vulnerability has no exploitation cost shown, because that vulnerability was not carried beyond confirmation.

4 CONCLUSION

The initial implementation of Multics is an instance of an uncertified system. For any uncertified system:

a. The system cannot be depended upon to protect against deliberate attack
b. System *fixes* or restrictions (e.g., query only systems) cannot provide any significant improvement in protection. Trap door insertion and distribution has been demonstrated with minimal effort and fewer tools (no phone taps) than any industrious foreign agent would have.

39 It should be noted that while the MIT system was crashed twice due to typographical errors during the penetration, the RADC system was never crashed.

IEEE COMPUTER SOCIETY

However, Multics is significantly better than other conventional systems due to the structuring of the supervisor and the use of segmentation and ring hardware. Thus, unlike other systems, Multics can form a base for the development of a truly secure system.

4.1 Multics is not Now Secure

The primary conclusion one can reach from this vulnerability analysis is that Multics is not currently a secure system. A relatively low level of effort gave examples of vulnerabilities in hardware security, software security, and procedural security. While all the reported vulnerabilities were found in the HIS 645 system and happen to be fixed by the nature of the changes in the HIS 6180 hardware, other vulnerabilities exist in the HIS 6180.[40] No attempt was made to find more than one vulnerability in each area of security. Without a doubt, vulnerabilities exist in the HIS 645 Multics that have not been identified. Some major areas not even examined are I/O, process management, and administrative interfaces. Further, an initial cursory examination of the HIS 6180 Multics easily turned up vulnerabilities.

We have seen the impact of implementation errors or omissions in the hardware vulnerability. In the software vulnerabilities, we have seen the major security impact of apparently unimportant ad hoc designs. We have seen that the development site and distribution paths are particularly attractive for penetration. Finally, we have seen that the procedural controls over such areas as passwords and auditing are no more than "security blankets" as long as the fundamental hardware and software controls do not work.

4.2 Multics as a Base for a Secure System

While we have seen that Multics is not now a secure system, it is in some sense significantly "more secure" than other commercial systems and forms a base from which a secure system can be developed. (See Lipner [21].) The requirements of security formed part of the basic guiding principles during the design and implementation of Multics. Unlike systems such as OS/360 or GCOS in which security functions are scattered throughout the entire supervisor, Multics is well structured to support the identification of the security and non-security related functions. Further Multics possesses the segmentation and ring hardware which have been identified [29] as crucial to the implementation of a reference monitor.

4.2.1 A System for a Benign Environment

We have concluded that AFDSC cannot run an open multi-level secure system on Multics at this time. As we have seen above, a malicious user can penetrate the system at will with relatively minimal effort. However, Multics does provide AFDSC with a basis for a *benign* multi-level system in which all users are determined to be trustworthy to some degree. For example, with certain enhancements, Multics could serve AFDSC in a two-level security mode with both Secret and Top Secret cleared users simultaneously accessing the system. Such a system, of course, would depend on the administrative determination that since *all* users are cleared at least to Secret, there would be no malicious users attempting to penetrate the security controls.

A number of enhancements are required to bring Multics up to a two-level capability. First and most important, all segments, directories, and processes in the system should be labeled with classification levels and categories. This labeling permits the classification check to be combined with the ACL check and to be represented in the descriptor segment. Second, an earnest review of the Multics operating system is needed to identify vulnerabilities. Such a review is meaningful in Multics, because of its well structured operating system design. A similar review would be a literally endless task in a system such as OS/360 or GCOS. A review of Multics should include an identification of security sensitive modules, an examination of all gates and arguments into ring 0, and a check of all intersegment arguments in ring 0. Two additional enhancements would be useful but not essential. These are some sort of *high water mark* system as in ADEPT-50 (see Weissman [31]) and some sort of protection from user written applications programs that may contain *Trojan Horses*.

4.2.2 Long Term Open Secure System

In the long term, it is felt that Multics can be developed into an open secure multi-level system by restructuring the operating system to include a security kernel. Such restructuring is essential since malicious users cannot be ruled out in an open system. The procedures for designing and implementing such a kernel are detailed elsewhere. [10, 12, 13, 20, 23, 26, 27, 30] To briefly summarize, the access controls of the kernel must always be invoked (segmentation hardware); must be tamperproof (ring hardware); and must be small enough and simple enough to be certified correct (a small ring 0). Certifiability is the critical requirement in the development of a multi-level secure system. ESD/MCI is currently proceeding with a development plan to develop such a certifiably secure version of Multics[1].

APPENDIX A Subverter Listing[41]

This appendix contains listings of the three program modules which make up the hardware subverter described

40 In all fairness, the HIS 6180 does proved significant improvements by the addition of ring hardware. However, ring hardware by itself does not make the system secure. Only certification as a well-defined closed process can do that.

41 The actual listing files from this and all subsequent appendices have been omitted, but can be found in [19].

IEEE COMPUTER SOCIETY

in Section 3.2.1. The three procedure segments which follow are called subverter, coded in PL/I; access_violations_, coded in PL/I; and subv, coded in assembler. Subverter is the driving routine which sets up timers, manages free storage, and calls individual tests. Access_violations_ contains several entry points to implement specific tests. Subv contains entry points to implement those tests which must be done in assembler.

The internal procedure check_zero within subverter is used to watch word zero of the procedure segment for unexpected modification. This procedure was used in part to detect the Execute Instruction Access Check Bypass vulnerability.

The errors flagged in the listing of subv are all warnings or obsolete 645 instructions, because the attached listing was produced on the 6180.

APPENDIX B Unlocked Stack Base Listing

This appendix contains listings of the four modules which make up the code needed to exploit the Unlocked Stack Base Vulnerability described in Section 3.3.3. The first two procedures, di and dia, implement step one of the vulnerability – inserting code into emergency_shut-down.link (referred to in the listings as esd.link.) The last two procedures, fi and fia, implement step two of the vulnerability – actually using the inserted code to read or write any 36 bit quantity in the system. Figure 9 in the main text corresponds to di and dia. Figure 10 corresponds to fi and fia. As in Appendix A, obsolete 645 instructions are flagged by the assembler.

APPENDIX C Trap door in check$device_name Listing

This appendix contains listings of the trap door inserted in check$device_name in Section 3.4.5.2 and the two modules needed to call the trap door. Check$device_name is actually one entry point in the procedure check$device_index. The patches are shown in the assembly language listing of the code produced by the PL/I compilation of check$device_index. Most of the patches were placed in the entry sequence to check$device_index, taking advantage of the fact that PL/I entry sequences contain the ASCII representation of the entry name for debugging purposes. Since the debugger cannot run in ring 0, this is essentially free patching space. Additional patches were placed at each return point from check$device_name, so that the trap door would be executed whenever check$device_name returned to its caller.

Zg is a PL/I procedure which calls the trap door to either read or write any 36-bit word accessible in ring 0. Zg uses zdata, an assembly language routine, to define a structure in the linkage section which contains machine instructions with which to communicate with the trap door.

The trap door algorithm is as follows:

1. Set the bp register to point to the argument rcode. Rcode has been bound to zdata$code in the procedure call from zg and must lie on an odd word boundary.
2. Compare the double word at bp|1 with the key string in the trap door to see if this is a legitimate user calling. If the keys do not match, then just return. If the keys do match, then we know who this is and must proceed.
3. Do an execute double (XED) on the two instruction at bp|3. This allows the caller to provide any instructions desired.
4. The two instructions provided by zdata at bp|3 and bp|5 are ldq bp|5 and stq bp|7. Bp|5 and bp|7 contain pointers to the locations from which to read and to which to write, respectively. These pointers are set in zg.

Finally, the trap door simply returns upon completion of the XED pair.

APPENDIX D Dump Utility Listing

This appendix is a listing of a dump utility program designed to use the trap door shown in Section 3.4.5 and Appendix C. The program, zd, is a modified version of the installed Multics command, ring_zero_dump, documented in the **MPM Systems Programmers' Supplement** [6]. Zd will dump any segment whose SDW in ring zero is not equal to zero. In addition, zd will not dump the ring zero descriptor segment, because the algorithm used would result in the ring 4 descriptor segment being completely replaced by the ring 0 descriptor segment which could potentially crash the system. Zd will also not dump master procedures, since modifying their SDW's could also crash the system.

APPENDIX E Patch Utility Listing

This appendix is a listing of a patch utility corresponding to the dump utility in Appendix D. The utility, zp, is based on the installed Multics command patch_ring_zero, documented in the **MPM System Programmers Supplement** [6]. Zp uses the same algorithm as zd in Appendix D and operates under the same restrictions. A sample of its use is shown below. Lines typed by the user underlined.

```
zp pds 660 123171163101 144155151156
660 104162165151 to 123171163101
661 144040040040 to 144155151156
Type "yes" if patches are correct: yes
```

As seen above, the command requests the user to confirm the patch before actually performing the patch. The patch shown above changes the user's project identification from Druid to SysAdmin.

IEEE COMPUTER SOCIETY

APPENDIX F Set Dates Utility Listing

This appendix is a listing of the set dates utility described in Section 3.4.4. The get entry point takes a pathname as an argument and remembers the dates on the segment at that time. The set entry point takes no arguments and sets the dates on the segment to the values at the time of the call to the get entry point. Set remembers the pathname as well as the dates and may be called repeatedly to handle the deactivation problem discussed in Section 3.4.4.

GLOSSARY

Access

"The ability and the means to approach, communicate with (input to or receive output from), or otherwise make use of any material or component in an ADP System." [7]

Access Control List (ACL)

"An access control list (ACL) describes the access attributes associated with a particular segment. The ACL is a list of user identifications and respective access attributes. It is kept in the directory that catalogs the segment." [3]

Active Segment Table (AST)

The AST contains an entry for every active segment in the system. A segment is *active* if its page table is in core. The AST is managed with a least recently used algorithm.

Argument Validation

On call to inner-ring (more privileged) procedures, argument validation is performed to ensure that the caller indeed had access to the arguments that have been passed to ensure that the called, more privileged procedure does not unwittingly access the arguments improperly.

Arrest

"The discovery of user activity not necessary to the normal processing of data which might lead to a violation of system security and force termination of the activity." [7]

Breach

"The successful and repeatable defeat of security controls with or without an arrest, which if carried to consummation, could result in a penetration of the system. Examples of breaches are:

a. Operation of user code in master mode;
b. Unauthorized acquisition of I.D. password or file access passwords; and
c. Accession to a file without using prescribed operating system mechanisms." [7]

Call Limiter

The call limiter is a hardware feature of the HIS 6180 which restricts calls to a gate segment to a specified block of instructions (normally a transfer vector) at the base of the segment.

Date Time Last Modified (DTM)

The date time last modified of each segment is stored in its parent directory.

Date Time Last Used (DTU)

The date time last used of each segment is stored in its parent directory.

Deactivation

Deactivation is the process of removing a segment's page table from core.

Descriptor Base Register (DBR)

The descriptor base register points to the page table of the descriptor segment of the process currently executing on the CPU.

Descriptor Segment (DSEG)

The descriptor segment is a table of segment descriptor words which identifies to the CPU to which segments, the process currently has access.

Directory

"A directory is a segment that contains information about other segments such as access attributes, number of records, names, and bit count." [3]

emergency_shutdown

"This mastermode module provides a system reentry point which can be used after a system crash to attempt to bring the system to a graceful stopping point." [6]

Fault Intercept Module (fim)

The fim is a ring 0 module which is called to handle most faults. It copies the saved machine state into an easily accessible location and calls the appropriate fault handler (usually the signaller).

Gate Segment

A gate segment contains one or more entry points used on inward calls. A gate entry point is the only entry in a inner ring that may be called from an outer ring. Argument validation must be performed for all calls into gate segments.

General Comprehensive Operating Supervisor (GCOS)

GCOS is the operating system for the Honeywell 600/6000 line of computers. It is very similar to other conventional operating systems and has no outstanding security features.

IEEE COMPUTER SOCIETY

HIS 645

The Honeywell 645 is the computer originally designed to run Multics. It is a modification of the HIS 635 adding paging and segmentation hardware.

HIS 6180

The Honeywell 6180 is a follow-on design to the HIS 645. The HIS 6180 uses the advanced circuit technology of the HIS 6080 and adds paging and segmentation hardware. The primary difference between the HIS 6180 and the HIS 645 (aside from performance improvements) is the addition of protection ring hardware.

hcs_

The gate segment hcs_ provides entry into ring 0 for most user programs for such functions as creating and deleting segments, modifying ACL's, etc.

hphcs_

The gate segment hphcs_ provides entry into ring 0 for such functions as shutting the system down, hardware reconfiguration, etc. Its access is restricted to system administration personnel.

ITS Pointer

An ITS (Indirect To Segment) Pointer is a 72-bit pointer containing a segment number, word number, bit offset, and indirect modifier. A Multics PL/I aligned pointer variable is stored as an ITS pointer.

Known Segment Table (KST)

The KST is a per-process table which associates segment numbers with segment names. Details of its organization and use may be found in Organick [22].

Linkage Segment

"The linkage segment contains certain vital symbolic data, descriptive information, pointers, and instructions that are needed for the linking of procedures in each process." [22]

Master Mode

When the HIS 645 processor is in master mode (as opposed to slave mode), any processor instruction may be executed and access control checking is inhibited.

Multics

Multics, the Multiplexed Information and Computing Service, is the operating system for the HIS 645 and HIS 6180 computers.

Multi-Level Security Mode

"A mode of operation under an operating system (supervisor or executive program) which provides a capability permitting various levels and categories or compartments of material to be concurrently stored and processed in an ADP system. In a remotely accessed resource-sharing system, the material can be selectively accessed and manipulated from variously controlled terminals by personnel having different security clearances and access approvals. This mode of operation can accommodate the concurrent processing and storage of (a) two or more levels of classified data, or (b) one or more levels of classified data with unclassified data depending upon the constraints placed on the systems by the Designated Approving Authority." [7]

OS/360

OS/360 is the operating system for the IBM 360 line of computers. It is very similar to other conventional operating systems and has no outstanding security features.

Page

Segments may be broken up into 1024 word blocks called pages which may be stored in non-contiguous locations of memory.

Penetration

"The successful and repeatable extraction and identification of recognizable information from a protected data file or data set without any attendant arrests." [7]

Process

"A process is a locus of control within an instruction sequence. That is, a process is that abstract entity which moves though the instructions of a procedure as the procedure is executed by a processor." [14]

Process Data Segment (PDS)

The PDS is a per-process segment which contains various information about the process including the user identification and the ring 0 stack. The PDS is accessible only in ring 0 or in master mode.

Process Initialization Table (PIT)

The PIT is a per-process segment which contains additional information about the process. The PIT is readable in ring 4 and writable only in ring 0.

Protection Rings

Protection rings form an extension to the traditional master/slave mode relationship in which there are eight hierarchical levels of protection numbered 0 – 7. A given ring N may access rings N through 7 but may only call specific gate segments in rings 0 to N-1.

Reference Monitor

The reference monitor is that hardware/software combination which must monitor all reference by any program to any data anywhere in the system to ensure the security rules are followed.

a. The monitor must be tamper proof.
b. The monitor must be invoked for every reference to data anywhere in the system.

IEEE COMPUTER SOCIETY

c. The monitor must be small enough to be proven correct.

Segment

A segment is the logical atomic unit of information in Multics. Segments have names and unique protection attributes and may contain up to 256K words. Segments are directly implemented by the HIS 645 and HIS 6180 hardware.

Segment Descriptor Word (SDW)

An SDW is a single entry in a Descriptor Segment. The SDW contains the absolute address of the page table of a segment (if one exists) or an indication that the page table does not exist. The SDW also contains the access control information for the Segment.

Segment Loading Table (SLT)

The SLT contains a list of segments to be used at the time the system is brought up. All segments in the SLT come from the system tape.

Signaller

"signaller is the hardcore ring privileged procedure responsible for signalling all fault and interrupt–produced errors." [6]

Slave Mode

When the HIS 645 processor is in slave mode, certain processor instructions are inhibited and access control checking is enforced. The processor may enter master mode from slave mode only by signalling a fault of some kind.

Stack Base Register

The stack base register contains the segment number of the stack currently in use. In the original design of Multics, the stack base was locked so that interrupt handlers were guaranteed that it always pointed to a writable segment. This restriction was later removed allowing the user to change the stack base arbitrarily.

Subverter

The subverter is a procedure designed to test the reliability of security hardware by periodically attempting illegal accesses.

Trap door

Trap doors are unnoticed pieces of code which may be inserted into a system by a penetrator. The trap door would remain dormant within the software until triggered by the agent. Trap doors inserted into the code implementing the reference monitor could bypass any and all security restrictions on the systems. Trap doors can potentially be inserted at any time during software development and use.

WWMCCS

WWMCCS, the World Wide Military Command and Control System, is designed to provide unified command and control functions for the Joint Chiefs of Staff. As part of the WWMCCS contract for procurement of a large number of HIS 6000 computers, a set of software modifications were made to GCOS, primarily in the area of security. The WWMCCS GCOS security system was found to be no more effective that the unmodified GCOS security, due to the inherent weaknesses of GCOS itself.

REFERENCES

1. *ESD 1973 Computer Security Developments Summary*, MCI-74-1, December 1973, HQ Electronic Systems Division: Hanscom AFB, MA.

2. *IBM System/360 Operating System Service Aids*, GC28-6719-0, June 1970, IBM Corporation.

3. *Multics Users' Guide*, AL40, Rev. 0, November 1973, Honeywell Information Systems, Inc.: Waltham, MA.

4. *The Multiplexed Information and Computing Service: Programmers' Manual*, Revision 14, 30 September 1973, Massachusetts Institute of Technology and Honeywell Information Systems, Inc.: Cambridge, MA.

5. *Summary of the H6180 Processor*, 22 May 1973, Information Processing Center, Massachusetts Institute of Technology: Cambridge, MA.

6. *System Programmers' Supplement to the Multiplexed Information and Computing Service: Programmers' Manual*, 1973, Massachusetts Institute of Technology and Honeywell Information Systems, Inc.: Cambridge, MA.

7. *Techniques and Procedures for Implementing, Deactivating, Testing, and Evaluating Secure Resource-Sharing ADP Systems*, DoD 5200.28-M, January 1973, Department of Defense: Washington, DC.

8. *WWMCCS Security System Test Plan*, 23 May 1972, Joint Technical Support Activity, Defense Communications Agency: Washington, DC.

9. Anderson, J.P., *AF/ACS Computer Security Controls Study*, ESD-TR-71-395, November 1971, James P. Anderson and Co., Fort Washington, PA, HQ Electronic Systems Division: Hanscom AFB, MA.

10. Anderson, J.P., *Computer Security Technology Planning Study*, ESD-TR-73-51, Vols. I and II, October 1972, James P. Anderson and Co., Fort Washington, PA, HQ Electronic Systems Division: Hanscom AFB, MA. URL: http://csrc.nist.gov/publications/history/ande72.pdf

11. Andrews, J., M.L. Goudy*, et al.*, *Model 645 Processor Reference Manual*, revision 4, 1 April 1971, Cambridge Information Systems Laboratory, Honeywell Information Systems, Inc.: Cambridge, MA.

IEEE COMPUTER SOCIETY

12. Bell, D.E. and L.J. LaPadula, *Secure Computer Systems: A Mathematical Model*, ESD-TR-73-278, Vol. II, MTR-2547, Vol. II, November 1973, The MITRE Corporation, Bedford, MA, HQ Electronic Systems Division: Hanscom AFB, MA.

13. Bell, D.E. and L.J. LaPadula, *Secure Computer Systems: Mathematical Foundations*, ESD-TR-73-278, Vol. I, MTR-2547, Vol. I, November 1973, The MITRE Corporation, Bedford, MA, HQ Electronic Systems Division: Hanscom AFB, MA.

14. Dennis, J.B. and E.C. Van Horn, *Programming Semantics for Multiprogrammed Computations.* Communications of the ACM, March 1966. 9(3): p. 143-155.

15. Downey, P.J., *Multics Security Evaluation: Password and File Encryption Techniques*, ESD-TR-74-193, Vol. III, June 1977, HQ Electronic Systems Division: Hanscom AFB, MA.

16. Goheen, S.M. and R.S. Fiske, *OS/360 Computer Security Penetration Exercise*, WP-4467, 16 October 1972, The MITRE Corporation: Bedford, MA.

17. Graham, R.M., *Protection in an Information Processing Utility.* Comm. ACM, May 1968. 11(5): p. 365-369.

18. Inglis, W.M., *Security Problems in the WWMCCS GCOS System*, Joint Technical Support Activity Operating System Technical Bulletin 730S-12, 2 August 1973, Defense Communications Agency: Washington, DC.

19. Karger, P.A. and R.R. Schell, *Multics Security Evaluation: Vulnerability Analysis*, ESD-TR-74-193, Vol. II, June 1974, HQ Electronic Systems Division: Hanscom AFB, MA. URL: http://csrc.nist.gov/publications/history/karg74.pdf

20. Lipner, S.B., *Computer Security Research and Development Requirements*, MTP-142, February 1973, The MITRE Corporation: Bedford, MA.

21. Lipner, S.B., *Multics Security Evaluation: Results and Recommendations*, ESD-TR-74-193, Vol. I, MTR-3267, October 1978, The MITRE Corporation, Bedford, MA, HQ Electronic Systems Division: Hanscom AFB, MA.

22. Organick, E.I., *The Multics System: An Examination of Its Structure*. 1972, Cambridge, MA: The MIT Press.

23. Price, W.R., *Implications of a Virtual Memory Mechanism for Implementing Protection in a Family of Operating Systems*, PhD thesis 1973, Carnegie-Mellon University: Pittsburgh, PA.

24. Richardson, M.H. and J.V. Potter, *Design of a Magnetic Card Modifiable Credential System Demonstration*, MCI-73-3, December 1973, HQ Electronic Systems Division: Hanscom AFB, MA.

25. Saltzer, J.H., *Protection and the Control of Information Sharing in Multics.* Comm. ACM, July 1974. 17(7): p. 388-402.

26. Schell, R.R., P.J. Downey*, et al.*, *Preliminary Notes on the Design of Secure Military Computer Systems*, January 1973, HQ Electronic Systems Division: Hanscom AFB, MA. URL: http://csrc.nist.gov/publications/history/sche73.pdf

27. Schiller, W.L., *Design of a Security Kernel for the PDP-11/45*, ESD-TR-73-294, MTR-2709, December 1973, The MITRE Corporation, Bedford, MA, HQ Electronic Systems Division: Hanscom AFB, MA.

28. Schroeder, M.D. and J.H. Saltzer, *A Hardware Architecture for Implementing Protection Rings.* Comm. ACM, March 1972. 15(3): p. 157-170.

29. Smith, L.A., *Architectures for Secure Computing Systems*, ESD-TR-75-51, MTR-2772, April 1975, The MITRE Corporation: Bedford, MA, HQ Electronic Systems Division, Hanscom AFB, MA.

30. Walter, K.G., W.F. Ogden*, et al.*, *Primitive Models for Computer Security*, ESD-TR-74-117, 23 January 1974, Case Western Reserve University, Cleveland, OH: HQ Electronic Systems Division, Hanscom AFB, MA.

31. Weissman, C. *Security Controls in the ADEPT-50 time sharing system*. in Fall Joint Computer Conference. 1969, Vol. 35. AFIPS Conference Proceedings, AFIPS Press, Montvale, NJ. p. 119-133.

IEEE COMPUTER SOCIETY

Looking Back at the Bell-La Padula Model

David Elliott Bell
Reston VA, 20191
December 7, 2005

Abstract

The Bell-La Padula security model produced conceptual tools for the analysis and design of secure computer systems. Together with its sibling engineering initiatives, it identified and elucidated security principles that endure today. This paper reviews those security principles, first in their own time, and then in the context of today's computer and network environment.

1. Looking Back

I look back at the Bell-La Padula Model over a career in security engineering that began with a concentrated burst of security modeling between 1972 to 1975. It is difficult, therefore, to limit myself to modeling and to exclude security topics without which real systems would never reach the field. I choose, then, to look back on both the modeling work and its engineering siblings so as to highlight their contributions to the DNA of network and computer security. What follows is not a synthesized chronicle of everything that happened but my own experiences and knowledge since the publication of the Bell-La Padula model.

2. Before the Bell-La Padula Model

In the late 1960's, developments in commercial operating systems suggested the possibility of tremendous cost savings. Time-sharing was starting to provide commercial customers the ability to share the leasing costs of IBM and other big-iron computers through simultaneous or sequential use of the expensive mainframe computers. For those in classified government circles, this new capability promised even more savings. Before time-sharing, separate computers had to be used for each different security level which was processed on computers, or careful "color changes" had to be made so that the same equipment could be used sequentially to process information at different security levels (referred to as "periods processing"). There was therefore the possibility of sharing those computer systems across security levels, with an important proviso. It was crucial that that processing artifacts of each security level (files, registers, data) be kept rigorously separate with a high degree of confidence.

An initial effort in this direction was commissioning computer experts to test the security robustness of computer systems that were developed in response to market forces. The experts were called "tiger teams." The success of the tiger teams was spectacular. "It is a commentary on contemporary systems that none of the known tiger team efforts has failed to date" [1]. The situation was in reality even worse than it first sounds. Tiger Teams, flush with success in attacking and taking over system A, would try their successful system-A attacks on system B. Alarmingly, many previous attacks worked immediately. Even more worrying were the possibilities opened by a successful attack. After capturing the system and inserting a back-door entrance, penetrators could report the initial flaw and gain a reputation for good citizenship. This planting of back-doors, particularly back-doors that would persist through system and compiler recompilations, was documented in an Air Force report [2] and was the direct stimulus for the back door Ken Thompson described in his Turing lecture [3] [4].

The conclusions drawn from Tiger Teams included the ultimate futility of "penetrate and patch" and the necessity of designing and building computer systems using a sound notion of "security" in computer systems. The U.S. Air Force initiated a set of engineering tasks, paired with several parallel modeling efforts to produce "... a formal statement of what is meant by a secure system — that is a model or ideal design" [5].

3. Bell-La Padula Model, 1972–1975

3.1. Problem Statement

In the summer of 1972, The MITRE Corporation initiated its task to produce a report entitled "Secure Computer Systems." The report was to describe a "mathematical model of security in computer systems." This task was one

of several in an overall security project, mostly engineering tasks.

The modeling task fell to Len La Padula and me. Our initial review of available literature showed that some of existing papers were theoretical (emphasizing abstract computation), while others were intimately tied to specific computer operating systems. Our plan was to stay free of system-specific details, first identifying and investigating general principles, and afterwards addressing specific solutions.

3.2. Mathematical Foundations

Definition. Our initial work focused on a definition of "security" within a mathematical (conceptual) framework. At the start, we viewed access monolithically, rather like possession of a book from a lending library. Having just completed abstract investigations into data sharing and appreciative of the complexities of "deadly embrace," we began to think about the complexities that would result when an "object" (the generalization of all things that could hold information) had its security level changed. Would the system immediately change the security level? Would "subjects" currently accessing the object have their access terminated? Wouldn't sudden termination of access imply that the classification had been increased? A mole in the organization could cache copies of a wide array of documents, determine when something was upgraded, and pass copies along to her evil spy master. On the other hand, if one took the tack of delaying the upgrade of the classification until all users had voluntarily given up access, there was the possibility of indefinite delay: user A has access; user B requests and gets access; user A gives up access; user C requests and gets access; user B gives up access; user A requests and gets access. In the midst of these considerations we attended a project status meeting and provided a brief summary of the complexities of changing an object's security classification dynamically. We were directed to exclude dynamic changes of classification from our investigations. The direction to exclude dynamic changes of classification seemed to remove all interest from the problem. We set about to demonstrate that fact and to document our results.

Précis. Our report [6] provided a conceptual framework for talking about computer security, a vocabulary, and an initial result that showed that security (as defined) is inductive. We termed this result the "Basic Security Theorem," in contradistinction to the "Complex and Sophisticated Security Theorem" that we had hoped to produce[1]:

Basic Security Theorem. *Let* $W \subseteq R \times D \times V \times V$ *be any relation such that* $(R_i,\ D_j,\ (b^\star, M^\star, f^\star),\ (b, M, f)) \in W$ *implies*

(i) $f = f^\star$

(ii) every $(S, O) \in b^\star - b$ *satisfies* SC *rel* $f^\star$.

$\Sigma(R, D, W, z)$ *is a secure system for any secure state* z.

[1] This theorem and those that follow are included to show general form and changes to "the model" over time. The casual reader is not expected to grapple with the theorems' details. If the details are of interest, refer to the original reports.

3.3. A Mathematical Model

Our initial report left much to be explored. Finer-grained access control reflecting modes of access was missing. Consideration of access modes led to the unexpected identification of a hard-to-name information flow property, the $\star$-property. The relation W that conceptualized allowable changes of state was not constructive and was therefore insufficient for the analysis and formulation of core system calls that change the security state.

Access Modes. There is a dilemma in dealing with access modes in a general, conceptual way. One advantage of speaking in terms of α and ρ, as was done in the Bell-La Padula model, is that the mind focuses on the logic of the argument, preventing connotations from words (like "read" or "execute") from biasing the analysis. This primes the pump of readers' intuitions. A disadvantage lies in the same lack of connotation. In this volume, connotation-light mathematical symbology was used throughout, with the notable exception of the naming of access modes. The access modes within the model were r, w, a, e, and c, the first four obvious cognates for operating-system access modes *read*, *write*, *append*, and *execute* and the last representing "control" of extending or rescinding access to an object. (It was clear that the variety of control in setting access controls justified deferring a more detailed treatment until later.) This priming of the pump proved useful when the report was first published. In later use however, the connotations of access modes in actual computer systems caused some difficulties. Engineers insisted that execute access required "reading" the code before executing it. As a result, Multics's execute mode was later listed as corresponding to $\{r, e\}$. In later modeling work, the original modes of r, w, a, and e were replaced by the four combinations of view and alter access modes, view indicating the ability to see an object's contents, and alter indicating the ability to modify its contents.

Origin of $\star$-property. Careful consideration of generalized access modes led us to the realization that the lending-library analogy had outlived its usefulness. In the first volume, access was binary: either the file was accessed (checked out from the library) or not accessed (on the shelves). With fine-grained access modes, many different patrons could have simultaneous access in different modes. The important insight was that with files, unlike

physical books, a user could add marginalia without other users realizing. Information could also be altered or re-

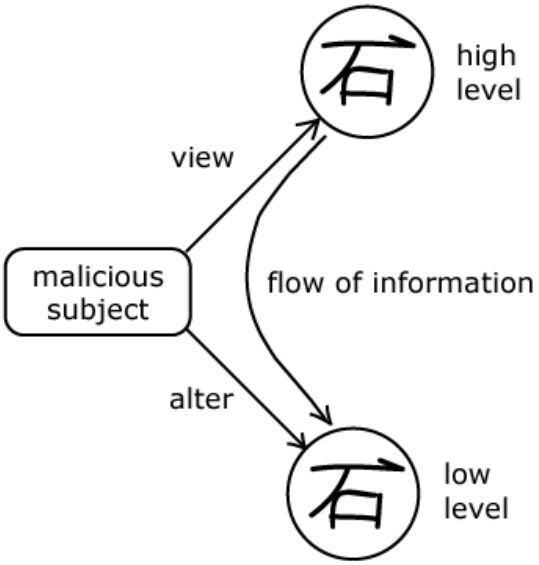

Figure 1. Information Flow

moved with no trace. Since unrestricted users could alter the contents of files undetected, there was the danger of information being copied from one file to another. If a paragraph from the intelligence summary could be copied into the bowling scores, then the system would have lost accurate control of the situation. Our analysis paralleled that in [7], where it was observed that one cannot protect resources with finer granularity than the protection mechanisms themselves support: different diamonds in a safety-deposit box receive exactly the same level of protection. Preventing this unwanted transfer of information admitted of two immediate solutions. One was to monitor every transfer of information (every `load` instruction, every `store`, and every `copy`). The other was to prevent simultaneous access to two objects if flow of information between them could be objectionable. The latter course was chosen, not least because of the overhead of making policy decisions for each instruction. The approach taken to preventing an unwanted transfer of information was to deny the second access request. That is, if the high-level object is successfully opened for read, a subsequent request to open the low-level object for write would be denied. If the first request is to write the low-level object, the later request to read the high-level object would be denied.

A condition to prevent deleterious flows was easily formulated, but a descriptive name was elusive. When I first raised the idea, I scribbled the heading "⋆-property" on the blackboard over a figure much like figure 2. After a burst of energetic discussion, I pointed out that if we didn't change

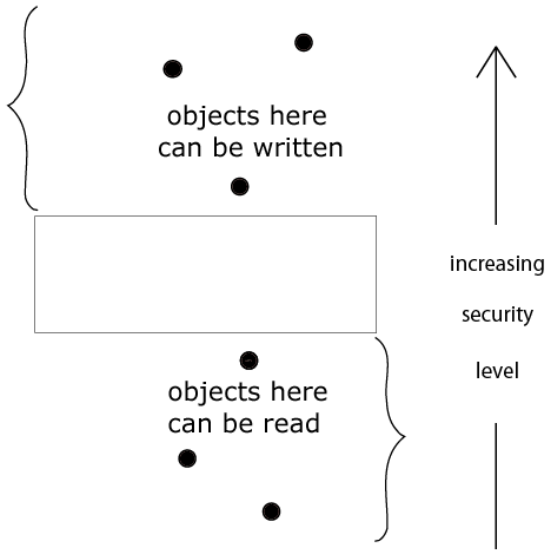

Figure 2. Original ⋆-property

the name right then, we'd be stuck with it forever. Nothing came to us and we continued our discussion. "⋆-property" it remained.

"Rule" Structure. W in the first volume was an undefined relation that conceptualized allowable changes of state. In the first volume, a relation with structure was not required in order to establish that volume-I security is inductive. If the model were to provide direct assistance in the formulation of core system calls, a more constructive form was desirable. The approach adopted was to specify W as a function of a set of "rules," each addressing a particular change of state. The conditions limiting the assembly of rules ω to define a complete relation $W(\omega)$ were minimal: each had to address a well-formed class of request and no two rules could have responsibility for the same request for change of state. Within that structure, a set of rules governing every possible change of state was devised: getting access to objects (in the four access modes, read, append, execute, and write); releasing access; giving access to another subject; rescinding other subjects' accesses; changing security levels; creating objects; and deleting them. Each of ten rules was then proved to preserve simple-security (that is, the subject's classification is greater than the object's classification); discretionary-security (that is, the subject is listed as having access to the object); and ⋆-property. Any combination of non-overlapping rules resulted in a conceptual system that stayed in secure states if it began in one.

Précis. The second volume [8] was a step away from the totally conceptual towards the pragmatic engineering needs of building a secure computer system. Aspects of this direction are found in the inclusion of access modes

and the use of rules that addressed all conceivable actions between subjects and objects rather than an undifferentiated relation W. The original definition of security was refined into simple-security, while nuances of different access modes, discretionary-secrity and $\star$-property were added. With these underlying changes, a corresponding Basic Security Theorem was stated and proved:

Theorem 4-1. *Let $\omega = \{\rho_1, \rho_2, \ldots, \rho_{10}\}$, the ρ_i as defined in the section entitled* The Rules*, and z_0 be a secure state which satisfies $\star$-property. Then $\Sigma(R, D, W(\omega), z_0)$ is a secure system which satisfies $\star$-property.*

3.4. Refinement of a Mathematical Model

At the outset of the second year's modeling work, there did not seem to be many pending topics and only one staff member (me) was assigned modeling duties. However, the engineering tasks attempting to use the modeling results in building secure prototypes were having difficulties. Their simplifications and frustrations led to a number of euphemistically termed "refinements" to make the model more usable and more attuned to the realities of computer systems.

Object Hierarchy. The first refinement addressed control of access privileges, based on the object hierarchy within the operating system. For Multics (and by inheritance, for Unix), control of objects (segments in Multics, directories and files in Unix) is limited by access to the object's parent directory. As a result, applying the general notion of control in volumes II and III to Multics required an enormous amount of interpretation. The engineers building secure Unix prototypes and participating in securing Multics wanted and needed some conceptual guidance on the security implications of this variety of control. That is, they preferred me to make the conceptual leap rather than themselves. To deal with this diffuse, implicit control, the model had to be extended to include an object hierarchy H that indicates the structure of objects in any state. The collection of rules for building and analyzing systems was augmented by the addition of four alternative control rules: rules to give and rescind access in a system with implicit hierarchical control; and rules to create and delete objects, with and without "compatibility." The term "compatibility" was used for systems where the security levels are monotonically non-decreasing from the root directory down each pathname. That is, each object has a security level the same as or greater than that of its parent.

The development of compatibility is worth a brief discussion. Since metadata about a file resides in its parent directory in a hierarchical file system, the accuracy and protection of the parent is crucial to the proper mediation of access to the file. In many cases, files and directories are even different instances of a single underlying operation system construct, segment for Multics and file for Unix. I

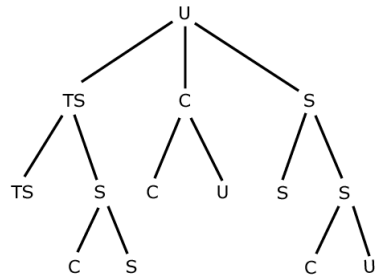

Figure 3. Anti-Compatibility

believed that a parent should be more classified than its children because it contains crucial information about them. As a result, my position was that a pathname should be monotonically non-increasing (staying the same or decreasing) away from the root, except for the first step. Since the root is already an exception (having no parent), an additional exception did not seem of much concern. This way, no subject could introduce errors into the metadata of any object without having been granted a higher level of clearance that the object itself. Researchers at Case Western Reserve University argued the reverse [9]. Since pathname resolu-

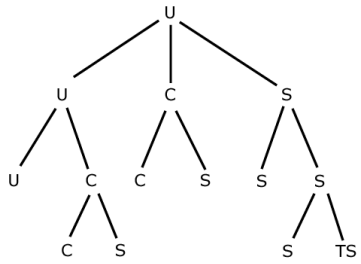

Figure 4. Compatibility

tion requires reading all intermediate nodes, they asserted that the path should be monotonically non-decreasing away from the root (staying the same or increasing). This disagreement was resolved by program direction, not by debate. I incorporated the Case position into the model as directed, terming it "compatibility" and giving Case credit, subtly disavowing responsibility.

There are two items of note. One was the dynamic nature of conceptual developments. The second was my error of conflation: "if this is important, it must be highly classified." With the hindsight of thirty years, I believe the issue was not the sensitivity of the object's metadata contents but its integrity (accuracy and modification limited to authorized agents). While some metadata can be read as half

of a covert channel, there is no conceptual reason that the classification of an object and that of its metadata have any particular relation to each other.[2]

Simplification of ⋆-property. The second refinement

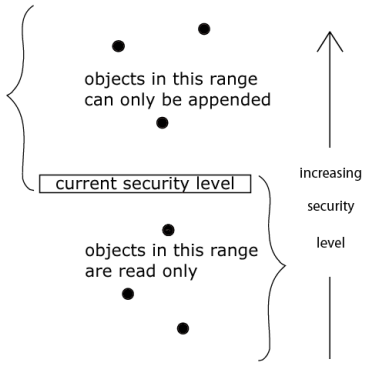

Figure 5. Revised ⋆-property

simplified the ⋆-property based on an engineering short cut. Lee Schiller noted that the original formulation of ⋆-property required comparing the security level of a newly requested object to that of every object to which the subject currently has access. His simplification was to record each subject's "current security level" in a system variable and to compare the new object's security level to the current-security-level, replacing many comparisons with a single one. For viewing accesses, the current security level had to be greater or equal to (or to "dominate") that of the new object. For altering accesses, the current security level had to be less than or equal to (or be dominated by) that of the new object. For access that included both view and alter, the two security levels had to be the same. This engineering short cut not only simplified the statement of the ⋆-property but also narrowed the gap between practical implementations and modeling versions. This refinement required the addition of the current security level to the modeling definitions, then recasting rules ρ_1, ρ_2, and ρ_4 (get-read, get-append, and get-write) in terms of current security level.

Untrustworthy and Trusted Subjects. The third refinement was a revision of the ⋆-property that introduced the concept of the "trusted subject." The stimulus for this revision came from the engineering task focused on building a secure operating system prototype. The intention was to apply the ⋆-property to every process within the prototype. Consider the scheduler. Its function is to swap jobs when conditions require. If the current process were running at TOP SECRET, then the scheduler would have to read all the current state information and write it into swap space. If the next process were UNCLASSIFIED, the scheduler would similarly have to read all swapped out information, move it into place for execution, then kick off the new process. In summary, the scheduler would have to read and write TOP SECRET information and then read and write UNCLASSIFIED information. "What security level," the engineers asked me, "shall we assign to the scheduler?" On reflection, the premise of the original ⋆-property was that *no* subject could be trusted not to copy part of the intelligence summary into the bowling scores. That assumption clearly does not apply to the core of a secure operating system where one can and would insist that the entirety of such core processes be carefully and thoroughly reviewed. The model was altered to identify a subset S' of the full set S of all subjects. These subjects were the subjects that "are untrustworthy and may mix information as described" [10]. The concept of "⋆-property" was replaced by "⋆-property relative to S'" and the subjects outside S (mathematically, the set $S - S'$) were not subjected to the restrictions under the original ⋆-property. Subjects as a whole were divided into two parts, "untrusted subjects" and "trusted subjects." Those "trusted subjects" were precisely those who "will never mix information of different security levels" [11].

Précis. The three refinements all derived from engineering experience in trying to apply the security model to actual system implementations. Raising these issues to the project modeler benefited both the modeling and the engineers: the model was refined to represent actual computer systems more faithfully and the modeling results provided justification and more explicit guidance to system prototyping and implementation.

Revised BST [12]. *If one desires a secure computer system which exhibits only compatible states satisfying the ⋆-property, one can use the set of rules*

$$\omega'_{iii} = \{\rho_1, \rho_2, \rho_3, \rho_4, \rho_5, \rho_{12}, \rho_{13}, \rho_{15}, \rho_{16}, \rho_{17}\}$$

together with a compatible, secure initial state z_0 which satisfies the ⋆-property [relative to S'].

Note that omitted rules ρ_6 through ρ_{11} are the non-hierarchy versions of giving and rescinding access, creating and deleting objects, and changing the security level function f. Those non-hierarchical rules were neither voided nor superseded and are available wherever they are applicable.

3.5. Unified Exposition & Multics Interpretation

Background. The first three volumes of the Bell-La Padula model were produced during two fiscal years. In the following year, there were other priorities. The year after that, an additional modeling effort was mounted. The

[2]The cross-product of a data sensitivity lattice with an integrity lattice happens to be a sublattice of a boolean lattice. Metadata integrity *could* be handled with modern lattice policies, but that subtlety of thought came later.

reasons were several. One was that the changes in the model over three volumes made it difficult for readers and engineers to keep the definitions and state-changes rules straight. The other was the large conceptual distance between the model as it stood and the Multics operating system that was in the process of being hardened and secured.

The "unified exposition" part of the tasking was easy for us to understand. The "Multics interpretation" part required an understanding of the Multics operating system. A request for information and assistance from the staff assigned to securing Multics resulted in us receiving a copy of Elliott Organick's *The Multics System: An Examination of Its Structure* [13] with an injunction to learn the system from the book. Learning Multics from Organick was highly educational. The exercise revealed operating system theory and practice in a concentrated, focused way. Independent study provided cross-training for systems engineering and analysis, in addition to security modeling and "security engineering."

Précis. The fourth volume gathered the changing definitions and results into a single place [14]. It provided new "rules" tailored to the Multics system. It provided proofs for the new Multics-specific rules,

It also included a time-sensitive statement that sounded like eternal truth: "All significant material to date on the mathematical model has been collected in one place in the Appendix of this report" [15]. This statement was true at the time with the known audience (computer security specialists in 1974). It was not true for later readers that had not worked in computer security during the 1970's.

This volume completed the transition of Bell-La Padula modeling from the purely conceptual to the system-specific. The general conceptual tools, tempered by engineering use and frustration, had been brought to bear on a specific system problem, the securing of Multics.

Theorem A10 [16]. *Let ρ be a rule and $\rho(R_k, v) = (D_m, v^\star)$, where $v = (b, M, f, H)$ and $v^\star = (b^\star, M^\star, f^\star, H^\star)$.*

1. *if $b^\star \subseteq b$ and $f^\star = f$, then ρ is ss-property-preserving.*
2. *if $b^\star \subseteq b$ and $f^\star = f$, then ρ is $\star$-property-preserving.*
3. *if $b^\star \subseteq b$ and $M^\star_{ij} \supseteq M_{ij}$,, then ρ is ds-property-preserving.*
4. *if $b^\star \subseteq b$, $f^\star = f$, and $M^\star_{ij} \supseteq M_{ij}$, then ρ is secure-state-preserving.*

4. Observations on Modeling Results

4.1. Clear Definition

The Bell-La Padula Model demonstrated the importance of a clear definition of the "security" being addressed. Without a clear definition, one faces unending complaints about "essential" aspects of security being omitted. "How can you call a system secure if it doesn't prohibit (or require) ℵ?"

4.2. Grounded in Engineering Reality

The modeling also implicitly demonstrated the importance of being grounded in immediate engineering problems. A purely academic exercise in computer security would not have unearthed and addressed the practical issues in building or retrofitting an operating system for sound security.

4.3. Toolbox Produced

Consider the Bell-La Padula modeling reports as a toolbox for the design, analysis and implementation of secure computer systems. After the first four reports had been published, the toolbox was full and complete. It included a descriptive capability: a precise definition of "security" that incorporated information compromise, fine-grained control of individuals to information objects, and preventive measures to prohibit information from flowing from high classification levels to lower classification levels. There were general results guaranteeing that security would be present provided the system began in a secure state and never introduced a security violation during a change of state. There were "rules" for handling a wide array of routine state changes (getting and releasing access; giving and rescinding access privileges; creating and deleting objects) as well as a method of combining arbitrary sets of such rules to describe a particular instance of a secure system. The complete set of rules included both generic rules as well as rules closely tailored to the Multics operating system, the first instance of a specific modeling solution. I believed then as now that no radical security modeling would be required later, with two possible exceptions. One would be the need to analyze a form of "security" beyond those specified: compromise, fine-grained discretionary access control, and $\star$-property-like information flow. Another would be the need to analyze technology artifacts that did not patently correspond to the descriptive machinery already available. Between 1975 and the present, both exceptions were experienced, but only infrequently.

4.4. Avenues of Futility

At the time, no one felt it necessary to state that pinning one's security hopes on having smarter geeks than the opposition was a failed concept. Neither did most experts believe that market forces would produce secure systems: they hadn't produced systems that stopped the 1970's tiger teams. Moreover, prominent industry representatives were

arguing that the only security required was personnel security, while technical security was bunk.

4.5. Strong, Built-in Security

Experts agreed that deep, significant security required security being built in and being built for high confidence. Today, we would say such systems require "high assurance."[3] Commercial systems, even commercial systems with some security labels, were not sufficient to keep information of adjacent collateral classification levels separate: UNCLASSIFIED from CONFIDENTIAL, for example. Too much was known about the weakness of such systems through tiger team results to believe that feature hardening or security appliques (security features added on top, or inserted in the middle) could really protect one's resources from a dedicated, resourceful, and intelligent foe. Imagine the devastation if your foe had the skills and resources of the wily hacker [17], much less those of highly skilled security specialists [18].

4.6. Self-Protection and Subversion

Both the engineering and modeling tasks had addressed flanking attacks in the form of subversion. The last modeling report discussed "sabotage" ("undesired alteration or destruction of information by purposeful action of an agent") and integrity, noting that a system's ability to protect itself is essential and that a lack of strong self-protection makes conceptual analysis worthless [19]. The engineering tasks went further in demonstrating how penetration and the insertion of backdoors and other maleficent logic could completely undermine the intended security characteristics of a system. Just as important, the demonstrated harm that subversion could do necessitated anti-subversion requirements, especially formal verification and trusted distribution.

It is worth including a description of one of the most ominous subversion attacks. It is called the "two-card loader," through analogy to mainframe loaders. A general-purpose loader could be packed into two punched cards. The hard-wired loader would read those two cards into memory then pass execution to the two-card loader. It would then load any program for execution. A two-card loader subversion would add a general-purpose program loader to a system. The loader begins execution after receipt of an obscure signal, similar to the signals used to unlock "Easter eggs" in commercial software. This kind of addition to an allegedly secure system would allow an attacker to craft a pointed attack much later and launch that attack on command. The possibility of commercial software having covert two-card-loader logic makes tight controls over a secure system's configuration vitally important.

[3] By high assurance I mean confidence in proper design and implementation represented by systems at B3 and A1. Marketing claims that B1-like systems are "high assurance" are purposefully misleading or a reflection of ignorance.

5. General Developments, 1975–2005

Engineering and conceptual efforts continued from 1975 through 1983, but I was not involved. When I returned to security work in 1983, there were three parallel and interrelated strands of security development: consolidating the successes of computer security, a technology shift to smaller computers and more networking, and initial glimmerings of new security policies.

5.1. Consolidation

At the beginning of the 1980's, there was a concentrated effort to solidify and build on the successes of the 1970's work in computer security and to learn from the failures. The various engineering tasks, conceptual tasks, and research prototypes had identified security principles, methods and techniques and proved those techniques in exemplar systems such as the Kernelized Secure Operating System (KSOS) [20], the Provably Secure Operating System (PSOS) [21], the Kernelized Virtual Machine (KVM) [22], and Multics [23]. Steve Walker at the Office of the Secretary of Defense initiated the Computer Security Initiative with three goals: to formulate a metric for measuring the security of systems, to establish a center whose mission was to measure commercial systems against that metric, and to initiate a technical conference dedicated to computer security. The metric was the *Trusted Computer System Evaluation Criteria* [24], published in 1985. The center was the Department of Defense Computer Security Evaluation Center (later the National Computer Security Center). The conference was the Department of Defense/National Bureau of Standards Computer Security Conference (later the National Computer Security Conference).

The construction of the *TCSEC* was an exercise in giving partial credit. The best security known became the highest security class, A1. The lower classes were developed as logical subsets of security requirements down to "no security at all," or D. Solid commercial systems with good discretionary access control were expected to meet the requirements of the C2 class, "Controlled Access Protection." Adding security labels to some of the objects in the system resulted in the B1 class, "Labeled Security Protection." C2 and B1 systems were viewed as no stronger than that required to keep order between cooperative colleagues. At B2, "Structured Protection," the first approximation of a really secure system could be seen. At B3, "Security Domains," the system has its core (its "trusted computing

base") minimized and it is the lowest class that is "highly resistant to penetration." A1, "Verified Protection," added life-cycle protection against changes to the hardware and software (anti-subversion), trusted distribution of the system and updates, and the use of "formal verification."

I had doubts whether "formal verification" was necessary, based on my reading of the 1970's proofs-of-correctness literature and more particularly on my co-authorship of a MITRE report [25] on the topic of software validation for certification. In the mid-1970's, there seemed to be a disconnect between most of the literature on proof-of-correctness and the situation of validating a secure system. Proofs-of-correctness had focused on simple comparisons of an algorithm to a program embodying that algorithm: a principal class of papers addressed bridge-bidding programs, for example. The challenge for verifying a secure system was avoiding insecurity and maintaining a state of security. As the *TCSEC* was being finalized, I was surprised to learn that past intentions to verify automatic verification systems had been quietly abandoned. While at the Computer Security Center, I became more familiar with both the theory and practice of automatic verification systems and my reservations were not reversed. Work-arounds to assure that the verification tools could process the large specification files included manual stubs, drivers, and artful commenting-out of sections of the specification to assure that the tools could run to completion. In many ways, human brain-power provided a back-stop for the limitations of the tools. Nevertheless, I came to feel that formal specifications held some value. My feeling was captured best by Marv Schaefer, observing that the greatest benefit of formal verification is gained in writing down the specification. Normally loquacious verificationists were struck dumb.

Part of the original motivation for the *TCSEC* and the Trusted Product Evaluation Program was to streamline and regularize the formal process of approving Department of Defense systems for operation – certification and accreditation (C&A). While there were policies and procedures for assessing secure computer systems, there were problems with having each acquisition's engineering staff assess the underlying operating system or systems for security compliance. Not only was there no reuse of the analytical results, but also there was no guarantee of consistency across acquisitions. If all certification engineers facing, say, a Trusted Solaris component could refer to a published summary of its security properties in the context of its *TCSEC* rating of B1, then redundant analyses would be avoided and consistency across different acquisitions would be increased. Moreover, since computer security was a specialty field, the acquisition engineering staff were not always trained and current in the relevant security topics. The intention was to centralize and reuse the security evaluation results to obviate redundant analyses, to assure consistent results, and thus to provide an acquisition task with a solid, published product evaluation report.

The Trusted Product Evaluation Program was begun immediately after the Center was formed in 1981. The first candidate systems included the commercial "archetype" systems, the systems whose successes were the basis for the requirements in the *TCSEC*. The initial set of candidates included SCOMP at A1 (verified security) and Multics at B2 (structured protection). At the lower levels were Unisys's MCP/AS, IBM's RACF, Hewlett Packard's MPE V/E, CA's ACF2/VM, CA's TOP SECRET/NVS, and CDC's Network Operating System at C2 (best commercial practice); and Univac's OS 1100 at B1 (labeled security protection). Over time, many more C2-candidate systems entered the program and completed evaluation. The most significant was Windows NT 3.5 in 1995. The most significant almost-B1 system in today's context is Trusted Solaris V2.5.1 which received an EAL4/B1 rating in 1998. Both of these systems, of course, are inadequate to counter any focused attack; their proper use is among cooperating friends. At the higher levels, the successful systems were VSLAN, Multics, and Trusted Xenix at B2; XTS-300/400 at B3; and SCOMP, GEMSOS, Boeing's MLS LAN at A1. Honorable mention also goes to DEC's Security-Enhanced VMS (SEVMS), an A1 candidate nearing final evaluation when the product was withdrawn from the market [26]. The success of these systems was far less than might have been hoped, but it was a substantial achievement.

5.2. Technological Changes

At the same time that the computer security successes of the 1970's were being consolidated, Moore's Law and general networking were changing the face of computing. Computers began to shrink and traditionally clear categories of computers (computers, minicomputers, microcomputers) began to blur and overlap. The technological changes began to accelerate away from the assumptions and verities of computer security just as they were being codified.

5.3. Policies

From the beginnings of the Computer Security Initiative, there were objections that the Department of Defense and the intelligence community security policies based on classifications did not apply in the civilian and commercial world. Work in security policies remained largely the bailiwick of the classified world and pure academics until 1989. Up until that time, new conceptual policies were primarily minor alterations to existing policies, variations on discretionary security, for example. When David Clark and David Wilson published "A Comparison of Commercial and Military Security Policy" [27] in 1987, it was widely hailed and

stimulated further consideration of novel security policies. Its primary contribution to security policies was to introduce access triples (user, program, file), where previous work had used duples (user, file) or (subject, object). The following year witnessed an explosion of Clark-Wilson response papers (see for example Lee [28] and Shockley [29]). In 1989, Brewer and Nash published "The Chinese Wall Security Policy" [30], abstracting British financial regulations. Its contribution was a user's free-will choice limiting future actions in a non-discretionary way. The multiplicity of these policies, however, was more apparent than real, as will be described in paragraph 6.3.

5.4. Cross-Currents

The technology movement away from single logical platforms well understood in the computer security world posed a dilemma for the Computer Security Center with regard to the *TCSEC* and the product evaluation program. A series of initiatives focused on the special security issues, first in networks and later in database management systems. The results were the *Trusted Network Interpretation of the Trusted Computer System Evaluation Critieria* (*TNI*) [31] and the *Trusted Database Interpretation of the Trusted Computer System Evaluation Critieria* (*TDI*) [32]. Absent clear promulgated guidance on networking issues, product evaluation faced a hard choice: exclude networking (unrealistic) or include networking without clear guidance (risky). Neither choice was good. Caution led to the exclusion of networking features in the early days.

The growth of networking brought the "composition problem" to the fore: what are the security properties of a collection of interconnected components, each with known security properties? Efforts were made to derive methods and principles to deal with the unconstrained composition problem, with limited success. In the sense of product evaluation and eventual system certification and accreditation, one needed to leverage the conclusions about components to reach conclusions about the whole without having to reanalyze from first principles. Unfortunately, closed-form, engineering-free solutions were not discovered and may not be possible.

6. Conceptual Design, 1975–2005

6.1. Consolidation

Part of consolidation was the inclusion of a requirement for security policy modeling for higher security products (B2 and above) in the *TCSEC*. In addition, the *TCSEC* listed the Bell-La Padula model as a good representative of a model with the characteristics required. As a result, I undertook modeling interpretation several times in early product evaluations (Multics at B2 and SCOMP at A1). Later, I produced a modeling interpretation for Trusted Xenix as part of a product evaluation [33].

The formulation of interpretations of the *TCSEC* for networks (*TNI*) and later for databases (*TDI*) necessarily included the topic of mathematical modeling. Conceptualizing networks, in particular, pivoted on the scope of the network components being considered. A homogeneous network (one viewable as a single virtual networked system) admits of a conceptual design. Not so with an *ad hoc* heterogeneous network (one without a single worldview). The problem was precisely the unconstrained composition problem and the need to consider engineering interfaces in a general way, as described below.

6.2. Technology Changes

In the early 1980's the A1 security modeling requirement was levied on Blacker Phase 1 (BP1), an inherently networked system [34]. The conceptual difficulties were several. BP1 was composed of three different components, often geographically remote. The three componens had different design specifications and only working in concert did they achieve the requirements of the overall system. In addition, the networking nature of the BP1 components raised puzzling questions about all the A1 requirements. Specifically, each of the components was a trusted computer system in itself, while participating with its siblings to produce a "trusted network system." The dual faces of the A1 requirements were documented for the project and were later provided as input to the invitational workshop in New Orleans, convened to begin work on guidance for trusted networks. Application of traditional modeling results to the individual components was straightforward, but addressing the larger BP1 network was not at all obvious. Eventually, it became clear that the system itself was concerned with an entirely different set of subjects and objects than were the operating systems of the components. The system administered a policy concerning the labeled "connections" that subscriber hosts could establish. Thus, at the network level, the active entities were "hosts" and the resources were "connections." This realization led to a modeling interpretation of the Bell-La Padula model that conceptualized hosts and connections (called *liaison* in the model) [35]. Note that this interpretation did not build on the Multics-specific rules, but on the general rules in volume 3, augmented by rules constructed to address the specific topic of access-controlled *liaison*. This was an example of modeling being required because available resources did not patently apply.

In the *TNI*, composition is addressed at length, but composition there is tightly constrained. In broad outline, it is similar to the approach taken in BP1. To evaluate an network composed of independently evaluated components,

one must have network policies for identification and authentication, audit, discretionary access policy and mandatory access policy; a network architecture; protocols for communicating user identification, authentication, audit, labeling; and defined interfaces for reporting events to audit components. This constitutes a network-view together with engineering limits and requirements that made the *TNI* usable and successful [36].

In conceptualizing heterogenous networks, the general composition problem is unavoidable and one faces enormous difficulties in the absence of a network-view like that of the *TNI*. With the increasing use of networking and systems construction from smaller quasi-independent parts, understanding the security properties of a collection of connected secure systems is extremely important. I found none of the conceptual and modeling results fully satisfying (primarily because they ignored the engineering issues included in the *TNI*) and therefore was unconvinced. From a purely modeling point of view, there is an explicit presumption of a global awareness of all aspects of the conceptual system. Few security models (if any) even include a way of talking about an interface to another system. It is therefore hard to see how two different models, even two instances of the same model, could relate to each other and produce a single security model. In a broader context, the *TCSEC* requirements outside the modeling requirements are even more difficult to satisfy. In fact, I conjectured that there is no closed-form solution to the composition of several secure systems that does not include a network-view and engineering interfaces and protocols.

Consider two identical A1 systems. Those systems *might* be combined so that they mesh perfectly, both operationally and theoretically, in an A1 composite. They could also be also be combined so badly that the composite has no security at all (and would be rated D). One could probably artfully adjust the requirements failed in such as way as to make the composite any *TCSEC* level between D and A1. It is the engineering details of the interface that determine whether systems complement and enhance each other or undermine each other's original security qualities.

The unavoidability of heterogeneous networks and the lack of a purely conceptual way to analyze their security properties was a hard dilemma. One could not stand in the way of the network tide coming in. Nor could one insist on homogeneous networks throughout. In the absence of multilevel networking equipment that shielded unilevel hosts and networks, there were very few practicable courses of action.

What was generally chosen was to deploy isolated networks. The rationale was that if all users and all hosts on a particular network were cleared for the information on that network and were trusted with the network's operational mission, then one could manage without multilevel secure networks. Unfortunately, the isolation between networks was incomplete. There were and are operational needs to move information between networks of different classifications and sensitivity levels. Those connections were accomplished with guards and other dedicated equipment whose function was to enforce the rules about communication between the networks. Over time, the multilevel nature of these components came to be ignored, abetted by the usual way of viewing networks with their details hidden. A prime example is the network cloud. It is important to remember that the cloud is made up of processing platforms and communication lines. Hence, a thin line between two separate network clouds is, on closer inspection, a combination of communications lines and computing platforms. That being the case, the following "Computer Security Intermediate Value Theorem" (CS-IVT) becomes important:

CS-IVT. *If computer A at security level α is connected to computer B at security level β through a network cloud and $\alpha \neq \beta$, then some processing platform in the cloud is multilevel.*

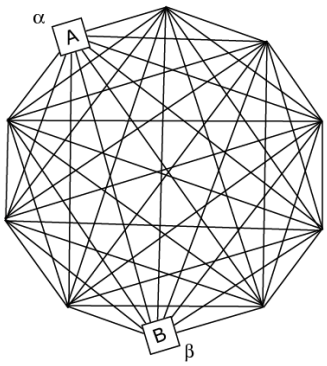

Figure 6. Intermediate Value Theorem

***Proof Sketch*:** If it were not so, then each node on any path through the cloud to the distant end would be single-level at the same security level as its immediate predecessor. As a result, the first and last nodes would be operating at the same level, a contradiction.

The irreducible complexity of computer security cannot be eliminated by assigning a multilevel function to a component that is represented by a line in a network cloud or a wiring diagram. Claims that highly secure systems can be constructed from low-security or untrusted system components and lines on a blackboard have been roundly criticized every time they have been propounded, and with good reason: they have to contend with the CS-IVT. Connecting two isolated networks with a slim line creates a single network with a single cut-point. The CS-IVT then says that some

platform is multilevel. If neither cloud is multilevel, then the slim line is. This is an illustration of the conservation of complexity: you can relocate the complexity of multilevel security, but you cannot eliminate it. Unfortunately, some have forgotten this lesson or are trying to ignore it.

6.3. Policies

In 1987 before the explosion of new policies stimulated by Clark-Wilson, I began to ponder a remark made by Marv Shaefer. I had just made a presentation on lattices for commercial isolation. As I was returning to my seat, Marv observed that we could not then support a policy expressed as an *OR*. "Why not?" I wondered. I began a personal investigation which quickly turned into self study in lattice theory using Birkhoff's classic text [37]. I came to realize that naive use of lattice theory in computer security's infancy had led to an identification of the fundamental elements in a policy (which I termed "policy alphabet") with the lattice's "atoms" (the elements directly above the $\oslash$ element. This realization justified the working solution I had earlier

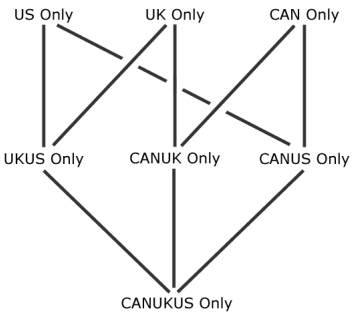

Figure 7. CANUKUS

crafted for multilateral sharing, such as CANUKUS information (information shared in various combination between the governments of Canada, the United Kingdom, and the United States). Further, a fundamental boolean lattice result had implications for security policies and for implementations. That result is as follows [38]:

Thm 11. *The meet-irreducibles of the boolean lattice on the alphabet A are*

$$\bigwedge_{a \in A} s(a),$$

where $s(a)$ is either a or $\neg a$ for a in the alphabet A.

Since there are many different-seeming formulations of boolean lattices, one can choose the formulation that is most familiar. All representations lead to the same boolean lattice (up to isomorphism). Most telling, one of the formulations combines uninterpreted symbols with *AND*, *OR*, and *NOT*. Surely *most* security policies can be so defined. For example, the usual boolean lattice *can* deal with *OR*s of the policy elements

$$\{\textit{FINANCIAL}, \textit{ENGINEERING}, \text{STRATEGIC}\}$$

by using the meet-irreducibles (atoms)

$$\{F \wedge E \wedge S, F \wedge E \wedge \neg S, \\ F \wedge \neg E \wedge S, F \wedge \neg E \wedge \neg S, \\ \neg F \wedge E \wedge S, \neg F \wedge E \wedge \neg S, \\ \neg F \wedge \neg E \wedge S, \neg F \wedge \neg E \wedge \neg S\}.$$

Any security policy that can be defined with *AND*/*OR*/*NOT* is isomorphic to any other one of the same width (number of atoms). If they have different widths, the smaller one is isomorphic to a sublattice of the larger one. An implementation built to support any boolean lattice could support any other boolean policy, given enough width and the provision of a personality module. This result was published in 1990 and demonstrated that statements that the military security lattices are not applicable to other situations were in error [39].

After presenting my paper, I asked Jim Anderson whether he'd heard my talk. He had not, but observed that if I could use my results to support ORCON, he knew people who could use it immediately. Even 1-step ORCON would be a big help. (ORCON stands for "Originator Control." It refers to access granted only for reading. Any use or citation must be explicitly requested and granted by the originating organization before it can be used.) My first thoughts were that ORCON could not be done. If you provide an electronic copy of a report, no more control can be exerted. My second thoughts considered the publication process for classified documents more closely. Classified publications are first drafted and then reviewed before publication and dissemination. From that perspective, a solution was constructed that marked each object with the originating organization and distinguished between DRAFT and PUBLISHED materials. Changes of state were governed by approval officials for each organization. An analyst preparing a DRAFT could use any information to which she had access in her own organization and any PUBLISHED information that other organizations had made available to her organization. Publication (producing a PUBLISHED clone of a DRAFT document) required approval by the local organization and any organization whose information was included. With joint approval, roll-back, and limited changes of state, one-step ORCON had a conceptual design. With only a few extensions, n-step ORCON did too.[4]

With multilateral sharing, one-step ORCON, and n-step ORCON solved, I attempted to collect all the claims of

[4] Note that this conceptual design requires a homogeneous environment. All the computers and components must be cooperating to enforce the policy. The design does not solve ORCON in a heterogeneous environment.

"different" security policies in the literature, noting in particular whether the authors claimed their policy was outside the DoD's classification policy. I structured my analysis in terms of an abstract computer that included a lattice policy built with *AND*/*OR*/*NOT* and access control lists. For this abstract machine, I produced constructive solutions for every security policy in the literature. In fact, two solutions were provided, a strong solution building on nondiscretionary security policy and a weak solution building on discretionary security policy. The weak solution was included for those situations where strong, labeled security policy is not available. The results were reported in "Putting Policy Commonalities to Work" [40]. This was an example of conceptual work addressing policies outside confidentiality, discretionary security, and $\star$-property. This paper showed tangentially that the profusion of new security policies did not pose the problem it might have. Had all the new policies been fundamentally different, vendors would have had to choose which policies to support and which to neglect. With the ability of any boolean policy implementation to support any other boolean policy, the plethora of policies merely provided comfortable terminology for all communities.

7. Observations for the 21st Century

7.1. Overview

Our computer security legacy at the beginning of the 21st Century is not extensive, but neither is it inconsiderable. It is the result of dedicated work by master technologists and government champions. Lasting principles were hammered out conceptually, then refined and honed on engineering workbenches. The lessons were codified and an immense effort to prime the pump with secure, trusted systems was undertaken, compensating for the ineffectiveness of market forces. In the opening years of this century, our legacy includes two very high security products and the community knowledge to deploy them wisely. One could argue that we are in a much better situation now than in the early 1980's. The general feeling, however, is that computer and network security is in decline. Why is that? I believe it is the confluence of three currents of change: technology, the processes for fielding secure systems, and Government's role in nurturing and expanding our secure system resources.

7.2. Technology

Technology's increasing pace of development and shortening product cycles have made most computer users full-time beta testers. Something can be imagined, demonstrated, and become mission-critical before a stable version is available. In Internet time, "immediate gratification takes too long" [41] is an understatement. Our networks are assemblies of beta products, hooked together heterogeneously with more attention to features than to fundamental security. Added to that is the normal marketing of commercial products. One's own feature set, or dominance of the market, or previous deployment — *any* impression of a market advantage can and will be used to make a sale. There is nothing wrong with that in any particular case. It is an unbroken sequence of such decisions and interactions with other currents that causes problems.

7.3. Fielding Systems

At its inception, the Computer Security Initiative's intent to improve security certification suffered from a dearth of secure products. Doubtful or hostile program managers (government and contractor) could cite competitiveness regulations and system requirements not met (sometimes carefully crafted) by available secure products to seek waivers. This reluctance to embrace change was neither unexpected nor blameworthy. That initial reluctance, however, combined with an increasing pace of technological change and slow increases in the availability of secure products, resulted in stultification and no meaningful progress towards the routine use of secure products.

The formal process of evaluation suffered under shrinking time-to-market schedules because analysis and understanding take time and thought. The structural changes to product evaluation initiated in the early 1990's had the effect of terminating community evaluation of highly secure systems. The stock of secure-system resources was therefore frozen and attrition further thinned the ranks. Without sales of their secure products, the vendors rightly viewed them as under-performers.

On the certification and accreditation front, quick was the enemy of strong security. System acquisitions and deployments increasingly began to face hard choices between sound security and speedy approval. Interested parties in the form of vendors lobbied in favor of speedy approval, and incidentally, sales of their products. I know from direct experience that successful system acquisition requires balancing many different demands and imperatives. One must make compromises to deliver at all. The best one can hope is to minimize compromises on essential issues. Such compromises, however, are local optimizations. A sequence of local optimizations need not produce global optimization. Were I currently a program manager who needed a highly secure component, I would far rather leverage someone else's selfless act than act selflessly myself. The running joke is that everyone wants to be second through the gate.

With few secure resources to build upon, with little C&A to leverage, and with active lobbying by the vendors of low-

security systems, it is easy to see why individual program managers and certifiers do not lead ambitious initiatives to better the security in deployed systems. Surely someone somewhere should be doing so.

7.4. Nurturing Secure System Resources

It is hard to argue persuasively that anyone is currently nurturing our commercial security resources. From the 1960's through the 1980's, the military services and ARPA funded research and development in computer security. In the early 1980's, the Department of Defense assigned formal evaluation of commercial products to the National Security Agency, paired with the mission of encouraging secure products and mandating their use. None of these missions flourish today.

At the beginning of the 21st Century, product evaluation for high-security systems is moribund. DARPA correctly views computer and network security as past the research stage. Individual program offices rightly have primary focus on their program, not on the greater good. No one has the mission of "selfless acts of security."

7.5. What Can We Do?

The first thing we must do is tell ourselves the truth about security. Mostly isolated networks have not solved our security problems. The Computer Security Intermediate-Value Theorem still holds: slim lines between single-level networks *are* multilevel devices, secure or not. Because the slim lines are multilevel, it is unconscionable to use overly weak components. Such connections require high security, meaning A1, although in some cases one could accept a B3 solution. Systems of C2 or B1 heritage (Windows NT/XP/Vista or Trusted Solaris, respectively) are totally inadequate. Their forebears could not thwart the Tiger Teams of the 1970's; they cannot stop a skilled and dedicated attacker today. A Cuckoo's Egg attack on an operational system would be bad enough. A two-card loader subversion requiring only a six-line "toehold" would be catastrophic [42] [43].

As a corollary to telling ourselves the truth, we must look askance at those who insist that semi-isolated networks *have* solved our problems. Do they not know? or do they ignore the risks they expose us to?

Not only can we use our scarce high-security resources for the critical network connections, the current prevalence of networking actually works to our advantage in this case. There was once the vision of a world of computing where every computing platform was multilevel and secure. Contrary to the assertions of critics, no one I knew ever believed multilevel security had to be pervasive to succeed at all. Every multilevel project I ever worked on explicitly acknowledged and dealt with a mixed environment: multilevel secure platforms and single-level platforms, interacting. (At a conceptual level, a single-level system is a degenerate multilevel system: its highest and lowest levels are the same.) Now, by contrast, our networks are heterogeneous and the connections between networks of different security levels are relatively few. There is the opportunity, therefore, to place high-security products at the weakest points, the connections between the not-quite-isolated networks.

Consider the kinds of connections one might want between networks. One might want a common file server. One might want a web server. The connection might be a multilevel database of a simple kind. Current web practice builds pages from information assembled from many sources. This makes utilization of several single-level databases or a simple SeaViews-style secure database eminently reasonable [44]. One might want thin clients attached to a multilevel server, itself attached to resources of different security levels. All of these could easily be built on current high-security platforms. None of them require functions available exclusively on weak platforms.

Suppose one built these applications on a *Yahoo!*-like base without a high level of security. The problems for a configurable *Yahoo!* portal are primarily functional: database access, web display, and the marshaling of searches via multiple search engines. Only protection against defacement and masquerade have a security flavor. Extending a *Yahoo!* situation to the connection of networks of differing security levels is a massive and fundamental change. The functional issues are the same, but the security requirements cannot be satisfied by tighter configuration or by constantly vigilant geeks. Without a solid, secure foundation, such functionality is built on such a shaky foundation that it cannot be trusted with multiple security classifications.

The next thing we must do is use the resources we have. The hard-won triumphs of the Computer Security Initiative have been neglected so long that it is a miracle anything has survived. We must constantly be on the lookout for opportunities to use them to good advantage. Knowing the conflicting pressures on active acquisitions, we must generate opportunities to perform selfless acts of security in the form of crafting and sharing reference implementations of widely needed components. A good list to start with is the following:

- high-security file servers,
- high-security static web servers,
- simple high-security multilevel databases,
- high-security dynamic web sites, and
- high-security thin-client architectures.

None of these breaks new ground in either security or in general functionality, but each of them would be an extremely valuable building block. (In fact, the first three build to the fourth.) We as a community need a champion to lead these and similar reference implementations for the good of the entire community. We cannot wait for market forces to reverse course and surprise us with truly secure building blocks.

8. Conclusion

Our computer security legacy needs to be used and it needs to be nurtured. If we do not, we risk losing what little we have and having our enduring security principles fade from community consciousness. In addition, we must remember that

- Tight configuration and smart geeks are not a substitute for good security.
- Multilevel secure components are unavoidable.
- Multilevel links between security levels must be as strong as we can make them.
- Government must perform its crucial role in nurturing our commercial security resources.

We have known for nearly a third of a century how to build and deploy strong and secure systems. Our shared knowledge applies in today's networked world with its emphasis on fast, secure information sharing. In the 21st Century, we should utilize our existing resources in the slim lines between networks. It is time we got started.

Acknowledgment

Thanks to all of my readers, with special thanks to Roger Schell who reminded me of the *TNI*'s treatment of composition.

References

[1] J. P. Anderson, "Computer Security Technology Planning Study," ESD–TR–73–51, Vol. I, AD–758 206, ESD/AFSC, Hanscom AFB, MA, October 1972, p. 4.

[2] Paul A. Karger and Roger R. Schell, "Multics Security Evaluation: Vulnerability Analysis," ESD–TR–74–193, Vol. II, ESD/AFSC, Hanscom AFB, MA, June 1974.

[3] Ken Thompson, "Reflections on Trusting Trust," *Comm ACM*, August 1984, **27**(8), 761–763.

[4] Ken Thompson, "On Trusting Trust," **Unix Review**, November 1989, **7**(11), 70–74.

[5] J. P. Anderson, "Computer Security Technology Planning Study," ESD–TR–73–51, Vol. II, ESD/AFSC, Hanscom AFB, MA, October 1972, p. 16.

[6] D. Elliott Bell and Leonard J. La Padula, "Secure Computer Systems: Mathematical Foundations," MTR–2547, Vol. I, The MITRE Corporation, Bedford, MA, 1 March 1973. (ESD–TR–73–278–I)

[7] Jerome H. Saltzer, Michael D. Schroeder, "The Protection of Information in Computer Systems," *Proceedings of the IEEE*, **63**(9) (September 1975), pp. 1278–1308.

[8] Leonard J. La Padula and D. Elliott Bell, "Secure Computer Systems: A Mathematical Model," MTR–2547, Vol. II, The MITRE Corporation, Bedford, MA, 31 May 1973. (ESD–TR–73–278–II)

[9] K. G. Walter *et al.,* "Primitive Models for Computer Security," ESD–TR–74–117, Electronic Systems Division, Hanscom AFB, MA, January, 1974.

[10] D. Elliott Bell, "Secure Computer Systems: A Refinement of the Mathematical Model," MTR–2547, Vol. III, The MITRE Corporation, Bedford, MA, December 1973. (ESD–TR–73–278–III), p. 25.

[11] *Ibid.*

[12] *Ibid.,* p. 34.

[13] Elliott I. Organick. *The Multics System: An Examination of its Structure*. The MIT Press: Cambridge, MA, 1972.

[14] D. Elliott Bell and Leonard J. La Padula, "Secure Computer Systems: Unified Exposition and Multics Interpretation," MTR–2997, The MITRE Corporation, Bedford, MA, July 1975. (ESD–TR–75–306)

[15] *Ibid.,* p. 5.

[16] *Ibid.,* p. 98.

[17] Clifford Stoll. **The Cuckoo's Egg**. Doubleday: New York, NY, 1989.

[18] Paul Karger and Roger R. Schell, "Thirty Years Later: Lessons from the Multics Security Evaluation," *Proceedings*, 18th ACSAC, Las Vegas, NV, December 2002, 119–126.

[19] Bell and La Padula, *op. cit.,* 1975, p. 70–71.

[20] E. J. McCauley and P. J. Drongowski, "KSOS—The design of a secure operating system," *Proceedings*, AFIPS 1979 NCC, v48, 345–353.

[21] P. G. Neumann, R. S. Boyer, R. J. Feiertag, K. N. Levitt, and L. Robinson, "A provably secure operating system: The system, its applications, and proofs," Technical Report CSL–116, SRI International, 1980.

[22] Marvin Schaefer, R. R. Linde, *et al.*, "Program Confinement in KVM/370," *Proceedings*, ACM National Conference, Seattle, October, 1977.

[23] Jerome Saltzer, "Protection and the Control of Information in Multics," *Comm. ACM* **17**(7), July 1974, 388–402.

[24] *Department of Defense Trusted Computer System Evaluation Criteria,* DoD 5200.28–STD, December 1985.

[25] D. Elliott Bell and Edmund L. Burke, "A Software Validation Technique for Certification: The Methodology," MTR–2932, Vol. I, The MITRE Corporation, Bedford, MA, April 1975. (ESD–TR–75–54)

[26] Paul Karger *et al.*, "A VMM Security Kernel for the VAX Architecture," *Proceedings*, 1990 IEEE Symposium on Security and Privacy, Oakland, CA, 7–9 May 1990, 2–19.

[27] David Clark and David Wilson, "A Comparison of Commercial and Military Computer Security Policies," *Proceedings*, 1987 IEEE Symposium on Security and Privacy, Oakland, CA, 27–29 April 1987, 184–194.

[28] T. M. P. Lee, "Using Mandatory Integrity to Enforce 'Commercial' Security," *Proceedings*, 1988 IEEE Symposium on Security and Privacy, Oakland, CA, April 1988, 140–146.

[29] William R. Shockley, "Implementing the Clark/Wilson Integrity Policy Using Current Technology," *Proceedings*, 11th National Computer Security Conference, Baltimore, Maryland, October 1988, 29–37.

[30] David Brewer and Michael Nash, "The Chinese Wall Security Policy," *Proceedings*, 1989 IEEE Symposium on Security and Privacy, Oakland, CA, May 1989, 206–214.

[31] *Trusted Network Interpretation of the Trusted Computer System Evaluation Criteria*, DoD 5200.28–STD, 31 July 1987, NCSC–TG–005.

[32] *Trusted Database Management System Interpretation of the Trusted Computer System Evaluation Criteria*, DoD 5200.28–STD, April 1991, NCSC–TG–021.

[33] D. Elliott Bell, "Trusted Xenix Interpretation: Phase I," *Proceedings*, 13th National Computer Security Conference, Washington, DC, 1–4 October 1990, 333–339.

[34] Clark Weissman, "BLACKER: Security for the DDN, Examples of A1 Security Engineering Trades," *Proceedings*, 1992 IEEE Computer Security Symposium on Research in Security and Privacy, 4–6 May 1992, Oakland, CA, 286–292.

[35] D. Elliott Bell, "Secure Computer Systems: A Network Interpretation", *Proceedings*, Second Aerospace Computer Conference, McLean, VA, 2–4 December 1986, 32–39.

[36] *TNI*, *op. cit.,* Appendix A.

[37] Garrett Birkhoff, *Lattice Theory* (1st ed.), American Mathematical Society: Ann Arbor, Michigan, 1948.

[38] *Ibid.*, p. 163.

[39] D. Elliott Bell, "Lattices, Policies, and Implementations," *Proceedings*, 13th National Computer Security Conference, Washington, DC, 1–4 October 1990, 165–171.

[40] D. Elliott Bell, "Putting Policy Commonalities to Work," *Proceedings*, 14th National Computer Security Conference, Washington, DC, 1–4 October 1991, 456–471.

[41] Carrie Fisher. **Postcards from the Edge.** Simon & Schuster: New York, NY, 1987.

[42] Cynthia E. Irvine, "Considering Lifecycle Subversion," Invited presentation, MLS Workshop, Alexandria, VA, 24 September, 2003.

[43] Emory A. Anderson, Cynthia E. Irvine, and Roger R. Schell, "Subversion as a Threat in Information Warfare," *Journal of Information Warfare*, **3**(2), June 2004, pp. 52–65.

[44] Dorothy Denning, *et al.,* "A Multilevel Relational Data Model," *Proceedings*, 1987 IEEE Computer Security Symposium on Research in Security and Privacy, 27–29 April 1987, Oakland, CA, 220–242.

Looking Back: Addendum

David Elliott Bell

Abstract

The picture of computer and network security painted in my 2005 ACSAC paper was bleak. I learned at the conference that the situation is even bleaker than it seemed. We connect our most sensitive networks to less-secure networks using low-security products, creating high-value targets that are extremely vulnerable to sophisticated attack or subversion. Only systems of the highest security are sufficient to thwart such attacks and subversions. The environment for commercial security products can be made healthy again.

1 Introduction

In the preparation of a recent ACSAC paper [1], I confirmed my opinion that computer and network security is in sad shape. I also made the editorial decision to soft-pedal criticism of my *alma mater*, the National Security Agency. At the conference, I found that I wasn't pessimistic enough and that there are additional disturbing developments.

Muting my criticism served neither my audience nor the national interest well.

2 Highlights of "Looking Back"

In today's networks, multilevel secure connections between networks are unavoidable.[1] The threat of sophisticated attack or subversion makes low-security systems unacceptable for connections between isolated networks. High-security systems must be used.[2] Government Off-The-Shelf (GOTS) products have never been satisfactory substitutes for commercial products. Before the Trusted Product Evaluation Program (TPEP), market forces alone never produced medium-security systems, much less high-security systems. A third way was needed — the changed business environment produced by Steve Walker's Computer Security Initiative.[3]

What we needed then and what we need now are "selfless acts of security" that lead to strong, secure commercial products.[4] Market-driven self-interest does not result in altruistic product decisions. Commercial and government acquisitions with tight deadlines are the wrong place to expect broad-reaching initiatives for the common good. Government must champion selfless acts of security separately from acquisitions.

3 NSA Has the Mission

In 1981, the Computer Security Evaluation Center was established at the National Security Agency (NSA). It was charged with publishing technical standards for evaluating trusted computer systems, with evaluating commercial and GOTS products, and with maintaining an Evaluated Products List (EPL) of products successfully evaluated. NSA was in effect appointed the Department of Defense (DoD) champion for high-security products and systems.

Changes in DoD and national policy have not fundamentally changed NSA's computer security mission (now, information assurance missions). NSA remains a government champion of standards and product evaluation of medium- and high-security systems.[5] [6]

4 Environment for High Security

Steve Walker's Computer Security Initiative intended to change the playing field for security systems. It succeeded. The combination of security standards, a list of evaluated

[1] This section is a précis of the relevant portions of my earlier paper. In the sections to follow, the supporting facts and reasoning have been relegated to appendices to simplify presentation. Appendix A is a plan of action. Appendix B is a summary of the results from last year's paper needed for this exposition.

[2] The author believes that the term "high assurance" has been debased by its frequent application to B1 and EAL4 systems, systems that provide no resistance to attack. In this paper, the term "high security" is used to refer to B3/A1 or EAL6/EAL7 systems. "Medium security" refers to B2/EAL5 systems. "Low security" refers to all other systems.

[3] Steve Walker, while at the Office of the Secretary of Defense, began the Computer Security Initiative to make major changes to computer security practice in the late 1970's. See appendix D.1 for more details.

[4] I use the term "selfless acts of security" to refer to actions or activities that have beneficial security results outside the immediate organization that performs them. Research into networking technology that led to the Arpanet and then to the Internet was a selfless act of networking. Developing a security metric and insisting on systems that measure up against that metric were selfless acts of security. Producing security archetypes or reference implementations of common networking components (file servers, dynamic web servers) would be selfless acts of security.

[5] The National Institute of Science and Technology (NIST) shares responsibility for running the National Information Assurance Partnership.

[6] See appendix C for NSA's original mission, policy changes since the Center's inception, and the current mission under DoD Directive 8500.1, "Information Assurance (IA)."

security products, and enforced policy to *use* evaluated products changed industry's calculus for decisions about "trusted" operating systems. IBM was disqualified from an NSA contract because it did not propose evaluated products. The DoD was serious about security and the marketplace took note.

Some of the archetype systems that had demonstrated and proved the consensus security principles were commercial products and underwent product evaluation (SCOMP at A1 and Multics at B2). Honeywell retained its Multics product line until it was cancelled in 1985. Honeywell (and subsequent owners of the technology) continued to use their own money to support and improve its SCOMP/XTS product line up to the current day. Private money was also put into developing new high-security systems. Gemini's Gemsos and Boeing's Secure LAN pursued A1 ratings successfully. Digital Equipment sought A1 for its Security Enhanced VMS (SEVMS) and was on a path to success when the product was withdrawn. Verdix, IBM Federal, and Trusted Information Systems pursued B2 ratings. Amdahl decided to submit its Multiple Domain Feature for evaluation at a B2 level, as did IBM with its PR/SM.[7]

With the transition from the *TCSEC* to the Common Criteria and with a shift in NSA's priorities to NSA-led security efforts like the Multilevel Information Systems Security Initiative (MISSI), the positive environment for high and medium security evaporated. The implicit assurance of swift accreditation for MISSI-based systems and an NSA imprimatur led to a rapid collapse of the commercial high-security marketplace. MISSI did not deliver on its promises and the business environment remains stunted.[8]

5 NIAP Performance

Since the inauguration of the National Information Assurance Partnership (NIAP), NSA has been the DoD lead for standards (the *Common Criteria* and protection profiles), new product evaluations, and maintenance of previously rated products. Its execution of standards work has been professional and thorough. New product evaluations have numbered one hundred twenty-seven, but only one of them was high security (EAL7) and another two were medium security (EAL5). The only evaluation against a validated U. S. Government Protection Profile was XTS-400's EAL5 augmented validation. The other validations were against product-specific Security Targets.[9]

NSA has not nurtured pre-existing high-security products well. The RAting Maintenance Program (RAMP) set up under the Trusted Product Evaluation Program (TPEP) provided a way to keep an evaluation rating of an evolving product current. RAMP was part of product evaluation — the phase immediately after the award of a rating. Since NIAP began, no effective method of transitioning existing high-security products into the NIAP family has been implemented. NSA/CCEVS's[10] position is that vendors should start over with a new *Common Criteria* evaluation. RAMP was intended to preclude complete re-evaluations of evaluated products. CCEVS's position is technically indefensible and reneges on the promise of RAMP.[11]

6 Competition with Vendors

Part of NSA's mission is to encourage high-security commercial products and their utilization. That encouragement has always been done through a combination of jawboning and leading by example. Leading by example includes both utilizing existing solutions in one's own operational systems and devising new solutions. The combination of urging the use of high-security components and using them yourself is a powerful argument to uncertain government players.

Unfortunately, there is an imbalance in NSA's leading by example. There has been much activity in devising new solutions under the Information Assurance Technical Framework (IATF)[12] and demonstration initiatives such as SELinux, NetTop, and recently the separation kernel.Not much has been evident in utilizing pre-existing high-security products for NSA projects. This is in contrast to the broad advocacy of NSA's products, most of which are low security. SELinux is not a trusted operating system according to NSA's SELinux FAQ web page[13] and commercial versions are a low-security EAL4. NetTop has not been evaluated but it builds on an EAL4 base. An EAL7 U. S. Government Protection Profile for a separation kernel is in development,[14] but its technical basis is unconvincing and frequently hard to follow.[15]

Since the technology transfer program for NSA's exemplar systems has been weak, NSA's internal initiatives are called "sandboxes" by critics, with some justification. Internal development activities should also address the need for high-security solutions that are evaluated at arm's length from the designers. They should then be effectively integrated into the marketplace to complement and not to compete with existing high-security commercial products.

[7] Neither Amdahl nor IBM received a B2 rating, but IBM received an EAL4 rating and several EAL5 ratings under the Common Criteria in Germany.

[8] See appendix D for additional details.

[9] One difficulty with the *Common Criteria* scheme is that product-specific Security Targets can be written to avoid hard security problems. As a result, even EAL7 does not assure one that a product does not have fatal flaws.

[10] Common Criteria Evaluation and Validation Scheme.

[11] See appendix E for details.

[12] "The IATF is a common reference guide for selecting and applying adequate and appropriate IA and IA-enabled technology in accordance with the architectural principles of defense-in-depth ..." [2]. DoDI 8500.2 directs NSA to "[m]aintain, update, and disseminate the Information Assurance Technical Framework (IATF) ... in coordination with the National Institute of Standards and Technology (NIST)" [3].

[13] Appendix F includes the relevant text from the FAQ.

[14] "U. S. Government Protection Profile for Separation Kernels in Environments Requiring High Robustness," dated 1 July 2004, is a protection profile "in development" [4].

[15] See appendices F, G, and H for details on SELinux, NetTop, and the separation kernel, respectively.

NSA's jawboning should be refocused on encouraging the use of commercial high-security products, rather than proselytizing for NSA's low-security demonstration systems.

7 Prevalence of Low Security

Very difficult computer- and network-security problems confront us today. This is best seen in the need for increased communication between different security domains and in the need to reduce desktop clutter caused by separate workstations for each different security domain. Both of these situations call for high levels of security, both from a policy perspective and from first principles.

Currently most of these critical needs are being filled by low-security components, whose deployments are contrary to DoD policy.[16] Even in the absence of similar policy, such deployments are ill-advised, verging on irresponsible. An attack on a weak connection no more sophisticated than the Cuckoo's Egg[17] gives an attacker complete control. Subversion of a weak connection allows adversaries to mount attacks of their choice at any time in the future. Such attacks can steal information, alter critical information, or bring down computers or entire networks. The worst subversion, called the "two-card loader," cannot be prevented or even found in low-security systems.[18] Direct attacks like the Cuckoo's Egg can only be thwarted by full mediation and self-protection, as required by B2/EAL5 or better systems. Complete prevention of subversion requires life-cycle security of the form found in A1 systems.[19]

There are many factors leading to this prevalence of low-security systems. One is the paucity of available high-security components. Another is a Morton's fork: either unawareness of these resources or a refusal to use them. Neither fork is justifiable.

NSA bears some responsibility for the low level of awareness because it has not jawboned for the right things and it has not led by example. DoD initiatives built on low-security platforms contribute to the problem.

Knowing about high-security resources and rejecting their use is nearly always wrong. On time, on budget, but unacceptably weak is not a defensible compromise. NSA could help this situation by crafting solutions for common problems utilizing existing high-security products.[20] Reference implementations of this sort would be a boon to program managers by lessening their technical, schedule, and certification risk. They could also be a springboard for updating the Information Assurance Technical Framework with content rather than the current placeholders for multilevel workstations, multilevel servers, and multilevel network components.

8 Conclusion

NSA's mission to nurture commercial high-security systems is languishing. At the same time, NSA resources are being expended in apparent, and sometimes explicit, competition with commercial vendors. These sandbox endeavors suffer from unacceptably low security. Existing high-security components are being overlooked or rejected in government initiatives where their use is crucial.

The United States should not be using low-security solutions in its most critical and most targeted components. The cost of a sophisticated attack or subversion here is extremely high. All players need to recognize the problem and take action to make sure that high-security components are used where they are needed.

NSA needs to re-focus on highly secure commercial products. It should incorporate existing high-security products into the NIAP family. It should encourage the development and deployment of new high-security products, especially through updates of the Information Assurance Technical Framework. NSA should redouble its efforts to encourage proper use, to lead by example, and to recreate the business environment where high-security products make good business sense.

All government organizations should utilize existing high-security products to solve their hardest security challenges: interconnecting single-level networks, providing multilevel secure workstations (both thin client and thick client), and providing timely information interchange among traditional enclaves of classified information. Relying instead on new solutions that (even if successful) will only be available in five to ten years, is not only impractical but also negligent of the public good.

Acknowledgments

The insights of my readers made their peer review more like a collaboration. My thanks to Dave Amos, Jeremy Epstein, Paul Karger, Kenneth Olthoff, Tom Parenty, Ed Reed, Marv Shaefer, Roger Schell, Brian Snow, and Steve Walker.

[16] DoDI 8500.2 requires "controlled interfaces" for connections between a classified network and a lower-classified one. See appendices I, J and K for details on initiatives that are using low security rather than available high security.

[17] See appendix B.3 for a description.

[18] See appendix B.4 for details.

[19] The *Common Criteria* does not include strong requirements for trusted distribution. Instead, it has a single Delivery family, ALC_DEL, which requires that a developer document procedures for delivery and use those procedures. Hence, the EAL levels do not speak to trusted delivery at all. What one needs to prevent subversion of a high- or medium-security system is A1-style configuration management and trusted delivery.

[20] Such as data sharing between security domains, multilevel dynamic web servers, and multilevel thick and thin clients. See appendix A.2.

A Plan of Action

This appendix is an abbreviated plan of action. The first subsection is an outline of the plan. Rationale for the course of action is provided in the second subsection.

A.1 Summary

- Encourage commercial high-security products
  - Sponsor high-security reference implementations of common components
  - Establish an interim High-Assurance-Platform initiative using existing commercial products
  - Develop a high-security DoDIIS Trusted Workstation (DTW)
  - Develop a high-security Multiple Domain Dissemination System (MDDS)
- Advocate use of commercial high-security products
- Decrease advocacy of NSA's low-security initiatives
- Debate the technical merits of NSA's separation kernel
- Incorporate High-Security EPL Products into NIAP
  - Update or create new protection profiles to match B3 and A1
  - Vet the new or updated B3/A1 protection profiles
  - Add Gemsos, XTS-400, and Secure LAN to the Validated Products List.

A.2 Reasoning

The positive business environment that emerged from the Computer Security Initiative built on four pillars:

- a standard way of specifying security requirements
- evaluation of products against those requirements
- policy to require use of evaluated products
- evidence that the policy would be enforced

The current situation is that there is a standard way of specifying security requirements (the validated U. S. Government Protection Profiles), a program of evaluating products (CCEVS) and policy to require use of validated products (DoDD 8500.1 and NSTISSP No. 11), but the evidence that the policy will be enforced is weak. This weakness shows itself in several ways:

- Advocating weak (EAL4) NSA products
- Advocating those NSA products over stronger commercial ones
- Designing new ten-year solutions but not dealing with interim needs

NSA should re-energize its role to encourage industry to produce high-security products as the DoD lead in NIAP, "promoting the development of technically sound security requirements for IT products ... and appropriate measures for evaluating [them]" [5]. One useful initiative in this direction would be to assemble basic high-security components, built on existing high-security platforms. This effort should be simultaneous implementation on both Gemsos and XTS-400 with a cross-feed of lessons learned and shared source-code licenses for databases.

Two extremely common components are web-based connections between security domains and thin-client operations. For web-based connections, the order of implementation could be a multilevel file store, a static multilevel webserver, an extremely simple SeaViews-like multilevel database system, and finally a dynamic multilevel webserver. Multilevel thin-client architectures using high-security platforms can be achieved with sound security engineering. All the difficult problems for this initiative have already been worked out. What is required is to apply engineering talent and a modest amount of money. The experiences and the results could feed directly into updating placeholder sections in the IATF [6], as well as lead to high-security versions of the DTW and MDDS.

A second worthy initiative would be to address the same needs that the High Assurance Platform is addressing,[21] but sooner. A parallel effort to address these needs in the interim using commercial high-security products would be helpful to the user community at the same time as it reinforces NSA as the advocate for commercial high-security products.

A third initiative would be to increase the security of low-security DoD initiatives such as the Multiple Domain Dissemination System (MDDS) and the DoDIIS Trusted Workstation (DTW) by porting to or re-implementing on high-security commercial platforms.

A fourth initiative would be to establish a consensus concerning the separation kernel, or partitioning kernel, or "Multiple Independent Levels of Security" (MILS). Advocates should gather or produce publication-quality materials covering relevant technical topics such as definitions, descriptions, and rationale. Critics and advocates should then debate the issues publicly. After this trial by fire, either the concept will be tested and stronger or its flaws will be clear. In the interim, efforts to further proselytize or to finalize protection profiles should be put on hold.

As a last initiative, a review of existing U.S. Government Protection Profiles would be beneficial in assuring that all the important needs of the U.S. government are included as well as to determine whether there are updates needed. In this review, participants should include not only evaluators and Common Criteria staff but also vendors and village elders in the computer and network security community.

These initiatives would demonstrate NSA's commitment by publicly utilizing existing high-security products in vis-

[21] In a five- to ten-year timeframe.

ible endeavors and in leading by example. Combined with a scale-back in technical advocacy for NSA's low-security demonstration systems, the business atmosphere could be improved substantially.

B Previous Results

B.1 Intermediate Value Theorem

The Computer Security Intermediate Value Theorem (IVT) can be used to show that connections between single-level networks *are* multilevel.

CS-IVT. *If computer A at security level α is connected to computer B at security level β through a network cloud and $\alpha \neq \beta$, then some processing platform in the cloud is multilevel.*

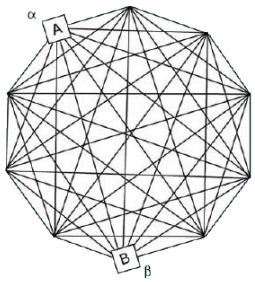

Figure 1. Network Cloud

***Proof Sketch*:** If it were not so, then each node on any path through the cloud to the distant end would be single-level at the same security level as its immediate predecessor. As a result, the first and last nodes would be operating at the same level, a contradiction [7].

In layman's language, every network path between two nodes of different security levels includes a multilevel node, strong or weak as the case may be.[22]

B.2 Inevitability of Multilevel Connections

Consider two networks operating at two different levels, as shown in figure 2. When they are connected as shown,



Figure 2. Mostly Isolated Networks (MIN)

they become a single network. By the IVT, every path from the left network to the right network has a multilevel node on it. But every node in the individual networks is single-level by assumption. Thus, the multilevel node exists on the connection between the two networks.

For good reasons, the classified networks of the United States are currently operated as single-level networks with limited connections between them. Since those connections must be multilevel, one would prefer them to be as strong as we know how to make them. Unfortunately that is not the case.

B.3 Cuckoo's Egg Attack

The Cuckoo's Egg attack described by Cliff Stoll was made possible by the ability of a user application to substitute the attacker's program for a system program (*atrun*) that ran every five minutes with superuser privileges. "The whole operation depended on his being able to move a file anywhere he wished" [8]. The initial step was to log on to an existing account with a stolen password. Logged on, the attacker created a bogus version of *atrun* then substituted it for the system's *atrun* using a feature of Emacs email. Unix actually allowed Emacs to replace another account's file without permission and without effective security checks. When the bogus *atrun* ran, it created a new superuser account for the attacker and restored the legitimate *atrun*. Success depended on a violation of presumed discretionary access controls: no one should be able to overwrite privileged code with code of their choosing. A properly configured, self-protecting B2 or better system would have prevented this substitution.

B.4 Subversion

The Cuckoo's Egg breach was discovered by a smart geek who noticed a 75¢ discrepancy in the accounting. This kind of attack is bad, but a smart geek can find it. Subversions are much worse. In 1974, a subversion of Multics was demonstrated that required only ten 36-bit words [9]. Recently, the "two-card loader" has gained notoriety. The two-card loader is named after the mainframe loader that was punched into two cards (too large to fit on one card). The hardware reads the two loader-cards into a well-known location in memory, then transfers control to the first line of the loader program. The loader program, in turn, reads in the rest of the card deck and transfers control to the program contained therein.

A two-card loader subversion of an operating system reads in a malicious program as data, then transfers control to it. If a two-card loader is hidden in a commercial (or GOTS) operating system, then it can lie silently waiting for its trigger before doing its single, very simple job. A geek cannot find a well-written two-card loader. Exemplar subversions with a six-line "toehold" have been demonstrated [10], whereas Microsoft was unable to find an entire video game hidden in Excel before release.

[22] "Multilevel" here means processing information of more than one security level or classification at a time. This includes a node that is sequentially single-level, swapping out levels when they are dormant.

Systems meeting A1 requirements (specifically the configuration-management and trusted-distribution requirements) can prevent subversion through life-cycle control over design, source code, and the object code.

C NSA's Mission

Steve Walker began the Computer Security Initiative in 1977 to pursue a third way to get truly secure computer systems into the marketplace [11]. The three parts of the initiative were a technical center for product evaluations, criteria against which evaluations could be performed, and a technical conference dedicated to computer-security issues. The first two invitational workshops were held in 1977 and 1978. Those workshops became the DoD/NBS Computer Security Conference.[23] In 1981, the Department of Defense Computer Security Evaluation Center was established. In 1983, the first version of the *Trusted Computer System Evaluation Criteria* (*TCSEC*) was published [12]. The Department of Defense version was published in 1985 [13]. The Trusted Product Evaluation Program (TPEP) was begun in 1981, using preliminary versions of the *TCSEC* which had been produced as part of the Computer Security Initiative.

C.1 Original Direction

In 1982, Department of Defense Directive 5215.1 [14] gave the National Security Agency (NSA) the responsibility to create the Computer Security Evaluation Center "as a separate and unique entity within the NSA" and charged it to execute nine tasks, including:

1. "Establish and maintain technical standards and criteria for the evaluation of trusted computer systems ...."
2. "Conduct evaluations of selected industry and government-developed trusted computer systems against these criteria."
3. "Maintain and publish an EPL [Evaluated Products List] of the selected industry and government-developed trusted computer systems that is suitable for use by the DoD Components."

C.2 Criteria

The *TCSEC* delineated seven classes of "trusted computer system," divided into four divisions. The best security that had been demonstrated was called "A1"; the other classes defined lower levels of security that could be used in situations not demanding the highest level of security. Security sufficient to thwart a frontal attack is found in systems from B2 up to A1.

[23] Renamed the National Computer Security Conference and finally the National Information Systems Security Conference.

C.3 Evaluation Program

The Trusted Product Evaluation Program was a program to evaluate and record the fidelity of commercial products to classes within the *TCSEC* for later use by system certifiers and accreditors. Products successfully completing a "formal evaluation" were placed on the Evaluated Products List (EPL). TPEP and EPL constituted an institutional program of selfless security.

C.4 Maintaining Ratings

One element missing from the original TPEP was the care and maintenance of successful evaluations. Since 1981 was at the beginning of the incredible shrinking product cycle, trusted systems faced a serious dilemma.

The dilemma lay in the need for trusted systems to be both commercially up to date and current in their evaluation. Since the product cycle was shrinking but the evaluation process was as short as it could be, trusted systems were behind the leading edge of commercial features and thus less appealing as building blocks. In addition, changes to a trusted system could require that at least part of its evaluation be re-visited. A complete re-write of the TCB would clearly have a substantial effect on the evaluation results. A change to non-TCB software would require no re-evaluation. Most changes fell between those extremes. Informed consideration of the changes, in the context of the security requirements and the previous evaluation, was required to determine the extent and nature of the re-evaluation effort.

TPEP dealt with maintaining evaluation ratings through the "RAtings Maintenance Program" (RAMP). A vendor employee (known as a Vendor Security Analyst or VSA) was vetted to monitor product changes in-house, with the ability to recognize and report both significant and minor security-related changes. The VSA periodically presented the changes to an NSA review board, which reviewed the RAMP materials. Depending on the review, NSA might accept the VSA's attestations as to the impact of the change and the testing performed, or could require additional government review.

When RAMP was under discussion, no one proposed having a vendor start over. The goal of TPEP was to put actually-secure systems into the marketplace and into the field. The evaluation process served that goal but did not supplant it. Since high-security systems are expensive to evaluate (tens of millions of dollars for an A1 system in the 1980's)[24] and take a long time to evaluate (ten years from initiation to final evaluation), the appropriate focus for RAMP was the potential for new security problems, in the context of the previous evaluation.

[24] NSA's figures for Gemsos's A1 rating were $14M for development and $50M for evaluation [15].

C.5 Current Product Evaluation and Validation

With the transition from the *TCSEC* and TPEP to the *Common Criteria* [16] and NIAP,[25] NSA's mission stayed the same, but the details changed. Currently, responsibility for evaluating secure commercial products is split between commercial evaluation labs and NSA. The labs have responsibility for systems aspiring to low security (with assurance at EAL4 or below). NSA retains overall responsibility for medium- and high-security commercial products in the United States, at levels EAL5 through EAL7.

C.6 Current Policy on Information Assurance

NSA's direction under current policy is substantially the same as it was originally.

1. "Establish and maintain technical standards and criteria for the evaluation of trusted computer systems . . . ."

   Under DoDD 8500.1, NSA is the DoD lead for NIAP, which includes responsibility for technical standards and criteria.

2. "Conduct evaluations of selected industry and government-developed trusted computer systems against these criteria."

   NSA validates evaluations under the *Common Criteria* against Protection Profiles and Security Targets.

3. "Maintain and publish an EPL of the selected industry and government-developed trusted computer systems that is suitable for use by the DoD Components."

   NSA maintains the Validated Products List (VPL), the modern equivalent of the EPL.

D Business Environment for High Security

D.1 Environment before TPEP

The problem that Steve Walker's Computer Security Initiative addressed was the failure of both methods of obtaining secure computer systems for DoD use — (a) using the security provided in commercial products and (b) crafting government-specific products, either as special versions of commercial products or as government-produced systems.

Commercial products did not provide any useful security. The "Tiger Teams" of the late 1960's demonstrated that attacking commercial systems always worked: "It is a commentary on contemporary systems that none of the known tiger team efforts has failed to date" [18]. Moreover, representatives of IBM, provider of 95% of the computers sold in the world, loudly and frequently asserted that technical security was bunk and that the only security required was personnel security.

Attempts to produce computers that met the DoD's needs for security had also proved ineffective. Paying a computer vendor to produce a special government-only version was a temporary fix. Fast-burner programmers wanted to work on new products, ones that would make their reputations [19]. Moreover, government-specific product versions remain available only as long as government funding continues. To keep up with new commercial releases, more government funds are required to update the government version. Internal government systems fared no better. Government does not employ system designers and implementors in the same numbers and at the same level of quality as do commercial companies. In addition, product support is not a natural government function and government does it badly.

One intention of the initiative was to change the environment for the conceptualization, design, and implementation of commercial computer systems. This was to be achieved by creating a standard metric for specifying security desired by the DoD and by promulgating policy that required the use of products that had measured up to the metric.

Before the initiative, government had scoped out the technical issues in computer security through full or partial funding of all the exemplar secure systems: Kernelized Secure Operating System (KSOS) [20], the Provably Secure Operating System (PSOS) [21], the Kernelized Virtual Machine (KVM) [22], and Multics [23]. These archetypes paved the way for business environment changes by demonstrating the technical feasibility of high-security systems.

D.2 Environment under TPEP

With the *TCSEC*, the TPEP, the EPL, and prescriptive DoD policies, all the levers to alter the business environment for secure systems were in place. Multics and SCOMP (KSOS-6) were commercial products and they moved forward into formal product evaluation. DoD policy was issued, requiring all system acquisitions to have security requirements, phrased in terms of the *TCSEC* classes. Since policies are only effective if they are enforced, IBM's disqualification from NSA's Minstrel acquisition made the point. IBM began to bid other companies' products and all players took the policy seriously. As a result, the dynamics of business decisions changed where security was concerned.

Though profitable, Honeywell's B2 Multics stayed in the marketplace only till its cancellation in 1985. On the other hand, Honeywell continued to invest in SCOMP, moving from the original custom hardware through Level 6 Honeywell hardware, Intel 486/386 combinations, and Pentium, finally to Zeon. The software was continuously maintained and extended to include TCP/IP and middleware for the de-

[25] The National Information Assurance Partnership is a "joint initiative between the NSA and the National Institute of Standards and Technology responsible for security testing needs of both IT consumers and producers and promoting the development of technically sound security requirements for IT products and systems and appropriate measures for evaluating those products and systems" [17].

velopment of guard applications, all with internal funds.

Gemini Computers, Boeing, Verdix, IBM Federal, Trusted Information Systems, and Digital Equipment undertook new medium- to high-security products without government funding, based on their calculation that the products would have a market. Amdahl decided to undertake revisions of its Multiple Domain Feature (MDF) [24] sufficient to allow a product evaluation for the same reasons, hiring Trusted Information Systems to assist in the effort. IBM had developed PR/SM in competition with MDF. Later, it also pursued product evaluation [25].

NSA's actions in the 1980's, coupled with DoD and National policy, created an environment where secure products were viewed by the marketplace to confer competitive advantage in the ability to get government business. The marketplace responded with its own medium- to high-security products.

D.3 Environment in Transition

In the early 1990's, an update to the *TCSEC* was begun. That effort went through several stages, resulting in harmonization of many nations' individual criteria into the *Common Criteria*. The result emphasizes product characterization over engineering methods and techniques proven through implementation and disputation. This untested revision to the community's security metric was a first ominous change.

A second indicator was the 1994 declaration by the Director of Central Intelligence and the Deputy Secretary of Defense that "Intelink [is] the strategic direction for Community product dissemination systems" [26]. Intelink used Internet technology to publish and read intelligence products. It joined independent agencies that lived under sometimes different policy regimes using low-security servers.[26] Not surprisingly, a prototype initially focused on allowing intelligence analysts to reference open resources such as Reuters and to exchange intelligence products was soon required to sanitize information and share it with the warfighter. In 1996, ASD/C4I asserted that "[e]fforts to provide more intelligence products for the warfighter at the collateral level must be a top priority" [28]. The security context for Intelink had changed dramatically. By then-existing (and current) DoD policy, those Intelink connections required high-security platforms. Low-security platforms, at best B1/EAL4, were put between intelligence and collateral networks. The prototyping and expansion of Intelink was a bad example in that the DoD was connecting separate security domains with low-security servers when high-security servers were both available and suitable for the task.

Then in 1995, the Multilevel Information Systems Security Initiative (MISSI) was announced. Its stated intention was to

> "be the security cornerstone for the defense information infrastructure. It will provide a multilevel security capability for networked automated information systems while ensuring that users can access only that information they are authorized, information is protected from unauthorized modification and users are identified and authenticated. MISSI will give us a single, integrated, consistent security infrastructure for all our needs: e-mail, electronic data interchange, electronic commerce, intelligence, and command and control, to include our business systems" [29].

Its "original goal was to provide a set of products and an architectural framework that would facilitate the development of multilevel secure NISs [Networked Information Systems]" [30].

Its effect was to kill the high-security marketplace.

MISSI was undertaken when the market boasted three evaluated A1 products, two B2 products, one promising A1 candidate, and two promising B2-like candidates. MISSI eventually reduced its goals to such an extent that its hallmark was the term "managed risk," a laudable-sounding concept that "has been used to justify use of low- or medium-assurance components to secure classified data (especially at the SECRET level) without much analysis of the threat or evaluation of the adequacy of the offered countermeasures" [31]. A1 vendors saw their business prospects collapse, virtually overnight. Boeing's Dan Schnackenberg believed that NSA through MISSI put all the Class A1 vendors out of business [32].

MISSI used the failed approach of funding government-specific products.[27] It also returned to an over-reliance on cryptography (Fortezza) over computer security.[28] It implied a quick path to accreditation: NSA was the official source of high-grade crypto *and* the executive agent for computer and information security. The promises of MISSI were never fulfilled and the changed environment created by the success of the Computer Security Initiative was destroyed. Those who really needed high security were the biggest losers when MISSI reneged and the high-security marketplace dwindled almost to nothing.

During the transition, the DoD and NSA in particular championed connections between different security domains, passing over available high-security products in favor of low-security ones. NSA then directly competed with high-security vendors with the introduction of MISSI, making promises that were not kept. Both by example and by explicit competition, the nurturing environment that grew out of the Computer Security Initiative was ruined.

[26] The required web statistics package Wusage 5.0 was ported to Solaris 2.4, SunOS 4.1.x, DEC's DigitalUnix 3.2, IBM's AIX, and Windows NT. AIX was B1 and Windows NT was C2. The others were unrated [27].

[27] Secure Computing Corporation's LOCK for the Secure Network Server and Trusted Mach for desktops.

[28] Computer security and cryptographic means supplement each other. Neither is a good substitute for the other. The best synergy results when they are used cooperatively and each is used to its strength.

D.4 Environment Now

Today, there are three surviving medium- to high-security products, Aesec's Gemsos, Boeing's Secure LAN, and BAE Systems's XTS-400. They no longer have to compete with MISSI, but they do have to contend with the current NSA competition: SELinux, NetTop, NSA's separation kernel, and the High Assurance Platform.[29] In a sense, the situation is the same as it was in 1996. The positive business environment for high-security products and systems is gone and the player to beat is NSA. Experience shows that today's promises should be viewed with caution.

E Performance under NIAP

Since the transition to the *Common Criteria*, NSA has had three responsibilities for trusted systems: (a) nurturing existing high-security products, (b) evaluating new medium- to high-security products at assurance levels from EAL5 through EAL7, and (c) participating in the Common Criteria working groups for the criteria and for processes and standards. Only the last task has received substantive attention.

E.1 Nurturing Evaluated Systems

At this writing, there are only three product lines with *TCSEC* ratings of B3 or A1: Aesec's Gemsos (A1), Boeing's Secure LAN (A1), and BAE Systems's XTS-400 line (B3, a descendant of the A1 SCOMP).[30] The CCEVS organization[31] at NSA is responsible for this activity, but its current position on RAMP is that each system would have to re-establish its security *bona fides* by a complete re-evaluation under the Common Criteria.[32] This position flies in the face of common knowledge at the time that RAMP was established: only new security problems need be addressed. Insistence on procedure over substance does not serve the public interest. Since the Department of Defense now requires Common Criteria evaluations on products acquired, this lack of action on NSA's part has left the United States without the beneficial use of the best computer security products that have ever been produced.

E.2 Evaluating New Systems

Since the establishment of the National Information Assurance Partnership (NIAP) in 1997, there have been 127 systems evaluated. Of those, only 3 were in the EAL5 to EAL7 range, and only one was against a validated U. S. Government Protection Profile.[33] One Tenix system (the Data Diode) was evaluated at EAL7 against a product-specific security target. A companion product from Tenix (the Interactive Link) was evaluated at the same time as EAL5 augmented, also against a product-specific security target. BAE Systems's XTS-400/Stop 6.1.E was evaluated at EAL5 augmented. In short, no new general purpose operating system[34] has been evaluated and listed since NIAP began.[35]

E.3 Encouraging New Systems

NSA has not encouraged new commercial, high-security systems. Vendors have not spontaneously produced high-security systems. This is not a surprise, since as I discussed in my previous paper, market forces have never spontaneously produced high- or medium-security systems. NSA is more culpable than other players, since it has the responsibility to encourage new high-security technology and has not discharged its duties well.

E.4 Common Criteria

Under NIAP, an international group is responsible for maintaining the *Common Criteria* itself and for identifying and resolving related problems. NSA and the National Institute of Science and Technology are the official United States representatives to this group and have executed their duties responsibly, with the possible exception of a new section on "composition" in the latest draft.[36]

F SELinux

"Security-Enhanced Linux" (SELinux) was begun to demonstrate that labels and mandatory access control *could* be added to Linux code [35]. Later, type enforcement was added and the original purpose as an exemplar of one security feature (labels) was replaced with more general trusted operating system goals. Recent versions have a wide variety of mechanisms and settings, to the extent that an expert in SELinux internals is sometimes required to make changes to the configuration [36]. According to NSA's summary web page on SELinux,

> 12. Is Security-enhanced Linux a Trusted Operating System?
>
> No. The phrase "Trusted Operating System" generally refers to an operating system that provides sufficient support for multilevel security and evidence

[29] See appendices F, G, H, and I, respectively, for more details.

[30] Aesec, Boeing and BAE Systems are the current owners of the listed products. Only Boeing was the original recipient of the EPL rating.

[31] Common Criteria Evaluation and Validation Scheme.

[32] NSA's position on previously evaluated products was assembled from private comments by vendors, private comments by NSA representatives, and public statements by NSA representatives at the 2005 ACSAC conference.

[33] These figures are from [33].

[34] Nor M-component à la the *Trusted Network Interpretation* (*TNI*).

[35] IBM's evaluations of PR/SM were all performed in Germany [34].

[36] A quick reading of the draft text indicates that it stands in direct contradiction to my conclusions about composition requiring engineering constraints to succeed. If my reading is accurate, the draft text should be revised.

> of correctness to meet a particular set of government requirements. Security-enhanced Linux incorporates useful ideas from these systems but focuses upon mandatory access controls. This work is being combined with other efforts (e.g., auditing and documentation) to construct a "trusted" system that can be evaluated. The initial focus of Security-enhanced Linux development has been to create useful functionality that delivers tangible protection benefits in a wide range of real-world environments in order to demonstrate the technology [37].

SELinux is not listed on the Validated Products List, although Red Hat's Enterprise Linux is listed at EAL4. SELinux is nevertheless being touted and advocated for widespread adoption. Advocacy of SELinux by NSA staff directly competes with medium- and high-security products that can provide the same features and far greater assurance than does SELinux.

G NetTop

NSA's NetTop [38] builds on SELinux. It is advocated as a high-assurance solution to replace multiple workstations on the desktop. Several vendors have NSA certificates and describe their NetTop products as EAL4, but no NetTop product is listed on the VPL.

Since NetTop attaches to networks of different security levels, DoDI 8500.2 would require it to be a "controlled interface," as "addressed in separate guidance" [39].

NetTop's reliance on SELinux's low level of security limits its assurance aspirations to EAL4, less than the requirement for a strong connection between different security domains.

The NSA imprimatur on weak-security NetTop products directly discourages the use of high-security workstations or thin clients attaching to domains at different security levels, a situation that DoDI 8500.2 says requires high security.

H Separation Kernel

Since 1996, NSA staff have aggressively pressed an updated version of separation kernel under various rubrics — separation kernel, partitioning kernel and "Multiple Independent Levels of Security" or MILS.[37] A search of the open literature shows that little was published before 2002. Both the open and informal literature are replete with emphatic assertions but no justification, neither for the claims of prior failure nor for the optimism that This Time It Will Be Different. Add to that numerous factual errors in exposition and one can only wonder why NSA feels the topic is mature enough for codification as an official U. S. Government Protection Profile.[38]

The available and restricted literature do not provide a clear and concise definition of "separation kernel." It appears to be a small variation on Amdahl's MDF [44] and John Rushby's separation kernel [45]. Unlike those efforts, however, NSA's separation kernel has been tested neither in the marketplace (like MDF) nor in the marketplace of ideas (like Rushby). Assertions that high security can be achieved without trust (in the *TCSEC* sense) have been strongly rebutted. Paul Karger's thesis, similar to Rushby's separation kernel work but less widely known, makes the point clearly that multilevel components are required to realize the concept.[39]

Advocates of NSA's separation kernel assert that multilevel security is not needed; data isolation, control of information flow, periods processing and fault isolation suffice.[40] This position is undermined by [48] which points out that since all boolean lattices are isomorphic (up to the number of atoms), all boolean policies are similarly isomorphic. The traditional classification/category policy is a boolean policy. Moreover, any policy that can be described using uninterpreted symbols and *AND*, *OR*, and *NOT* is also a boolean policy. Isolation can be so described. Isolation is a boolean policy. Thus, isolation is isomorphic to traditional non-discretionary (mandatory) access control. In summary, isolation is not different from nor simpler than multilevel security.

NSA's separation kernel and the related MILS have gotten far ahead of themselves. There is no clear and concise definition. There is little open literature on it. What literature exists is too much hortatory and not enough analytical. It does not build on community archetypes. It has not been subjected to critical peer review. In spite of these hurdles unjumped, setting it in concrete – whatever "it" is – is eighteen months along.

Success in getting adherents on this bandwagon cannot have rested solely on its technical basis.[41] It has probably benefited from hard problems in avionics and real-time operating systems that the separation kernel, if successful, would solve. It has also benefited from NSA's position and reputation.

Similar promises about separation kernels in the past have come to little, even with the efforts of such as John Rushby. It would be best for NSA and for users of the separation kernel to get all the ducks in a row *before* institu-

[37] In [40] published in 2004, the work is described as on-going for ten years. A 2005 paper from the University of Idaho [41] references "High Assurance Architecture via Separation Kernel" [42], an informal communication dated 1996.

[38] According to the NIAP website, the "U. S. Government Protection Profile for Separation Kernels in Environments Requiring High Robustness" [43] is currently "in development" as a U. S. Government validated Protection Profile.

[39] "...the network communications processors are multilevel secure ...." [46].

[40] See [47] for example.

[41] It is worth noting that enthusiasm for the separation kernel is not universal in the embedded and real-time communities. Pointed criticism of the draft "Partitioning Kernel Protection Profile" includes the following: (1) "PKPR ignores real-time"; (2) "PKPR passes over most of the Common Criteria functional requirements"; (3) "PKPR reliance on a simple use of memory protection hardware mechanisms rules out most of the RTOS market"; and (4) "PKPR requires that validating each component provides for compositional correctness." Regarding (4), "This is simply incorrect in any but the simplest systems" [49].

tionalizing it as a protection profile and *before* widespread advocacy for its adoption.

Adherents believe this separation kernel to be a great new hope for security. Let them document their case in a form suitable for publication. Let advocates and critics have a debate on the issues. After a trial of fire, the separation kernel will emerge hardened and improved; or its flaws will be manifest.

I High Assurance Platform

According to its flyer, "[t]he High Assurance Platform (HAP) Program is a National Security Agency (NSA) initiative whose main objective is to develop the technologies and standards that provide a highly assured computing environment for applications" [50]. The primary capabilities are to provide access to multiple domains from a single workstation and to allow secure data movement between domains. According to DoDI 8500.2, this kind of connection requires a controlled interface [51].

Although HAP focuses on workstations and servers, its *purpose* is to enable connections of these platforms to multiple security domains and to transmit data between them. Their approach is multilevel hypervisors, a topic of some subtlety [52]. In essence, HAP intends to insert high-assurance code (the assured information sharing service) between the hypervisor (untrusted) and applications (untrusted) [53]. This kind of appliqué, inserting a security layer between untrusted parts, has never worked and certainly is not the basis of actually high security. Experience has shown that evolving COTS C1/D class materials into a PL5-accreditable system is an impossibility.

There are at least two problems with the current HAP program. One is the lack of appreciation of unconstrained composition.[42] HAP needs global network conditions and a network worldview as a framework into which individual platforms can fit.

The second difficulty is the rejection of existing high-security systems as solutions to this problem.[43] This is particularly troublesome, since normal development and deployment cycles mean that current HAP efforts will not be fielded for at least five – and more likely ten – years.

Lacking a *TNI*-like perspective, HAP still believes that it can solve unconstrained composition when no one else has succeeded. I believe it may be impossible. The laudable goals of HAP are undercut by the refusal to use proven technologies in the short term, preferring untested technologies only available five to ten years from now. Its approach conflicts with DoD policy about connections between security levels.

[42] Two A1 systems *could* be combined to make a A1 combination. Improper combination would make the result less than A1. Artful failure of requirements could result in any *TCSEC* class between D to A1. Composition without global engineering constraints may not be possible. See [54] .

[43] At the 14 August, 2006 HAP Vendors Forum, HAP management made it clear in its opening remarks that references to or advocacy for existing high-security products were not welcome and that such products would not be considered within HAP.

J Multiple Domain Dissemination System

The Multiple Domain Dissemination System (MDDS) is a multilevel server that provides cross-domain interactions via the user's web browser [55]. Its principal component is a product called WebShield. WebShield is hosted on Trusted Solaris, an EAL4 operating system. WebShield is designed to attach to two or more networks simultaneously and to pass `http` requests made in one domain to other domains, returning the response to the originator. Searches can be cloned to all relevant security domains and the results collated and displayed.

Deployment of WebShield between networks of different classifications would be disallowed by current DoD policy [56]. EAL4 systems are vulnerable to Cuckoo's Egg attacks and two-card loader subversion. They are not strong enough for use in MDDS. MDDS's deployment puts at risk all networks to which it connects.

K DTW

According to the Rome Lab web site, the DoDIIS Trusted Workstation is

> "a certified multi-level security (MLS) information system utilizing both commercial and government off-the-shelf technologies. Composed of server hardware running a trusted operating system and ultra thin-client devices at each user's desktop, the DTW provides simultaneous access to multiple security domains from a single workstation. DTW additionally provides a secure means of transferring data between security domains" [57].

The "trusted operating system" mentioned is the EAL4 Trusted Solaris. Current DoD policy would disallow DTW's connection of its Citrix servers to multiple security domains [58]. Connecting DTW's weak-security servers to two or more security domains is technically weak and introduces severe vulnerabilities, not only for direct users but also for users of the attached networks.

References

[1] D. E. Bell, "Looking Back at the Bell-La Padula Model", *Proc.* ACSAC, 7–9 December 2005, 337–351.

[2] DoD Instruction 8500.2, "Information Assurance (IA) Implementation," February 6, 2003, ¶E3.2.4, p. 32.

[3] *Ibid.*, ¶5.6.6, p. 5.

[4] U. S. Government Protection Profile for Separation Kernels in Environments Requiring High Robustness," version 0.621, NSA/IAD, 1 July 2004.

[5] DoD Directive 8500.1, "Information Assurance (IA)," October 24, 2002, p. 22.

[6] "Information Assurance Technical Framework (IATF)," Release 3.1, September, 2002, ¶6.7.2, ¶6.7.3, and ¶6.7.4, pp. 6.7-23.

[7] Bell, *op. cit.*, p. 312.

[8] C. Stoll. **The Cuckoo's Egg**. Doubleday: New York, NY, 1989, p. 29.

[9] P. A. Karger, R. R. Schell, "Multics Security Evaluation: Vulnerability Analysis," ESD–TR–74–193, Vol. II, ESD/AFSC, Hanscom AFB, MA, June 1974.

[10] C. E. Irvine, "Considering Lifecycle Subversion," OSD Invitational MLS Workshop, Alexandria, VA, 24 September, 2003.

[11] S. T. Walker, "The Advent of Trusted Computer Operating Systems," *Proc.* National Computer Conference, May, 1980, 655–665.

[12] Department of Defense Trusted Computer System Evaluation Criteria, CSC-STD-001-83, 15 August 1983.

[13] Department of Defense Trusted Computer System Evaluation Criteria, DOD 5200.28-STD, December 1985.

[14] Department of Defense Directive 5215.1, "Computer Security Evaluation Center," October 25, 1982, change 1, November 16, 1994.

[15] R. R. Schell, "High Assurance MLS Systems through Proven Technology," AFCEA MLS Panel, Omaha, NB, 24 May 2005.

[16] *Common Criteria for Information Technology Security Evaluation*, Version 3.1, CCMB-2006-09-001, September, 2006.

[17] DoD Directive 8500.1, *op. cit.*

[18] J. P. Anderson, "Computer Security Technology Planning Study," ESD–TR–73–51, Vol. I, AD–758 206, ESD/AFSC, Hanscom AFB, MA, October 1972, p. 4.

[19] T. Kidder, **Soul of a New Machine**. Little, Brown: Boston, 1981.

[20] E. J. McCauley and P. J. Drongowski, "KSOS—The design of a secure operating system," *Proc.* AFIPS 1979 NCC **48**, 345–353.

[21] P. G. Neumann, R. S. Boyer, R. J. Feiertag, K. N. Levitt, and L. Robinson, "A provably secure operating system: The system, its applications, and proofs," Technical Report CSL–116, SRI International, 1980.

[22] M. Schaefer, R. R. Linde, *et al.*, "Program Confinement in KVM/370," *Proc.* ACM National Conference, Seattle, October, 1977.

[23] J. Saltzer, "Protection and the Control of Information in Multics," *Comm. ACM* **17**(7), July 1974, 388–402.

[24] R. W. Doran, "Amdahl Multiple-Domain Architecture," *Computer* **21**(10), October, 1988, 20–28.

[25] P. A. Karger, private communication, 2006.

[26] R. J. Woolsey (DCI), J. M. Deutch, (DepSecDef), Memorandum for Co-Chairmen, Intelligence Systems Board, Subject: Intelink, 11 August 1994.

[27] F. T. Brown. **Top Secret Intranet.** Prentice Hall: New Jersey, 1998.

[28] E. Paige Jr., "The Rapid Expansion of Intelink," **Defense Issues**, **11**(66), June 11, 1996.

[29] E. Paige Jr., "From the Cold War to the Global Information Age," **Defense Issues**, **10**(34), February 27, 1995.

[30] F. Schneider, *ed.*, **Trust in Cyberspace.** National Academy Press: Washington, DC, 1998, box 4.4.

[31] *Ibid.*, p. 109.

[32] D. Schnackenberg, OSD Invitational MLS Workshop, Alexandria, VA, 24 September, 2003.

[33] A. Dale, "National Information Assurance Partnership (NIAP) Common Criteria Evaluation and Validation Scheme (CCEVS)," Briefing to IAPB, 23 March 2006.

[34] "PR/SM^{TM} LPAR for the IBM System z9^{TM} Enterprise Class and the IBM System z9^{TM} Business Class from International Business Machine Corporation (IBM)," Bundesamt für Sicherheit in der Informationstechnik, BSI-DSZ-CC-0378-2006, September, 2006.

[35] NSA's SELinux FAQ, NSA web site, 2006.

[36] "An introduction to SELinux," LWN.net, Article 103203, 2006, with comments.

[37] NSA's SELinux FAQ, *op. cit.*

[38] NetTop Technology Profile Fact Sheet, NSA web page, 2006.

[39] DoDI 8500.2, *op. cit.*, E4.A4, Enclave and Computing Environment, p. 85.

[40] R. Beckwith, W. M. Vanfleet, L. MacLaren, "High Assurance Security/Safety for Deeply Embedded, Real-Time Systems," Embedded Systems Conference, 2004.

[41] J. Alves-Foss, C. Taylor, P. Orman, "A Multi-layered Approach to Security in High Assurance Systems," *Proc.* 37th Hawaii International Conf. on System Sciences, 2004.

[42] P. White, M. Vanfleet, C. Dailey, "High Assurance Architecture via Separation Kernel," internal communications, 1996.

[43] NSA/IAD, "U. S. Government Protection Profiles for Separation Kernels in Environments Requiring High Robustness," version 0.621, NSA, Ft. Meade, MD, 1 July 2004.

[44] Doran, *op. cit.*

[45] J. Rushby, "A trusted computing base for embedded systems," *Proc.*, 7th DoD/NBS Computer Security Initiative Conference, Gaithersburg, MD, September 1984, 294–311.

[46] P. A. Karger, "Non-Discretionary Access Control for Decentralized Computing Systems," M.I.T. Laboratory for Computer Science, MIT/LCS/TR-179, May 1977, p. 54.

[47] W. M. Vanfleet *et al.*, "MILS: Architecture for High-Assurance Embedded Computing," STCS CrossTalk, August, 2005.

[48] D. E. Bell, "Lattices, Policies, and Implementations," *Proc.* 13th National Computer Security Conference, Washington, DC, 1–4 October 1990, 165–171.

[49] V. Yodaiken, "Reponse to 'Partitioning Kernel Protection Profile,' " unpublished draft. `[http://www.yodaiken.com/papers/securityx.pdf]`

[50] HAP Flyer, August, 2006.

[51] DoDI 8500.2, *op. cit.*

[52] P. A. Karger, "Multi-Level Security Requirements for Hypervisors," *Proc.* ACSAC, 7–9 December 2005, 267–275.

[53] HAP flyer, *op. cit.*

[54] Bell, "Looking Back at the Bell-La Padula Model," *op. cit.,* ¶6.2.

[55] W. Neugent, "Assured Information Sharing," *The Edge,* The MITRE Corporation, Fall 2005, **9**(2), 12.

[56] DoDI 8500.2, *op. cit.*

[57] Rome Lab web site: `www.rl.af.mil/tech/programs/jedi/`

[58] DoDI 8500.2, *op. cit.*