IT Security History & Architecture

(How Did We Get Into This Mess?)

by

Dr. Steve G. Belovich

CEO, IQware, Inc.
330-659-6300
www.IQware.us
Steve.Belovich@IQware.us

5/18/10

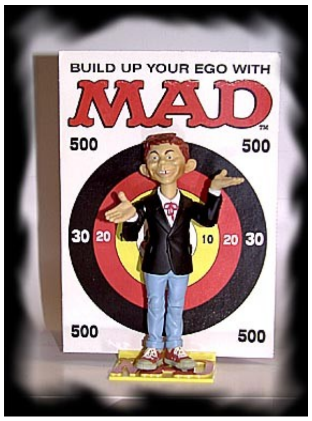
1.0 IT Security Overview

1.1 Recent Security Issues

The past year has witnessed an amazing number of articles, reports, seminars and news stories about successful hacking attempts and the lack of data and/or network security. The GAO recently reported that:

> "Despite indications that agencies have improved their compliance with parts of the Federal Information Security Management Act (FISMA), many major agencies still consider their information security controls a significant deficiency or material weakness" - GAO May 2009

Even "Mighty Google" is not immune and is a target, as this article shows:

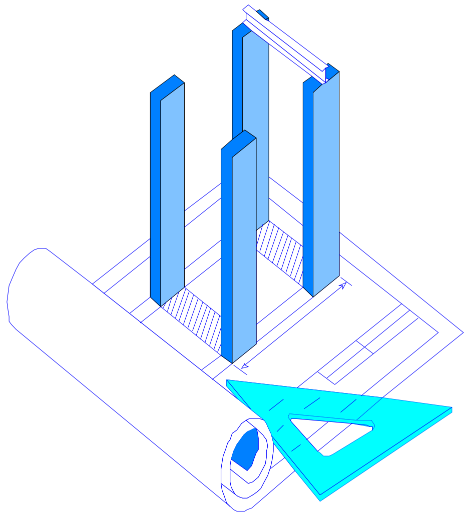


What, Me Worry?

> "Ever since Google disclosed in January (2010) that Internet intruders had stolen information from its computers, the exact nature and extent of the theft has been a closely guarded company secret. But a person with direct knowledge of the investigation now says that the losses included one of Google's crown jewels, a password system that controls access by millions of users worldwide to almost all of the company's Web services, including e-mail and business applications.
>
> The theft began with an instant message sent to a Google employee in China who was using Microsoft's Messenger program, according to the person with knowledge of the internal inquiry, who spoke on the condition that he not be identified. By clicking on a link and connecting to a "poisoned" Web site, the employee inadvertently permitted the intruders to gain access to his (or her) personal computer and then to the computers of a critical group of software developers at Google's headquarters in Mountain View, Calif. Ultimately, the intruders were able to gain control of a software repository used by the development team." - NY Times, April 19, 2010

A common thread that runs through these articles is the network. Invariably, the focus is always on intrusion detection and how the network is configured. Firewalls, anti-virus software and anti-spyware are front-and-center as are algorithms for identifying intrusions. Perimeter security has been the "accepted" defensive approach and the three main hacking steps of scanning, footprinting and enumeration are given little attention. Also ignored are the vulnerabilities that are built-in to the TCP/IP protocol itself which permits challenge and response without authentication.

Scant attention is paid to the serious vulnerabilities of software assets, e.g., the source code base, "make" files, module management systems, libraries, etc. due to poorly-planned access mechanisms and deployment. After all, hackers gained access to Google's code base through a web browser which, in retrospect, seems a huge oversight.

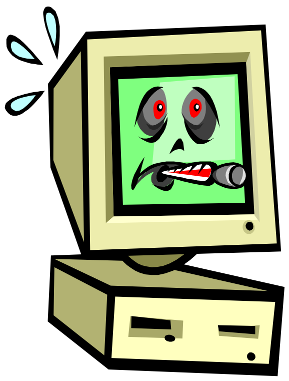


What is totally ignored in analyzing IT security issues is the fundamental engineering & architecture of the IT systems that were penetrated and what can - or should - be done about that. This is at the heart of the "security problem" and also represents a key rate-limiting step preventing much-needed advances in IT systems' operational capability.

Another problem is confusing IT system architecture and engineering with mere programming or coding. Architecture and engineering refers to how and where information is acquired, archived, analyzed, transformed, distributed and presented. **Architecture and engineering also govern how information is protected and how access is granted, controlled and monitored.** Coding is merely a vehicle for doing that.

Given all this media attention, what else could be said about data/network/software security that has not already been said? Well, actually, a lot. In the rush to point out the vulnerabilities, threats, exposures and liabilities scant attention has been given to the real fundamental issues: 1) what the "security problem" really is, 2) what caused it, 3) why you are not safe no matter how many "official NIST certifications" you may have and 4) what actions should you take.

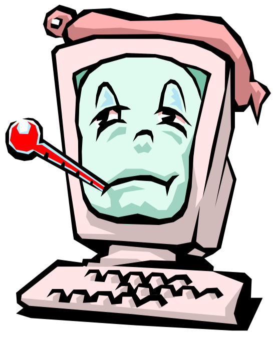
1.2 What's *Really* The "IT Security Problem"?



The "ITSP" (IT Security Problem) is a generic term for the problems that arise when trying to achieve a set of operation-related goals. There are six members of that set:

1. IT systems should do exactly what they are intended to do
2. IT systems should operate when intended to do so
3. IT systems should work on behalf of duly-authorized personnel
4. IT systems should NEVER do what is NOT intended
5. IT systems should NEVER operate when NOT intended
6. IT systems should NEVER work on behalf of NON-authorized personnel

The ITSP is much more than a mere "network protection" issue. That is a major reason why attempts to secure IT systems by merely securing the network via firewalls and intrusion detection schemes do not work very well – they only focus on the network and ignore everything else. It's also another reason why following every guideline and meeting various certifications will not make you invulnerable and may, in fact, leave you even more exposed because the root causes of IT system vulnerability are not addressed.



So, is this the real security problem? Not entirely. **The real security problem is bad software design, poor implementation and the mass deployment of critical tasks onto fundamentally non-secure platforms.** Note that this problem has little to do with specific "software tools" or network designs. Rather, it deals with the deeper issues of how we engineer software, the qualifications of who engineers the software and how we select and deploy IT systems and software to run our organizations.

1.3 Why Understand The History of IT Security?

Understanding the history of IT security explains why we are in this situation of seemingly never-ending vulnerability to data theft, cyber attacks and hacking. It also significantly impacts our "go-forward" options due to the inertia of installed base and existing infrastructures. Ignorance is always costly, so knowing the root cause(s) helps us prevent repeat errors and guides us to effective, long-term solutions. If the problem is understood, a solution is possible. Conversely, if the real problem is not understood, a real solution is impossible.

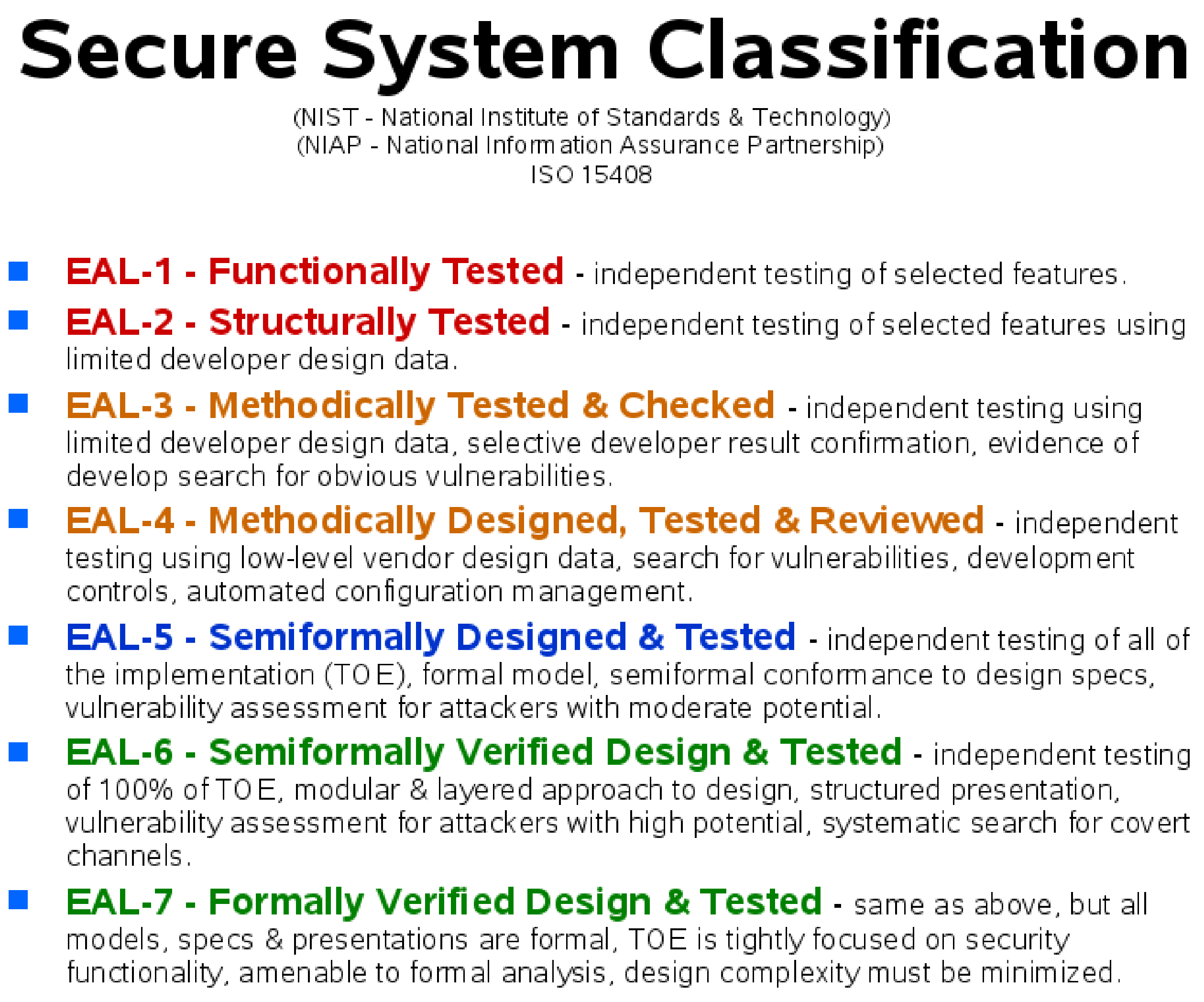
1.4 Security Standards and Certifications

Another big issue is certification. Because the need for security is so evident and the lack of security is so prevalent, various standards and certifications have arisen to "prove" certain levels of security. The DoD (Department of Defense), the NIST (National Institute of Standards and Technology), NIAP (National Information Assurance Partnership) and the ISO (International Standards Organization) have all issued and/or endorsed standards for system security. The two main standards are DoD 5200.28 (the so-called "Orange Book) and ISO 15408, shown below.

Secure System Classification

(DOD)

- **D - Minimal Protection**
- **C - Discretionary Protection**
  - **C1 - Discretionary security protection** - separates users & data, uses credible controls to enforce access limitations on an individual basis.
  - **C2 - Controlled Access Protection** - users individually accountable for their actions, security audit trail, resource isolation.
- **B - Mandatory Protection**
  - **B1 - Labeled Security Protection** - security policy model, keeps integrity of sensitivity labels, sensitivity labels must be held in all major system data structures, demonstration of reference monitor implementation.
  - **B2 - Structured Protection** - formal security policy model, discretionary & mandatory access control enforcement extended to all subjects & objects, separation of critical & non-critical system elements, stringent configuration management controls, covert channels are addressed, relatively resistant to penetration.
  - **B3 - Security Domains** - Reference monitor mediates all accesses of subjects to objects, be 100% tamperproof, TCB (trusted Computing Base) only contains security-relevant code & data structures, system engineered for minimal complexity, security-relevant events are signaled, system recovery is required, highly resistant to penetration.
- **A - Verified Protection**
  - **A1 - Verified Design** - Functionally same as B3, full mathematical verification of design.



Secure System Classification

(NIST - National Institute of Standards & Technology)
(NIAP - National Information Assurance Partnership)
ISO 15408

- **EAL-1 - Functionally Tested** - independent testing of selected features.
- **EAL-2 - Structurally Tested** - independent testing of selected features using limited developer design data.
- **EAL-3 - Methodically Tested & Checked** - independent testing using limited developer design data, selective developer result confirmation, evidence of develop search for obvious vulnerabilities.
- **EAL-4 - Methodically Designed, Tested & Reviewed** - independent testing using low-level vendor design data, search for vulnerabilities, development controls, automated configuration management.
- **EAL-5 - Semiformally Designed & Tested** - independent testing of all of the implementation (TOE), formal model, semiformal conformance to design specs, vulnerability assessment for attackers with moderate potential.
- **EAL-6 - Semiformally Verified Design & Tested** - independent testing of 100% of TOE, modular & layered approach to design, structured presentation, vulnerability assessment for attackers with high potential, systematic search for covert channels.
- **EAL-7 - Formally Verified Design & Tested** - same as above, but all models, specs & presentations are formal, TOE is tightly focused on security functionality, amenable to formal analysis, design complexity must be minimized.

1.5 The "Weak Spot" of Security Standards and Certifications

The problem with all of these standards and certifications is that they are imperfect and incomplete. The big flaw with the ISO 15408 standard is that it only focuses on the "TOE" (Target of Evaluation). What this means is that a system can have the highest rating and still be vulnerable, since the vulnerable aspect of the system was neither tested nor evaluated. This is not good.

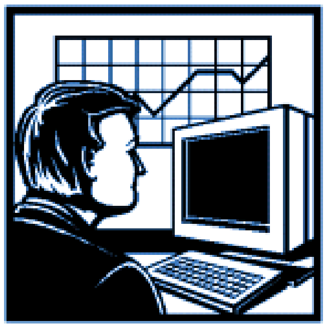
The DoD standard does mandate some system structure so there is a lot more confidence in using that standard. However, that standard is incomplete and difficult to apply. Further, few organizations are really skilled at application of these standards and there is a lot of politics in the process. **Consequently, such certifications are more for legal defense (e.g., for defense against negligence) than they are for actual cyber defense.**

2.0 A Very Quick History of Computer & O/S Technology

2.1 The 1950s

In the 1950s, IBM dominated the landscape and hardware ISAs (Instruction Set Architectures) were changing constantly which meant that the "operating software" (precursor to the operating system) was redesigned with each new machine. The concept of an operating system (O/S) was introduced and it was usually an "add-on" because the profits came from hardware sales. The O/S handled single-user and/or batch operations and provided very simple file systems and related file services. The need for security did not exist because there was no remote access and physical security of the building and computer hardware equated to IT system security. Physical access meant that you were authorized – simple and effective.

2.2 The 1960s and 1970s



In the 1960s and early 1970s, computing hardware was moving from "one-at-a-time" to automated production. Operating Systems (O/S) concepts were evolving and the concept of microprogramming was introduced by IBM in 1960. So-called "mini-computers" (e.g., PDP-8, PDP-12, and the PDP-11 series) were introduced by DEC in the mid 1960s and early 1970s. These machines fit into 19-inch racks, were air-cooled and could run on 208V (three-phase), 220V (bi-phase) or 120V single-phase. They used 7400 series TTL logic (SSI/MSI chips) and could be mass produced (I actually own a few of these machines which I repaired as a grad student over 25 years ago).

During this time, O/S technology made the leap from single-user/batch to multi-user and time-sharing. This leap – and it was a big one – meant that hardware & software mechanisms had to be invented to provide protection so that no user's program could "escape from its playpen" and interfere with the operation of other users' programs or the system itself. Thus the concept of security in the form of memory protection was introduced at both the hardware and software levels. The concept of memory management was also introduced which created "virtual memory" which could be allocated to different regions of physical memory "on demand".

2.3 Programmers Are Born

These concepts allowed the logical (and later the physical) segregation of programming from hardware design. People writing the instructions (the "code" or the "software") for the computer did not need to know exactly what the machine really did or how it really did it. Thus, "programmers" were created who were able to do their job without being hardware engineers and the field of "Computer Science" was born. This field originated out of electrical engineering and mathematics, but its modern incarnation has forgotten large portions of those disciplines.

Further, the relatively small installed base of machines allowed for a lot of experimentation so that good ideas could be brought to market and bad ideas were quickly buried. Multi-user protection mechanisms improved, memory management got "smart" and operating system services expanded greatly.

2.4 Compilers Get Smart Because Hardware is Smarter

Compilers also got smart, with improved optimization techniques that took advantage of the tremendous advances in hardware technology. Some of those hardware advances include multi-level set associative cache RAM, pipelined CPUs, instruction pre-fetching, score-boarding, "eager" branch execution, multi-port I/O and the migration of more functionality into the firmware and/or the hardware to free up the O/S from the details of disk management, etc.

In the late 1970s, the crippling limitation of address space was aggressively addressed by DEC and IBM when they expanded to 32bit (VAX architecture) and 44bits (ESA architecture) respectively.

Sixty-four-bit architectures came in 1992 with the introduction of the DEC Alpha 21064 microprocessor. Others followed including Intel and IBM. Compilers lagged but eventually caught up with new, larger data types including 64-bit integers, 128-bit floating point numbers and expanded virtual memory address space management.

2.5 Installed Base Grows – Creating Dependency

Meanwhile, the installed base of computers was exploding at a phenomenal rate. Further, businesses were becoming totally dependent upon these machines and became less tolerant of shutdowns for any reason – including new hardware installation and upgrades. Software applications were being written and there was little organization or thought given to what do we do in a year or two when we have to expand our capability? **No one really thought that one through because that was not budgeted – very scary.**

2.6 Installed Base Impedes Technological Advancement

What this meant was that expansion in fundamental computing technology (e.g., the introduction of newer and better Instruction Set Architectures) actually slowed down because the sheer size of the installed software and hardware base severely limited new experimentation and discovery. Although new fundamentally better hardware and software designs could be brought to market quickly, the market simply could not absorb them. Shutdowns for any reason became intolerable, whether it be for maintenance or a complete new machine and software applications.

So, the computer industry continually improved the hardware & software, but the Instruction Set Architectures (ISAs) were largely preserved. The sheer size of the installed software base also prevented rapid change, no matter how "good" that change might have been for the industry.

2.7 The Economics of A New ISA & O/S Does Not Compute

The economics of the installed based has prevented major innovations to ISAs during the past twenty-five years. This "slow down" was a business necessity to preserve existing operations for customers while still selling them new hardware and software. It's a real tough sell when you have to tell your customers to throw everything out and buy all new and different stuff - especially when you sold them the stuff that you're now telling them to toss out!

The ISA is essentially the interface between hardware and software and thus had to remain static for upward compatibility purposes. There has been little innovation in this critical area for twenty-five years. There are too many economic barriers preventing the creation and deployment of the key hardware components required for a secure system, such as support for multi-mode instruction set execution and duplicate register sets.

Any fundamentally new O/S would require the purchase and deployment of an entire set of new apps - which economically simply could not happen. To avoid alienating the customer base, changes had to be made slowly (if at all) to preserve existing architectures. That meant – and continues to mean - uncomfortable trade-offs between capability and what could be sold. Experimentation and invention in this critical area cannot proceed economically because there is too much already built on top of what is currently deployed. **In short, while some *evolution* is still happening, *revolution* has almost ceased.**

3.0 Secure System Requirements: The Reference Monitor

3.1 Requirements For A Secure System

In parallel with this explosion in hardware capability, security concepts also expanded, backed by lots of research effort. The net result of the research was that "finding and patching" security "holes" was not the way to go. Security "leaks" could not be plugged because there was no assurance that there would not be another undiscovered "leak". There was no way to be sure that all possible IT "doors" were properly "guarded".

Rather, security had to be designed in to an operating system (and into an IT system) from the ground up. Note that security had two main operational components: (1) multi-user protection and (2) preventing unauthorized accesses.

A lot of research was done in the 1960s to figure out how to deal with multi-user protection and preventing unauthorized system access. The results of this research revealed the necessary components of a secure, trustworthy system. These components are summarized below.

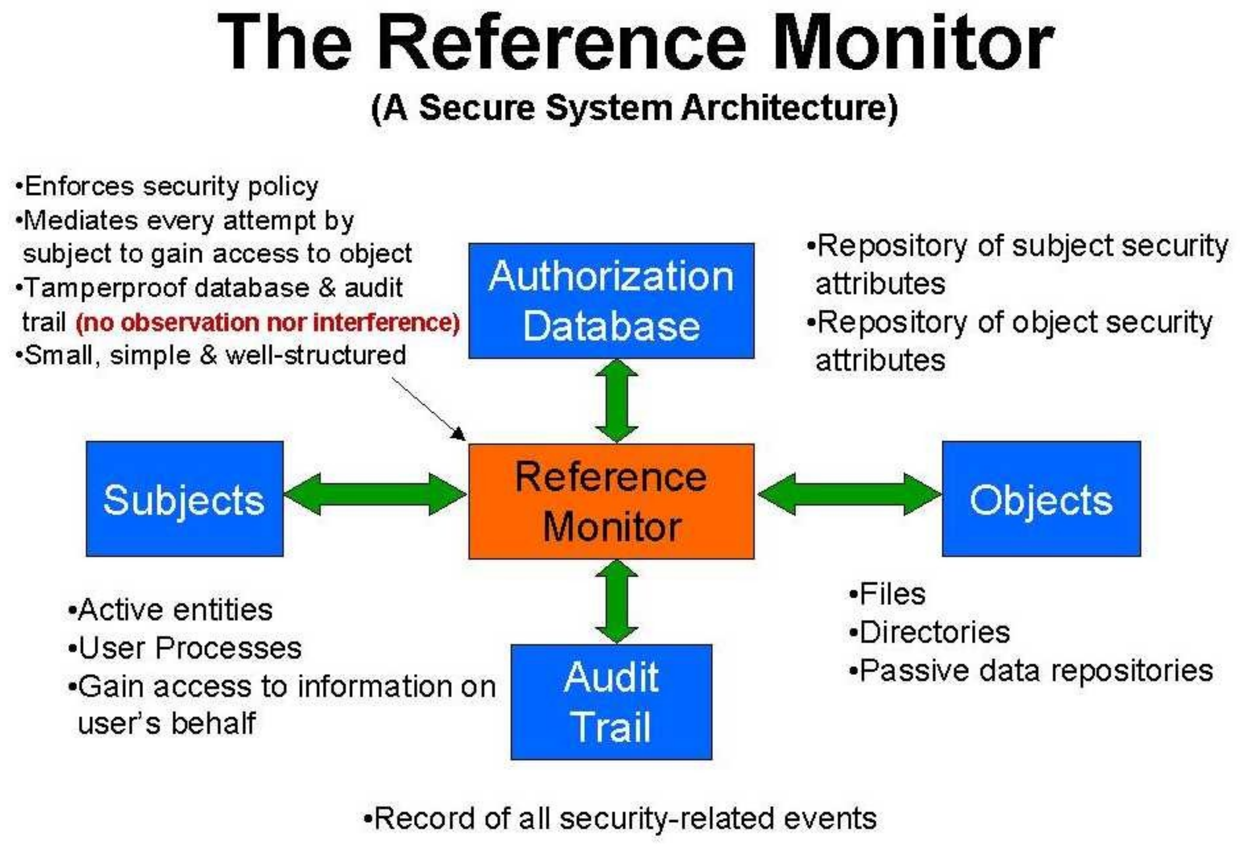
1) Policy
Security Policy - System must enforce a well-defined security policy.
Marking - System must associate all objects with access control labels (sensitivity & access modes).

2) Accountability
Identification - System must identify individuals and their various authorizations in a secure manner.
Audit Trail - System must keep & protect audit trail so actions may be traced to responsible party.

3) Assurance
Evaluation - System must have hardware/software mechanisms that can be independently evaluated to assure that policy & accountability are enforced.
Continuous Protection - System must continuously protect trusted mechanisms that enforce policy & accountability from tampering.

3.2 The Reference Monitor

As part of these requirements for a secure system, the "Reference Monitor" concept was introduced. This was a logical structure built into the lowest level of the Operating System (O/S) which adjudicated the access of any subject to any object. The Reference Monitor is shown in the diagram below:


The Reference Monitor
(A Secure System Architecture)
•Enforces security policy
•Mediates every attempt by subject to gain access to object
•Tamperproof database & audit trail (no observation nor interference)
•Small, simple & well-structured
Authorization Database
•Repository of subject security attributes
•Repository of object security attributes
Subjects
Reference Monitor
Objects
•Active entities
•User Processes
•Gain access to information on user's behalf
Audit Trail
•Files
•Directories
•Passive data repositories
•Record of all security-related events

The reference monitor mediates all accesses of objects by subjects. With properly defined subjects and objects, the reference monitor (RM) provides a trusted – and verifiable - security policy enforcement mechanism. The reference monitor, combined with the principles of a secure system architecture, can provide trustworthy, verifiable enforcement of a security policy.

3.3 Where The Reference Monitor Is (or Is Not) Used

Some operating systems incorporated this concept at the lowest layer, right above the FLIH (first level interrupt handler). Most did not. The reason was the installed base of existing systems prevented radical modifications to the underlying software structures. **Adding the reference monitor would have been a very radical modification, requiring a new file system.** Simply put, the size and inertia of the installed base prevented the fundamental re-engineering and re-deployment required for a truly secure O/S.

Many institutions were very resistant to change because they had to operate 24-by-7 and could not simply shutdown, reinstall a new O/S, install new applications and reboot. So, although a solution to security was known and understood, simple economics prevented its widespread adoption.

4.0 The Desktop Revolution (how RAM & disks got really cheap)

4.1 Consumer Market Economics Limits Design Choices

While these advances were going on in the mainframe and mini world, the same thing was happening in the PC world – only far worse. The microprocessor (which first appeared in 1970 with the TI 4040) allowed for the cheap introduction of home computers (e.g., TRS-80, Apple in 1976 and the IBM PC in 1980).

Cost dominated the retail market - as it still does. The cheapest, simplest design is the one that wins. Security, performance, etc. are all secondary to cost. The O/S was made simple and dumb with bare minimum support for a file structure. In fact, QDOS (**Q**uick 'n' dirty **D**isk **O**perating **S**ystem) was the forerunner of DOS which led to Windows 3.0, Windows 3.11, NT 3.51, NT4.0, W2K, W-XP, etc. Of course, CP/M (**C**omputer **P**rogram/**M**onitor) was out there in the early 1980s and many concepts embodied within DOS were taken from CP/M.

Because early PCs had to be cheap, they were necessarily extremely limited in capability and were initially 8-bit microprocessor-based (the Motorola 6502 for Apple and the Intel 8080 for the TRS-80). So nearly all of the advances in mainframe and mini technologies, e.g., interleaved RAM, hierarchical storage, set associative cache RAM, memory management, multi-port I/O and multi-threaded applications were deliberately eliminated from the design of the PC. They had to be for cost considerations.

Applications (Apps) were also simple and dumb and carried with them whatever run-time functions were needed. Apps also had to handle their own memory management because the O/S did nothing there. Early desktop O/S's (e.g., early versions of DOS) did not even have a print spooler. When you told an IBM PC to print in 1983, then that's exactly what it did - and nothing else. You had no control over the machine except to type CTRL-C and abort the print job.

4.2 System Security Deliberately Eliminated in PCs

When the desktop operating system (O/S) was designed, all of the security concepts learned in the mainframe/mini world were tossed out because they were not required for the intended use of a home computer. The protection mechanism was physical: lock it up. In the early 1980s, there were metal frames made to hold early PCs and locks on the case to prevent physical access to the innards. Such mechanisms were easily defeated, but tampering would leave a physical trace.

<u>The essential features of multi-user support, multi-user protection and system security were deliberately eliminated from the early desktop O/S design.</u> They were not needed for early PCs which were designed for the home market where there would be one user at a time and security was not a concern. Price and convenience drove the design and it still does today. Why take up extra RAM, disk and CPU cycles when the market did not need it, did not want it and would not pay for it?

4.3 Early Security Focused on Anti-Piracy

In the 1980s, "software security" became a concern because of software piracy or illegal software duplication. The big "PC Security" issue was focused on preventing the user from doing something to the "outside (e.g., illegally copying an application). Now, PC security tries to stop the "outside" from doing something to the user!

On the application side, "dongles" or hardware keys were used to ensure that the application would only run on the intended hardware. Of course, such devices did nothing for data protection nor for ensuring that the application would run properly, but those vulnerabilities were not considered threats at that time. The main purpose of hardware keys was to thwart copying of software and its illegal use on different hardware.

Further, the technology of the time helped prevent the copying of applications because CD/DVD RW devices did not exist and jump drives were only a dream.

4.4 Dial-Up Networking

The concept of networking home computers was unanticipated in the late 1970s and early 1980s. Networking originally consisted of remote users on "dumb terminals" using modems to access a bigger machine. The Bell 103 modem standard was the first and then Hayes became the dominant player and the famous "AT" modem commands became the *de facto* modem control standard. This allowed control and data to be shared on the same link, a concept called "in-band signaling".

Later, PCs replaced the "dumb terminal" via terminal emulation programs but access was still limited to dial-up connections to a larger machine. In the late 1970s and early 1980s, hooking up two PCs via some sort of network had no apparent purpose. PCs still had single-task operating systems (e.g., DOS 2.3) and Windows 3.1 was still in the planning stages. Robust network protocols were still experimental, the "physical layer" of the OSI (Open Systems Interconnect) model was still evolving and no stable networking technology existed.

4.5 The Invention of the Internet

In parallel with this was the development of the Internet, which was really born out of Dr. Leonard Kleinrock's work at MIT in the early 1960s, along with DARPA (Defense Advanced Research Projects Agency). That work was started in the 1960s and grew throughout the 1970s. It was called DARPANET at first and later on the "D" was dropped and became ARPANET. General Electric was also involved with GE Information Services Network, as was TYMNET, which were terminal-oriented and supported both interactive and batch processing.

The main networking goal in the early days was simply getting it to work! Early protocols were simple and some complexity was added later on to prevent errors such as lockups, and other early "denial of service" situations that had a variety of causes, including a lack of reassembly buffers for lengthy messages. The concept of a "store and forward" network was explored and refined. This involved breaking up messages into manageable chunks called "packets", transmitting them over the network and then reassembling them at the receiving point.

4.6 Early Network Protocols Ignore Security

The engineering emphasis on those early network protocols was ease of connectivity, maximum utilization of expensive bandwidth and the reliability of the connection. Getting the entire network to operate correctly was the goal. All else was secondary. Security was not an issue and was largely ignored. The technologies used were intended to be convenient and easy-to-use so that hooking up to the network would be a quick and easy thing to do. The two main protocols that arose were TCP (Transmission Control Protocol) and IP (Internet Protocol). These were eventually merged and became what we now know as TCP/IP.

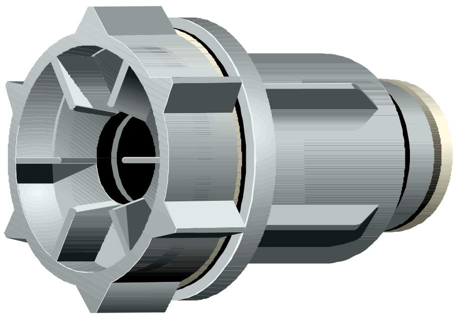
Security was not required for early networks because access was physically controlled. Also, the built-in access control mechanisms of the mainframe or central machine were well-established, well-understood and were the "guardians of the gate" to prevent unauthorized access. The network simply presented the access request to the mainframe and it had the responsibility of granting or preventing access. That worked fine for that time.

4.7 PC Operating Systems Had No Secure Foundation

While networking was improving, the initial O/S (Operating System) designs for PCs discarded or ignored the "mainframe/mini" concepts of shared resources, multi-user access, memory protection, multi-layer operation modes (e.g., kernel, executive, supervisor, user), user isolation, file-level access protection, ACLs (Access Control Lists), privileges, quotas, etc.

These engineering concepts were essential for a secure system because they allowed many users to share the computing resources (CPUs, RAM, disks, etc.) without interference. One user's mistakes did not percolate over into another user. The operating system handled the all the housekeeping and ensured that the entire computing system operated correctly even if an individual user did something stupid. The mainframe O/S was designed to detect and prevent that from happening and these technologies were maturing.

These critical engineering concepts were not included in the architecture of the home computer. Such technologies were very costly, memory-intensive, CPU-intensive and served no logical purpose for a home computer. More importantly, including these concepts in the O/S served no **economic** purpose for the home computer. **Price always drives the consumer market** and there was simply no demand for such features.



The problem is that such features really need to be engineered in from the beginning in order to work properly. Adding them afterward is nearly impossible - and it has not occurred yet in the PC market. We are now living with the consequences of that.

4.8 Networking PCs Requires A Secure O/S

Once networking was expanded to include local networks of personal computers, then access control for personal computers became an issue. It now mattered who could access which computer and when they could do that. It now mattered who could access what specific resource of what machine and when. Privileges (what you're allowed to do) and quotas (how much of something you're allowed to use) now became important.

So, the personal computer now needed a secure foundation and it just wasn't there. Usernames, passwords and some limited permission management were the best that could be done. However, such access control mechanisms were crude and easily defeated. There was no underlying security mechanism for the PC operating system and no easy way to add it either. The size of the installed base and the economics of the consumer market prevented the much-needed re-engineering of the desktop's operating system. Why bother to create it if you cannot sell it?

As an example, just adding proper “object marking” (a key requirement for a secure system) would require a brand new file system which would force the replacement of the entire installed base of PCs and software. **By way of a benchmark, that installed base is about a trillion dollars.**

4.9 The Fatal Flaw: Deploying Critical Stuff On A PC

Although the weaknesses of the desktop operating system were well-known early on, IT managers found the technology to be very attractive, convenient, easier to understand and cheaper. Mainframes and minis cost too much and required constant "care and feeding". Worse, maintenance contracts were expensive and it took too long for someone to arrive when something malfunctioned. After all, if it worked at home, it should be fine for the enterprise, right? "Scale-up" just seemed easy.

So, PCs migrated from the price-driven consumer market to the enterprise. Critical tasks such as accounting, payroll, invoicing and other operations were taken off of the mini-computer or mainframe and placed on the desktop. It was cheaper, more convenient and there were no monthly computer maintenance fees to pay. All in all, it looked like a smart move for business.

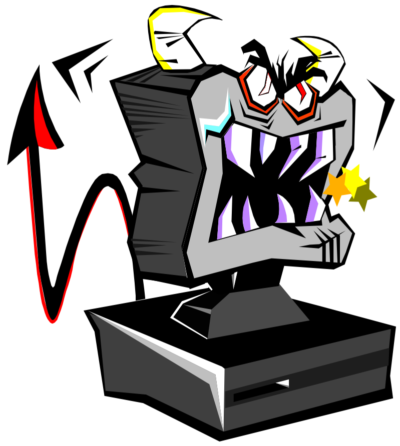
The problem with that move was that the requirements of a business are far different than those for an individual user. As business needs expanded, the demands on the desktop grew accordingly. Unfortunately, the desktop was not engineered to support those expanded demands. Rather, the desktop was engineered to meet the needs of the consumer market which wanted the machine for entertainment purposes, web surfing and social networking rather than for "traditional computing".

Multimedia, audio processing, video handling, interactive gaming and the like were all critical requirements for the consumer market but did little for the business market. In fact some of the multimedia features actually caused new problems in the O/S as new releases came out.

Although the business market cared about that, the consumer market did not. Sales-wise, consumer PC sales outnumber business PC sales by nearly 1000-to-1, so that part of the market controls everything else – and it still does today.

Clearly, business requires more secure systems but the marketplace is not providing it because it's listening to the consumer side. **Truly effective security is just not possible without fundamentally changing the desktop. That can't happen due to the size of the installed base and the corresponding economics that prevent change. So here we sit!**

5.0 Sidebar #1: Evil Software Development



While the PC revolution was occurring in hardware, software was undergoing a revolution as well. The demand for applications grew far faster than the ability of the market to deliver them. This high demand put an unbelievable time pressure on software suppliers and systems integrators to get the job done on schedule – regardless of the number of bugs. Time-to-market was (and remains) more important than getting it right.

Software, which was formerly done by engineers who really knew and understood what the hardware could do, was now done by programmers who had little understanding of what the compiler, linker, run-time system and underlying hardware actually did with the source code that they wrote. During the past 20 years, this dichotomy has gotten far worse.

There are no universal software quality, reliability and safety standards. This is in sharp contrast to consumer products where safety standards and testing laboratories are in abundance. Software development and purchasing remains very much "*caveat emptor*".

Complicating this issue is the fact that the science of software is in its infancy. We still do not know how to produce large pieces of software that are consistently bug-free. Software is largely handcrafted and trial-and-error is the rule (what are Beta sites for?). The plethora of platforms and development tools prohibit universal standardization. And with good reason - whatever platform and operating system are chosen today as "the standards" will certainly be obsolete within a few years.

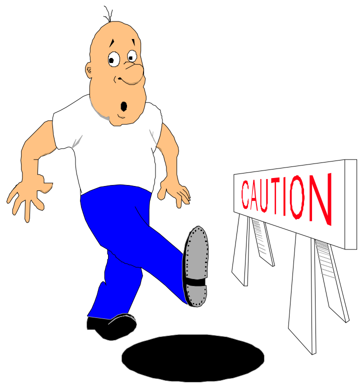
Many organizations have made the mistake of early and excessive standardization and a large number still keep making it. There is much yet to invent in the computer hardware and software world. Insistence on one system as the standard at this point would be quite premature.

What does this mean? **Well, a lot of bad software got into the market and stayed there because there were no economic consequences for producing a poor product. People bought it anyway.** Users were (and still are) largely clueless about software quality. They have little idea of what to look for, what questions to ask and how to separate fact from fiction. The best marketed programs and software systems get the most air time and become *de facto* "standards" even though there is little unbiased, trustworthy evidence on which to make an informed buying decision.

Another problem that occurred was "disposable code". Under-estimates of system longevity led to a "just get it working & ship it" approach to software development. Why plan more than a few years ahead when there's no budget to do that? Project managers get paid and promoted based upon short-term metrics. None of them get much credit for designing systems that work too far into the future. In fact, there are financial incentives not to do that. An “upgrade” or ”fix” can always be sold at a later date – long after the customer has securely chained themselves to the current software version and cannot easily change horses.

Buying software places obligations and constraints on future buying decisions. Many companies, having standardized on a certain IT system merely because the site license was cheap, found that the cost to convert all their data to a new IT system far exceeded the original purchase price of the old IT system. Choosing software merely on the basis of acquisition cost and ignoring future operational dependency and flexibility is arguably one of the biggest mistakes that can be made.

6.0 Sidebar #2: Software Engineers Vs. Programmers



In the US, there is no universally-recognized, formal certification process required to be a programmer. Some programmers are graduates of CIS (Computer and Information Science) programs, some are engineers and many are neither. Novell and Microsoft have tried to create proprietary certification with their CNE (Certified Network Engineer) and MCSE (Microsoft Certified System Engineer) training.

Many states have made such titles illegal because they mislead the public on who is really an engineer. Graduates of such training are not required to have an ABET-accredited engineering degree or a PE (Professional Engineer) license so they cannot be called engineers. This effort is in the public interest because software impacts public safety. By way of information, Ohio has rendered the MCSE and CNE titles unusable unless you are an actual engineer. Nevada also has strict engineer title laws.

The “science” of computer science has a long way to go. Few useful software development paradigms exist and the graduates are not adequately trained in their use or are even aware of their existence. The professors themselves are ignorant of current software development practices and have little to offer their students in the way of helpful suggestions. Having been a Computer Engineering professor at a large university, I can personally attest to the appalling lack of understanding of software engineering issues on the part of a few of my former colleagues.

Some organizations, such as Carnegie-Mellon’s SEI (Software Engineering Institute) are combating this widespread ignorance. Local SPIN groups (Software Process Improvement Network), an outgrowth of CMU’s SEI, are also assisting in this effort. However, as long as time-to-market issues dominate software development (rather than safety or correctness), there will be little incentive to change.

Software engineers, on the other hand, have a lot more science and technology background than do programmers or computer science majors. Because they are degreed engineers, they have the ABET-approved engineering core which includes physics, chemistry, math, engineering design, etc. Software engineers, at the graduate level, also learn management and other business aspects of the software design and production process.

Software engineering programs are still nascent, but they are gaining traction as employers realize that programmers alone cannot get the job done. It is the beginning of wisdom when a software developer learns the difference between getting a program to work – and getting it right. Experience has verified that a team of engineers is generally better to use for a large software project where reliability and flexibility is required.

7.0 What to Do?

There are no quick fixes to this growing problem. One thing, however, is almost certain. The growing body of lawsuits on software security, safety and reliability issues will lead to federal and/or state regulation. While no one welcomes this prospect, it is due to the inability and unwillingness of the software industry to police itself. The identical thing occurred with the automobile industry in the early 1900s. Now, we have the NTSB (National Transportation and Safety Board) and other organizations charged with ensuring travel safety.

In the meantime here are some helpful suggestions which, if carefully followed, will reduce your risk.

1. Recognize that desktop technology is not secure and was never intended nor designed to be secure. So, do not deploy critical applications on such systems. Just don't do it. The desktop is best suited to serve as an interface to a centrally-managed, secure application using a very thin-client architecture.
2. If something is available via a web browser, it can be hacked. All web browsers on desktop operating systems are vulnerable. So, do not allow browser-based access to anything critically important. Use a thin client like XLIB for desktop to support a centrally-managed GUI rather than a browser.
3. Understand that your network will always be polluted to some extent. The TCP/IP protocol is flawed because it permits challenge/response without authentication. So, it will always be possible to do remote foot-printing, scanning and enumeration – which are the three essential steps in the hacking process. Proper firewall configuration – and the use of only "stateful" firewalls – will help a lot but cannot completely prevent unauthorized traffic.
4. Deploy critical applications only on secure O/S platforms. If the O/S itself is not secure, the application deployed on top of it cannot be secure.
5. Spend the bucks to design new systems right the first time. It is never cheaper to redo. Also, the opportunity cost of not having the system deployed properly can be huge.
6. Plan for "rolling upgrades" with system segmentation. Use multi-vendor standards for GUI, database access and network communication. That way, you can upgrade portions of your system without disturbing the rest of it. Multi-vendor standards ensure that you have alternative sources for critical pieces of software. If you cannot get access to your data unless it's through a single-vendor's proprietary interface, shy away from that.
7. Keep the architecture flexible so it can adapt as your business needs change. You want your IT systems to enable your organization - not limit your growth.
8. Choose stuff because it works and it's reliable – not because it's cheap or convenient. The money that you save will far outweigh the little extra in up-front cost.
9. Call (330-659-6300 x221) or email (Steve.Belovich@IQware.us) us at IQware because solving this issue is what we do. We can help!