

# **Closed versus Open Systems: Converged Information Warfare**

**William M. Eldridge  
Paul Flint**

**March 18, 2003**

There is some irony in delivering a talk on Information Warfare on the eve of probable war against Iraq. An anecdote comes to mind, a war games exercise last fall in which a retired general played a particular Mideast despot fighting against the overwhelming forces of a certain world superpower. Reportedly, this savvy general dismissed traditional radio communications for message-laden motorcyclists, and instead of missiles, relied on harmless-looking rubber dinghies in the Gulf. As a result, electronic surveillance failed, several military ships were sunk, and the war games had to be restarted with a less radical approach to modern warfare.

Whether the story is true or not, the lesson is pertinent for anyone involved in Information Warfare – the enemy does not have to engage you on your terms, against your strengths. Ever since the Battle of Concord, when the Yankees chose to fire from behind trees rather than parade through an open field, the cry of the overassured and assailed has wrung out, “Why don’t they play fair?” Expensive firewalls are circumvented using war dialing, e-mail viruses, Web trojans, packet fragmenters, man-in-the-middle attacks, and a host of other techniques. And while systems get more elaborate to provide more services, the methods for exploitation multiply as well.

However, this talk is not specifically about war or growing security demands – months before 9/11 the Pentagon took its internet systems off-line due to hacker actions and security is a daily topic of newspaper stories and CERT security announcements, so this is old news. It is to address the relative strengths and weaknesses of open and closed systems in Information Warfare, both offensive and defensive, and hopefully with an unbiased view towards each. Additionally, we will assess an arena not often looked at in this regard – a government-surrogate radio broadcast facility – along with other security issues in government. Since the conversion of radio to new paradigms of Information Technology, such as digital recording systems and networked audio distribution, issues of Information Warfare take on a new light.

The three basic principles of security are Confidentiality, Integrity and Availability. In most people’s minds, it is the Confidentiality that primarily symbolizes “Security”, yet an audio server that is unavailable or a file that plays the wrong 8am program are both catastrophes in the radio world. In fact there are numerous issues in radio distribution that can be thought of as Security as never before. Some that come to mind include:

- Time synchronization, especially for digital SMPTE/MIDI/wordclock machines
- File system space for servers
- Lack of standard computer maintenance
- Software bugs (including consoles, routers, VOIP, hybrid and PABX equipment)
- User- or admin-installed software and conflicts
- Auto-installed software
- Viruses (e-mail, downloads, floppies, et al, including for the phone system)
- Network intrusion via internet, dial-in, physical console, et al.

- Denial-of-service, either through breakage or performance degradation
- Spoofing or man-in-the-middle attack
- Untested backups and equipment failures

A quick description of each might be in order. In a modern digital facility, time synchronization is important across all equipment, and loss of time sync can mean failure to record or broadcast programs, speedups or dropouts in recordings, missed queues and other problems. If time is derived via one machine on the internet, and that time server changes or goes down, the facility can suffer as well. The issue of file space on a shared inventory server means that a user taking up too much space and no quotas can mean a broadcast recording won't have enough space, as might a runaway log, disk failure or other problem. PC's sometimes aren't maintained (defragged, virus-scanned, check error logs) in the same way that tape machines received regular cleaning and parts-replacement.

Software glitches can appear in any piece of gear that has software, meaning almost everything these days. Keeping abreast of software and firmware upgrades and fixes is important. At the same time, new software and fixes on a computer can install new DLL's (shared libraries) or take up resources to make the system unstable, even if blessed by the vendor. To make matters worse, many new packages now try to do auto-updates, so the system admin may not even know what change has occurred when, only that something is now broken. Even software from official sites may contain viruses, making virus scanning essential, and new types of viruses such as e-mail and Visual Basic worms have broadened the playing field.

Hacking attempts on the Internet are rampant – the average new machine is only on the net a few hours before hackers start scanning ports for weaknesses - and frequently new machines have lots of unneeded services running with lots of unknown (to the admin) vulnerability. Dial-in modems and both traditional and voice-over-internet phone systems are susceptible to attack, as are systems in-house by disgruntled employees or unobserved visitors. Systems can be taken down by accident as well, just by sending a huge print job on a poorly designed network. A “spoofing” attack (posing as another identity) or a man-in-the-middle attack (intercepting and altering data before passing it on) takes on new meaning if the data is a VOIP dial-in program or newsfeed – perhaps an irate hacker substituting his own commentary for a live field report. Daily backups, whether of program material, station databases, or system backups, need to be verified to be trusted and archives need to be renewed (the CD and the DAT tape are not nearly as indestructible as once thought).

With a modern broadcast facility so dependent on computer and digital technology, all of the security issues of the IT world await, but generally with a much smaller budget and less computer-savvy staff to handle it. Naturally, there are some IT improvements to be had as well – automated backups, speedy restores, non-linear editing, integrated text-and-audio – but radio life grows more complex, just as the paperless office has yet to appear. Worse, those in media tend to be less equipped to handle new digital developments than traditional IT counterparts, and the implementation of security tends to be regarded as a luxury or an option. This is not to question the competence of broadcast engineers, it is simply to note that the focus of the field has changed while the new IT paradigms have not been, operationally, budget-wise and philosophically. To be fair, it can be noted that lots of traditional IT personnel take security for granted as well, or are less fluent in it than recent developments demand.

While the broadcast world has been hit harder than others by this transition, many other sectors are greatly affected, even government. Freedom of Information requirements place

restrictions on dealing with public data in the workplace. Many branches of government require an active Web presence and e-mail response, as well as call centers with help menus, database archives for records and documents, and expanded self-serve facilities for the public. Yet when things go wrong, the results are worse than crises of old. Paper trails destined for databases hit snags, resulting in halfway systems, or systems lose disks and whole sets of records are lost. Intruders can deface a public website, not only causing an eyesore but knocking out the organization's main function at the same time. Private data can accidentally be mixed with public records for release, causing embarrassment and lawsuits, while other data might be garbled to send Social Security benefits to wrong addresses or miscalculating important Treasury data. The results, and need to prevent these problems, all lie under the Information Warfare umbrella. While this may seem too broad, the analogy can be made that an army marches on its feet so boots are as important as bullets. With a serious unified preparedness focus, security problems can be addressed fully, in the areas and proportions as required. Encryption may be required in some areas, greater redundancy or replication in others, and perhaps fewer data restrictions in still others. Tradeoffs involving manpower, funding, task priorities, systems and operational requirements, data hierarchies, et al, weigh down this evaluation.

In the new dismal science of Information Warfare, risk analysis reveals acceptable risk. Risk is the opposite of assurance, and is nothing new. Information risk, the opposite of information assurance, is a new branch of an old tree. To assess risk, we first must define it with the equation below:

Risk = function of (Threats, Vulnerabilities)

**Threats** are forces committed to disruption of your service, and **vulnerabilities** are the opportunities available to disrupt it. Combined, they create the security risk facing the facility, and inherently create a **conflict in security architecture**. This new form of conflict in information technology exists at the security architecture level of operating systems. For openers, the information systems you are using have design or management goals "engineered in", as would any technical system. These goals immediately include:

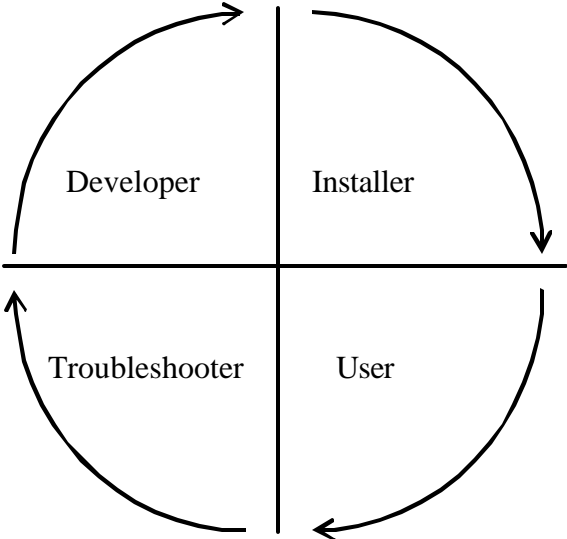
- consumption of resources (your budget);
- reproduction and growth (resources and platforms); and
- elimination of competing systems ("compatibility").

Whether these goals are part of the actual operating system code or the corporate/organizational goals of the product developer is a moot point. To function properly, information systems must have dominance in your business functions. In most cases this domination is symbiotic; the system enhances and generates value, at least for a time. Over time, they lose their value and begin to be a drain on resources.

Inherited in this shift into the IT world is the debate between open- and closed-systems. As noted by Open Source advocate Eric Raymond (<http://www.opensource.org/sco-vs-ibm.html>), the presumption that software development required highly structured, centrally organized teams has taken a beating from the scattered ad hoc groups of (white hat) "hackers" or programmers on the internet. More recently, the Open Source movement has retrenched a bit with the dot.com crash and the entrance of the likes of IBM into the Open Source community. The reality is that the computer world has been a bit of a hybrid of open/closed systems since its inception. Nonetheless, there are

various issues surrounding the tradeoffs between the two that go beyond pure advocacy and religion, and tread on real technical choices IT managers have to make, only some of which deal with the question of security or Information Warfare. While academics can take hardcore positions on either side, real-world implementations demand a more thorough analysis of tradeoffs, and more concrete answer yea or nay.

To begin with, the new-think has become that Open Source internet teams are quicker adapting to bugs, provide quicker response time to problems than stodgy corporations, and provide more thorough peer-review than their corporate counterparts. The flip-side is that Open Source projects frequently fizzle out as developers lose interest, often do not cater well to business users, and show an inherent bias against the most common system, Windows. Other stereotypes exist, and it's more illuminating to place these in a more concrete context, especially as concerns our theme. The notional diagram below illustrates traditional roles necessary in the life cycle of software projects, such as operating systems.



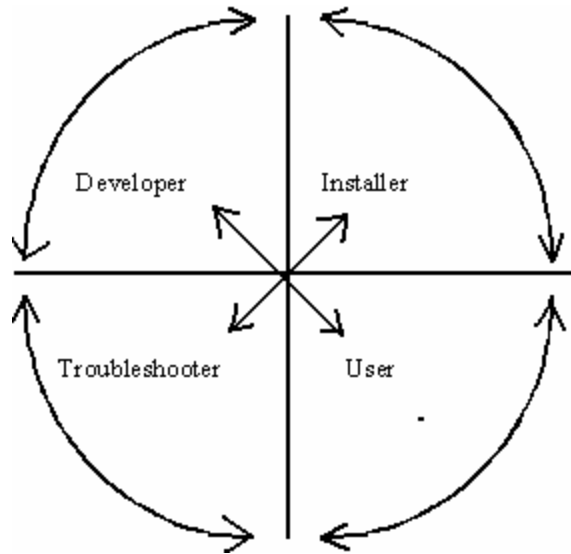
**Developers** can be an individual or a corporate entity. These are the most powerful system players. They control the source code, and through this control pretty much everything else. Sometimes but not always acting as an agent of the developer,

**Installers** apply the code to meet digital and real-world needs and requirements. They often initially control security objects, but rarely become involved in source code.

**Users** interact with the software of the digital ecology for entertainment or gain. They generally are not permitted access to either the source code or system security beyond their own identity.

**Troubleshooters** get the user out of trouble. They diagnose, secure, perform break-fix tasks and often encounter miscoding (bugs) of either innocent or malicious nature, which need to be fed back to the developer to allow the system to evolve. Typically, troubleshooters have supervisory access to the security features of the system. Currently troubleshooters seldom have access to the source code.

In the next diagram below, we see one frequent result of the Open Source paradigm – a fully meshed circle. In this case, the user has direct access to the developer, the installer and troubleshooter are in contact, and there’s quick sharing of information. Not all projects are so responsive, and some focus much more on developer and troubleshooting issues than end-user support. Nevertheless, cross-communication is in general much greater in this model.



Of course “more” is not always “better”, especially in regards to security and Information Warfare. A set of hypothetical questions is provided here to provide a setting against which to evaluate issues surrounding Open Source versus closed development. It should be noted that “Open” differs from “Free” – we are primarily interested in the question of transparent and modifiable code, as opposed to cost-free systems, even though the two do interact and the revenue issues can bias the answer away from Open systems. This is a real factor, and should be a consideration for system decision makers. Additionally, this is not specifically a clash between Linux/Unix and Windows, even though the two provide illustrations of interesting factors.

1. Would Open Source be a good choice as a defensive system?

As open-ended as this question is, it provides some hard questions. One issue raised is that of “Security through Obscurity”. While traditionally, security purists say this is a bad idea, others point out that as long as you don’t rely on it as your only security measure, it can make you invisible to 95% of your attackers, which is a good thing. Putting a Web service on a non-traditional port or giving a misleading service response can put off easy “Script Kiddie” attacks (which can occasionally bloop in success), letting you focus your energies on more serious intrusions. Windows 2000 encourages you to change the top-level Administrator account to a different name, while virtually 100% of Unix have a “root” account, along with other accounts having system privileges. Obviously a brute-force password attack on a non-existent Administrator

account is fruitless, so this should be seen as a good idea. It's some consolation that OS security was hardly existent through Windows 98 while robust in Unix from the beginning, however viewing issues from a historical perspective is not necessarily important to the IT manager.

A second example is of ACL's (Access Control Lists), which in the Windows world give multiple group lists and more granular permissions to file and folder access. While the Linux community has attempted to implement this for years, with limited success, HP has had a successful implementation of it on its HP/UX Unix for years. In this case, it appears that a commercial closed source system has been more effective than an open source effort.

A third example is Sun's NIS for network account permissions, which is known to be weak security-wise yet ships with most Linux versions, versus Sun's NIS+, which has a stronger security-model but has never inspired the Open Source community to write a free server version. In this case, the free-to-choose aspect of Open Source has left what appears to be a glaring security need, only partially filled by using Samba/NT-like security. Again, a commercial While in this case Sun has a partially closed system, the company has long been supportive of complementary Open Source efforts, both in its OS and add-ons such as Java, and much Open Source software was written originally for Sun in the pre-Linux days.

These three examples are illustrative of the fact that commercial companies do in fact write successful "secure" software. However whether it is "secure" enough can partly be measured through response times on bug fixes – CERT advisories routinely track time between bug report and fix, and Open Source projects such as Apache, Samba and Sendmail have frequently done very well in this regard. (Sun historically has had a strong bug-tracking system in place, while Microsoft has had numerous snafus in repairing damaged goods). Recently, the Open Source Bind project, which runs most of the Internet's DNS, received a black eye by providing fixes much faster to those with commercial support contracts than others, putting much of the Internet at security risk. This also raises an interesting question about the GPL (Gnu Public License, the free copyright standard for much of the Open Source movement), since the notion of how quickly releases have to be made is not mandated. In cases where vital systems are at stake and hackers have had secret exploits for weeks, every minute of vulnerability is critical.

On the flip-side, frequently responses and fixes from Open Source authors come very quickly and helpfully, and there are numerous examples of excellent Open Source projects, including compilers, Web servers, time sync programs, scripting languages, operating systems (e.g. Linux and FreeBSD), and more and more audio/video and graphics-oriented pieces. Additional projects include XML tools, software routers, conferencing systems, databases and so on. Many of these run on Windows as well, some exclusively. Apple's new Darwin-OS/X line is based on an Open Source Unix environment with many Open Source utilities thrown in. IBM's investment into GNU/Linux systems is serious, and the investment amounts to more than just money – it includes code returned to the Open Source stable.

One aspect that can make commercial systems more bulky and less-secure is the need to offer backwards compatibility and new features, and a more streamlined system might prove safer both in basic code and in actual use and configuration. An example of this might be Oracle 9i database versus MySQL (a fast Open Source model). That the maintainer of MySQL is a commercial company seems beside the point – they have a commitment to a free, high-performance, open product. Though they do provide a bit better response to feature requests for those with support contracts, they are explicit in this fact.

2. When facing an adversary who uses Open Source software, does this adversary hold any advantage?

First, if the system is better designed through whatever Open Source peer review, then the answer might be a cautious “yes”. Many Open Source projects are essentially one- or two-man efforts, which is not inherently bad, but may contain “personality” flaws (some projects have tyrants, others benevolent dictators, others democracies, still others anarchy). The same of course can be true for commercial products. Foremost of concern is the skill of the programmers, including those who might provide bug-fixes. Without programming talent, including an understanding of security concerns, no project commercial or otherwise will be appropriate for use.

Secondly, if a package is open for optimization, reconfiguration and upgrading, yet the maintainer does a “plug-and-pray” install and just lets it run, much of the Open Source advantage is lost. Similarly, a raw unconfigured Windows machine is also prone to stability, performance and security issues. There are enough weaknesses in any system and its applications to require modifications, and if the staff is not trained or does not have time to properly maintain systems, the results are predictable and not very pretty. A good firewall can remove a good amount of danger, but applications attacks such as e-mail viruses have improved, so even a minimal effort requires e-mail filtering and scanning of downloads in addition to a firewall. However, should an administrator pare down number systems and packages to a needed minimum, and spend some time on figuring out which defaults can and should be changed, he/she can find much improvement from little steps. Additionally, problems with software are often trivial to fix, even for non-programmers, such as a misformatted line in program. A little bit of experience helps to know which problems are more serious, and by limiting the packages on a machine, one can follow more closely the important group developments on the Internet.

A third aspect to this question involves program and security logs. If the maintainer is not monitoring these logs and a machine is directly connected to the Internet, it is now a safe bet that the machine will be compromised sooner rather than later. As noted, a new machine starts getting banged on almost immediately, and it may only take 3 seconds for a successful intrusion to get in and cover its tracks (generally using automated tools that are banging on hundreds of machines at the same time). If you didn't notice the original exploit, you are equally unlikely to notice the intruder residing on the system unless he/she gets sloppy, deliberately disturbs the machine, or gives some indication via a performance hit. Since a system log can log hundreds of thousands of lines per day, reading the whole log for all your machines is out of the question. Knowing the types of activity that should trigger alarms allows the admin to reduce the work to a manageable level while saving the record for analysis in case of suspected attack. Even spam (unwanted e-mail) can be handled with automated systems using Internet records. While this may seem a side-issue, the level of spam often approaches 40% these days, and can overload mail systems, waste users time reading junk, and risk the user missing important mail amongst the weeds.

The last issue to discuss here is the actual knowledge of how the system works. It is a fairly safe bet that few people know how Digital (now Compaq) OpenVMS machines work, and combined with being designed well, they are hard to hack into. When comparing two common systems, there is some advantage to knowing the exact details of a program, but many exploits are derived from basic hacking concepts, such as how to get a program to overflow and how the system reacts to that overflow. Once the exploit is discovered, news travels fast, but it seems that these tricks happen just as often for closed programs as for open. As to whether the fix will be available quickly, there is no magic to it being Open Source or a commercial support contract: if the programmers and

users push for a fix, it will be fixed, just as with a commercial company. Following the track record of the software group is more important than generalized statements about development method.

3. Can opponents using Open Source-based tools and techniques dominate closed-source opponents in Information Warfare?

One aspect of achieving domination is working faster and adapting quicker. If colleagues share information, can refine techniques, can test on more diverse systems, can share results and provide other feedback, they stand a better chance of overwhelming a relatively static target. However, defense systems are not for naught, whether open or closed. Well-designed firewalls are not by default exploitable. Well-patched machines and updated services outside the firewall or in the DMZ can have no known exploits whether open or closed. Properly authenticated dial-in access that provides direct connection to a particular service with no extraneous permissions or protocols can be save and usable at the same time. The pattern reads something like an insurance statistics for car wrecks – while the accident rate may be 10% of drivers per year, the rate for those who drive carefully might be only 0.5%, and would still be related to how much you drive. The fewer users and services allowed on a machine, the more careful the configuration, the better monitored it stays, these are all determining factors for threat evaluation.

4. Is Open Source strategically or tactically superior to closed source systems?

Again, the question centers around configuration weaknesses and project strengths as much as any theoretical issues. There are well-developed Open Source packages for probing networks, such as nmap, nessus and fragroute. At the same time, there are excellent proprietary tools providing full LAN/WAN analysis, intrusion detection, and the like. How the tools are used is as important as the nature of the tools and the policies they represent. Checking your systems for weak passwords is ineffective if you don't prevent users from creating bad passwords an hour later. Backing up your systems is useless if the tape gets damaged or the backup command had bad syntax.

A measure frequently used for Unix (closed and open) systems and less common on Windows is actively recording all system files, sizes and changedates (preferably with checksums). Should a hacker gain access and change files (perhaps adding a trojan into the directory listing program), there is usually some indication via a change in size or added group or user permissions. Having a recovery kit available (booting off a read-only disk with safe and assured versions of system utilities) is then necessary for the disinfecting/locking down phase, which also includes restoring good versions of files from backups and identifying where the intrusion came from.

5. Will closed-source systems always imitate Open Source systems (or vice-versa)?

Of course not – many systems are just good inventions or obvious needs, whether implemented as closed or open. Many others build on known concepts, whether closed or open, and some set out to directly copy existing packages. In the latter case, it may be a copyright infringement whether moving from closed to open or the other direction. The diffuse nature of the Internet can make it difficult to enforce copyrights on work stolen by Open Source projects, though frequently project leaders respond reasonably when project names or ideas are too close to existing projects. For projects going the other direction, open code (such as GPL copyright) may be hidden in closed,

compiled systems, though the EFF (Electronic Frontier Foundation) has sometimes stepped in to contest egregious acts. Eric Raymond (ibid) notes that an early important Open Source case involved source for BSD, a popular version of Unix itself, which was proven to be almost completely free of proprietary code.

6. For gaining control or dominance over information, is Open Source particularly suited as an offensive weapon?

Many hacking tips are effectively Open Source tools, though frequently they are reference implementation prototypes only, and possibly do not fully explore the vulnerability. However, the hole is usually known at that point, and should a defender be aware of the nature of the exploit, a fix is attempted whether an exploit has been defined or not. (Many of the bugs reported are theoretical in nature, with no obvious exploit implemented). Should an attacker have a closed weapon, there seems nothing to gain and much to lose by divulging the contents unless to other attackers for refinement. At that point, it is simply another Open Source project, though “Black Hat”. Should a defender get hold of the code, the nature of the exploit is usually obvious and a fix can then be attempted. Generally, exploits exist for some time before they become identified by CERT or other security teams, so the time-lag between identification and repair is even more critical, as the number of unnoticed exploited systems may already be high. Once a system is exploited, the ability to create other entries less noticeable than the break-in is enhanced.

7. Can sources and methods be made anonymous?

Hacker techniques include numerous ways of hiding origins, including proxy machines to take the heat, spoofing addresses, posing as a valid service or using a non-logged protocol. Most attacks come from already exploited machines, not from a hacker’s own machine. Hacker code itself can be encrypted and left in known repositories around the Internet. There are public code sites as well, usually implementing reference exploits, that can be useful for a script kiddie to get his or her feet wet, as well as to provide some guidance to an administrator on what is available. In general, if you can find the code easily, its level of sophistication and usefulness is low.

Another aspect of code anonymity is code signatures, or the identification of author by coding style, much as writing style can be analyzed (and far harder than so-called “virus signatures” which have blatant characteristics). As areas such as on-line crime become more sophisticated, countermeasures need to keep pace.

8. Is Open Source harder to trace?

In the sense that Open Source actually has coding styles intact, whereas compiled code (especially if stripped of internal symbols) is probably much harder to gauge. As for particular binary programs, equating two as equal requires simply data length and checksum information on the programs. For non-equal programs (perhaps a new release of an old package), program behavior or techniques might give a sign if sufficiently complex, and string information in the code can be printed for clues.

As far as tracing illegal borrowing of code, different levels of complexity exist, so that specific algorithms for MPEG encoding might be obviously lifted from another program, a client-server handshaking algorithm might be generic enough to be obvious one way or the other.

While the subject of copyrights might seem disjoint from the issue of security, illegal use of software is a threat to a company or government organization simply due to the civil and criminal penalties imposed for violations. A manager would be uncomfortable asking for additional funding to pay a large fine for someone's installed software, or having to get confiscated equipment out of the docket for obscene material. Additionally, the effect on the organization could be decreased funding from irate Congressmen or other overseers. A third case might be where a vendor has provided software to an organization, including sourcecode, and it is up to the organization to maintain confidentiality on this code. The DMCA (Digital Millenium Copyright Act) has greatly altered the playing field, generally favoring copyright owners over consumer rights. Open Source and free software come with fewer strings attached, and can make elaborate license and software installation tracking systems unnecessary.

9. Is a closed system more secure than an open one?

This aspect again ranks a big "depends". Poor programming can frequently be exploited even without exact code details. Obscure systems using unusual techniques, file names or file placement, algorithms, et al, can be difficult to hack into, and frequently fall in the "not worth the trouble" category, except for the few up for an unknown challenge. Security does involve playing the odds to a great degree, though comfort usually comes from a combination of disguise and hardening. In some cases, a typical system can be disguised as an obscure system through the fingerprint of its response to port probes and other attempted intrusions. A known security defense technique involves creating "honey pots" – machines that imitate behavior of known services while logging intrusion attempts. This provides several pieces of information – any access attempt is known to be illicit, as the machine has no valid user function, the address of the attacker is now known and can be correlated with attempts and logs on regular servers, and the techniques of the attacker can be analyzed to determine if any new type of exploit has suddenly become popular. It is this last aspect that is most important, as a rash of attempts at port 443 can signal a new Win2000 exploit, a long port 80 string focused at a C:\winnt\system32 can point out a new IIS exploit, and so on.

10. Does the compartmentalization that closed source instills help security?

The compartmentalization of code is only one area that affects security. While Open Source software seems to favor a "shotgun" approach, the reality is that most programmers work on specific areas of code. Having the code areas well-defined, with their necessary functions identified, is a good thing, even while excessive software engineering or top-down design might make the development process slower (erroneously identified as an open vs. closed issue). Code bloat not only leads to bugs, it leads to security flaws as well. Properly checking pointers and possible user inputs for bad or mischievous data becomes difficult, handling errors becomes cumbersome, and in general, the design gets lost in the excessive implementation details. The sheer number of lines of code does not by itself indicate bloat if the code is well-defined and the pieces interact sanely. But generally in a large project when new features are added pell-mell, the structure and legibility suffer. And though issues of improper user execution (such as promotion to superuser permission) is the most prominent security issue, system availability is crucial on many systems, so a program crash can be just as unwanted. It might be that the program can be restarted as long as the system stays running (as opposed to the famous Windows Blue Screen of Death), in which case proper error handling and clean exit is at least preferable to system instability. As with all aspects

of security, there can be tradeoffs in level of importance when determining requirements – trying to maintain unreachable levels of performance or assurance in one area can adversely effect another.

#### 11. Is empirical comparative evidence or casework available?

Although the authors have not conducted extensive tests, one sample data point is provided by the recent mutual probe between Radio Free Asia and Radio Free Europe. Here, a more open facility (Linux systems, services and routers) was compared against a more closed plant (Cisco, Windows, Lotus, et al). With similar levels of preparation for the test, scores on preparedness were very similar in regards to probes and vulnerabilities. The biggest weakness identified was staff time available for routine security work in the face of continuous “higher priorities”.

One aspect of the mutual probe was each side’s intro attempt into offensive hacking techniques, usually not on a network administrator’s to-do list. While informational, it is not believed that the short ramp-up time in this field makes for a rigorous stress test of security systems, which should be handled either through professional outside sources, trained (and continually training) internal staff, or a combination of both. However, the short focus on offensive techniques highlights various defensive security issues, including inappropriate approaches and unnoticed potential breaches. More casework in this area will be forthcoming.

#### 12. Is abstract or mathematically rigorous proof of superiority/inferiority possible?

While “rigorous” may be a bit overboard, the differences between open and closed development is subject to the same analyses as various manufacturing, production and collaboration models as other lines of work. Product might be specified in lines of code, modules, bugs and fixes per unit time, time required per change (code complexity), et al. Additionally, more abstract approaches such as game theory can be used to analyze the strengths and weaknesses of each style, and customer survey and test group techniques from marketing theory can identify programmer patterns and preferences. Concepts to test might be whether commercial developers look harder for bugs, whether Open Source programmers take on larger parts of the code base, the percentage of programmers from different categories familiar with various security concepts, and so on. The definition of superior is of course not absolute – just as Mac, Linux and Windows users all think their systems are the best, programmers tend to favor their approaches over others, choosing the attributes they find important. The famous dialog between Linus Torvalds and Andrew Tannenbaum is a case in point – it contrasted the use of monolithic kernels with microkernels – much of the theory obviated by the simple fact that monolithics were just faster to get out the door (<http://www.oreilly.com/catalog/opensources/book/appa.html>). Similarly, a security “purist” might recommend certain practices, yet if in real life one were able to get more security efficiency out of an uglier approach, the ugly approach would win. Facility security is a time-consuming process, and anything that makes it faster and easier should improve the overall security responsiveness, as it competes with the non-security tasks that also face the staff.

#### 13. Would Open Source and closed systems perform in similar or different ways in threat environments?

Certainly the staff expectations are different from a closed environment, and their maintenance techniques and reactions can be quite different. A Windows bug or exploit will require a Microsoft

fix, whether configuration or patch, or it will require an expedient workaround. A Gnu/Linux error may be fixable by the code author, if the package and offending code can be identified, in which case an e-mail to the program or protocol newsgroup or mailing list becomes a first step. Additionally, depending on the administrator's diagnostic and programming abilities, he/she may look for particular places where the program fails, try to look at the code at that point visually or with a debugger, may try to code a fix to submit, may recompile with patches or newer CVS (Concurrent Versions System for sharing incremental software development) versions, and so on. Closed systems will require some kind of diagnostics to provide feedback to the vendor, but focuses more frequently on the trouble call for a solution. It is more likely that commercial product will seek a workaround rather than a coding fix, as coding updates are frequently a function of multiple groups, not all programmers, and can entail stability considerations, needs of other customers, deadlines, et al. One of the peculiar aspects of Open Source development is that frequently you are directly in touch with the code developers themselves. If a closed commercial endeavor allows this kind of developer access, the response may be similar. However, a supervisor trying to reach a milestone on an important new release may be less than pleased to have a programmer go off a trivial bug.

Additional staff responses may differ on amount of reliance on systems, with the closed systems possibly favoring more passive use in more cases, though this is simply conjecture. There are robust systems that contain logging, scanning, paging, easy GUI configurations and script control, and other fine-tuning options. Administrators trained in their use may take full advantage of advanced features and closely track alarms, logs and other indicators. Sometimes closed systems are installed by vendors and the amount of baselining and tailoring for the environment can be much more than a typical Open Source environment. However this is more of a money issue, as well as reflecting a frequent bias that if something was paid for, it deserves more attention – with the same focus an open system may be set up just as completely. However, there is also sometimes more ego in an Open Source environment that may be counterproductive in a security setup – having best security practices and configurations in place at the beginning can be more important than doing it yourself. Another area of exception is GUI tools, where open systems have traditionally lagged behind commercial systems due to (white hat) hacker biases.

The last focus on the question is that in general, the equipment and systems should function similarly in both an open and a closed environment. Unless something is seriously remiss (and you should be tracking systems both before and after deployment to see if they really is a suitable system for production), most systems should be implementing their advertised features within a reasonable area of competence. An Intrusion Detection system generally works or it doesn't. During baselining you should see if any types of attacks are missed, if e-mail/paging is misconfigured, etc. After that, the system should just work. Databases generally just keep functioning, as do mail, Web and file servers. Monitors for disk and CPU usage, free memory, network access and so on are straightforward in both Open Source and proprietary versions.

There may be some need to provide some data massaging for reports, but closed programs can have output piped into bash or Python scripts as easily as open programs – at least on a Unix machine. There are different approaches to scripting and control languages in different systems, and in this particular area, it may be felt that a more flexible command-line access can improve monitoring and maintenance. Of course it helps to have both graphical and command-line control, and doing a date-sort with a title bar is very straightforward in a GUI environment.

Of course once an issue has been detected, an appropriate threat response needs to be taken, and this can be both automated and manual tasks. Failover systems, backups and recovery disks, virus

and intrusion scans, fault debugging and system patches are all needed to keep the systems up and running. While the exact details of the recovery plan may differ between open and closed systems, the important aspect is to verify that the plan can be implemented and will succeed in advance. The alternative is a leap into the unknown, politely known as “unacceptable risk”.

###

Bill Eldridge is Director of Technical Development at Radio Free Asia, and an enthusiast of music, systems and languages (both natural and artificial).

###

Paul Flint is an independent technical consultant, security analyst, author and inventor with a mere quarter century's professional experience in broadcast management, appraisal, engineering, systems network design and security architecture. Mr. Flint has experience, credentials and publications in the areas of Military, Government, Research, Education, Mass Media and Commercial Information Assurance. Additionally, Paul Flint possesses an extensive background in broadcast management, engineering, evaluation and appraisal. He currently lives in Arlington, Virginia with his children, his wife, and her two cats.