Open and Closed Systems are Equivalent (that is, in an ideal world)

Ross Anderson

Cambridge University, England; email rja14@cl.cam.ac.uk

1 Introduction

People in the open source and free software community often argue that making source code available to all is good for security. Users and experts can pore over the code and find vulnerabilities: 'to many eyes, all bugs are shallow', as Eric Raymond puts it [1]. This idea is not entirely new. In the world of cryptography, it has been standard practice since the nineteenth century to assume that the opponent knows the design of your system, so the only way you can keep him out is by denying him knowledge of a temporary variable, the key [2].

However, open design is not an idea that everyone accepts, even now. Opponents of free software argue that 'if the software is in the public domain, then potential hackers have also had the opportunity to study the software closely to determine its vulnerabilities' [3]. This issue is now assuming economic and political importance, as the antitrust settlement between Microsoft and the Department of Justice compels Microsoft to make a lot of information about interfaces available to its competitors – but with the provision that data whose disclosure might prejudice security may be withheld [4]). Unsurprisingly, Microsoft is now discovering that many more aspects of its systems are security-relevant than had previously been thought.

There is a related issue: whether information about discovered vulnerabilities may be published. In February 2003, Citibank obtained an injunction prohibiting any reporting of security vulnerabilities of automatic teller machine systems disclosed by myself and two colleagues at a trial which we were attending as expert witnesses. This was counterproductive for the bank, as it compelled us to publish our attacks in an invited talk and a technical report in the days just before the gagging hearing. We were slashdotted and the technical report was downloaded over 110,000 times [5, 6]. But this is unlikely to be the last time that gagging orders are used against security vulnerabilities; if anything, the US Digital Millennium Copyright Act and the EU Intellectual Property Rights Enforcement Directive [7] will make them even more common.

So there is growing public interest in the question of whether openness is of more value to the attacker or the defender.

This question is much more general than whether software source code should be available to users. A wide range of systems and components can be either easier, or more difficult, to test, inspect and repair depending on the available tools and access. Hardware devices can often be reverse engineered with surprisingly

little effort – although the capital resources needed to fabricate a compatible clone might be scarce. The difference between 'open' and 'closed' may also be legal rather than technical; if laws prohibit the reporting of defects, or the manufacture of compatible products, this can have much the same effect as logical or physical tamper-resistance. So in what follows I will talk about 'open systems' versus 'closed systems', which differ simply in the difficulty in finding and fixing a security vulnerability.

In May 2002, I proved a controversial theorem [8]: that, under the standard assumptions of reliability growth theory, it does not matter whether the system is open or closed. Opening a system enables the attacker to discover vulnerabilities more quickly, but it helps the defenders exactly as much.

This caused consternation in some circles, as it was interpreted as a general claim that open systems are no better than closed ones. But that is not what the theorem implies. Most real systems will deviate in important ways from the assumptions of the standard reliability growth model, and it will often be the case that open systems (or closed systems) will be better in some particular application. My theorem lets people concentrate on the differences between open and closed systems that matter in a particular case.

2 An illustration: auction equivalence

Computer scientists are familiar with some kinds of equivalence theorem. For example, Turing's work teaches us that, in some sense, all computers are equal. A machine that is Turing-powerful can be used to simulate any other such machine; a TRS-80 can in theory emulate an Ultrasparc CPU. But no computerist would understand that as a claim that any old toy computer can take over the hosting of our University's research grant database.

The equivalence of open and closed systems is a different kind of result, more like some equivalence theorems found in economics. To illustrate how such results work, let's consider the revenue equivalence theorem in auction theory.

Auctions have been around for thousands of years, and have long been a standard way of selling things as diverse as livestock, fine art, mineral rights and government bonds. How to run them has recently become a hot topic among both technologists and economists. Huge amounts have been raised in many countries from spectrum auctions, and eBay has become one of the most successful Internet companies. Auctions are also proposed as a means of allocating scarce resources in distributed systems. However, it's not always obvious how to design the most appropriate type of auction. Consider the following three schemes.

1. In the *sealed-bid* auction, everyone submits a sealed envelope containing their bid. The auctioneer opens them and awards the contract to the highest bidder.
2. In the *English* auction, the auctioneer starts out the bidding at some reserve price, and keeps on raising it until one bidder remains, who is the winner. The effect of this is that the bidder who places the highest valuation on the

contract wins, but at the valuation of the next-highest bidder (plus the bid increment).

3. The *all-pay* auction is similar to the English auction, except that at each round all the bidders have to pay the current price. Eventually, there is only one bidder left, who gets the contract – but the losers don't get a refund. (This scheme models what happens in litigation, or in a symmetric war of attrition.)

The fundamental result about auctions is the *revenue equivalence theorem* which says that under ideal conditions, you get the same revenue from any well-behaved auction [9]. The bidders will adjust their strategies to the rules set by the auctioneer, and the auctioneer will end up with the same amount of money on average.

Yet, in practice, the design of auctions matters enormously. During the recent spectrum auctions, very small changes in the rules imposed by different governments led to huge differences in outcomes. The UK and Danish governments raised huge amounts of money, while the Dutch and the Austrians got peanuts. How can this be squared with theory?

The answer is that auctions are often not well-behaved, and conditions are rarely ideal. For example, revenue equivalence assumes that bidders are risk-neutral – they are indifferent between a certain profit of $1bn and a 50% chance of a profit of $2bn. But established phone companies may be risk-averse; they may see a failure to get into the 3g market as a threat to their existence, and pay more at a sealed-bid auction out of fear. Another problem is that bidders were often able to abuse the bidding mechanism to signal their intentions to each other [10]. Yet another is entry deterrence; incumbents in an industry may be able to scare away new entrants by a variety of tactics. Yet another is that if the private information of the bidders is correlated rather than independent, the English auction should raise more money than the sealed-bid auction [11]. Yet another is that if some of the bidders have budgetary constraints, the all-pay auction may raise more money (which may help to explain why litigation consumes such a large share of America's GDP).

So the revenue equivalence theorem is important for auction designers. It should not be seen as claiming that auction rules don't matter, but rather as identifying those conditions which do matter.

With this insight, let's return to the equivalence of open and closed systems. First, we'll take a look at the standard assumptions and results of reliability growth theory.

3 Security Reliability Growth

Safety-critical software engineers have known for years that for a large, complex system to have a mean time to failure (MTTF) of 100,000 hours, it must be subject to at least that many hours of testing [12]. This was first observed by Adams in 1985 in a study of the bug history of IBM mainframe operating systems [13], and has been confirmed by extensive empirical investigations since.

The first theoretical model explaining it was published in 1996 by Bishop and Bloomfield [14], who proved that under standard assumptions this would be the worst-case behaviour. Brady, Anderson and Ball tightened this result by showing that, up to a constant factor, it was also the expected behaviour [15].

Such reliability growth models were developed for software reliability in general, but they can be applied to bugs of any particular type – such as defects that might cause loss of life, or loss of mission, or the breach of a security policy. They only require that there are enough bugs for statistical arguments to work, and that a consistent definition of 'bug' is used throughout.

When we test software, we first find the most obvious bugs – that is, the bugs with the lowest mean time to failure. After about ten minutes, we might find a bug with a 10-minute MTTF. Then after half an hour we might get lucky and find a bug with an MTTF of 42 minutes, and so on. In a large system, luck cancels out and we can use statistics. A hand-waving argument would go as follows: after a million hours of testing, we'd have found all the bugs with an MTTF of less than a million hours, and we'd hope that the software's overall reliability would be proportional to the effort invested.

Reliability growth models make this more precise. Suppose that the probability that the i-th bug remains undetected after t random tests is $e^{-E_i t}$. The Brady-Anderson-Ball model shows that, after a long period of testing and bug removal, the net effect of the remaining bugs will under certain assumptions converge to a polynomial rather than exponential distribution. In particular, the probability E of a security failure at time t, at which time n bugs have been removed, is

$$E = \sum_{i=n+1}^{\infty} e^{-E_i t} \approx K/t \tag{1}$$

over a wide range of values of t. In the appendix, I sketch the proof of why this is the case. For present purposes, note that this explains the slow reliability growth observed in practice. The failure time observed by a tester depends only on the initial quality of the code (the constant of integration K) and the time spent testing it thus far.

Does this theory apply to security vulnerabilities? Recently, Rescorla has studied the available bug databases and concluded that the rate at which vulnerabilities are depleted by discovery is very low [16]. The visual trends one can see for bugs introduced in any particular year and then discovered in subsequent years show a slow decline; and in fact, once one allows for possible sampling bias it is even possible that the rate of vulnerability discovery is constant. The available data support the assumption that vulnerabilities can be considered independent, and are consistent with the model's prediction of very slow reliability growth as a result of vulnerability discovery and removal. The numbers of vulnerabilities per product (dozens to low hundreds) are also sufficient for statistical assumptions to hold.

4 Equivalence of open and closed systems

Consider now what happens if we make the tester's job harder. Suppose that after the initial alpha testing of the product, all subsequent testing is done by beta testers who have no access to the source code, but can only try out various combinations of inputs in an attempt to cause a failure. If this makes the tester's job on average λ times harder – the bugs are λ times more difficult to find – then the probability that the i-th bug remains undetected becomes $e^{-E_i t/\lambda}$, and the probability that the system will fail the next test is

$$E = \sum_{i=n}^{\infty} e^{-E_i t/\lambda} \approx K/\lambda t \tag{2}$$

In other words, the system's failure rate has just dropped by a factor of λ, just as we would expect.

However, what if all the testing to date had been carried out under the more difficult regime? In that case, only $1/\lambda$ the amount of effective testing would have been carried out, and the λ factors would cancel out. Thus the failure probability E would be unchanged.

Going back to our intuitive argument, making bugs five times more difficult to find will mean that we now work almost an hour to find the bug whose MTTF was previously 10 minutes, and over three hours for the 42-minute bug (Fenton and Neil suggest that λ lies between 3 and 5 for mature systems [17]). But the reliability of software still grows as the time spent testing, so if we needed 10,000 hours of testing to get a 10,000-hour-MTTF product before, that should still hold now. We will have removed a smaller set of bugs, but the rate at which we discover them will be the same as before.

Consider what happens when proprietary software is first tested by insiders with access to source code, then by outsiders with no such access. With a large commercial product, dozens of testers may work for months on the code, after which it will go out for beta testing by outsiders with access to object code only. There might be tens of thousands of beta testers, so even if λ were as large as 100, the effect of the initial, open, alpha-testing phase will be quickly swamped by the very much greater overall effort of the beta testers.

Then a straightforward economic analysis can in principle tell us the right time to roll out a product for beta testing. Alpha testers are more expensive, being paid a salary; as time goes on, they discover fewer bugs and so the cost per bug discovered climbs steadily. At some threshold, perhaps once bug removal starts to cost more than the damage that bugs could do in a beta release product, alpha testing stops. Beta testing is much cheaper; testers are not paid (but may get discounted software, and still incur support costs). Eventually – in fact, fairly quickly – the beta test effort comes to dominate reliability growth.

So, other things being equal, we expect that open and closed systems will exhibit similar growth in reliability and in security assurance. This assumes that there are enough bugs to do statistics, that they are independent and identically

distributed, that they are discovered at random, and that they are fixed as soon as they are found.

5 Symmetry Breaking

This does not of course mean that, in a given specific situation, proprietary and open source are evenly matched. A vendor of proprietary software may have other reasons for not making source code available. Microsoft managers once argued that they feared an avalanche of lawsuits by people holding software patents with little or no merit, but who hoped to extract a settlement by threatening expensive litigation. The technical assumptions of reliability growth theory could also fail to hold for many reasons, some of which I'll discuss below. If the analogy with the revenue equivalence theorem is sound, then this is where we expect the interesting economic and social effects to be found.

Even though open and closed systems are equally secure in an ideal world, the world is not ideal, and is often adversarial. Attackers are likely to search for, find and exploit phenomena that break the symmetry between open and closed models. (This is also similar to the auction theory case; phone companies spent considerable sums of money on hiring economists to find ways in which spectrum auctions could be gamed [10].)

5.1 Transients

Transient effects may matter, as K/t holds only at equilibrium. Suppose that a new type of abstract attack is found by an academic researcher and published. It may be simple to browse the GNU/Linux source code to see if it can be applied, but much more complex to construct test cases, write debugging macros etc, to see if an exploit can be made for Windows. So there may be time-to-market issues for the attacker.

According to Adams, IBM fixed mainframe operating system bugs the eighth time they were reported [13]. Leung studied the optimal frequency of security updates from the customer perspective: because of the risk that applying a service pack may cause critical systems to stop working, it may be quite rational for many customers to delay application [18]. Vendors also delay fixing bugs, because it costs money to test fixes, bundle them up into a service pack and ship them to millions of customers. So there may be time-to-market issues for the defenders too, and at several levels.

Transient effects may be the dominant factor in network security at present, as most network exploits use vulnerabilities that have already been published and for which patches are already available. If all patches were applied to all machines as soon as they were shipped, then the pattern of attacks would change radically. This is now rightly an area of active research, with engineers developing better patching mechanisms and security economists engaged in controversy. For example, Rescorla argues that, in order to optimise social welfare, vulnerability disclosure should be delayed [16], while Arora, Telang and Xu argue that either disclosure should be accelerated, or vendor liability increased [19].

5.2 Transaction costs

Time-to-market issues largely depend on the effects of a more general problem, namely transaction costs.

Transaction costs may persuade some vendors to remain closed. For example, if source code were made available to beta testers too, then the initial reliability of beta releases would be worse, as the testers would be more efficient. Fairly soon, the reliability would stabilise at the status quo ante, but a much larger number of bugs would have had to be fixed by the vendor's staff. Avoiding this cost might sometimes be a good argument against open systems.

5.3 Complexity growth

Software becomes steadily more complex, and reliability growth theory leads us to expect that the overall dependability will be dominated by newly added code [15]. Thus, while we may never get systems that are in equilibrium in the sense of the simple model, there may be a rough second-order equilibrium in which the amount of new code being added in each cycle is enough to offset the reliability gains from bug-fixing activities since the last cycle. Then the software will be less dependable in equilibrium if new code is added at a faster rate.

So commercial featuritis can significantly undermine code quality. But software vendors tend to make their code just as complex as they can get away with, while collaborative developers are more likely to be 'scratching an itch' than trying to please as many prospective customers as possible [1]. Certainly products such as OpenOffice appear to lag their commercial competitors by several years in terms of feature complexity.

5.4 Correlated bugs

Just as correlated private information can break the equivalence of different types of auction, so also can correlations between security vulnerabilities cause the equivalence of attack and defence to fail.

Recently, most reported vulnerabilities in operating systems and middleware have related to stack overflow attacks. This may have helped the attackers in the beginning; an attacker could write a test harness to bombard a target system with unsuitable inputs and observe the results. More recently, technological changes may have tilted the playing field in favour of the defenders: the typical information security conference now has a number of papers on canaries, static code analysis tools and compiler extensions to foil this type of attack, while Microsoft's programmers have been trained in their own way of doing things [22]. There is also the prospect that better implementation of execute permissions in future processors may eventually make stack overflow attacks obsolete.

Correlated bugs are not, however, the defender's friend. In extreme cases, they may make security systems very brittle. The cryptographic processors used by banks to protect cash machine PINs, for example, have been around for some 20 years. Their design was relatively obscure; some products had manuals

available online, but few people outside the payment industry paid them any attention. After the first attacks were published in late 2000, this changed. Many further attacks were soon found and the technology has been rendered largely ineffective [5, 20].

5.5 Code quality

In the ideal case, system dependability is a function only of the initial code quality K and the amount of testing t. However, it is not clear that code quality is a constant. Many people believe that open systems tend to have higher quality code to begin with, that is, a lower value of K.

Knowing that one's code may be read and commented on widely can motivate people to code carefully, while there may also be selection effects: for example, programmers with greater skill and motivation may end up working on open systems. A lot of labour is donated to open system projects by graduate students, who are typically drawn from the top quartile of computer science and engineering graduates. Meanwhile, commercial deadlines can impose pressures that cause even good coders to work less carefully (see section 5.8 below for more on this). Open systems may therefore start out with a constant-factor advantage.

5.6 Effectiveness of testing

Just as K can vary, so can t. It is quite conceivable that the users of open products such as GNU/Linux and Apache are more motivated to report system problems effectively, and it may be easier to do so, compared with Windows users who respond to a crash by rebooting and would not know how to report a bug if they wanted to.

An issue that may push in the other direction is that security testing is much more effective if the testers are hostile [21]. Evaluators paid by the vendor are often nowhere near as good at finding flaws as the people who attack a system once it's released – from competitors to research students motivated by glory. In many cases, this effect may simply tweak the value of λ. However, there have been occasional step-changes in the number and hostility of attackers. For example, after Sky-TV enciphered the channel containing 'Star Trek' in the early 1990s, students in Germany could no longer get legal access to the programme, so they spent considerable energy breaking its conditional access system [20]. In the case of Windows versus GNU/Linux, people may be more hostile to Windows both for ideological reasons and because an exploit against Windows allows an attacker to break into more systems.

What is the net effect on t (and K)? Recently, both Windows and GNU/Linux have been suffering about fifty reported security vulnerabilities a year (for precise figures by product and release, see [16]). Given that Windows has perhaps 10-20 times as many users, one would expect t to be larger and thus K/t to be smaller by this amount; in other words, we would expect Windows to be 10-20 times more reliable. As it clearly isn't, one can surmise that different values of

K and of testing effectiveness (in effect, a multiplier of t) help GNU/Linux to make back the gap.

5.7 Policy incentives on the vendor

In addition to the code and testing quality effects, which work through individual programmers and testers, there are also incentive issues at the corporate level.

The motivation of the vendor to implement fixes for reported bugs can be affected in practice by many factors. The US government prefers vulnerabilities in some products to be reported to authority first, so that they can be exploited by law enforcement or intelligence agencies for a while. Vendors are only encouraged to ship patches once outsiders start exploiting the hole too.

5.8 Time-to-market incentives on the vendor

There are also the issues discussed in [31]: the economics of the software industry (high fixed costs, low variable costs, network effects, lock-in) lead to dominant-firm markets with strong incentives to ship products quickly while establishing a leading position. Firms will therefore tend to ship a product as soon as it's good enough; similarly, given that fixing bugs takes time, they may fix only enough bugs for their product to keep up with the perceived competition. For example, Microsoft takes the perfectly pragmatic approach of prioritising bugs by severity, and as the ship date approaches the bug categories are allowed to slip. So more severe bugs are allowed through into the product if they are discovered at the last minute and if fixing them is non-trivial [24].

5.9 Industry structure issues for the vendor

The size of the vendor and the nature of sectoral competition can be the source of a number of interesting effects. Gal-Or and Ghose show that larger firms are more likely to benefit from information sharing than smaller ones, as are firms in larger industries; and that information sharing is more valuable in more competitive industries [23]. The critical observation is that openness saves costs – so the biggest spenders save the most.

The extent to which industries are vertically integrated may also matter. Many vulnerabilities affecting Windows PCs can be blamed on Microsoft as the supplier of the most common operating system and the dominant productivity application, as well as web server and database products. On the other hand, smartcards are typically designed by one firm, fabricated by a second using components licensed from multiple specialists, then loaded with an operating system from a third firm, a JVM from a fourth, and a crypto library from a fifth – with power analysis countermeasures bought in from yet another specialist. On top of this, an OEM will write some applications, and the customer still more.

The security of the resulting product against a given attack – say, fault induction – may depend on the interaction between hardware and software components from many different sources. Needless to say, many of the component

vendors try to dump liability either upstream or downstream. In such an environment, obscure proprietary designs can undermine security as they facilitate such behaviour. Laws such as the EU electronic signature directive, which make the cardholder liable for security failures, may compound the perverse incentive by leading all the other players to favour closed design and obscure mechanisms [25].

5.10 PR incentives on the vendor

Firms care about their image, especially when under pressure from regulators or anti-trust authorities. Our team has long experience of security hardware and software vendors preferring to keep quiet about bugs, and only shipping patches once their hand is forced (e.g., by TV publicity). They may feel that shipping a patch undermines previous claims of absolute protection. Even if 'unbreakable security' is not company policy, managers may not want to undermine assurances previously given to their bosses. So there may be information asymmetries and principal-agent effects galore.

The argument is now swinging in favour of policies of vulnerability disclosure after a fixed notice period; without the threat of eventual disclosure, little may get done [26, 27]. This is not going to be a panacea, though; on at least one occasion, a grace period that we gave a vendor before publication was consumed entirely by internal wrangling about which department was to blame for the flaw. In another case, vendors reassured their customers that attacks colleagues and I had published were 'not important', so the customers had done nothing about them.

5.11 Operational profile

Another set of issues have to do with the operational profile, which is how the reliability community refers to test focus. The models discussed above assume that testing is random, yet in practice a tester is likely to focus on a particular subset of test cases that are of interest to her or are easy to perform with her equipment.

However, the individual preferences and skills of testers still vary. It is well known that software may be tested extensively by one person, until it appears to be very reliable, only to show a number of bugs quickly when passed to a second tester [28]. This provides an economic argument for parallelism in testing [15]. It is also a strong argument for extensive beta testing; a large set of testers is more likely to be representative of the ultimate user community.

Experienced testers know that most bugs are to be found in recently added code, and will focus on this. In fact, one real advantage that source code access gives to an attacker is that it makes it easier to identify new code. In theory, this does not affect our argument, as the effects are subsumed into the value of λ. In practice, with systems that depart from the ideal in other ways, it could be important.

5.12 Adverse selection

Operational profile issues can combine with adverse selection in an interesting way. Security failures often happen in the boring bits of a product, such as device drivers and exception handling. The tempting explanation is that low-status programmers in a development team – who may be the least experienced, the least motivated, the least able (or all of the above) – are most likely to get saddled with such work.

5.13 Coase's Penguin and the Wild West

A related argument for closed systems is as follows. Think of the Wild West; the bandits can concentrate their forces to attack any bank on the frontier, while the sherriff's men have to defend everywhere. Now, the level of assurance of a given component is a function of the amount of scrutiny that it actually gets, not of what it might get in theory. As testing is boring, and volunteers generally only want to fix failures that irritate them, the amount of concentrated attention paid by random community members to (say) the smartcard device drivers for GNU/Linux is unlikely to match what an enemy government might invest [29].

A counter-argument can be drawn from Benkler's model, that large communities can include individuals with arbitrarily low reservation prices for all sorts of work [30]. A different one arises in the context of reliability growth theory. Efficacy of focus appears to assume that the attacker is more efficient than the defender at selecting a subset of the code to study for vulnerabilities; if they were randomly distributed, then no one area of focus should be more productive for the attacker than any other.

The more relevant consideration for security assurance is, I believe, the one in [31] – that a large number of low-probability bugs structurally favours attack over defence. In an extreme case, a system with 10^6 bugs each with an MTTF of 10^9 hours will have an MTBF of 1000 hours, so it will take about that much time to find an attack. But a defender who spends even a million hours has very little chance of finding that particular bug before the enemy exploits it. This problem was known in generic terms in the 1970s; the above model makes it more precise. (It also leads to Rescorla's disturbing argument that if vulnerabilities truly are uncorrelated, then the net benefit of disclosing and fixing them may be negative – patched software doesn't get much harder to attack, while software that's not patched yet becomes trivial to attack [16].)

5.14 Do defenders cooperate or free-ride?

We mentioned above that the users of open systems might be better at reporting bugs. Such factors are not restricted to the demand side of the bug-fixing business, but can affect the supply side too. The maintainers of open systems might take more pride in their work, and be more disposed to listen to complaints, while maintainers working for a company might be less well motivated. They might see bug reports as extra work and devise mechanisms – even subconsciously – to

limit the rate of reporting. On the other hand, a corps of paid maintainers may be much easier to coordinate and manage, so it might get better results in the long term once the excitement of working on a new software project has paled. How might we analyse this?

We mentioned industries, such as the smartcard industry, where many defenders have to cooperate for best esults. Varian presents an interesting analysis of how defenders are likely to react when the effectiveness of their defence depends on the sum total of all their efforts, the efforts of the most energetic defender, or the efforts of the least energetic defender [32]. In the total-efforts case, there is always too little effort exerted at the Nash equilibrium as opposed to the optimum, but at least reliability continues to increase with the total number of participants.

6 Conclusion

The debate about open versus closed systems started out in the nineteenth century when Auguste Kerckhoffs pointed out the wisdom of assuming that the enemy knew one's cipher system, so that security could only reside in the key. It has developed into a debate about whether access to the source code of a software product is of more help to the defence, because they can find and fix bugs more easily, or to attackers, because they can develop exploits with less effort.

This paper gives a partial answer to that question. In a perfect world, and for systems large and complex enough for statistical methods to apply, the attack and the defence are helped equally. Whether systems are open or closed makes no difference in the long run.

The interesting questions lie in the circumstances in which this symmetry can be broken in practice. There are enough deviations from the ideal for the choice between open and closed to be an important one, and a suitable subject for researchers in the economics of information security. The balance can be pushed one way or another by many things: transient effects, transaction costs, featuritis, interdependent or correlated vulnerabilities, selection effects, incentives for coders and testers, agency issues, policy and market pressures, changing operational profiles, and the effects of defenders who cheat rather than collaborate. (This list is surely not complete.)

Although some of these effects can be modelled theoretically, empirical data are needed to determine which effects matter more. It might be particularly interesting, for example, to have studies of reliability growth for code that has bifurcated, and now has an open and a closed version.

In conclusion, I have not proved that open and closed systems are always equivalent. They are in an ideal world, but our world is not ideal. The significance of this result is, I hope, to have made a start towards a better understanding of the circumstances in which open systems (or closed systems) are best – and to help us focus on the factors that actually matter.

Acknowledgements: I got useful comments from Rob Brady, Hal Varian, Jacques Crémer, Peter Bishop, Richard Clayton, Paul Leach, Peter Wayner, Fabien Petitcolas, Brian Behlendorf, Seth Arnold, Jonathan Smith, Tim Harris, Andrei Serjantov, Mike Roe and Eric Rescorla; and from attendees at talks I gave on this subject at the Toulouse conference on Open Source Software Economics; at City University, London; at the National E-science Centre, Edinburgh; and at the Cambridge Seminar on Economics, Networks and Security.

References

1. ES Raymond, "The Cathedral and the Bazaar", 1998, at `http://tuxedo.org/~esr/writings/cathedral-bazaar/`
2. A Kerckhoffs, "La Cryptographie Militaire", in *Journal des Sciences Militaires*, 9 Jan 1883, pp 5–38; `http://www.cl.cam.ac.uk/users/fapp2/kerckhoffs/`
3. K Brown, "Opening the Open Source Debate", 2002, Alexis de Toqueville Institution, at `http://www.adti.net/html_files/defense/opensource_whitepaper.pdf`
4. Collar-Kotelly J, US District Court, District of Columbia, Civil Action No. 98-1232 (CKK), United States of America vs Microsoft Inc., Final Judgment (November 12 2002), at `http://www.usdoj.gov/atr/cases/f200400/200457.htm`
5. RJ Anderson, M Bond, "Protocol Analysis, Composability and Computation", in *Computer Systems: Papers for Roger Needham*, Microsoft Research, February 2003, at `http://cryptome.org/pacc.htm`
6. M Bond, P Zielinski, "Decimalisation Table Attacks for PIN Cracking", Cambridge University Computer Laboratory Technical Report no. 560, at `http://www.cl.cam.ac.uk/TechReports/UCAM-CL-TR-560.pdf`
7. Draft *'EU Directive on the enforcement of intellectual property rights'*, at `http://europa.eu.int/comm/internal_market/en/indprop/piracy/index.htm`
8. RJ Anderson, "Security in Open Versus Closed Systems – the Dance of Boltzmann, Coase and Moore", at Open Source Software Economics 2002, at `http://www.idei.asso.fr/ossconf.html`
9. P Klemperer, "Auction Theory: A Guide to the Literature", in *Journal of Economic Surveys* v 13 no 3 (1999); at `http://www.paulklemperer.org/`
10. P Klemperer, "Using and Abusing Economic Theory—Lessons from Auction Design", 2002 Alfred Marshall lecture to the European Economic Association, at `http://www.paulklemperer.org/`
11. P Milgrom, R Weber, "A Theory of Auctions and Competitive Bidding", *Econometrica* v 50 no 5 (1982), pp 1089–1122
12. RW Butler, GB Finelli, "The infeasibility of experimental quantification of life-critical software reliability", *ACM Symposium on Software for Critical Systems*, New Orleans ISBN 0-89791-455-4 pp 66–76 (Dec 1991)
13. Adams E. N., "Optimising preventive maintenance of software products", *lBM Journal of Research & Development*, Vol. 28, issue 1 pp 2–14 (1984)
14. P Bishop, R Bloomfield, "A Conservative Theory for Long-Term Reliability-Growth Prediction', *IEEE Transactions on Reliability* v 45 no 4 (Dec 96) pp 550–560
15. RM Brady, RJ Anderson, RC Ball, *'Murphy's law, the fitness of evolving species, and the limits of software reliability'*, Cambridge University Computer Laboratory Technical Report no. 471 (September 1999), available at `http://www.cl.cam.ac.uk/ftp/users/rja14/babtr.pdf`

16. E Rescorla, "Is finding security holes a good idea?", in *Workshop on Economics and Information Security*, May 13–15 2004, Minneapolis, at http://www.rtfm.com/bugrate.html and http://www.dtc.umn.edu/weis2004/agenda.html
17. NE Fenton, M Neil, "A Critique of Software Defect Prediction Models", in *IEEE Transactions on Software Engineering* vol 25 no 5 (Sep/Oct 1999) pp 675–689; at http://www.dcs.qmul.ac.uk/~norman/papers/defects_prediction_preprint105579.pdf
18. KS Leung, "Diverging economic incentives caused by innovation for security updates on an information network", available at ¡http://www.sims.berkeley.edu/resources/affiliates/workshops/econsecurity/¿
19. A Arora, R Telang, H Xu, "Timing Disclosure of Software Vulnerability for Optimal Social Welfare", in *Workshop on Economics and Information Security*, May 13–15 2004, Minneapolis, at http://www.dtc.umn.edu/weis2004/agenda.html
20. RJ Anderson, *'Security Engineering – A Guide to Building Dependable Distributed Systems'* Wiley (March 2001), ISBN 0-471-38922-6
21. RJ Anderson, SJ Beduidenhoudt, "On the Reliability of Electronic Payment Systems", in *IEEE Transactions on Software Engineering* vol 22 no 5 (May 1996) pp 294–301; at http://www.cl.cam.ac.uk/ftp/users/rja14/meters.ps.gz
22. M Howard, D LeBlanc, *'Writing Secure Code'*, Microsoft Press 2002 (second edition 2003), ISBN 0-7356-1588-8
23. E Gal-Or, A Ghose, "The Economic Consequences of Sharing Security Information", to appear at *Workshop on Economics and Information Security*, 2003
24. N Myrhvold, *personal communication*
25. N Bohm, I Brown, B Gladman, "Electronic Commerce: Who Carries the Risk of Fraud?", Journal of Information Law and Technology, October 2000, at http://elj.warwick.ac.uk/jilt/00-3/bohm.html
26. Rain Forest Puppy, "Issue disclosure policy v1.1", at http://www.wiretrip.net/rfp/policy.html
27. D Fisher, "OIS Tackles Vulnerability Reporting", Eweek.com, March 20 2003, at http://www.eweek.com/article2/0,3959,950860,00.asp
28. PG Bishop, "Rescaling Reliability Bounds for a New Operational Profile", in *International Symposium on Software Testing and Analysis* (ISSTA 2002), Rome, Italy, July 22-24, 2001
29. M Schaefer, *Panel comments at Oakland 2001*
30. Y Benkler, "Coase's Penguin, or, Linux and the Nature of the Firm", at *Conference on the Public Domain*, Nov 9-11, Duke Law School, available at http://www.law.duke.edu/pd/papers/Coase's_Penguin.pdf
31. RJ Anderson, "Why Information Security is Hard – An Economic Perspective", in *Proceedings of the Seventeenth Computer Security Applications Conference*, IEEE Computer Society Press (2001), pp 358–365; available at http://www.cl.cam.ac.uk/ftp/users/rja14/econ.pdf
32. H Varian, "System Reliability and Free Riding", available at http://www.sims.berkeley.edu/resources/affiliates/workshops/econsecurity/econws/49.pdf

Appendix

The following exposition is taken from [15], and uses an argument familiar to students of statistical mechanics. If there are $N(t)$ bugs left after t tests, let the

probability that a test fails be $E(t)$, where a test failure counts double if it is caused by two separate bugs. Assume that no bugs are reintroduced, so that bugs are removed as fast as they are discovered. That is:

$$dN = -Edt \tag{3}$$

By analogy with thermodynamics, define a 'temperature' $T = 1/t$ and 'entropy' $S = \int dE/T$. Thus $S = \int tdE = Et - \int Edt$. This can be solved by substituting equation 1, giving $S = N + Et$. The entropy S is a decreasing function of t (since $dS/dt = tdE/dt$ and $dE/dt < 0$). So both S and N are bounded by their initial value N_0 (the number of bugs initially present) and the quantity $S - N = Et$ is bounded by a constant k (with $k < N_0$), that is:

$$E \leq k/t \tag{4}$$

Et vanishes at $t = 0$ and $t = W_0$, where W_0 is the number of input states the program can process. It has a maximum value $Et = k$. We now wish to show that this maximum is attained over a wide range of values of t, and indeed that $Et \approx k$ for $N_0 \ll t \ll W_0$. This will be the region of interest in most real world systems.

We can write the above equation as $Et = k - g(t)$ where $0 \leq g(t) \leq k$. Since $g(t)$ is bounded, we cannot have $g(t) \sim t^x$ for $x > 0$. On the other hand, if $g(t) = At^{-1}$, then this makes a contribution to N of $-\int g(t)dt/t = A/t$, which is reduced to only one bug after A tests, and this can be ignored as $A < k$. Indeed, we can ignore $g(t) = At^{-x}$ unless x is very small. Finally, if $g(t)$ varies slowly with t, such as $g(t) = At^{-x}$ for small x, then it can be treated as a constant in the region of interest, namely $N_0 \ll t \ll W_0$. In this region, we can subsume the constant and near-constant terms of $g(t)$ into k and disregard the rest, giving:

$$E \approx k/t \tag{5}$$

Thus the mean time to failure is $1/E \approx t/k$ in units where each test takes one unit of time.

More precisely, we can consider the distribution of defects. Let there be $\rho(\epsilon)d\epsilon$ bugs initially with failure rates in ϵ to $\epsilon + d\epsilon$. Their number will decay exponentially with characteristic time $1/\epsilon$, so that $E = \int \epsilon\rho(\epsilon)e^{-\epsilon t}d\epsilon \approx k/t$. The solution to this equation in the region of interest is

$$\rho(\epsilon) \approx k/\epsilon \tag{6}$$

This solution is valid for $N_0 \ll 1/\epsilon \ll W_0$, and is the distribution that will be measured by experiment. It differs from the ab-initio distribution because some defects will already have been eliminated from a well tested program (those in energy bands with $\rho(\epsilon) \sim \epsilon^x$ for $x > -1$), and other defects are of such low energy that they will almost never come to light in practical situations (those in energy bands with $\rho(\epsilon) \sim \epsilon^x$ for $x < -1$).