Software Failures, Security, and Cyberattacks

by Charles Perrow, Yale University, New Haven CT, USA

This paper will be concerned with faulty software code that is combined with an integrated, rather than modular architecture, thus posing a cyberterror threat to our critical infrastructure. While faulty software code is ubiquitous and enables cyberattacks, I will argue that the greater threat for catastrophic events are managerial strategies. Large, dominant companies favor integrated, rather than modular software-system architectures. A small part of the software embedded in critical technical systems may compromise the much larger part.

1 Introduction

Software failures have yet to have catastrophic consequences for society, and their effect upon critical infrastructures has been limited (Rahman et al. 2006). While faulty software code is ubiquitous, it is not often exploited, and when it is, it is largely because hackers use a fault to penetrate the system, mostly to commit financial fraud. The extent of the fraud is huge, but the financial firms that bear most of the cost tolerate it; Internet financial transactions have grown much faster than the amount of fraud. This paper will be more concerned with faulty code that is combined with an integrated, rather than modular architecture, thus posing a cyberterror threat to our critical infrastructure. (I will also limit the discussion to operating systems (OS), and ignore the newer security breaches involving the root directory, buggy servers, and such things as the domain name system DNS.) As software becomes ever more ubiquitous, it is finding its way into all of our critical infrastructures, including those loaded with deadly substance. It may be only a matter of time – five or ten years perhaps – before we have a software failure, whether inadvertent or deliberate, that kills 1,000 people or more. But, as yet, our risky systems have proven to be robust, even with ubiquitous software failures.

The current and mounting concern is the risk of cyberattacks that deny service or take over systems. While faulty software enables cyberattacks, I will argue that the more serious cause is managerial strategies that make attacks easier, because they favor integrated, rather than modular system architectures. The Internet runs on UNIX, which is quite secure, but the user community mostly utilizes vulnerable Windows products when accessing the Internet, allowing intrusion from malicious hackers, business competitors, foreign states, and from terrorists, though this last has yet to occur.

2 Control Systems

The highest rate of cyber attacks in the U.S. – largely unsuccessful as yet – are directed at companies dealing with the nation's critical infrastructure, including power supply and -transmission, chemical plants, financial institutions, transportation, and even manufacturing. Of particular concern are attacks on distributed control systems (DCS), programmable logic controllers (PLC), supervisory control and data acquisition (SCADA) systems, and related networked-computing systems. I will refer to all of these as "control systems" or SCADA systems. The security aspect is not limited to Internet security; indeed, according to one estimate, 70 percent of cybersecurity incidents are inadvertent, and do not originate in the Internet (Weiss et al. 2007). However, non-Internet incidents will be random, while a strategic adversary may direct targeted attacks from the Internet.

2.1 Attacks upon SCADA Systems

SCADA systems automatically monitor and adjust switching, manufacturing, and other process control activities, based on digitized feedback data gathered by sensors. They usually lie in remote locations, are unmanned, and are accessed periodically via telecommunication links. One source notes that there has been a tenfold increase in the number of successful attacks upon SCADA systems between 1981 and 2004, but without disclosing their actual number (Wilson

2005). These software failures might seem to be surprising, since these are proprietary systems that are, uniquely, custom-built, and use only one or a few microprocessors or computers. They are thoroughly tested and in constant use, allowing bugs to be discovered and corrected. Their software stems predominantly from organizations such as SAP, a huge software and service firm, or IBM – the largest software firm. IBM's CICS service runs ATM programs, credit card transactions, travel reservations, real-time systems in utilities and banks, and much more. SAP is used in most of the Fortune 500 workstations and "is a more potent monopolistic threat to the U.S. than Microsoft" (Campbell-Kelly 2003, p. 197). It is first and foremost among financial management systems, human capital management, enterprise asset management systems, and manufacturing operations (Bailor 2006). Depending upon how financial size is measured, SAP usually rates among the top ten in terms of software revenue.

Apparently, SAP- and CICS softwares are very secure and reliable in themselves, as they are continually tested in operation, and their vendors work extensively with the customers (Campbell-Kelly 2003, p. 191–198; Cusumano 2004). They are not "plug and play" software, but are linked to such software. Increasing numbers of organizations want their industrial control systems to be linked to more general office programs, because of the valuable data they generate, because the data can be accessed online, and for accounting and other business reasons. This is the source of two types of problems.

First, information technology (IT)-experts working mainly from the front office have little understanding of the industrial control systems they link up to, and control system professionals have little understanding of IT operations. The number of experts with knowledge of both fields is roughly estimated to be about 100 in the U.S. (according to one expert in personal communication 2008). Consequently, faulty interactions between the two systems cause errors in cybersecurity that can disrupt operations, and although there are no known instances of this, it leaves the systems vulnerable to deliberate attacks (Weiss 2007). To the annoyance and alarm of cybercontrol-system experts, IT experts do not acknowledge this problem area. Complexly-interactive systems require unusual organizational structures and leadership to surmount such problems.

2.2 Safety of Commercial-off-the-shelf Software

Second, and much more important, the computers are connected through the operating systems with computers in the front office of the firm. These have applications based upon widely-used commercial-off-the-shelf (COTS) software products. By integrating the front office with the industrial operating systems, no matter how reliable and secure the latter are, they are affected by the insecurity of the COTS. The most common source of these front-office products is likely to be Microsoft, as its Office programs dominate the market. Not all COTS products are from Microsoft. Apple's Mac products are COTS, and so is "open source" software, such as Unix and its offspring Linux. But the vast majority of COTS products that run in the Internet have a Microsoft origin. Microsoft accounts for only about 10 percent of the software production (Campbell-Kelly 2003, p. 234), but most software is written for custom, in-house applications, or to connect with chips in stand-alone applications, down to the lowly electric toasters. A much smaller amount of software is plug-and-play, that is, "shrink wrap", mass-market software. Microsoft writes over half of that software, and the critical infrastructure uses it.

The problem here is, obviously, connecting reliable systems to non-secure, bug-laden software, whether it is in the server, or in the operating system, such as Windows XP or Vista, or applications that run on it, such as Office or PowerPoint. These products are necessarily quickly pushed onto the market to gain a competitive edge. When designing malicious code, attackers take advantage of vulnerabilities in software. In 2006, there were more than 8,000 reports of vulnerabilities in marketed software, most of which could easily have been avoided, according to Carnegie Mellon University's computer emergency response team (CERT 2007).

SAP has a close working relationship with Microsoft, so they know what they are linking

up to, and undoubtedly try to insure that the Microsoft products they connect to are reliable and secure. But Microsoft products are not very reliable and secure, though the company has reportedly improved them in the last decade. Until recently, studies consistently showed that open source software and Unix and Linux operating systems were more reliable and secure and that they could produce patches more quickly when needed. More recent research has challenged that; Microsoft is doing somewhat better than Apple in patching and in removing bugs, but it was a Windows-vulnerability that allowed the immensely powerful Stuxnet worm to infect the Siemens-centrifuges in Iran.

2.3 Components Interdependency: Modularity and Complexity

Some very interesting work on error propagation strongly supports the idea that open source software in particular, and Apple software to a lesser degree, is more resilient than proprietary software. It involves modularity, where components within a module exhibit high interdependency, while the modules themselves are independent. Complexity, which is the enemy of reliability, can be reduced through modularizing a system (for a more extended discussion of modularity in a different context, see Perrow 2009).

Building from an integrated design is, in many cases, cheaper and faster than modularity. There is no need for complicated interfaces between modules; there will be more common modes that reduce duplications of all kinds of inputs and components, and there are fewer assembly problems. If it also prevents competing applications from running on the system because of its integrated design, there are good reasons to prefer it. But it increases complexity, and thus allows the unexpected interaction of errors, and necessitates tight coupling, both of which can lead to "normal accidents" (Perrow 1999). Modular designs facilitate testing, since modules can be isolated and tested, then the interfaces of the modules tested with the modules they interact with, whereas integrated designs can only be tested by testing the system as a whole. Modularity promotes loose coupling, so that errors do not interact and cascade through the system.

Modularity also allows freedom for innovation within the module, irrespective of other modules or the system as a whole, as long as interface requirements are met. Modular designs make rapid product change easier, since the whole system does not need to be redesigned – something Microsoft has found to be very difficult and time-consuming. Most important, a hacker or terrorist who is able to penetrate a module – e.g., an application that floats on top of the OS – cannot as easily reach into the kernel of the OS, since the application or module is not integrated into the kernel but only connected to it by the interface, which can more easily be protected from an intruder. The denial-of-service (DoS) attack upon Estonia in 2007 was made possible because Microsoft software allowed intruders to establish *botnets* and make a DoS attack (Perrow 2007). It has occurred before; NATO was the target of a much smaller, but still disruptive attack in 1999, when it was fighting in Serbia. It is estimated that over half of the 330 million PCs in the US are infected with bots. But no discussion of these attacks seems to have made the connection between bot-vulnerability and Microsoft's integrated architecture.

Some authors argue that open source software is inherently more modular than proprietary software. Alan MacCormack et al. (2006) compared programs developed with open source software with those developed in proprietary systems. The former had fewer "propagation costs" – a measure which registers the extent to which a change in the design of one component affects other components. Open source software has a more modular architecture, largely because multiple users in different locations work on particular parts of it, rather than on the whole system. Proprietary systems are more strongly integrated, and are designed by a collocated team. MacCormack and associates compared products that fulfil similar functions, but were developed either by open-source- or closed-source developers. They found that changes in the former were limited to the module, whereas in the latter, the changes affected many more system components. The proprietary systems were thus less

adaptable when changes were made. The implication is that, when there are threats to functions in the system, such as attempts to penetrate or take the system over, the open-source programs will be more responsive in thwarting the threats and isolating them, although do not discuss this aspect (MacCormack et al. 2006).

While MacCormack et al. found that Apple's Macintosh system was indeed more modular than the proprietary systems they examined (they could not include Microsoft products because their kernels are not available for examination), the Mac was considerably less modular than open-source systems such as Linux. In one striking "natural experiment", they compared Mozilla, a proprietary system, before and after a major rewrite that was designed to reduce its complexity. The redesign managed to make it even more modular than a Linux system (MacCormack et al. 2007a). Thus, collocated teams can intentionally design in modularity , though modularity is more likely to be a product of an architecture that is iteratively designed by dispersed software writers. In other studies, MacCormack and associates (2007b) managed to match five examples of designs where they could compare the open source- and the proprietary products, and found striking support for their hypothesis that the distributed teams generated loosely-coupled systems, and single teams generated tightly-coupled ones. Tightly-coupled systems were more vulnerable to errors. Organizational structures are important in complex, tightly-coupled systems. As they put it in one paper: "Tightly coupled components are more likely to survive from one design version to the next, implying that they are less adaptable via the processes of exclusion or substitution; they are more likely to experience 'surprise' dependency additions unrelated to new functionality, implying that they demand greater maintenance efforts; and they are harder to augment, in that the mix of new components is more modular than the legacy design" (MacCormack et al. 2007b, p. 26).

Thus, it may be much more difficult to attack open-source systems than proprietary ones, unless the latter are explicitly modular in their architecture. A Congressional Research Service report on software and critical infrastructure stresses the vulnerability of using COTS products on otherwise secure and reliable systems (Wilson 2005). Unfortunately, it does not mention the source of most COTS products. The operating systems and programs are not likely to be Apple products – which account for less than five percent of the market – but are quite likely to be Microsoft systems, namely one of the many versions of Windows, which can be configured to run SAP- or IBM programs. But SAP or IBM programs are generally run on Linux or Unix systems. Even if the organization's computers are running on Linux or Unix, Windows Office applications can be adapted to run on systems such as Linux or Unix. Thus, a small part of the software in use in critical systems may compromise the much larger part, making Microsoft's software the "pointy end" of both the reliability-, and, as we shall see, of the security problem.

3 The Cyber-threat

Various actors have gained unauthorized access to nuclear-power plants and other power stations, financial institutions, corporations, intelligence agencies, and the U.S. Defense Department. As yet, we have not identified terrorists among them; they are more likely to be "hackers," corporations, or foreign nations. I have argued, but cannot prove, that the problem lies in insecure and faulty software, much of it from Microsoft.

An academic expert said in 2003, "There is little evidence of improvement in the security features of most [software] products; developers are not devoting sufficient effort to apply lessons learned about the sources of vulnerabilities [...] We continue to see the same types of vulnerabilities in newer versions of products that we saw in earlier versions. Technology evolves so rapidly that vendors concentrate on time to market, often minimizing that time by placing a low priority on security features. Until their customers demand products that are more secure, the situation is unlikely to change" (Wilson 2005, p. 65). Why is there a lack of demand for more secure products? There are several reasons.

(1) There is a substantial problem of information. Since about 80 percent of the breaches are not publicly announced, graded by threat intensity, and analyzed, it is difficult to know the extent of the problems, and who or what is at fault. At best, we get vague estimates of intrusions, etc., but little indication of their seriousness, and no indication of which software was running at the time. The victims, such as firms, are unwilling to disclose their failures for reasons of proprietary rights, reputation, and security.

(2) The field of software applications is evolving so fast that users are continually putting their operating systems and application programs to uses that were unforeseen by those who designed the product; it is impossible to anticipate just how a software program is going to be used, including the other programs it will interact with, intentionally or unintentionally. The problem of faulty or incomplete specifications is repeatedly noted in the literature on failures, and it applies to security as well, particularly when secure systems are linked to insecure programs running in the Internet.

(3) It is an article of faith in the software branch that the evident shortage of qualified programmers has led to "quick and dirty" training to meet the demand, without adequate private or public funds to increase the training's quality (Jackson et al. 2007). This, along with organizational production pressures, may account for a good bit of the sloppy software in existence. For some reason unclear to me, bright students in the U.S. have shunned engineering in general and programming in particular to the advantage of other fields, even though programming seems to be lucrative. It may have something to do with the general decline in mathematical literacy among young people.

(4) There is a market failure. When Microsoft gained control of the PC market, reliability was not a pressing concern; customers wanted features, and very few were running critical systems on their PC. Security was not a concern, because there was no Internet. When the Microsoft operating system expanded, the new versions had to be compatible with the older ones, retaining the unreliability and lack of security that became increasingly problematical. By the time Microsoft products were tied into our critical infrastructure, there was no incentive to bring out new products that addressed reliability and security concerns; these would have been incompatible with previous ones, and most important, the market for secure software was and still is quite small. The market failure is that, to hamper competition, the company persisted in using an architecture that made its product vulnerable to intruders, and since it had extensive market control almost from the start, competitors with less vulnerable products could not establish the critical mass of users or the easy interoperability necessary to increase their market share.

Why, then, if we have what was once called a "monoculture" problem, where 90 percent of operating systems, with all their faulty software, come from Microsoft, have we not seen vastly more bugs, crashes, penetrations resulting in economic fraud, denial of service, data theft, or damage to equipment in our critical infrastructure? Because the operating systems are only one part of the larger system. A widespread Internet worm, for example, may affect millions of computers, but so far has not affected the billions of users on the net. These billions have millions of different configurations of antivirus-protection software and, at different levels, they are running on different networks with different firewalls and router policies, and on different servers with different services. There is an inescapable modularity here. While we need a monoculture at the top, since we all need to use TCP/IP, HTML, PDF, and a lot of other devices, below that level there is far less interdependence than we might think. A hacker or a national state can easily shut down the email facilities of a small part of the U.S. Defense Department, but for the U.S. or Israel to target the centrifuges built by Siemens and used in Iran, or cause a nuclear-power plant meltdown, would require extraordinary resources, and would still be a single target. Breaking the Windows monopoly with cloud computing will certainly help reduce the intrusions that hackers, thieves, corporate spies, national states, and the still unrealized threat that terrorists pose, but the internet is still a remarkably robust distributed system.

4 Conclusion

Nowadays, the overriding problem has been the sheer size and complexity of the systems named above, making specifications regarding all possible interactions and uses nearly impossible. Simultaneously, certain software systems reach a precarious level of dissemination, not only in private households, but also in large technical systems. In my opinion, virtually the entire US critical infrastructure is at risk. It is ripe for a "normal accident" that threatens to have catastrophic consequences one of these days (Perrow 1999).

References

Bailor, C., 2006: For CRM, ERP, and SCM, SAP Leads the Way. Destination CRM.com. July 5; http://www.destinationcrm.com/articles/default.asp?ArticleID=6162 (download 28.10.11)

Campbell-Kelly, M., 2003. From Airline Reservations to Sonic the Hedgehog: A History of the Software Industry. Cambridge, MA

CERT, 2007: CERT Statistics. CERT; http://www.cert.org/stats/cert_stats.html (download 28.10.11)

Cusumano, M.A., 2004: The Business of Software. New York

Jackson, D.; Martyn, Th.; Millett, L.I., 2007: Software for Dependable Systems: Sufficient Evidence? Washington, DC: National Research Council

MacCormack, A.; Rusnak, J.; Baldwin, C.Y.; 2006: Exploring the Structure of Complex Software Designs: An Empirical Study of Open Source and Proprietary Code. In: Management Science 52 (2006), pp. 1015–1030

MacCormack, A.; Rusnak, J.; Baldwin, C.Y.; 2007a: Exploring the Duality Between Product and Organizational Architectures: A Test of the Mirroring Hypothesis. Working Paper 08-039, Harvard Business School; http://www.hbs.edu/research/pdf/08-039.pdf (download 28.10.11)

MacCormack, A.; Rusnak, J.; Baldwin, C.Y.; 2007b: The Impact of Component Modularity on Design Evolution: Evidence from the Software Industry. Working Paper 08-038, Harvard Business School; http://www.hbs.edu/research/pdf/08-038.pdf (download 28.10.11)

Perrow, C., 1999: Normal Accidents: Living with High Risk Technologies. Princeton, NJ

Perrow, C., 2007: Microsoft Attacks Estonia. In: Huffington Post May 26, 2007

Perrow, C., 2009: Modeling Firms in the Global Economy: New Forms, New Concentrations. In: Theory and Society 38 (2009), pp. 217–243; http://www.springerlink.com/content/x865g84476223212/ (download 28.10.11)

Rahman, H.A.; Beznosov, K.; Marti, J.R., 2006: Identification of Sources of Failures and their Propagation in Critical Infrastructures from 12 Years of Public Failure Reports; http://www.ece.ubc.ca/~rahmanha/cris2006_CS2_paper.pdf (download 28.10.11)

Weiss, J.M.; Cybersecurity Committee on Homeland Security's Subcommittee on Emerging Threats, and Science and Technology; U.S. House of Representatives, 2007: Control Systems Cyber Security. Washington, DC: U.S. Government. October 17, 2007

Wilson, C., 2005: Computer Attack and Cyberterrorism: Vulnerabilities and Policy Issues for Congress. Library of Congress. April 1, 2005

Contact

Prof. em. Charles Perrow
Yale University
Sociology Department
P.O. Box 20 82 65
New Haven CT 06520-8265, USA

« »