Tony's Law

Dror G. Feitelson
School of Computer Science and Engineering
The Hebrew University of Jerusalem
91904 Jerusalem, Israel

Somebody didn't tighten the lid, and the ants got into the honey again. This can be prevented by placing the honey jar in a saucer of water, but it's a nuisance, takes more counter space, and one must remember to replenish the water. So we try at least to remember to tighten the lid.

In the context of security, the software industry does not always tighten the lid. In some cases it fails to put the lid on at all, leaving the honey exposed and inviting. Perhaps the most infamous example of recent years is the WINvote voting machine, dubbed the worst voting machine in America. A security analysis by the Virginia Information Technologies Agency in 2015 found, inter alia, that the machines used the deprecated WEP encryption protocol, that the WEP password was hardwired to "abcde", that the underlying Windows XP (which had not been patched since 2004) administrator password was set to "admin" with no interface to replace it, and that the votes database was not secured and could be modified [7]. These machines had been used in real elections for more than 10 years.

Such cases constitute malpractice, and call for regulation. Regulation is necessary because not everything can be trusted to market forces. There are many examples in diverse industries. The sale of alcohol to minors is prohibited. Construction and housing cannot use asbestos and lead-based paints due to public health concerns. The automotive industry is required to install seat belts and report pollution levels. Aviation is strictly regulated, including airspace utilization (distances between planes), aircrew work schedules, aircraft noise levels, and more. Advertisers are required to add frightening warning labels on ads for cigarettes.

Computers are regulated in terms of electrical properties, such as the FCC regulations on radiation and communication. But the software running on computers is not regulated. Nearly forty years ago, in his Turing Award acceptance speech, Tony Hoare had the following to say about the principles that guided the implementation of a subset of Algol 60 [2]:

> The first principle was *security*. [...] A consequence of this principle is that every occurrence of every subscript of every subscripted variable was on every occasion checked at run time against both the upper and the lower declared bounds of the array. Many years later we asked our customers whether they wished us to provide an option to switch off these checks in the interests of efficiency on production runs. Unanimously, they urged us not to – they already knew how frequently subscript errors

Table 1: *Changes in software and computing in the last 30 years.*

1980s	**2010s**
C pointers	Java garbage collection
Emacs	Eclipse
Math library	Frameworks
Ad hoc programming	Agile methodology
Waterfall	Evolution / continuous integration
Flowcharts	UML
Write your own sort	Copy from Stack Overflow
Computer room	Computer in your pocket
Hard disk	Cloud
Text terminals	Touch screens
Email	Internet of things
No regulation	No regulation

> occur on production runs where failure to detect them could be disastrous. I note with fear and horror that even in 1980, language designers and users have not learned this lesson. **In any respectable branch of engineering, failure to observe such elementary precautions would have long been against the law**. [emphasis added]

He said this when personal computers and the Internet were in their infancy, long before the world-wide web, DDoS attacks, and data breaches. Indeed, a lot has changed during this time (Table 1). But one thing that has not changed is the lack of any meaningful regulation on the software industry.

In retrospect, Hoare's pronouncement exhibited great foresight. To this day buffer errors represent the single most common vulnerability[1], even more so among high-severity vulnerabilities (Figs. 1 and 2). Just imagine if a law requiring bounds checks had been enacted more than 40 years ago, and there were no buffer overflows today. As it stands, Microsoft for one instituted its Security Development Lifecycle as a mandatory policy in 2004. This includes — among many other features — the option to require compilation with flags that insert bounds checks and the option to ban unsafe library functions. On the one hand this demonstrates that such practices are just a matter of deciding to use them. On the other hand they are still not universally required, and indeed even Microsoft products still occasionally suffer from buffer issues[2].

Similar sentiments have been repeated several times since Hoare's speech. Twelve years ago David Patterson (then ACM's president) put forward the "SPUR manifesto" [3], suggesting that the development of 21st century computer (software) systems should focus on Security, Privacy,

[1]The NIST National Vulnerability Database uses 124 of the nearly 1000 types listed in the Common Weakness Enumeration to categorize vulnerabilities. In 2015–1017 Buffer errors CWE-119 accounted for 15.2–18.4% of all vulnerabilities each year. The next highest categories were Information leak/disclosure CWE-200 at 9.3–10.9%, Permissions, privileges, and access control CWE-264 at 8.2–10.0%, and Cross-site scripting CWE-79 at 7.3–11.2%.

[2]One recent example: Microsoft Office Equation Editor stack buffer overflow, 15 Nov 2017, www.kb.cert.org/vuls/id/421280.

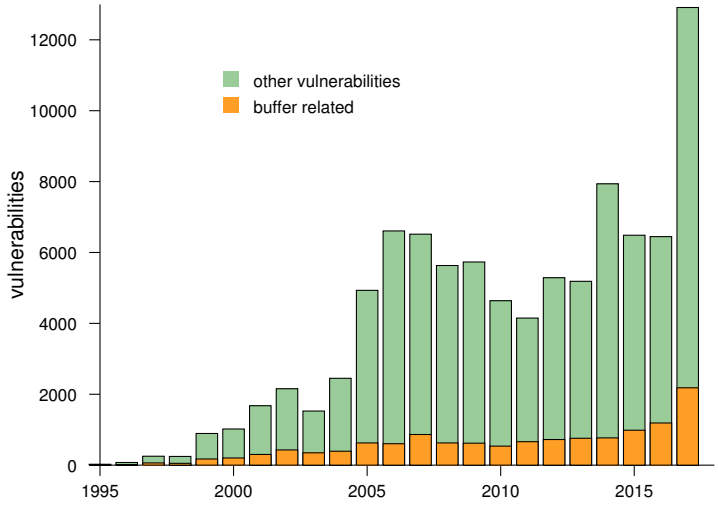

Figure 1: *The number of software vulnerabilities cataloged by the NIST National Vulnerability Database skyrocketed in 2017, and the fraction of vulnerabilities involving buffers (either categorized as "buffer error" or containing the keyword "buffer") kept pace.*

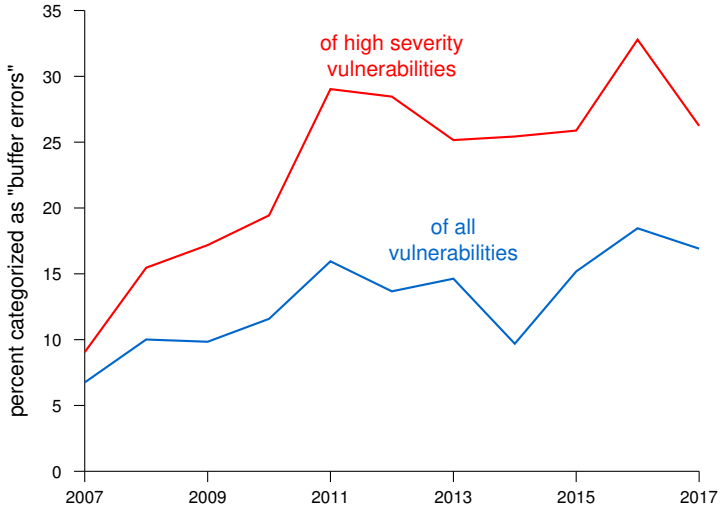

Figure 2: *According to the National Vulnerability Database, since the beginning of the decade around 15% of all vulnerabilities have been related to buffer errors, and this rises to between a quarter and a third of the vulnerabilities if only those with a high severity score are considered*

.

Usability, and Reliability. The goal should be to be as safe as 20^{th} century banking, as low-maintenance as 20^{th} century radio, and as reliable as 20^{th} century telephony. But twelve years have passed, and it seems that the focus on low cost, multiple features, and above all time to market is as strong as ever. Manufacturers of home appliances compete, inter alia, by offering superior warranties for their products. The software industry, in contradistinction, has been getting away with software that comes "without warranty of any kind, expressed or implied, including, but not limited to, the implied warranties of merchantability and fitness for a particular purpose".

Indeed, lectures such as Patterson's are typically either ignored, or else they stir up a chorus of naysayers. The typical arguments are the perceived monetary costs, the difficulties or even the impossibility of implementation, and the fear of reduced innovation and technological progress. Schneider, in a recent viewpoint, also notes the need for a detailed cost/benefit analysis to ascertain what society is wiling to pay for improved security, where the costs also include reduced convenience (due to the need for authentication) and functionality (due to isolation) [4]. And indeed all regulations are, by definition, limiting. But do we really need to wait for a large-scale security catastrophe, possibly including significant loss of life, before we act at all? As the Microsoft example shows, extensive technological solutions and best practices actually already exist. It is just a matter of making their use pervasive.

So why are software security faults tolerated? A possible explanation is that software deficiencies have so far been less tangible than those of traditional industries. Many people install multiple locks on their doors and would consider holding intruders to their homes at gunpoint, but fail to take sufficient safeguards to protect their home computers from hackers. The problems resulting from identity theft are much more common but also much more bureaucratic, boring, and less visual relative to the problem of exploding gas tanks in pickup trucks.

But above all else, it seems that there is a market failure in incentivizing the industry to take the required actions [1, 6]. Buyers will not pay a premium for value (security) they cannot measure, and which in many cases does not affect them personally and directly. Approaches suggested by economists to measure the value of protection don't help because the cost of a security catastrophe is up to anyone's imagination. This has prevented an insurance industry for software producers from emerging, and as Anderson and Moore write, "if this were the law, it is unlikely that Microsoft would be able to buy insurance" [1]. In practice, the reduction in stock value after disclosing a vulnerability is less than 1% [5]. The abstract danger of large-scale attacks leading to financial loss and even loss of human life is not enough to change this.

At the same time, we are inundated by increasing numbers of reports of data breaches and hackers infiltrating into various systems (see Table 2 for prominent recent examples). Some of these incidents demonstrate that extensive physical civil infrastructures are at peril across the globe — including hospitals, power plants, water works, transportation systems, and even nuclear facilities. And the root cause at least in some cases is the failure of the software to take appropriate precautions.

The software systems in a modern car — not to say a passenger plane or a jet fighter — are of a scope and complexity that rivals any operating system or database produced by the traditional software industry. Indeed, every industry is now a software industry. And the products of every

Table 2: *Notable security incidents from the last decade.*

Year	*Incident*	*Significance*
2007	Massive DDoS attacks on organizations and infrastructure in Estonia	First demonstration of extensive country-wide disruptions, possibly in connection to international relations
2010	The Stuxnet cyber-weapon is used to disable physical centrifuges used in Iran's nuclear program	Demonstration of potential impact on computer-controlled physical infrastructure, and demonstration of cyber-weapons that jump air-gaps and remain undetected for long periods
2013	Yahoo is hacked and data about all 3 billion user accounts is stolen	Biggest data breach of its kind
2016	Hackers break into DNC computers and disseminate confidential documents	Strategic hacking with possible effect on the outcome of the US presidential elections
2016	DDoS attacks using a botnet of some 1.5 million IoT devices (ironically, mainly security cameras)	Demonstration of new vulnerabilities resulting from technological progress and insufficient consideration of security
2017	The WannaCry ransomware infects more than 200,000 computers in 150 countries, causing disruptions such as the closing down of 16 hospitals in the UK	Demonstration of global-scale cyber crime and putting human lives at risk

industry are vulnerable due to software defects. In such a context, required software regulation includes:

- Transparency: the obligation to report all exploits including their technical details.
- The prohibition of dangerous practices, such as not using type-safe languages and appropriate encryption.
- Holding companies accountable for their unsafe practices.

These requirements need the backing of legal regulations, because market forces compel industry not to invest in security too much. The market promotes a race to the bottom: except in niche applications, whoever is faster to market and cheaper wins, and whoever is tardy due to excessive investment in security loses. Regulation is the only way to level the playing field, forcing everybody to invest in what they know to be needed but think they can't afford to do when the competition doesn't.

Of course, it won't be easy to implement these ideas and agree on the myriad details that need to be settled. Who gets to decide what is a "dangerous practice"? How do we deal with installed

systems and legacy code? Who is charged with enforcing compliance? Moreover, it is not clear how to make this happen at the political level. And in addition, no single country has jurisdiction over all software production. So a system of certification is required to enable software developers to identify reliable software, and to perform due diligence in selecting what other software to use.

International frameworks already exist that show that these issues can be solved. The EU General Data Protection Regulation (GDPR), which concerns the rights of individuals to control how their personal information is collected and processed, is an encouraging example. Another example is the Common Criteria for Information Technology Security Evaluation, an international framework for the mutual recognition of secure IT products. But this covers only high-level desiderata for security, not the regulation of low-level technicalities. This gap is partly filled by the Motor Industry Software Reliability Association (MISRA), which has defined a set of suggested safe coding practices for the automotive industry. However, these are not required by any formal regulations.

Protracted discussions on what to do and what we are willing to pay for it are counterproductive. Such things can not be planned in advance. Instead we should learn from the iterative approach to constructing software: try to identify the regulations that promise the highest reward for the lowest cost, work to enact them, learn from the process and the results, and repeat.

On the bottom line, regulation is in the interest of the long-term prosperity of the software industry no less than in the interest of society as a whole. Software vendors with integrity should stop resisting regulation and instead work to advance it. The experience gained will be all-important in discussing and enacting further regulations, both in a preemptive manner and, God forbid, in the aftermath of a security catastrophe.

References

[1] R. Anderson and T. Moore, "*The economics of information security*". *Science* **314(5799)**, pp. 610–613, 27 Oct 2006, DOI: 10.1126/science.1130992.

[2] C. A. R. Hoare, "*The emperor's old clothes*". *Comm. ACM* **24(2)**, pp. 75–83, Feb 1981, DOI: 10.1145/358549.358561.

[3] D. A. Patterson, "*20^{th} century vs. 21^{st} century C&C: The SPUR manifesto*". *Comm. ACM* **48(3)**, pp. 15–16, Mar 2005, DOI: 10.1145/1047671.1047688.

[4] F. B. Schneider, "*Impediments with policy interventions to foster cybersecurity*". *Comm. ACM* **61(3)**, pp. 36–38, Mar 2018, DOI: 10.1145/3180493.

[5] R. Telang and S. Wattal, "*An empirical analysis of the impact of software vulnerability announcements on firm stock price*". *IEEE Trans. Softw. Eng.* **33(8)**, pp. 544–557, Aug 2007, DOI: 10.1109/TSE.2007.70712.

[6] M. Y. Vardi, "*Cyber insecurity and cyber libertarianism*". *Comm. ACM* **60(5)**, p. 5, May 2017, DOI: 10.1145/3073731.

[7] Virginia Information Technologies Agency, "*Security assessment of WINvote voting equipment for department of elections*", 14 Apr 2015. https://www.wired.com/wp-content/uploads/2015/08/WINVote-final.pdf.