How to Win an Evolutionary Arms Race

Anil Somayaji
Carleton University

Just about every computer user today is engaged in an evolutionary arms race with virus writers, spam distributors, organized criminals, and other individuals attempting to co-opt the Internet for their own purposes. These adversaries are numerous, adaptable, persistent, and unscrupulous. They're creating independent agents—mobile programs—that, once released, take on lives of their own. Their attacks are becoming more sophisticated every day, and the situation will likely become much worse unless we, as defenders, take drastic steps.

We can't hope to completely defeat all our attackers. For every individual we arrest, every worm we successfully eradicate, another virus writer and his or her program will propagate on the Internet. In an evolutionary arms race, the best we can hope for is to stay in the game no matter how the adversaries change and adapt. This game is ultimately one of survival, and so far, our computers aren't very good at playing it on their own.

To keep up with malware writers, software producers in both the commercial and open-source software worlds have adopted various automatic software update mechanisms. Some of these mechanisms distribute updates after requesting a user's permission; others install updates automatically. Although such systems provide some short-term relief, they will likely soon become ineffective, and further, they will also become extremely dangerous once they are inevitably co-opted by attackers. If we want the Internet to remain a viable way to communicate and collaborate, we must adopt another, perhaps radically different, model for securing our computers.

To better understand this conclusion, we should first re-examine why developers and users are embracing automated update systems.

Faster, easier updates

Automated update mechanisms are designed to solve one simple problem: humans can't be counted on to download and install anything in a timely fashion. Home users forget to update their systems (if they know to do it at all), and administrators already have too much to do. Yet, malicious code can spread across the Internet in minutes. Thus, software companies such as Microsoft, Symantec, and Red Hat have created systems that remind users to initiate automated update processes or that bypass the user completely and install necessary updates on their own.

To date, almost every virus or worm epidemic has exploited a security vulnerability for which there was an existing software patch or virus signature. However, before software developers can distribute a fix, they must identify the problem, craft a solution, and test that solution to ensure no untoward side effects. If McAfee makes a mistake with a virus signature, it could disable or even delete legitimate users' applications (legitimate applications have already been disabled by virus scanners; see http://news.com.com/2100-7350-5361660.html). If Microsoft makes a mistake, numerous machines might no longer even boot. The time between vulnerability disclosure and exploit distribution is sometimes only a few days. If software developers haven't been previously notified, there's little chance that they can release a well-tested fix in this time frame. Further, with fast-propagating worms and viruses, antivirus developers must play catch-up: Malware can infect thousands of machines before the causative agent is even identified.

So, while automated update systems solve one problem (available updates not being distributed), they don't address the more fundamental problem—developing and testing those updates. Although antivirus developers have automated the signature creation and testing process somewhat, they're hampered in their quest to detect zero-day worms by concerns over false positives. Software patches can potentially stop novel viruses and worms. However, patches must still be developed manually, and because they're complex and intrusive, they're likely to cause problems. Corporate IT departments have traditionally required extensive internal testing of any updates before deployment precisely because of these types of issues. Unfortunately, because of the frequency and urgency of updates, corporations no longer have this luxury.

Although the current situation

isn't good, things could conceivably become much worse. What if attackers discovered new vulnerabilities once a day? Once an hour? Once a minute? Malware developers can achieve such a rate of discovery by working in parallel and automating vulnerability discovery and exploitation. Because developers must extensively test each update to ensure that it doesn't break existing functionality, they can't deploy fixes at the same rapid rate. In the long run, developers won't be able keep up with attackers as long as they're required to respond to each vulnerability as it's discovered.

Several technologies exist, such as nonexecutable stacks and anomaly-based intrusion prevention systems, that promise to prevent exploits without signatures or software patches. Many of these systems have considerable merit. Experience teaches us, however, that attackers can circumvent or even directly exploit such systems. Software patches, of course, can be created to fix specific limitations, but then we're back where we started.

We're faced with adversaries that can potentially deploy attacks faster than we can deploy defenses, even if we use automated update systems. Advanced defense technologies can help, but because they're imperfect and therefore vulnerable to exploitation, they don't change the overall balance of power. Is there anything we can do, then, to even the scales—to play this evolutionary game so that we have a chance to keep up?

How animals play the game

Although our computer systems are new at this game, living organisms have been fighting such battles for millions of years. In particular, large animals such as humans face a threat environment analogous to that our computer systems face. Our ancestors survived in a world filled with numerous pathogens such as bacteria, viruses, and parasites—all without help from modern medical science. To be sure, the human body has formidable perimeter defenses such as skin, mucous membranes, and stomach acid. These barriers are complemented by a complex immune system, part of which is adept at handling most common threats (the "innate" immune system), and an adaptive system that can handle most other invaders. Even with these formidable defenses, though, people still get sick, and many die of disease. Furthermore, new threats constantly arise that defeat old defenses: Bacteria develop antibiotic resistance, and viruses such as HIV emerge that our immune systems can't defeat.

Despite the magnitude of these threats, our bodies go to extraordinary lengths to preserve genome integrity. Germ-line cells—those that eventually form eggs and sperm—are segregated during early embryonic development. The DNA in many of our immune-system cells changes in response to various pathogens we're exposed to, but we pass none of these changes to the next generation. Such "code conservatism" has probably arisen because our genomes are rather fragile: Duplicated or deleted chromosomes frequently cause severe birth defects, and even a one-letter (single nucleotide) mutation in a critical section of an embryo's DNA can cause a miscarriage. Bacteria can tolerate large-scale changes in their genomes—they can lose entire genetic subsystems (plasmids) and acquire new ones from other bacteria or the external environment. The complexity of most animal genomes seems to preclude such flexibility.

All animals evolve over time. Those better able to survive and reproduce pass on their genes, and those who can't adapt are eliminated. The opposition is also evolving, though, and they do so much more rapidly. Humans reproduce every 20 years, but bacteria can reproduce every 20 minutes.

Given such numerous and adaptable foes, it would seem that all large animals, including humans, should have become extinct long ago. Fortunately, nature has devised strategies that balance the scales by ensuring that we don't give pathogens a "stationary target." In particular, we've survived using three basic moves: sex, death, and speciation. Confused? Well, consider the following.

Sexual reproduction

Many researchers have observed that diverse populations are more resistant to infection than populations of similar individuals. Less well appreciated is the fact that the primary mechanism animals use to generate and preserve such diversity is sexual reproduction. Using a systematic mechanism for selecting and combining "code" from two parents to create offspring, animals can create novel variants without resorting to random code changes. Code segments, or DNA, that are particularly fragile tend to be identical in both parents, so the recombination process won't perturb these parts. On the other hand, differences between the parents are chosen randomly and passed on to the child. Sex produces such diversity that—except for identical siblings—no two individuals born have ever had the same genome.

We're faced with adversaries that can potentially deploy attacks faster than we can deploy defenses, even if we use automated update systems.

Preprogrammed death

Although we might not like it, death is a useful and probably necessary evolutionary adaptation for complex animals. A natural ecosystem can only accommodate a limited number of individuals from a given species. If these limits are simply determined by external factors (for example, amount of food, rate of predation and disease, and so on), successful animals will choose to reproduce very infrequently to reduce competition. The genomic composition of such a population, though, tends toward a "temporal monoculture"—a set of genomes that, while diverse, changes little over time. When a pathogen or predator eventually develops an effective attack, it will have done so observing this static population; so, it's more likely that the attack (or set of attacks) will kill most of the species. Even worse, the survivors will be less able to repopulate the ecosystem because they've evolved to reproduce at a slow rate. Thus, to ensure that species shuffle around their genes on a regular basis, we all have a built-in self-destruct mechanism that makes room for new variants—our children.

Speciation

These powerful mechanisms aren't always enough, however. Flaws present in all members of a given species must be preserved to ensure mating ("backward") compatibility, but pathogens might eventually find and exploit these flaws. For life to continue, there must be a way to fix such basic weaknesses. Speciation is that mechanism. The conventional wisdom in biology is that the combination of mutation and a reproductive barrier are enough to create a new species (see http://evolution.berkeley.edu/evosite/evo101). Researchers such as Lynn Margulis now argue, though, that most new species develop through the merging of functional genetic units—genomes.[1] Her theory of symbiogenesis is too complicated to explain fully here, but from a programming perspective it's almost painfully obvious: It's simply much more likely that new programs with novel functionality will arise from the merger of two separate, working programs than from a series of random changes to a single program.

Software development's biological nature

Many parallels exist between biology and current software practices. For example, we can view merging, splitting, and recombining code in open-source projects such as the Linux kernel as a kind of "sexual recombination." Code forks that result in similar yet incompatible software systems (for example, the split between the various BSD descendents) are a kind of speciation. And the efforts commercial vendors make to phase out old software versions are (often unsuccessful) attempts at "killing" certain software varieties.

Clearly, we're seeing limited forms of software sex, death, and speciation. However, to produce a sufficiently diverse and dynamic software ecosystem that's resistant to malicious software, we would have to accelerate and streamline these processes. Such radical changes would almost certainly bring some pain in the short term. Even worse, we couldn't guarantee that malware authors wouldn't adapt to these techniques as well—after all, we wouldn't be solving the "secure composition" problem because under such a regime, each individual program would have just as many vulnerabilities as it would otherwise, if not more.

Consider, though, that this approach's fundamental premise is that security through obscurity can be effective if a sufficient amount of obscurity exists and if the nature of the obscurity changes frequently. The real question is, can we create mechanisms that transform vulnerabilities so that an attacker would need specific knowledge—a key, in effect—to craft a successful exploit? Work in software diversity and obfuscation[2,3] has been somewhat successful in achieving this goal. This work, however, has been limited by the assumption that transformations shouldn't change programmer-specified code behavior. We should be able to draw inspiration from genetic algorithms. The challenge is to find automated methods for exchanging code between applications that can also simultaneously preserve the important functionality of each.[4]

Although adapting the concepts of sex, death, and speciation to computing is risky, the simple truth is that automated update systems and other security technologies won't be able to protect computer systems from all malicious software. Perhaps by changing the game we can ensure that the Internet remains a resource for everyone—not just the bad guys. □

References

1. L. Margulis and D. Sagan, *Acquiring Genomes: A Theory of the Origins of Species*, Perseus Books Group, 2002.
2. E.G. Barrantes et al., "Randomized Instruction Set Emulation to Disrupt Binary Code Injection Attacks," *Proc. 10th ACM Conf. Computer and Communication Security* (CCS 03), ACM Press, 2003, pp. 281–289.
3. S. Bhatkar et al., "Address Obfuscation: An Efficient Approach to Combat a Broad Range of Memory Error Exploits," *Proc. 12th Usenix Security Symposium*, Usenix, 2003, pp. 105–120.
4. M. Mitchell, *An Introduction to Genetic Algorithms*, Bradford Books, 1996.

Anil Somayaji *is an assistant professor in the School of Computer Science at Carleton University. His research interests include computer security, operating systems, complex adaptive systems, and artificial life. He received a PhD in computer science from the University of New Mexico. He is a member of the ACM, the IEEE Computer Society, and Usenix. Contact him at soma@scs.carleton.ca.*