Engineering Challenges in Immunizing Systems Against Malicious Code

Craig Chamberlain, Daniel Geer Sc. D., Verdasys, Inc.
(craig / geer) @verdasys.com

The Problem

In the absence of a strong generalized defense, biologic hosts must experience an infection to develop antibodies to it. Inherently reactive, this offers little or no resistance to a previously unknown pathogen. For that, the organism cannot rely on time-lapsed development of pathogen-specific antibodies, but must instead be able to immediately disambiguate "self" and "not-self," which Forrest, et. al, (1996) were perhaps the first to attempt adapting to the digital world. Our challenge, as engineers, is to make such determinations on the fly in a way that is both effective, prompt, and non-interfering with needed work output of people and/or machines.

Background (Optional)

Self-replicating mobile code appeared in the 1970s becoming widespread in the 1980s. Frederick Cohen introduced the term computer virus in 1984, defining it as "a program that can 'infect' other programs by modifying them to include a possibly evolved copy of itself." Viruses spread by inserting code into other programs such that the virus code executes along with the host program. Peter Szor (2005) defines a worm as a virus which replicates across networks and generally does not need to [and may never] infect a host file. A Trojan Horse is a program that appears legitimate while secretly doing something else. Gasser (1988) notes the Trojan horse problem was identified remarkably late in the development of computers and he and Thimbeleby, et. al, (1998) separately discuss how Trojan Horse detection and prevention remains fundamentally unsolved.

Strictly speaking, perfect detection of malicious programs is unsolvable -- it is undecideable. Cohen (1987) proved that no single algorithm can detect all virus programs which might exist, and separately (1994) showed that access control is ineffective against malicious code. Firewalls and network layer defenses are similar, but cannot be perfect and for the same reasons. Szor (2005) and Chess & White (2000) both point out that there is a threshold of code mutation beyond which viruses can exist but which no algorithm can detect. While much good has come from a variety of pattern matching and heuristic code scanners, nevertheless they can only disinfect what they are trained to identify. File integrity checking can reliably detect changes to file systems but at the cost of potentially high numbers of false positives, e.g., due to

patching and self-modifying programs such as those using run-time packers. Grayaznov (1999) notes that integrity checking tools are themselves targeted by slow infector viruses, which wait for files to be modified and then piggyback their infection onto the legitimate modification thus to escape notice.

As described by Szor (2005) , first- and second-generation virus detectors used a variety of code and pattern matching techniques and, later, algorithmic detection in a custom language. Next was CPU emulation techniques to detect polymorphic viruses, which partially mutate their own code in order to frustrate pattern matching scanners, and Szor notes the significant challenge posed by polymorphic viruses that use entry point obscuring techniques (effectively randomizing their location within the infected host file in order to make detection more difficult). Szor also notes that some metamorphic viruses, which produce true mutations that do not resemble the parent, would require a pattern matching against an infeasible number of patterns. Christodrescu, et. al, (2003) found that even simple code obfuscation techniques, such as inserting NOP instructions defeated commercial virus scanners.

There are alternatives to pattern matching. Szor describes geometric detection (examining changes to file structures), heuristic analysis and behavior blocking tend (which generate false positive and false negatives). Neural networks can produce acceptable rates of false positives, but require considerable training, are subject to overtraining, and tend toward unacceptable false positive ratios when malicious code closely resembles non-malicious code.

Intrusion prevention systems (IPS) have scalability problems including administration and configuration costs. Like intrusion detection, intrusion prevention technologies face significant technical challenges in achieving acceptable rates of false positives and false negatives, especially when an IPS intervenes in the application space to block behavior that may be malicious. The complexity of modern operating system and application behavior make it especially difficult to protect against a newly emerged threat for which the IPS agent has no training.

Meeting the Challenge

Specifically, confront the problem of malicious code under these practical requirements:

1. Prevent de novo infection by malicious programs including viruses, worms and Trojan horse programs.

2. ...without first obtaining new code, virus definitions, signature files, or data.

3. ...consistent with intermittent connectivity including no facility to download data files.

4. Maximally generic and thus independent of specific pattern matching or algorithmic methods of code examination and classification.

5. Be configurable for mandatory enforcement, i.e., disallowance of override once installed.

6. Be nevertheless manageable at enterprise scale and with a constant flux of changes and adds to people and facilities.

7. Fail closed.

We describe the regulation of file and network subsystem calls using a host agent that acts as a kernel level security tool in the spirit of Anderson's (1972) Reference Monitor, specifically an inescapable and invisible event trap for all kernel-level operations involving data, broadly defined. Events, once detected may be logged, may trigger a broad range of actions, or may be discarded as chaff. This method has the substantial advantage of scale-independence relative to the number of attack methods with which it may come in contact, and as a consequent its rule set is small and needs no reactive updates to new threats. The solution is real, has been found to perform well in laboratory and field trials. As with any practical COTS product, it does not do everything, e.g., our method does not prevent purely in-memory threats which generate no I/O activity. We believe that as with all engineering, the right problem statement and the right abstraction tend to produce the best results. We will share the results of our field trials.