Natural Selection Considered Harmful

Olivia Lucca Fraser

2019-02-27

Abstract

A brief history of those few times where computer virus design and evolutionary computation have crossed paths.

While evolutionary techniques have been more or less frequently employed in the field of *defensive* security – where they are put to work much in the same way as other machine learning algorithms, and built into next-generation firewalls, intrusion-detection systems, and so on – there has been far less exploration of these techniques in the realm of offensive security. This is not to say, however, that the idea has never occurred to anyone – the idea seems to have captured the imagination of hackers, malware engineers, and cyberpunk science fiction authors, ever since there have been such things.

1969: Benford

The earliest occurrence of the concept of evolving, intrusive code that I was able to excavate dates to sometime around 1969, in an experiment performed – and subsequently extrapolated into fiction – by the astrophysicist and science-fiction author, Gregory Benford, during his time as a postdoctoral fellow at the Lawrence Radiation Laboratory, in Livermore, California. "There was a pernicious problem when programs got sent around for use: 'bad code' that arose when researchers included (maybe accidentally) pieces of programming that threw things awry," Benford recalls of his time at the LRL.

> One day [in 1969] I was struck by the thought that one might do so intentionally, making a program that deliberately made copies of itself elsewhere. The biological analogy was obvious; evolution would favor such code, especially if it was designed to use clever methods of hiding itself and using others' energy (computing time) to further its own genetic ends. So I wrote some simple code and sent it along in my next transmission. Just a few lines in Fortran told the computer to attach these lines to programs being transmitted to a certain terminal. Soon enough – just a few hours – the code popped up in other programs, and started propagating.

Benford's experiments unfolded in relative obscurity, apart from inspiring a short story that he would publish in the following year, entitled "The Scarred Man". As far as we can tell, however, the invocation of "evolution" remained entirely analogical, and did not signal any rigorous effort to implement Darwinian natural selection in the context of self-reproducing code. It was nevertheless an alluring idea, and one that would reappear with frequency in the young craft of virus programming.

1985: Cohen

Though anticipated by over a decade of scattered experiments, the **concept** of "computer virus" made its canonical entrance into computer science in the 1985 dissertation of Fred Cohen, at the University of Southern California, *Computer Viruses Computer Viruses* is a remarkable document. Not only does it provide the first rigorously formulated – and *formalized* – concept of computer virus, which Cohen appears to have discovered independently of his predecessors (whose work was confined to obscurity and fiction), explore that concept at the highest possible level of generality, in the context of the Turing Machine formalism, develop an elegant order-theoretic framework for plotting contagion and network integrity, leverage language-theoretic insights to subvert then-hypothetical anti-virus software through Gödelian diagonalization, and suggest a number of defenses, such as the cryptographic signing of executables, which are still used today, it also hints – elliptically – at the potential for viral evolution. At first glance, what Cohen calls the *evolution* of a virus resembles what would later be called *polymorphism* or even *metamorphism* – the process of altering the *syntactic* structure of the pathogen in the course of infection, so that the offspring is not simply a copy of the parent. This is indeed enough to expose the virus to a certain amount of differential selective pressure, so long as antiviral software (the virus's natural predator) pattern matches on the virus's syntactic structure (the precise sequence of opcodes used), or on some low-level features on which the syntax supervenes (one or more bitwise hashes of the virus, for example). But Cohen goes a step further than this, and considers a far broader range of infection transformations that do *not* preserve semantic invariants. That is to say, he considers reproduction operators – operators embedded in the virus itself, which, following Spector we can call "autoconstructive operators" – which generate semantically dissimilar offspring.

Cohen thus deploys all the essential instruments for an evolutionary treatment of viruses:

1. reproduction with variation (the "genetic operators")
2. selection (detection by recognizers, or "antivirus" software)
3. differential survival (there is no recognizer that can recognize every potential virus, as a corollary of Rice's theorem)

He goes no further in systematizing this dimension of the problem, unfortunately, and nowhere in this text do we find anything that either draws on or converges

with contemporaneous research into evolutionary computation as a mechanism for program discovery or artificial intelligence.

Cohen can hardly be blamed for this, of course. The dissertation as it stands is a work of rare ambition and scope. The casual observer of virus research and development over the past three decades, however, might be surprised by the impression that so little has been done to bridge the distance that lay between it and study of evolutionary computation. While the rhetoric surrounding the study of computer viruses remained replete with references to evolution, to ecology, to natural selection, and so on, efforts to actually integrate the two fields appear to have been rare.

This impression is not wholly accurate, however. Closer study shows us that the experimental fringe of the vx scene has indeed retained an interest in exploring the use of genetic methods in their work. If this has gone relatively unnoticed by the security community, this is likely for one or two reasons:

1. the vxers who have implemented genuinely evolutionary methods in their work seem to be motivated primarily by hacker's curiosity and not by monetary gain. The viruses they write are intended to be more playful than harmful, and it appears that several of the evolutionary viruses I have found were sent directly by their authors to antivirus researchers, or published, along with source code and documentation, on publicly accessible websites and vxer ezines.

2. Of course, we should consider the non-negligible selection effect implied in reason #1: it's not surprising that the viruses *that I was able to find* in the course of writing this chapter are those circulated by the grey-hat vxer community, as opposed to those developed, or contracted, by intelligence agencies and criminal syndicates, who tend to hold somewhat more stringent views on matters of intellectual property. And so a second, plausible-enough explanation presents itself: it is possible that far less playful evolutionary viruses *do* exist in the wild, but that they tend to either go undetected, are used primarily for targetted operations less exposed to the public, or that they are not being properly recognized or reported in the security bulletins released by the major antivirus companies.

Nonheritable Mutations in Virus Ontogeny

For reasons of stealth, virus writers have explored ways of incorporating variation into their mechanisms of infection and replication. The first trick to surface was simple encryption, employed for the sake of obfuscation rather than confidentiality. This first became widely known with the Cascade virus, circa 1988. Viruses using this obfuscation method would encrypt their contents using variable keys, so that the bytewise contents of their bodies would vary from transmission to transmission. The encryption engine itself, however, would remain unencrypted and exposed, and so antiviral software simply looked for

recognizable encryptors instead.

Next came oligomorphic viruses, starting with [Whale](https://en.wikipedia.org/wiki/Whale_(computer_virus in 1990, would use one of a fixed set of encryption engines, adding some variability to the mix. This would make the problem of detection some 60 or 90 times harder, depending on the number of engines, but such distances are easily closed algorithmically.

Next came polymorphic engines, which would scramble and rebuild their own encryption engine with each transmission, while preserving all the necessary semantic invariants. The antivirus developers countered by running suspicious code in emulators, waiting until the body of the virus was decrypted before attempting to classify it.

The last and most interesting development in this (pre-genetic) sequence rests with *metamorphic* viruses, which redirected the combinatorial treatment that polymorphics reserved for the encryption engine onto the virus body as a whole. There was no longer any need for encryption, strictly speaking, since the purpose of encryption in polymorphism is to obfuscate, not to lock down, and this allowed viruses to avoid any reliance on the already somewhat suspicious business of decrypting their own code before running.

In biological terms, what we're seeing with both polymorphic and metamorphic viruses is a capacity for ontogenetic variation. While it is possible for the results of metamorphic transformations to accumulate over generations, in most cases (unless there are bugs in the metamorphic engine), these changes are semantically neutral, and do not affect the functionality of the code (though this raises a subtle point regarding what we are to count as 'functionality', especially when faced with detectors that turn syntactic quirks and timing sidechannels into a life-or-death matter for the virus).

It is nevertheless evident how close we are to an actual evolutionary process.

2002: MetaPHOR

In 2002, Mental Driller developed and released a virus that bridged the gulf between metamorphic viruses and a new variety of viruses that could be called "genetic". MetaPHOR is a highly sophisticated metamorphic virus, capable of infecting binaries on both Linux and Windows platforms. Written entirely in x86 assembly, it includes its own disassembler, intermediate pseudo-assembly language, and assembler, as well as a complex metamorphic and encryption engines. Its metamorphic engine mutates the code body through instruction permutation, register swapping, 1-1, 1-2, and 2-1 translations of instructions into semantic equivalents, and the injection of 'garbage code', or what those working in the field of genetic programming call "semantic introns".

But the final touch is the use of a simple genetic algorithm, which is responsible for weighting the probabilities of each metamorphic transformation type. As Mental Driller comments in the MetaPHOR source code:

> I have added a genetic algorithm in certain parts of the code to make it evolve to the best shape (the one that evades more detections, the action more stealthy, etc. etc.). It's a simple algorithm based on weights, so don't expect artificial intelligence :) (well, maybe in the future :P).

The way it works is that each instance of the virus carries with it a small gene sequence that represents a vector of weights – one for each boolean decision that the metamorphic engine will make when replicating and transforming the virus, in the process of infection. These are modified a little with each replication. The hope is that the selective pressure imposed by antiviral software will select for strains of the virus that have evolved in such a way as to favour transformations that evade detection, and shun transformations that give the virus away. (Descendants of the virus, for instance, may adapt in such a way as to never use encryption, if that should turn out to a tactic that attracts the scanners' attention, in a given ecosystem. Or they may evolve to be less aggressive in infecting files on the same host, or filter their targets more carefully according to filename.)

2004-2005: W32/Zellome

The frequent invocation of ecological and evolutionary tropes in virus literature, combined with the lack of any genuine appearance of evolutionary malware, has led many to speculate as to its impossibility. The most frequently cited reason for the unfeasibility of viral evolution is computational brittleness – the claim being that the machine languages (or even scripting languages) that most viruses are implemented in are relatively intolerant to random mutation. The odds that a few arbitrary bitflips will result in functional, let alone 'fitter', code is astronomically small, these critics reason. This is in contrast to the instruction sets typically used in GP and ALife, which are *designed* to be highly fault-tolerant and evolvable.

This is so far from being an insuperable obstacle that it suggests its own solution: define a more robust meta-grammar to which genetic operators can be more safely applied, and use those higher-level recombinations to steer the generation of low-level machine code.

We can find this idea approximated in a brief article by ValleZ, appearing in the 2004 issue of the vxer ezine, *29A*, under the title "Genetic Programming in Virus". The article itself is just a quick note on what the author sees as interesting but in all likelihood impractical ideas:

> I wanna comment here some ideas i have had. They are only ideas… these ideas seems very beautiful however this seems fiction more than reality.

ValleZ goes on to sketch out the main principles behind genetic programming, and then gets to the crux of the piece: "how genetic programming could be used

in the virus world".

As already noted, most of the essential requirements for GP are already present in viral ecology: selective pressure is easy to locate, given the existence of antiviral software, and replication is a given. However, ValleZ notes, the descendant of a virus tends to be (semantically) identical to its parent, and even when polymorphism or metamorphism are used, the core semantics remain unchanged, and there is no meaningful accumulation of changes down generational lines.

ValleZ suggests the use of genetic variation operators – mutation, and, perhaps, in situations where viruses sharing a genetic protocol encounter one another in the same host, crossover – in virus replication. They would take over the work that is usually assigned to polymorphic engine, with the added, interesting feature of generating enough semantic diversity for selective pressures to act on. But for this to work, they note, it would be necessary to operate not on the level of individual machine instructions (which are, as noted, rather brittle with respect to mutation) but higher-level "blocks", envisioned as compact, single-purpose routines that the genetic operators would treat as atomic.

The idea is left only barely sketched out, however, and ValleZ concludes by reflecting that it seems more an idea "for a film than for real life, however i think its not a bad idea :-m".

In 2005, an email arrived in the inbox of the virus researchers Peter Ferrie and Heather Shannon. Attached was a sample of what would go on to be known as the W32/Zallome worm. The code of the worm appeared unweildly and bloated, but its unusual polymorphic engine captured the analysts' attention.

2010-2011: Second Part to Hell: Evoris and Evolus

Second Part to Hell's experiments in viral evolution appear to be the most sophisticated yet encountered. SPTH begins by identifying computational fragility as the principal obstacle to the the evolvability of virus code as implemented in x86 assembly. An obvious way to circumvent this problem, SPTH reasons, is to have the genetic operators operate, not on the level of architecture-specific opcodes, but on an intermediate language defined in the virus's code itself.

SPTH designed his IL to be as highly-evolvable as possible, structured in such a way that an arbitrary bit-flip would still result in a valid instruction, so that they could be permuted or altered with little risk of throwing an exception, and so that there would exist a considerable amount of redundancy in the instruction set: 38 semantically unique instructions are defined in a space of 256, with the remainder being defined as NOPs, affording a plentiful supply of introns, should they be required.

"The mutation algorithm is written within the code (not given by the platform, as it is possible in Tierre or avida)", SPTH notes, referring to two well-known Artificial Life engines. ties in with "red pill" motif The same is true of the IL syntax. In fact, what's particularly interesting about this project, and with the

problem of viral evolution in general, is that the entire genetic machinery must be contained either in the organism itself, or in features that it can be sure to find in its environment. In Evoris, the only mechanism that remains external to the organism is the source of selective pressure – antivirus software and attentive sysadmins. Two types of mutation are permitted with each replication: the first child is susceptible to bit flips in its IL sequence, with a certain probability. With the second, however, the IL instruction set may mutate as well, meaning that the virtual architecture itself may change shape over the course of evolution. Interestingly, the first-order mutation operators in the virus are themselves implemented with the viral IL, and so a mutation to the alphabet – one that changes the **xor** instruction to a **nop**, for instance – may, as a consequence, disable, or otherwise change the functioning of, first-order mutation (as SPTH observed in some early experiments).

Evolus extends Evoris to include a third type of mutation: "horizontal gene transfer" between the viral code and files that it finds in its environment. Since the bytes taken from those files will be interpreted in a language entirely foreign to their source, there's no real reason to expect any useful building blocks to be extracted, unless, of course, the Evolus has encountered another of its kind, in which case we have something analogous to crossover. (Horizontal gene transfer with an arbitrary file would then be analogous to "headless chicken crossover", with the random bytes being weighted to reflect what the distribution found in the files from which the bytes are sourced.)

Though SPTH's results were fairly modest, the underlying idea of having the virus carry with it its own language for genotype representation, and to take cares to ensure the evolvability of that language – and to expose the genetic language itself to mutation and selective pressure – is inspired, and turns SPTH's experiments into valuable proofs of concept. With them, at least two major obstacles to the use of evolutionary techniques in the field of offence **have** been addressed and, to some extent, solved by the vx community: the problem of code brittleness, or the viability of genetic operators, and the problem of self-sufficiency (unlike academic experiments in evolutionary computation, the virus must carry an implementation of the relevant genetic operators with it everywhere it goes – "the artificial organisms are not trapped in virtual systems anymore", SPTH writes, in the conclusion to the first of his series of essays on Evoris and Evolus, "they can finally move freely – they took the redpill").

The Future of Evolutionary Viruses

Interestingly, even in the virus scene, which is certainly where we find the most prolonged and serious interest in evolutionary computation among black and grey hat hackers, the uses to which evolutionary methods are put tend, for the most part, to be fairly modest, and oriented towards defence (defending the virus from detection). When genetic operators are employed, they tend to serve as part of a polymorphic or metamorphic engine, and the force of selection principally makes itself felt through antivirus and IDS software. Outside of science

fiction however, we have not seen any discernable attempt to put evolutionary techniques in the service of malware that *learns*, in a fashion comparable to what we see with next-generation defence systems. There is nevertheless a tremendous amount of potential in this direction, and the threat of unpredictable, evolving viral strains emerging from this sort of research is one that hasn't failed to capture the imagination.

In a paper presented at the 2008 *Virus Bulletin* conference, two artificial life researchers, Dimitris Iliopoulos and Christoph Adami, together with malware analyst Péter Ször of Symantec, outline the threat that such technology may pose and the extent to which it would be feasible to produce greatest risk, it seems, concerns the possibility of detecting such malware. Existing obfuscation techniques, they note, all share the same theoretical limit: though polymorphic and metamorphic variants of a malware strain may evade literal signature detection, and and syntactic/structural detection, they do tend to share common *semantic* invariants, and remain vulnerable to detection by means of a well-tuned behavioural profile. "Simply put," they write,

> biological viruses are constantly testing new ways of exploiting environmental resources via the process of mutation. In contrast, computer viruses do not exhibit such traits, relying instead on changing their appearance to avoid detection. *Functional* (as opposed to cryptic) variation, such as the discovery of a new exploit or the mimicry of non-malicious behaviour masking malicious actions, is not part of the arsenal of current malware.

Evolutionary techniques, by contrast, could allow for the generation of malware instances whose semantic variation is bounded in extremely minimal, abstract, and subtle fashions, as demanded by the task at hand, offering little to no foothold for existing detection technologies. If allowed to develop more freely, moreover, with no selective pressures beyond replication, survival, and the subversion of the systems intended to stop them – and if they could incubate in environments where those particular pressures are gentle enough to allow for relatively "neutral" (non-advantageous, but non-deleterious) exploration of their environment – then "the emergence of complex adaptive behaviors becomes an expected result rather than an improbability, as long as exploitable opportunities exist within the malware's environment."

Iliopoulos, Ször, and Adami, here, are discussing the use of evolutionary techniques in virus generation, rather than payload generation, as examined in this thesis – and, indeed, as we'll see, despite the relative dearth of concrete advancements, the theme of evolutionary computation has been a preoccupation of virus writers ever since the first computer virus was crafted, a phenomenon we don't see paralleled in other fields of offensive/counter-security. There are challenges facing the deployment of evolutionary malware "in the wild" that we don't encounter when developing it "in vitro" – that is to say, in a virtual laboratory, where selective pressures can be fine-tuned with care, rather than left to external circumstance. Where the research presented here rejoins Ször,

Iliopoulos, and Adami's anticipations is in examining the results of relatively free and unconstrained exploration of a host environment by evolutionary malware, where the tether to semantic invariance is intentionally kept as loose as possible and the specimens have the ability to salvage and recombine whatever functional code they can from their hosts:

> A threat might be able to snatch code from another program in its environment. We have seen examples of a virus like Pinfi jumping on top of worms to replicate in new environments as a combination threat. Security products do not always recognize the worm once it is infected with a virus, and the combination helps the survival of both threats. Disinfected worm copies can also escape attention, as can copies of worm replicas that changed due to transfer of code over network channels. It is conceivable that an evolutionary function in the malware could snatch clean or malicious code from other programs. It could integrate code from other programs by identifying function prolog and epilog code. When this takes place, a function is safely inserted into the code base of the evolutionary virus as a new 'function' by running the newly acquired code as a new thread. Existing features might be replaced by the code, which could end up producing reliable output to a given input. (For example, as long as function X returns values greater than 0, it is accepted.) Even complete functionality might be snatched from another clean program, or another virus as well. As previously predicted [17], a cooperation protocol can enhance sharing of features between malicious executables as well. Code snatching is a tried and true function of almost all biological organisms. Bacteria exchange code in small segments called plasmids, while viruses routinely integrate bacterial code into their own. Often, viruses carry this piece of code to other bacteria, a phenomenon known as transduction.