The Prank

(chapter written by Bradford Tuckfield in May 2022)

It all started with a prank. It was 1982, and Rich Skrenta, the prankster, was a high school freshman in upstate Pennsylvania. Like many 14-year-olds, before and since, he liked to play practical jokes on his friends.

Practical jokes change as technology advances. The Roman Emperor Elagabalus, for example, supposedly made the world's first whoopee cushion out of an animal bladder, using it to embarrass his least favorite dinner guests. In early-80's Pennsylvania, whoopee cushions were not unheard of. But technology had moved on since ancient Rome, and Rich Skrenta chose a new medium for his pranks: floppy disks.

Skrenta started small. He would alter the code on a game disk so that it would display a funny message on the screen of whatever computer tried to run it. He gained a reputation among his friends for loaning altered floppy disks that performed little pranks like this - sometimes showing a funny message, or shutting down the computer, or playing a random noise, or flipping the screen upside down, or causing some other harmless annoyance.

As Skrenta played more and more of these little jokes on his friends, he gradually became a skilled coder. Displaying a short message is a simple thing to write code for, but he was also learning how to write code to delete and copy files and how to overwrite system configurations. He was coming to understand the most minute details of how all of the parts of a computer worked together, from the floppy disk whose code he was altering, to the firmware that allowed basic input and output, to the individual software programs that were running, to the operating system that was managing the complex interactions of all of these parts. Not only did he learn to write code to adjust the behavior of each of these computer parts, but he could do it in assembly language, a notoriously difficult and opaque programming language much harder to master than the comparatively friendly languages in common use today.

Over the winter break of the 1981-82 school year, Skrenta wrote code for a prank more elaborate than any he had ever tried before. He called this new prank the "Elk Cloner," and it was nothing more than a computer program consisting of a few hundred lines of code that he stored on a floppy disk. Skrenta gave the Elk Cloner the ability to perform more than a dozen little tricks. Some of the tricks were recycled from his previous pranks - it could invert text on a computer's screen, play noises, and display silly messages, for example. But this time, he also wrote code for a capability that he had never added to a floppy disk before: he gave the Elk Cloner the ability to copy and paste its own code.

Copying and pasting might seem like a trivial capability for a code snippet to have. It's one of the first skills that people learn when they're starting to use a computer, and for a human it's as simple as pressing CTRL+C and CTRL+V. But think about what it means for a computer program to copy itself. If your friend gives you a disk that has some some

prank code on it that turns your screen upside down, then maybe you laugh or roll your eyes, and then you delete it, forget about it, and that's the end of the prank.

Or is it? If that prank code also makes a copy of itself and saves it to your hard drive, then that copy would "live" on your computer even after you threw away the prank disk. Then, the next time you insert a clean disk into your computer, the prank code might copy itself onto your clean disk. If you send that "clean" disk to your friend in Japan, your friend will get pranked just like you did. Not only that, the prank code could also copy itself from the disk you sent and paste itself to the hard drive of your friend's computer in Japan. If your friend then shares other disks with 10 of his friends, your friend's computer might have copied the prank's code to each of those disks as well. You can imagine that before long, a little snippet of prank code that you had completely forgotten about could copy itself onto more and more computers and disks until it spread literally all over the world, tampering with computers owned by millions of people you had never met. The trivial capability of a program to copy and paste its own code can cause a child's prank to expand from a one-time annoyance to a global debacle.

This is exactly what the Elk Cloner did. Starting on a single, thin floppy disk, it copied and pasted itself to the computers of Rich Skrenta's friends and classmates. From there, it kept spreading as people shared computers and floppy disks with their friends and relatives. Before long, it was spreading to people who Skrenta had never met, and, by tirelessly copying and pasting itself to new "hosts," it eventually spread all over the world.

When a computer program copies and pastes itself, it's doing what biologists call self-replication, the same thing that bacteria do when they go through mitosis and not so different from what humans do when we have children. Self-replication is one of the main abilities that we use to distinguish living things from non-living things. A computer program's ability to copy and paste its own code thus gives it something in common with bacteria and people, and even gives it a debatable claim to be literally alive. Rich Skrenta, a high school freshman who had wanted nothing more than to make his friends laugh, had created a self-replicating program that spread on a large scale on personal computers, for the very first time ever. He had created more than just a prank. By creating something with the capacity to disrupt a computer's normal functioning, a powerful ability to spread, and a claim to be a living thing, Rich Skrenta had created the world's first computer virus.

The Elk Cloner

The Elk Cloner, Rich Skentra's prank-turned-first-global-computer-virus, consists of 400 short lines of code. Its most notorious output is its "poem." After the Elk Cloner infects a computer, it counts the number of times the computer has been rebooted. On the fiftieth reboot after infection, it displays the following text on the screen:

```
Elk Cloner:  The program with a personality

  It will get on all your disks
    It will infiltrate your chips
```

```
    Yes it's Cloner!

It will stick to you like glue
  It will modify ram too
    Send in the Cloner!
```

You may or may not agree that the Elk Cloner program has a personality, as the poem's subtitle claims. But in this poem we can get a small glimpse of the teenage Rich Skrenta's personality. We can see his youthful excitement about his creation, and the thrill he clearly got from modifying RAM and bypassing computers' standard defenses against infiltration. This excitement was not only limited to Skrenta - it was in the air in the tech world of the late 70's and early 80's. These were the very first years of mass production of affordable personal computers, including the Apple II model that Skrenta was using for his pranks. It was a time of accelerating economic growth as well, and the new inventions and new money of the time made many people believe that tech and code could make anything possible. The poem's command to "Send in the Cloner!" was matched by implicit but equally exuberant demands from American consumers of the time, to send in the personal computers, send in the software, and send in the dazzling innovations as quickly as they could come.

You can see all of the code for the Elk Cloner on Rich Skrenta's website, at http://www.skrenta.com/cloner/clone-src.txt. It's written in standard assembly language, and at first glance, much of it seems innocuous. Some parts of the code print messages, some parts load programs, some parts of it count the number of times the computer has booted. None of those parts give the Elk Cloner the capacity to spread all around the world or earn a place in history. The scary part of the code, and the innovative part, is the subroutine called "CLONE," shown here in its entirety:

```
CLONE   CLC
     JSR READ
     LDA IDENT
     STA $B3C0
     LDA VERSN
     STA $B3C2
     JSR WRITE
     LDA $AA68
     STA $B7EA
     LDA #$02
     STA $B7F4
     STA $B7EC
     LDA #$08
     STA $B7ED
     LDA #$0
     STA $B7EB
     STA $B7F0
     LDA #$95
     STA $B7F1
```

You're looking at 19 lines of code that transformed an inert computer program into something "alive" - a virus that shook the world. This subroutine accomplishes the cloning task of the Elk Cloner, or in other words it copies the full text of the program's

code to the memory of the host computer. (The host computer will then run the same subroutine and copy the Elk Cloner to whatever disk is inserted into it, so the spread can continue from disk to computer to disk to computer and so on forever.) Instead of ending with a chuckle from his friends, these 19 lines guaranteed that Skrenta's Elk Cloner prank would gradually spread around the world and into the history books.

Today, there are anti-malware companies that carefully track the global spread of computer viruses. But the Elk Cloner was born in the ancient history before any of those companies existed, before any rigorous computer virus tracking was being done, and even before the word "virus" had ever been used to describe a harmful computer program. For these reasons, it's hard to know exactly how widely or how quickly the Elk Cloner spread. Skrenta began the virus's dissemination by infecting his high school's computer, guaranteeing that any floppy disk inserted at his school would be infected by the Elk Cloner. After that, the spread was gradual but persistent, going from floppy disk to computer to floppy disk from his schoolmates to their friends and families and eventually beyond.

Though we can't precisely quantify the extent of its spread, the Elk Cloner definitely made a mark on the tech world. Since this was the first virus to spread widely on personal computers, there was nothing that stood in its way: no security professionals to consult about how to fix it, no automatically downloaded patches from the manufacturer, and no antivirus software to load to kill it. So, it spread gradually for several years, until its first media appearance: a *Scientific American* story described it in 1985 as the first ever computer virus. Eventually, Apple made changes to its DOS operating system to make it secure against the Elk Cloner's style of chicanery. But years later, Skrenta was still hearing stories from strangers who had been unwitting victims of his prank, including a Gulf War sailor whose computer was infected nearly a decade after the virus was initially released.

Since the Elk Cloner's effects were relatively benign, and some of them weren't even noticeable to the untrained eye, its spread did not cause huge waves in the press or the popular consciousness. But in the years since 1982, computer viruses have taken on a significance greater than that of the teenage pranks they were born from. Viruses have become a huge danger for individuals and businesses, a method for nations to fight each other, the impetus behind a multi-billion dollar antivirus industry, and a massive drain on the world's economy. Because of this, the Elk Cloner, though regarded as relatively unimportant in its own era, has taken on a heavy significance, and cast a shadow through time much larger than itself.

When Skrenta wrote the Elk Cloner, he was a self-taught high school kid, unfamiliar with the deep theories of computer science, and unaware of the place his program would take in history. But the Elk Cloner's ability to make copies of itself was something computer scientists had been thinking about and theorizing about actively for decades. Even though the Elk Cloner was the world's first widely-spreading virus, the earliest beginnings of the story of computer viruses lay decades before, in the theories and imaginings of these great computer scientists.

A possibility presents itself

Suppose that you wanted to build a machine from scratch that could make physical copies of itself. How would you build it? This question seems simple enough at first glance, but it gets harder to answer the more you think about it. For example, how simple should your machine be? If a machine is too simple, like just a few pieces of metal soldered together, it won't have the sophistication required to perform construction tasks like collecting materials and soldering them together to reproduce itself. If it can't do construction, then it won't be able to copy itself. But if your machine is complex enough and finely crafted enough to be able to do construction, with a built-in soldering iron and attached computer chips, then the task of making a copy of itself becomes impossible, since even the best chip can't do the silicon mining and photolithography that chip manufacturing requires. Every addition that makes your machine better at making copies also makes the copying task harder, so finding the right balance between simplicity and complexity is extremely difficult.

Even if you can find the right level of complexity for your self-copying machine, you have to face the question of how you store the instructions for how to make copies. Human engineers today use design tools like CAD drawings to express the way machines should be constructed. But will your machine be able to read and interpret CAD drawings? Even if it can, you might face an infinite recursion problem: if your machine contains CAD drawings, then it will have to reproduce those CAD drawings in order to reproduce itself. But then it will need instructions for how to draw those CAD drawings. Those instructions will also have to be reproduced in order to do full self-reproduction. So then it will need instructions for how to create those instructions, and instructions for how to create the instructions for the instructions, and so on, seemingly forever.

Even if your machine can read and reproduce CAD drawings, they still won't be enough, since your machine will have to know what physical steps are required (hammering a nail, welding, screwing a screw, etc.) to build what the CAD drawing indicates, and how to perform those steps perfectly. Many tasks that seem painfully simple to humans, like moving around the physical world to collect materials and put them together, are very hard to teach a machine to do in a reliable way.

The great scientist John von Neumann thought deeply about these issues for much of his career. He gave a series of lectures about the idea of self-replicating machines in 1949. In 1966, one of his collaborators edited a book that described all of his related ideas. The book, called *Theory of Self-Reproducing Automata*, consists of 403 pages of thickly interspersed theories, equations, thought experiments, and editorial interjections. It is a tour de force, drawing on information theory, thermodynamics, geometry, neuroscience, and most importantly, computing theory, to answer questions about whether and how machines could be designed and created to reproduce themselves. But even after 403 pages of magisterial erudition, the answers it reaches are not final or entirely complete. It's understandable that von Neumann couldn't answer all of the difficult open questions in computer science in 1966 when we consider that digital

computers had only existed for a couple of decades at that time. It takes many decades or even centuries for new fields of science to truly mature, and during the mid-twentieth century, computer science was still in its infancy.

To this day, no one has ever designed and created a fully self-reproducing machine in the physical world. Since this task is so difficult, von Neumann focused in his book on something simpler and easier: a self-reproducing "automaton," or in other words a self-reproducing collection of code. This is an approach that computer scientists have taken since the beginning of the field itself in the 1930's - instead of trying to make scientific discoveries in the complicated physical world of silicon chips and floppy disks and monitors and speakers and sockets, they focus on the simpler, "pure" world of zeroes and ones and "automata" that represent an abstract, theoretical version of computers. Von Neumann, by taking this approach in his work, was continuing a tradition that started with the founders of the field of theoretical computer science, including Alan Turing and Alonzo Church. Von Neumann was able to use their theoretical computer scientific ideas to describe a system of automata (in other words, theoretical versions of chips and code snippets) and exactly how they could construct each other, and exactly how they could store instructions about themselves. In theory, at least, von Neumann showed that a computer program could replicate itself.

But just because a computer program can replicate itself in theory doesn't mean that it's easy to write a self-replicating computer program in practice. It's no different in the biological world: just because mitosis is technically possible for cells, it doesn't mean that it's easy to create a cell, molecule by molecule from scratch, that can reproduce. The theory is difficult enough to understand, but translating that theory into practice is a greater challenge still. The computer scientists had shown that there was a possibility that computer programs could self-replicate, but it would take another generation of tinkerers, researchers, and pranksters to bring that possibility to life.

From theory to practice

After the theoretical advances of von Neumann and other computer scientists, there were some slow steps that brought computer viruses closer to existing in reality. At first, it started with more research. One of von Neumann's collaborators, Arthur W. Burks, not only edited von Neumann's lectures and research papers, but also wrote his own dissertation on the ideas of self-reproducing computer programs. Research papers started to appear in scholarly journals describing in more detail how computer programs might reproduce themselves, and even started to provide bits of code that could be used for self-reproduction.

It was not until 1969 that any of these ideas finally became a reality. That year, someone (evidently a prankster just like Skrenta) wrote a simple program that ran on one of the computers owned by the University of Washington. This program, called RABBITS, was extremely simple: all it did was continually make copies of itself on the computer where it was running. Those copies then continually made copies of themselves just like the original was doing, and the copies of the copies made copies of themselves,

and so on. Just like a fork in the road turns one road into two, we sometimes call the process of copying code "forking." We could say that each copy of RABBITS was a "fork" of the original that, as soon as it was created, forked itself in turn.

You can imagine that the constant, multiplying forking of the RABBITS program could cause problems. For one thing, if left unchecked, it could easily eat up all of the available storage space on a computer - especially on the types of computers that existed in 1969, which had very little memory to begin with. Not only that, but this kind of uncontrolled forking can take up all of the resources of the processor: eventually the computer's chip spends all of its computing power making copies of RABBITS instead of doing other important things like reading input from the mouse or keyboard or displaying text on the screen. An uncontrolled self-forking program could easily cause any computer to quickly crash. That's why this type of attack has come to be called a fork bomb.

The RABBITS program was similar to the Elk Cloner in a few ways: both took advantage of gaping security holes in contemporary technology, and both caused plenty of annoyance, though neither was created with truly malicious intent. But there are some differences between them in their technical details. The Elk Cloner worked by "attaching" itself to an existing program - the Apple II operating system. It made copies of itself, but only one copy per computer, or one copy per floppy disk. RABBITS made a huge number of copies of itself, but was a standalone program that didn't require attachment to any other existing program.

The style that the Elk Cloner used to reproduce - parasitic attachment to an existing program that helps it copy itself - is similar to the way biological viruses reproduce, and that's why we call the Elk Cloner a virus. RABBITS, by contrast, runs and reproduces alone, and does its damage quickly because of its quick reproduction, or forking. That's we typically classify RABBITS as a "fork bomb" rather than a true virus. Both viruses and fork bombs, along with other types of harmful computer programs, are referred to collectively as malware. Even though RABBITS wasn't truly a virus, it represented an important milestone in the history of viruses, since it represented the first time that a virus-like program had been created in practice instead of just described in theory.

Fortunately for the world, there were a few things that made RABBITS relatively weak as malware. First, it quickly crashed whatever computer it was on. Malware tends to be more successful when it doesn't completely destroy the first computer it infects, since destroying its host also destroys its opportunities to spread itself to other computers. In this way, malware is similar to biological diseases: both spread better when the host stays alive for as long as possible. Another thing that ensured that RABBITS wouldn't destroy the world's computers is that it didn't have any way to jump between computers - it only copied itself to the single computer where it was currently running, so the worst case scenario would be the destruction of one computer, not the destruction of every computer in the world. Finally, fork bombs are relatively easy to prevent - just change the computer's settings so there's a maximum number of processes that any single program can run, and then no program or fork bomb will be able to reproduce endlessly.

The restriction of RABBITS to a single computer decreased its potency as a destructive force. But as technology to connect computers became more widespread, it gradually became easier for malware to spread between computers and across large distances. In the late 1960's, the US Department of Defense funded the creation of the ARPANET, a network of computers that's known today as the predecessor of the internet. The goal of the ARPANET was to make communication and collaboration between different computers easy. To achieve this goal, its creators ensured that any machine that was connected to the ARPANET could communicate directly with any other connected machine at any time. This connectivity enabled the creation of some great inventions we take for granted today, like email. But it was also a security problem, since connected computers could request that any other connected computers boot up and run programs - including harmful programs.

Bob Thomas, an employee of the company that created the ARPANET, put this security issue to the test in 1971. He wrote a little program called Creeper that could ask ARPANET computers whether they were idle, and if they were idle, ask them to boot up. When they booted up, Creeper would send them all the files they needed to boot, plus a copy of Creeper, and instructions to run Creeper constantly. The copy of Creeper running on the new machine would follow the same process, booting up other machines and copying itself to them in turn. Creeper didn't do anything too malicious to computers that were running it - all it did was display a simple message: "I'M THE CREEPER; CATCH ME IF YOU CAN".

Though Creeper was simple, it was not easy to catch. By the time you deleted it from one machine, it had probably already spread to several others. By the time you deleted it from those other machines that it had spread to, it had probably spread back to the first machine. You might think of each instance of Creeper as a "segment," and since all the segments were connected through network communications, together they formed a multi-segment "worm."

Just like RABBITS, the Creeper was also similar to the Elk Cloner in many ways: both were annoying but not malicious self-reproducing programs that displayed taunting messages. But the Creeper also fails to meet the definition of a true computer virus, since it was a standalone program that didn't need to alter any other program in order to run and replicate. Programs like Creeper that form multiple connected segments on different, networked machines that can reproduce alone aren't technically called viruses. Instead, we call them worms.

Even though the Creeper was able to spread between computers, its spread was still limited, since only a small number of computers were connected to the ARPANET in 1971. Most people don't regard Creeper as something that spread "in the wild," since they think of ARPANET as more like a single "lab" than an organic collection of computers.

In 1975, another virus-like program appeared. This one was called ANIMAL. It appeared to be a guessing game - the program would tell the user to think of an animal, and then it would ask questions to try to find out what animal the user was thinking of.

But all of this was just a distraction. While the user was answering questions about animals, the ANIMAL program was making copies of itself on every directory the user's account had access to. This could mean that it would clog up the system's memory like RABBITS, or it could even mean that it would spread across a network like the Creeper.

Since ANIMAL appeared to be a benevolent, fun program (a guessing game), but was hiding an evil purpose (its secret self-copying code), we call it a Trojan Horse. Technically, Trojan Horses like ANIMAL are not computer viruses, since they don't reproduce in the parasitic style of viruses like the Elk Cloner. RABBITS, the Creeper, and ANIMAL were all predecessors of the Elk Cloner, but since they weren't technically viruses, and none of them spread to personal computers, the Elk Cloner has a secure place in history. It was the first true computer virus that spread organically between personal computers all around the world. What started as an idea in a series of lectures in 1949 finally became a reality in the winter of 1982.

Slow contagions

In the world of technology and programming, we're accustomed to things happening quickly. Someone has an idea for a computer program, some coders leap into action, and within a matter of hours or days, you can download a working prototype. At least, that's how we often think of the process of technological innovation. Sometimes it seems like the prototype arrives even before the thought. But the arrival of computer viruses didn't match that speed: the foundations of theoretical computer science were laid in the 1930's, then self-reproducing automata were described in 1949 and explored for several decades after that, and only in 1982 did a real computer virus spread in the wild.

Given the typical lightning speed of tech, contrasted with the several decades it took for computer viruses to appear, a natural question to ask is: why did it take so long for viruses to crop up? There are several illuminating answers to this question.

One simple answer is that a virus can only copy and paste its own code if its code is written down. The very first digital computers were programmable in the sense that they could follow instructions and "programs," but the way they were programmed was very different from how we program computers today. Today, we write computer programs as text, that can be easily copied and pasted. But for the earliest digital computers, operators gave instructions to the computers by physically moving their parts and by plugging different cords into different sockets. Even if these early computers had been programmed with a virus or something like a virus, it would be essentially impossible for this virus "program" to spread to other computers, since computers of the time didn't have the ability to physically move or adjust the physical programming other computers using the methods of the time. Only after 1948, when computers started storing programs as text in their memory would self-copying become a real possibility for computer programs.

Another reason why viruses took so long to appear in the world is that a virus can only copy and paste its own code if another machine is capable of understanding and

running its code. Today, a program that can run on one Windows computer should be able to run on any other Windows computer. The Windows operating system is standard and widespread all around the world. But in the earliest days of computers, there were no standard operating systems that would enable one computer to run a program that was written on a different computer. Some of the first operating systems were invented in the late 1960's, and only after they had been invented and spread would it be possible for several different computers to run the same program without any alteration.

A crucial reason why viruses took so long to appear in the world could be summarized as "dry tinder." Viruses are defined by their ability to spread, and if there's nothing to spread to, there can be no virus. If only one computer existed in the world, then even the most talented and malevolent programmer couldn't create a functioning computer virus, because there would be nothing for it to spread to. By analogy, consider the question of why telemarketing was not invented the same day that the telephone was invented. It probably wouldn't be profitable to call Alexander Graham Bell repeatedly, all day, every day. A certain critical mass of telephone users would be required to support telemarketing as a profitable activity. By the same token, for a virus to really spread widely, there should be a large, critical mass of compatible computers that regularly share programs with each other, so that there are enough "infectable" computers susceptible to catching the virus.

When Skrenta created the Elk Cloner in 1982, the dry tinder of common, compatible computers had been accumulating and was reaching a critical level. In the early 1970's, there were almost no computers in homes in America - computers were almost only found in labs and offices. By 1982, the personal computer revolution had begun and it never stopped. Nearly 13 percent of Americans owned a computer in 1982 and that number was growing fast: within 14 years it would more than triple.

As personal computers were getting into more and more homes, the technology for sharing and collaborating was starting to develop and mature. For teenagers who liked to play games, this meant floppy disks. Kids could get disks that contained games, play the games on their family's computers, and then swap disks with their friends, who could run the same disks on their own families' computers. Precocious kids could alter the code on the disks, like Rich Skrenta did when he created the Elk Cloner. The penetration of standardized, mutually compatible personal computers into the consumer market, the easy sharing enabled by floppy disks and networks, the text storage of computer programs, and the lax tech security of a world that had never seen a large-scale virus breakout: all of these ingredients combined to form the perfect conditions for the Elk Cloner (and later viruses) to spread. But it took many years for all of these ingredients to come together.

The final reason why computer viruses took so long to show up in the real world is restraint. When Rich Skrenta wrote the Elk Cloner, there were thousands of talented computer programmers in the world. Plenty of them had the chops to write a program like the Elk Cloner, and let it loose on the world. Many of them even told Skrenta in later years that they had had the same idea. But all of them, whether because of timidity or

maturity, had the self-restraint to prevent themselves from releasing a virus into the world. Skrenta evidently didn't have this self-restraint, and released the Elk Cloner virus into the world without a second thought. When Skrenta was asked why he didn't have this level of restraint, he gave a simple answer while laughing. "I was 14," he said.

The Elk Cloner came at the culmination of decades of research and gradual technological development. Its release was a crucial turning point in the history of viruses. It proved that amateurs from anywhere could write programs with the potential to spread of their own accord and disrupt or even destroy other computers far removed across time and space. The Elk Cloner would open floodgates. Within a few years, there were even more computers, sharing code was even easier, and there were more pranksters, researchers, and bad people who knew how to write code for viruses. There would be more viruses, and bigger problems arriving at an accelerating rate in the years to come. The Elk Cloner would be the first short chapter in the long story of computer viruses, a story that began with an anonymous teenager's prank but continues to shake the world and its nations today.

A new kind of life

Biological viruses like the flu, the common cold, and COVID-19 don't have cells, don't eat, don't have nervous systems or senses or brains, and don't move of their own volition. But since they have genetic material (RNA), and since they can reproduce (by hijacking the reproduction process of a living cell), many biologists take the position that they are a living part of the "tree of life," or at the very least that they're a gray area between the living and the non-living - that they're alive at least in a weak way.

Software viruses have their own kind of genetic material (their code), and their own ability to reproduce (by hijacking the operating system and hardware of a computer). Like biological viruses, they can spread around the world, influence our lives, and devastate our economies. If biological viruses are alive, then it's not impossible to argue that computer viruses also constitute a type of life.

If we can believe that computer viruses are alive, then they are the first living thing that has ever been completely designed, manufactured, and created from scratch by humans. It's exciting to think of what we're capable of - that we can actually design and create and sustain a new type of life. On the other hand, it's alarming that the first type of life we've been able to create is as destructive as computer viruses tend to be. Stephen Hawking, the great physicist, believed that computer viruses were alive in some way. He described his opinion of them as follows:

> *"I think computer viruses should count as life ... I think it says something about human nature that the only form of life we have created so far is purely destructive. We've created life in our own image."*

There are some excuses we could make for creating a destructive form of life rather than one that's helpful and attractive. The first natural excuse is that harmful programs are much easier to create than helpful ones. Every programmer knows this - even if you set out with perfectly pure intentions, any little mistake or oversight in your code could make a computer crash, could give the wrong answers, could accidentally delete something, or otherwise cause some kind of damage or waste. Even talented, well-intentioned programmers create nine harmful programs for every one helpful program, since destruction is inherently easier than creation.

Another important excuse for why we created viruses before other, more benevolent types of life is that, despite the name, computer viruses don't need to be destructive and evil. Even from the earliest days of research on computer viruses, computer scientists were pointing out that helpful computer viruses could exist. One early example that was described by Fred Cohen, a prolific virus researcher, is a so-called "compression virus." A compression virus, if one existed, would spread across computers, just like the Elk Cloner. But instead of playing tricks on users or displaying taunting messages, a compression virus would automatically compress files that were taking up space in the computer's memory. Compressing files would seem to be something unequivocally helpful, so if a virus like this spread around the world, it would be a beneficial life form that could give us all some more space on our computers.

We know exactly how computer viruses came into existence - through a prank. By contrast, we don't know exactly how biological life began in the universe. One recent study suggests that lightning strikes could have played a key role in making the right elements available to mix together and spontaneously form living things on Earth. Lightning would match the gravity and drama that we might naturally associate with an event as weighty as the genesis of life - the same gravity and drama that we encounter in the origin stories of the major global religions.

But life doesn't necessarily need to begin with a serious, dramatic moment like a lightning strike. The writer R.A. Lafferty wrote a sci-fi story in which he described one man's effort to discover the ultimate origin of life in the universe. In that story, an astronaut finds an alien civilization in which no one ever had ever died - their ancestors, throughout all of their generations of existence, just shrank and got increasingly sleepy as they got older, without ever dying. He realizes that if he can track down the oldest ancestors of these aliens, they must have a memory of how their civilization began. They could tell him how their species came to exist and how they got their spark of life, whether from God or a lightning strike or something else. They could tell him how life itself began.

In the story, the astronaut tracks down the oldest grandparents of the aliens, wakes them up, and he asks them how life began. They reply, in their strange, ancient grammar:

> *"Oh, it was so funny how it began. So joke! So fool, so clown, so grotesque thing! Nobody could guess, nobody could believe.... Is too joke to tell a stranger...so funny, so unbelieve."*

They never tell the astronaut the ultimate origin of life, preferring to simply laugh at their recollection of it, but it's clear that the origin story is not something serious like a lightning strike or a booming serious voice from the clouds. We humans have never seen life truly come from nothing. The closest we've come, if you can take computer viruses seriously as a form of life, is the Elk Cloner and its successors. Just like in Lafferty's story, this form of life started with something funny, a prank, a silly joke, an attempt to make friends smile.

Computer viruses may seem like serious, frightening things. They can devastate nations, delete your family photos, drag down an economy, take your credit card number, stop innovation in its tracks, and spy on us in our most private moments. They can be destructive and even deadly. But don't let them scare you too much. Never forget that viruses started with a humble attempt to make people smile - the whole story of computer viruses started with a prank.