# Computer Virus Enhancement

John P Lynch
Division of Computer Science
University of Minnesota, Morris
Morris, Minnesota, USA 56267
lynch446@morris.umn.edu

## ABSTRACT

This paper gives insight into computer virus computation and the applications of evolutionary computation and advanced algorithms. Genetic modifications from evolutionary algorithms can be made for generating viruses and advanced forms of search. These search algorithms are for finding specific items in a system such as a folder in a file system or an exploited port number on a switch to connect to another computer in a network. The parts and phases of the computer virus will be handled to give the reader a better understanding of how and where the virus may be improved by some form of evolutionary computation. The advances in malware means that anti-malware improves as well to combat the increasing strength of evolving malware.

## 1. BACKGROUND

In order to better understand the content, definitions on what is applied throughout the paper will be given. A *computer virus* is a malicious software program that replicates itself to perform different tasks on a computer system in different parts of the system. Computer viruses have different phases and parts that are active throughout their lifespan. *Epidemiology* is the study of how a computer virus may infect other computers in a network or system. Epidemiology includes measurements such as speed and percentage of infection to vulnerable systems. A*population* is a particular section or group of individual code based in a similar generation of creation. *Genetic algorithms* are a type of evolutionary algorithm based on natural selection relying on mutators to change and reshape the formations of each new population. *Mutators/genetic modifications* are operators to create genetic diversity. They are used in genetic algorithms to randomize selected portions of what the algorithm recreates. *Anti-malware* is software to detect, prevent, or 'cure' malware. Advanced search (with heuristics) are simple rules applied to some versions of search for finding data in structures or systems. *Topology* is the arrangement of nodes and links in a computer system network. This study includes how computers connect to each other from switches and servers.

A computer virus has several operations and functions. There are four phases of the computer virus. The *dormant phase* is when the virus may be infected in a system yet is causing no actions or functions yet. It is waiting for the *trigger phase* of the virus which is the initializing code to perform its functions. The trigger can be a timer or an execution of detected events. These events can be something as simple as waiting for a condition or by double clicking an application holding the virus. The *propagation phase* of the virus is when the virus is replicating itself and triggering the new virus as it's created. The *execution phase* is the actual virus' intended functions to be performed. The virus may delete many local files, aside from itself, or try to continue to propagate and slow a system down from using disk space and power.

The parts of the computer virus are portions of code and methods to complete the viruses different functions. The infection mechanism of the virus is the function of reproducing with searching system areas that a virus uses to propagate. This is usually a simple search algorithm that attempts to find all other folders on a disk and copy itself into the folder before activating its new copy. The payload of the virus is considered the malicious portion of the virus. The payload may be a copying method to steal information or a destructive bit of code to modify a system. The trigger is the initializing code or feature that delivers the payload portion of the virus when it is activated.

Evolutionary algorithms are based on the idea of reconstructing sets of data as a population. We create an initial population of data for accomplishing whatever task we need from that algorithm. Eventually we evaluate the "fitness" of each set of data as the effectiveness of the data to accomplish what we need from the algorithm. After ranking the population we take the strongest of the population and attempt to mutate the data using whatever we have for the changeable genes. The genes are based on what can be changed to accomplish a task differently. If code wanted to create populations with different names, the characters of the name in each generation would vary, these characters. Figure 1 creates the basis of an evolutionary algorithm, however it lacks an explanation of how this can be applied to computer viruses.

For a constructed example, the virus that we will be using as an example throughout the paper will be a '.bat' file named "supahVirus.bat". This '.bat' file will be built to infect some average low-end windows system such as Windows


*UMM CSci Senior Seminar Conference, December 2016* Morris, MN.

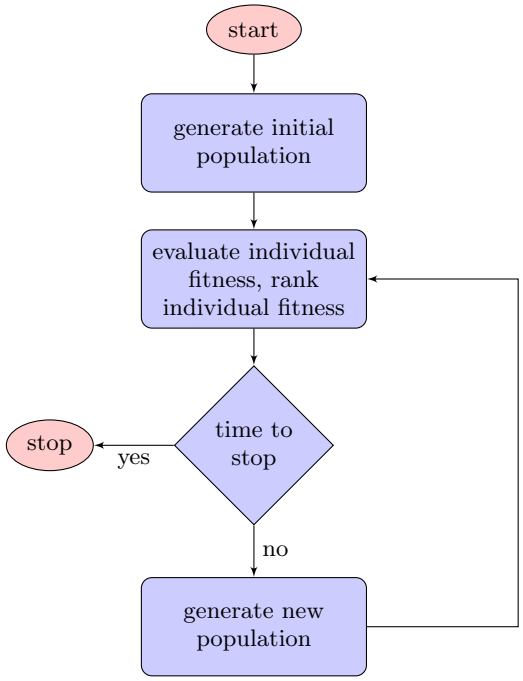


**Figure 1: evolutionary algorithm basic structure**

Vista Home. To illustrate what the virus is hoping to accomplish consider some network of computers where there is a folder named 'secretFolder' in each system on the network that we want to gather information on. The owners of these computers don't want us to find the unique information on each system so we developed this virus to search out the folder and copy it to send to our system.

## 2. GENETIC CHANGES TO A VIRUS

Genetic modifications are used to create different formations of code via genetic algorithms. These algorithms define and refine from different strains of viruses to create new strains that can infect and propagate to recreate themselves over and over. From genetic modifications they become stronger and stronger to accomplish different tasks that would be otherwise too difficult to successfully work for all kinds of systems based on one model. One major factor in the measured strength of computer viruses is in their ability to self replicate using the infected system.

### 2.1 Propagation phase of the virus

The propagation phase of the virus as previously explained is the phase in which the virus self-replicates and spreads. The virus would hold some action that it would activate to begin replicating and spreading. This means that our "supahVirus.bat" will call a method "swarmMeth". This method will repeatedly recreate the same computer virus in different parts of the system which may prove problematic. Most anti-virus software that doesn't want a virus to 'reproduce' looks for repeating programs that seem malicious. For the virus a possible solution to hiding from anti-malware is to change the virus each time so that we are not recreating the same code, names, and files that the anti-malware would pick up on. The software would only see that files and code are being written, and not that the same files and code are being recreated. This modified propagation could use something that doesn't just make the virus harder to be observed by anti-malware, but also more powerful each time it is created.

### 2.2 Genetic algorithms and mutators for malware propagation

To create a genetic algorithm that would mutate our virus each time it is created we begin with the foundation of the formula that will go into our example "supahVirus.bat". A 'true' genetic algorithm is supposed to more accurately portray a Darwinian scenario with parent variants of the stronger versions of modifications. A 'child' is then created by the two parents combinations which would share overlap and randomly choose between the two parents differences. For the sake of simplicity we will be using a simplified version which creates very minimal changes to genes while still accomplishing the designated task of our "supahVirus.bat".

```
swarmMeth() {
 initialize(thisVirus);
 powerOfThisVirus = evaluate(thisVirus);
 while true do {
  mutatedVirus = mutate(thisVirus);
  powerOfMutation = evaluate(mutatedVirus);
  if powerOfMutation >= powerOfThisVirus{
   initialize(mutatedVirus);}}}
```

To measure the power of a computer virus, we only need to take into account the speed of successfully creating viruses and the speed of running the code of our virus. To check for this speed we created a method "evaluate" in our code that simply runs and checks if the virus created would work properly and would work as fast or faster than the current virus in initialization. One advantage of stealth in a computer system is that it is relatively easy to trick basic anti-malware. Something as simple as a change in name will allow new generations of viruses to gain mild variations on names such as "supahVirus" changing to "superbVirus". These changes in name allow a virus to successfully hide from most anti-malware that simply detects repeating file and program names. This is proven by the paper of evolvable malware by Sadia Noreen. [4]

This method should bring about multiple changes from each new mutator given to create multiple viruses. Each new virus that is created that functions similar to the original virus and is as fast or faster are selected to be initialized throughout the computer system in different areas. We create a simplistic check that allows us to determine the speed of the virus. The problem with this check is that it is based on the size of the virus created in code while still pertaining to what the virus was designed to accomplish. The check would simply be whether the size of the code remains the same and wouldn't be larger than the initially tested against population. One virus may have a mutator which changes the function of copying various files in the folder it was initialized in and it would instead start copying and removing those same files before sending them to other areas in the system. Another possible mutation that anti-malware may not perceive is the different ways that the virus may use the computer's systems to perform its actions. This means that

one version of "supahVirus.bat" may produce a terminal and begin processes to search through file systems while another version could hijack the windows explorer and directly act like a user or administrator to the file systems.

# 3. INFECTION AND SEARCH IN A COMPUTER VIRUS

A very important part or operation of the virus is to search a computer system for different files, folders, or disks to infect and continue propagation. To properly propagate it is required that the virus spreads so that it does not simply stay in the point of infection. To search for the "secretFolder" we require the operation of the "supahVirus.bat" to recreate itself in each new folder. Eventually the virus finds and starts copying the contents of "secretFolder" after we propagate into it.

## 3.1 Infection mechanism/vector

In all computer viruses there is a mechanism that typically uses a search routine to find new files or disks to infect. This portion of the virus doesn't exactly seek to just copy itself, (although that is the intention of the infection mechanisms other method) but to also spread so that it can properly disperse copies into other parts of the disk. To accomplish spreading to propagate the virus "supahVirus.bat" will require another method that we will call "mechMeth()".

```
mechMeth () {
 while true do{
  for each folder f in folders {
   if (folder.name == "secretFolder"){
    copyAndTransfer (f)
    break
   }else{
    swarmMeth()
    into (swarmMeth.mutatedVirus, f)
    initialize (mutatedVirus)}}}}
```

This infection mechanism would search through at top levels 'folders' and attempt to go into each file system and initialize the newly created mutated viruses in each. To properly do this we would need to call into the top-level of the file system. This is much more easily accomplished by a ".bat" file given that it can open and write to a terminal which can write and run code to the disk and the highest priority folders that can be infected. This method was improved and revisioned from Sadia Noreen. [4]

## 3.2 Heuristic search

Our heuristic search prioritizes systems that we can infect and ignores systems that don't match our preferred system. What this means for "supahVirus.bat" is that when we are in the dormant phase of our virus, (before we actually begin to infect systems) we want to seek out the highest system or folder that has the most vital information that we want to affect. This is done by our ".bat" file having simple terminal access that can write the actual payload of the virus delivered to the highest writable folders of the disk. The virus initializes and starts in top level folders before propagating into lower level folders in it's file system. Each new folder then infinitely recreates the virus into each folder recursively.

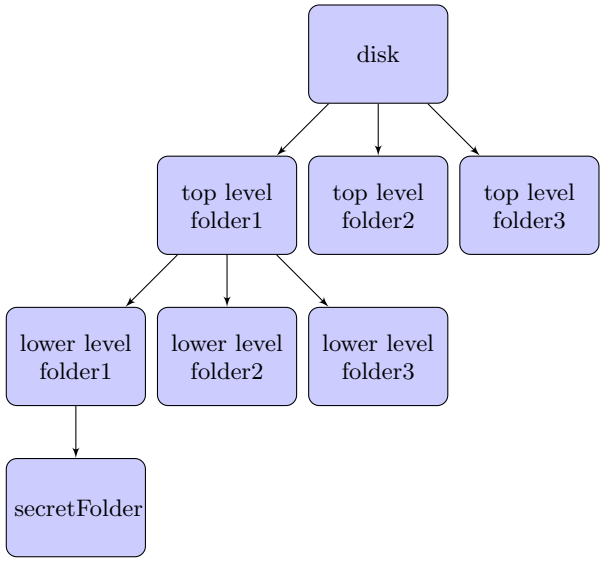


**Figure 2: computer search**

To speed up the process the heuristic that we include in our infection mechanism method "mechMeth()" will be one that prioritizes the list of folders for what is most likely holding the "secretFolder". To do this we may create a data structure which would be a map of folders to search through in our 'for' loop that automatically prioritizes those folders for having a higher chance of containing the target folder. To change the code we simply change 'folders' in the for loop to the data structure with priority map values being to each folder which would most likely contain the target.

```
mechMeth () {
 priorityfolders = priorityMap(heuristic);
 for each folder f in priorityfolders {
  if (f.name == "secretFolder"){
   copyAndTransfer (f)
  }else{
   swarmMeth()
   putinto (swarmMeth.mutatedVirus, f)
   initialize (mutatedVirus)}}}
```

This increases the average speed for possible folders to search through by using knowledge of what is most common for where the target folder is. The upper bound remains the same as all possible 'for' loops can be executed before finding the target folder. The lower bound lowers to the shortest possible path to the target folder from the disk. The average speed increases to become closer to the lower bound. Each new virus prioritizes which folders may hold the secret folder and may reduce the time to be linear.

Observing figure 2 we prioritize the top level folder to propagate into folder 1 and continue to lower level folders to continue. When that newly created virus is pushed into top level folder 1 and initializes it creates another priority map based on its knowledge until it finds lower levels that hold the secret folder.

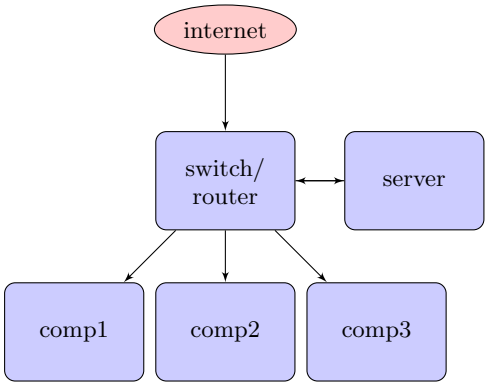


**Figure 3: basic switch/server layout**

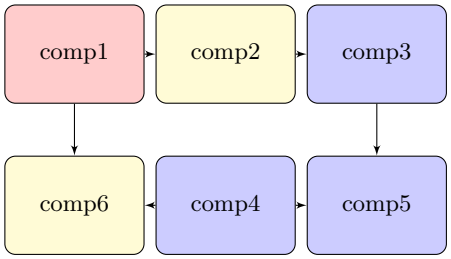


**Figure 4: adaptive network control**

# 4. EPIDEMIOLOGY

Changing to the topic of systems on a network instead of just an internal file system, the effectiveness of a computer virus to spread to other systems can be improved.To measure the effectiveness of a computer virus' ability to spread we create a biological analogy. The analogy is broken down into three states of possible infection regarding separate computer systems: clean, infected, vulnerable. The analogy is meant to show that connected computers that have exploits are considered vulnerable. For this we consider models of servers that connect various computers to a 'local' network as opposed to server to server exchange as 'global'. Servers are considered transmittable for our virus "supahVirus.bat" and can move the virus from computer to computer or server to server while not being under the effects of being considered infected.

According to figure 3 we determine that a system uses a search for each computer attached to each other computer across a network by the port number inherited by each computer to the switch. What this actually means is that each computer system is only attached to each other system through a switch and port number on the switch. Using a normal search, "supahVirus.bat" would not be fast or effective in finding vulnerable systems to infect. The virus will search for each computer in an adjacent server and only travel from server to server after attempting each computer in a 'local' level.

## 4.1 New search in framework

Now for each virus that is recreated in the propagation phase there could be a possible breach into a new server. From each server we may propagate multiple times for each computer that we observe and try to find ways to breach each of them. Using the same kind of advanced search from the previous section that we used to find breaches we may also generalize it to find vulnerable computers to infect. Using a heuristic that doesn't prioritize computer systems that would not match the kind of systems that our "supahVirus.bat" could infect, the virus doesn't attempt to spread to systems that it has some knowledge of that it couldn't infect.

Part of the example we have is a computer system that uses the Linux operating system, according to the style of our "supahVirus.bat" we could not infect it as well. With the knowledge of each system we look at before attempting to pass the virus into that system, we increase speed of infection to other systems by not attempting systems that are not considered vulnerable to us. This can change the speed of our search from an average speed to a faster speed. The virus may still spread to systems in the slowest possible way however if there are conditions that simply don't work to the virus' best advantage. There are speed and stealth tests of a piece of evolved malware called bagle by [4], these tests show the improved effectiveness of evolvable malware in computer systems.

## 4.2 Representations of infection

In figure 4 we show that each computer is connected very closely with each other computer on a local network. These computers are not entirely close given switches and routers that require points of entry into different sections of the network. These connections allow a virus to spread from computer to computer either through the switch or directly from computer to computer that have access to each other. The biological analogy previously explained show that each computer system can infect each other computer system adjacent in either port or direct connection. The vulnerable systems are infected from the search through port numbers and variations of exploits as explained by Shouhuai and Wenlian Xu. The viruses find vulnerabilities in a system's connection and knowledge of connectivity. Vulnerable computers can be infected through exploits attached to each system whether it's commands that travel to other computers or found local email information to travel across the internet.

## 4.3 Trojan horse optimization

A Trojan horse is not actually a virus but acts similar to one in it's malware content. Rather than searching for a possible exploit or breach it relies on user error which would allow the user themselves to trigger the malware. We could imagine that our "supahVirus.bat" file is a Trojan horse doesn't use a breach search algorithm. "supahVirus.bat" still acts like our original virus which searches through file systems to alter their data. According to testing done by Andrea Cani in the paper on automated malware creation, using a trojan horse called "timid" and genetic algorithms created different generations of the modified trojan horse.[2] These generations were harder to detect for anti-malware as previously explained and were spread across a file system much like our "supahVirus.bat".

# 5. PARALLEL SECURITY TO MALWARE GROWTH

Much like in viruses, there are advanced forms of computation that, when applied to anti-malware, can improve the performance marginally. To use detection of signatures of code and not just the forms of the code we gain insight into how to find evolving malware. Like in epidemiology in adaptive systems, the advanced anti-malware methods taken by the researchers (Xu, et. al) proved that there are ways to control computer virus propagation across a network. This can be applied to personal systems as well from anti-malware in detecting machine generated viruses.

## 5.1 Detecting machine generated malware

To properly demonstrate machine generated malware detection, the testing done by Andrea Cani and their team began with the "timid" trojan horse having several generations that all evolved in mass quantities throughout the simulated system. To find the trojans in the simulation, the anti-malware used a form of signature detection. *Signature detection* was when anti-malware would collect samples of what it believed to be malware and would look for consistent repeating patterns in code. This took a sample of the detected virus and began testing against the different files in the system until it found matches and would count towards the found number of detected malware. To ensure that the code wouldn't copy the signature of something insignificant the code sample used as a signature to test against the other files and viruses would have to be some payload portion of the virus. The payload is the 'logic bomb' of the virus that actually is the code to activate the changing or 'malicious' method or mechanism in the virus. For our "supahVirus.bat" this would be the previously created method "swarmMeth" that copies the virus and initializes it in whatever folder it is called into. For the team, a simulation proved successful after each generation was taken and computed for each new form of the detectable signature. New signatures were added to the possible signatures to detect in a virus for the anti-malware.[2]

## 5.2 Modeling timing parameters for virus generation

For creating anti-malware that seeks to slow if not halt a virus in a network, there are two unique parameters to consider when adjusting for large networks to control virus populations. According to a research paper written by Yang and Chenxi Wang [5], the two most powerful methods for halting virus growth in a system or network are as follows; the vigilance of a user of a system or system in a network, infection delay based on systems throttling computational power when a virus is detected. When throttling the CPU of a system, this slows the virus and its ability to propagate. After slowing the system down the anti-malware can begin searching the system for the virus and all copies of it. This requires that the anti-malware be outside of common system use to find disk use and change it. The anti-malware may just gain access to the processes of the computer and halt processes that it detects to be malicious.

## 5.3 Adaptive network thresholds and control

Despite the faster natures of the viruses, there are ways of controlling a network infected by computer viruses. According to research done by Shouhuai Xu and their team, there are strategies that would allow the conditions of servers susceptible to virus attacks to be more safe. Adaptive network control that bases itself in two strategies of semi-adaptive and fully-adaptive defenses. The fully adaptive defenses are based on no input parameters including the threshold of when and where a virus is likely to spread. The semi-adaptive scenario is based on having the assumption that there is a virus in the network that is spreading and must be mapped to where it will spread next. From the research it was concluded that there were fully and semi adaptive defense scenarios that allowed the control or destruction of the virus in a network. To gain some perspective on this, we will use "supahVirus.bat" in our example network. The virus is in some state of topology that may be in a portion of the servers. For some amount of servers/systems 'X' there are 'Xi' infected systems connected to another list of systems 'Xv' for vulnerable. By the semi-adaptive defenses an anti-malware program can tell what servers are infected by "supahVirus.bat" and can add them to 'Xi' while strengthening the vulnerable servers 'Xv' to detect when a new unwanted program is initialized. This scenario goes on to spread like a virus to clean each system in 'Xv' from detected unwanted programs. The anti-malware would detect the programs using CPU in too large of quantities and clean programs it would deem as unnecessary. From the scenarios presented, there would eventually be a state where all 'Xi' are eliminated from each 'Xv' space working out almost like battle lines. This can be explored more in the research paper by the Xu team. [7]

# 6. CONCLUSIONS ON ADVANCED MALWARE AND ANTI-MALWARE

Evolutionary algorithms and advanced computation in different viruses and anti-malware create new directions for both software. From given examples and dissected processes we have proven that there are applications of advanced and evolutionary computation and those applications can have benefits to computer viruses and anti-malware. Despite not increasing the speed of the malware optimally, the virus is still improved in its ability to hide from anti-malware. There is future research involving computer viruses that could prove useful in testing anti-malware and computational strength involving advancing models.

# 7. REFERENCES


[1] T. Bäck. Evolution strategies: Basic introduction. In *Proceedings of the 15th Annual Conference Companion on Genetic and Evolutionary Computation*, GECCO '13 Companion, pages 265–292, New York, NY, USA, 2013. ACM.

[2] A. Cani, M. Gaudesi, E. Sanchez, G. Squillero, and A. Tonda. Towards automated malware creation: Code generation and code integration. In *Proceedings of the 29th Annual ACM Symposium on Applied Computing*, SAC '14, pages 157–160, New York, NY, USA, 2014. ACM.

[3] Y. Kandissounon and R. Chouchane. A method for detecting machine-generated malware. In *Proceedings of the 49th Annual Southeast Regional Conference*, ACM-SE '11, pages 332–333, New York, NY, USA, 2011. ACM.

[4] S. Noreen, S. Murtaza, M. Z. Shafiq, and M. Farooq. Evolvable malware. In *Proceedings of the 11th Annual Conference on Genetic and Evolutionary Computation*, GECCO '09, pages 1569–1576, New York, NY, USA, 2009. ACM.

[5] Y. Wang and C. Wang. Modeling the effects of timing parameters on virus propagation. In *Proceedings of the 2003 ACM Workshop on Rapid Malcode*, WORM '03, pages 61–66, New York, NY, USA, 2003. ACM.

[6] Wikipedia. Computer virus — Wikipedia, the free encyclopedia, 2016. [Online; accessed 10-October-2016].

[7] S. Xu, W. Lu, L. Xu, and Z. Zhan. Adaptive epidemic dynamics in networks: Thresholds and control. *ACM Trans. Auton. Adapt. Syst.*, 8(4):19:1–19:19, Jan. 2014.