Taking Computer Virus Detection to a New Level

Kurt Bagby
Department of Computer Science, Faculty of Science, University of Auckland
Princes Street, Auckland, New Zealand

Abstract

The battle between computer virus writers and antivirus software writers has been going on ever since the first antivirus program was written. Now, more than ever, it seems that this battle may be coming to a head. With the advent of much more complex viruses, antivirus software and human computer antivirus specialists will have to come up with a new way to deal with the ever continuing problem of viruses. Current antivirus detection techniques such as the classic signature based virus detector employed by Norton AntiVirus are essentially doomed. It seems, however, that Mother Nature herself has provided the very solution that the computer industry needs to win this war. That tool is known as an immune system.

1. Introduction

The first known computer virus arrived sometime during the 1980's. Since they have arrived on the scene, computer viruses have exploded both in number and in their complexity. Overall, they have become a nuisance to both the home user and large corporate businesses. Just like viruses in nature, the amount of damage a computer virus can cause can be next to nothing or somewhat colossal. Likewise, the intent behind virus creation can range from a simple prank to being a carefully devised plan tailor-made for destruction.

The ability of viruses has come a long way. The first kind of "real" viruses were known as stealth viruses. This meant that the viruses tried to hide themselves from being detected. As viruses became more complex their characteristics changed as well. They became polymorphic (to do with encrypted viruses where the decryption method is variable), they used armouring (an attempt to prevent anti-virus researchers from disassembling them), and became multipartite (the ability to infect both normal programs and boot sectors). As time got on even further, combinations of these were all incorporated into one virus.

In the present, viruses have become multi-platform and most recently, metamorphic (“there is no way to decrypt the code of the virus to a constant form” [Schreiner2002]). With these recent advances in the world of virus writing, traditional virus detection methods will not work anymore. Therefore the need for a new way to detect computer viruses is essential in today’s world.

But the increasing complexity of viruses is not the only reason a new virus detection scheme should be sought. Two more reasons for a new detection scheme are:

1. The rate at which viruses are written is quite high. It is hard for the anti-virus world to keep up with and keep track of all the new viruses that are emerging today; and
2. The increase in the world’s interconnectivity via the internet and emailing systems and the increase in the world’s interoperability are making it considerably easier for viruses to spread. Updates in current anti-virus protection systems will not be able to propagate as fast as the spread of new viruses. [Kephart1994]

Thus the need for a more complex computer virus detection scheme is multi-dimensional as well. Rather than depending on the knowledge of already known viruses, computers will have to start defending themselves much like a human immune system defends itself against new viruses. Given this analogy, this paper aims to introduce the concept of a computer immune system and how it is more applicable in today’s world.

2. Current anti-virus techniques

Probably the most widely acknowledged virus detection and prevention system of today is the program known as “Norton AntiVirus”. Norton AntiVirus is a signature based virus detector. The idea behind signature based virus detection techniques is quite simple. Basically, a list of known virus signatures is stored in a file. Signatures are just a small sequence (normally 16 – 32 bytes long) of instructions that represent a virus and are guaranteed to be found in each occurrence and derivation of that virus [Kephart1994]. The signature that the virus is based on is supposedly (sometimes bad signatures have been selected) not expected to be contained in any normal file within a computer system. When Norton AntiVirus is run, it compares programs within a computer to the list of known signatures. If there is a comparison, then the corresponding program is denoted as being of viral nature and the corresponding actions are taken to undo the virus' damage and ultimately, delete the virus.

With the recent advent of metamorphic viruses, a detection technique such as this will not work anymore. This is because metamorphic viruses are never in a constant state and thus signatures cannot be determined for them. Another reason that this technique will fall by the wayside is that the amount of viruses being created each year may be too much for virus researchers to keep up with (in 1994, two or three viruses were being created each day [Kephart1994]). This is also a result of the complexity of viruses. The more complex a virus is, the longer it takes for the virus to be analysed and a signature to be determined. So the combination of increasing numbers of viruses and an increasing complexity in viruses is a losing battle for the anti-virus researchers. Furthermore, with the internet becoming more and more widespread, viruses can travel much quicker around the world and can cause considerable damage very quickly if the intent is there. Therefore we need to find a method which is fast, accurate and has the ability to "spread the news", to combat computers against viruses. This is where Mother Nature herself has provided the solution.

3. An immune system for computers

Based on the immune system for humans and other vertebrates, one of the latest ideas in protecting computers against viruses is to arm them with an immune system of their own. In an extremely simplified version of nature's immune system, what happens is that entities known as T cells recognize particular antigens (unwanted foreign particles or viruses) and have them killed. The idea behind the T cells recognizing antigens is that they recognize anything that isn't part of their host, i.e. they have learnt not to recognize their host.

For the same idea to be used in computers, the computer too has to learn how to recognize what does belong to itself from what doesn't belong to itself. A couple such dynamic anomaly detectors exist, namely *activity monitors* and *integrity management systems*, which try to determine self from non-self. Activity monitors alert users when something that is rarely associated with normal occurs and integrity management systems warn users when something suspicious has happened to their files [Kephart1994]. The problem is that these detectors have a more than acceptable rate of mistaking what is denoted as normal behaviour with non-normal behaviour. Because of this, users then tend to ignore their warnings (kind of like the story about the boy who cried, "Wolf").

The problem with the two aforementioned methods is that they have a problem when determining self from non-self. By supplying computers with their own immune system, they can perform their own training specific to themselves and thus the problem of determining self from non-self will be assuaged. In addition, the need for signature based detection programs for the detection of viruses will not be needed in the fact that users will not have to constantly update the anti-virus program. The program will update itself when the time is appropriate, i.e. when a virus strikes. This doesn't mean, however, that anti-virus, signature based programs will completely disappear, nor will the need for human interaction completely diminish. There will always be extremely exceptional viruses that will need human interaction to solve them, much like when a human becomes extremely sick they normally proceed to see a doctor. Also, the signature based virus programs will still be distributed, just not as often. They will act like vaccines when the treatment of more complex viruses is found.

The question for the moment then is how can an acceptable level of the recognition for self and non-self be determined, and if it can, how will a computer diagnose itself like a biological immune system?

3.1. Determining self from non-self

In [Forrest1994], a method for determining between self and non-self is described which functions in a very similar way to the production of T cells in the human immune system. They form a base testing environment with strings. Firstly, they define a set of strings all of equal length which are denoted as self. Next, they randomly generate strings of the same length and test them against the strings that represent self. If they match a self string then they are discarded otherwise they are stored as detectors (non-self strings). A match is defined as one string being the same as another in r-consecutive positions within the string. Here r is an integer. For example, consider the two strings AHGTKJFS and KFFTKJHJ. Within each of the strings there is a match of TKJ in the fourth to sixth position:

self string	A	H	G	T	K	J	F	S
random string	K	F	F	T	K	J	H	J
position	1	2	3	4	5	6	7	8

If r is set to 1, 2 or 3 then this would be considered a match and the random string would be discarded as it matches self. If there wasn't a match, the string would be added to the list of detectors.

When the required amount of detectors is achieved, the algorithm stops and the list of strings which define non-self have been created. The amount of characters in the alphabet, the length of the string, the size of r and the amount of strings which denote non-self all contribute towards the probability of an anomaly being detected.

This idea is then extended to executable code with characters of a string being defined as instructions or op-codes. Once training has been completed, the program can be checked against the detectors at a later date. If any matches are made then the program is known to have been tampered with. This technique also has the advantage that if someone tries to change the detectors, the same result will occur because the changed detectors will end up matching the self strings. Thus we still know that a modification has been made.

3.2. Diagnosing

In [Kephart1995] an example of how to analyse and diagnose a virus is given. Firstly, if a virus has been detected, the logical step is to search the list of known signatures for a match with the current virus. If there is a match then the appropriate action can be taken otherwise a different strategy needs to be employed. Decoy programs are released which attract the virus to infect it. Several of these need to be released so that the program has several samples of the virus to work with. Several samples of the virus are needed because the algorithm does not have such a detailed understanding of machine code as does a human expert. By comparing the infected samples of the decoy programs with legitimate ones, the algorithm can work out how the virus has attached itself to the host. From here it can formulate the repair language for the currently infected program.

When the repair language has been formulated, a signature for the virus needs to be created so that if the same virus is encountered at a later date, it can be dealt with more efficiently. This task is a non trivial task as signatures need to be selected such that they are common to all instances of the virus but at the same time do not appear in legitimate programs. As a rule of thumb, data is not used to create signatures. Rather executable code is used as this is the least likely to change, although with the recent advent of morphological viruses this provides yet another problem. Once a set of common signatures have been found, the signatures are compared against a substantial number of legitimate programs. The signature with the smallest probability of being amongst the legitimate programs is selected as being the most appropriate signature for the virus and is added to the database of known viruses.

3.3. Spreading the word

As a final solution to the problem of deleting a new virus, other computers need to know how to get rid of the virus if it should happen to attack them as well. In [Kephart1994] a method is proposed whereby if a computer finds way to diagnose a particular virus, as a final step it sends a message containing the signature and repair method for the virus to its immediate neighbours in the network. If one of the neighbours has the virus, it uses the repair program to fix the problem and then sends the same message to all of its neighbours. If a neighbour doesn't have the virus, it just stores the signature and repair program in its database in case the virus should come its way. An uninfected computer does not propagate the message further. This concept is shown in figure 1.

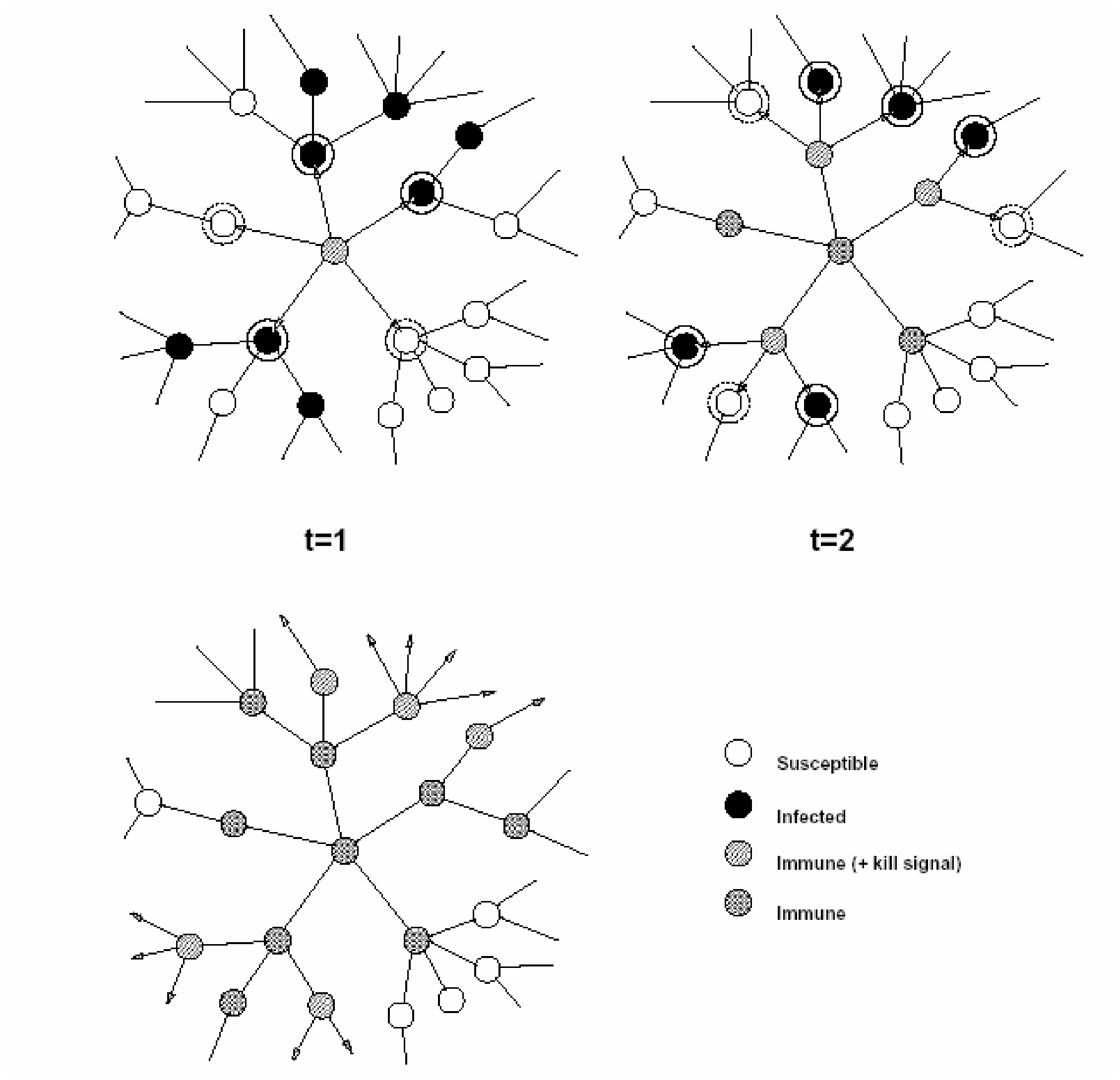

Figure 1. (taken from [Kephart1994])

4. Discussion

There are several problems with the above techniques in determining self from non-self and diagnosing viruses. One event that could occur is that a virus could be written using the commands that make up self. Although this would prove to be very difficult, it is a possibility and shouldn't be discarded. This problem was addressed in [Forrest1994]. They say that if they were to incorporate variable length self strings, the amount of common substrings available for the production of a virus diminishes quickly. A point that wasn't addressed in [Forrest1994] was if it was possible to find commands that are not contained in the self set, and are undetectable by the detector set. If this is a possibility then the list of commands to use in order to create a new virus would increase. The task of determining which commands are unable to be detected by the detector set would prove to be a very difficult task as well. This applies to both of the abovementioned attacks as determining what the commands are for self would take a similar approach.

Another problem which affects this entire system is that when an anomaly is detected, how do we know if it is a virus or not? The anomaly that has been detected could result from the fact that someone has modified a program. If this can occur then another interesting problem arises. How often do we need to train a program to know the difference between self and non-self? If we wanted the detector set to be a current set all the time, then, for executable code, we would have to retrain the system every time we recompiled any code. This would definitely prove to be time consuming, especially since the detector set is formed by the generation of random strings. [Forrest1994] suggests that the process could be sped up by taking away the random selection of non-self strings and employing some other algorithm. They also mention that by using such an algorithm, some regularities could be noticed and thus exploited.

So if we didn't generate a detector set after every time we compiled, how often would we generate a detector set? We could generate the detector set after each session we have with the computer but this could result in a new virus being added to the detector set. This would happen if a virus entered the computer during your session thus creating yet a third problem. Whenever we want to generate a new detector set for a modified program, how can we test whether this new program has a virus or not? Maybe the generation of global detection set could be created for this particular program. By doing this though, we would have to know what types of instructions are going to be in future modifications of the program as well. This would restrict a programmer's ability to create programs freely and would also raise the chances of a virus writer determining what is in the detector set for a specific program. So the trade-off between usability and security arises.

People can argue that a detector set wouldn't have to be created every time a program is recompiled and that a generic detector set would probably suffice. We must remember that viruses aren't going to be sitting around on the phone line waiting for you to change you program so that they can infect your computer, although this is definitely a possibility. If such a virus was written, would it always be waiting there? Such a virus could be there at one point in time, but every time a person logged on to a computer for the rest of that computer's life time? The possibility of this is very slim.

This brings us back to the generic detector set, just how generic should this detector set be and, would one have to be created for every single program within the computer? This brings on the task of determining which programs would belong to one set of programs and which would belong to another. The question of whether a program belongs to more than one set would also crop up and the corresponding program would end up being tested twice for viruses. Is double checking in such a case a good thing or just an efficiency deficiency? Given this, could we then create a detector set that encompasses the whole computer? Although this would be the ideal situation, it would probably be the ideal situation for virus writers as well because the instruction set they would have to create a new virus would be much greater.

This still doesn't completely solve the problem of being able to write programs freely for a programmer though. And the installation of new software for a normal user would still pose a problem. With such a generic set, each time new software is installed, the detector set would have to be recreated. Creating a detector set for the whole computer would definitely be a very time consuming task. Such a task would probably have to take place each time a programmer creates a program as well.

From all of this, we are brought back to the question of how to determine whether a detected anomaly is a virus or not. [Kephart1994] actually uses a combination of activity monitors and integrity management systems to determine whether the anomaly is a virus or not. As mentioned earlier though, such systems have generated a more than acceptable false alarm rate. The exact details of how they used them were not presented though they did say that if the behaviour resulting from such an anomaly was related to other virus behaviour it was very probable that the anomaly was viral. A careful use of the two techniques may provide a more robust detection system.

If we were to imagine that the problem of determining whether a virus is present or not were solved, we can bring about the next argument which involves diagnosing the virus. In the procedure presented in this paper, it is said that several infected decoy programs are needed to determine how to diagnose an infected file. Nothing is mentioned about how computationally expensive this process may be. When the computer decides to fix itself, is the computer supposed to devote itself completely to fixing itself or is the computer expected to carry on working whilst the fixing process continues in the background? This is much like whether a person should stay home or go to work when they are sick. How long is a computer expected to take to fix itself? Ideally, we would want the computer to still be usable when it is "sick" but then there would be performance issues and the problem of determining which processes are allowed to run would also have to be defined as well. This would have to happen because if some processes are allowed to run, they too may become infected. And what happens if the computer cannot find a solution? More the likely, the logical thing that would happen would be similar to what is happening right now when new viruses are detected. But the idea behind the immune system is that such a case should not happen. If it can happen, it should be a very small possibility.

Another problem is signature selection. It has been known that even human computer virus experts have selected bad signatures for viruses before [Kephart1994]. When such an incident occurs users have been known to delete perfectly legitimate files because they were scared of what would happen if they left what they though was a virus alone. And the problem with metamorphic viruses still hasn't been solved. Signatures for such viruses cannot be obtained. When such a virus strikes do we have to diagnose the problem the long way every time, i.e. using the immune system? Such an event also should rarely occur.

Problems to do with the distribution of the solution for any given virus among a network also surface. As mentioned in section 3.3, the virus solution stops propagating once it reaches uninfected computers. Is this solution feasible? The virus could appear in some totally unrelated part of the network which hasn't received a solution yet. Why should the whole process of creating a solution run through again? A solution to this could be that computers ask the entire network to check whether they have a solution for the virus. But this too is very time consuming. The question then is do we set a certain time value for a diagnosis to travel around a network. If so, how do we know exactly how long this time value is? The diagnosis for the virus should reach every computer and people are adding and removing themselves from networks all the time. And then we have the problem of when a new machine arrives on the network. Such a machine doesn't have all the new virus information that hasn't yet been incorporated with the latest anti-virus program. Software would have to be included into computers that can ask a network for all such information the first time they are connected to a network, or computers that are already part of a network could check whether new computers have such information, or both. This is all added complexity and also would provide a nice loophole for virus writers and other types of intruders to exploit. Within networks, it seems that we aren't just creating a computer immune system, but actually a computer network immune system.

5. Conclusion

Viruses are becoming more and more complex every day. Likewise, proposed solutions to diagnosing computer viruses are also becoming more and more complex. In this paper an idea inspired by the human immune system to create an immune system for computers has been explained and various problems associated with its implementation have been discussed.

Creating such a system is indeed a huge step in protecting computers against viruses. It seems that the techniques used to create such a system are still very much in their infancy and much research still needs to be carried out especially with regards to determining self from non-self and whether such an anomaly is viral or not. It seems that there is a definite tradeoff between several items within the immune system's creation. Whether each individual program should know itself, the whole computer should know itself, or if this is user dependant still needs to be determined. The size of the detector set and the memory requirements for detector sets still need to be evaluated. As in any program, there is always the tradeoff between time and how well the program works. This particularly applies to where the computer diagnoses itself. Distributing a diagnosis for a virus is seen as a completely new problem altogether.

Overall, an immune system for computers would be a very sought after commodity. Many questions still remain open and much research still needs to be carried out but from the information presented in this paper, it is clear that the computer immune system is well under way.

6. References

[Deeb2002] Deeb K, Lewis S., "A biological approach to the development of computer autoimmune systems", [Conference Paper] Foundations of Intelligent Systems. 13th International Symposium, ISMIS 2002. Proceedings (Lecture Notes in Computer Science Vol.2366), Springer-Verlag. Berlin, Germany, 2002, pp.514-25.

[Forrest1994] Forrest S, Perelson AS, Allen L, Cherukuri R., "Self-nonself discrimination in a computer", [Conference Paper] Proceedings of 1994 IEEE Computer Society Symposium on Research in Security and Privacy (Cat. No.94CH3444-7). IEEE Comput. Soc. Press, Los Almitos, CA, USA.1994, pp.202-12.

[Kephart1994] Kephart, J. O., "A biologically inspired immune system for computers", [Conference Paper] Artificial Life IV. Proceedings of the Fourth International Workshop on the Synthesis and Simulation of Living Systems. MIT Press. Cambridge, MA, USA, 1994, pp.130-9.

[Kephart1995] Kephart J. O, Sorkin G.B, Arnold W.C, Chess D.M, Tesauro G.J, White S.R., "Biologically inspired defenses against computer viruses", [Conference Paper] IJCAI-95. Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence. Morgan Kaufmann Publishers, San Mateo, CA, USA, Part vol.1, 1995, pp.985-96.
http://www.research.ibm.com/antivirus/SciPapers/Kephart/IJCAI95/paper_distrib.html

[Schreiner2002] Schreiner, K., "New viruses up the stakes on old tricks", Internet Computing, IEEE, Volume: 6 Issue: 4, Jul/Aug 2002
Page(s): 9 -10.