# An OS Integrity Measurement System based on Epidemiology

K. Venugopal Dasarathy
Dept. of Computer Science & Engineering
NITK Surathkal
Karnataka, India

Samuya Hegde
Asst Prof
Dept. of Computer Science & Engineering
NITK Surathkal
Karnataka, India

Radhesh Mohandas
Adjunct Faculty
Dept. of Computer Science & Engineering
NITK Surathkal
Karnataka, India

## ABSTRACT
Consider an analogy when dealing with human diseases, when a person discovers something different happening to him/her, a common course of action is to know if others have experienced the same thing. In this paper we propose a design for an integrity system for a connected network that attempts to measure the degree of infection of a system on the network using an epidemiological model. Furthermore we present the outcome of simulations that model the process of infection over a network and show how the infectiousness degree of a program varies with parameter values of the model.

## Keywords
Malware, rootkits, epidemiology.

## 1. INTRODUCTION
Since the first documented reports of microcomputer viruses in the mid-1980's, they have spread throughout the world. Computer viruses have very similar characteristics to biological diseases; an important aspect is that of a disease epidemic.
Antivirus scanners relying on purely signature based detection face a losing battle with malware today, due to new viruses being released daily.
Behavioral based integrity checkers[6] combined with scanners are a better choice but many legitimate programs also play with essential resources of the system.
Another aspect of malware that is of interest is the need for it to spread, much like a biological disease. The spread of a virus across a network shares common concepts with biological epidemiology[10]. In what is to follow, we describe a epidemiological model for the classification of malware.

## 2. EPIDEMIOLOGY, THE MARKOVIAN MODEL AND THE AIE VALUE
The model we consider is the one defined by Laura Billings et all[1]. It is built on the basic model by Kephart and White[2].
To model the spread of viruses throughout a network, we assume that the network consists of $N$ nodes.
Each node can be in one of two *medical conditions*: susceptible ($S$) or infected ($I$ ). We assume discrete time steps of arbitrary units, so at any given time $n$, $S = N - I$ . Infected nodes can remain infected or become susceptible by being cured. Likewise, susceptible nodes can remain susceptible or. become infected. Denote the probability that a susceptible node becomes infected as $\mu$. This rate depends on two parameters, the connectivity of the network ($c$) and the probability of transmitting an infection ($\beta$). The probability that an infected node becomes susceptible is $\delta$. Fig. 1 illustrates the transitions between these two medical conditions, with their associated probabilities. A discrete-time Markov chain is a dynamical system composed of **S** discrete states. In this Letter, the number of nodes in each medical condition will define a Markov state, forming the pair $(I, S)$. Because $S$ is explicitly defined by $I$ , we only refer to $I$ , and in this case, **S** = $N$+1. At each time step $n$, the Markov chain can change states. We compute the probability of transitioning from state $I$ to state $I'$

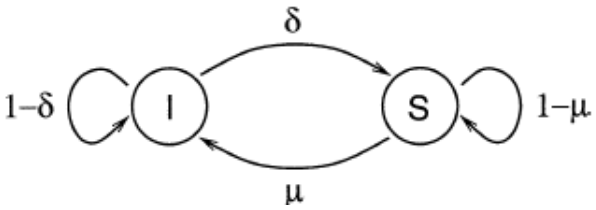



**Fig 1: State Flow diagram between the two states**

Now what we are interested is in finding the probability to infect one more node. Thus $I'=I+1$
From[1], we derive this probability as

$$P_{I,I'} = \sum_{x=1}^{min\{I,S-1\}} \binom{I}{x} (\delta)^x (1-\delta)^{I-x} \binom{S}{x+1} (\mu)^{(x+1)} (1-\mu)^{S-(x+1)} \quad (1)$$

This is the probability that a particular virus infects one more node. At this point we define a value to describe the infectiousness of a program. We shall call this value as AIE(Am I Evil). If R is the observed rate of spread, then we define

$$AIE=( P_{I,I'} + R)(0.5) \quad (2)$$

In simple words, the AIE is an estimated threat measure of malware, based solely on its spread patterns for a network.

The application of such a global measure of threat for malware over a network is desirable as it may not be possible to identify malware by monitoring local behavior alone. The proposed architecture described in section 4 incorporates this concept.

# 3. THREAT ANALYSIS

This project is an inquiry into the behavior of malware, and it concerns with the spread of malware in particular.
Some key points to be mentioned are described below.

## 3.1 Rootkits and Hooks

An aspect of focus here is on rootkit detection, particularly system hooks. Three common hooks that are common amongst rookits are Import Address Table (IAT) hooks, System Service Dispatch Table (SSDT) hooks and Interrupt Descriptor Table (IDT) hooks [4].

Since there can be legitimate software that hook these tables, just a mere detection of these hooks on a single system seems insufficient.

## 3.2 Relationship between Malware and Rootkits

Rootkits' relationship to malware is twofold: To put a rootkit on a computer, other malware has to load it. And after the rootkit is loaded, it's often used to hide more malware. Rootkits created with malicious intent (some rootkits are benign or even beneficial) collectively make up a specific category of malware; however, not all malware programs are rootkits.
Thus the detection system must also resolve dependencies in addition to rootkit detection.

## 3.3 The Intent to Spread

Most malware have the intent to spread from host to host. Any computer or network connected to the Internet is a viable target for a malware or rootkit attack.

# 4. SYSTEM ARCHITECTURE

The basic block diagram of the architecture is depicted in figure 2.

The block diagram depicts two functional entities called the Local Integrity monitor (LIM) and the ISever (IS). As the name suggests, the LIM is a local monitoring entity running on any of the nodes of the network. It is upto the LIM to detect local discrepancies and to intimate the IS the aforementioned. The block diagram depicts two functional entities called the Local Integrity monitor (LIM) and the ISever (IS). As the name suggests, the LIM is a local monitoring entity running on any of the nodes of the network. It is upto the LIM to detect local discrepancies and to intimate the IS the aforementioned

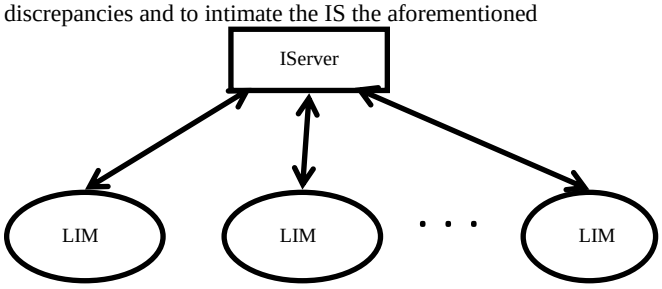



**Fig 2: Global view of the system**

## 4.1 Local Integrity Monitor (LIM)

This is the integrity monitoring system running on local machines. The services offered by the Local Imonitor are as follows

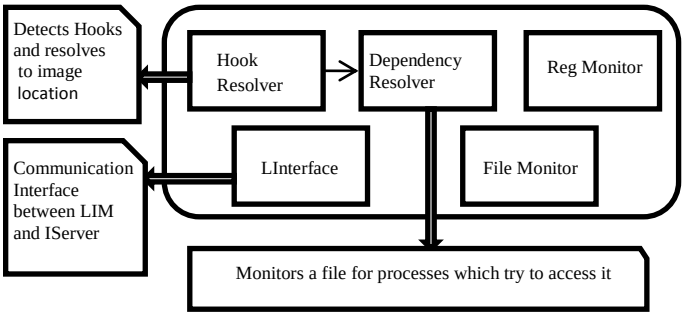



**Fig 3 Components of the LIM**

### *4.1.1 Rootkit Resolver*

This module is responsible for peering into the SSDT, IAT and IDT hooks and resolving the address where the hook points to.
If a hook is found, it computes the SHA-1 hash of the resolved module which is sent to the IServer.

### *4.1.2 Dependency Resolver*

Once a hook has been discovered, the Dependency Resolver monitors the resolved hook to see if any other process tries to obtain a handle to it. It also tires to find all open handles by the monitored process, and also logs if the process is hidden or not. It also computes hash of this process to be sent to the IServer.

### *4.1.3 LInterface*

Is the communication interface between the LIM and the IServer.

### *4.1.4 File Monitor*

The IMonitor maintains a store of hashed values of files which are tagged unsafe by the IServer. This module prevents/ prompts the user if an attempt is made to load such a file

### *4.1.5 Reg Monitor*

Watches a list of registry keys, updated by the IServer.

## 4.2 IServer

It is up to the IServer(IS) to fix up probabilities and classify the discrepancies as good or bad.For example, say an LIM detects an SSDT hook, but not all hooks are bad, so it tries to resolve the address of the hook. On resolving the module(which is probably a sys file) it calculates the hash of the file and sends it to the IS. The IS then verifies whether the hook is malicious by checking its database for the file. The maliciousness is dependent on number of number of LIMs that reported it, the locality of those that reported it and probably more that are yet to be discovered. Also, if the IS tells a LIM that a particular file is malicious, the LIM then uploads the file to the IS, where the file is analyzed by security analysts, who come up with

solutions. The LIM is also capable of receiving solutions from the IS, and thus recover.

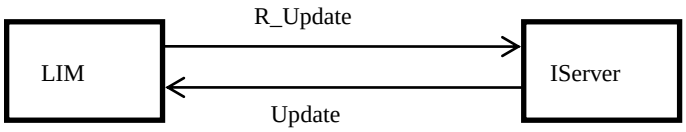



**Fig 4: Communication between the LIM and IServer; the traffic between the components is encrypted.**

## 5. SYSTEM OPERATION

If the LIM detects a discrepancy (a kernel hook perhaps), it tries resolves the cause to a module (if possible). On obtaining it, it computes the hash of the module and compares it with it's own database of stored hashes.

In the database, each detected file has a value associated with it that represents its degree of maliciousness. This value that we shall call "AIE" (Am I Evil) has range(0-1) where 1 means a certainly malicious file. The calculation of this value has been described.

The LIM only refers to the IS when it detects a previously unknown module, or one whose hash is less than a critical vale. (set by user). The relationship between the LIM and IServer is shown in figure 4.

When the LIM receives the UPDATE message, it chooses whether to update it's database or not. If the LIM does not have the hash in it's database, and the AIE value in the UPDATE message is less than the critical value, it may choose to ignore it.

The flow diagram between various components of the LIM is shown in figure 5. When the Hook monitor detects and resolves a hook, it adds the complete filename, (say \??\C:\hide.sys) to Hlist, which is

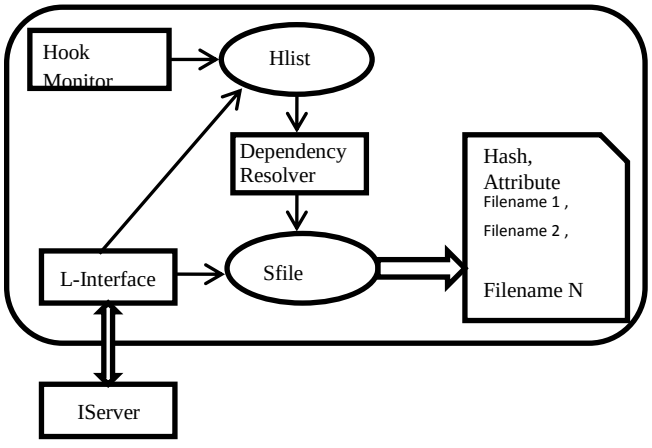



**Fig 5: Conceptual flow between the components of the system**

a shared memory buffer. The dependency resolver monitors the filename from hlist to reveal additional dependencies. Each dependency is stored in stored in a unique file (Sfile), the name of which is present in Hlist.

An Sfile is a file that is written by the dependency resolver when it monitors detected hooks. Each Sfile has the following: SHA-1 hash of the file, AIE, it's attributes followed by a list of filenames (which is also an Sfile) which denote the dependencies of the file . Finally the LIM sends the packet to the IS, whose format is shown in figure 6.The attributes filed is a 16 bit value whose format is shown in figure 7. The attribute field is indented for future work and is not used by the server for classification or identification of malware. However, since it is worth noting that most malware arise from existing code, a classifier can also be

The IS recalculates AIE on every request as

$$\textbf{\textit{AIE(t)}} = \textbf{\textit{AIE(t-1)}} + \textbf{\textit{Average}}(\textit{sum of AIE of dependencies})/\textbf{\textit{AIE(t-1)}} \quad (3)$$

## 6. SIMULATION AND RESULTS

The network was simulated with 70 nodes by varying the different parameters. The goal was to test it for different behaviors and observe the generation of AIE for each test case. As expected, the case where a program hardly spreads gives a small value of AIE over time. Whereas a virus that spreads at fast rates escalates the AIE at high rates.

.Also a virus that spreads at some rate but having a high cure rate eventually dies out, hence AIE value tends to decrease for such a case.

In all there are three parameters that affect AIE, i.e. cure rate, network connectivity and infection rate. Figure 7 shows the graph of a scenario where the infection rate of the system is greater than the cure rate.

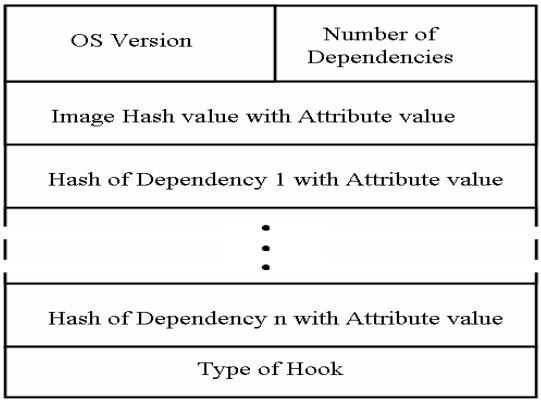
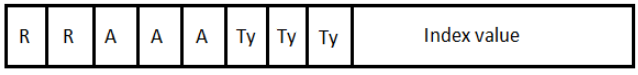



**Fig 6: Packet format of R_Request sent by LIM to IServer**




R- Reserved
A- Attribute
Ty- Type of Hook
Index Value- A byte value that indicates the index of the hook in the table(IAT,IDT or SSDT)

**Fig 7: The Attribute field in the R_Request packet**

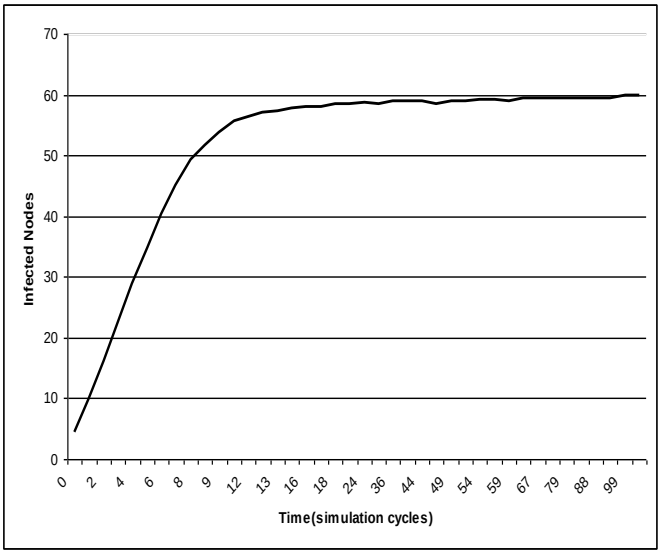



**Fig 8 Time series data for the simulation for 70 nodes with $\beta$ = 0.1200,$\delta$ = 0.2000, and $c$ = 0.0505.**

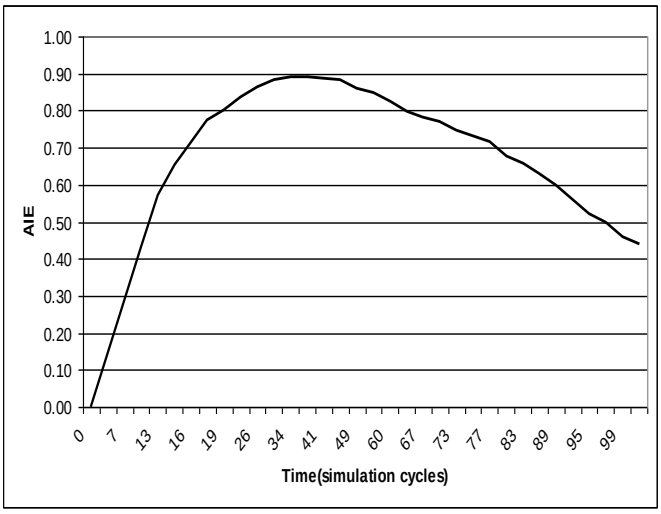



**Fig 9: Shows the AIE value v/s time for a simulation for 70nodes with $\beta$ = 0.1200, $\delta$ = 0.2000, and $c$ = 0.0505.**

In figure 9, it is seen that as the infection does not spread after 60 nodes, the AIE value tends to decrease. This seems intuitive as the number of seemingly infected nodes is 60 but still no new nodes are being infected. Thus the program's AIE value tends to decrease

## 7. CONCLUSIONS

In this paper we have proposed a design for an integrity measurement system, which uses heuristics from computer epidemiology. We have validated the proposed integrity measurement system by simulating the conditions for a viral epidemic. Tests from the simulation exhibit expected desirable behavior of the system. From the simulations, it can be verified that if the infection rate of a virus is higher than the cure rate, the AIE value escalates. Thus the proposed integrity measurement system is capable of successfully detecting rootkits and measuring severity of infectiousness of the system under test.

The components of the proposed system are designed to be generic. The future work involves, 1) Extending the components to encorporate higher details of rootkit detection; 2) Embedding a signature based detection in the local integrity monitor.

## 8. REFERENCES

[1] Lora Billings, William M. Spears, Ira B. Schwartz "A unified prediction of computer virus spread in connected networks" Physics Letters A 297(2002) 261-266J

[2] J. Kephart, S. White, in: Proceedings of the 1991 IEEE Computer Society Symposium on Research in Security and Privacy, 1991, pp. 343–359.

[3] N. Bailey, The Mathematical Theory of Infectious Diseases and Its Applications, Oxford University Press, New York, 1975.

[4] Allen W.H., Ford R. , " How Not to Be Seen II: The Defenders Fight Back" Security & Privacy, IEEE, Nov.-Dec. 2007,pp. 65- 68

[5] Y.-M. Wang et al., "Detecting Stealth Software with Strider GhostBuster," *Proc. IEEE Int"l Conf. Dependable Systems and Networks (DSN 05)*, IEEE CS Press, 2005, pp. 368–377.

[6] James Butler and Sherri Sparks. "Shadow walker: Rasing the bar for windows rootkit detection." In *Phrack 63*, July 2005.

[7] Mihai Christodorescu, Somesh Jha, Sanjit Seshia, Dawn Song, and Randal Bryant. "Semantics-aware malware detection." In *Proceedings of the 2005 IEEE Security and Privacy Conference*, 2005.

[8] James Butler and Greg Hoglund. "VICE–catch the hookers!" In *BlackHat USA*, July 2004. ttp://www.blackhat.com/presentations/bh-usa-04/ bh-us-04-butler/bh-us-04-butler.pdf

[9] Joanna Rutkowska. "System virginity verifier: Defining the roadmap for malware detection on windows systems." In *Hack In The Box Security Conference*, September 2005

[10] J. O. Kephart, S. R. White and D. M. Chess. "Computers and Epidemiology". IEEE Spectrum, 20-26. May 1993.