High Performance Network Worm Detection Engine using Memory Efficient Circular Buffer

Muhammad Fermi Pasha[††], Ahmad Manasrah[††], and Saravanesh Supramaniam[††], Rahmat Budiarto[†], Sureswaran Ramadass[†]
[†]National Advanced IPv6 (NAv6) Center, University of Sains Malaysia
Level 6, School of Computer Sciences Building, University of Sains Malaysia
11800 Penang, Malaysia
[††]Research and Development Division, iNetmon Sdn. Bhd.
Suite 242, Kompleks Eureka, University of Sains Malaysia
11800 Penang, Malaysia
{rahmat, sures}@nav6.org, {fermi, ahmad, saravanesh}@inetmon.com

Abstract

This paper presents the implementation of a memory efficient circular buffer to ensure a high performance network worm detection engine. A worm detection engine's primary function is to detect the existence of worms in a particular network. The method of worm identification may vary depending on the engine. However, all approaches to worm detection share the same need to receive and process vast amounts of data in their quest for worms. This demands the existence of a buffer to store the data as it is being processed in order to avoid bottlenecks, memory overloading, and eventually packet loss. This paper proposes the implementation of a highly efficient circular buffer technology to support a high performance network worm detection engine. It is demonstrated, through experiments, that our proposed circular buffer technology is better compare to conventional memory buffers due to its ability to work in parallel and rotate incoming and outgoing data through an array of file buffer locations.

1. Introduction

Recently, worms have emerged as a serious threat to networks worldwide. Worms are basically computer programs which posses the ability to self-replicate and send multiple copies of themselves to all nodes inhabiting a network. They mainly spread by exploiting inherent vulnerabilities in our operating systems. Realizing this state of affairs, many worm detection engines have been conceptualized to combat this rise in worm attacks. They function by analyzing the traffic on a network, specifically the content of packets being sent and received. Due to the heavy processing load involved, these engines require a memory buffer to store data on the packets as they are being captured and subsequently analyzed.

The main focus of this paper is the implementation of our proposed circular buffer technology to enhance the capability of the worm detection engine. A circular buffer is an efficient method of temporary storage allocation which entails the rotation of data through an array of buffer positions. In a circular buffer, the data writer advances one step every time new data is entered into the buffer. Once the end of the buffer is reached, this process is restarted once again from the beginning of the buffer. Data reading is done in the exact same manner.

A circular buffer holds several advantages when compared to a conventional buffer. Firstly, it ensures approximately constant-time insertion and removal of data values. In addition, it also avoids the producer-consumer conundrum by enabling the packet analyzer to read up the packets from the circular file buffer in a smooth and efficient manner. This process is done concurrently with the insertion of data by the packet capturing engine. Careful calibration is done to ensure that the buffer writing process is done marginally faster than the packet analysis to avoid buffer overflow. All these will help to ensure a highly efficient worm detection engine.

Two High Level Circular Buffer (HLCB) algorithm, which splits the packet capturing and analysis modules, are also proposed to ensure optimal buffer efficiency and to control our proposed circular buffer technology. The details of this circular buffer technology will be presented at Section 3.

Looking at other point of view, our proposed circular buffer technology is totally software based and no special hardware required. Most of work done in the field trying to implement efficient buffering is by implementing hardware based buffering [1][3] or by implement the buffering at kernel level [2][7] which is definitely will be a good and fast solution but expensive.

2. Worm Detection Engine Architecture

Before expounding on the worm detection engine architecture we have to firstly understand the inherent nature of worm attacks. Worms differ from viruses, in a way that they do not require an attachment to an existing program. Instead, they cause harm by a variety of ways including bandwidth consumption, whereas viruses affect files on a computer. Worms, destructive or otherwise, leave behind unique signatures on the packets that they are transmitted upon. These eclectic signatures act as beacons to signal the existence of worms in the vicinity. It is part of the individual packet payload. Potential signatures can be as wide ranging as file attachment names, file extensions, email subjects, email content and key words amongst others.

Therefore, the core principle behind any worm detection engine is an effective and rapid identification of these tell tale signs to identify worms before they spread. And this is what we applied to our worm detection engine. Our worm detection engine will constantly listen to incoming/outgoing packets from the network in an active or passive manner. Subsequently, it will match these packets to an inherent database of defined worm signatures. The examples of detectable payloads by our worm engine are shown in Figure 1.

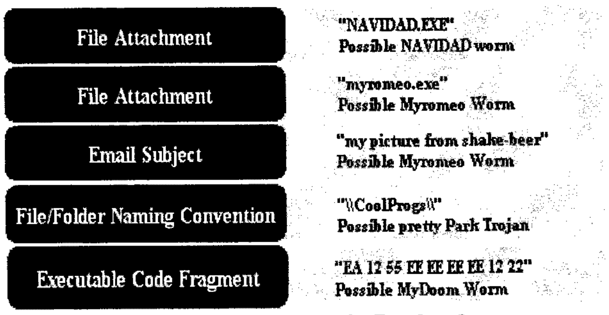

Figure 1: Detectable Payloads

The most important features of our worm detection engine include the ability to:-

- Detect viruses on-demand
- Log the results of virus detection endeavors and inform the user of possible worm activity
- Perform vital administrative functions within a stipulated period.
- Suggest a solution on how to remove the worm.

Figure 2 visualize the architecture of our worm detection engine. The underlying structure of our worm detection engine includes a packet reader module, a worm parser and a worm alert module. The first module, packet reader, primary functions is passively listening to the stream of all incoming and outgoing packets.

Once this is done, it filters and parses packets as they arrive. Being a passive monitoring tool this module ensures that network packets can be captured without any repercussions to network resources in terms of lag or additional bandwidth. The second component to this engine is the worm parser. True to its name, it parses all the captured packets from the packet reader module. It reads through the stream of data in each packet and scans for potential worm signatures. This is done by cross referencing it to an extensive worm database. In the event that a potential worm is discovered, this module sends a message to the worm alert module. This module reacts by sending out an alert onscreen to inform the user that a suspected worm has been identified.

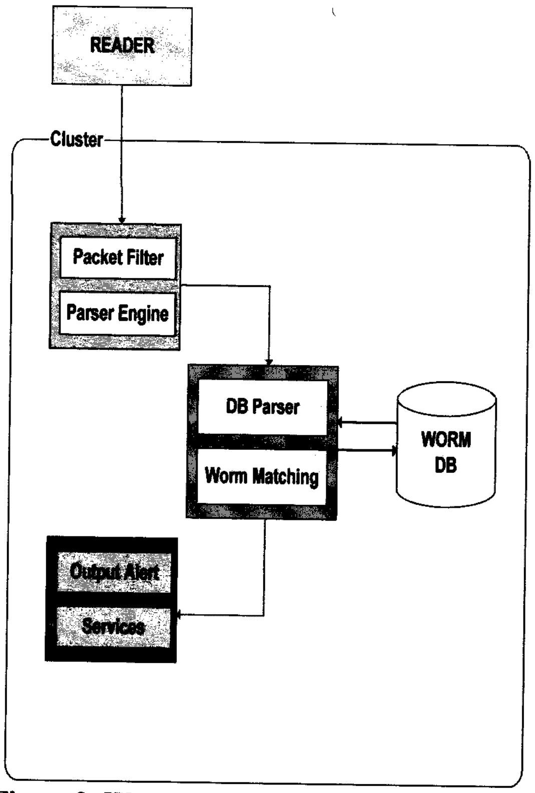

Figure 2: Worm Detection Engine Architecture

3. Memory Efficient Circular Buffer

Here we propose new memory efficient circular buffer technology. The circular buffer is designed in a way that it will avoid packet loss. We also propose new High Level Circular Buffer (HLCB) algorithm used to control our circular buffer technology. The algorithm will split the activity of capturing packets and analyzing packets into two separate processes. That is, instead of using threads, which is more likely to be error prone and cannot run concurrently, the proposed circular buffer algorithm uses different operating system process to do the task. Hence, the issues of racing threads, where the capture thread and the analyzer thread race each other to gain spot at the CPU, can be avoided. Figure 3 depicts the architecture of the proposed circular buffer.

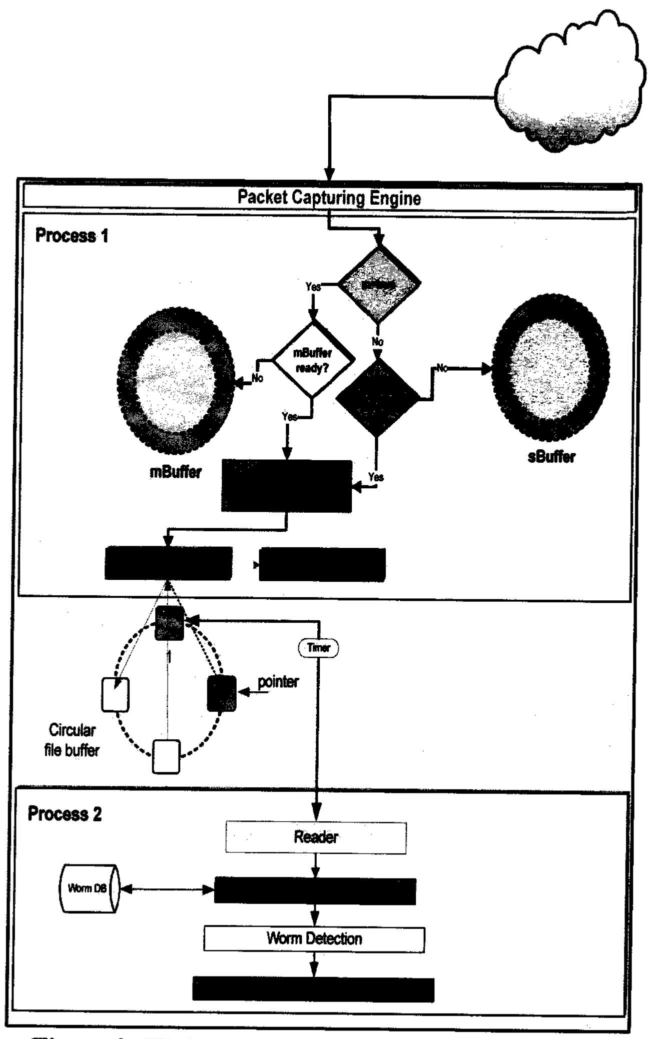

Figure 3: High performance file-based circular buffer technology architecture

Since the proposed circular buffer technology use two different processes to allow fast processing, the proposed HLCB algorithm is divided into two parts, HLCB1 and HLCB2. The first one is designed to control the first process which is part of the packet capturing process, and the second one is specifically designed to enable second process to read up the packets from the circular file buffer smoothly avoiding the producer-consumer problem. Both HLCB algorithms are presented in the following subsection.

3.1 HLCB1 Algorithm

Below is the algorithm of HLCB1 on the first process:

```
Start
   Start 1 sec timer.
   Initialize the 10 files.
   Initialize active-file-pointer to 1.

   Get packet from CaptureEngine.
     If m is true
         Dump the packet into mBuffer and set
          m to false.
     Else
         Dump the packet into sBuffer and set m
         to true.

     Every 1 sec,
       Try locking the file pointed by
        active-file-pointer.
        If file is locked by other process
          Wait and keep trying, and keep
           reading packets
        If m is false
            Dump mBuffer into the locked file.
         Else
            Dump sBuffer into the locked file.
       Unlock the file upon finish writing.
      Increase active-file-pointer by 1.
 End
```

3.2 HLCB2 Algorithm

Below is the algorithm of HLCB2 on the second process:

```
 Start
   Start 1 sec timer.
   Initialize active-file-pointer to 1

   Every 1 sec,
     Try locking the file pointed by
     active-file-pointer.
      If file is locked by other process
        Wait and keep trying.
      Else
        Open the file and read the content
        into Reader.
     Unlock the file upon finish reading.
     Increase active-file-pointer by 1.
 End
```

4. Experiments and Results

The experiments were carried out to compare the performance of our proposed circular buffer technology with the common memory based buffering techniques. The following environment settings are used during the testing.

- x86 processor 2.4GHz
- 1GB RAM
- Windows XP SP2
- 10/100 Base Ethernet connected into LAN with link speed of 100Mbps.

For the purpose of this experiment, we implement a commonly known memory buffer. To make the comparison fair, we also design the memory buffer to be circular. This memory buffer implementation is using multi-threading to do the read and write into the buffer. Hence, the capture engine thread will capture packet from the network and write it into the memory buffer and the reading threads will read the packets from the memory buffer once available.

Technically speaking, memory buffer should be faster opposed to our proposed file based circular buffer technology. But the results presented on Table 1 and Table 2 clearly shows that our proposed circular buffer technology outperformed the memory buffer implementation. Our circular buffer technology can maintain packet capturing and analyzing at much higher speed without loosing any single packets. In worm detection, loosing even just 1 packet meaning can degrade the detection accuracy as it is possible that the loosed packet was the one that contain worm.

Table 1: The proposed file based circular buffer technology

Circular File Buffer				
Packet speed (pps)	[illegible]	[illegible]	[illegible]	[illegible]
500	25894	8.2	89%	0%
1500	[illegible]	[illegible]	[illegible]	[illegible]
2000	33634	9.51	88%	0%
2500	[illegible]	[illegible]	[illegible]	[illegible]
3000	38688	10.2	87%	0%
3500	[illegible]	[illegible]	[illegible]	[illegible]

Table 2: Memory Buffer Experimental Results

Circular Memory Buffer				
Packet speed (pps)	[illegible]	[illegible]	[illegible]	[illegible]
500	25923	6.3	89%	0%
1500	[illegible]	[illegible]	[illegible]	[illegible]
2000	115829	40.71	88%	5%
2500	[illegible]	[illegible]	[illegible]	[illegible]
3000	141819	99%	37%	59%
3500	[illegible]	[illegible]	[illegible]	[illegible]

The following discussions are best to further explain the results:

- During low traffic (500pps), the performance of memory buffer is equivalent or better in terms of CPU usage since writing to memory is faster and requires no CPU power compare to writing to file. This is also due to the fact that at this speed, the threads still can run one after another effectively.
- Memory buffer performance starts to degrade when the traffic is above 1500pps. The threads starts to race to gain place at the CPU to be executed which in turn causing the CPU to increase and the memory usage is also increasing since the reading threads has lower priority and slow in reading the packets and therefore the capture threads filling the buffer fast. Because of this, the system starts to loose packets.
- During high traffic (3500pps) we can see that our proposed circular buffer technology can cope and work faster and therefore maintaining a high performance of our worm detection engine. At this speed, the memory buffer implementation loose 63% of the packets transmitted in the network because the buffer is already full and yet the reading thread don't have enough chances to be executed by the CPU to read the packets from the buffer.

5. Conclusion

In this paper we have presented a novel file based circular buffer technology to support a high performance worm detection engine. This circular buffer technology is controlled by two high level circular buffer algorithms named. We have shown that using a file as opposed to memory buffer is not necessarily slower. The trade off is well paid since at high speed network processing power and the ability to run concurrently and in parallel but a bit slow on writing to file buffer are much needed than the ability to write faster to memory buffer but had to share the CPU between the threads.

Lastly we conclude that an effective worm detection engine must consider two important factors. Firstly, the engine has to be fast enough to detect any potential worm in a network before it has the time and space to cause significant damage. This is where our circular buffer technology greatly contributes as if we loose packets is almost the same with we loose a worm and let it slipping through the network undetected. Secondly, the worm database used has to be constantly updated to enable it to stay one step ahead of the latest worm varieties.

References

[1] S. Iyer, R. R. Kompella, and N. McKeown, "Techniques for fast packet buffers," In. Proceedings. of GBN 2001, Anchorage, 2001.

[2] L. Deri, "Improving Passive Packet Capture: Beyond Device Polling" in Proceedings of SANE 2004, October, 2004.

[3] H. Y. Cho and W.H.Mangione-Smith, "Fast Reconfiguring Deep Packet Filter for 1+ Gigabit Network" in Proceedings of the 13th Annual IEEE symposium on FCCM 2005.

[4] U. Nordqvist and D. Liu, "Power Optimized Packet Buffering in a Protocol Processor" in Proceedings of IEEE International Conference on Electronics, Circuits and Systems (ICECS), 2003.

[5] S. Barbagallo, M.Lobetti Bodoni, D.Medina, G.De Blasio, M.Ferloni and D.Sciuto, "A Parametric Design of Bui-in Self-Test FIFO Embedded Memory," in IEEE International Symposium on Defect and Fault Tolerance in VLSI Systems, 1996.

[6] F. Castaneda, E.C.Sezer, J.Xu, "WORM vs. WORM: Preliminary Study of an Active Counter-Attack Mechanism" in Proceedings of ACM Workshop on Rapid Malcode (WORM'04), October, 2004.

[7] N.Weaver, D.Ellis, S.Staniford, and V.Paxson, "Worms vs. perimeters: the case for hard-LANs" in Proceedings of 12th Annual IEEE Symposium on High Performance Interconnects, 2004.

[8] j-Protect, A worm detection and protection software by iNetmon. http://www.inetworm.org