Power Efficient Secure Web Servers

Sujatha Sivabalan, P. J. Radcliffe
(Corresponding author: Sujatha Sivabalan)
The Department of Electrical and Computer Engineering& RMIT University
Swanston Street, Melbourne, Australia
(Email: s3365148@student.rmit.edu.au)
(Received Feb. 14, 2017; revised June 5, 2017)

Abstract

The power consumption of web servers and associated security devices is becoming an increasing issue both from an economic and environmental perspective. This paper analyses the power consumption of both security software and web server software and concludes that traditional architectures waste energy with repeated transitions up and down the TCP/IP stack. This contention is proved by comparing the energy usage of a traditional architecture and a new architecture whereby IDS functionality is moved into the web server and all operations share HTTP packets. Based on these findings we propose a novel alternative power efficient architecture for web servers that may also be usable in other network systems.

Keywords: Central Processing Unit CPU; Intrusion Detection System IDS; Intrusion Prevention System IPS; Web Server

1 Introduction

our world is steadily accelerating towards an Internet based economy where web servers are most significant. In an enterprise or commercial data centre, the power usage of web servers [13, 26, 29] is becoming a major concern, particularly where the web servers need to handle a heavy traffic load or may be subject to Denial of Service (DoS) attack [38]. It is desirable to reduce the power consumption in such systems both from an environmental and economic point of view. Furthermore, devices that can cope with high traffic loads are very expensive as well as consuming considerable power [6, 17, 18, 20, 22, 35, 37]. The web server system in a web server, or web server farm, consumes power for different reasons. Such systems use power to provide web services for the ingress traffic and for a number security functions that may be separate boxes or virtual machines.

The goal of this paper is to analyse the power usage of the security software or devices within the system. Such a comparison is traditional impossible as the software resides on a variety of boxes which each box having different internal hardware and CPU types. Even in a cloud environment, a software application such as an IDS may be run on different types of CPUs and this will confuse any attempt to measure power consumption. The exception here is Software Defined Networks (SDN) where security applications can all be run on the one hardware platform [40]. This approach is used to achieve the goals of this paper, all programs can be run on the same hardware and their power consumption compared by measuring CPU utilization.

This paper focuses on IDS/IPS systems as they have the most scope to be absorbed into other devices. Two well-known and widely used IDS/IPS programs were selected for testing, Bro [8] and Snort [31]. Bro [21] is capable of sophisticated packet analysis and a full IDS function. Snort [1, 3, 9, 15, 21] has some IPS capability but it is essentially an IDS where detection based on packet signature matching.

The key research outcomes described in this paper include:

- Power consumption analysis of two traditional security applications Bro and Snort.
- Power consumption analysis of a new daemon developed [27, 28] by us that integrates into Apache.
- Based on the findings, a novel alternative architecture for webservers is proposed that can reduce total power usage.

This paper organized as follows: Section 2 reviews web server power consumption and discusses existing work related to reducing IDS and security device power consumption. Section 3 measures and analyses the power consumption of BRO and Snort. Section 4 extends the experiments by replacing Bro and Snort with an IDS daemon that works with Apache. Section 5 uses the findings from the experiments to propose a novel architecture that reduces CPU load and hence power consumption. Section 6 provides conclusions to the work and offers some further research directions.

2 Related Work

2.1 CPU Utilization and Power Consumption

Power management of web servers systems is an area of ongoing research. Sharma *et al.* [26] states that power consumption on a web server is economically and ecologically significant. In 2002 Bohrer *et al.* [7] started focussing on managing power consumption in web servers and their experimental results shows that the CPU consumes the largest fraction of the system Power. Mukherjee *et al.* [24] experimentally observed that Disk and Memory I/O usage had a negligible effect on server power consumption, whereas the CPU utilization is linearly related to power consumption. Given these observations, it is important to minimize CPU utilization in order to reduce power consumption. Minas *et al.* [23] mentioned that CPU usage in a web server is the most significant factor in power consumption and his results on a quad core Intel processor shows that for a given CPU, the power consumption is linearly related to the CPU utilization. Other processors have a similar relationship. From these observations, we can conclude that it is appropriate to use CPU utilization as a proxy for power consumption of software providing that the CPU type and hardware is the same. A weakness for evaluating the power usage of several devices is that the CPU can vary between physical devices or even virtual machines in the cloud and so cumulative CPU utilization is not a good measure of power usage.

2.2 IDS Power Consumption

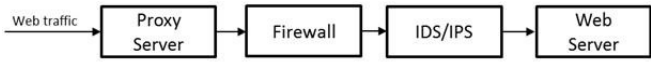

Figure 1: Web server in a Host-based system.

In a host-based system as shown in Figure 1, devices such as proxy servers, firewalls, and Intrusion Detection/Prevention systems (IDS/IPS) secure web servers. These services are crucial as they can eliminate or reduce attack traffic. It is notable that in Figure 1 the traffic is received, analysed, and transmitted three times before getting to the web server. Only the analysis activity has value with the repeated reception and transmission being an overhead that consumes power. The IDS function is of particular interest as if it is found to be inefficient in terms of power use then its functionality might be able to be absorbed into other devices. There is significant research on ways to improve the performance of IDS/IPS that reduces CPU load and hence power consumption. Zaman *et al.* [39] implemented a light weight IDS to overcome resource consumption but this achieved poor detection rate based on two different approaches. Wheeler *et al.* [37] introduced three levels of parallelism using node, component and sub component level and stateless analysis but this achieved little in the way of power savings.

Vasiliadis *et al.* [35] implemented a multi-parallel IDS architecture (MIDEA) for high-performance processing and stateful analysis of network traffic and while this worked well it is expensive and complex to implement [9]. The major flaw of these [35, 37, 39] methods is repetition [9] where one packet undergoes the reception, inspection, and transmission several times hence there is waste of CPU computing time and electrical power. Waleed *et al.* [9] suggested parallel NIDS to reduce packet drop rate and process more packets in less time during heavy traffic, however sudden increases in traffic will result in packet loss. This approach requires many IDS in parallel that in turn increases the CPU load and leads to increased power consumption and cost. Several researchers [16,26-30] have used various methods to implement a lightweight IDS approach but the overall result has been to increase the workload on the web server. The methods reviewed all have at least one of two flaws that contribute to power consumption:

- There is multiple reception, analysis, and retransmission of HTTP packets which costs computing time and electrical power.
- All the security devices placed at the network ingress point must handle the full load of normal and DDoS traffic thus requiring powerful, power hungry and expensive devices.

3 Power Analysis of Traditional Approaches: BRO and SNORT

3.1 Experimental Setup

This section describes the experimental set up used by the authors to measure the CPU load used by Bro and Snort and then discusses the result.

The two experimental setups depicted in Figure 2 & Figure 3, both used i7 desktop computers running Linux Mint 13. Figure 2 illustrates the low traffic test bed that used only two systems; one serves as the victim machine with the detection functionality and other serves as the attacker machine. Figure 3 illustrates the high traffic test bed that used twenty computers, one was the victim web server plus the IDS functionality and the others servers acted as the zombie machines that attacked the victim. The CPU load measurement is the load on one CPU of a multi-core processor and it is independently measured for both the web server and IDS function. The server on the victim machine was an Apache 2 web server with three web pages including images. The attacker machines generate random DDoS traffic aimed at these web pages. Each IDS only sees the incoming web page requests as the outgoing web pages come from the system's server and are assumed safe.

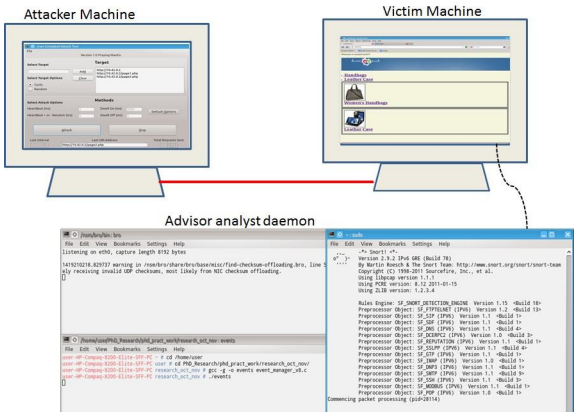

Figure 2: Experimental set up low attack traffic

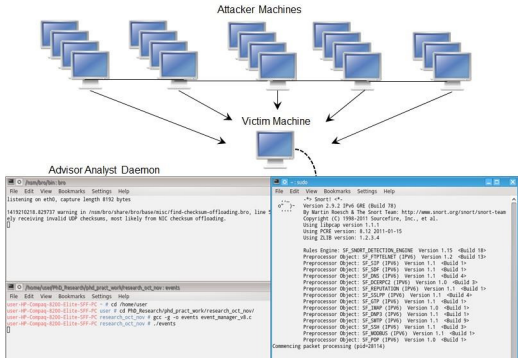

Figure 3: Experimental set up high attack traffic

Snort rule:

```
alert tcp
$EXTERNAL_NET any -> $HOME_NET 80
(msg:"http_request"; content:"GET"; http_method; sid:10000;)
```

Bro rule:

```
event http_request(----)
{++http_request_count;}
event bro_done()
{print fmt("Bro is done");
print fmt("http_request_count: %s", http_request_count);
```

Figure 4: Bro and Snort programming without HTTP analysis

Snort Rule:

```
alert tcp
$EXTERNAL_NET any -> $HOME_NET 80
(msg:"Over burst limit"; content:"GET"; http_method;
 classtype:web-application-activity;
 threshold:type threshold,track by_dst,count 10,seconds 11;
 sid:10000; rev:1;)
```

Bro Rule:

```
module HTTP;
event http_request(-----)
{ if(cSid$resp_h in Site::local_nets)
 { if(method == "GET" && cSid$orig_h == orig_h)
  {  burst_limit + 1;
    set_duration - 1 secs; }
  else
  { burst_limit = 0;
  set_duration = 11secs;
  }
  if(burst_limit >= 10 && set_duration == 0 secs)
  {NOTICE ([$note = DDoS_attack, $msg=fmt("over burst limit")]);
   burst_limit = 0;
   set_duration = 11secs; }
  }
}
```

Figure 5: Bro and Snort programming with HTTP analysis

3.2 Implementation

Bro-IDS version 2.2 and Snort version 2.9 used. In order to compare the CPU utilisation, the scripts written in two ways, to receive the HTTP traffic but do nothing, and receive the traffic and apply analysis. In Snort, rules were created and saved in myrules.rules. The main concern is to execute only this rule by commenting out all other rules and included only myrules.rules in snort.conf file. For Bro the events are created for module HTTP and saved as $PREFIX share bro site myrules.bro. All bro scripts executed from command terminal using.\bro–i–eth0 myrules.bro. Figure 4 & 5 shows a very simple script using Snort and Bro. Both the IDS programs receive traffic without any filters assigned. This configuration tests the load required just to receive and build up the data packets inside the IDS software and excludes the load required to analyse packets.

The script in Figure 5 applies a filtering process on HTTP traffic at the egress point on IDS. The HTTP analysis is programmed to apply an over-use page limit rule. This rule triggers if there is 10 HTTP requests of selected web pages within 11sec from the attacker machine to the victim. Bro scans the abnormal traffic and records results in the notice log. Snort generates an alarm file when it detects abnormal network traffic but Snort cannot perform complex rules as found in Bro. This work also used a complex Bro script to detect an SQL injection attack [32]. A standard Bro installation has this script in $ PREFIX\share\bro\policy\protocols\http\detect-sqli.bro. The type of attack traffic chosen was typical web page requests which are not very large. Preliminary work showed that for such traffic the load was proportional to the number of requests, the CPU load was the same for a few requests from 20 clients or the same number from one client. Attacks from clients using heavy weight payloads can be rejected based on size, similarly HTTP PUT requests can also be rejected if they are inappropriate for the page or too large.

3.3 Experimental Results

All IDS systems were tested with different attack traffic rates in four different ways to compare the CPU load;

- HTTP attack traffic and the complex rules (Bro only).
- HTTP attack traffic with HTTP analysis and rules
- HTTP attack traffic with HTTP no analysis and no rules, and
- With no load.

Each test was executed for 10 minutes, which ensured repeatable measurements with timing differences less than 0.5% between runs. Comparing Table 1 (HTTP reception with analysis and rule check) and Table 2 (HTTP reception without analysis and no rule check) for the same attack traffic rates:

Table 1: HTTP reception (analysis and rules)

Row No	Protocol	Rule	Using	DDoS attack traffic	Attack Traffic #1 (Pack/sec)	IDS CPU Load	Attack Traffic #2 (Pack/sec)	IDS CPU Load	Attack Traffic #3 (Pack/sec)	IDS CPU Load	Attack Traffic #4 (Pack/sec)	IDS CPU Load	Attack Traffic #5 (Pack/sec)	IDS CPU Load	Attack Traffic #6 (Pack/sec)	IDS CPU Load
A	HTTP	yes	BRO	yes	150	17.30%	300	21.3%	525	23.8%	3200	51.7%	4200	87.2%	6500	99.7%
B	HTTP	yes	SNORT	yes	150	2.60%	300	4.8%	525	6.8%	3200	15.3%	4200	28.7%	6500	41.3%

Table 2: HTTP reception (no analysis and no rules)

Row No	Protocol	Rule	Using	DDoS attack traffic	Attack Traffic #1 (Pack/sec)	IDS CPU Load	Attack Traffic #2 (Pack/sec)	IDS CPU Load	Attack Traffic #3 (Pack/sec)	IDS CPU Load	Attack Traffic #4 (Pack/sec)	IDS CPU Load	Attack Traffic #5 (Pack/sec)	IDS CPU Load	Attack Traffic #6 (Pack/sec)	IDS CPU Load
A	HTTP	no	BRO	yes	150	17.2%	300	19.6%	525	22.2%	3200	50.9%	4200	83.7%	6500	97.5%
B	HTTP	no	SNORT	yes	150	2.6%	300	4.7%	525	6.8%	3200	16.7%	4200	30.9%	6500	46.2%

- Note the column "Attack Traffic #2" in Table 1 and Table 2 (HTTP reception without analysis and no rule check), the CPU load of Bro increases from 19.6% to 21.3% when HTTP analysis is added. Snort has a 0.1% increase when HTTP analysis added.

- At "Attack Traffic #3", Bro shows a 1.6% increase on a base of 22.2% when HTTP analysis added. Snort shows no CPU load increase

The conclusion from this comparison is that the addition of HTTP analysis adds little to the CPU load and that most of the load is taken with no load activities plus a per received packet load. Comparing Table 1(HTTP reception and analysis) and Table 3 (no HTTP traffic) for high traffic level and zero packets/sec shows that the Bro CPU load dropped from 99.7% to 13.8% by moving from Table. 1 to Table 3.For Snort the change was 41.3% to 0.1%. Surprisingly the result shows that for the IDS software tested, the IDS CPU load increased because of the reception of traffic and not because of the processing of the traffic.

Table 4 shows the results for Bro with a complex rule for detecting SQL injection attacks. The CPU load (Table 4) is only marginally different to the other HTTP rule (Table 1) and no rules (Table 3). Again the CPU load seems related to packet reception rate and not the HTTP analysis.

3.4 IDS Load Prediction

Figure 6 & 7 graphs the packet rate against the CPU load for Snort and Bro with line of best fit. The result is a useably linear relationship between CPU load and packet rate. This means it is possible to measure the performance of a system using the low traffic configuration of one attacker and one victim as shown in Figure 2, then use regression [19, 11, 33] to produce the line of best fit and hence estimate the CPU load at higher traffic rates. This approach saves time and resources as determining CPU load at high traffic rates normally requires a powerful attack generator.

The CPU load is approximately of form given in equation number Equation (1) where a and b are constants

$$CPULoad\% = a + b * htmlpack/sec \quad (1)$$

This results in Equation (2) which can be used to predict the CPU load at a given traffic level

$$Bro\_CPU\_load = 0.0137 * packets/sec + 15.902 \quad (2)$$

(95% Confidence Interval = 10.19%)

$$Snort\_CPU\_load = 0.006 * packets/sec + 1.5743 \quad (3)$$

(95% Confidence Interval = 3.91%)

Using Equation (2) for the Bro CPU load at the last data point of 6500 HTML packets/sec predicts a CPU load of 104.9% which is close to the 99.7% actually measured. Using Equation (3) for the Snort CPU load at the last data point of 6500 HTML packets/sec predicts a CPU load of 40.6% which is close to the 41.3% actually measured.

3.5 Observation

The experimental results show that the main CPU load from Bro and Snort caused by HTTP packet reception and not the analysis of that packet.

4 Power Analysis of Two-Dimensional Web Page Daemon (TDWD)

Given the surprising experimental result that most load goes into packet reception it follows that CPU load and hence power consumption would be reduced if the IDS function was built into an existing program that already performed packet reception. This may occur at the firewall or at the web server. To test this idea we developed

Table 3: Without HTTP load

Row No	Protocol	Using	DDoS attack traffic	IDS CPU Load
A	HTTP	BRO	no	13.8%
B	HTTP	SNORT	no	0.1%

Table 4: Bro with HTTP and the SQL injection rule

Row No	Protocol	Rule	Using	DDoS attack traffic	Attack Traffic #1 (Pack/sec)	IDS CPU Load	Attack Traffic #2 (Pack/sec)	IDS CPU Load	Attack Traffic #3 (Pack/sec)	IDS CPU Load
A	HTTP	SQL	BRO	yes	150	16.00%	300	18.0%	525	22.0%

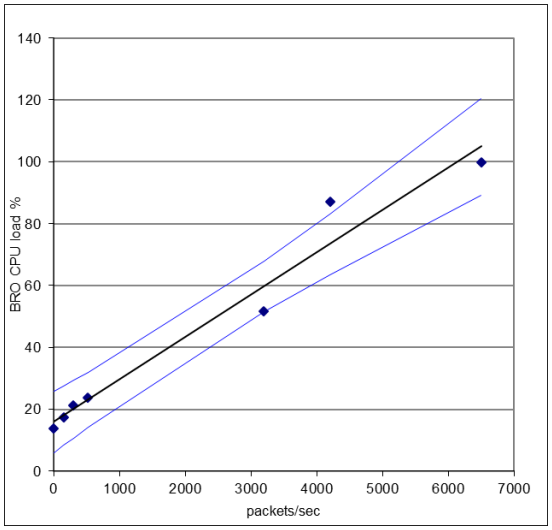

Figure 6: Bro CPU load with HTTP analysis

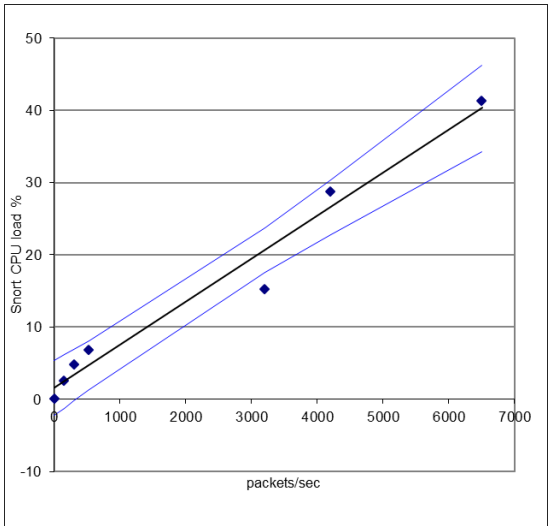

Figure 7: Snort CPU load with HTTP analysis

a novel IDS program called the Two-Dimensional Web page Daemon (TDWD) which we built into the Apache web server [27, 28]. This daemon saves web requests in a two dimensional link list, first by user IP and then by time. Analysis of this two dimensional list can determine if a user has violated a usage rule. Each web page calls a PHP script that sends the web page details to this daemon via shared memory. In these experiments the daemon then performed a rule based filter much like the Bro filter previously described that identified web page requests from the one IP at a rate of more than 10 HTTP requests in 11 seconds. The measurement and analysis of power usage was performed the new IDS daemon in the same way as done for Bro and Snort. The results are included in Table 5 in Row C. Additionally the Apache CPU load was measured at each traffic level. Row C in Table 5 and 6 shows TDWD handling all HTTP traffic and Snort and Bro given no HTTP traffic. The high traffic load of 6500 packets/second will not be analysed as the 99.7% CPU usage for Bro may mean that Bro is losing packets as suggested by the anomalous lower Apache CPU load. Using TDWD to handle HTTP saves Bro or Snort considerable CPU load and only marginally increases the CPU load on the web server as a result of running the daemon.

Comparing Table 5 (for a high traffic level of 4200 packets/sec) and Table 6(zero packets/sec) shows that the Bro CPU load dropped from approximately 87% to 14%, for Snort the change was 29% to 0.1%. This matches the scenario where all HTTP traffic is handled by TDWD. These reductions were bought at the cost of running TDWD which consumes only 3.3% of CPU load itself and approximately 10% extra CPU in Apache. The overall saving is approximately 60% CPU load for Bro and 16% for Snort. The reason for these savings is that the TDWD and Apache combination eliminates the double reception of packets and so reduces the overall CPU load on the web server system. Figure 8 & 9 graphs use data in Table 5 and Table 6 and show the CPU load of each IDS when receiving HTTP, and the resulting load on Apache.

Table 5: IDS receiving HTTP traffic with analysis and rules

Row No	Protocol	Rule	Using	DDoS attack traffic	Attack Traffic #1 (Pack/sec)	IDS CPU Load	Apache Load	Attack Traffic #2 (Pack/sec)	IDS CPU Load	Apache Load	Attack Traffic #3 (Pack/sec)	IDS CPU Load	Apache Load	Attack Traffic #4 (Pack/sec)	IDS CPU Load	Apache Load	Attack Traffic #5 (Pack/sec)	IDS CPU Load	Apache Load	Attack Traffic #6 (Pack/sec)	IDS CPU Load	Apache Load
A	HTTP	yes	BRO	yes	150	17.3%	1.7%	300	21.3%	3.6%	525	23.8%	6.3%	3200	51.7%	33.1%	4200	87.2%	39.5%	6500	99.7%	48.6%
B	HTTP	yes	SNORT	yes	150	2.6%	1.7%	300	4.8%	2.9%	525	6.8%	6.1%	3200	15.3%	30.6%	4200	28.7%	43.3%	6500	41.3%	62.9%
C	HTTP	yes	Linklist	Yes	150	0.7%	3.0%	300	1.0%	6.2%	525	1.1%	8.8%	3200	2.4%	41.7%	4200	3.3%	51.7%	6500	4.3%	67.3%

Table 6: Without HTTP load

Row No	Protocol	Using	DDoS attack traffic	IDS CPU Load
A	HTTP	BRO	no	13.8%
B	HTTP	SNORT	no	0.1%
C	HTTP	Linklist	no	0.5%

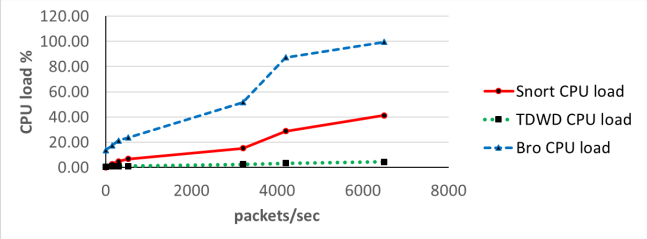

Figure 8: Bro, Snort and TDWD IDS CPU load when receiving HTTP

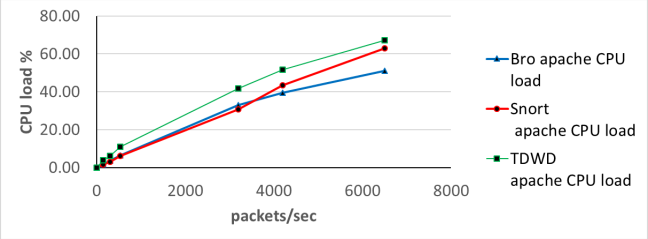

Figure 9: Bro, Snort and TDWD Apache CPU load for each IDS

4.1 Comparison of Power Usage By BRO, Snort And TDWD

The existing work described in section 2.1 showed that CPU utilisation is linearly related to power consumption. We stressed one core of an i7 CPU using the StressLinux [33] program and measured the power consumed, the results are depicted in Figure 10 with a 95% confidence interval of 0.6 watt. As per the literature there is a useably linear relationship between CPU load on one core and power consumption. Giorgio *et al.* [34] used Equation (4) to predict power consumption given CPU core utilization. Pmin is the power consumption of the entire CPU when the CPU core dedicated to the task of interest has zero utilization. Pmax is the total power consumed but the entire CPU when the task of interest is using one core at 100% utilization.

$$Power\_Consumed = (Pmax - Pmin) * Utilisation + Pmin \quad (4)$$

Given the calibration of CPU load to PC power, the power consumption of Snort, Bro, and TDWD can be graphed. From Figure 11 at high traffic rate of 6500 HTML packets/sec, Bro consumes 66 watts of the PC's power whereas Snort consumes 50 watts. The result shows that the TDWD can save approximately 25 watts compared to Bro and 10 watts compared to Snort.

This experiment has shown that an IDS that shares packet reception with another application can significantly reduce CPU use and hence power usage. The main CPU load of an IDS function is not the analysis function but a per packet load which includes the TCP stack converting between the application layer and the physical layer.

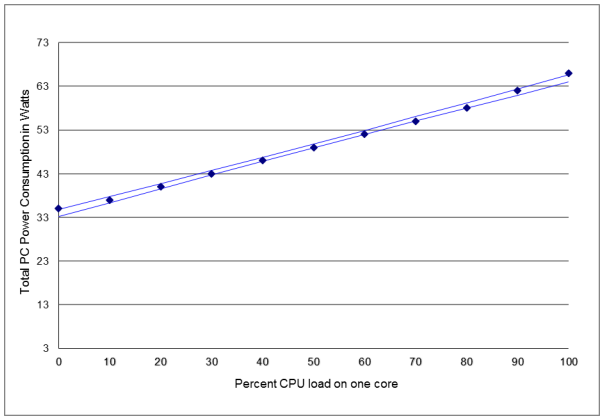

Figure 10: Power consumption vs, CPU load

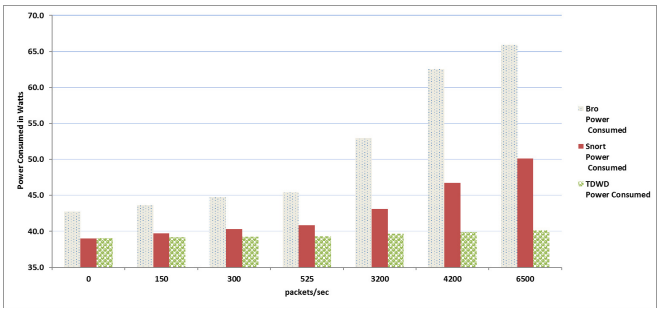

Figure 11: Power consumption of Bro, Snort and TDWD

5 Discussion of Alternative Novel Architecture

The architecture in Figure 12 shows a traditional web server with a proxy server, a firewall, a IDS and a web server where there is packet reception on each box. The same architecture is used even with a SDN implementation [40] where these functions might be applications on the one host.

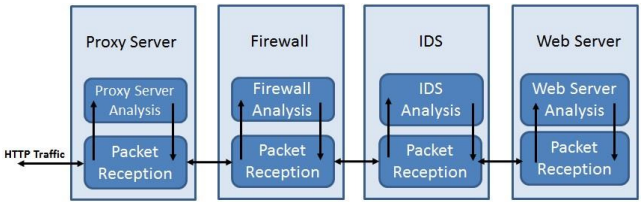

Figure 12: Traditional architecture

The research in this paper has found that the power usage in an IDS is mainly caused by packet reception rather than the packet analysis activity. This suggests that at least for SDN it would save CPU and power to have one packet reception and to pass complete IP, TCP or UDP packets between applications as shown in Figure 13.

The objection to this new architecture, for SDN or non-SDN networking, is that the security may be weaker. If any individual program is penetrated then the other programs or even the operating system may be at risk, a problem that could not occur if the functions were in separate boxes. The ability of operating systems to provide secure silos for applications is improving. One example of such an operating system is Security Enhanced- Linux (SE-Linux) [30] that provides excellent security between applications. Android has some capability in this domain as each application is treated as a separate Linux user and so the full force of the standard Linux security system is available to stop applications from interfering with each other [4]. The adoption of this new architecture is contingent on a secure operating system such as SE- Linux, being proved acceptable and having been trialled in a hostile network environment.

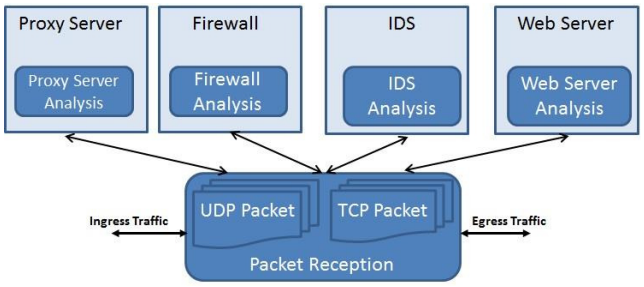

Figure 13: Novel architecture

Apache does have the internal architecture to implement the new architecture as shown in Figure 13 For example the Apache version 2 filter mod_clamav [16] scans the content delivered by the proxy module (mod_proxy) for the viruses on email using the Clamav virus scanning engine. ClamAv (Clam Antivirus) is a host based Intrusion Detection system (HIDS) written by Andreas Muller and in [25] he did mention that the processing delay has been reduced when comparing with other antivirus tools. Most likely, as this online database has shown, the removal of another packet reception process reduced CPU load. Likewise any traditional IDS like Bro or Snort could be rewritten to run as an Apache web server module and so reduced CPU load. Several modules in Apache server such as mod_status, mod_rewrite are useful for implementing server security with low CPU overhead.

6 Conclusion

A web server or web server farm may be composed of thousands of web servers and security devices. Security devices such as IDS/IPS play a crucial role in safeguarding these web servers but they do consume significant CPU time and thus electrical power. Given the large number of web server farms in the world, it is important to reduce power consumption for such farms. This paper has shown a surprising result that the power consumption of the IDS programs Snort and Bro (and most likely other security

software) in a web server depends mainly the packets received per second and depends very little on the analysis performed by these programs. This makes intuitive sense as it takes significant CPU time to operate a full TCP/IP stack. Furthermore, the CPU load is a useably linear function of packets per second and so CPU load for high traffic rates can be estimated from low traffic rate measurements.

The results also have implications for the architecture of network systems. In a new proposed architecture where programs share a common CPU (as may happen with Software Defined Networks) then CPU load (and hence electrical power) can be saved if a module receives packets to the level of IP, UDP, TCP, or high level such as HTTP, and then these formed packets are shared between higher level applications. The CPU load of packet reception is done once for several applications and not repeated for every application. Security implications need careful consideration as the infection of one program may result in the easy penetration of another program or the operating system on the same CPU. Secure operating systems such as SE-Linux hold some hope of providing a secure way to implement to new architecture proposed.

The work in this paper points to several further topics for research. The first is to test whether SE-Linux or other operating systems can provide the secure software silos needed to run the new architecture. Such a system could be trialled against known attacks and then placed in a honey pot arrangement to further stress the system. Another topic worth exploring is implementing the new proposed architecture in the module based Apache web server. What security functions make sense in such an architecture where there is not a high level of security between the web server and the security programs?

References

[1] M. Akhlaq, F. Alserhani, A. Subhan, I. U. Awan, J. Mellor, and P. Mirchandani, "High speed NIDS using dynamic cluster and comparator logic," in *IEEE 10th International Conference on Computer and Information Technology (CIT'10)*, pp. 575-581, 2010.

[2] S. M. Alqahtani, M. A. Balushi, and R. John, "An intelligent intrusion detection system for cloud computing (SIDSCC'14)," in *International Conference on Computational Science and Computational Intelligence (CSCI'14)*, pp. 135-141, 2014.

[3] F. Alserhani, M. Akhlaq, I. U. Awan, J. Mellor, A. J. Cullen, and P. Mirchandani, "Evaluating intrusion detection systems in high speed networks," *Fifth International Conference on Information Assurance and Security (IAS'09)*, pp. 454-459, 2009.

[4] Android, *System Permissions*, 27 Aug. 2015. (`http://developer.android.com/guide/topics/security/permissions.html`)

[5] T. Bhaskar and S. D. Moitra, "A hybrid model for network security systems: Integrating intrusion detection system with survivability," *International Journal of Network Security*, vol. 7, pp. 249-260, 2008.

[6] E. Biermann, E. Cloete, and L. M. Venter, "A comparison of intrusion detection systems," *Computers & Security*, vol. 20, pp. 676–683, 2001.

[7] P. Bohrer, E. N. Elnozahy, T. Keller, M. Kistler, C. Lefurgy, C. McDowell, *et al.*, "The case for power management in web servers," in *Power Aware Computing*, pp. 261-289, 2002.

[8] Bro, *The Bro Network Security Monitor*, Dec. 15, 2017. (`http://www.bro.org`)

[9] W. Bulajoul, A. James, and M. Pannu, "Network intrusion detection systems in high-speed traffic in computer networks," *IEEE 10th International Conference on e-Business Engineering*, pp. 168-175, 2013.

[10] A. Chonka, W. Zhou, J. Singh, and Y. Xiang, "Detecting and tracing DDoS attacks by intelligent decision prototype," *Sixth Annual IEEE International Conference on Pervasive Computing and Communications (PerCom'08)*, pp. 578-583, 2008.

[11] Colby, *LINEST in Excel*, Dec. 15, 2017. (`http://www.colby.edu/chemistry/PChem/notes/linest.pdf`)

[12] M. A. Eid, H. Artail, A. I. Kayssi, and A. Chehab, "LAMAIDS: A lightweight adaptive mobile agent-based intrusion detection system," *International Journal of Network Security*, vol. 6, pp. 145-157, 2008.

[13] A. Gandhi, M. Harchol-Balter, R. Das, and C. Lefurgy, "Optimal power allocation in server farms," in *Proceedings of the Eleventh International Joint Conference on Measurement and Modeling of Computer Systems (SIGMETRICS'09)*, pp. 157-168, 2009.

[14] A. A. Hadi, F. H. J. Azmat, and F. H. M. Ali, "IDS using mitigation rules approach to mitigate ICMP attacks," in *International Conference on Advanced Computer Science Applications and Technologies (ACSAT'13)*, pp. 54-59, 2013.

[15] R. A. Kemmerer and G. Vigna, "Intrusion detection: A brief history and overview," *IEEE Computer*, vol. 35, no. 4, Apr. 2002.

[16] T. Kojm, *Clam AntiVirus: User Manual*, ClamAV, 2012.

[17] F. Y. Leu, J. C. Lin, M. C. Li, C. T. Yang, "A performance-based grid intrusion detection system," in *29th Annual International Computer Software and Applications Conference (COMPSAC'05)*, pp. 525-530, 2005.

[18] F. Y. Leu, J. C. Lin, M. C. Li, C. T. Yang, "Integrating grid with intrusion detection," in *19th International Conference on Advanced Information Networking and Applications*, pp. 304-309, 2005.

[19] B. Liengme, *Regression Analysis - Confidence Interval of the Line of Best Fit*, Dec. 15, 2017. (`http://people.stfx.ca/bliengme/exceltips/regressionanalysisconfidence2.htm`)

[20] A. X. Liu and M. G. Gouda, "Complete redundancy removal for packet classifiers in TCAMs," *IEEE Transactions on Parallel and Distributed Systems*, vol. 21, pp. 424-437, 2010.
[21] P. Mehra, "A brief study and comparison of Snort and Bro open source network intrusion detection systems," *International Journal of Advanced Research in Computer and Communication Engineering*, vol. 1, pp. 383-386, 2012.
[22] D. S. A. Minaam, H. M. Abdual-Kader, and M. M. Hadhoud, "Evaluating the effects of symmetric cryptography algorithms on power consumption for different data types," *International Journal of Network Security*, vol. 11, pp. 78-87, 2010.
[23] L. Minas and B. Ellison, "The Problem of Power Consumption in Servers," *Dr. Dobb's Journal*, May 2009.
[24] T. Mukherjee, G. Varsamopoulos, S. K. S. Gupta, and S. Rungta, "Measurement-based power profiling of data center equipment," in *2007 IEEE International Conference on Cluster Computing*, pp. 476-477, 2007.
[25] A. Muller, *An Apache Virus Scanning Filter*, mod_clamav 0.23, Dec. 15, 2017. (`http://software.othello.ch/mod\_clamav/`)
[26] V. Sharma, A. Thomas, T. Abdelzaher, K. Skadron, and L. Zhijian, "Power-aware QoS management in Web servers," in *24th IEEE Real-Time Systems Symposium (RTSS'03)*, pp. 63-72, 2003.
[27] S. Sivabalan and P. J. Radcliffe, "A novel framework to detect and block DDoS attack at the application layer," in *IEEE TENCON Spring Conference*, pp. 578-582, 2013.
[28] S. Sivabalan and P. Radcliffe, "Real time calibration of DDoS blocking rules for Web Servers," *Computers*, vol. 4, pp. 42-50, 2016.
[29] N. Sklavos and P. Souras, "Economic Models and Approaches in Information Security for Computer Networks," *International Journal of Network Security*, vol. 2, pp. 14-20, 2006.
[30] S. Smalley, C. Vance, and W. Salamon, *Implementing SELinux as a Linux Security Module*, NAI Labs Report, vol. 1, pp. 139, 2001. (`http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.178.6001\&rep=rep1\&type=pdf`)
[31] Snort, *Snort Web Page*, Dec. 15, 2017. (`http://www.Snort.org`)
[32] M. Stampar, "Inferential SQL injection attacks," *International Journal of Network Security*, vol. 18, pp. 316-325, 2016.
[33] StressLinux, *Welcome to stresslinux*, Dec. 15, 2017. (`https://www.stresslinux.org/sl/`)
[34] G. L. Valentini, S. U. Khan, and P. Bouvry, "Energy-efficient resource utilization in cloud computing," *Large Scale Network-centric Computing Systems*, John Wiley & Sons, Hoboken, NJ, USA, 2013.
[35] G. Vasiliadis, M. Polychronakis, and S. Ioannidis, "MIDeA: A multi-parallel intrusion detection architecture," in *Proceedings of the 18th ACM Conference on Computer and Communications Security*, Chicago, Illinois, USA, 2011.
[36] R. E. Walpole, R. H. Myers, S. L. Myers, and K. Ye, *Probability and Statistics for Engineers and Scientists*, Macmillan, New York, 1993.
[37] P. Wheeler and E. Fulp, "A taxonomy of parallel techniques for intrusion detection," in *Proceedings of the 45th Annual Southeast Regional Conference (ACM-SE'07)*, pp. 278-282, 2007.
[38] Y. Xie, S. Z. Yu, "Monitoring the application-layer DDoS attacks for popular websites," *IEEE/ACM Transactions on Networking*, vol. 17, pp. 15-25, 2009.
[39] S. Zaman and F. Karray, "Lightweight IDS based on features selection and IDS classification scheme," *International Conference on Computational Science and Engineering*, pp. 365-370, 2009.
[40] N. Zilberman, P. M. Watts, C. Rotsos, and A. W. Moore, "Reconfigurable network systems and software-defined networking," *Proceedings of the IEEE*, vol. 103, pp. 1102-1124, 2015.

Biography

Ms.Sujatha Sivabalan biography. received a B.Eng from Bharathidasan University(India)in 2004 and M.Eng from Anna University (India) in 2007.Currently doing Ph.D. in School of Electrical and Computer Engineering, RMIT University, Australia. Her research interest includes network security, DDoS attack detection and blocking.

P. J. Radcliffe biography. received a B.Eng from Melbourne University (Australia) in 1978 and worked for Ericsson Australia R&D for 7 years followed by consulting to various companies. He joined Royal Melbourne Institute of Technology (RMIT) and was awarded and M.Eng. in 1993 and a PhD in 2007. Main research interests include network protocols, Linux, and embedded systems. He received a national teaching award in 2011 and in 2012 received the RMIT early career researcher award.