(WHASG) Automatic SNORT Signatures Generation by using Honeypot

Hesham Altwaijry and Khalid Shahbar
Department of Computer Engineering, College of Computer and Information Science, King Saud University
Email: twaijry@ksu.edu.sa, Shahbar@hotmail.com

***Abstract*—An Intrusion detection system (IDS) is an important network security component that is used to monitor network traffic and detect attack attempts. A signature based intrusion detection system relies on a set of predefined signatures to detect an attack. Due to "zero-day" attacks (i.e. new unknown attacks) conventional IDS will not be able to detect these new attacks until the signatures are updated. Writing efficient new signatures to update the IDS signature database requires that the attack is first detected then studied and analyzed. These new rules should be general enough to include any modification of the attack pattern and specific so that normal traffic remains unblocked. Writing these signatures manually requires significant effort, time and knowledge to work properly. In this paper, a web based honeypot is used to generate SNORT intrusion detection system signatures (Rules) for HTTP traffic automatically. These new rules are integrated into the IDS signatures data base. We then verify the efficiency of the modified rules and show that the new rules are able to detect and block these attacks.**

***Index Terms*–Automatic signatures generation, SNORT Rules, intrusion detection system, IDS, signature based.**

I. Introduction

Intrusion detection systems gather information, analyze then detect if there is an attack. There are two techniques to analyze the information gathered. Anomaly based IDS build a baseline of normal network behavior, then look for abnormal behavior in the network activity (i.e. a deviation from normal network behavior using statistical methods) to detect an attack. The other technique used is signature based or misuse based which stores known attacks types then compares the network traffic with these types (signatures) to detect an attack. A key requirement here is that the signatures must be predefined so that the IDS can identify the attack attempts.

Because of the frequent appearance of new threats most signature based intrusion detections system provide the option for the user to add new signature to the signature database. Writing a new signature requires a good level of knowledge in programming, network infrastructures, operating systems, security policies, and network protocols. These new signatures could depend on the IP source, IP destination, the protocol field, or the payload itself. The main drawback of the signature based IDS is an attack is not detected if it is not in the signatures database. Additionally small modifications in the attack method for the predefined signature might cause the attack to be undetected by the IDS.

Automatically generating signatures requires multiple steps. These steps start with traffic capture followed by traffic classification and application identification. Finally, the suspicious packet is inspected. Ifa threat is detected then the proper signature should be written. This signature must be tested to ensure that it does not affect normal traffic or overlap with existing signatures. After all these steps have been undertake then and only then is the created signature added to the IDS signatures database.

There are numerous traffic capturing applications, for example TCP dump, and Wireshar. However, the output of these traffic capturing applications needs to be formatted into a form that is compatible with the next stage in the automatic signature generation process.

Traffic classification and application identification requires deep inspection of the packet's payload. There has been a significant research effort in the automatic classification of network traffic. The port number cannot be used in traffic classification; this is because applications that use peer to peer protocols assign port numbers dynamically. By using packet payload inspection application protocol session reconstruction is feasible thereby achieving signature matching [1], [2] and [3].

Teodoro, Feldstedt, and Zúñiga [4], proposed a system that depends on finding abnormal activities in a specific network service, which are analyzed to generate specific signatures for these specific network services. Massicotte, Labiche, and Briand [5] used a data mining algorithm to automatically generate IDS verification rules to reduce the number of false alarms. The alarms generated by SNORT are analyzed to distinguish between valuable alarms and false ones. The result from the analysis is used to modify SNORT rules.

Hwang, Cai, Chen, and Qin [6] used a weighted signatures generation algorithm to extract signatures from anomalous behavior in the network. Then the signatures created are added to the IDS signature's database.

Honeycomb [7] uses honeyed, (an open-source honeypot,) to inspect traffic that is used to generate signatures. For each connection state, headers are

Manuscript received October 20, 2012; revised January 15, 2013; accepted March 11, 2013.

doi:10.4304/jcp.8.12.3280-3286

compared to detect matching IP networks, initial TCP sequence numbers among other traffic analysis techniques. When the same destination port is present in the connection the pattern is detected and a signature is added to the signatures pool.

Autosig [8] creates signatures based on the classification of the applications. It divides the traffic into common fixed-size short blocks and classifies them into groups according to the position of the block in the traffic. LASER algorithm [9] was introduced to create application level signatures. It focuses on general application classification which is not specific to intrusion detection systems or threat traffic. The algorithm searches for the signature of an application where the length of the sequence of substrings in the signature reflects how accurate the identification of the signature will be.

Worms have their ways in spreading to infect the biggest possible number of users. Inspecting how worms behave to avoid detection by security systems has inspired many researchers [10], [11], [12] and [13]to find methods that could detect worms automatically, thereby creating signatures for worms automatically and identifying worms among network traffic.

Anita et al [14], define the signature to be the sequence of system calls created by each application. A database for each application is created. This database stores the applications sequence of system calls. Then traffic is compared with known application system calls. Any abnormality will be considered as an attack.

II. Overview of Web Based Honeypot for Automatic Signatures Generation (WHASG)

To generate signatures automatically, attack patterns should be first detected. Researchers have used many different methods to extract suspicious traffic from normal network traffic. However, doing so will increase the probability of classifying normal traffic as suspicious traffic. Therefore, in our work, we have relied on honeypot to identify suspicious traffic. Honeypots have different level of interaction with users; they are used to simulate different network components [15] and [16] such as Email servers, database services, web applications, and web servers. The difference in the level of interaction and types of honeypots defines what information can be extracted from honeypot logs. This information is critical in determining the accuracy in classifying received traffic as normal traffic or suspicious traffic, in identifying known attacks types and new unknown attacks, and the amount of information that could be gathered about the attacks and the attacker. GlastopfNG [17] is a web based honeypot that is used to collect attacks. It simulates server response to attackers. The server used in this honeypots is a mini HTTPD server modified to fit the purpose of the honeypot. When an attacker sends a request it will get the proper response depending on the request sent. The reply send by the honeypot depends on the analysis of the request. This is done to convince the attacker that the exploit succeeded and the vulnerability actually exists in the system under attack. We have used it as our attack source to collect attack information which is the used by our signatures generator to extract the information needed to build a proper SNORT rule [18]. Python is used to collect the honeypot output, analyze it, generate signatures, and build SNORT rules. We will concentrate on using Web-based Honeypots for Automatic Signatures Generation (WHASG). The honeypot log file where all new attacks are logged will be used and analyzed to extract useful information that identifies an attack. The output of WHASG is a set of new rules that will be added to the SNORT rules database and could be used directly to detect attacks in real traffic, in real time. Whatever rules are created they will not be repeated in the database if the same attack arrives even from different attackers. This is due to existence of the Rule Module in WHASG where a repetition function is called whenever a new rule is generated to ensure that rules file will include non duplicated rules.

A. WHASG Features

Real Time Automatically generated signatures: The signatures of attacks that already exist in SNORT's signatures database were written by security experts or by users. These rules required time to look at the log file or to search for security vulnerabilities. This is typically what is done when a new signature is added to the IDS. However, attacks arriving to WHASG will be added directly to the rules database and therefore the database will be updated immediately. Adding new attacks depends on the honeypot and its ability to attract new attacks. Writing a rule for a known attack will produce a more efficient and accurate rule than a rule written for a general attack. These new rules will help to prevent the spread or action that the attack may achieve if there is not any defense at all.

Flexible infrastructures Integration: The main modules and rules generator do not depend on a specific honeypot. Whenever the information about the web server request is available then the analysis starts and a new rule will be generated.

Add, Remove, and Update Rules: With the use of an enhanced structure, it is natural to notice that the number of signatures generated increases during the arrival of new attacks that are not in the signatures database. Decreasing the number of signatures is also normal because of the enhancer module which when triggered will evaluate all existing signatures and remove any signature that is included in a general signature, or to update the signature itself when more information is available to the enhancer that was not known before.

False Positive Controlled Rate: WHASG manual configuration can decrease the amount of false positives by a noticeable amount. This is due to the information offered in the configuration file that could be used to identify what should be an alarm and what is normal traffic. For example, Web Distributed Authoring and Versioning (Web Dav) is an additional method that could be used by web servers to enable web applications to be more interactive with users by providing more options for application developers. Therefore, if Web Dav is enabled by the user and it is added to the configuration the new method is considered a normal request and will not trigger an alarm. Adding more information about what the

environment looks like will help to define what the expected traffic that will arrive to the web server is.

B. WHASG Construction

The main purpose of writing WHASG is to generate signatures automatically. Therefore WHASG consists of many modules; each module is responsible for a specific task such as reading attack logs, parsing attack requests, and generating sequential signature identification numbers.

The control module is the main module in WHASG, it communicates with all other modules to coordinate the generation of correct signatures.

WHASG works in two modes: Auto mode and Manual mode. In auto mode rules generated are added directly to the rules database. In manual mode adding any rules require user permission. The default configuration of WHASG when working in auto mode is not to trigger the rule generation module for any attacks received by the honeypot. HEAD or normal GET request will be only shown in the screen and will be logged to the WHASG log file but it will not added to rules database.

Known search engines requests will be ignored and not considered as attacks when working in auto mode. Search engines uses web crawlers or web spidering to add new sites and check for indexes and other sub domains. They send requests to web servers that include asking about robot.txt file. When a honeypot received such a requests, it could consider them as an attack because of the attempt to access robot.txt file. In WHASG, search engines play important role in attracting more attacks. For this reason the control module will not include search engine requests in the analysis unless WHASG is working in Manual mode and configured to include search engines in attack analysis.

The Manual Mode in WHASG will set user inputs and use them in configuration. For example whenever a new attack arrives at the network, it will trigger the rule module and generate a rule that is added directly to the rules database. However if WHASG is configured to work in manual mode whenever a new attack arrives it will give the option to the user to approve this new rule or cancel it or even edit this new rule. This adds more flexibility to the configuration of WHASG.

Manual mode also sets some values that configure how to deal with different HTTP requests. Search engine dorks and Trusted sited can be added to WHASG list so they will be considered as safe sites and any request from them will not be logged. It will only be shown on the screen. This option should be handled carefully, because excluding sites from inspecting process is not safe unless the sites are ensured to be safe. But this option helps in testing and in using local host address and in not to increasing false positive alarms.

The rules module is where the signatures generated. It receives all the information required from the main module then it will generate the signature. To avoid duplicating signatures or writing two different signatures with the same SID, the rules module contains functions that will handle these issues.

All modules communicate with configuration and variables files to achieve what is required. Dividing WHASG into modules that exchanges information between them enables future expansion or any required development. This includes adding new honeyposts to WHASG system or adding more rule generation options. Fig. 1 shows WHASG Modules.

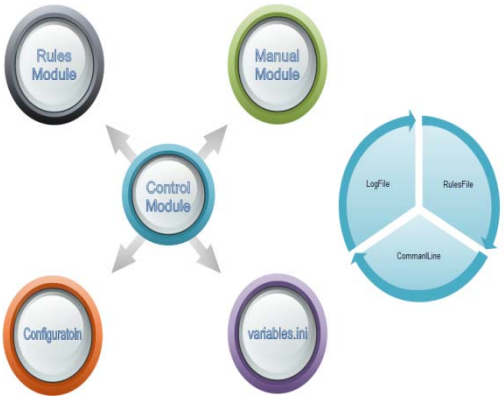

Figure 1. WHASG modules

III. WHASG Signature Generation Mechanism

A. Information Extraction

Honeypot log files contain attacks that were received by interacting with attackers. Most of these attacks are new attacks. To extract useful information from the honeypot log file and format the information to be ready for signature syntax, we include the first few steps in WHASG that are needed to prepare the log file and then extract the proper signatures:

- Start by reading the whole log file storing all attack information.
- The signature generation will vary depending on the information extracted from each attack.
- The proper information that forms the signature is stored in variable and send to the signature generator to form the new signature.
- Newly created signatures are checked by using string matching to search in SNORT signature database. Finding if the generated signature preexists in the database is non-trivial due to the multiple methods a signature could be written.

To avoid missing any entry in the log file where the request does not belong to a known vulnerability and it could not classified according to a previously known vulnerability or a deviation from normal HTTP protocol, the requesting string itself is extracted from the attack information and added as content to be used to generate new signature, this way WHASG will not miss any attack that is already in the log file.

B. Signature Container Array

To have efficient signature generation an enhancer to the signature generated is required. We constructed the enhancer by using what we called the Signature Container Array. Its function is to analyze given strings to extract minimum common substrings that when used in signatures will identify the original strings with the minimum possible signatures.

To achieve this task, an array is created where the size of the array varies depending on the strings size. The variable array size helps in working with different string sets where each group represents a set of different attack similarities. Empty elements are used as placeholders in the array where no strings are used. After the array is created the first container is built by using the first column in the array as a reference substring. The length of each element in the container is varies depending on the original strings. The first element in the container is compared with all elements in the array. A search will be done using string matching where the existence of a substring in any element in the array will be considered as a match. The search of each substring in the container will continue until all substrings are compared with all elements in the array. A threshold value is set to store the iteration of the substring in the array. After all substrings are compared, the result will form a first level container. Each element in the first level container represents possible strings that could be part of the signature. The array is called again and the substring is used with the first element in the first level container to form a new substring couple that will be used to compare with the array elements.

The chosen substring must satisfy two conditions:

- It is not part of the first level container. This is to avoid duplication in signatures strings.
- The substring and the container element must both be on the same request to form one substring couple match.

When a match is formed, the comparison starts again with each row in the array. The number of appearance of a substring matches is calculated and compared to other array elements. When all match comparisons are completed, a second level container is built. The final step is to remove duplicated values if any. Signatures are then ready to be generated using strings from the second level container. Figure 2 shows flow chart of Signature Container Array process.

IV. Results

A. Data Captured and Traffic Database

To test signatures generated by WHASG we used real traffic that is captured by SNORT. The Traffic used to test WHASG signatures includes different data types. Most attacks that arrived at the honeypot could not be detected by SNORT. These attacks were mixed with normal traffic that does not contain any suspicious traffic, i.e. normal requests for a web server and HTTP traffic.

SNORT gives the ability to compare packets logged by honeypot with packets logged by SNORT by using the time stamp that identifies each received packet. Data that we captured contained TCP, UDP, and ICMP traffic with most of the data being TCP; this is due to the web server running in the honeypot which simulates web server response. The important data we looked for is HTTP traffic where all web application and web server attacks rely on HTTP protocol.

By pointing the IP address of a registered domain to the local address of the web server, the address of this web server is indexed by search engines. This way it will attract more traffic to the server and allow us to capture different traffic which will help in testing signatures generated by WHASG.

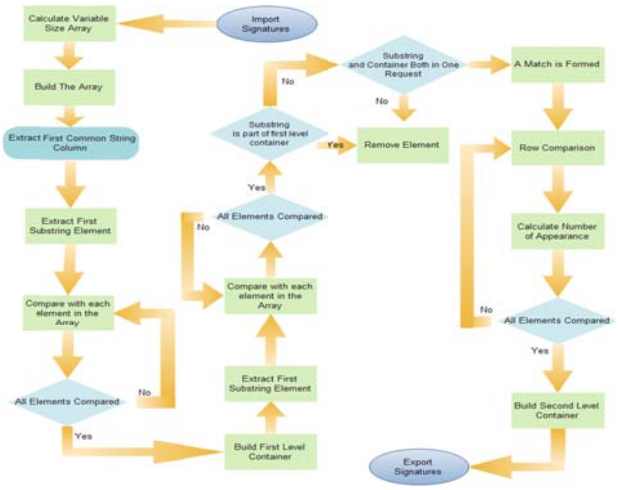

Figure 2. Signature container array

In order to have better performance evaluation of our generated signatures the data recorded is divided into two groups. The first group has data containing attacks that the honeypot has already logged. The second data set is data that has no attacks logged or detected by honeypot. This method allows us to evaluate WHASG within two different environments which occur within any real system.

Our data was collected over a period of 9 weeks. In this period both clean traffic and attacks are collected and stored for testing and analysis. Classification of data collected is done at the end of the 9th week. Time stamps in both the honeypot and SNORT log are used TCP dump is used to identify packets that included attacks detected by honeypot. This is useful in determining false positive and false negative alarm rate. Set of clean data where no attacks were detected is useful in testing for false positive rate. Other sets that include attacks will be used to test false negative alarm rate.

On average, each week 12000 packets are logged with their payloads for further inspection. Data collected were real traffic and this lets the type of traffic collected weekly vary in their content. Synthetic traffic was also used. It was mixed with captured data to provide variations of attacks within normal traffic. These variations of attacks were not found in the captured data. Attacks arrives at the honeypot are new attacks which will not include known vulnerabilities, this could affect the result of SNORT detection and increase false negative rate.

Real environment attacks include both known and new attacks; synthetic attacks are used to simulate a real environment when adding known vulnerabilities to the data set. Actual known attacks are not important as new attacks; this is because signature based IDS already have signatures that can detect known attack types, while new attacks would not be detected unless the signature

database is updated. The important part in our test is to determine that the signatures generated are able to trigger an alarm when any type of attack that is included in signature database appears in the test set. To have results that reflect real environment then both known attacks and new attacks are included and tested.

B. Generated Signatures Results

Evaluation of WHASG signatures is done by comparing results of SNORT detection before and after adding WHASG signatures to the signatures database. Logged traffic was used to test detection of WHASG. The data set used contains on average 25 packets captured by TCP dump where new attacks construct most of attacks data in the captured traffic.

Data used shows different amounts of attacks. Alerts raised by SNORT could indicate the existence of attacks in the data used in testing. To better evaluate the generated signatures, data used should include both normal traffic and attack traffic. The alarm rate for SNORT ensures that attacks are correctly logged by honeypot, correctly imported from log file, and correctly created if attacks source is not honeypot. Although alarm rate does not explain detection accuracy it still shows how much variation in the data source could change raised alarms.

The way WHASG was installed where search engines are used to include domain name that points to a honeypot address will increase normal traffic by a noticeable amount. This is due to the honeypot acting as a web server and search engines indexing in addition to normal request received by the web server. This will lower alarm rate but in fact, the number of attacks is still high (in thousands).

For example, the alarm rate of SNORT from logged data is 0.3 % of total traces. This rate increased to 3.09 % when adding WHASG signatures to SNORT database. These rates represent alarm rate from total traces. Different sets that contain normal traffic mixed with synthetic traffic could trigger alarms with 0.086 % when using SNORT signatures, adding WHASG signatures increased the alarm rate to 0.334%. The difference between this set and previous one in alarm rate percentage is not large because attacks that included in the test set changed from new vulnerabilities where WHASG depends on honeypot to detect them to normal traffic with traces of attacks. When captured attacks logged with SNORT are used that include the new attacks we got 3.71 % alarm rate when WHASG signatures are included to SNORT database where it was 0.29 % when only SNORT is used to detect attacks. The alarm rate shows slight change from 1.62% when SNORT is used to 1.75% when using WHASG signatures when synthetic data set is used. Alarm rate gives indication about amount of alarm triggered by SNORT and WHASG when different set of data is used. When data collected in short period of time with synthetic collection of attacks then alarm rate is highly increased. For example, when data collected in one day where most of data is suspicious traffic then alarm rate will show noticeable difference between SNORT and WHASG. Figure 3 shows 34% alarm rate for SNORT and 45% alarm rate when adding WHASG signatures. We got these rates because: data is only logged for one day, type of traffic used contain more attacks than normal traffic, and attacks used are mixed of known type so SNORT can detect them and altered attacks where they could not be detected by SNORT.

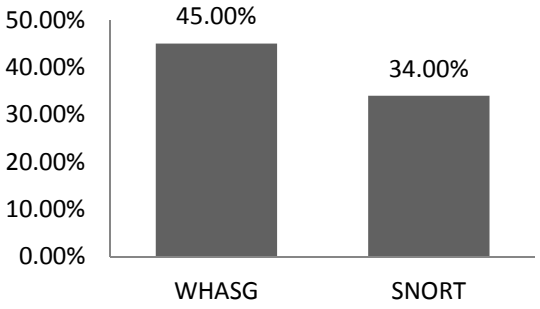

Figure 3. Alarm rate for one day data

When using data collected from all sets, alarm rate will show lower rate. This is because data is collected in long period, it contains normal requests, attacks are not controlled which means attackers choose type of request they send. Figure 4 shows the average alarm rate for the whole data including normal traffic.

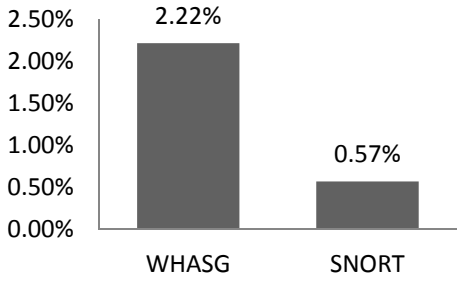

Figure 4. Average alarm rate

We can tell from figure 4 above that it is obvious data collected contains more ordinary web traffic, because the average alarm rate is measured for long period then the attacks spread over all data, also we can tell that WHASG triggers more alarms than SNORT which is good as an indication of detection of new attacks. Both of them have similar alarm rates because of variations in the data used for testing. When WHASG adds new signatures for unknown attacks then it can detect attacks that SNORT will not detect them, from the average alarm rate we can know what type of data both systems are dealing with. But average alarm rate will not show accuracy of these alarms or missed attacks that supposed to be alarmed.

In order to evaluate the efficiency of signatures written for IDS, alarm rate could not be used to show detection ability because it will show only different response to data used in testing but if the data used contains new type of attacks that are not detectable by SNORT or if the added signatures increase number of alarm then alarm rate will not show these important factors. To do so, two main criteria are used to determine improvement of IDS before and after adding new signatures to the signatures database. False Positive Alarm Rate and False Negative Alarm Rate. False Positive Alarm Rate measures the percentage of

alarm triggered by IDS where it supposed not to raise an alarm. False Negative Alarm Rate measures when the IDS miss an attack that is supposed to be detected.

C. False Negative Alarm

It is expected to have a high detection rate when using rules generated by WHASG. This is because attacks collected by the honeypot will be analyzed by WHASG and signatures will be generated accordingly When using these data to test WHASG it is logical to have a high detection rate and low false negative alarm rate. Otherwise the generated signatures are not effective and can't detect attacks that originally used to generate signatures.

We will compare the result when using signatures created by WHASG and when using SNORT without adding WHASG's signatures. Attacks collected are used to test false negative alarm rate, WHASG achieves a very good results as expected. The false negative rate of SNORT is 64 % while when adding WHASG signatures we get only 1%. Figure 5 shows false negative alarm rate for both SNORT and WHASG.

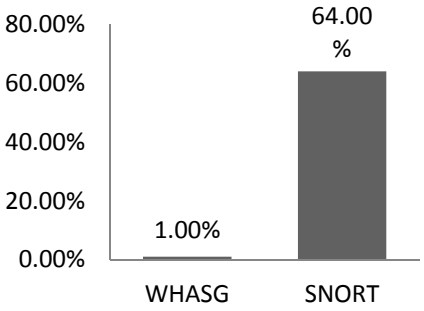

Figure 5. SNORT and WHASG false negative alarm rate

When we added new generated signatures to the SNORT rules database we increase the detection ability of SNORT. These new generated signatures are did not exist in SNORT data base because WHASG only generates new signatures when SNORT could not detect attacks. So it is expected and required to have better detection ability when adding signatures using WHASG.

We used attacks logged by the honeypot to generate new signatures then adding them to the rules database, parts of our test set are extracted from these attacks that are collected by the honeypot. Because not all of these logged attacks could be detected by SNORT signatures, result with WHASG should show better result than SNORT.

Our goal here is to measure how WHASG doing toward data that should trigger alarm not to test SNORT itself. The best to use for this purpose is attacks originally caused WHASG to generate new signatures. Real environment contains normal traffic more than attacks. Also new attacks and known type of attacks both could be seen by any system. Using normal traffic in testing is also important to ensure that new generated signatures will not trigger unnecessary alarms.

Detection rate is measured by showing percentage of true alarm from all attacks. Obviously it is another measurement for false negative alarm rate.

D. False Positive Alarm

There is a tradeoff between the accuracy we could achieve which measured in false negative rate and the false positive alarm rate. When more precise signatures are written then they will increase the false alarm rate, SNORT shows 3.17% false positive alarm rate, when WHASG signatures are added to SNORT this rate is increased to 5.54% which is still a good result. Figure 6 shows false positive alarm rate for both SNORT and WHASG:

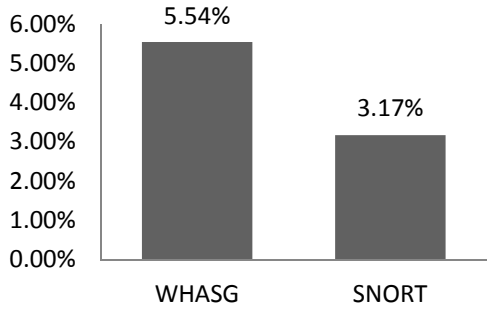

Figure 6. SNORT and WHASG false positive alarm rate

Number of checked cases will affect false positive alarm rate, when adding more signatures to signatures database as we do in WHASG then number of inspection for packets will increase which definitely will increase false positive alarm specially when normal traffic weight much in data set. As shown previously in figure 4, our data contains normal sufficient amount of normal traffic specially when measured for the whole testing period. When some type of normal data will trigger false positive alarm then when using many of this type in the data then the alarm will be shown repeatedly and will increase the false alarm rate. When working automatically to generate signatures then generated signatures should be general enough to cover wide area of attack similarities.

In fact, when generating signatures automatically these signatures should balance detection ability between general cases and specific cases. There is a tradeoff between false positive alarm rate and false negative alarm rate, we could focus on lowering false positive alarm but this will increase false negative alarm. Obviously, false negative alarm rate is more important than false positive alarm. Checking more false alarms will cost security administrator more time and effort but losing one attack might shut down the system completely or cause serious damage.

E. Overall Results

Our python code (WHASG) which is used to integrate honeypot detection ability with SNORT signatures worked on automatically generate signatures for new type of attacks discovered by honeypot. Signatures generated are efficient enough not to miss attacks that already exist in the signature database and not to increase the false positives to a non acceptable rate.

None of the signatures generated by WHASG have any syntax error that will affect SNORT detection whatever the types of attacks detected by honeypot are. WHASG

always generate signatures with valid syntax. Options available in WHASG configuration help in generating better signatures. Inputs from user will help in enhancing generated signatures when working on manual mode. This will lower false positive rate and increase false negative rate. However, results we got about WHASG are measured when working in Auto mode not Manual mode.

Comparing both SNORT and WHASG shows noticeable improvement to SNORT when adding signatures generated by WHASG. Detection rate is enhanced when using WHASG signatures. False Negative alarm reduced from 64% to only 1%. False Positive alarm increased from 3.17% to 5.54%.

V. Conclusion

WHASG successfully generates signatures automatically by using web based honeypot; signatures generated are effective, free of syntax errors, totally generated by WHASG without any manual need when working in Auto mode. In spite WHASG is written to extract signatures automatically not to increase SNORT detection rate or improve false alarms rate but results we got show improvement in SNORT capabilities in detection, decrease in false negative alarm rate with slight increase in false positive alarm rate.

Using honeypot as a source of attack information that is used to extract and generate signatures automatically added another value to WHASG rather than generating signatures automatically where when zero-day attacks discovered by honeypot WHASG will generate signature that detect this new attack.

References

[1] J. Newsome, B. Karp, and D. Song, "Polygraph: Automatically generating signatures for polymorphic worms," in *Proc. IEEE Symposium on Security and Privacy*, 2005.

[2] Y. Wang, Y. Xiang, and S.-Z. Yu, "Automatic application signature construction from unknown traffic," in *Proc. IEEE International Conference on Advanced Information Networking and Applications*, 2010

[3] D. Brumley, J. Newsome, D. Song, H. Wang, and S. Jha,"Theory and techniques for automated generation of vulnerability-based signatures," *IEEE Transaction on Dependable and Secure Computing*, vol. 5, no. 4, October-December 2008.

[4] P. G. Teodoro, P. M. Feldstedt, and D. R. Zúñiga, "Automatic signature generation for network services through selective extraction of anomalous contents," in *Proc. Sixth Advanced International Conference on Telecommunications*, 2010.

[5] F. Massicotte, Y. Labiche, and L. C. Briand, "Toward automatic generation of intrusion detection verification rules," in *Proc. Annual Computer Security Applications Conference*, 2008.

[6] K. Hwang, M. Cai, Y. Chen, and M. Qin, "Hybrid intrusion detection with weighted signature generation over anomalous internet episodes," *IEEE Transaction on Dependable and Secure Computing*, vol.4, no. 1, January-March, 2007.

[7] C. Kreibich, J. Crowcroft and Honeycomb, "Creating intrusion detection signatures using honeypots," *ACM SIGCOMM Computer Communication*, vol. 34 issue 1, January 2004.

[8] M. Ye, K. Xu, J. Wuand, and H. Po, "AutoSig-automatically generating signatures for applications," in *Proc. IEEE Ninth International Conference on Computer and Information Technology*, 2009.

[9] B. C. Park, Y. J. Won, M. S. Kim, and J. W. Hong, "Towards automated application signature generation for traffic identification," in *Proc. IEEE Network Operations and Management Symposium*, 2008.

[10] C. Kruegel, E. Kirda, D. Mutz, W. Robertson, and G. Vigna, "Polymophic worm detection using structural information of executables," in *Proc. Int'l Symp. Recent Advances in Intrusion Detection*, 2005.

[11] T. Detristan, T. Ulenspiegel, Y. Malcom, and M. V. Underduk. "Polymorphic shellcode engine using spectrum analysis." [Online]. Available: http://www.phrack.org/issues.html?issue=61&id=9#article, 2003.

[12] H. Rajabi, M. N. Marsono, and A. Monemi,"A framework for automated malcode signatures generation," in *Proc. IEEE Student Conference on Research and Development*, 13 - 14 December 2010, Putrajaya, Malaysia.

[13] H. Ah Kim and B. Karp, "Autograph: Toward automated, distributed worm signature detection," in *Proc. 13th Conference on USENIX Security Symposium*, vol. 13, 2004.

[14] A. Jones, and S. Li, "Temporal signatures for intrusion detection," in *Proc. Computer Security Applications Conference*, 2001.

[15] L. Spitzner, "Honeypots: Tracking hackers," 2002.

[16] N. Provos and T. Holz,"Virtual honeypots: From botnet tracking to intrusion detection," *Addison Wesley Professional*, July 16, 2007.

[17] GlastopfNG project. [Online]. Available: http://dev.glastopf.org/projects/glastopfng/wiki

[18] SNORT R Users Manual 2.9.1, "The Snort Project," September 20, 2011.

Hesham A. Altwaijry obtained his master's and Ph.D. degrees in the electrical engineering from Stanford University, California, USA, in 1993 and 1997 respectively. His interests are computer arithmetic, computer architecture and VLSI design. He is currently the department chairman in the computer engineering department in the college of Computer and Information Sciences, King Saud University, Riyadh, Saudi Arabia.

Khalid Shahbar obtained his master degree from the computer engineering department in King Saud University, in 2011. His interests are computer networks and network security. He is currently working as a network engineer in Saudi Arabia. He intends to peruse his Ph.D. in the field of network security