Agent-based honeynet framework for protecting servers in campus networks

Rishikesh Misal[1], Deepak Patil[2], Rodney Rodrigues[3]
Computer Engineering Department,
Vidyalankar Institute of Technology,
Wadala , Mumbai

Abstract— **Protecting Servers from various internet worm codes is of utmost priority. A script that run without the intervention of a host and can exploit any vulnerability and can harm delicate is one of the biggest threat an organization can face. We provide an Agent based honeynet architecture to protect the servers inside a campus network. A two level security check is conducted before the packet is allowed to enter the production servers inside the network. An Intrusion Detection System (IDS) can detect an intrusion based on either the packet type or if the contents match to one of the signatures of known worms. Since worm codes can always be obfuscated we make a second level check on the pocket that is conducted by our honeypot agent. Based on the performance monitor parameters the classifier algorithm will term the incoming packet is a worm code or not. Once the packet is termed as a malicious packet, signature generation module is used to generate the signature of the code using hex code eliminating all irrelevant data. Once the signature is generated it can be used at the agent side to check against the incoming packet's signature this will reduce the time taken to term a packet is malicious or not.**

I. INTRODUCTION

Protecting servers from worm attacks is the prime objective of our project. Current practice for worm is retroactive and manual i.e. only after a new worm is first detected and analyzed by human who identifies a signature can be containment process be initiated. Honeypot agent allows the process to become more automated as it can incorporate our classifier algorithm with it. Accessing confidential data has always a slow response time since it goes through many security checks. Attacks can be of different kinds ranging from intrusion to exploitation of vulnerabilities to destroying servers. To detect attacks on the server based on disallowed transport layer protocols is considered as an intrusion. To detect such intrusion an IDS (Snort) is used which will trigger a message stating an intrusion is detected based on the packet type. If the packet type is legitimate but the contents of the packets are malicious these will be detected using the signature which Snort possesses. Now, even if the contents of the packet are malicious but it doesn't match with any of the signatures in Snort and the packet type is valid, such packets are handled by the honeypot agent which has a classifier algorithm running constantly looking for any performance changes whenever a packet arrives. Worm codes start running once it is allowed to reside inside the honeypot which will trigger the performance monitor to log the changes occurring. These logs are used by the classifier to term whether the incoming packet is malicious or not. Since this process takes more time, we use another module called packet called Signature Generator which creates signature of detected worm codes. Once the signatures are generated these signatures will be checked against the incoming packets which will reduce the time taken for a packet to be termed as malicious.

II. LITERATURE SURVEY

Honeypots:

A honeypot is a security resource whose value lies in being probed, attacked or compromised. Honeypots are a highly flexible tool that can be applied to a variety of different situations. For example, honeypots can be used to deter attacks, similar to the functionality of an Intrusion Detection System. Honeypots also can be used to capture and analyze automated attacks, such as worms, or act as early indication and warnings sensors. Honeypots also have the capability to

research the activities of the blackhat community, capturing the keystrokes or conversations of attackers. How you use honeypots is up to you.

Worms:

A computer worm is a standalone malware computer program that replicates itself in order to spread to other computers. Often, it uses a computer network to spread itself, relying on security failures on the target computer to access it. Unlike computer virus, it does not need to attach itself to an existing program. Worms almost always cause at least some harm to the network, even if only by consuming bandwidth, whereas viruses almost always corrupt or modify files on a targeted computer.

- Most common worms are spread in one of the following ways:
  - Files sent as email attachments
  - Via a link to a web or FTP resource
  - Via a link sent in ICQ or IRC message
  - Via P2P (peer-to-peer) file sharing networks
  - Some worms are spread as network packets. These directly penetrate the computer memory, and the worm code is then activated.

Intrusion Detection System:

Intrusion Detection can be defined as the "the act of detecting actions that attempt to compromise the confidentiality, integrity or availability of a resource." More specifically, the goal of intrusion detection is to identify entities attempting to subvert in-place intrusion Detection.

Common types of intrusion Detection Systems:

- Network Based (Network IDS)
  Network based intrusion detection attempts to identify unauthorized, illicit, and anomalous behavior based solely on network traffic. Using the captured data, the IDS system processes and flags any suspicious traffic. Unlike an intrusion prevention system, an intrusion detection system does not actively block network traffic. The role of a network IDS is passive, only gathering, identifying, logging and alerting. Examples of network IDS: Snort
- Host Based (HIDS)
  Often referred to as HIDS, host based intrusion detection attempts to identify unauthorized, illicit, and anomalous behavior on a specific device. HIDS generally involves an agent installed on each system, monitoring and alerting on local OS and application activity. The role of a Host IDS is passive, only gathering, identifying, logging, and alerting. Examples of HIDS: OSSEC- Open source Host-based IDS, Tripwire.

Classifier:

The main contribution of our approach is that the knowledge is acquired automatically using inductive learning, given a dataset of known worms (avoids the need for manual acquisition of knowledge). While the new approach does not prevent infection, it enables a fast detection of an infection which may result in an alert, which can be further reasoned by the system administrator. Further reasoning based on the network- topology can be performed by a network and system administration function, and relevant decisions and policies, such as disconnecting a single computer or a cluster, can be applied.

Generally speaking, malcode within the same category (e.g. worms, Trojans, spyware, adware) share similar characteristics and behavior patterns. These patterns are reflected by the infected computer's behavior. Thus, we hypothesize that it is feasible to learn the computer behavior in the presence of a certain type of malcode, which can be measured through the collection of various parameters along time (CPU, Memory, etc.) In the proposed approach a classifier is trained with computer measurements from infected and not infected computers. Based on the generalization capability of the

learning algorithm, we argue that a classifier can further detect previously unknown worm activity.

Signature:

A network IDS signature is a pattern that we want to look for in traffic. In order to give you an idea of the variety of signatures, let's quickly review some examples and some of the methods that can be used to identify each one:

- Connection attempt from reserved IP address. This is easily identified by checking the some address field in an IP header.
- Email containing a particular virus. The IDS can compare the subject of each email to the subject associated with the virus-laden email, or it can look for an attachment with a particular name.
- DNS buffer overflow attempt contained in the payload of a query. By parsing the DNS fields and checking the length of each of them, the IDS can identify an attempt to perform a buffer overflow using a DNS field. A different method would be to look for exploit shellcode sequence in the payload.
- Denial of service attack on a POP3 server caused by issuing the same command thousands of times. One signature for this attack would be to keep track of how many times the command is issued and to alert when that number exceeds a certain threshold.
- File access attack on an FTP server by issuing file and directory commands to it without first logging in. A state- tracking signature could be developed which would monitor FTP traffic for a successful login and would alert if certain commands were issued before the user had authenticated properly.

N-gram:

N-grams are all substrings of a larger string with a length n. A string is simply split into substrings of a fixed length n. For example, the string "MALWARE" , can be segmented into several 4-grams: "MALW" , "ALWA", "LWAR" , "WARE" and so on**.**

III. IMPLEMENTATION

The complete implementation has been divided into 3 Sections. Section I discusses Snort IDS which is used to detect known attacks. Section II discusses classification algorithm to detect whether a worm is present on the agent or not. Section III discusses signature generation algorithm used.

SECTION-1

Snort has existing rules like:

alert tcp $EXTERNAL_NET any -> $HOME_NET21 (msg: "PROTOCOL-FTP satan scan"; flow:to_server,established; content: "pass-satan"; fast_pattern:only; ,metadata:ruleset community, service ftp; classtype: suspicious-login; sid:359; rev:12;)

Such rules detect and block known attacks such as suspicious login via ftp in the above case.

Also, we can add new rules like:

alert tcp any any-> 23 (msg: "Intrusion detected…..Telnet Packet"; sid:496; rev:1;)

The above rule can be used to block any telnet packet coming into the system.

SECTION-2

Collecting training data for the classifier:

It is done by recording logs from windows system performance in the presence and absence of worms. We select the following attributes to be recorded at every 1 second interval:

- ICMP messages sent per second
- ICMP messages per second
- ICMP echo sent per second
- % Memory committed bytes in use
- % processor time

Stop recording logs when enough training data has been collected

Pre-processing training data:

- Remove all double quotes

- Convert comma separated log (csv) to space separated log.
- Fuzzify the log values: Divide each of the above five attributes values into four intervals and assign one of the following fuzzy descriptors: low (1) , medium (2) , high (3), very high (4) to the values.
    - ICMP messages sent per second:
      Between 0 and 2 - Low/1
      Between 2 and 3- Medium/2
      Between 3 and 4- High/3
      Above 4- very high/4
    - ICMP messages per second:
      Between 0 and 2 - Low/1
      Between 2 and 4 - Medium/2
      Between 4 and 5 - High/3
      Above 5 - very high/4
    - ICMP echo sent per second:
      Between 0 and 0.33 – Low/1
      Between 0.33 and 0.66 - Medium/2
      Between 0.66 and 1 - High/3
      Above 1 - very high/4
    - % Memory committed bytes in use:
      Between 0 and 20 - Low/1
      Between 20 and 30 - Medium/2
      Between 30 and 45 - High/3
      Above 45 - very high/4
    - % Processor time:
      Between 0 and 25 - Low/1
      Between 25 and 50 - Medium/2
      Between 50 and 75 - High/3
      Above 75 - very high/4
- Adding worm attribute to training data:
  We gave the following weights to five attributes:
    - ICMP messages sent per second- 8.33%
    - ICMP messages per second- 8.33%
    - ICMP echo sent per second- 8.34%
    - % Memory committed bytes in use- 25%
    - % Processor time – 50%

Then the total number of points for each tuple in training data was calculated using the formula: $weights_i$*$fuzzy\text{-}descriptor_i$ where i represents the five attributes

Threshold was selected by observing that very high (4) processor usage and low (1) value for
other attributes was the minimal condition required to indicate the presence of worm.

Threshold = 0.0833*1+0.0833*1+ 0.0834*1+0.25*1+ 0.5*4= 2.5

All tuples having point value greater than threshold are labeled as a worm in the training data.

Testing the classifier:

- Keep recording windows performance logs for the above 5 attributes.
- Keep pre-processing the collected logs in real time as above except the last step.
- Keep running naïve Bays classifier (described below) on the pre-processed logs.
- If the classifier detects a worm, an email is sent to admin that a worm is detected and worm packets are sent to the signature generator for creating a signature for the newly encountered worm.

Naïve-Bayes Classifier:

Given a set of variables, X={x1,x2,x3,….,xd}, we want to construct the posterior probability for the event C_j among a set of possible outcomes C={c1,c2,c3….,cd}. In a more familiar language, X is the predictors and C is the set of categorical levels present in the dependent variable. Using Bayes rule:

$$p(C_j|x_1,x_2,\dots,x_d) \propto p(x_1,x_2,\dots,x_d|C_j)p(C_j)$$

where $p(C_j|x1,x2,\dots..xd)$ is the posterior probability of class membership, i.e., the probability that X belongs to C_j. Since Naïve Bayes assumes that the conditional probabilities of the independent variables are statistically independent we can decompose the likelihood to a product of terms:

$$p(X|C_j) \propto \prod_{k=1}^{d} p(x_k|C_j)$$

and rewrite the posterior as:

$$p(X|C_j) \propto p(C_j) \prod_{k=1}^{d} p(x_k|C_j)$$

Using Bayes rule above, we label a new case X with a class level C_j that achieves the highest posterior probability.

Although the assumption that the predictor (independent) variables are independent is not always accurate, it does simplify the classification task dramatically, since it allows the class conditional densities p(xk|Cj) to be calculated separately for each variable, i.e., it reduces a high-dimensional density estimation task to a one-dimensional kernel density estimation. Furthermore, the assumption does not seem to greatly affect the posterior probabilities, especially in the regions near decision boundaries, thus, leaving the classification task unaffected.

SECTION-3

Pre-processing of data for signature generation:

- Worm code is sent to pre-processing algorithm where all the comments will be deleted. Comments used in worm codes obfuscate the real code so that matching algorithm will get confused due to comments.

Signature generation module:

- Once the code is pre-processed it's unique instances of keyword are generated using signature generation algorithm.
- This algorithm then converts these instances into hex code, which is treated as worm signature.
- The generated code is stored on the system for matching purpose.

N-gram technique:

N-grams are basically substrings of a larger string having length n. N-gram substring is generated simply by splitting a given string into substrings of length n. Example, string= " computer worm is malware ". N-grams for given string= "com", "put", "erw", "orm", "ism", "alw", "are". Here N=3. N can be any natural number.

As mentioned in above example N can be any natural number. We have taken value of N as 3.

We selected n as 3 because it provides detection of very small amount of changes in the code. Different values of N are tested before finalizing the value as 3. While testing N's value lowest numbers of false positives are generated for N=3 as compared to other values.

Matching of signatures:

- The generated signature is converted into n-grams.
- N-gram technique is already mentioned above.
- Generated n-grams are matched with incoming packets.
- Match ratio is obtained for signatures.
- A particular threshold value is decided to term the packet as malicious or not.
- If match ratio is above decided threshold value then packet is malicious else it is not.

IV. FUTURE SCOPE

The System can be expanded by including the ability to protect against all types of unknown attacks. Also combining the classifier with the signature generation can be done to make the system much more automated. Adding signature matching before the packet is sent to the agent can save time for detection of unknown worms. Also increasing the accuracy of signature generation can reduce number of false positives and negatives.

V. CONCLUSION

The system has been made to be eventually useful to organizations where a strict level of security is required as in the case of exchange of confidential data. The system in its current form is only able to detect whether a worm is present or not via the classifier. Signature generation is a separate module which is not yet integrated with the classifier. Thus, a

system for detecting and protecting against unknown worms was presented and implemented.

REFERENCES

[1] Robert Moskovitch, Yuval Elovici, Lior Rokach "Detection of Unknown Computer Worms based on Behavioral Classification of the Host", Ben Gurion University, Israel http://www.wi-fi.org/discover-wi-fi/wi-fi-direct

[2] Igor Santos, Yoseba K Penya, Jaime Devesa, Pablo G. Brings "N-grams-based file signature for malware detection", Deusto Technological Foundation

[3] I.S. Kim, M.H. Kim "Agent-based honeynet framework for protecting servers in campus networks", Soongsil University, Korea

[4] Lance Spitzner "Honeypots: Tracking Hackers", Addison Wesley

[5] Margaret H Dunham "Data Mining: Introductory And Advanced Topics", Pearson Education

[6] Young Tang, Shigang Chen "Defending against Internet worms: A signature-based approach", University of Florida, USA