International Journal of Advanced Research in Computer Science & Technology (IJARCST 2014)

Vol. 2, Issue 3 (July - Sept. 2014)

ISSN : 2347 - 8446 (Online)
ISSN : 2347 - 9817 (Print)

A New Intrusion Detection System for Modern Web-sites

[I]J. Srinivasarao [II]M. Mahesh Kumar

[I]Student, [II]Assistant Professor

[I,II]Dep. of IT, LBRCE, JNTUK University, Mylavaram, Andhra Pradesh, India

,

Abstract

Web-Delivered Services became an integral part of our daily life which enables us to communicate the personal information from anywhere. To manage this and to deal with the data complexity, web services have stimulated to a multitier design front-end logic and data from web server are outsourced to a database or file server. In this paper, we present Double Guard, an IDS system that models the network behavior of user sessions across both the front-end web server and the back-end database. By monitoring the both web and database activity it can identify the attacks that the IDS could not be identifying properly. We implemented Double Guard using an Apache web server with MySQL and deployed in processed real-world traffic over both dynamic and static web applications. Double Guard, finally able to expose a wide range of attacks greater accuracy for static and dynamic web services.

Key words

web server, dynamic web, double guard, IDS

I. Introduction

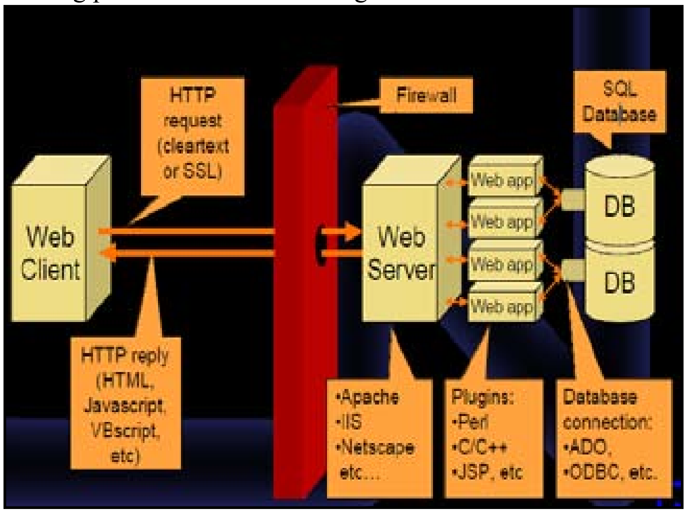
All most all works are now a day is being done over internet including selling and purchasing also. Web-Delivered Services and applications have increased in both reputation and complication over the past few years. Daily tasks, such as banking, travel, and social networking, are all done via the web. Such services naturally employ a web server front end that runs the application user interface logic, as well as a back-end server that consists of a database or file server. Due to their omnipresent use for personal and/or corporate data, web services have always been the target of attacks. These attacks have freshly become more different, as awareness has shifted from attacking the front end to exploiting vulnerabilities of the web applications [6], [5], [1] in order to fraudulent the back-end database system [14] (e.g., SQL injection attacks [2], [3]). A superfluity of Intrusion Detection Systems (IDSs) currently examines network packets individually within both the web server and the database system. However, there is very little work being performed on multitier Anomaly Detection (AD) systems that generate models of network Behavior for both web and database network interactions. In such multitier architectures, the back-end database server is often protected behind a firewall while the web servers are remotely accessible over the Internet. Unfortunately, though they are protected from direct remote attacks, the back-end systems are susceptible to attacks that use web requests as a means to exploit the back end. To protect multitier web services, Intrusion detection systems have been widely used to detect known attacks by matching misused traffic patterns or signatures [4], [10-12]. A class of IDS that leverages machine learning can also detect unknown attacks by identifying abnormal network traffic that deviates from the so-called "normal" behavior previously profiled during the IDS training phase. Individually, the web IDS and the database IDS can detect abnormal network traffic sent to either of them. However, we found that these IDSs cannot detect cases wherein normal traffic is used to attack the web server and the database server. For example, if an attacker with non admin privileges can log in to a web server using normal-user access credentials, he/she can find a way to issue a privileged database query by exploiting vulnerabilities in the webserver. Neither the web IDS nor the database IDS would detect this type of attack since the web IDS would merely see typical user login traffic and the database IDS would see only the normal traffic of a privileged user. This type of attack can be readily detected if the database IDS can identify that a privileged request from the web server is not associated with userprivileged access. Unfortunately, within the current multithreaded web server architecture, it is not feasible to detect or profile such causal mapping between web server traffic and DB server traffic since traffic cannot be clearly attributed to user sessions. In this paper, we present Double Guard, a system used to detect attacks in multitier web services. Our approach can create normality models of isolated user sessions that include both the web frontend (HTTP) and back-end (File or SQL) network transactions. To achieve this, we employ a lightweight virtualization technique to assign each user's web session to a dedicated container, an isolated virtual computing environment. We use the container ID to accurately associate the web request with the subsequent DB queries. Thus, Double Guard can build a causal mapping profile by taking both the web server and DB traffic into account.

II. Threat Model And System Architecture

We primarily set up our threat model to comprise our assumptions and the types of attacks we are aiming to defend against. We assume that both the web and the database servers are susceptible. Attacks are network borne and come from the web clients; they can launch application layer attacks to conciliation the web servers they are connecting to. The attackers can bypass the web server to openly attack the database server. We presuppose that the attacks can neither be detected nor barred by the current web server IDS, that attacker may take over the web server after the attack, and that later they can get hold of full control of the web server to launch consequent attacks. For example, the attackers could modify the application logic of the web applications, eavesdrop or hijack other users' web requests, or intercept and modify the database queries to steal sensitive data beyond their privileges.

Alternatively, at the database end, we suppose that the database server will not be entirely taken over by the attackers. Attackers may wallop the database server through the web server or, more directly, by submitting SQL queries, they may acquire and infect sensitive data within the database. These assumptions are practical since, in most cases, the database server is not out in the open to the public and is therefore thorny for attackers to completely take over.

We assume no prior knowledge of the source code or the application logic of web services deployed on the web server. In addition, we are analyzing only network traffic that reaches the web server and database. We assume that no attack would occur during the training phase and model building.


HTTP request (cleartext or SSL)
Firewall
SQL Database
Web Client
Web Server
Web app
Web app
Web app
Web app
DB
DB
HTTP reply (HTML, Javascript, VBscript, etc)
•Apache •IIS •Netscape etc…
Plugins: •Perl •C/C++ •JSP, etc
Database connection: •ADO, •ODBC, etc.

A. Architecture and Confinement

All network traffic, from both legitimate users and adversaries, is received intermixed at the same web server. If an attacker compromises the web server, he/she can potentially affect all future sessions (i.e., session hijacking). Assigning each session to a dedicated web server is not a realistic option, as it will deplete the web server resources. To achieve similar confinement while maintaining a low performance and resource overhead, we use lightweight virtualization.

It is possible to initialize thousands of containers on a single physical machine, and these virtualized containers can be discarded, reverted, or quickly reinitialized to serve new sessions. A single physical web server runs many containers, each one an exact copy of the original web server. Our approach dynamically generates new containers and recycles used ones. As a result, a single physical server can run continuously and serve all web requests. However, from a logical perspective, each session is assigned to a dedicated web server and isolated from other sessions. Since we initialize each virtualized container using a read-only clean template, we can guarantee that each session will be served with a clean web server instance at initialization. We choose to separate communications at the session level so that a single user always deals with the same web server. Sessions can represent different users to some extent, and we expect the communication of a single user to go to the same dedicated web server, thereby allowing us to identify suspect behavior by both session and user. If we detect abnormal behavior in a session, we will treat all traffic within this session as tainted. If an attacker compromises a vanilla web server, other sessions' communications can also be hijacked. In our system, an attacker can only stay within the web server containers that he/she is connected to, with no knowledge of the existence of other session communications. We can thus ensure that legitimate sessions will not be compromised directly by an attacker.

Fig. 1 illustrates the classic three-tier model. At the database side, we are unable to tell which transaction corresponds to which client request. The communication between the web server and the database server is not separated, and we can hardly understand the relationships among them. Fig.2 depicts how communications are categorized as sessions and how database transactions can be related to a corresponding session. According to Fig.1, if Client2 is malicious and takes over the web server, all subsequent database transactions become suspect, as well as the response to the client. By contrast, according to Fig.2, Client 2 will only compromise the VE2, and the corresponding database transaction set will be the only affected section of data within the database.

B. Building the mapping Model

This container-based and session- separated web server architecture not only provide the isolated information flows that are separated in each container session but also enhances the security performances. It allows us to recognize the mapping between the web server desires and the ensuing DB queries, and to utilize such a mapping model to detect anomalous behaviors on a session/client level. In typical three-tiered web server architecture, the web server receives HTTP requests from user clients and then issues SQL queries to the database server to retrieve and update data. These SQL queries are causally dependent on the web request hitting the web server. We want to model such causal mapping relationships of all legitimate traffic so as to detect abnormal/ attack traffic.

In practice, we are unable to build such mapping under a classic three-tier setup. Although the web server can distinguish sessions from different clients, the SQL queries

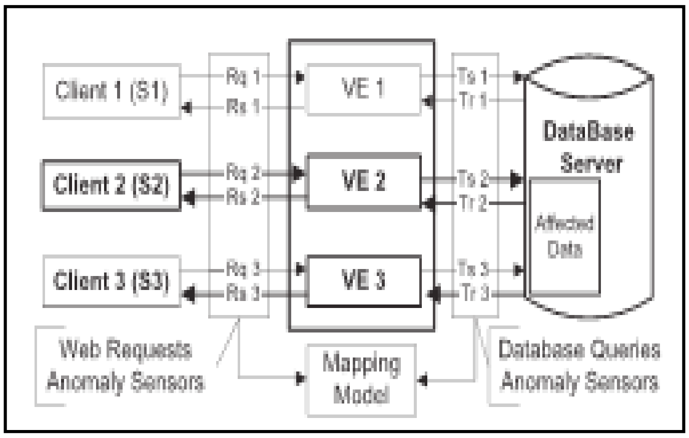


Fig. 1: classic three tire model

are mixed and all from the same web server. It is impossible for a database server to determine which SQL queries are the results of which web requests, much less to find out the relationship between them. Even if we knew the application logic of the web server and were to build a correct model, it would be impossible to use such a model to detect attacks within huge amounts of concurrent real traffic unless we had a mechanism to identify the pair of the HTTP request and SQL queries that are causally generated by the HTTP request. However, within our container-based web servers, it is a straightforward matter to identify the causal pairs of web requests and resulting SQL queries in a given session. Moreover, as traffic can easily be separated by session, it becomes possible for us to compare and analyze the request and queries across different sessions. Section 4 further discusses how to build the mapping by profiling session traffics.

Once we build the mapping model, it can be used to detect abnormal behaviors. Both the web request and the database queries within each session should be in accordance with the model. If there exists any request or query that violates the normality model within a session, then the session will be treated as a possible attack.

III. Double Guard

Vulnerabilities Due to inappropriate Input Processing Cross Site Scripting is a distinctive attack method where in attackers embeds malevolent client scripts via genuine user inputs. In Double Guard, the entire user input values are normalized so as to build a mapping model based on the structures of HTTP requests and DB queries. Once the malicious user inputs are normalized, Double Guard cannot detect attacks hidden in the values. These attacks can occur even without the databases. Double Guard offers a complementary approach to those research approaches of detecting web attacks based on the characterization of input values [5,6,7].

A. Possibility of Evading Double Guard

Our hypothesis is that an mugger can obtain "full control" of the web server thread that he/she connects to. That is, the attacker can only take over the web server occurrence running in its remote container. Our architecture ensures that every client be definite by the IP address and port container pair, which is exclusive for each session. Therefore, hijacking an existing container is not viable because traffic for other sessions is never directed to an full container. If this were not the case, our architecture would have been similar to the conventional one where a single web server runs many different processes. Moreover, if t he database authenticates the sessions from the web server, then each container connects to the database using either admin user account or non admin user account and the connection is authenticated by the database. In such case, an attacker will authenticate using a non admin account and will not be allowed to issue admin level queries. In other words, the HTTP traffic defines the privileges of the session which can be extended to the back-end database, and a non admin user session cannot appear to be an admin session when it comes to back-end traffic.Within the same session that the attacker connects to, it is allowed for the attacker to launch "mimicry" attacks. It is possible for an attacker to discover the mapping patterns by doing code analysis or reverse engineering, and issue "expected" web requests prior to performing malicious database queries. However, this significantly increases the efforts for the attackers to launch successful attacks. In addition, users with non admin permissions can cause minimal (and sometimes zero) damage to the rest of the system and therefore they have limited incentives to launch such attacks.By default, Double Guard normalizes all the parameters. Of course, the choice of the normalization parameters needs to be performed vigilantly. Double Guard offers the ability of normalizing the parameters so that the user of Double Guard can choose which values to normalize. For example, we can choose not to normalize the value "admin" in "user = 'admin'." Likewise, one can choose to normalize it if the administrative queries are structurally different from the normal-user queries, which is common case. Additionally, if the database can authenticate admin and non admin users, then privilege escalation attacks by changing values are not feasible (i.e., there is no session hijacking).

IV. Modeling Deterministic Mapping

Due to their diverse functionality, special web applications exhibit dissimilar characteristics. Many websites serve only fixed content, which is updated and often managed by a Content Management System (CMS). For a static website, we can build an accurate model of the mapping relationships between web requests and database queries since the links are static and clicking on the same link always returns the same information. However, some websites (e.g., blogs, forums) allow regular users with non administrative privileges to update the contents of the server data. This creates tremendous challenges for IDS system training because the HTTP requests can contain variables in the passed parameters.
For example, instead of one-to-one mapping, one web request to the web server usually invokes a number of SQL queries that can vary depending on type of the request and the state of the system. Some requests will only retrieve data from the web server instead of invoking database queries, meaning that no queries will be generated by these web requests. In other cases, one request will invoke a number of database queries. Finally, in some cases, the web server will have some periodical tasks that trigger database queries without any web requests driving them. The challenge is to take all of these cases into account and build the normality model in such a way that we can cover all of them.
All communications from the clients to the database are separated by a session. We assign each session with a unique session ID. Double Guard normalizes the variable values in both HTTP requests and database queries, preserving the structures of the requests and queries. To achieve this, Double Guard substitutes the actual values of the variables with symbolic values

A. Modeling for Static Websites

In the case of a static website, the nondeterministic mapping does not exist as there are no available input variables or states for static content. We can easily classify the traffic collected by sensors into three patterns in order to build the mapping model. As the traffic is already separated by session, we begin by iterating all of the sessions from 1 to N. For each rm Є REQ, we maintain a set AR to record the IDs of sessions in which rm appears. The same holds for the database queries; we have a set AQsfor each Qs Є SQL to record all the session IDs. To produce the training model, we leverage the fact that the same mapping pattern appears many times across different sessions.
The algorithm we used to monitor the system is-

1. Monitoring algorithm

- Input: system log
- 1. Extract the request arrivals for all sessions, page viewing time and the sequence ofN requested objects for each user from the system log.
- 2. Compute the entropy of the requests per session using the formula:
- $H(R) = -j\ Pj(rj) \log Pj(rj)$
- 3. Compute the trust score for each and every user based on their viewing time and accessing behaviour .

2. Detection Algorithm

- Input the predefined entropy of requests per session and the trust score for each user.
- Define the threshold related with the trust score (Tts)
- Define the threshold for allowable deviation (Td)
- For each session waiting for detection
- Extract the requests arrivals
- Compute the entropy for each session using (4)
- $Hnew(R) = -j\ Pj(rj) \log Pj(rj)$
- Compute the degree of deviation:
- $D = |Hnew(R)| - |H(R)|$
- If the degree of deviation is less than the allowable threshold (Td), and user's trust score is greater
- than the threshold (Tts), then

- Allow the session to get service from the web server
- Else
- The session is malicious; drop it

V. Testing for Static Websites

Once the normality model is generated, it can be engaged for training and detection of anomalous behavior .During the testing phase, each session is compared to the normality model. We begin with each distinct web request in the session and, since each request will have only one mapping rule in the model, we simply compare the request with that rule.

A. Modeling of Dynamic Patterns

Fascinatingly, our blog website built for testing purposes shows that, by only modeling nine fundamental operations, it can cover most of the operations that appeared in the genuine captured traffic. For each operation (e.g., reading an article),we build the model as follows: in one session, we act up on only a single read operation, and then we get hold of the set of triggered database queries. Since we cannot guarantee that each user perform only a single operation inside each session in real traffic, we use a tool called Selenium [7,8] to separately produce training traffic for each operation. In each session, the tool performs only one basic operation. When we repeat the operation multiple times using the tool, we can easily substitute the different parameter values that we want to test (in this case, reading different articles).
Finally, we obtain many sets of queries from one sessionand assemble them to obtain the set of all possible queries resulting from this single operation.

VI. Detection for Dynamic Websites

Once we build the separate single operation models, they can be used to detect abnormal sessions. In the testing phase, traffic captured in each session is compared with the model. We also iterate each distinct web request in the session. For each request, we determine all of the operation models that this request belongs to, since one request may now appear in several models. We then take the entire corresponding query sets in these models to form the set CQS. For the testing session i, the set of DB queries Q should be a subset of the CQS. Otherwise, we would find some unmatched queries. For the web requests in Rii, each should either match at least one request in the operation model or be in the set EQS. If any unmatched web request remains, this indicates
that the session has violated the mapping model.

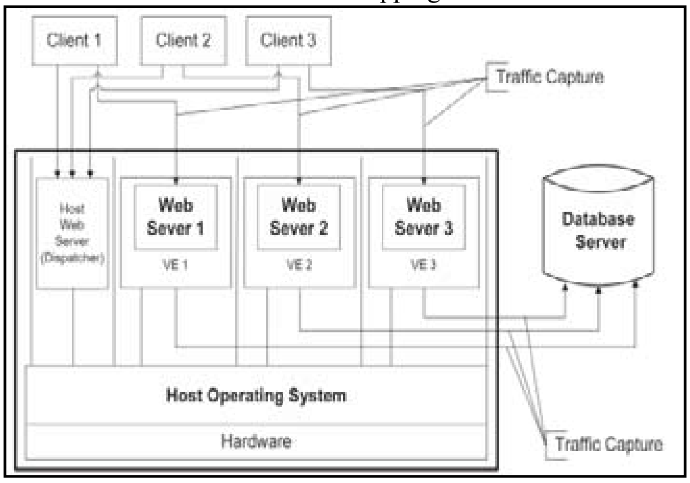


Fig. 2 : Model prototype

VII. Performance Evaluation

We employed a model of Double Guard using a web server with a back-end DB. We also set up two testing websites, one static and the other dynamic. To evaluate the detection results for our system, we investigated four classes of attacks, as discussed in Section 3, and measured the false positive rate for each of the two websites.

A. Implementation

In our prototype, we prefer to dispense each user session into a different container; nevertheless, this was a design pronouncement. For instance, we can allocate a new container per each new IP address of the client. In our accomplishment, containers were salvaged based on events or when sessions time out. We we reable to use the same session tracking methods as put into practiced by the Apache server (cookies, mod_usertrack,etc.) because lightweight virtualization containers do not enforce high memory and storage overhead.
Thus, we could maintain a large number of parallel-running Apache instances similar to the Apache threads that the server would maintain in the scenario without containers. If a session timed out, the Apache instance was terminated along with its container. In our prototype implementation, we used a 60-minute timeout due to resource constraints of our test server. However, this was not a limitation and could be removed for a production environment where long-running
To test our system in a dynamic website scenario, we setup a dynamic Blog using the Word press [8] blogging software. In our deployment, site visitors were allowed to read, post, and comment on articles. All models for the received front-end and back-end traffic were generated using these data.
We discuss performance overhead, which is common for both static and dynamic models, in the following section. In our analysis, we did not take into consideration the potential for caching expensive requests to further reduce the end-to-end latency; this we left for future study.

B. Attack Detection

Formerly the model is built, it can be used to perceive malicious sessions. For our static website testing, we used the fabrication website, which has regular visits of around 50100 sessions per day. We accumulated regular traffic for this construction site, which totaled 1,172 sessions. We used the sql map [12], which is an automatic tool that can generate SQL injection attacks. Web server scanner tool that executes wide-ranging tests, and were used to engender a number of web server-aimed http. We performed the equivalent attacks on both Double Guard and a classic three tier architecture with a network IDS at the web server side and a database IDS at the database side. As there is no popular anomaly-based open source network IDS available, we used Snort [1,9] as the network IDS in front of the web server, and we used Green SQL as the database IDS. For Snort IDS, we downloaded and enabled all of the default
rules from its official website.
We put Green SQL into database firewall mode so that it would automatically white list all queries during the learning mode and block all unknown queries during the detection mode. Table 2 shows the experiment results where Double Guard was able to detect most of the attacks and there were 0 false positives in our static website testing.

VIII. Conclusion

We proposed an intrusion detection system that builds models of normal performance for multi tiered web applications from both front-end web (HTTP) requests and back-end database (SQL) queries. Disparate earlier approaches that correlated or summarized alerts generated by self-governing IDSs, Double Guard forms a container-based IDS with multiple input streams to produce alerts. We have justified that such association of input streams provides a better characterization of the system for abnormality detection n because the intrusion sensor has a more precise normality replica that identifies a wider range of threats. We achieved this by isolating the flow of in format ion from each web server session with a lightweight virtualization.

References

[1] *Meixing Le, AngelosStavrou, Brent ByungHoon Kang, "DoubleGuard: Detecting Intrusions in Multitier Web Applications" IEEE TRANSACTIONS ON DEPENDABLE AND SECURE COMPUTING, VOL. 9, NO. 4, JULY/ AUGUST 2012*

[2] *C. Anley, "Advanced Sql Injection in Sql Server Applications,"technical report, Next Generation Security Software,Ltd., 2002.*

[3] *K. Bai, H. Wang, and P. Liu, "Towards Database Firewalls," Proc.Ann. IFIP WG 11.3 Working Conf. Data and Applications Security(DBSec '05), 2005.*

[4] *B.I.A. Barry and H.A. Chan, "Syntax, and Semantics-BasedSignature Database for Hybrid Intrusion Detection Systems,"Security and Comm. Networks, vol. 2, no. 6, pp. 457-475, 2009.*

[5] *D. Bates, A. Barth, and C. Jackson, "Regular ExpressionsConsidered Harmful in Client-Side XSS Filters," Proc. 19th Int'lConf. World Wide Web, 2010.*

[6] *M. Christodorescu and S. Jha, "Static Analysis of Executables toDetect Malicious Patterns," Proc. Conf. USENIX Security Symp.,2003.*

[7] *M. Cova, D. Balzarotti, V. Felmetsger, and G. Vigna, "Swaddler:An Approach for the Anomaly-Based Detection of State Violationsin Web Applications," Proc. Int'l Symp. Recent Advances in IntrusionDetection (RAID '07), 2007.*

[8] *H. Debar, M. Dacier, and A. Wespi, "Towards a Taxonomy ofIntrusion-Detection Systems," Computer Networks, vol. 31, no. 9,pp. 805-822, 1999.*

[9] *V. Felmetsger, L. Cavedon, C. Kruegel, and G. Vigna, "TowardAutomated Detection of Logic Vulnerabilities in Web Applications,"Proc. USENIX Security Symp., 2010.*

[10] *Y. Hu and B. Panda, "A Data Mining Approach for DatabaseIntrusion Detection," Proc. ACM Symp. Applied Computing (SAC),H. Haddad, A. Omicini, R.L. Wainwright, and L.M. Liebrock, eds.,2004.*

[11] *Y. Huang, A. Stavrou, A.K. Ghosh, and S. Jajodia, "EfficientlyTracking Application Interactions Using Lightweight Virtualization,"Proc. First ACM Workshop Virtual Machine Security,2008.*

[12] *H.-A. Kim and B. Karp, "Autograph: Toward Automated Distributed Worm Signature Detection," Proc. USENIX SecuritySymp., 2004.*

[13] *C. Kruegel and G. Vigna, "Anomaly Detection of Web-BasedAttacks," Proc. 10th ACM Conf. Computer and Comm. Security(CCS '03), Oct. 2003.*

[14] *S.Y. Lee, W.L. Low, and P.Y. Wong, "Learning Fingerprints for aDatabase Intrusion Detection System," ESORICS: Proc. EuropeanSymp. Research in Computer Security, 2002.*

SrinivasaRao Jalasutram received his B.Tech degree in Information Technology and Engineering from Paladugu Parvathi Devi College of Engineering & Technology, surampall, near Nunna, Vijayawada, A.P, India, in 2012. Currently pursuing M.Tech in Lakireddy Bali Reddy College of Engineering, Mylavaram, India. His research interest includes: Network security.

M. Mahesh Kumar received her B.Tech degree in IT from Koneru Lakshmaiah College Of Engineering. Vaddeswaram. (Nagarjuna University) in 2010 and completed his M.Tech degree in Computer CSE from JNTU Kakinada in 2013. Currently working as an Assistant Professor in Lakireddy Bali Reddy College of Engineering, mylavaram. His research interest includes: Network security and image processing.