Journal of Information Security and Applications 67 (2022) 103169

Contents lists available at ScienceDirect

Journal of Information Security and Applications

journal homepage: www.elsevier.com/locate/jisa

A deep learning assisted personalized deception system for countering web application attacks

Waleed Bin Shahid, Baber Aslam, Haider Abbas *, Hammad Afzal, Saad Bin Khalid

National University of Sciences and Technology (NUST), Islamabad, Pakistan

ARTICLE INFO

Keywords:
Web security
Deception
Web deception
Web attacks
HTTP
Web honeypot

ABSTRACT

Recent years have seen momentous growth in web attacks that has motivated researchers to come up with sophisticated techniques to tackle them. Lately, there has been growing interest to counter web attacks using deception techniques because they help in realizing attacker behavior, motives and abilities besides protecting the website. This paper proposes a complete high interaction web deception system which is assisted by a hybrid attack detection module comprising of a deep learning based classifier coupled with a cookie analysis engine that helps in attacker profiling. The detection module routes malicious HTTP (Hypertext Transfer Protocol) requests to the dockers based deception system which is controlled and managed by a docker controller. The proposed containerized approach makes the system efficient, reduces latency and enhances runtime development. The key feature of attacker profiling empowers the proposed system to deal with attackers carrying zero day attack payloads besides providing efficient session management and scenario based emulation. The proposed deception system caters for all major web application attacks and has high attacker engagement when tested in a real-time environment. Moreover, the proposed framework is scalable, agile and supports easy framework modification making it suitable even for IoT (Internet of Things) networks. The proposed attack detection module gave an accuracy of 99.94% and is less time consuming than other research works because of its profiling feature. These features give the proposed framework a high competitive edge over other web deception solutions.

1. Introduction

The widespread use of internet has enabled everyone to make data and information easily accessible by sharing it on websites. Although this makes our lives easier as the data we need is just few clicks away, but there is a dark side as well. These websites can be vulnerable to severe data breaches and cyber-attacks resulting in loss of availability, integrity and confidentiality of not just the data placed there but also the back-end infrastructure like databases, connected systems and end users who access the website. These vulnerabilities are surfaced because of many issues like misconfigurations, improper data handling, insecure coding practices, insecure database connections, lack of input sanitization and data validation and weak authentication and access control. These web attacks result in massive information, economic and reputational loss which becomes an even serious issue in the absence of cyber-crime regulations.

Security researchers have come up with many ways and techniques to counter these attacks on web applications [1]. Since web application attacks take place on the application layer of the TCP/IP model, therefore, commonly used network firewalls do not stop them. Security researchers have come up with dedicated Web Application Firewalls (WAFs) [2], as shown in Fig. 1, which are designed to detect and stop these attacks. Likewise, attackers also keep on updating themselves with advanced approaches and ways to bypass these defensive techniques either by using commonly available Web Application Vulnerability Scanners (WAVs) or carefully crafted custom scripts. Since WAFs are mostly rule-based, they are good at detecting WAVs but perform poorly against zero day attacks, custom scripts and payloads.

To overcome these problems, security researchers have moved towards using advanced machine learning [3] and deep learning techniques for detecting these web attacks [4,5]. These techniques have the ability to detect anomalies with very high accuracy as compared to rule-based techniques [6,7]. However, all these techniques are defensive in nature and focus only on detecting and stopping attacks on web applications. Lately, interest in studying attacker behavior, tactics, techniques and capacity has grown very rapidly.

Deception is the art of luring attackers by deliberately inserting and advertising weaknesses in the system. This technique has been used for a long time in order to protect systems and resources from

* Corresponding author.
E-mail address: dr.h.abbas@ieee.org (H. Abbas).

https://doi.org/10.1016/j.jisa.2022.103169

Available online 19 April 2022

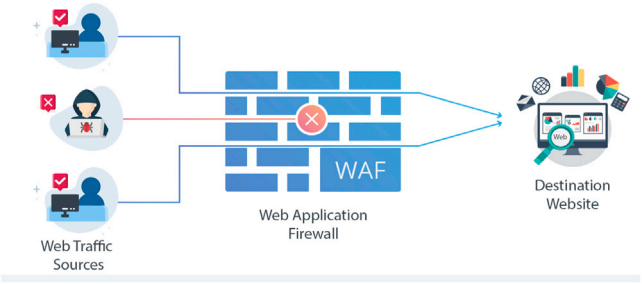

Fig. 1. Web application firewall.

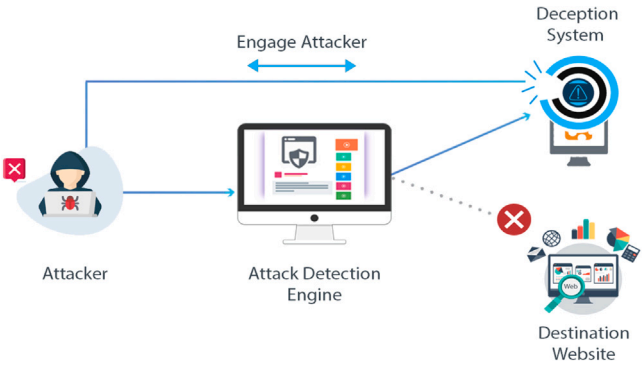

Fig. 2. A simple web deception system.

all kinds of attacks [8]. Honeypots provide the most basic means of deception whereby attackers are lured in to a fake system with honey information, credentials and data [9]. This relieves the original system of attacker's attention thereby keeping it safe and protected. This honeypot technology has also been employed in order to protect web applications as they are one of the most common attack targets. Simple honeypots have a lot of deployment, scalability and interaction issues which put a cap on their performance. Most honeypot solutions are hard to operate as they require advanced skillset to manage and update. Moreover, if compromised, they can also be used as a pivot point to launch attacks on the actual system resulting in more damage [10].

Last decade has witnessed growing interest of security researchers in creating full-fledged deception systems which are scalable, lightweight and are hard to find out and compromise. These deception systems are also helpful in learning about attacker's motives, ability, tactics and methodologies that help in improving security of the actual system. Like many other areas, websites can also be protected by creating these sophisticated deception schemes as shown in Fig. 2. These systems lure the attacker in, never let him feel trapped, engage him and extract maximum information. Most of the existing web deception techniques are not lightweight, are resource intensive and do not work good in real time. Moreover, they are not linked with a powerful attack detection module which should augment their performance.

The proposed web deception framework addresses all these issues related to existing techniques that use cyber deception to counter web attacks. It is a continuation of our research work which discusses web attack detection using a deep learning based hybrid approach [11]. The main research contributions of the proposed framework are listed as follows:

1. A novel lightweight, scalable and agile high-interaction web deception framework has been proposed which is based on docker containers with low process utilization, easy runtime development and smooth deployment.
2. The framework caters for all major web attacks mentioned in the OWASP top ten category whereas many real world attacker engagement scenarios have also been covered, implemented and discussed.
3. Strengthening of deception framework as it is coupled with a hybrid web attack detection framework comprising of a deep learning based module and a Cookie Analysis Engine for attacker profiling. This combination of attack detection with deception system strengthens the later to have session maintenance and provide customized deception and scenario based emulation in a very efficient manner.
4. An efficient framework which saves useful processing time when deployed in real time as the attacker profiling feature makes deception way more convenient by making it simpler and less time consuming because of the key functionality of session maintenance and attacker profiling.

This paper is divided into five sections. Introduction is followed by Section 2 which consists of relevant research works being carried out in the field of web deception. These techniques have been compared and evaluated on the basis of their strengths, weaknesses and attacks covered which necessitates the idea to perform further research in this area. Section 3 explains the proposed web deception system along with the deep learning based attack detection and attacker profiling module whereas Section 3.5 discusses attacker engagement scenarios. Section 4 discusses performance evaluation, testing and validation of the proposed framework along with comparative analysis with related research works. The research is concluded in Section 5 along with an insight into the future work.

2. Related work

Several research works have discussed honeypots and deception systems to counter web application attacks. Prime focus of researchers is to attract the attackers to deceptive web pages that have lures inside and later analyzing the attack statistics. There is a lack of noteworthy research on diverting real-time web attacks launched on a website towards a deception system which should provide personalized deception.

Timothy Barron, et al. [12] proposed two techniques, web tripwires and login rituals which rely on deception to extend authentication in web applications. The proposed system relies on user's habits on the website to protect against authentication compromise. Amirreza Niakanlahiji, et al. [13] presented their research work Honey-Bug which offers a personalized deception plan for web applications based on learning attacker's behavior over time. The research work is strengthened by an evidential reasoning framework being used for attacker profiling which is not based on machine learning but on common WAF rules [14]. Kui Jiang, et al. [15] presented a web honeypot for Cross Site Scripting (XSS), SQL injection and file inclusion attacks. In this research work, data analysis has been strengthened using an ensemble based machine learning approach, called Regret. NR Fitri, et al. [16] came up with a low interaction honeypot along with a firewall to counter open-source Slowloris DDoS attacks on Apache web servers. Octavian Grigorescu, et al. [17] created honeypots based on *Capture the Flag* games of varying complexity to keep the attackers occupied and learn their behavior related to website and port scanning. All of these proposed research works do not address flexibility, scalability and ease of implementation. Moreover, none of these have augmented their functionality using deep learning approaches.

Djamaluddin, et al. [18] presented an approach based on implementation of deception and Moving Target Defense (MTD) while utilizing multiple web servers, applications, languages and databases. Daniel Fraunholz and Hans D. Schotten [19] proposed a web deception technique based on fake banner replies, robot.txt generation, fake error code responses, honey and decoy files, adaptive delaying and honey comment injections. Blake Henderson, et al. [20] developed a honeypot system for spies who collect documents after searching world wide

web for intelligence related information. Zhaopeng Jia, et al. [21] presented a micro-honeypot which tracks a web attacker based on browser fingerprinting techniques. Marius Musch, et al. [22] proposed an easy to deploy automatic generation of low interaction honeypots for web applications. Ionuţ Cernica, et al. [23] proposed a high interaction Wordpress honeypot module for finding new vulnerabilities in plugins and themes. Ahmed El-Kosairy and Marianne A. Azer [24] came up with a web deception scheme based on game theory along with the use of honey-web and honey-tokens to counter attacks on production websites. Daniel Fraunholz, et al. [25] combined deception-based techniques in a reverse proxy, named Cloxy, to provide deception as a service to web applications by using honey files, code obfuscation, cookie scrambling and content modification techniques.

Jianboa Lin, et al. [26] presented an approach based on Web Shadow service by cloning the actual protected website and adding deceptive lures to detect targeted persistent attacks. Supeno Djanali, et al. [27] proposed a low-interaction honeypot designed to reveal attacker's identity in case of XSS or SQL injection attacks, using the LikeJacking technique [28]. They targeted real-life attackers unlike Glasptop [29] that used Google Hack Database for creating a fake website page. John P. John, et al. [30] presented a honeypot approach that attracts attackers looking for vulnerable web pages using different techniques. Tomohisa Ishikawa and Kouichi Sakurai [31] proposed a web application deception proxy to counter XSS and SQLi attacks. The proxy analyzes HTTP response from the original web application and inserts a decoy if it contains the form. This not only increases the number of parameters but also increases assessment time by automated scanners or penetration testers. Nikos Virvilis, et al. [32] presented a web server honey tokens technique like fake entries in robots.txt file, using invisible links and honey-token HTML comments etc.

Supeno Djanali, et al. [33] proposed a Raspberry Pi based honeypot for SQL injection-based attacks by acting as a proxy between attacker and the web server, thereby hiding the later. Ioannis Koniaris, et al. [34] proposed a honeypot which trap attackers trying to exploit the SSH service to gain illegal web server access. The proposed work used Kippo SSH honeypot [35] as a baseline. G Leaden, et al. [36] proposed a honeypot which purports to be a REST API in order to prevent and analyze XSS and DDoS attacks. This approach makes it extremely difficult for the attackers to fingerprint the honeypot. The proposed work also uses delaying tactics against WAVs. Cristiano de Faveri and Ana Moreira [37] proposed a model based on aspect oriented techniques and Software Product Lines (SPL) for generating adaptive and defensive deception strategies by incorporating honeywords to mitigate offline password attacks. SDandy Kalma Rahmatullah, et al. [38] proposed a Cubieboard based honeypot for protecting the real web server against file inclusion and directory bruteforcing attacks.

Several research works have used containerization to implement their honeypots and deception systems. SMartin Valicek, et al. [39] discussed creating high interaction honeypots using dockers consisting both Windows and Linux. The research work highlights docker flexibility but does not address web attacks. Dubravko Sever and Tonimir Kišasondi [40] discussed the efficiency and security issues in docker based honeypot systems and claim that although dockers are more efficient but their security can be enhanced by following certain best practices. Fabio De Gaspari, et al. [41] used dockers to come up with a new architecture named AHEAD for active defense. The proposed work discourages the addition of new fake systems and empowers the production system to be capable of providing active defense. Andronikos Kyriakou, et al. [42] came up with a docker based honeypot approach by deploying Cowrie [43], Dionaea [44] and Glastopf [29]. The proposed framework makes use of Elastic Stack for better visualization on a single platform. Sergiu Eftimie and Ciprian Racuciu [45] have also discussed cloud-based honeypot using dockers in order to reduce implementation and maintenance complexity.

The growing threat of web attacks has motivated researchers to come up with techniques that not only thwart these attacks but also extract maximum possible information about the attackers through deceptive engagement. A wide variety of lures are used for trapping but their deceptive effectiveness varies and is dependent on parameters like architecture, interaction level, runtime development and process utilization. Table 1 presents a summary of related work on web deception by highlighting deceptive effectiveness and employed techniques. Further analysis of these techniques lead us to following observations.

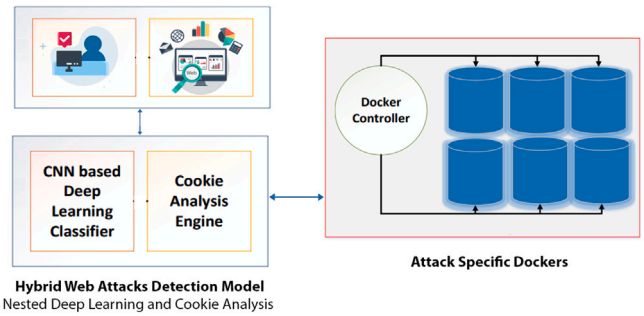

Fig. 3. Proposed web deception framework.

1. Most of the existing web deception techniques do not offer customized dynamic deception and rather rely on static honeypots in order to engage the attackers because they do not maintain attacker sessions resulting in low productivity when dealing with experienced attackers and zero-day attacks.
2. Techniques which focus on customized web deception for attackers are not supported by ample detection modules for attack detection and attacker profiling correspondingly.
3. The existing techniques are not flexible as they do not support framework modification and therefore cannot be applied in real time or in other scenarios. (See Table 1.)

3. Proposed personalized web deception framework

The proposed web deception framework caters for a large number of web application attacks in real time by engaging the attacker in an efficient way through deceptive lures running in separate dockers which are controlled and managed by the docker controller. The proposed framework is suitable for deployment in any scenario as it is also coupled with a hybrid web attack detection framework as highlighted in Fig. 3 where traffic detected as benign is routed to the website whereas malicious HTTP traffic is diverted towards the deception system. The hybrid web attack detection framework comprises of two modules, a convolutional neural network based deep learning classifier and a Cookie Analysis Engine (CAE). This hybrid detection framework when coupled with the proposed deception system enables the later to manage user sessions, enhances agility, effectiveness and robustness as explained in Section 4.

3.1. Hybrid web attack detection framework

The hybrid web attack detection framework comprises of a Convolutional Neural Network based classifier and a Cookie Analysis Engine (CAE). The framework detects and counters web attacks in real time and also profiles the attacker, a feature which helps in adding more efficiency to the system [11]. Fig. 4 shows step by step details of how the framework works.

Table 1
Existing work on countering web attacks through honeypots and deception.

Research works	Year	Deceptive mechanism	Technique(s) used	Attacks covered
Timothy Barron, et al. [12]	2021	Reverse Proxy	Login rituals and web tripwires	Authentication Attacks
Niakanlahiji et al. [13]	2020	Customized deception	Evidential reasoning framework for attacker behavi	Attacks covered by ModSecurity [14]
Jiang and Zheng [15]	2020	Crawler technology	Ensemble based machine learning technique	SQLi, XSS and file inclusion
Fitri et al. [16]	2020	Low interaction	Packet arrival rate analysis	Slowloris DDoS
Grigorescu et al. [17]	2020	High interaction	CTF type games	Port and web scanning
Djamaluddin et al. [18]	2018	Deception and MTD	Multiple web servers, applications, languages and databases	Fingerprinting and vulnerability scans
El-Kosairy and Azer [24]	2018	Deception using game theory	Honey web and tokens	Ransomware and intrusion detection
Musch et al. [22]	2018	Low interaction honeypot	Automatic network interaction	Login and file upload attacks
Fraunholz et al. [25]	2018	Deception as a service	Reverse Proxy	Common web attacks
Fraunholz and Schotten [19]	2018	Deceptive defense strategy	Fake banner replies & error codes, robot.txt generation, honey files	Website reconnaissance
Jia et al. [21]	2018	Micro-honeypot	Browser fingerprinting for attacker detection	Attacker identity gathering
Cernica and Popescu [23]	2018	High interaction	Web proxy	Vulnerabilities in Word press themes and plugins
De Faveri and Moreira [37]	2018	SPL based honeypot	Honey-words	Offline password attacks
Lin et al. [26]	2017	Web shadow service	Honey accounts, tokens	Persistent targeted attacks
Ishikawa and Sakurai [31]	2017	Web application deception proxy	Adding decoy functions to original web application	XSS & SQLi
Leaden et al. [36]	2017	API honeypot	honeypot disguised as REST API	DDoS & XSS
Djanali et al. [27]	2014	Low interaction	Likejacking technique	SQLi and XSS
Virvilis et al. [32]	2014	Low interaction	Honey tokens	Web scanning and fingerprinting
Koniaris et al. [34]	2013	High interaction	Kippo SSH honeypot [35]	SSH attacks
John et al. [30]	2011	Heat seeking honeypot	Dynamic generation and deployment of honey web pages	SQLi, XSS and password attacks
Müter et al. [46]	2008	High interaction	Transparent link advertising	Website reconnaissance & fingerprinting

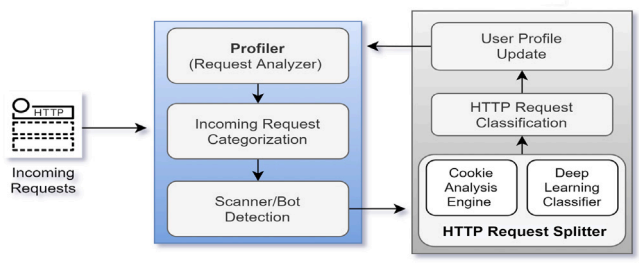

Fig. 4. Web attacks detection methodology.

Step-1. The incoming HTTP request is initially analyzed by the profiler which has been updated and maintained over a period of time. If the user profile associated with the incoming HTTP request is found out to be benign and a static page has been requested, the page will be opened without triggering either the deep learning classifier or the cookie analysis engine. Moreover, requests by malicious users will be blocked straight away (later forwarded to the deception system), while all other requests will be forwarded for further analysis. This request categorization is useful as it adds efficiency and reduces latency in the system.

Step-2. After the request has been assessed and profiled by the profiler, it is then forwarded to a bot and scanning detection engine which checks it with respect to time, requested URL and socket information of the request initiator. If a bot or scanner is detected, the request will be turned down and the user will be profiled malicious.

Step-3. After passing the bot/scanner detection module, the incoming request is split in a way that the cookie(s) parameter (session related data) is analyzed by the Cookie Analysis Engine while remaining HTTP parameters go to the deep learning classifier.

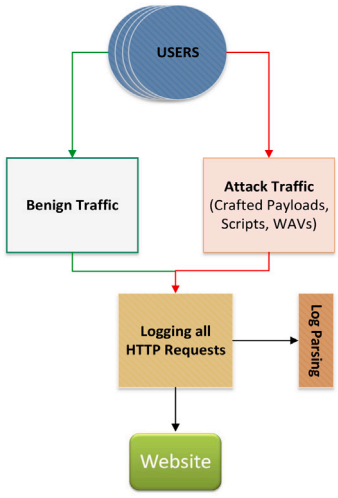

Fig. 5. Testing dataset generation process.

Step-4. The final decision about the incoming request is based on the judgment of CAE and deep learning classifier as discussed later in the section. Based on this decision, the user profile is updated accordingly.

Datasets. In the deep learning based classification module, we made use of a publicly available benchmark dataset [47] in a way that 60% of it was used for training, 20% for validation (dev-set) and the remaining 20% for testing purpose. This dataset contains a large number of benign (*119586*) and malicious HTTP requests (*104001*) covering a wide variety of web attacks listed by OWASP. The records in this dataset are in an extended form of HTTP/1.1 protocol. Many research works mentioned in Section 2 also use this benchmark dataset in order to train and test their attack detection models.

Table 2
HTTP features and their justification.

Features	Justification for selection
Request Type	This field is deliberately modified by attackers to request some resource by tampering HTTP verbs. For example, the OPTIONS Request Type would let an attacker analyze the response headers to find out all supported HTTP request methods.
Accept Header	Attackers often manipulate this field to check for security mis-configurations on directory files. It is used in context with URL and Request Type headers.
Encoding	This field is helpful in finding out Attacker's impersonation of an authorized user's HTTP request because the website already knows about all information about user preferences.
Language	Like encoding this field is also related to user's preferences and any change or abnormality can be noticed if analyzed properly.
User Agent	This feature is pretty helpful as WAVS related details can be obtained, if not forged. Moreover, website security mechanism uses this to find any difference in preferences.
Cookie (Data)	Cookies contain user session information and authentication levels and user preferences. Attackers modify/forge/inject cookies in order to hijack sessions and perform other attacks.
Content Length	This field helps in finding out attackers who send multiple POST HTTP requests to web forms (using WAVS or otherwise) having differing lengths. It is also used in detecting HTTP Request smuggling attack.
URL	This parameter is vital when linked with user's profile and authorization levels. It helps in detecting directory brute-forcing, DoS and DDoS attacks. When looked upon in conjunction with other features, it becomes very useful in detecting attack traffic.

In order to further test and analyze the performance of the proposed classifier, we also generated our own dataset of web attacks by attacking a vulnerable PHP/MySQL web application with different OWASP top attacks comprising of custom scripts and payloads along with their permutations. The dataset generation process is further elaborated in Fig. 5. The generated dataset had a healthy combination of top web application attacks making it suitable for testing the trained classifier's performance. Since the vulnerable target web application had all features of a normal website, it was therefore accessed over a period of time with normal traffic to generate the benign testing dataset.

Features for classifier training. Selection of features is an important step in training the deep learning classifier. While dealing with web application attacks, fields of an HTTP request appear to be pretty interrelated. Any single feature or a combination of multiple features is not alone helpful in determining the type and intent of the request. For instance, many rule based analysis applications would simply term directory brute-forcing requests as benign. Also, IP addresses are not allowed in user submissions, and if required by the website, such request will be termed as malicious, despite being benign. Experienced attackers can easily misuse most fields in the HTTP request in order to analyze the associated response that helps in revealing useful data to them. There are no meta-features used in training the classifier.

Therefore, determining as to which particular HTTP request parameter, or which parameter combination helps more in labeling an incoming HTTP request is an arduous task. Since there are numerous types of web attacks, therefore, detecting them needs thorough analysis of useful fields in the incoming HTTP request. The used features for training the deep learning classifier along with the justification for their selection is mentioned in Table 2.

Feature transformation. Once necessary features have been obtained from the HTTP request, they need to be transformed from categorical to numerical form. We have transformed all categorical features except the cookie(s), as they are to be analyzed by the Cookie Analysis Engine. The remaining features will be used to train the deep learning classifier. We have used the *sklearn pre-processing* label encoder which uses the *Fit-Transform* method for converting categorical features to integers ranging from *0* to *total.features-1*. This is helpful because we are using these features to train a web attack detection model which is further being used in the deception system. Table 3 further elaborates how the selected feature transformation function converts categorical values to numeric ones.

Cookie analysis engine. The Cookie Analysis Engine profiles malicious users by analyzing cookie(s) field in the incoming HTTP request. This analysis encompasses finding the presence of advertising content, scripts, payloads, jargon and content that fails sanitization and validation checks. Attackers generally put such content in he cookie field to analyze the website's response or to extract information. If the analysis finds out no such anomaly, the cookie(s) is marked benign. Failure to meet integrity and sanitization checks will mark an incoming cookie(s) as malicious. Table 4 describes the functionality of CAE and how it categorizes users as malicious (**M**), suspicious (**S**) or benign (**B**) based on different cookie mutations, presence of third party cookies, integrity failures and sanitization issues.

Table 3
Feature transformation using Fit_Transform()

Features	Numerical value
Request Type	3
Accept Header	0
Encoding	1
Language	2
User Agent	5
URL	4

Deep learning classifier. One dimensional CNN was used because HTTP request logs are represented by a single sequence of multiple features. Different layers of the convolutional neural network with their parameters are explained in Fig. 6. After setting up all the layers, the model with all its parameters was compiled. As far as tuning the hyper parameters is concerned, empirical search along with using the validation dataset helped in arriving at optimal values/set. Later, the model was evaluated on the testing dataset and showed very high accuracy and outperformed all traditional classifiers and ModSecurity [48], a popular open-source WAF. The deep learning model was trained on Intel(R) Core(TM) i7-7500U CPU @ 2.70 GHz 2.90 GHz and 16 GB RAM.

Once an incoming web request is analyzed by the deep learning classifier and CAE, a final decision will be made where the verdict of deep learning classifier has more significance.

- An incoming request is marked benign, only when both the CAE and deep learning classifier label it as benign.
- A request is marked suspicious only when the deep learning module labels it as benign while the CAE says its suspicious
- The request is marked as malicious if the deep learning classifier labels it as such irrespective of the CAE's decision

Rationale for optimization. This detailed analysis of cookie(s) and other HTTP request parameters by the CAE and deep learning classifier respectively optimizes the framework performance because attacker profiles are being maintained with the help of their detection, as highlighted in Fig. 4. For any new incoming request, the profiler will first check whether the malicious profile exists against that cookie. If found, the incoming request will not be forwarded to the website but to the deception system without bothering the deep learning classifier. Those requests pass through the deep learning classifier whose respective profiles do not exist. The classifier along with CAE will analyze such requests and update the profiler accordingly. Therefore, all incoming

Table 4
CAE's decisions on different cookie mutations.

Third party cookie(s) found	No	No	No	Yes	Yes	Yes	Yes	Yes	Yes
Sanitization checks ↑	–	–	–	Yes	Yes	Yes	No	No	No
Local cookies(s) integrity check	No	No	Yes	Yes	No	No	No	No	Yes
Sanitization checks ↑	Yes	No	–	–	Yes	No	No	Yes	–
CAE decision	S	M	B	S	S	M	M	M	M

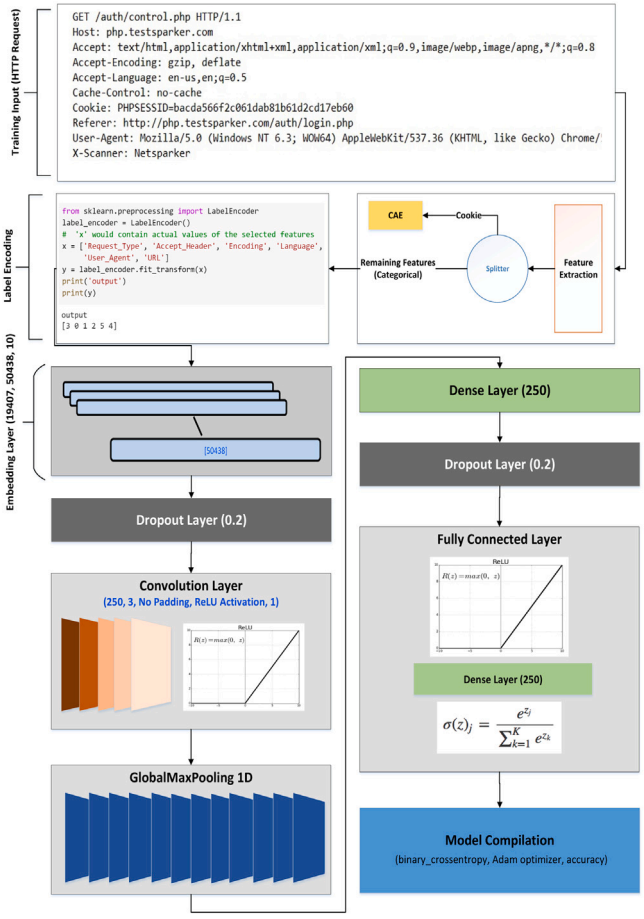

Fig. 6. 1D CNN model in detail.

requests do not necessarily pass through the deep learning classifier which saves useful processing time and optimizes the overall deception process by making it quicker. This hybrid web attacks detection module also enhances productivity and effectiveness of the proposed deception framework because deception.

3.2. Docker based customized web deception framework

The deep learning based web attack detection module along with the Cookie Analysis Engine will divert attack traffic towards the deception system whose prime responsibility is to engage the attacker based on his profile by providing an environment that looks normal. This module will also extract attack related information and eventually safeguard the actual website. In order to develop any deception system, efficiency, robustness and agility are some quality attributes that must be accounted for. Therefore, docker based architecture has been chosen to develop the proposed framework as it provides the required architectural simplicity and operational ease which is vital for any deception framework that emulates an actual system.

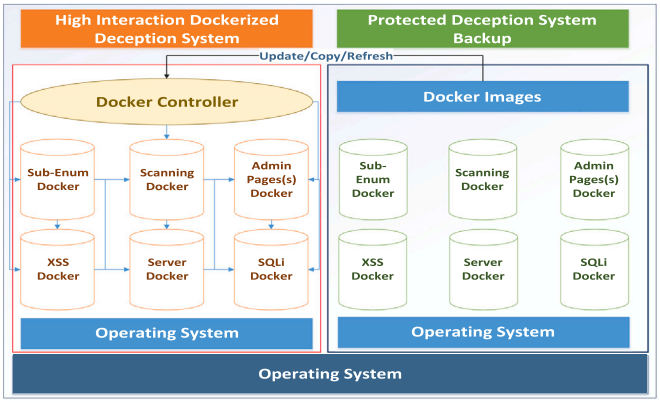

Fig. 7. Docker containers in the deception module.

Using the docker approach. Dockers are commonly used as they have a lot of performance and operational benefits over virtual machines. Web applications are based on a request and response system where users(s) generate an HTTP request for a particular web page which is served with an HTTP response. In case the user generates a malicious HTTP request, a deception system should send an immediate and well-suited response. This can be achieved by adopting an approach which is efficient, modular, agile, portable and scalable, thereby minimizing latency. Dockers in comparison to virtual machines provide all of them because they share the host operating system. Moreover, they are memory efficient, can easily be deployed and have less boot time. The selection of this approach is more important in deception system because the attacker profiling feature enables us to provide customized deception which requires easy runtime development. Dockers can provide this service without much of a problem.

One observation against the use of dockers are their security issues because of the shared host kernel and Operating System (OS). An OS level issue in one docker would affect all dockers sharing the same OS. This phenomenon is serious when dealing with attacks which actually affect the underlying OS like malicious code, ransomware and kernel level hooks etc. Generally, Remote Code Execution (RCE) attacks affect the underlying OS. Majority of web attacks like SQL injection, XSS, CSRF, XXE, cookie manipulation and session hijacking etc. reside entirely on the application layer throughout their execution and do not interfere with the underlying OS. Therefore, OS manipulation or corruption is unlikely which means that the docker approach is safe for building a web deception system. The composition of dockers in the deception system is shown in Fig. 7, where the docker controller manages all dockers and fetches securely placed docker images if and when required. The controller is highly safeguarded and is behind the secure curtain of hybrid web attack detection module.

Docker security. While developing the deception system, the security of dockers must be held paramount so that any infiltrator who has been routed towards any deceptive docker does not subvert its security in order to damage the system. For that purpose, following security measures have been adopted.

1. All dockers are run as a non-root user because by default the container is run as Root *UID 0*

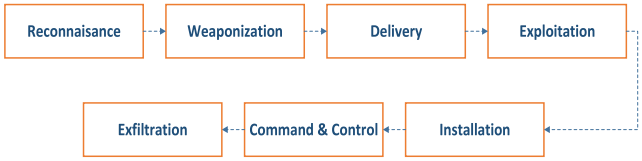

Fig. 8. Cyber kill chain.

2. Only necessary services and packages have been installed in all dockers, e.g. SQL docker only has necessary services to emulate an SQL injection attack while the sub-domain enumeration docker contain dummy sub-domains
3. Regular scanning of all dockers against system level vulnerabilities other than the ones placed intentionally
4. Docker Controller uses Docker Content Trust to maintain the integrity of the docker images which are pulled from registries
5. No confidential data which holds value for the website or respective organization has been placed in the docker
6. To enforce more security and thwart privilege escalation attacks by malicious attackers, the *setuid* and *setgid* permissions have been removed from the docker binaries
7. The dockers are carefully updated and the *HealthCheck* utility is also regularly used to check their proper functioning

3.3. Preparing dockers for pre-attack phase

Information gathering and reconnaissance is the very first step attackers adopt before launching any attack. It includes sub-domain finding, port scanning, finding the running services, virtual host discovery, banner grabbing of the server, knowing about used frameworks, plugins, back-end databases and cookie analysis etc. According to the cyber-kill chain [49], as mentioned in Fig. 8, this first step governs the attack type, methodology and attack motives.

Dedicated dockers have been developed which deal with all such reconnaissance and information gathering activities launched by attackers during the pre-attack phase.

Port security docker. Attackers are deeply interested in finding open ports and services running on those ports. In order to guarantee security of the main admin page, we have fixed it on an unknown port which is only available for legitimate connections after port knocking. The admin will perform port knocking on a sequence of ports before an IPTables command opens the admin page on the specified hidden port. This port would not appear in scanning as common network scanners do not perform port knocking or scan unknown ports generally. Moreover, a fake admin page is running on the common HTTPS port to lure the attackers who have made use of crawlers and scanners to get to this page. Attackers demanding this page will be logged by the hybrid attack detection module and docker controller and they will be provided with a dummy admin page which might or might not carry any vulnerabilities. The port security docker contains few open ports where different vulnerable pages are running. Attackers trying to connect through those ports will be logged, engaged and routed to other dockers as shown in Fig. 9.

The entire flow of how the port security docker works after a port scan by attacker is detected is elaborated in Algorithm 1. This docker will divert attacker(s) to other attack dockers (like XSS, SQL etc.) depending upon the type of exploit they bring. It holds significant importance because majority of attackers perform scanning before launching attacks.

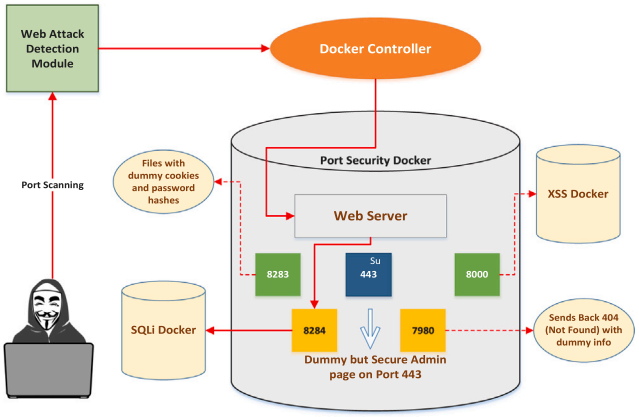

Fig. 9. Port security docker.

Algorithm 1 Deceiving Port Scanning Attackers

Input: Port Scan Request ($Packet_{port\_scan}$)
Define: ✓: Exists, x: Does not exist

if $Packet_{port\_scan}$ detected **then**
  if profile(✓) **then**
    $Divert(docker_{controller})$
    $Restore_{session}$
  end if
  if profile(x) **then**
    $Profiler(update)$
    $Divert(docker_{portsecurity})$
    $Display(Page_{vulnerable})$
    if Exploitation(XSS) **then**
      $Divert(docker_{XSS})$
      if Docker(x) **then**
        $spawn(docker_{XSS})$
      end if
    end if
    if Exploitation(SQL) **then**
      $Divert(docker_{SQL})$
      if Docker(x) **then**
        $spawn(docker_{SQL})$
      end if
    end if
  end if
end if

Sub-domain enumeration docker. Attackers use certain tools or WAVs for finding sub-domains of the target website. Such probes are detected by hybrid web attack detection module and are routed to the deception module using *IPTables* rules. The sub-domain enumeration docker contains few dummy sub-domains based on the common word-list used by crawlers and WAVs. This docker gives either a *Page Found 200 OK* or *Page Found 302 URL Re-direction* response on few requested sub-domains. These pages displayed to the attacker contain SQL injection, XSS bugs and other vulnerabilities and the docker will also be able to communicate with other attack dockers as shown in Fig. 10.

Honey information for banner grabbing. Banner grabbing allows the attacker to extract maximum information about the server including web framework, libraries, plugins being used, error pages and detection of the content management system. Moreover, this entire process of banner grabbing cannot be categorized as malicious because normal users can also make use of online utilities, tools and plugins to extract this information from the website. In order to counter information

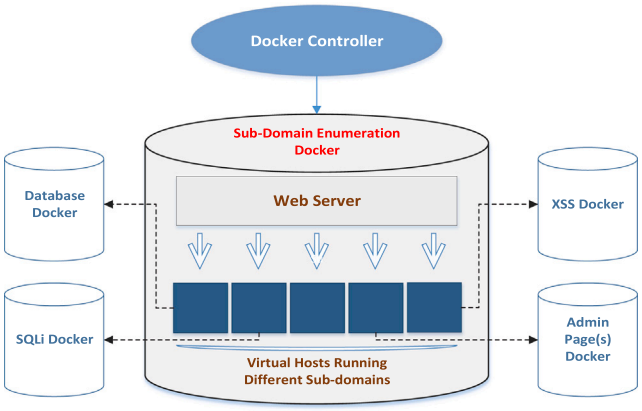

Fig. 10. Sub-domain enumeration docker.

disclosure through banner grabbing, following changes have been made to the actual website.

1. HTTP configuration file (*httpd.conf*) on the server has been amended to contain downgraded (vulnerable) server version with known CVE (Common Vulnerabilities and Exposure) number.
2. Wrong framework information has been provided to deceive the attacker by making necessary changes in the *wp-content* file.
3. All plugins which reveal actual framework version have also been changed in order to confuse the attacker.
4. Since most utilities try finding some *svg*, *PNG* or other related files to link plugins with the respective framework version, therefore they have also been changed to something misleading.
5. Attackers also check and analyze the JavaScript files along with the web application's code to find the libraries being used. In order to deceive the attacker, misleading information has been added.
6. Error pages and error information is yet another source of vital information for the attacker which helps in knowing weaknesses of the website and its connection with the backend database. Since all configuration files and related information has been changed, therefore, error page will reveal false and deceiving information to the attacker which will further lure him up.

Honey information in cookie field. The hybrid web attack detection module differentiate attackers from normal users and performs their profiling using the CAE. Since the framework and server information has been downgraded therefore that misleading information is written to the cookie after the attacker is detected. In future, once any user bringing a cookie with that information will straightaway be categorized as attacker and routed to the deception system.

3.4. Preparing common attack dockers

After adding deceiving information to lure the attacker in pre-attack phase, specific attack dockers have been prepared. These attack dockers are not just linked with their pre-attack counterparts but with other dockers as well. These dockers have multiple lures which help in engaging the attacker up to multiple levels by providing customized lures dynamically. The profiling of attackers performed in the CAE significantly enhances the performance of these attack dockers.

SQL injection docker. Much like the XSS docker, the SQL injection docker is also connected to multiple other dockers, especially the SDE (Sensitive data exposure) docker, database docker and other information gathering ones because attacker will first perform reconnaissance, sub-domain enumeration and scanning in order to finally perform SQL

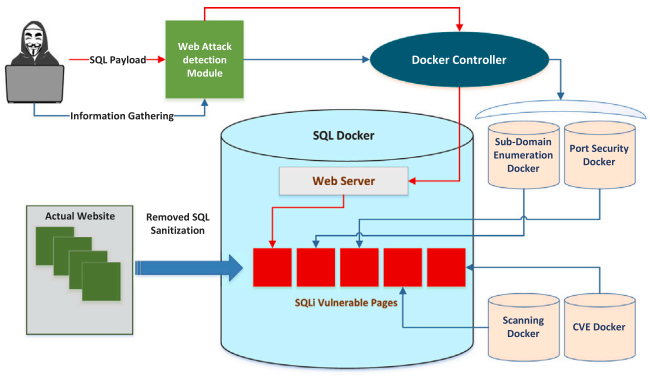

Fig. 11. SQL injection attack docker.

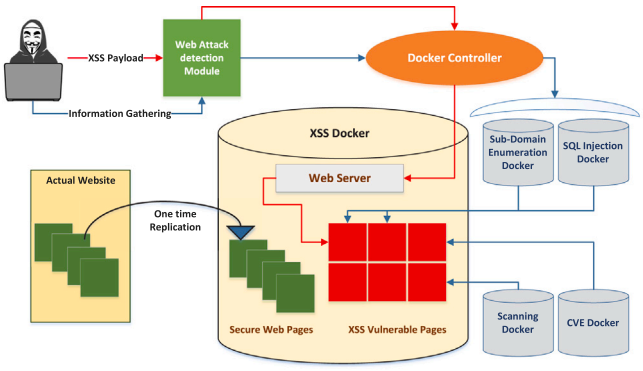

Fig. 12. Cross site scripting docker.

injection attack. Comparative analysis by a smart attacker would be tough in case of SQL injection attacks because as soon the attacker supplies any SQL payload, he will be directed towards the deception system which will let him achieve the desired results. Therefore, we have replicated few *authentication* pages from original website by removing SQL sanitization checks. These included removing parametrized queries and validation checks and not escaping the user supplied input. Fig. 11 shows the docker's composition and algorithm 2 demonstrates how an attacker is engaged.

Cross site scripting docker. Cross Site Scripting attacks are one of the most common types of web attacks. Only those web pages which require user input are susceptible to XSS attacks therefore attackers insert malicious JavaScript which when executed results in website compromise. Before working on the XSS docker, the original website was patched against all XSS vulnerabilities. Attackers perform a lot of information gathering and enumeration before bombarding a target website with XSS payloads. Therefore, it is not just the hybrid web attack detection module which would divert an XSS probe towards this docker but the pre-attack dockers carrying XSS bugs will also divert the attacker here as shown in Fig. 12. The attacker can then perform *stored* or *reflected* XSS attack depending upon the vulnerability present. Another important thing to note here is that if the attacker somehow successfully launches a *Zero Day* attack on replicas of secure pages within the docker, that will be detected as the attacker is already profiled. This would help us in patching actual website.

While dealing with XSS attacks, we have to be aware of the smart attackers who can perform a detailed comparative analysis for finding whether a deceptive system is in place. For that purpose we adopted a two-pronged strategy.

1. Few web pages from the actual website where user data can be supplied were replicated. They were not made vulnerable

(by removing XSS sanitization and security checks) because any attacker who performs comparative analysis by using a totally different identity in parallel will be able to find the deceptive system.

2. Some new XSS vulnerable pages with no sanitization and security checks were created in the this docker. If any attacker shares link of these pages with a trusted and legitimate website user through phishing, the system will not serve that user mainly because the user has been compromised. Whereas, if that link has been shared with any random user visiting the website, that user will also be directly diverted to the deception system because the detection module and docker controller know about the malicious nature of this link.

Algorithm 2 Deceiving SQL Attackers

Input: Packet ($SQLi_{payload}$)
Define: x: Does not exist, r: total user requests
initialize: i ← 0

if $Packet_{SQLi}$ detected **then**
  if profile(x) **then**
    $Profiler(update)$
    $Check(payload)$
    if SQLi($Payload_{SDE}$) **then**
      $Divert(docker_{SDE})$
      if Docker(x) **then**
        $spawn(docker_{SDE})$
        $provide(data_{sensitive})$
      end if
    end if
    if SQLi($Payload_{HI}$) **then**
      $provide(data_{HI})$
      $provide(cookies_{honey})$
    end if
    if Attack($_{SQLi}$) **then**
      while i < r **do**
        $attacker\_engage()$)
        $provide(dummy\_values_{DB})$
        i ← i+1;
      end while
    end if
  end if
end if

CVE attack dockers. The actual web server, framework, libraries and error response information for the original website has already been changed (downgraded) so that any person with a malicious intent who wants to exploit vulnerabilities in these downgraded versions is lured in. Therefore, we have created a specific docker which runs the vulnerable web server and framework version with known CVE vulnerabilities. WAVs and scanning tools will reveal these CVE vulnerabilities to the attacker. Once launched, the attacker will be engaged in this docker and further routed to other dockers up to certain levels. The docker controller can even spawn a docker at runtime depending on the attacker's probe.

3.5. Prominent attacker engagement scenarios

The composition of pre-attack and attack dockers has enabled the proposed web deception system to lure attackers in a wide variety of ways with high engagement levels without noticing about the presence of the deception system. Moreover, the profiling feature helps in customized deception which further enhances the system's deceptive capability.

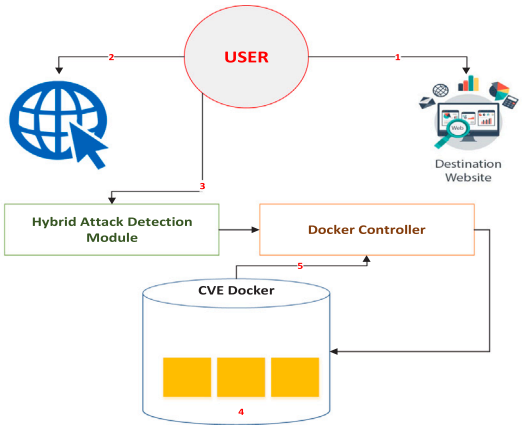

Fig. 13. Attacker reaching CVE docker through reconnaissance.

Table 5
Steps that lead to attacker launching CVE exploits.

Steps	Actions performed
1	User extracts server, framework and libraries info of the website
2	Gets CVE exploits for those (downgraded) versions from the internet
3	Attack launched. Detected and diverted to the CVE docker
4	CVE docker has vulnerable server & framework instances
5	Exploits succeed and response sent. User believes he is successful

Information gathering leading to CVE attacks. Since dummy and downgraded information has already been placed in order to lure the attacker, therefore, the attacker will search for existing CVE exploits for those downgraded versions in the pre-attack phase. Publicly available CVE payloads will help the attacker in creating the malicious HTTP request even without the assistance of WAVs. Once launched, the hybrid web attack detection will detect these malicious HTTP probes and forward them to the web deception module as elaborated in Table 5 and shown further in Fig. 13.

Engagement through scanning and enumeration. Scanning and enumeration are part of the information gathering phase and are among the first probes launched by the attackers. Since attackers are being profiled, therefore, it is easier to divert users performing scanning and enumeration to respective dockers via the docker controller. These dockers have lures and dummy information that will engage the attackers using different engagement techniques as highlighted in Fig. 14. Moreover, useful attacker specific information will be obtained through these deceptive lures which would help further secure the original system.

SQL injection attacks. SQL injection attack is the most common web attack as it exploits the lack of input validation and sanitization. SQL docker is activated when a user supplies malicious payload intending to exploit the backend database. The hybrid web attack detection module forwards SQL injection payloads to the deception system where the docker controller routes them to the SQL docker that is responsible for fetching dummy table, column and field values from the database docker as shown in Fig. 15. The controller ascertains existing load on the docker and its availability for the specified URL so that it can instantly spawn a new docker. Handling SQL attacks like this would help in finding most widely used SQL payloads and attack approaches which would ultimately help in further refining the actual website. This docker is connected to almost all pre-attack dockers and few other attack dockers in order to enhance attacker engagement.

Other web attacks. The proposed web deception system composed of inter-connected dockers is also capable of successfully absorbing other common web attacks with high levels of attacker engagement as mentioned in Table 6.

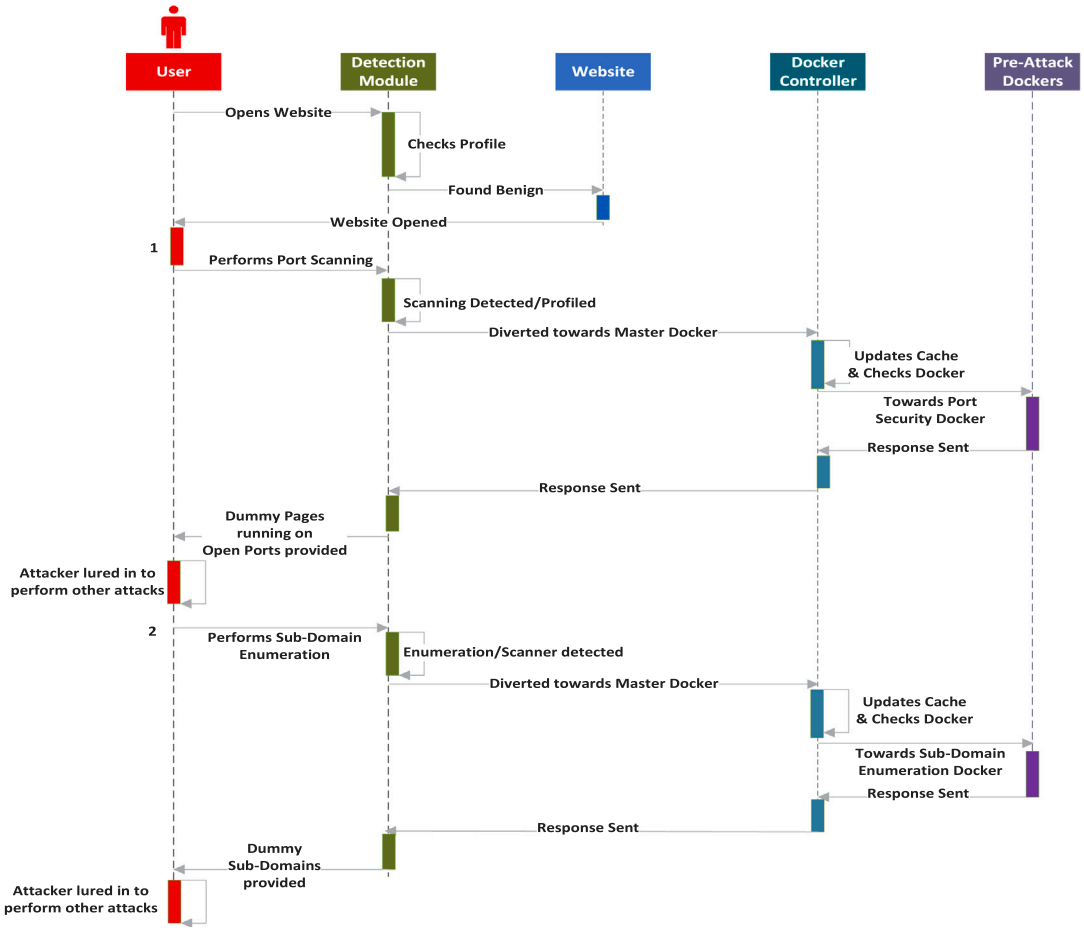

Fig. 14. Scanning and enumeration attack flow.

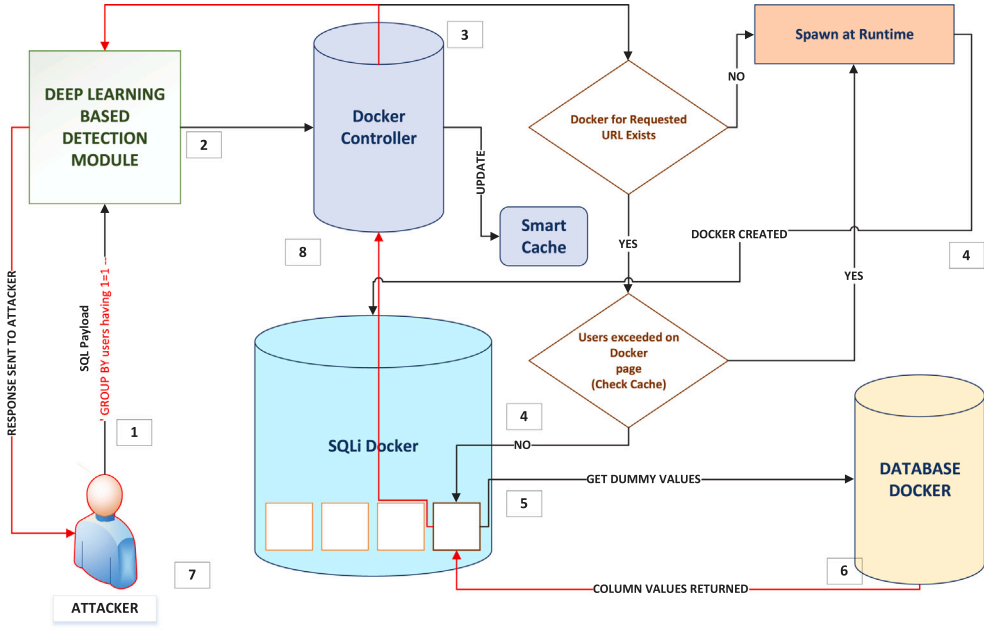

Fig. 15. SQL attack workflow.

4. Discussion, performance analysis and results

The proposed Web Deception system which is coupled with a hybrid web attack detection module was thoroughly analyzed and tested on real time data in order to validate the efficiency, concealment and robustness of the proposed framework. The proposed framework has the ability to not just detect web attacks but also helps in learning the attacker behavior, gather maximum information about the attackers

Table 6
Tackling of other common web attacks by proposed deception system.

Attacks	Attack handling by proposed Web Deception System
Unauthorized sessions	Attacker will grab honey cookies from the pre-attack dockers during information gathering. Attacker will be presented with fake privileged content after he sends the request with these honey cookies.
Security misconfiguration	Attacker will fall prey to deliberate misconfigurations and vulnerabilities that have been introduced in the main website along with respective dockers in the deception system which are configured to deal with such attacks.
Sensitive data exposure	Dummy sensitive data like hashes of passwords, database entries, authentication and logging information has been placed for smart attackers who will be able to find that after quite a guess game. Attackers who find and try using that data for login, security bypass and system breach will be lured in and engaged further with the help of other attack dockers.
Directory bruteforcing	Scanning based attacks using wordlists to bruteforce hosts, directories and sub domains will be entertained by the system by smart caching and spawning relevant dockers at runtime.

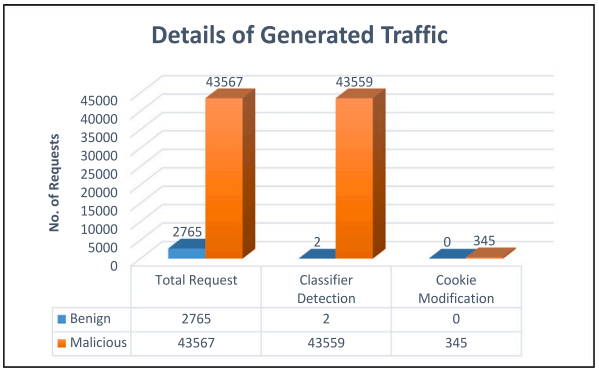

Fig. 16. Detail of generated traffic.

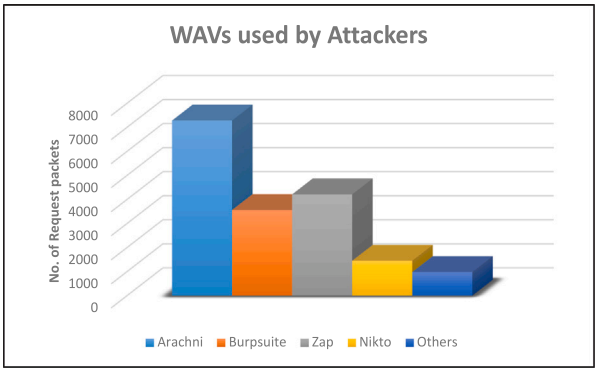

Fig. 17. WAVS usage by $\mathcal{T}_r$.

along with their attack methodologies and threat vectors, waste attacker's time through high interaction engagement and finally helps in improving the security posture of the actual.

4.1. Testing environment

The testing website was PHP based and was running on an Apache web server whereas another server was running the docker controller and relevant attack specific dockers. The testing website comprised of both static and dynamic pages. There were few authentication pages that were linked to the back-end database, few pages where users were allowed to upload specific files (preferably document and image files) and a few pages which had hyperlinks to other web pages. Before proceeding with the actual real-time testing, we added some honey information in the configuration details of the actual website. This honey information included downgraded (and vulnerable) version of the web server, wrong framework information, misleading JavaScript information along with deceiving error information. As discussed earlier, the purpose of this honey information is to lure in users with malicious intent.

In order to generate both benign and malicious traffic, we used the red teaming and blue teaming approach with 50 users having distinct IP addresses where 20 users were part of the blue team ($\mathcal{T}_b$) and the remaining 30 were part of the red team ($\mathcal{T}_r$). Web application vulnerability scanners (WAVS) and web browsers were used to generate web attacks. We then exposed the website with hidden deception system to these two teams having distinct IP addresses for a period of one month and analyzed the traffic, attack patterns, their engagement with the proposed deception system and their efforts to find the deceptive lures. Fig. 16 shows the total number of benign and malicious HTTP request packets being generated by $\mathcal{T}_b$ and $\mathcal{T}_r$ along with their detection by the hybrid web attack detection module. It is evident that cookies were only modified by $\mathcal{T}_r$.

Moreover, since attackers often rely on Web Application Vulnerability Scanners (WAVs) to scan a website for vulnerabilities, therefore, many of the requests by $\mathcal{T}_r$ were generated by these scanners as mentioned in Fig. 17.

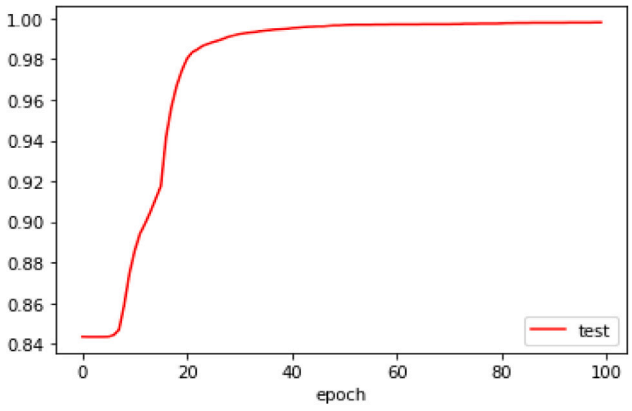

Fig. 18. Validation accuracy of the classifier.

4.2. Performance of the hybrid attack detection framework

The performance of proposed hybrid attack detection framework is very critical to the performance and efficiency of the web deception system. Since the attack detection framework comprises of the deep learning classifier and the Cookie Analysis engine, we have evaluated their performance both individually and collectively. The framework was implemented using Python.

Performance of deep learning classifier. The deep learning model was trained on the 60% training dataset and validated on the 20% validation dataset (dev-set). The validation accuracy and loss are elaborated in Figs. 18 and 19. After each epoch, hyper-parameters of the classifier were empirically tuned based on validation accuracy in order to get optimal values. The best model was subsequently tested on the remaining 20% previously unseen test dataset. The proposed classifier produced a testing accuracy of 99.94% which reflects that the model is performing pretty well and is not over-fitting.

In order to further evaluate the performance of the proposed classifier, we also tested it on a completely unknown private dataset

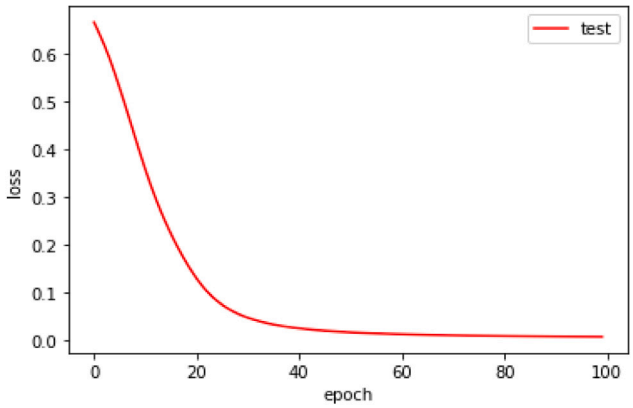

Fig. 19. Validation loss of the classifier.

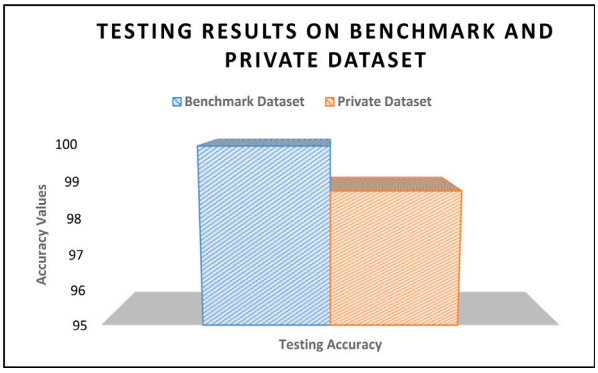

Fig. 20. Testing results comparison.

we generated, having same input features, as mentioned earlier. The proposed classifier gave an accuracy of 98.74% which shows that it is suitable for detecting web application attacks when deployed in real-time. The comparison of testing performance on benchmark dataset and private dataset is further highlighted in Fig. 20.

Performance of cookie analysis engine. Analyzing the performance of Cookie Analysis Engine was fundamentally important as it is directly linked to maintaining attacker sessions during the deception phase. It is the analysis of cookies which helps the proposed system maintain attacker profiles and provide customized deception. The CAE can only analyze cookie fields as the remaining fields are analyzed by the deep learning classifier, therefore, it would only categorize a cookie as either benign, suspicious or malicious, apart from using them for profiling. While analyzing cookies in the benchmark dataset [47], it was observed that most of the malicious requests had normal benign cookie fields. This shows that attackers actually prefer other HTTP fields to place/insert their malicious script or crafted payload. That is the prime reason we need the deep learning classifier. The combination of these two modules result in efficient attack detection along with successful attacker profiling as these two goals are fundamentally important in building an efficient web deception system. Moreover, the decision of the deep learning classifier about an incoming HTTP request is dominant as shown in Table 7.

The private testing dataset we created also made use of Web Application Vulnerability Scanners (WAVS) to generate malicious traffic. Since these WAVS do target the website with malicious cookies, therefore, this dataset had relatively larger number of malicious cookie as compared to the benchmark dataset. But again, the majority of malicious requests had their malicious scripts and payloads placed in other HTTP fields which were to be analyzed by the classifier.

Table 7
CAE's working.

CAE decision	Classifier decision	Final decision
Benign	Benign	Benign
Suspicious	Benign	Suspicious
Malicious	Benign	Malicious
Benign	Malicious	Malicious
Suspicious	Malicious	Malicious
Malicious	Malicious	Malicious

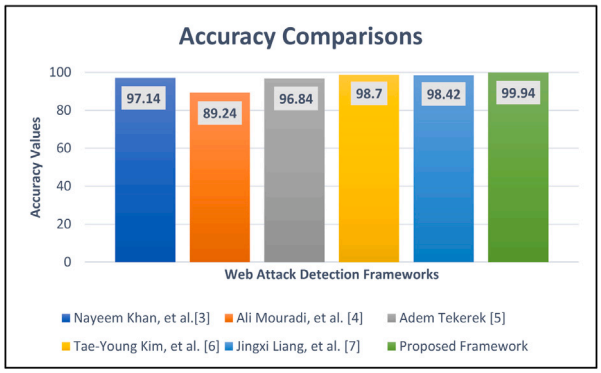

Fig. 21. Comparison of proposed framework with related works.

Nevertheless, the CAE detected malicious cookies with 99.3% accuracy in this dataset.

4.3. Comparative analysis of the attack detection framework

In terms of detection accuracy, the proposed web attack detection framework outperforms other research works who have used the same benchmark dataset [47] as shown in Fig. 21. This is because the proposed approach performs the key functionality of attacker profiling making it more suitable, feasible and optimal in real time environments also making it less resource intensive as compared to other solutions for detection web application attacks.

The attacker profiling feature which is performed through the Cookie Analysis Engine results in limited number of executions for the deep learning classifier. For testing, we launched 1000 HTTP requests (both benign and malicious) by five groups at regular intervals under different scenarios. Any other deep learning classifier would get triggered for each request no matter if it is by an attacker who has previously engaged or not. It was found out that in our case the deep learning classifier was only triggered 550 times resulting in just 55% engagement of the classifier without the system compromising on detection accuracy. For the remaining 450 requests, the CAE founds out that they were already profiled malicious and does not bother the deep learning classifier. This has been achieved with the help of key attacker profiling feature and makes the proposed detection framework way more convenient and less time consuming.

4.4. Performance benefits of using docker approach

Latency is one major drawback of using deception systems as they take more time in responding to users thereby raising suspicion. The proposed web deception framework has all the deceptive lures running in dockers managed by the docker controller making the system faster, robust and efficient. Moreover, attacker profiling helps in efficient and quicker handling of the attacker's request thereby significantly reducing latency as shown in Fig. 22 which is minutely different then response time of the actual website as shown in Fig. 23. This latency (or the packet response time) was calculated by analyzing the received traffic using a popular network analysis tool, Wireshark and a well known web

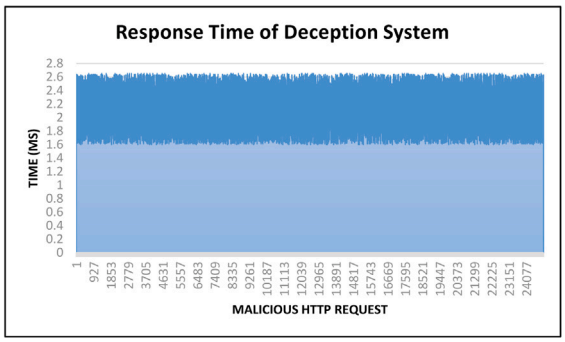

Fig. 22. Response time for anomalous HTTP requests.

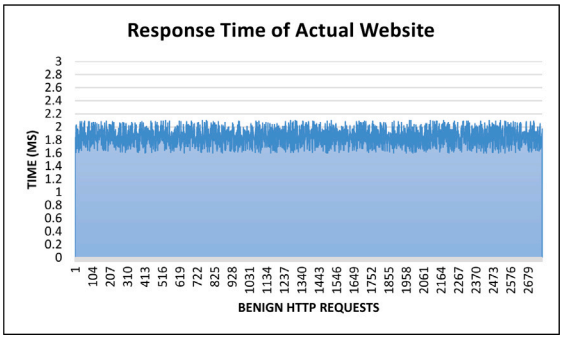

Fig. 23. Response time for benign HTTP requests.

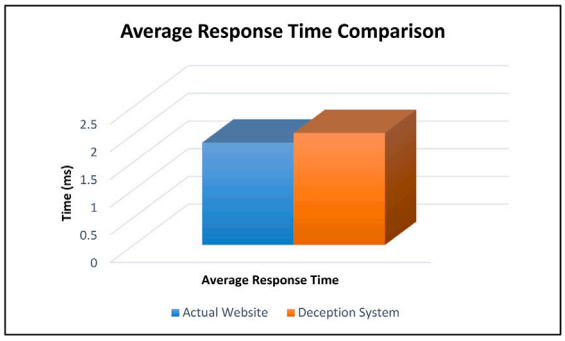

Fig. 24. Average response time (Benign vs. Malicious).

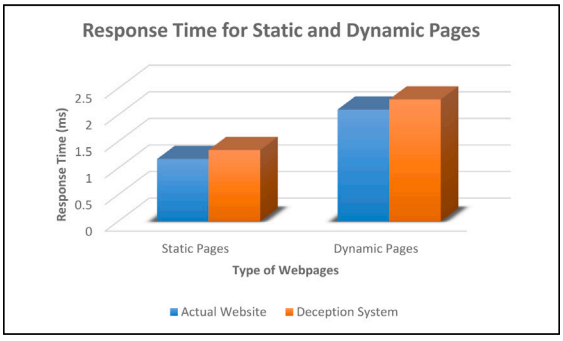

Fig. 25. Response time comparisons (Static vs. Dynamic).

proxy, Burpsuite [50]. In order to completely eradicate this difference, intentional delay can also be added to the response time of actual website.

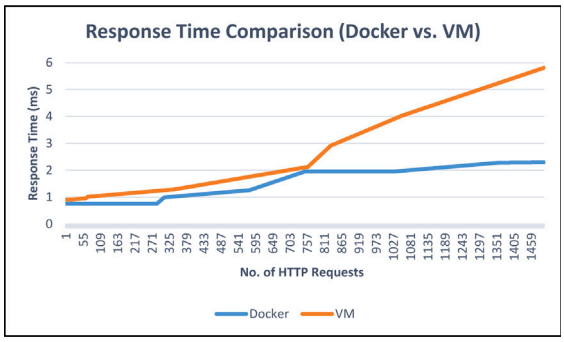

Fig. 26. Average response time (Docker vs. VM).

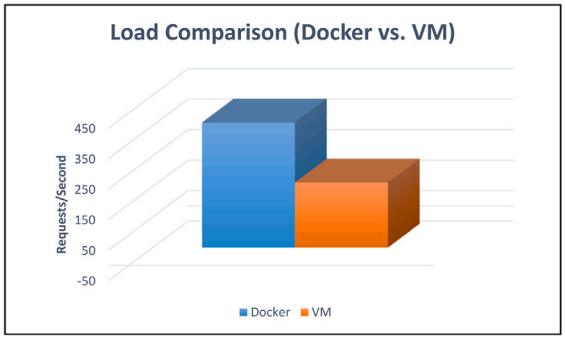

Fig. 27. Load comparison (Docker vs. VM).

Moreover, due to these optimization measures, difference of hardly a few fractions of a millisecond has been noticed in both response times as shown in Fig. 24. Response time difference in static and dynamic pages for both systems is shown in Fig. 25. This significant reduction in latency eradicate the chances for an attacker to know about the presence of deception system because legacy honeypots were slow in responding. As the proposed system is capable of providing customized and personalized deception to attackers depending on their past engagement history, significant reduction in latency was observed as the system knows what to offer.

In order to verify the choice of using dockers for building the deception system, we thoroughly evaluated our system's performance on both dockers and virtual machines with respect to latency, response time and throughput. Both systems were exposed to 1500 simultaneous HTTP requests and a detailed comparison was made on the basis of response time and load management as shown in Figs. 26 and 27. It was observed that web page response time in case of virtual machine exceeds abruptly when the number of HTTP requests increase overtime as it consumes more resources to respond to web request of any type. Apache benchmark tool was also used in order to find the number of HTTP requests which the system under examination can actually bear in a second. The docker based deception system was found to tolerate roughly two times more requests as compared to the virtual machine based deception system as it is light weight and optimized. Network latency plays a major role in performance decadence of virtual machines as compared to dockers who have better throughput and productivity thereby making them an optimal choice for constructing a highly interactive deception system of this sort.

4.5. Analyzing concealment of deception system

One of the fundamental requirements of any deception system is its ability to stay hidden from attackers for a long time. Attackers often employ various techniques, use specifically crafted scripts and payloads along with scanning to find hidden deception systems. Users first

Table 8
Success ratio of technology stack concealment.

Stats	Web server	CMS	JavaScipt	Database
Actual information	Apache httpd-2.4.47	PHP 7.4	JQuery 3.5.0	CentOS 8.1
Honey information	Apache httpd-2.4.25	PHP 7.0.1	JQuery 3.3.1	Ubuntu 18.04 LTS
Attempt	40%	27.3%	32.7%	29%
Concealment success	98.3%	99.01%	98.67%	97.3%

Table 9
Attacks and docker responses detail at t_0.

Docker	Requests	Information provided
Sub-domain enumeration	442	Fake and vulnerable sub domains
Port security docker	177	Fake admin page
Scanning docker	773	Dummy ports
CVE docker	39	Successful exploitation

try finding the technology stack for any website containing detailed information about Content Management System (CMS), framework information, e-Commerce platforms and Javascript libraries etc. This information helps the attacker launch targeted attacks. Since honey information about framework, libraries and web server etc. was added to deceive the attacker, it was expected that the same information will be extracted by either of the teams, $\mathcal{T}_r$ or $\mathcal{T}_b$, while trying to find about the website's technology stack. For that purpose, users used technology profilers like Wappalyzer [51], analyzed the error responses and analyzed the HTTP header.

Table 8 shows the success of technology stack concealment using honey information. It shows that very few users, generally less than 2% have figured out that honey information about technology stack has been placed but no one figured out the actual version information which the website under examination inhibited.

4.6. Measuring deceptive effectiveness through web attacks

Effectiveness of any deception system can be gauged by launching attacks and analyzing how well the attackers are lured in, engaged and dealt with. After finding the technology stack, attackers would naturally scan the website for open ports, vulnerabilities and CVEs at time t_0 as shown in Fig. 28, where a total of 1431 scanning attempts were launched on the target website by $\mathcal{T}_r$. All these scanning attempts/attacks were detected by the hybrid web attack detection module and routed to respective dockers via the controller and honey information was sent back to the attackers.

After analysis it was found out that attackers mainly chose NMAP and Hping3 for scanning while a few relied on custom scripts. In total 39 CVE attacks were launched at different intervals by the $\mathcal{T}_r$ to exploit a vulnerable Apache version. All these attacks were detected by the hybrid web attack detection model and routed to the CVE docker via the docker controller as shown in Fig. 29.

Detail of honey information provided to the attackers by the pre-attack dockers is mentioned in Table 9. This information lures the attacker to launch a variety of attacks. One key aspect to note here is that all users from $\mathcal{T}_r$ who launched these scanning attempts and probes have been profiled and subsequently routed to the deception system for the entire attack life cycle.

Based on the honey information provided in response to these distinct scanning probes, attackers launched a total of 822 various sorts of web attacks in the next phase, denoted as t_1. Since these users were already profiled, their HTTP requests carrying attack payloads were forwarded to the respective dockers which further lured these attackers by either forwarding their requests to other dockers or by handling them on their own. For instance, majority of the sub-domain enumeration requests were routed to vulnerable SQL pages in the SQL docker and few towards the XSS docker. Those wanting to get into the fake admin page were also routed accordingly. A detail of what

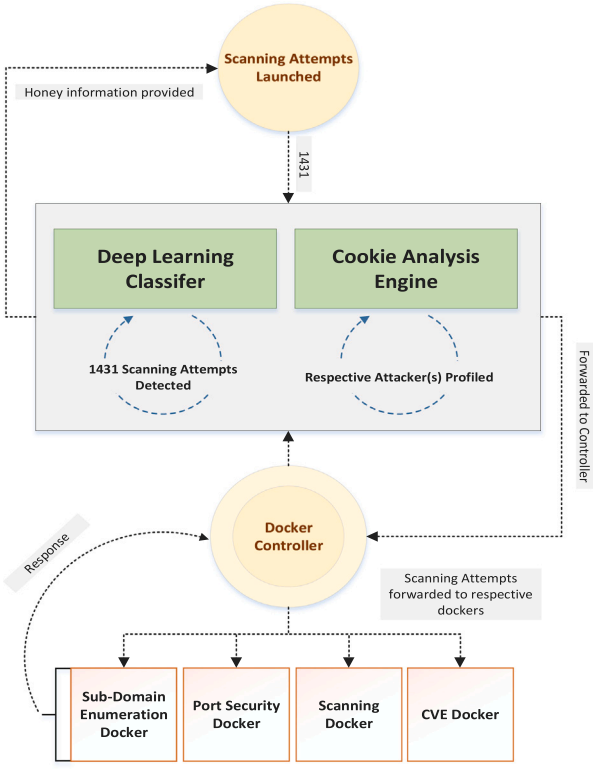

Fig. 28. Scanning attempts and their honey response at t_0.

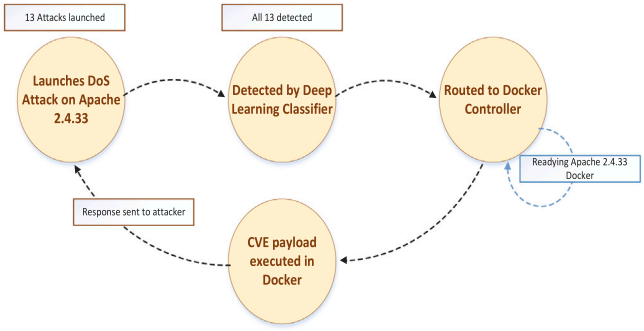

Fig. 29. Successful handling of the CVE attacks.

happened in the next two respective attacker engagement stages is mentioned in Fig. 30.

The proposed deception system engages attacker(s) to a maximum of ten levels. This means that the deceptive lures provided are able enough to guarantee attacker's presence even being on a high interaction level. Normally, the number of attackers do decrease with time as majority of them leave after performing limited number of attacks. Only a few users keep on trying a variety of techniques to further penetrate into the website and get trapped.

Moreover, there are some common attacks which are easier for the attackers to execute like the SQL injection attack. That is why the

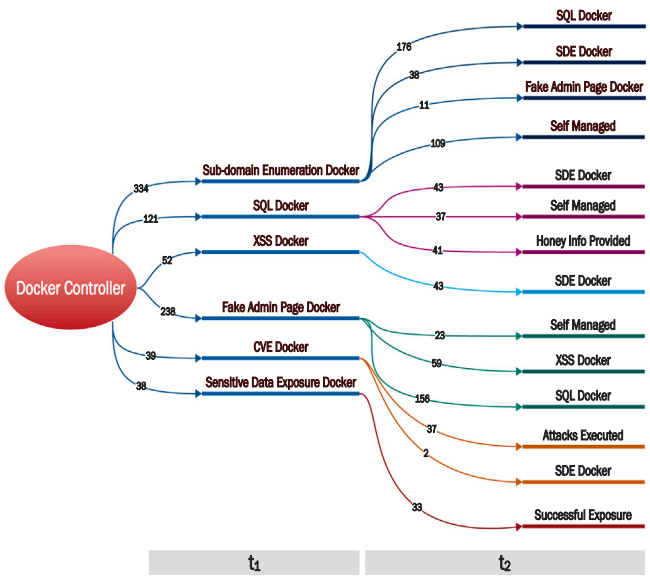

Fig. 30. Attack state of various dockers at t_1 and t_2.

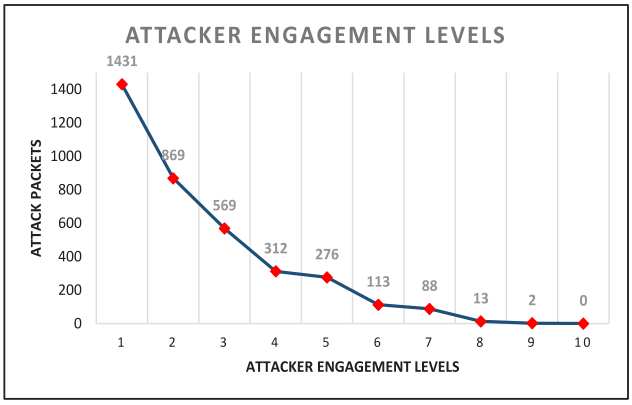

Fig. 31. Attacker engagement details from t_1 to t_{10}.

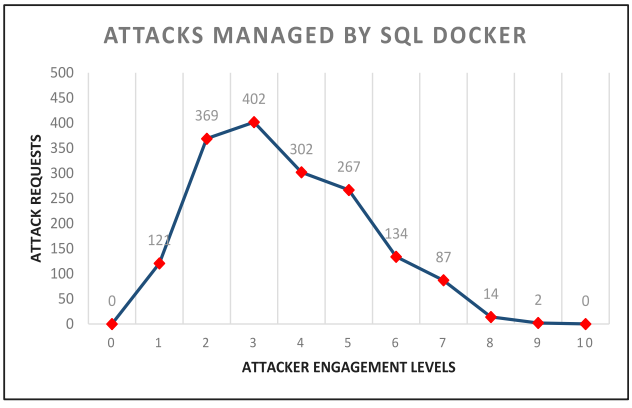

Fig. 32. Engagement on SQL docker from t_1 to t_{10}.

SQL docker remained the busiest during this entire testing exercise as mentioned in Fig. 32. The Sensitive Data Exposure docker was the least busy of all.

4.7. Comparative analysis

Providing customized and personalized deception to attackers is the prime feature of the proposed web deception framework. On that basis it is way more effective as compared to HoneyBug [13], because of a variety of reasons. Firstly, honeybug maintains user session through a session tracker by marking all requests using the session ID which can easily be corrupted by an experienced attacker. The request is then diverted to a rule based User Behavior Characterization module that ascertains the user's capability and motives. Honeybug borrows this ruleset of 58 rules from ModSecurity [14]. It has been discussed how weak ModSecurity is, when compared with any machine learning or deep learning based attack detection solution. The deceptive effectiveness of Honeybug has not been analyzed by following the red teaming approach where human attackers launch web attacks in real time. In contrast, the proposed framework detects the attacker and maintain profiles by passing the incoming web request through the hybrid attack detection model which is based on deep learning. Moreover, the proposed framework is faster as compared to Honeybug because of the optimized docker approach. The proposed web deception system was also compared with Glastopf [29], another widely used deceptive honeypot for web applications. The proposed deception framework was found to be more scalable, agile and robust as compared to Glastopf. Both solutions were compared on the basis of following parameters.

Session maintenance. Glastopf does not perform session maintenance as there is no session information and management in its raw deployment. Every new request is being handled differently because of this lacking. The proposed web deception system performs efficient session maintenance and management through attacker profiling. Moreover, the docker controller knows which requests have been forwarded to which docker along with maintaining user session. This helps in emulating specific attacks by analyzing the attacker's behavior and past activity.

Deceptive effectiveness. Glastopf has a default function which opens a new web page everytime which would easily be detected by the attacker in real-time. Moreover, it is good at deceiving WAVs but not humans who circumvent security by multiple levels of interaction and sophisticated, crafted payloads and malicious HTTP parameters. The proposed web deception framework is coupled with a hybrid web attack detection module which significantly enhances its deceptive effectiveness by effectively engaging human attackers in real time.

Architecture. Glastopf is sandbox based whereas the proposed framework is based on multiple high interaction dockers that are controlled by a docker controller. The controller routes the malicious HTTP requests being sent by the detection engine to respective dockers.

Runtime development. Glastopf is not suitable for making on the fly changes or any runtime development in its deceptive system/functionality whereas the proposed framework supports easy runtime development because the docker controller has the ability to spawn a new docker with negligible latency on per request basis.

Zero day engagement. Glastopf, or any other existing web honeypot or deception solution lacks the ability to engage attackers carrying Zero-day attack payloads. The attacker profiling feature enables the proposed deception framework to successfully engage attackers launching zero-day attacks and can subsequently patch the actual system (website) to be protected from that attack. This happens because in almost all cases attackers perform necessary scanning and reconnaissance before launching their zero-day payloads. They will be caught and profiled in the pre-attack phase thereby protecting systems from destructive zero days.

Scenario based emulation. Since Glastopf lacks scenario based emulation or interaction capability, therefore, the web application that is being protected has to be modified considerably to make the honeypot work. The proposed deception system can be implemented by little tweaking and necessary configurations by doing very little changes to the actual web application resulting easy deployment.

Table 10
Proposed web deception framework vs. Glastopf.

Functionality	Glastopf [29]	Proposed framework
Session maintenance	x	✓
Deceptive effectiveness	Low	High
Architecture	Single Unit	Feature specific
Runtime development	Difficult	Easy
Scenario based emulation	x	✓
Zero day engagement	x	✓
Centralized control	x	✓
Process utilization	High	Low
Framework modification	Difficult	Easy

Process utilization. The proposed web deception framework has vulnerability specific docker interaction where resources can be managed at the attack level for every docker. Therefore, it has a comparatively very low process utilization as compared to Glastopf in whose case effective resource utilization is difficult.

Framework modification. The proposed framework supports smooth framework modification and can easily be applied to web applications, Internet of Things [52] and other networks and information systems because it can easily be customized, is scalable and agile. Table 10 summarizes the comparison between Glastopf and the proposed web deception framework.

5. Conclusion and future work

Protecting web applications and their backened systems is of utmost importance keeping in view the rapidly evolving threat landscape. This research work proposes an efficient high interaction docker based web deception system which can cater for all major web application attacks by engaging the attacker in real-time. The proposed deception system is assisted by a deep learning based attack detection coupled with a cookie analysis engine which performs attacker profiling in realtime. This merging of deception with detection provides useful features like session management, deceptive effectiveness and support for dealing with almost all major web attacks. Moreover, the docker based approach used to build the deception system has resulted in a light-weight architecture, centralized control through docker controller, efficient process utilization and easy runtime development. The attacker profiling feature helps in catering for zero day attacks payloads as well which is a major contribution of the proposed framework.

The main task we intend doing in future is to expand the proposed system to IoT based networks. Keeping in view the ubiquity of IoT networks and the threats they face, a complete deception solution would be useful in not just thwarting attacks but also studying and analyzing the attackers' actions. We also intend to use Kubernetes for more efficient container orchestration.

CRediT authorship contribution statement

Waleed Bin Shahid: Design of Deception System, Attack Detection, Docker Setup/Deployment, Testing. **Baber Aslam:** Technical evaluation and feasibility of proposed technique, Docker Security. **Haider Abbas:** Technical review, Optimization of deception technique. **Hammad Afzal:** Deep Learning based attack detection, Technical review of proposed technique. **Saad Bin Khalid:** Practical deployment, Testing, Analysis of proposed technique.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgments

This research is sponsored by the Higher Education Commission (HEC), Pakistan (Grant: 2(1078)/HEC/ME/2018/707), through its initiative of National Center for Cyber Security for the affiliated lab National Cyber Security Auditing and Evaluation Lab (NCSAEL).

References

[1] Basit Abdul, Zafar Maham, Liu Xuan, Javed Abdul Rehman, Jalil Zunera, Kifayat Kashif. A comprehensive survey of AI-enabled phishing attacks detection techniques. Telecommun Syst 2021;76(1):139–54.
[2] Applebaum Simon, Gaber Tarek, Ahmed Ali. Signature-based and machine-learning-based web application firewalls: A short survey. Procedia Comput Sci 2021;189:359–67.
[3] Khan Nayeem, Abdullah Johari, Khan Adnan Shahid. Defending malicious script attacks using machine learning classifiers. Wirel Commun Mob Comput 2017;2017.
[4] Vartouni Ali Moradi, Teshnehlab Mohammad, Kashi Saeed Sedighian. Leveraging deep neural networks for anomaly-based web application firewall. IET Inf Secur 2019;13(4):352–61.
[5] Tekerek Adem. A novel architecture for web-based attack detection using convolutional neural network. Comput Secur 2021;100:102096.
[6] Kim Tae-Young, Cho Sung-Bae. Web traffic anomaly detection using C-LSTM neural networks. Expert Syst Appl 2018;106:66–76.
[7] Liang Jingxi, Zhao Wen, Ye Wei. Anomaly-based web attack detection: A deep learning approach. In: Proceedings of the 2017 VI international conference on network, communication and computing. 2017. p. 80–5.
[8] Zhu Mu, Anwar Ahmed H, Wan Zelin, Cho Jin-Hee, Kamhoua Charles, Singh Munindar P. A survey of defensive deception: Approaches using game theory and machine learning. IEEE Commun Surv Tutor 2021.
[9] Bringer Matthew L, Chelmecki Christopher A, Fujinoki Hiroshi. A survey: Recent advances and future trends in honeypot research. Int J Comput Netw Inf Secur 2012;4(10):63.
[10] Zobal Lukáš, Kolář Dušan, Fujdiak Radek. Current state of honeypots and deception strategies in cybersecurity. In: 2019 11th International congress on ultra modern telecommunications and control systems and workshops. IEEE; 2019, p. 1–9.
[11] Shahid Waleed Bin, Aslam Baber, Abbas Haider, Khalid Saad, Afzal Hammad. An enhanced deep learning based framework for web attacks detection, mitigation and attacker profiling. J Netw Comput Appl 2021;103270. http://dx.doi.org/10.1016/j.jnca.2021.103270, URL https://www.sciencedirect.com/science/article/pii/S1084804521002666.
[12] Barron Timothy, So Johnny, Nikiforakis Nick. Click this, not that: Extending web authentication with deception. In: Proceedings of the 2021 ACM Asia conference on computer and communications security. 2021. p. 462–74.
[13] Niakanlahiji Amirreza, Jafarian Jafar Haadi, Chu Bei-Tseng, Al-Shaer Ehab. HoneyBug: Personalized cyber deception for web applications. 2020.
[14] Trustwave. ModSecurity Open source web application firewall. 2020, https://modsecurity.org/. [Online; Accessed 21 February 2021].
[15] Jiang K, Zheng H. Design and implementation of a machine learning enhanced web honeypot system. In: 2020 13th International congress on image and signal processing, biomedical engineering and informatics. 2020, p. 957–61. http://dx.doi.org/10.1109/CISP-BMEI51763.2020.9263640.
[16] Fitri NR, Budi AHS, Kustiawan I, Suwono SE. Low interaction honeypot as the defense mechanism against slowloris attack on the web server. In: IOP conference series: Materials science and engineering, vol. 850, IOP Publishing; 2020, 012037.
[17] Grigorescu O, Săndescu C, Caba A. Web application honeypot published in the wild. In: 2020 19th RoEduNet conference: Networking in education and research. 2020, p. 1–6. http://dx.doi.org/10.1109/RoEduNet51892.2020.9324870.
[18] Djamaluddin Basirudin, Alnazeer Ahmed, Azzedin Farag. Web deception towards moving target defense. In: 2018 international Carnahan conference on security technology. IEEE; 2018, p. 1–5.
[19] Fraunholz D, Schotten HD. Defending web servers with feints, distraction and obfuscation. In: 2018 international conference on computing, networking and communications. 2018, p. 21–5. http://dx.doi.org/10.1109/ICCNC.2018.8390365.
[20] Henderson B, Mckenna S, Rowe N. Web honeypots for spies. In: 2018 international conference on computational science and computational intelligence. 2018, p. 1–6. http://dx.doi.org/10.1109/CSCI46756.2018.00009.
[21] Jia Z, Cui X, Liu Q, Wang X, Liu C. Micro-honeypot: Using browser fingerprinting to track attackers. In: 2018 IEEE third international conference on data science in cyberspace. 2018, p. 197–204. http://dx.doi.org/10.1109/DSC.2018.00036.
[22] Musch Marius, Härterich Martin, Johns Martin. Towards an automatic generation of low-interaction web application honeypots. In: Proceedings of the 13th international conference on availability, reliability and security. 2018. p. 1–6.

[23] Cernica I, Popescu N. Wordpress honeypot module. In: 2018 IEEE 16th international conference on embedded and ubiquitous computing. 2018, p. 9–13. http://dx.doi.org/10.1109/EUC.2018.00009.
[24] El-Kosairy Ahmed, Azer Marianne A. A new web deception system framework. In: 2018 1st International conference on computer applications & information security. IEEE; 2018, p. 1–10.
[25] Fraunholz Daniel, Reti Daniel, Duque Anton Simon, Schotten Hans Dieter. Cloxy: A context-aware deception-as-a-service reverse proxy for web services. In: Proceedings of the 5th ACM workshop on moving target defense. 2018. p. 40–7.
[26] Lin Jianbao, Liu Chaoge, Cui Xiang, Jia Zhaopeng. Poster: A website protection framework against targeted attacks based on cyber deception. In: 38th IEEE symposium on security and privacy. 2017.
[27] Djanali Supeno, Arunanto FX, Pratomo Baskoro Adi, Baihaqi Abdurrazak, Studiawan Hudan, Shiddiqi Ary Mazharuddin. Aggressive web application honeypot for exposing attacker's identity. In: 2014 the 1st international conference on information technology, computer, and electrical engineering. IEEE; 2014, p. 212–6.
[28] Sinha Rakhi, Uppal Dolly, Singh Dharmendra, Rathi Rakesh. Clickjacking: Existing defenses and some novel approaches. In: 2014 international conference on signal propagation and computer technology. IEEE; 2014, p. 396–401.
[29] Mphago Banyatsang, Mpoeleng Dimane, Masupe Shedden. Deception in web application honeypots: Case of glastopf. Int J Cyber-Secur Digit Forensics 2017;6(4):179–85.
[30] John John P, Yu Fang, Xie Yinglian, Krishnamurthy Arvind, Abadi Martín. Heat-seeking honeypots: Design and experience. In: Proceedings of the 20th international conference on world wide web. 2011. p. 207–16.
[31] Ishikawa Tomohisa, Sakurai Kouichi. Parameter manipulation attack prevention and detection by using web application deception proxy. In: Proceedings of the 11th international conference on ubiquitous information management and communication. 2017. p. 1–9.
[32] Virvilis Nikos, Vanautgaerden Bart, Serrano Oscar Serrano. Changing the game: The art of deceiving sophisticated attackers. In: 2014 6th International conference on cyber conflict. IEEE; 2014, p. 87–97.
[33] Djanali Supeno, Arunanto FX, Pratomo Baskoro Adi, Studiawan Hudan, Nugraha Satrio Gita. SQL injection detection and prevention system with raspberry Pi honeypot cluster for trapping attacker. In: 2014 international symposium on technology management and emerging technologies. IEEE; 2014, p. 163–6.
[34] Koniaris Ioannis, Papadimitriou Georgios, Nicopolitidis Petros. Analysis and visualization of SSH attacks using honeypots. In: Eurocon 2013. IEEE; 2013, p. 65–72.
[35] Valli Craig, Rabadia Priya, Woodward Andrew. Patterns and patter-an investigation into SSH activity using Kippo honeypots. Perth, Western: SRI Security Research Institute, Edith Cowan University; 2013.
[36] Leaden G, Zimmermann Marcus, DeCusatis Casimer, Labouseur Alan G. An API honeypot for DDoS and XSS analysis. In: 2017 IEEE MIT undergraduate research technology conference. IEEE; 2017, p. 1–4.
[37] De Faveri Cristiano, Moreira Ana. A SPL framework for adaptive deception-based defense. In: Proceedings of the 51st Hawaii international conference on system sciences. 2018.
[38] Rahmatullah Dandy Kalma, Nasution Suiya Michrandi, Azmi Fairuz. Implementation of low interaction web server honeypot using cubieboard. In: 2016 international conference on control, electronics, renewable energy and communications. IEEE; 2016, p. 127–31.
[39] Valicek Martin, Schramm Gregor, Pirker Martin, Schrittwieser Sebastian. Creation and integration of remote high interaction honeypots. In: 2017 international conference on software security and assurance. IEEE; 2017, p. 50–5.
[40] Sever Dubravko, Kišasondi Tonimir. Efficiency and security of docker based honeypot systems. In: 2018 41st International convention on information and communication technology, electronics and microelectronics. IEEE; 2018, p. 1167–73.
[41] De Gaspari Fabio, Jajodia Sushil, Mancini Luigi V, Panico Agostino. AHEAD: A new architecture for active defense. In: Proceedings of the 2016 ACM workshop on automated decision making for active cyber defense. New York, NY, USA: Association for Computing Machinery; 2016, p. 11–6. http://dx.doi.org/10.1145/2994475.2994481.
[42] Kyriakou A, Sklavos N. Container-based honeypot deployment for the analysis of malicious activity. In: 2018 global information infrastructure and networking symposium. 2018, p. 1–4. http://dx.doi.org/10.1109/GIIS.2018.8635778.
[43] Shrivastava Rajesh Kumar, Bashir Bazila, Hota Chittaranjan. Attack detection and forensics using honeypot in IoT environment. In: International conference on distributed computing and internet technology. Springer; 2019, p. 402–9.
[44] Ali P Dilsheer, Kumar T Gireesh. Malware capturing and detection in dionaea honeypot. In: 2017 innovations in power and advanced computing technologies. IEEE; 2017, p. 1–5.
[45] Eftimie Sergiu, Racuciu Ciprian. Honeypot system based on software containers. Sci Bull Nav Acad 2016;19(2):582.
[46] Müter Michael, Freiling Felix, Holz Thorsten, Matthews Jeanna. A generic toolkit for converting web applications into high-interaction honeypots. University of Mannheim; 2008, 280, 1–6.
[47] Giménez Carmen Torrano, Villegas Alejandro Pérez, Marañón Gonzalo Álvarez. HTTP DATASET CSIC 2010. 2012, https://www.isi.csic.es/dataset/. [Online; Accessed 1 April 2020].
[48] Trustwave. ModSecurity Open source web application firewall. 2020, https://modsecurity.org/. [Online; Accessed 28 March 2020].
[49] Hutchins Eric M. Proactively detect persistent threats. 2020, https://www.lockheedmartin.com/en-us/capabilities/cyber/cyber-kill-chain.html. [Online; Accessed 17 February 2021].
[50] Portswigger Web Security. Burpsuite. 2020, https://portswigger.net/burp. [Online; Accessed 28 March 2020].
[51] Alias Elbert. Identify technologies on websites. 2020, https://www.wappalyzer.com/. [Online; Accessed 28 March 2020].
[52] Ghasempour Alireza. Internet of things in smart grid: Architecture, applications, services, key technologies, and challenges. Inventions 2019;4(1):22.