Journal of Network and Computer Applications 213 (2023) 103596

Contents lists available at ScienceDirect

Journal of Network and Computer Applications

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

Dynamic Web Application Firewall detection supported by Cyber Mimic Defense approach

Mariusz Sepczuk

Faculty of Electronics and Information Technology, Warsaw University of Technology, Warsaw, Poland

ARTICLE INFO

Keywords:
Web Application Firewall
Mimic defense
DHR
Dynamic security
Web application security

ABSTRACT

With the increase of publicly available applications on the Internet, the number of new vulnerabilities increases. The currently used security methods are static and predictable and therefore have problems detecting unknown vulnerabilities. This issue creates an advantage for attackers — more attacks are carried out successfully than existing countermeasures that protect against them. It can especially be seen when considering the protection provided by Web Application Firewalls (WAF). Namely, it is often enough to obfuscate an attacker's payload to bypass security mechanisms successfully. Of course, many approaches are used to improve the protection provided by WAF, but this is associated with many problems, and a high level of security is expected almost from the moment such a device is deployed. One such approach may be the use of mimic defense, which is a proactive method of detecting unknown attacks. This paper presents the results of experiments in the network with web servers secured by WAF with additional protection provided by the mimic defense idea. The conducted research shows that the usage of mimic defense increases the number of detected and blocked attack attempts. It also introduces the unpredictability that an attacker has to confront when trying to carry out an attack. Moreover, the proposed concept allows for creating new temporary rules that supply the WAF while increasing the chances of detecting previously undiscovered attacks.

1. Introduction

The current development and accessibility of web applications over the Internet have made them an easy target for attackers. Regarding this, solutions protecting such applications have been introduced in the same way as in the case of protection at the network layer level. A typical Web Application Firewall (WAF) can detect threats in three ways: based on signatures (blocklists), based on the logic of the application (allowlist), and based on a hybrid approach (Applebaum et al., 2021; Razzaq et al., 2013). Each of these approaches has advantages but also limitations. Using only signatures, we can deploy and start WAF quickly, but its effectiveness can be questioned rapidly. It is often enough to modify accordingly the request sent to the webserver. In this case, WAF can recognize it as secure. To reduce the number of cases where HTTP requests are classified as safe, when, in fact, they are not (false negative), threat detection based on the application's logic was introduced. Before WAF begins to monitor and block insecure traffic, it must first "learn" the logic of the protected application. Although this approach provides effective protection, it is time-consuming. Thus, the hybrid solution is often deployed: signature rules and in parallel "learning mode". However, the hybrid approach in the short term does not solve the problem associated with the classification of HTTP requests as safe when in fact, they are not. One of the approaches used to increase network security is to use the concept of mimic defense. The mimic defense is another approach to improving the ability to detect potential attacks in the unequal battle between the defensive and the offensive.

The aim of this work is to present the new concept of using the mimic defense approach in the context of supporting the detection of threats by the web application firewall. Using the mimic defense technique in this context has at least two advantages. Firstly, it increases the detection level of attacks in the case of the configuration in which the firewall uses signature protection mode. Secondly, it supplies the firewall with knowledge about new attacks detected by the mimic defense components. Often, a simple obfuscation of the attacker's payload generally causes WAFs to fail to detect vulnerabilities. By using the mimic defense, such situations can be reduced.

The rest of the paper is organized as follows. Section 2 presents the description of protection methods provided by modern application firewalls. Section 3 outlines the assumptions of mimic defense. Section 4 reviews approaches used to increase the cost of an effective attack and the ability to defend against such attacks. Section 5 explains the proposed concept of the Web Application Firewall supported by

E-mail address: mariusz.sepczuk@pw.edu.pl.

https://doi.org/10.1016/j.jnca.2023.103596
Received 24 October 2022; Received in revised form 31 December 2022; Accepted 6 February 2023
Available online 11 February 2023

Table 1
Comparison of the most popular WAF solutions.

Feature	Akamai	Imperva	Cloudflare	F5	AWS	ModSecurity	WebKnight	IronBee	NAXSI	Shadow
Accessibility	Commercial	Commercial	Commercial	Commercial	Commercial	Open source	Open source	Open source	Open source	Open source
Variants of deployment	Cloud-based technologies	On-premises appliances, cloud-based technologies	Cloud-based technologies	On-premises appliances, cloud-based technologies	Partially on-premises appliances, cloud-based technologies	Software	Software	Software	Software	Software
Type of detections	Advanced Application and Network Layer, SQLI, Malicious Inclusion, XSS, DDoS	Runtime Application Self-Protection, API, Bot, DDoS, Attack Analytics, Client-Side Protection	OWASP Top 10, limits comment spam, DDoS, SQLI, blocks threats based on reputation, HTTP headers	Web app and API, automated attacks and bots, credential theft, attacks on mobile clients	XSS, SQLI, DDoS	SQLI, XSS, Malicious Inclusion, Brute force attacks, data leakage	SQLI, XSS, CSRF, parameter pollution, encoding exploits, bots, DoS, data leakage	The same as ModSecurity	Nginx Anti XSS and SQLI	SQLI, XMLI, Code and command injections, XSS, Malicious Inclusion, Backdoor access
Operating modes	Allowlist, blocklist	Allowlist, blocklist	Blocklist, partially allowlist	Allowlist, blocklist	Blocklist	Blocklist	Blocklist	Blocklist	Blocklist, partially allowlist	Blocklist, partially allowlist
Characteristic feature	Ensures Zero-Second DDoS Mitigation SLA	All-round website security solution decorated with all required features to ensure website security and integrity	Combines a reverse proxy with a content delivery network	Leverages advanced data analysis and machine learning technologies to detect and prevent cyberattacks	Works as an add-on to an existing subscription to cloud services	The most popular and constantly developed	Application firewall for the Microsoft IIS	Cloud-based WAF that could be affordable by everyone	Third-party Nginx module	Suite of tools designed to identify, record, and prevent web application attacks
Disadvantages	Lack of self-hosted option	Many features are not applicable to smaller organizations that do not have to monitor compliance	Acts as an intermediary, so if something goes the wrong site will go down as well	Rule creation is a relative weakness	Available only to customers who just use the company's Web Services	Tough to maintain rule set	Application firewall only for the Microsoft IIS	Still in the development phase and is not available in binary install packages	Not work for Apache or IIS, restricted only to XSS and SQL Injection attack	Does not block malicious requests

mimic defense components. Section 6 shows the achieved results of experiments showing the legitimacy of using a web application firewall in conjunction with the idea of mimic defense. Section 7 contains a discussion of the obtained results. Section 8 concludes the paper and outlines future work.

2. Protection methods provided by modern application firewalls

A web application firewall (WAF) is a type of firewall that is used to monitor and block suspicious traffic to and from the web application (Prokhorenko et al., 2016). WAFs work on an application ISO/OSI model layer, so they block attacks such as SQL Injection, Cross Site Scripting, Information disclosure, and many more. There are many WAF solutions available. The most popular ones are presented and compared in the Table 1. In addition, many approaches to detecting web applications attacks can be found in the literature (see Table 2). Unlike traditional firewalls that inspect IP addresses, ports, or geolocalization, WAFs filter traffic based on some rules and detect abnormal web application behavior. Usually, detection is performed using one of three major approaches: signature-based, traffic pattern analysis, and hybrid (combining the previous two) (Palka and Zachara, 2011a). The signature-based method uses a local list of signatures. Each signature includes a pattern (attack signature) that indicates a known attack. In this mode, WAF analyzes each request and response for the presence of signatures. On this basis, it passes traffic if no attack is detected, or, if an attack is detected, it blocks such traffic or creates an appropriate alert. The main advantage of using WAF with a signature-based approach is its simplicity and a high rate of blocking of specific traffic provided by WAF. It is enough to give the tool an appropriate list of attack patterns, and malicious behavior will be detected almost immediately after WAF is launched. Moreover, one simple signature usually blocks one attack pattern, so it is very straightforward to associate the particular signature with the location in the web application that is to be protected. In the past, WAF relied on local databases with signatures, but today this method has limitations in the context of new or unknown attacks. The signature-based approach's primary disadvantage is the constant updating of the signature list. Every time a new attack appears, a unique signature is created that can detect it. WAF cannot identify and block the recent attack without these new signatures. Consequently, this may lead to an effective attack execution, and thus several dozenrecent attacks occur each year. Even simply obfuscating malicious traffic is enough to bypass WAF based on signature protection and prevent it from being able to detect the attack.

The second approach is related to traffic analysis. By analyzing the traffic related to the web application, WAF can determine whether it is allowed or not. For example, if a parameter accepts only positive integer values less than 1000, and the user submits a negative value in the request, the request will be blocked because it is inconsistent with the "normal" allowed traffic for this application. Such an approach enables attacks to be detected even without a signature. In this method, WAF can work in two modes: monitoring (sometimes called learning) and blocking. In the monitoring mode, the application

Table 2
Comparison of WAF detection& blocking attacks techniques.

Web application solution	TokDoc: a self-healing web application firewall (Kruege, 2010)	A Machine-Learning-Driven Evolutionary Approach for Testing Web Application Firewalls (Appelt et al., 2018)	Leveraging deep neural networks for anomaly-based web application firewall (Vartouni et al., 2019)	Formal reasoning of web application Firewall rules through ontological modeling (Ahmad et al., 2012)	Improving Web Application Firewalls through Anomaly Detection (Betarte et al., 2018)	Anomaly-based web attack detection: a deep learning approach (Liang and Wen. Y. Wei, 2017)	Artificial neural network based web application firewall for SQL injection (Moosa, 2010)	Learning web application firewall-benefits and caveats (Palka and Zachara, 2011b)	Design and implementation of an artificial intelligence-based web application firewall model (Tekerek and Bay, 2019)	A self-learning anomaly-based web application firewall (Torrano-Gimenez et al., 2009)	Proposed solution (ModSecurity supported by a mimic defense approach)
Type of detection& description	Machine-learning (anomaly-based)	Machine-learning	Machine-learning (anomaly-based)	Signature-based	Signature and machine-learning (anomaly-based)	Machine-learning (anomaly-based)	Signature-based	Machine-learning	Signature-based detection (SBD) and anomaly-based detection (ABD) using an artificial neural network	Machine-learning (anomaly-based)	Signature-based
Description	TokDoc is a protocol-aware reverse HTTP proxy (the token doctor), that intercepts requests and decides on a per-token basis whether a token requires automatic "healing". Moreover, the proposed solution used an intelligent mangling technique, which, based on the decision of previously trained anomaly detectors, replaces suspicious parts in requests with benign data the system has seen in the past.	The ML-Driven approach is based on machine learning and an evolutionary algorithm to automatically detects holes in WAFs that let SQL injection attacks bypass them. The solution automatically generates a diverse set of attacks, sends them to a WAF under test, and checks if they are correctly identified.	Proposed method based on deep-neural-network and parallel-feature-fusion that features engineering as an integral part of them and plays the most crucial role in their performance. The proposed methods use the stacked autoencoder and deep belief network as feature learning methods, in which only normal data is used in the classification of the training phase, then, one-class SVM, isolation forest, and elliptic envelope are applied as classifiers.	A conceptual framework for structuring WAF management. The signature-based WAFs are managed by administrators resulting in complex, layered configurations which are difficult to manage and have suboptimal performance. The proposed framework allowed redundancies and relationships to be identified and for solutions to be more easily evaluated in the context of existing rules.	The created solution attempts to solve the problems of performance of ModSecurity using the out-of-the-box OWASP Core Rule Set (e.g., high rates of false positives, poor recognition of zero-day attacks). The solution supplements ModSecu-rity/CRS with a one-class classification suitable for use in the absence of specific training data and an n-gram anomaly detection-based approach for when application-specific training data is available.	A new approach for anomaly detection in web applications using Recurrent Neural Networks (RNNs). The performance investigation is focused on measuring the Accuracy, Sensitivity, and Specificity of the model.	The proposed Artificial Neural Network-based WAF (ANNbWAF) solution aims to produce a high accuracy in detecting SQL injection attacks without much loss in the performance when the number of negative patterns increases as the case of pattern matching based on regular expressions. The solution also has the potential to be extended to include other types of attacks that take advantage of the lack of input validation, such as Cross-Site Scripting (XSS) and other kinds of Injection Flaws.	Proposed solution implements a machine-learning-based WAF in Apache based on parameter length and character distribution, parameter class (i.e., numerical, URL, email, etc.) and enumerated type.	A hybrid learning-based web application firewall (WAF) model is proposed to prevent web-based attacks, by using signature-based detection (SBD) and anomaly-based detection (ABD). The detection of known web-based attacks is done by using SBD, while the detection of anomaly HTTP requests is done by using ABD. Learning-based ABD is implemented by using Artificial Neural Networks (ANN). Thus, an adaptation of the model against zero-day attacks is ensured by learning-based ABD using ANN.	Created approach is an anomaly-based WAF configured using an XML file describing in detail the web application's expected normal behavior. The XML file contains rules defining normal requests, including HTTP verbs, headers, and directories corresponding to the web application's folder structure, including input arguments and legal values defined by statistical properties.	WAF detection supported by the mimic defense approach

(*continued on next page*)

Table 2 (*continued*).

Advantaged	High level of false negative detection	Evaluation results suggest that the performance of ML-Driven (and its enhanced variant in particular) is effective at generating many undetected attacks and provides an excellent basis for identifying attack patterns to protect against them. Furthermost, it also fares significantly better than the best available tools (e.g., sqlmap)	The results show that in contrast to the traditional models, deep models, especially fusion models have better performance regarding accuracy and generalization in reasonable detection time.	Ease of managing and modifying current WAF rules	Improving false positive performance and n-gram anomaly detection providing better resilience against zero-day attacks.	The results for Accuracy and Sensitivity are much better for the proposed solution than for ModSecurity. Specificity at a similar level.	The solution based on an ANN for distinguishing between normal and malicious (SQL Injection) content of the training data gives the promising result of both accuracy and processing time, especially the second approach, which is a keyword-based approach.	The primary advantage of a Learning WAF is its ability to secure custom applications with no or little extra setup involved. The best environment for such a WAF is either one with high traffic and/or where complete acceptance tests are part of the deployment.	The proposed WAF achieves good levels of performance through its hybrid architecture. The faster signature-based phases are performed before, the slower anomaly-based phase. Previous normal and strange requests, identified by the anomaly-based detection phase, are fed back to the signature-based phases to increase detection performance.	The firewall achieves extremely high accuracy and low levels of false positives due to the XML file very closely matching the web application's normal behavior.	Detection of unknown attacks. Creation and supplying WAF with detection rules.
Disadvan-taged	Almost twice the median runtime compared to other proxies of this Type	Detection of SQL Injection attacks only	Usage of deep models can be time-consuming in training for higher orders of n-grams	Does not detect unknown attacks	The n-gram approach is effective with data related to the correct application traffic. In practice, obtaining a reliable set of such data can take a long time.	The work does not introduce any benchmark to compare their results against, although claiming their performance is comparable to the state-of-the-art. Therefore, there is no comparison of the solution's performance in the context of known and commonly used datasets.	Effective only against SQL Injection attacks. The quality of a trained ANN often depends on its architecture and the way the ANN is trained. More importantly, the quality of the trained ANN also depends on the quality of the training data and the features extracted from the data.	The major concern for the deployment of a Learning WAF into the production environment is the quality and completeness of the data used to train a WAF before it can reliably be used to control incoming traffic (collecting such data takes a long time)	The signature-based approach detects only selected attacks, the machine learning approach takes time due to the collection of a sufficiently large volume of good quality data	A limitation is the difficulty in creating the XML file because of the challenge of obtaining large volumes of nonmalicious traffic. Challenges are also experienced with websites that dynamically add resources or content	Approach introduces an additional delay in a protected network

firewall learns what traffic is allowed for such an application and its parameters. Such learning requires a certain amount of time for the AI/ML mechanisms in WAF to learn to filter traffic in the context of a given application properly. Time is also influenced by the size of a given application, i.e., the number of parameters sent in queries. This approach is efficient because it allows only the correct values of the transmitted application parameters. They are blocked if they do not comply with the general application logic. The disadvantage of this mechanism is the time needed to learn the application movement. If the time is too short, WAF will let malicious traffic through. WAF will not protect the application during this period if it is too long because it will keep learning. Moreover, the applications are constantly being developed. Changes are made, and new functions are added, so even if WAF has known the traffic of some part of the application, it may not be up-to-date anymore and requires re-learning the application traffic in the context of the introduced changes. Otherwise, requests related to new application functions will be blocked because, according to WAF, they will differ from the allowed traffic. Such a situation is a challenge in large microservice environments, where a given service may be changed several times a day.

The third, hybrid approach is most often used, i.e., the method in which WAF protects the application based on signatures and simultaneously learns the application and starts blocking traffic at the right moment. Of course, this is a trade-off between the signatures being protected quickly and the long time it takes to learn the application traffic and start blocking.

To increase the effectiveness of detecting new attacks, elements of mimic defense can be used for this purpose. Standard signature protection for unknown attacks will not start to detect until there is an appropriate attack pattern. Applying mimic defense in such a situation can block the attack. In addition, while waiting for the official signature rule, the relevant element of mimic defense can create a temporary rule that will feed the local WAF list, which will cause the attempt to attack using a given pattern to be blocked. Therefore, we can speak of a local, dynamic supply of WAF with new rules.

3. Concept of mimic defense

Many concepts in the field of cybersecurity are inspired by broadly understood nature. For example, such a situation occurs when naming of malware (e.g., viruses and worms), as well as in the case of DNA cryptography (the use of DNA sequence molecules in information protection Xiao et al., 2006), or IDS/IPS systems (the natural immune system is a prototype example of a biological system supporting defense against intruders). It is no different with mimic defense. The concept of mimic defense is derived from the features of the mimic octopus species (Hanlon et al., 2010). This octopus can imitate several species of animals (usually, the abilities of animals in this direction are limited to one), including stingray, jellyfish, salt fish, and a sea serpent. Depending on the context of the situation in which the animal can be found (defense or offense), it can quickly adapt to the environment, changing the imitation of one species into another. These features, i.e., the quick and varied adaptation to the environment in a given situation, inspired researchers to create the concept of mimic defense.

The idea of mimic defense (Hu et al., 2016; Wu, 2017) uses novel defense theory and methods to detect threats such as unknown security vulnerabilities in various areas of cyberspace. In particular, these may be vulnerabilities in web applications. The primary assumption of this concept is the introduction of a heterogeneous and redundant architecture using camouflage techniques in the target system to increase the dynamics, randomness, and diversity of this system. Fig. 1 shows the general scheme of the DHR (dynamic, heterogeneity, and redundancy architecture) architecture that implements the mimic defense approach. The architecture includes several modules: input agent, Arbitrator, heterogeneous executors and components, and strategic scheduling algorithm.

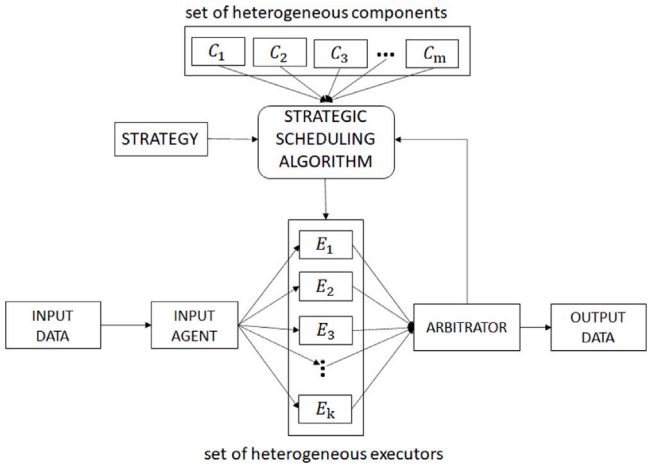

Fig. 1. DHR architecture.

Firstly, the input data is sent to the Input Agent, which duplicates it and distributes it to all executors (a set of heterogeneous executors) $E_i(k = 1, 2, 3, \ldots, k)$ to be processed appropriately. Based on the instruction from the Strategic Scheduling Algorithm module, the Input Agent selects executors to which input data should be sent. Each executor is a software or hardware entity (for example, a virtual machine with a web server) that consists of a different set of heterogeneous components $C_j(j = 1, 2, 3, \ldots, m)$ to provide diversity and an easier detection of a vulnerable component. The components are various elements of software or hardware (for example, different PHP versions or web application servers). Again, the Strategic Scheduling Algorithm module is responsible for which components will operate within a particular executor. After the input data has been processed, the executors send responses to the Arbitrator module, which, based on received responses, decides whether an attack has occurred and sends an appropriate message to the user (output data). Moreover, when an attack is detected, the relevant information is sent to the Strategic Scheduling Algorithm, which, based on the established strategy, locates a vulnerable executor, creates a new one without vulnerable components, and replaces it for the vulnerable one.

From the mathematical point of view, each executor can be treated as a vector with several components (see Eqs. (1) and (2)).

$$E_k = (C_{1k}, C_{2k}, C_{3k}, \ldots, C_{nk}) \tag{1}$$

$$C_m = (C_{m1}, C_{m2}, C_{m3}, \ldots, C_{mn}) \tag{2}$$

At the same time, the set of executors and components could be shown as a matrix (see Eqs. (3) and (4)). Every vector E_k consists of many different types of components, and vector C_m includes many different versions of the same kind.

$$E = \begin{bmatrix} C_{11} & C_{21} & C_{31} & \ldots & C_{n1} \\ C_{12} & C_{22} & C_{32} & \ldots & C_{n2} \\ C_{13} & C_{23} & C_{33} & \ldots & C_{n3} \\ \ldots & \ldots & \ldots & \ldots & \ldots \end{bmatrix} \tag{3}$$

$$C = \begin{bmatrix} C_{11} & C_{12} & C_{13} & \ldots & C_{1n} \\ C_{21} & C_{22} & C_{23} & \ldots & C_{2n} \\ C_{31} & C_{32} & C_{33} & \ldots & C_{3n} \\ \ldots & \ldots & \ldots & \ldots & \ldots \end{bmatrix} \tag{4}$$

The more executors, the more difficult the attack is to perform successfully. Naturally, not all executors can always be used. The number of executors that can be selected depends on the context of their use, particularly the legitimacy of using the selected components. It may prove that a given web application runs only on servers with a

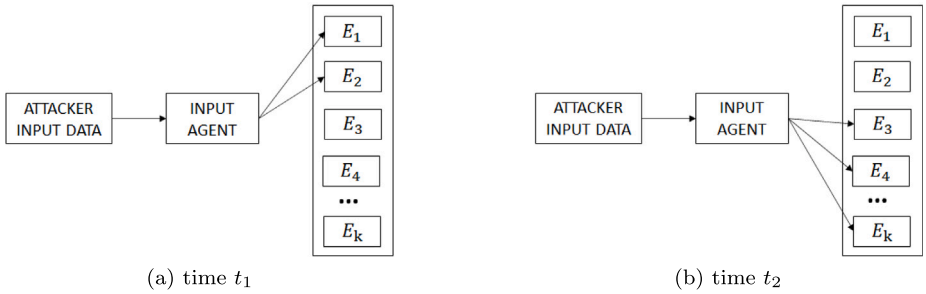

Fig. 2. Attack surface at different moment of time.

specific type of operating system installed and not on others. Therefore, when implementing the DHR architecture, it is necessary to define appropriate rules considering the dependencies between components.

In the context of the attack surface that defines the system elements that can be used to launch the attack, the mimic defense strategy introduces unpredictability to the attacker using specific system resources at a given moment (Wu, 2020). It means that in moment t_1 of time a set of executors used to act according to the attacker's request can be slightly different than in moment t_2 of time (compare Fig. 2). As a result, the attack surface changes over time depending on the selection of executors and related components.

4. Related papers

Popular, currently used approaches to ensuring security against attacks focus on three areas: information protection by securing the system with the use of, for example, encryption, access control or firewalls, detection of intrusions and vulnerabilities based on system logs or IDS/IPS traffic analysis, and the use of adequately prepared systems and applications to catch cybercriminals trying to use a security vulnerability (honeypot). The previously mentioned approaches to cyber protection are often static, certain, and do not consider the structure of the object they protect. Such a concept can be used in large networks as a basic form of implementation, and then, depending on the needs, it can be adapted accordingly. However, it usually works for attacks with known and established characteristics but is less effective for unknown vulnerabilities.

The nature of cyberspace is characterized by the ease of attack and the difficulty of defense. Typically, an attack, once carried out, can be repeated under similar conditions to achieve the desired effect. Introducing the dynamics, randomness, and redundancy of a system or network makes it difficult for attackers to obtain essential data, reuse a previously used vulnerability, or continue a further attack in depth. As a result, from the attacker's point of view, the cost of an attack that is required to be successful, i.e., attacking a system or network at a given time, increases.

Several proposals for new dynamic approaches to cyber defense can be found in the literature. These include dissimilar redundancy, intrusion tolerance, moving target defense, and mimic defense.

4.1. Dissimilar redundancy

The concept of Dissimilar Redundancy Architecture is a relatively old method of providing security in the aerospace industry (Yeh, 1998, 1996). Boeing 777 introduces a Fly-By-Wire (FBW) system that needs to ensure adequate integrity and availability in flight control systems. The idea of FBW assumes the triple hardware resources redundancy in the context of computing and electrical system, hydraulic power, and communication paths. Considering that, only one computer is active, and others are in standby mode. As soon as the computer is interrupted, for example, due to an error, one of the other computers immediately goes into active mode and takes over the operation. Such a concept ensures a high level of reliability and a probability of a system control failure of less than 10^{-9} over a ten-hour flight. Another example of Dissimilar redundancy or actually Dissimilar four-redundancy can be found in the paper (Kuang et al., 2015). Authors propose a solution to meet the reliability requirements from a concept of an electrical system More/All Electric Aircraft that can defeat the common mode fault in the redundant systems. In the paper (Guodong et al., 2009) can be found the description of a Twice Dissimilar Redundancy Computer based on Roll-Forward Recovery Scheme (TDCS) that helps avoid the correlated faults and realize fault-tolerance. To achieve this, TDCS applies two separate design principles that use different hardware, development solution, program language, and instruction set. Over time, it has been noticed that the concept can also be used in cybersecurity in systems defense. The paper (Zhang et al., 2020) refers to an essential software diversification tool chain that supports the protection of a system. This tool chain provides an integrated essential software heterogeneousness environment that can get multiple entities of different execution components to enhance the availability of cybersecurity defense with dissimilar redundancy and decrease operating costs.

4.2. Intrusion tolerance

The Intrusion tolerance system (ITS) is a system that continuously provides a service despite the successful attack. Such assumption protects against known attacks that could be detected by firewalls or secured by encryption, and against unknown attacks. It is possible by introducing activities such as reconfiguring hardware and software resources or limiting access to data. The ITS objective, according to the Zhao et al. (2008) can be considered in four areas: protective system's safe operation, guarantee service usability, guarantees on the secrecy of server data, guarantees on the authenticity and integrity of server data. Over time, new protocols and systems have been developed using intrusion tolerance. Paper (Welch et al., 2003) refers to a European project called Malicious- and Accidental-Fault Tolerance for Internet Applications (MATIA). The project's goal was a holistic investigation of the tolerance paradigm. It was done by research in three areas: definition of architectural framework and conceptual model, creating mechanism and protocols, and formal verification, validation, and assessment. As a part of the next project, the Organically Assured and Survivable Information System (OASIS) (Lala, 2003), a new tolerance architecture, was developed. The architecture includes several security domains that make it difficult for an attacker to launch an attack successfully. Therefore, the concept operates with unpredictable adaptation as protection against the attacks that try to predict and abuse this adaptation. Another solution that employs the Intrusion tolerance approach was created by Duke University, and called Scalable Intrusion-Tolerant Architecture for Distributed Services (SITAR). Its primary goal was to increase security by using redundancy to reconfigure systems (Wang et al., 2003a,b). Authors emphasize that their concept can be used with generic types of services, and thank to redundancy and diversity, tolerance refers both to internal and external attacks. The document (Bangalore and Sood, 2009) describes an approach

to securing web servers using Intrusion Tolerance. The Self Cleaning Intrusion Tolerance (SCIT) uses virtualization technology to make it difficult for the attacker to perform successful data gathering/damage by reducing the Internet exposure time of a particular virtual host to a few minutes. The authors of Castro and Liskov (2003) show a new replication algorithm, BFT, that can be used to build highly available systems that tolerate Byzantine faults. Another solution to tolerate Byzantine faults is presented in Malkhi et al. (2001). The Fleet is a middleware system implementing distributed data stores for persistent Java objects. Moreover, it was created to be highly available, dynamically extensible with new object types, and scalable to large numbers of clients and servers.

4.3. Moving target defense

The idea of Moving Target Defense (MTD) appears as a proactive defense method that prevents attacks. MTD is a set of strategies that improve security, resilience and the availability of applications and systems by a diversity of software and paths in the network. The aim of MTD is to control the attack surface by modifying different system configurations. Moreover, MTD introduces uncertainty and complexity for attackers because each time, the attacker gets a random and unpredictable view of the system, which decreases the opportunity for target identification, scanning, or successful exploits. In literature, many solutions use MTD, which can be divided into several areas of protection against attacks related to reconnaissance or scanning, denial of service (DoS), buffer overflow (BOF), worms, web applications, etc. In the paper (Achleitner et al., 2017) authors describe a reconnaissance deception system based on a software-defined network (SDN) paradigm to trick an attacker by simulating virtual topologies. The created system prevents network reconnaissance by delaying the scanning process, in which it is possible to find an active host and existing vulnerabilities, hiding essential and accurate information that can be used in the next step of attack with low influence on proper network traffic. Another technic against reconnaissance could be found in Duan et al. (2013). The paper shows the Random Route Mutation (RRM) approach that enables randomly switching the route between a given source and destination addresses. Finding satisfying paths can be achieved by solving constraints in the RRM model with a SMT solver. Moreover, this proactive strategy can protect against eavesdropping and DoS attacks. Article (Aydeger et al., 2016) explains how a software-defined network can be used to protect against crossfire DDoS attacks. The idea includes two original concepts. First, the detection performed by the SDN controller is based on identifying source–destination pairs engaged in a traceroute operation. Second, gathering information in the detection phase is used in the MTD defense mechanism based on the SDN controller's ability to monitor all switches in the network. In the next paper (Jia et al., 2013) can be found another anti-DDoS mechanism, MOTAG. This moving target defense mechanism secures service access for authenticated clients against flooding DDoS attacks. In the context of buffer overflow protection, two interesting position are worth considering: Shacham et al. (2004) and Kil et al. (2006). The first one is widely known as Address space layout randomization (ASLR), a computer security technique that involves randomly positioning the base address of an executable and the position of libraries, stack, and heap in the address space of a process. Position (Kil et al., 2006) refers to Address Space Layout Permutation (ASLP) that improves the address space randomization techniques in the context of increasing resistance to memory corruption attacks. ASLP is a novel binary rewriting tool that enables users to permute static code and data regions with an exact level. As mentioned earlier, the MTD technics can also be used to defend against worms. In the article (Al-Shaer et al., 2012) a novel framework called Random Host Mutation (RHM) is presented. It turns the end-host untraceable by IP addresses mutation without significantly impacting network performance. The authors proved that RHM effectively protects against stealthy scanning, different types of worm propagation, and attacks that need data from the reconnaissance phase for a successful start. A similar solution, Network Address Space Randomization (NASR), is shown in Antonatos et al. (2007). NASR is used to resist the influence of hitlist worms on the network by frequently changing nodes' IP addresses. Due to the rapid development of complex web applications, defense methods often do not keep up with attacks. To improve this situation, MTD techniques related to Game Theory can be used: Neti et al. (2012) and Sengupta et al. (2017). The papers apply the method to generate a switching strategy as a repeated Bayesian game based on the MTD architecture. It is worth emphasizing that the mentioned technique is not limited only to the web but can be used to secure databases regardless of the type and features.

4.4. Mimic defense

As mentioned in one of the previous subchapters, the mimic defense (MD) is an active defense mechanism that was constructed as an extension of Intrusion tolerance and Dissimilar redundancy concepts. MD aims to introduce diversification of variants of protected target that will run in parallel dynamically, and decide which results from variants are correct and secure. Contrary to MTD, MD protects against an attack that has already bypassed other security mechanisms (MTD instead hides critical information required to plan an attack from the attacker) by neutralizing malicious requests from the attacker by a voting module. Such an approach disturbs the attacker's perception based on information about an ineffective attack.

Despite the fact that MD is a relatively new concept, several solutions using MD elements have been created. Authors in the paper (Fan et al., 2018) propose mimic defense image encryption based on three assumptions. First, encryption and decryption areas are built according to the MD principle. Secondly, the input agent and the Arbitrator from the Central controller area are responsible respectively for distributing tasks to the executor and making a decision on decryption results. Finally, the decision based on decryption was made by a consistent decision mechanism. Papers (Hu et al., 2018; Xu, 2021) describe a mimic router. Since routers usually lack solutions to detect attacks such as firewalls or IDS, they are vulnerable. Taking over such devices by an attacker can cause chaos and a possibility of eavesdropping on transmitted data. The use of a MD-based model inside a router introduces multiple virtual executors on different routing planes. The purpose of such an idea is to construct a logical heterogeneous redundant processing unit in each plane that will cause difficulties for the attackers to exploit common security flaws in the executors. Another example of mimic defense can be found in Lin et al. (2017). The article introduces MDFS, a mimic defense theory-based architecture for Distributed File Systems, to achieve a high level of data protection. MDFS integrates dynamism, heterogeneity, and redundancy. The heterogeneity of MDFS comes from the diversification of software which refers to many instances of the same software. The redundancy of MDFS refers to the aspect that the data copies are located in various storage clusters. The dynamism of MDFS applies to rotating and reconfiguring used storage clusters. Due to the massive increase in the popularity and importance of cloud technologies, these environments are exposed to many threats. To protect them, the example of MD-based solutions were proposed in the cloud-computing (Wang et al., 2018), and edge-computing environment (Sang and Li, 2019). Article (Li et al., 2021) discusses the mechanism of the Mimic Defense Web Server (MDWS). The MDWS includes a heterogeneous executor set and an Arbitrator. The executors are the application servers built with the same functions but different configurations; the Arbitrator performs two actions: distributing the requests to all servers and voting. The structure of all application servers is hidden behind the Arbitrator; thus, it confuses the potential attacker and makes it difficult to carry out an effective attack. In addition to the approaches mentioned, solutions such as a mimic network operating system (Wang et al., 2017a), mimic DNS (Wang et al., 2017b) and mimic encryption (Li et al., 2018) can be found.

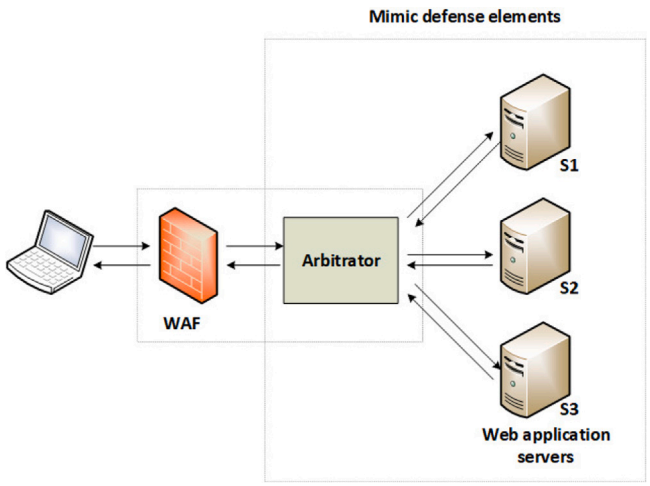

Fig. 3. Compare flow operations.

5. Proposed solution

As mentioned in the previous Sections of this paper, the mimic defense approach can increase the level of security provided by signature protection. The creation of such a solution was performed in two stages: defining the general concept and creating a test environment implementing this concept. Both phases will be described in this Section.

5.1. Description of the concept of web application firewall with the mimic defense elements

The realized concept of connecting mimic defense with the protection provided by the application firewall is presented in Fig. 3. This is a simplified version of the architecture discussed in Section 3. The simplification is due to the fact that the main emphasis in the research was on the very operation of mimic defense and the security it provides. The aspect of creating and exchanging vulnerable executors was omitted as part of maintaining the continuity of the network. As shown in Fig. 3, the presented approach consists of 4 parts:

- The attacker's PC that sends requests to the website,
- Application firewall that checks if the request sent by the user does not contain attack patterns,
- Arbitrator, an element of mimic defense that decides whether there has been an attempted attack that is not detected by the application firewall,
- Web application servers on which the users view websites are run.

The idea behind the operation of architecture is shown in Fig. 4. The HTTP request, which the application firewall has analyzed, goes to the Arbitrator module. This module copies the request and sends it to all (in this case, three) application servers. In practice, such an action means replacing data related to servers' IP addresses and the data in the headers of HTTP requests, including the session ID. Each server runs the same application but in a different configuration (e.g., a given server has various application server software installed on which the application runs, other versions of programming libraries, or other database engines). After receiving the request, each server sends the answer back to the Arbitrator. Based on the answers received, the Arbitrator decides whether or not there was an attempt to attack. It is done by comparing the body of the server response, which is treated as a string (see Fig. 5). Every string is compared with others, and depending on the result of this check, an appropriate decision is made. The selected response with appropriate web content is sent to the user if the attack is not detected. Otherwise, a general error message is sent, and the Arbitrator creates a rule transmitted and installed in real-time on the WAF to identify the future attack attempt. The rule includes a pattern that is characteristic of a given attack. Thanks to this, when someone tries to send an HTTP request with a typical, malicious string again, it will immediately be detected by WAF and blocked.

The arbitrator knows duplicate requests and distinguishes between the types of requests (e.g., GET and POST). Therefore, the created rule for GET request contains the path to the file and search parameters, and for POST request, search parameters from BODY. In particular, the rules are created as follows:

1. for HTTP GET Request

```
SecRule REQEST_URI "@contains <the path to the
file and
search parameters>" "id:<the rule id>,phase:1,
log,deny,status:403"
```

2. for HTTP POST Request

```
SecRule REQEST_BODY "@contains
<search parameters>"
"id:<the rule id>,phase:2,log,deny,status:403"
```

Additionally, using the proposed solution, we limit the traffic admitted to the network — WAF blocks malicious traffic without needing to re-analyze it by elements of the mimic defense. Of course, the procedure of installing a new rule on the Web Application Firewall must consider its operation's continuity. Although the new rule in the case of the mimic defense approach will not often appear, despite everything, it is essential to avoid a situation in which, by adding the new rule, the firewall will not work. Additionally, the Arbitrator records which executor is vulnerable and should be replaced. It should be emphasized that the Arbitrator does not return information on which component can be abused by an attacker but only indicates which executor contains a vulnerable component. The executor is replaced, and the replacement may concern all components or only one (depending on the adopted strategy). Many such substitutions for the same request in the long term may indicate which component is vulnerable (most or all components configurations will be checked, and based on this data, precise identification is possible).

The above description can be simplified to the following model:

(1) Arbitrator sends request to selected executor and get a response: $req(E_i) \rightarrow repE_i$ where $i \in \{1, 2, 3, \ldots, k\}$ and k is the largest executor index

(2) Arbitrator selects 3 of executors, sends to them request and wait for responses:

$$\begin{cases} req(E_a) & \rightarrow repE_a \\ req(E_b) & \rightarrow repE_b \\ req(E_c) & \rightarrow repE_c \end{cases}$$

where

$$a, b, c \in \{1, 2, 3, \ldots, k\} \wedge a \neq b \neq c$$

(3) Arbitrator compares all responses: $comp(repE_a, repE_b, repE_c)$

5.2. Analysis of the concept of web application firewall with the mimic defense elements

The proposed architecture of the solution is not an accurate representation of the concept of mimic defense but only a part of it. Due to the assumptions made, both advantages and disadvantages can be seen in the proposed solution. This subsection discusses the scope of the newly developed approach, as well as its advantages and disadvantages.

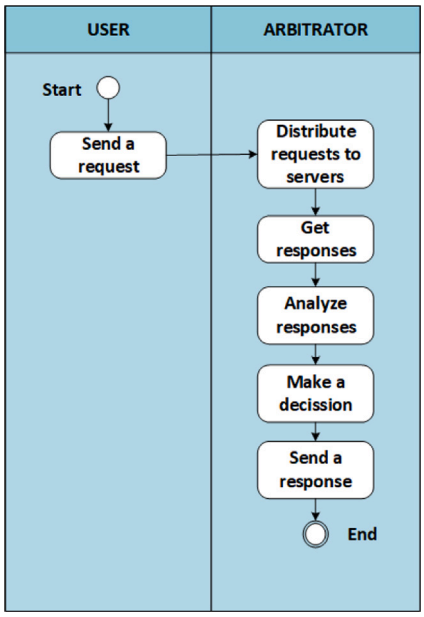

Fig. 4. Operation flow of created solution.

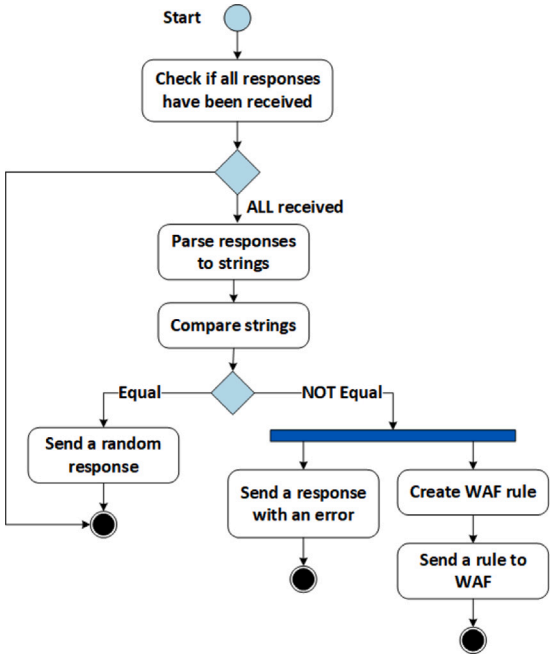

Fig. 5. Compare flow operations.

5.2.1. *Scope of the proposed solution*

As mentioned at the beginning of this subchapter, certain assumptions were made regarding the implementation of the concept of mimic defense supporting the detection of attacks by the WAF. The scope of the idea is as follows:

- The Arbitrator performs two actions: duplicating and distributing requests and deciding whether an attack occurs. This simplification is based on the assumption that the built environment should be as simple as possible and simultaneously provide the required functions.
- The Arbitrator decides whether an unknown attack could have occurred based on comparing HTTP responses. This is accomplished by converting the responses to strings and comparing them.
- The solution does not take into account the mechanism of exchanging vulnerable Executors. The Arbitrator only notifies the need to replace a given Executor. The proposed solution focuses mainly on the aspect of attack detection.
- The strategy for selecting the set of Executors to which the HTTP request will be sent is to choose three randomly from the available pool. Such a choice results from the fact that two responses from Executors can give different answers. Then with a high probability, the third answer may be the key to making the right decision (because, for example, the third answer will be the same as one of the others, so an attack probably exists).
- An environment consisting of 3 servers was built to demonstrate the attack detection concept.
- After detecting an attack, the Arbitrator creates a temporary rule for the WAF. Thanks to this, the firewall will block the attacker's reuse of the same payload without the support of mimic defense elements.

5.2.2. *Advantages of the proposed solution*

The use of the mimic defense concept brings many advantages. Below are the most relevant of them:

- Detection of unknown web attacks that WAF bypasses.
- Dynamic creation of temporary attack detection rules for WAF. Think of it as intelligent virtual patching.
- Web application traffic reduction. In a classic approach, without the participation of WAF, the Arbitrator analyzes each HTTP request received from the user. By supplying WAF with rules for detecting attack attempts previously discovered by the Arbitrator, malicious traffic will be blocked already at the analysis stage by the application firewall. Thus internal traffic will be limited.
- Introducing the uncertainty of the result of the request sent by the attacker. Because the Arbitrator selects a new set of Executors for each request, the Arbitrator's decision will also be made in the context of responses related to these Executors. This means that even if the attack succeeds once, it will be pretty difficult to reproduce, and the attacker will not be able to predict when the success will happen.
- By analyzing many Arbiters' decisions regarding the attack and indications of the Executor that should be replaced, it is possible to accurately identify the vulnerable component. It all depends on the number of combinations of elements that were used to build the Executor, but using all of them indicates a vulnerable component. Of course, the set of components should be matched to the detected vulnerability (a given set of components may have several vulnerabilities at the same time).

5.2.3. *Disadvantages of the proposed solution*

Naturally, in addition to the advantages, the usage of the idea of mimic defense with the assumptions carries several disadvantages:

- Introduction of additional delay related to operations performed by mimic defense elements (duplication of HTTP requests, sending and receiving responses, comparison, and decision to attack).
- The decision on the occurrence of an attack is made only by 1 Arbitrator. That may raise objections about the credibility of such an assessment. In particular, when an attacker compromises an Arbitrator, the decision made by it may be inappropriate.
- The decision about the existence of an attack is based on a 100% response match. This approach may be too restrictive, considering that each subsequent application version may differ from the previous. Although the tested web applications met this matching criterion, it is worth considering its mitigation to the selected response compliance threshold.
- At this early research stage, a simple comparison of responses will not detect all occurrences of an attack (e.g., the returned HTTP answers are the same but can contain the result of the attack). Therefore, in the next phase of research, the efficiency of attack detection by the Arbitrator should be improved.

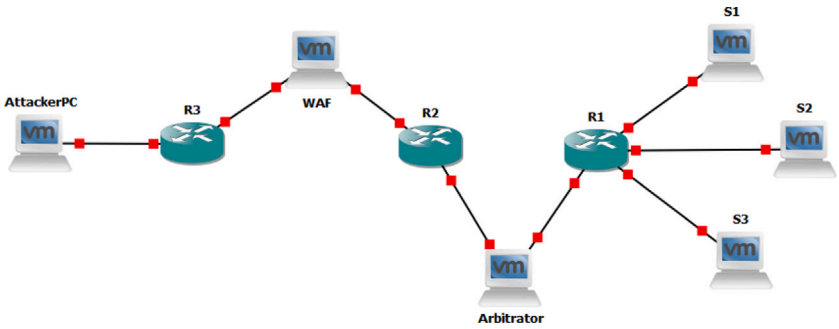

Fig. 6. Test environment with a proposed solution.

Table 3
Configuration of the mimic defense web servers.

	S_1	S_2	S_3
OS type	openSUSE 15.0	Fedora 31	Ubuntu 16.04.06
HTTP server type	Apache	OpenLiteSpeed	nginx
PHP version	php 7.0.5	php 7.3.13	php 7.2.5
DB version	10.2.14-MariaDB	10.3.27-MariaDB	mysql 5.7.33

5.3. Description of the test environment implementing the described concept

The concept discussed in the previous section has been implemented as a virtual environment in the GNS3 tool (see Fig. 6), which is a graphical network emulator that allows creating and testing networks consisting of virtual hardware.

ModSec was chosen as WAF in the HTTP reverse-proxy and uses available rules with signatures to detect web attacks. WAF ModSecurity was powered by rules from the OWASP ModSecurity Core Rule Set (rule set version 3.3.2 was used during the experiments). Malicious traffic detection accuracy was configured with a paranoia level parameter. The higher the paranoia level, the more rules are activated and hence the more aggressive the Core Rule Set is, offering more protection but potentially causing more false positives. That is why the paranoia level was set to 2. Another parameter illustrating the quality of attack detection is the anomaly score threshold. An anomaly score threshold is the cumulative anomaly score at which an inbound request or an outbound response will be blocked. When setting the value of this parameter, it should be kept in mind that increasing the anomaly score threshold makes the CRS less sensitive and less likely to block transactions. Therefore, an inbound anomaly score threshold was set to 5 and an outbound anomaly score threshold was to 4. An attacker PC was selected as a virtual machine with the Kali Linux operating system, which is perfect for performing security tests, especially website tests. The Arbitrator module was written in java. The choice of language was dictated by the pretty good speed of data processing (slower than C and C ++ languages, but still suitable for real-time task execution applications) (Nikishkov et al., 2003; Suzumura et al., 2008). The S1, S2, and S3 servers are three instances of the same application (three executors), consisting of different components. In particular, each server had a different version of the web application running, but, from the point of view of the operation, i.e., the application code sent to be displayed by the browser, it was the same application. Configuration details of S1, S2, and S3 servers are presented in Table 3.

6. Tests and results

6.1. Test scenarios

The definition of the test scenarios was conditioned by two factors: the selection of the appropriate set of widespread attacks on web applications and the existing content management solutions that will be available in several subsequent versions. In the context of the first criterion, the following groups of attacks (OWASP, 2022) and exemplary implementations were decided on:

1. Injections
   - Cross-site scripting (XSS) — a vulnerability associated with the injection of malicious javascript code in the data provided by the user; can be in the variant reflected and stored
   - SQL Injection — a vulnerability associated with the injection of special SQL statements intentionally changing the overall meaning of a new query
   - XML External Entity — a vulnerability that enables an attacker to interfere with processing XML data (parsing XML input) within a www application
2. Unauthorized access to files
   - Path traversal — a vulnerability that allows an attacker to read any files on the server where the web application is running
3. Data leakage — obtaining information that should not be available to every user
4. Information gathering — obtaining valuable data allowing the attacker to create new attack scenarios

While selecting attacks is easier, choosing appropriate CMSs that meet the relevant conditions is not. Usually, if a web application has a vulnerability, it is fixed in its next version. The new version is made public, and the old one is removed; thus, not many people can use it. It often happens that the detected vulnerability is still present even after a few repairs. In this case, the vulnerable component or function is totally removed from the application. One of the essential requirements in the experiment was to find CMSs that have several versions (diversity to mimic defense); additionally, each subsequent version is not significantly different from the other. The following open-source CMSs were selected: GetSimple CMS (Open source simple, 2022), Piwigo (Open source photo, 2022), and OpenCATS (Open source candidate, 2022), which are publicly available in subsequent versions, and each of them improves the security flaws present in the previous variant. These applications were installed on servers in the test environment in three different versions with various additional components (e.g., database, PHP version, etc.) shown in Table 3.

The test cases covered by the experiments can be divided into the following groups:

- attack detection tests with and without the use of mimic defense elements,
- predictability tests of responses related to the selection of executors,
- request handling time tests with and without mimic defense elements.

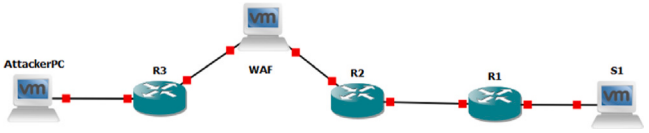

Fig. 7. Test environment for measuring the average HTTP request service time without mimic defense protection.

Table 4
The results of detecting an attack using the Mimic defense concept.

Type of attack	Only WAF detection	WAF & Mimic defense detection
Reflected cross site scripting	×	✓
Stored cross site scripting	×	✓
Path traversal	×	×
SQL Injection	×	✓
Disclosure of sensitive data	×	✓
Information gathering	×	✓
XML external entity	×	✓

6.2. Attacks detection tests

The first test case referred to the results of malicious web traffic detection in two scenarios. The first was related to the network from Fig. 6 which contained mimic defense components. The second simplified network diagram had only one application server and did not include the Arbitrator module, so in this example, an application server was protected only by WAF (see Fig. 7). For each test scenario, a malicious payload was sent from the selected category, and the object of observation was whether it was classified as an attack attempt.

The collected test results are in Table 4. As can be seen, in the case of the protection provided only by WAF (first column of the table), attempts at attacks were never detected. Such a situation does not mean that the application firewall was not working or was not detecting any attacks at all. From the attacker's point of view, simple attack attempts are usually noticed, so they should be made more difficult to detect, for example, by using obfuscation techniques. In this particular case, the detection was bypassed, and the introduced traffic was treated as allowed traffic. In turn, the second column contains the results for the case with the protection ensured by WAF and mimic defense elements and, as can be seen, six attack attempts were detected and blocked. It is quite a significant improvement in the detection of attack attempts compared to the variant with protection provided only by the application firewall.

In addition to detecting and blocking an attempted attack, the Arbitrator module has created an appropriate rule to detect malicious traffic. Apart from the attack pattern, the rule took into account the type of HTTP request (GET or POST) and the kind of action to be performed when WAF detects the pattern (HTTP 403 or 404 response). The created rule was then sent to the WAF server and added to the set of user rules. During another attempt to bypass application firewall security using the same obfuscated payload, malicious traffic was detected, blocked, and not even passed to the Arbitrator module.

6.3. Mimic defense responses predictability experiments

The next phase examined the choice of servers to which the request would be sent. In particular, it analyzed how such a choice changed over time in handling a new query. In other words, it is checked whether they will be sent to the same selected executors (servers) by sending queries one after another. The following cases were investigated:

- the total number of duplicate occurrences,
- the maximum number of events of identical duplicates,

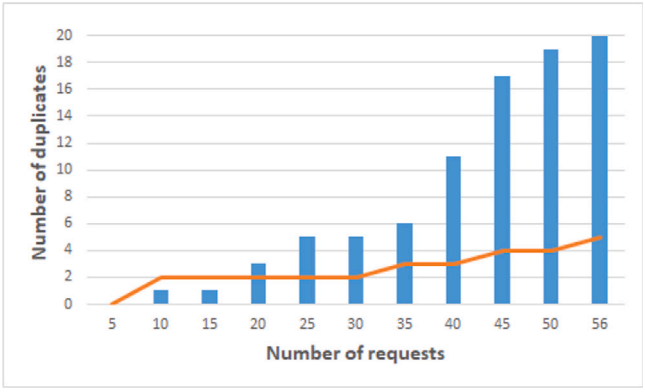

Fig. 8. Number of duplicates of the selected servers.

- maximal series length for duplicates.

The test environment consisted of 8 different executors (application servers) made of various components. Based on the established strategy, the Arbitrator decided which executors would be requested at the selected time. The strategy indicated that the Arbitrator must choose without repeating three executors in 8. The maximum number of 3 permutations with eight servers is 56. In 11 rounds, 5, 10, 15, 20, 25, 30, 35, 40, 45, 50, and 56 requests were sent one by one (one request in one moment) in sequence, observing how many sets of three randomly selected executors repeat. The aggregated results were plotted on Fig. 8. For the initial series of requests sent, the number of duplicate queries did not exceed 20%. Of course, the more such requests, the greater the number of duplicates (up to about 38%). Such a situation appears because of a small number of three executors' combinations which can be drawn — at some point, all combinations will be drawn at least once.

Additionally, a line graph was drawn in Fig. 8, which concerned the occurrence of the maximum number of identical duplicates (number of the same servers sets that were duplicated). As can be observed, this number value equals a maximum of 13% of all queries, usually less than 10%. From a security point of view, this is a very good result because the attacker has little chance of getting the same results under the same conditions (for the same executors). In other words, the probability that an attacker on many sent HTTP requests will receive the same responses from the same sets of executors is small. This unpredictability is possible due to the Arbitrator's set selection strategy. Thus, we can speak of unpredictability and uncertainty, making launching and repeating a successful attack challenging.

Mimic defense technic assumes that for every request at a different moment, the responses should come from different application servers — this introduces uncertainty of the result. In practice, this means that even if the attacker has successfully verified the existence of the vulnerability, it is highly probable that he will not be able to repeat this issue. The attacker is unaware of which servers the HTTP request was sent to, and the decision was made. The attacker sees only the final answer (response), and the entire mimic defense structure is transparent to him/her. In addition, examining the unpredictability of server selection, the Arbitrator's choice series of the same three servers were tested. The experiment was carried out 50 times, the selected results are shown in Table 5 and detailed in the Figs. 9–19. Three servers were drawn each time an HTTP request was sent in the round. Throughout the round, there could be series, i.e. situations where the same three servers were drawn in a row. Each selected three servers is treated as series with a length of 1 (a default series). A sequence consisting of the series length values was created for each round according to the following rules:

- For different triples of servers, a series value of 1 was assigned. For example, the triple of **S1S2S3, S2S3S4, and S3S4S5** was set to the value **1 1 1**

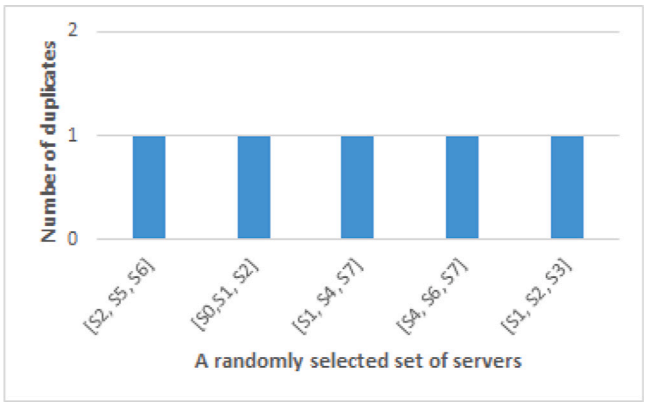

Fig. 9. Number of unique server sets for 5 requests sent.

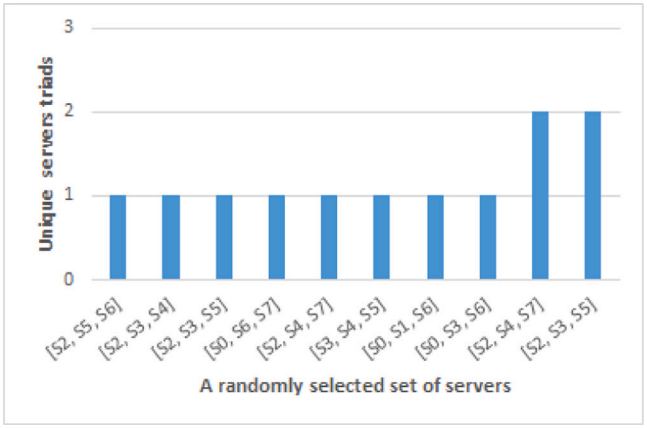

Fig. 10. Number of unique server sets for 10 requests sent.

Fig. 11. Number of unique server sets for 15 requests sent.

- For the repeating triples of servers, the value equal to the number of repetitions in the series was assigned. For example, the triple **S1S2S3, S1S2S3, S3S4S5** was set to the value **2 1**

The series occurrence is much more dangerous from the point of view of security. A series is a situation where the same triad of servers will be drawn immediately after each other (the same trio occurs when selecting servers S1, S2, S3, and S3, S2, and S1). Thus, the Arbitrator's server selection algorithm must give as few and as short bursts as possible. Analyzing the results from Table 5, it can be seen that in only two rounds, there were series greater than 1. The value of the coefficient of variation parameter for rounds with 46 and 56 requests

Fig. 12. Number of unique server sets for 20 requests sent.

Fig. 13. Number of unique server sets for 25 requests sent.

Table 5
The selected results of the unpredictability of the selection of executors.

Number of requests in a round	5	10	15	20	25	30	35	40	45	50	56
Number of series in a round	5	10	15	20	25	30	35	40	**44**	50	**55**
Standard deviation	0	0	0	0	0	0	0	0	**0.150**	0	**0.135**
Arithmetic average	1	1	1	1	1	1	1	1	**1.022**	1	**1.018**
Coefficient of variation [%]	0	0	0	0	0	0	0	0	**14.74**	0	**13.24**

was less than 15%. In practice, this means that the number of series was small and their length was short (in this case, equal to 2, see Figs. 17 and 19). In all 50 iterations of the trial, the most extended series that appeared was 3, and all series were less than 6% of all the repeating server triads.

6.4. Request handling time experiments

It is natural that supplementing additional network elements that process the traffic introduces further delay. Therefore, it was examined what time overheads are added by the proposed elements of the mimic defense in a created environment. For comparison, the times in two other cases were also discussed. The first one is a case where the network includes the Arbitrator, but it acts as a proxy (see Fig. 20) – it forwards traffic to the selected application server and receives responses without any analysis. The second one has no mimic defense

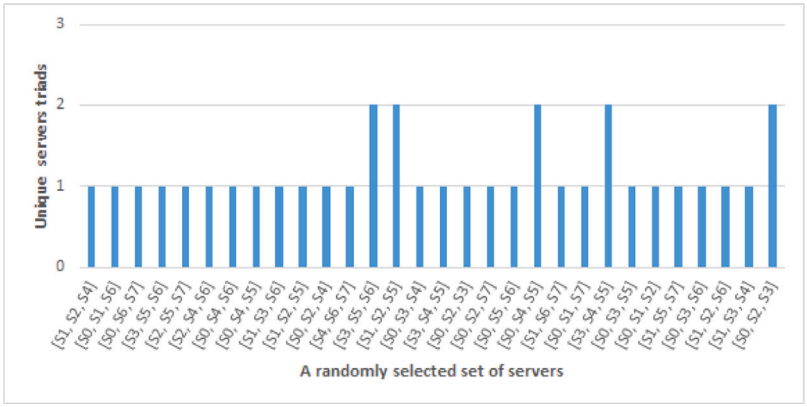

Fig. 14. Number of unique server sets for 30 requests sent.

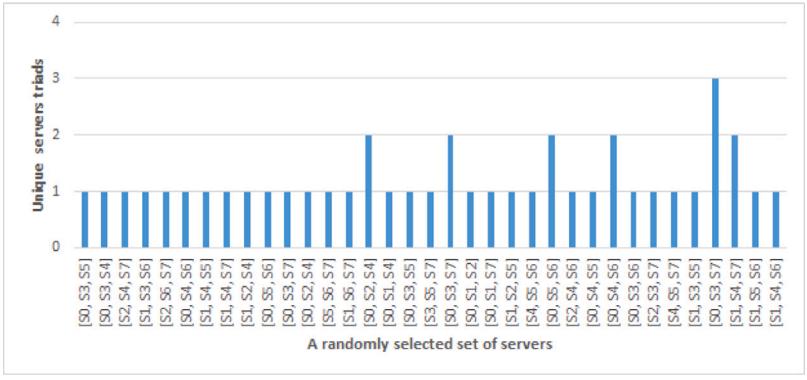

Fig. 15. Number of unique server sets for 35 requests sent.

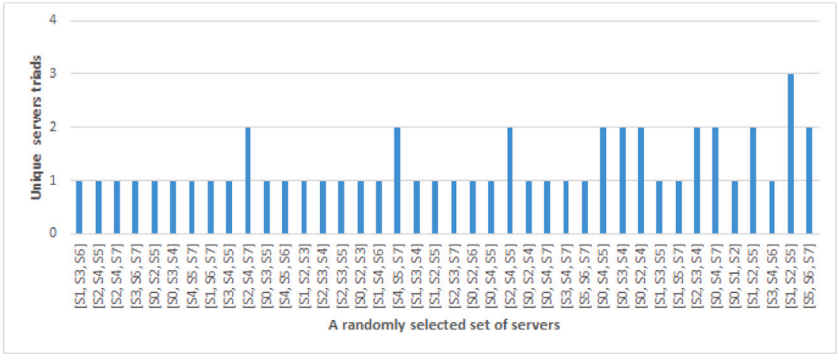

Fig. 16. Number of unique server sets for 40 requests sent.

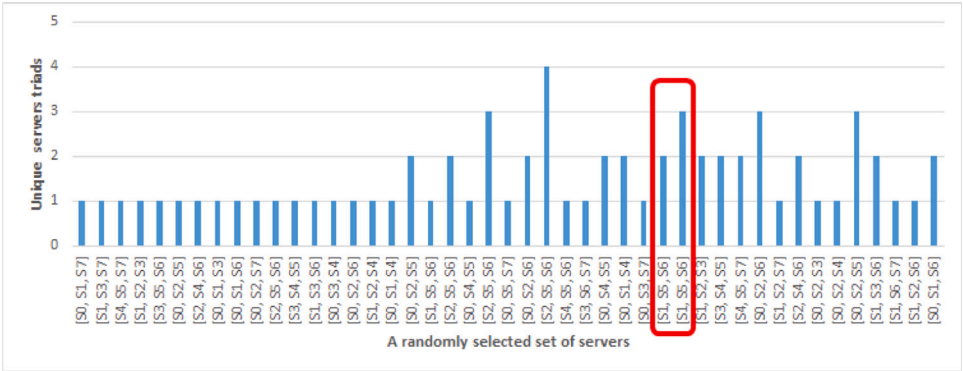

Fig. 17. Number of unique server sets for 45 requests sent.

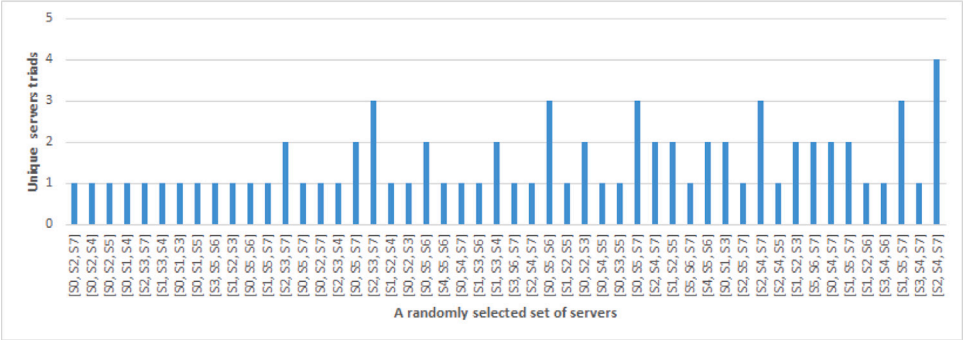

Fig. 18. Number of unique server sets for 50 requests sent.

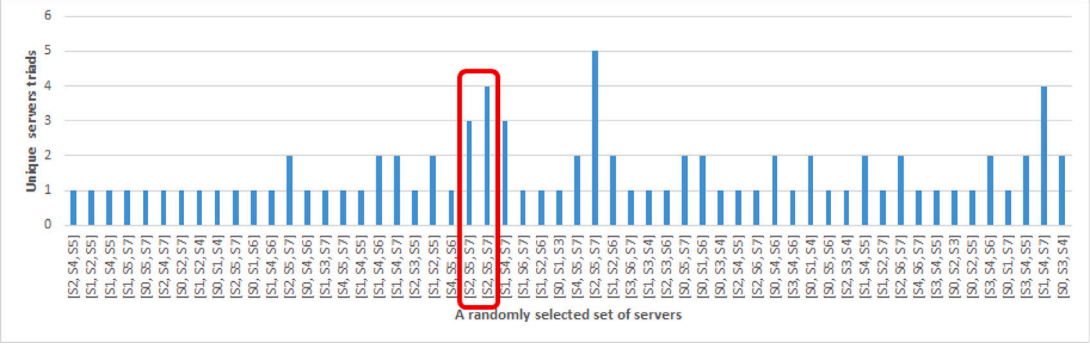

Fig. 19. Number of unique server sets for 56 requests sent.

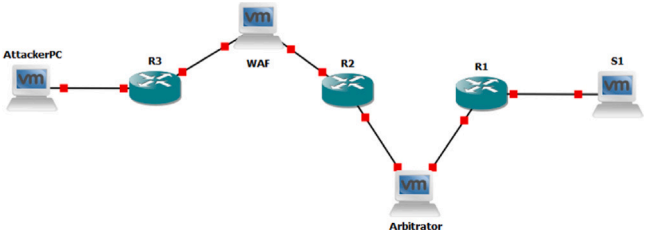

Fig. 20. Test environment for measuring the average HTTP request service time with Arbitrator-only element.

Table 6
Average requests handling time in three cases: MD (the network protected by WAF and mimic defense), only Arbitrator, without MD (network protected only by WAF).

No. requests	Average times [ms]		
	MD	Arbitrator only	Without MD
10	232.9	128.2	98.2
50	267.8	184.7	89.2
100	247.4	164.6	86.5
200	227.4	126.9	87.8
500	235.8	120.2	85.2
1000	239.7	133.7	85.6

elements (see Fig. 7) - the application server is protected only by the application firewall.

The experiment consisted in sending a series of requests (10, 50, 100, 200, 500, 1000), receiving the response, measuring the time from sending the request to receiving the response, and calculating the average in a given series. The obtained results are presented in Table 6; additionally, they are shown in Fig. 21. Besides, the parameters of the virtual machine on which the Arbitrator is running are 2CPU, 4 GB RAM, and 100 GB free disk-space.

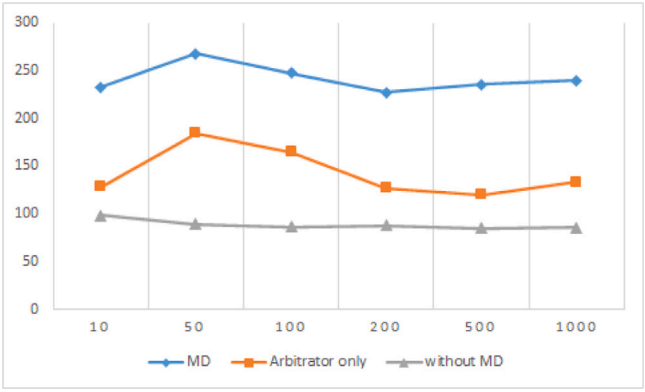

Fig. 21. Comparison of the requests handling time.

As expected, the longest request handling time is in the variant with mimic defense elements. This is because additional operations are introduced to handle the request, such as duplicating and sending it to three application servers, receiving all responses, comparing the responses, and deciding on the answer returned to the user. The chart shows that the difference in times between the with-MD and without-MD scenarios varies between 130 and 180 ms, which does not significantly impact the user's perception of the quality of service (it is acceptable Nah, 2004). In addition, it can be noticed that when in a network with an installed Arbitrator module, which acts as a proxy (receives requests, sends them after appropriate modification, receives responses, and forwards modified accordingly), the element introduces a delay between 30 and 100 ms. Therefore, the time of comparing the obtained responses is about 100 ms (including the Arbitrator's decision-making time from 20 ms to 60 ms).

7. Discussion

The obtained results confirm the effectiveness and legitimacy of using mimic defense to support the detection of attacks provided by WAF. Nevertheless, it is justified to consider when and under what conditions the usage of mimic protection is beneficial.

The classic model of DHR architecture has an Arbitrator module, but it does not indicate the number of such entities. The proposed solutions use only one of those in the experiment (a similar approach can be found in cited publications). Such a concept means it becomes a bottleneck of the solution regarding security and performance. In terms of security, a successful attack may distort the accuracy of the decisions made by the Arbitrator. Moreover, one may doubt whether one decision-making element is sufficient and can be trusted, or whether such decisions should be made with the use of several independent Arbitrator modules. From the point of view of network performance, the Arbitrator component must have adequate computing resources to receive numerous incoming inquiries. Lack of appropriate resources will result in the rejection of requests, significantly impacting the operation of the www service and user experience. In this case, it is worth considering a mechanism that bypasses the elements of mimic defense and, at the same time, ensures the continuity of the service.

Another challenge for mimic defense is the number of executors based on which the decision is to be made. Three application servers were used in the experiment to find out which server has vulnerable components. Two servers do not give a proper indication, and three are the minimum. For three servers, the combination of responses from the servers could be as follows: three responses were the same, two responses were the same, and each response was different. In the research described in the article, the Arbitrator received three or two of the same answers. Since it was known which versions of the web service were vulnerable in advance, it was easier to determine whether the three obtained responses indicated a lack of vulnerability. For there may be a situation where all three answers will be the same, but they will have the effect of using a vulnerability successfully. Such a situation occurred when detecting a Path traversal vulnerability for GetSimple CMS. All versions of this CMS were susceptible to this vulnerability, so they gave the same response, which was interpreted correctly by the Arbitrator module. In a situation where the two answers are the same, there is a high probability that we are dealing with an attempted attack. The problem arises in determining which responses contain attack content (unique responses from one server or identical responses from two different servers). In the case of three different responses, the Arbitrator can again not determine which server is vulnerable with a simple verification. A particular method of resolving the indicated doubts may be the increased number of servers based on which a decision is made. In particular, this approach can eliminate many situations where all the answers are identical. Increasing the number of servers to which the HTTP request is sent may lead to a situation where one new server will give a completely different response than the others and allow us to assume that the sent HTTP request has a malicious payload. Naturally, adding new servers to the general pool at a given moment increases the time needed to analyze the response and decide. Therefore, the appropriate selection of the number of executors should be considered at a given moment.

8. Conclusion and future works

This paper shows the concept of a Web Application Firewall supported by the Cyber Mimic Defense approach. Firstly, the advantages and disadvantages of the basic methods of operation of a web application firewall were discussed. Secondly, the concept of mimic defense was introduced to increase the level of detection of attack attempts offered by WAF. Then, a network architecture protected by WAF and including elements of mimic defense was proposed, and finally, experiments were carried out. The obtained results indicate the legitimacy of using WAF with elements of the mimic defense. The conducted research scenarios showed that the mimic defense mechanism significantly increases the detection level of attack attempts compared to the defense provided by the signature-based application firewall. In addition, it dynamically supplies WAF with new rules that block previously unknown attacks, thanks to which the application firewall will block any such attempt, thus limiting the actions taken by mimic defense elements. At the same time, the impact of this mechanism on the handling time of HTTP requests was shown. Moreover, some limitations of the approach are presented, which may form the basis for further research in this area.

In the future, the potential use of the presented protection approach provided by WAF and mimic defense in microservice environments will be explored. Such an idea is an interesting issue in the context of the assumptions of this environment, in which one microservice can be updated several times a day, and the classic WAF approach to learning traffic related to a microservice-based application becomes very limited.

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.

Data availability

Data will be made available on request.

Acknowledgments

Research was funded by (POB Cybersecurity and data analysis) of Warsaw University of Technology, Poland within the Excellence Initiative: Research University (IDUB) programme.

References

Achleitner, S., et al., 2017. Deceiving network reconnaissance using SDN-based virtual topologies. IEEE Trans. Netw. Serv. Manage. 14 (4), 1098–1112. http://dx.doi.org/10.1109/TNSM.2017.2724239.

Ahmad, A., et al., 2012. Formal reasoning of web application firewall rules through ontological modeling. In: 2012 15th International Multitopic Conference. INMIC, pp. 230–237. http://dx.doi.org/10.1109/INMIC.2012.6511505.

2022. Akamai web application firewall. https://www.akamai.com/products/web-application-protector, (access date: 20.12.2022).

Al-Shaer, E., et al., 2012. Random host mutation for moving target defense. In: Proc. Int. Conf. Security Privacy Commun. Syst.. pp. 310–327. http://dx.doi.org/10.1007/978-3-642-36883-7_19.

Antonatos, S., et al., 2007. Defending against hitlist worms using network address space randomization. Comput. Netw. 51 (12), 3471–3490. http://dx.doi.org/10.1145/1103626.1103633.

Appelt, D., Nguyen, C.D., Panichella, A., Briand, L.C., 2018. A machine-learning-driven evolutionary approach for testing web application firewalls. IEEE Trans. Reliab. 67 (3), 733–757. http://dx.doi.org/10.1109/TR.2018.2805763.

Applebaum, S., et al., 2021. Signature-based and machine-learning-based web application firewalls: A short survey. Procedia Comput. Sci. 189, 359–367. http://dx.doi.org/10.1016/j.procs.2021.05.105.

2022. AWS web application firewall. https://aws.amazon.com/waf/, (access date: 20.12.2022).

Aydeger, A., et al., 2016. Mitigating crossfire attacks using SDN-based moving target defense. In: Proc. IEEE Conf. Local Comput. Netw.. pp. 627–630. http://dx.doi.org/10.1109/LCN.2016.108.

Bangalore, A.K., Sood, A.K., 2009. Securing web servers using self cleansing intrusion tolerance (SCIT). In: 2009 Second International Conference on Dependability. pp. 60–65. http://dx.doi.org/10.1109/DEPEND.2009.15.

Betarte, G., et al., 2018. Improving web application firewalls through anomaly detection. In: 2018 17th IEEE International Conference on Machine Learning and Applications. ICMLA, pp. 779–784. http://dx.doi.org/10.1109/ICMLA.2018.00124.

Castro, M., Liskov, B., 2003. Practical Byzantine fault tolerance and proactive recovery. ACM Trans. Comput. Syst. (TOCS) 20 (4), 398–461. http://dx.doi.org/10.1145/571637.571640.

2022. Cloudflare web application firewall. https://www.cloudflare.com/waf/, (access date: 20.12.2022).

Duan, Q., et al., 2013. Efficient random route mutation considering flow and network constraints. In: Proc. IEEE Conf. Commun. Netw. Security (CNS). pp. 260–268. http://dx.doi.org/10.1109/CNS.2013.6682715.

2022. F5 web application firewall. https://www.f5.com/products/security/advanced-waf, (access date: 20.12.2022).

Fan, Y., et al., 2018. A new method of image encryption based on mimic defense. In: 2018 10th International Conference on Communications, Circuits and Systems. ICCCAS, pp. 418–421. http://dx.doi.org/10.1109/ICCCAS.2018.8768938, December.

Guodong, W., et al., 2009. An efficient forward recovery checkpointing scheme in dissimilar redundancy computer system. In: 2009 International Conference on Computational Intelligence and Software Engineering. http://dx.doi.org/10.1109/CISE.2009.5366252.

Hanlon, R.T., et al., 2010. A mimic octopus in the atlantic: flatfish mimicry and camouflage by macrotritopus defilippi. Biol. Bull. 218 (1), 15–24. http://dx.doi.org/10.1086/bblv218n1p15.

Hu, H., et al., 2016. Mimic defense: a designed-in cybersecurity defense framework. IET Inf. Secur. 12 (3), 226–237. http://dx.doi.org/10.1049/iet-ifs.2017.0086.

Hu, H., et al., 2018. Mimic defense: a designed-in cybersecurity defense framework. IET Inf. Secur. 12 (3), 226–237. http://dx.doi.org/10.1049/iet-ifs.2017.0086.

2022. Imperva web application firewall. https://www.imperva.com/products/web-application-firewall-waf/, (access date: 20.12.2022).

2022. IronBee web application firewall. https://github.com/ironbee, (access date: 20.12.2022).

Jia, Q., Sun, K., Stavrou, A., 2013. MOTAG: Moving target defense against internet denial of service attacks. In: Proc. 22nd Int. Conf. Comput. Commun. Netw.. ICCCN, pp. 1–9. http://dx.doi.org/10.1109/ICCCN.2013.6614155.

Kil, C., et al., 2006. Address space layout permutation (ASLP): Towards fine-grained randomization of commodity software. In: Proc. 22nd Annu. Comput. Security Appl. Conf.. ACSAC, pp. 339–348. http://dx.doi.org/10.1109/ACSAC.2006.9.

Kruege, T., 2010. TokDoc: a self-healing web application firewall. In: Proceedings of the 2010 ACM Symposium on Applied Computing (SAC '10). Association for Computing Machinery, New York, NY, USA, pp. 1846–1853. http://dx.doi.org/10.1145/1774088.1774480.

Kuang, X., et al., 2015. Design of airborne electrical load management center with high reliability based on dissimilar redundant technique. In: 2015 18th International Conference on Electrical Machines and Systems. ICEMS, IEEE, pp. 1198–1203. http://dx.doi.org/10.1109/ICEMS.2015.7385221.

Lala, Jaynarayan H. (Ed.), 2003. Foundations of intrusion tolerant systems. In: DARPA Organically Assured and Survivable Information Systems. OASIS, IEEE Computer Press, http://dx.doi.org/10.1109/FITS.2003.1264922.

Li, B., et al., 2018. Mimic computing for password recovery. Future Gener. Comput. Syst. 24, 58–77. http://dx.doi.org/10.1016/j.future.2018.02.018.

Li, G., et al., 2021. A framework for mimic defense system in cyberspace. J. Signal Process. Syst. 93, 169–185, doi:https://link.springer.com/article/10.1007/s11265-019-01473-6.

Liang, J., Wen. Y. Wei, Z., 2017. Anomaly-based web attack detection: a deep learning approach. In: Proceedings of the 2017 VI International Conference on Network, Communication and Computing. pp. 80–85. http://dx.doi.org/10.1145/3171592.3171594.

Lin, Z., et al., 2017. MDFS: A mimic defense theory based architecture for distributed file system. In: 2017 IEEE International Conference on Big Data (Big Data). pp. 2670–2675. http://dx.doi.org/10.1109/BigData.2017.8258229, December.

Malkhi, D., et al., 2001. Persistent objects in the fleet system. In: Proceedings DARPA Information Survivability Conference and Exposition II. DISCEX'01, Vol. 2, June. pp. 126–136. http://dx.doi.org/10.1109/DISCEX.2001.932165.

2022. ModSecurity web application firewall. https://github.com/SpiderLabs/ModSecurity, (access date: 20.12.2022).

Moosa, A., 2010. Artificial neural network based web application firewall for SQL injection. Int. J. Comput. Inf. Eng. 4 (4), 610–619. http://dx.doi.org/10.5281/zenodo.1329474.

Nah, F.F., 2004. A study on tolerable waiting time: how long are web users willing to wait? Behav. Inf. Technol. 23 (3), 153–163. http://dx.doi.org/10.1080/01449290410001669914.

2022. NAXSI web application firewall. https://github.com/nbs-system/naxsi, (access date: 20.12.2022).

Neti, S., et al., 2012. Software diversity: Security, entropy and game theory. In: Proc. USENIX Summit Hot Topics Security.

Nikishkov, G.P., et al., 2003. Comparison of C and java performance in finite element computations. Comput. Struct. 81 (24–25), 2401–2408. http://dx.doi.org/10.1016/S0045-7949(03)00301-8.

2022. Open source candidate/applicant tracking system - OpenCATS. https://www.opencats.org/ (access date: 17.07.2022).

2022. Open source photo management software - Piwigo. https://piwigo.org/ (access date: 17.07.2022).

2022. Open source simple CMS - GetSimple CMS. http://get-simple.info/ (access date: 17.07.2022).

OWASP, 2022. Web application security testing cheat sheet. https://cheatsheetseries.owasp.org/cheatsheets/Web_Service_Security_Cheat_Sheet.html (access date: 17.07.2022).

Palka, D., Zachara, M., 2011a. Learning web application firewall-benefits and caveats. In: International Conference on Availability, Reliability, and Security. Springer, Berlin, Heidelberg, http://dx.doi.org/10.1007/978-3-642-23300-5_23.

Palka, D., Zachara, M., 2011b. Learning web application firewall-benefits and caveats. In: International Conference on Availability, Reliability, and Security. Springer, Berlin, Heidelberg, pp. 295–308. http://dx.doi.org/10.1007/978-3-642-23300-5_23.

Prokhorenko, V., et al., 2016. Web application protection techniques: A taxonomy. J. Netw. Comput. Appl. 60, 95–112. http://dx.doi.org/10.1016/j.jnca.2015.11.017.

Razzaq, A., et al., 2013. Critical analysis on web application firewall solutions. In: 2013 IEEE Eleventh International Symposium on Autonomous Decentralized Systems. ISADS, IEEE, http://dx.doi.org/10.1109/ISADS.2013.6513431.

Sang, X., Li, Q., 2019. Mimic defense techniques of edge-computing terminal. In: 2019 IEEE Fifth International Conference on Big Data Computing Service and Applications (BigDataService). pp. 247–251. http://dx.doi.org/10.1109/BigDataService.2019.00043, April.

Sengupta, S., et al., 2017. A game theoretic approach to strategy generation for moving target defense in web applications. In: Proc. 16th Conf. Auton. Agents Multiagent Syst.. pp. 178–186. http://dx.doi.org/10.5555/3091125.3091155.

Shacham, H., et al., 2004. On the effectiveness of address-space randomization. In: Proc. 11th ACM Conf. Comput. Commun. Security. pp. 298–307. http://dx.doi.org/10.1145/1030083.1030124.

2022. Shadow deamon web application firewall. https://shadowd.zecure.org/overview/introduction/, (access date: 20.12.2022).

Suzumura, T., et al., 2008. Performance comparison of web service engines in php, java and c. In: 2008 IEEE International Conference on Web Services. IEEE, http://dx.doi.org/10.1109/ICWS.2008.71.

Tekerek, A.D.E.M., Bay, O.F., 2019. Design and implementation of an artificial intelligence-based web application firewall model. Neural Netw. World 29 (4), 189–206. http://dx.doi.org/10.14311/NNW.2019.29.013.

Torrano-Gimenez, C., Perez-Villegas, A., Alvarez, G., 2009. A self-learning anomaly-based web application firewall. In: Computational Intelligence in Security for Information Systems. Springer, Berlin, Heidelberg, pp. 85–92. http://dx.doi.org/10.1007/978-3-642-04091-7_11.

Vartouni, A.M., Teshnehlab, M., Kashi, S.S., 2019. Leveraging deep neural networks for anomaly-based web application firewall. IET Inf. Secur. 13 (4), 352–361. http://dx.doi.org/10.1049/iet-ifs.2018.5404.

Wang, F., et al., 2003a. SITAR: a scalable intrusion-tolerant architecture for distributed services. Found. Intrusion Toler. Syst. 359–367. http://dx.doi.org/10.1109/DISCEX.2003.1194957.

Wang, D., et al., 2003b. Security analysis of sitar intrusion tolerance system. In: Proceedings of the 2003 ACM Workshop on Survivable and Self-Regenerative Systems: in Association with 10th ACM Conference on Computer and Communications Security, Fairfax, Virginia. http://dx.doi.org/10.1145/1036921.1036924.

Wang, Z., et al., 2017a. Design and implementation of mimic network operating system. J. Comput. Res. Dev. 54, 2321–2333. http://dx.doi.org/10.7544/issn1000-1239.2017.20170444.

Wang, Z., et al., 2017b. DNS architecture based on mimic security defense. Acta Electron. Sin. 45, 2705–2714. http://dx.doi.org/10.3969/j.issn.0372-2112.2017.11.018, (in Chinese).

Wang, Y., et al., 2018. Scientific workflow execution system based on mimic defense in the cloud environment. Front. Inf. Technol. Electron. Eng. 19, 1522–1536. http://dx.doi.org/10.1631/FITEE.1800621.

2022. WebKnight web application firewall. https://www.iis.net/downloads/community/2016/04/aqtronix-webknight, (access date: 20.12.2022).

Welch, I., et al., 2003. Architectural Analysis of MAFTIA Intrusion Tolerance Capabilities. Technical Report CS-TR-788, University of Newcastle upon Tyne.

Wu, J., 2017. Introduction to Cyberspace Mimic Defense. Science Press.

Wu, J., 2020. Cyberspace mimic defense generalized robust control and endogenous security. In: Wireless Networks. Springer, Cham, http://dx.doi.org/10.1007/978-3-030-29844-9.

Xiao, G., et al., 2006. New field of cryptography: DNA cryptography. Chin. Sci. Bull. 51, 1413–1420. http://dx.doi.org/10.1007/s11434-006-2012-5.

Xu, J., 2021. Research on cyberspace mimic defense based on dynamic heterogeneous redundancy mechanism. J. Comput. Commun. 9 (7), 1–7. http://dx.doi.org/10.4236/jcc.2021.97001.

Yeh, Y.C., 1996. Triple-triple redundant 777 primary flight computer. In: 1996 IEEE Aerospace Applications Conference. Proceedings, Vol. 1. pp. 293–307. http://dx.doi.org/10.1109/AERO.1996.495891.

Yeh, Y.C., 1998. Design considerations in boeing 777 fly-by-wire computers. In: Proceedings Third IEEE International High-Assurance Systems Engineering Symposium (Cat. No. 98EX231). pp. 64–72. http://dx.doi.org/10.1109/HASE.1998.731596.

Zhang, G., et al., 2020. Practical software diversification tool chain for dissimilar redundancy architecture. In: 2020 IEEE 9th Joint International Information Technology and Artificial Intelligence Conference. ITAIC, pp. 1518–1521. http://dx.doi.org/10.1109/ITAIC49862.2020.9338810.

Zhao, K., et al., 2008. Surveys on the intrusion tolerance system. In: Bond, P. (Ed.), Communications and Networking in China. Chinacombiz. In: Communications in Computer and Information Science, vol. 26, Springer, Berlin, Heidelberg, http://dx.doi.org/10.1007/978-3-642-00205-2_11.