Expert Systems With Applications 168 (2021) 114386

Contents lists available at ScienceDirect

Expert Systems With Applications

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

Resolving cross-site scripting attacks through genetic algorithm and reinforcement learning

Iram Tariq, Muddassar Azam Sindhu *, Rabeeh Ayaz Abbasi, Akmal Saeed Khattak, Onaiza Maqbool, Ghazanfar Farooq Siddiqui

Department of Computer Science, Quaid-i-Azam University, Islamabad, 45320, Pakistan

ARTICLE INFO

Keywords:
Cross site scripting (XSS)
Machine learning
Genetic algorithm
Reinforcement learning
Threat intelligence

ABSTRACT

Cross Site Scripting (XSS) is one of the most frequently occurring vulnerability. The impact of XSS can vary from cosmetic to catastrophic damages. However, detection of XSS efficiently is still an open issue. Cross site scripting has been dealt with static and dynamic analysis previously. Both techniques have shortcomings and fail due to frequent variations in XSS payloads. Therefore, in this paper, we have proposed the use of Genetic Algorithm (GA) along with Reinforcement Learning (RL) and threat intelligence to overcome XSS attacks. For validation, the proposed approach is applied on a real dataset of XSS attacks. Results show better performance of our proposed approach when compared to the approaches reported in the literature. In addition to better performance, our method is not only flexible to changes in XSS payloads, but the results are also more understandable to end users. Moreover, our approach shows improvement when the number of attacks is increased.

1. Introduction

Computer software has become indispensable to humans at present due to its vast variety, functionality, and varied target domains. These varied different types of software are developed using different techniques (language, APIs, frameworks and IDEs). These technologies are getting advanced and are beneficial in many ways but on the other hand their use is also prone to security vulnerabilities (Chan et al., 2014; Gupta et al., 2016; zhong Zhang & Chai, 2016). A remarkable increase in dependence of modern society on web applications has been observed in the last two decades. With this increase in the usage of web applications, security can be compromised due to increasing vulnerabilities (Jovanovic et al., 2010; Ndichu et al., 2019). Therefore, automated tools for vulnerability detection are required to overcome erroneous, time consuming and costly manual testing (Jovanovic et al., 2010).

A *vulnerability* is an example of a fault within a program code, which when exploited can cause a violation in a security policy. It is caused by a mistake in design, coding or configuration. A vulnerability not only makes the target system vulnerable but can also expose the attached supporting systems thus making them the focus of its launched attack (Dowd et al., 2006).

One of the most serious and frequently occurring vulnerability and a related web attack is XSS, that can harm both individual privacy and economics (Zhou & Wang, 2019). XSS has been enlisted in every top ten list of OWASP (Open Web Application Security Project) till now. OWASP defines XSS as,

> "Cross site scripting is one of injection type which focuses on injection of malicious scripts to vulnerable and benign websites".

XSS has different flavors from persistent to non-persistent attacks and its impact varies from cosmetic (prompting alert) to catastrophic (stealing cookies by violating users' privacy) damage. There are three types of XSS attacks reported in the literature: stored, reflected and DOM based XSS. But these types are not completely distinct in their behavior, in fact they overlap. Consequently, XSS is considered to have two types which are *client XSS* and *server XSS*. The *client XSS* occurs when the DOM is updated on an unsafe JavaScript call, through the data supplied by the intruder. If the intrusion is from a user then it is called a *"reflected client XSS"*, otherwise, it is called a *"stored client XSS"* in the case of malicious data originating from a web server. Similarly, *server XSS* results because of untrusted data becoming part of the response generated by the server in the form of HTML. The *server XSS* becomes a *"reflected server XSS"* when the source of malicious data is a user. It is termed a *"stored server XSS"* if the generated response originates from a server.

* Corresponding author.
E-mail addresses: iramtariq@cs.qau.edu.pk (I. Tariq), masindhu@qau.edu.pk (M.A. Sindhu), rabbasi@qau.edu.pk (R.A. Abbasi), akhattak@qau.edu.pk (A.S. Khattak), onaiza@qau.edu.pk (O. Maqbool), ghazanfar@qau.edu.pk (G.F. Siddiqui).

https://doi.org/10.1016/j.eswa.2020.114386
Received 7 August 2020; Received in revised form 2 November 2020; Accepted 25 November 2020
Available online 27 November 2020

Different types of techniques have been utilized by researchers in the literature to analyze XSS vulnerabilities (Durai & Priyadharsini, 2014; Harman & O'Hearn, 2018). Static, dynamic and hybrid analyses are the three techniques in this context. In static analysis, the source code is analyzed without its execution. An abstraction of source code is performed to extract vulnerabilities out of it. Generalization is often used for doing abstraction which may result in a static analysis being sound (misses no vulnerability) but not having completeness (reports false positive(s)).

In a dynamic analysis, a program is executed against some input to analyze its runtime behavior. A dynamic analysis cannot be done in an exhaustive manner even for small programs because the set of input test cases can potentially be infinite, therefore, analysis on every possible input of the input domain of the program is impossible. A dynamic analysis at its best, can be complete (approve secure programs) but can never be sound (can never disapprove false vulnerabilities). Apart from soundness, its drawbacks include requiring a continuous working run time environment, a complex software is often required for performing a dynamic analysis and it is costly in terms of the resources consumed. A combination of static analysis and dynamic analysis is called a hybrid analysis. Using a hybrid approach for XSS could merge the drawbacks of both the approaches.

Recently, machine learning approaches have been used to overcome XSS attacks, but these approaches are not adaptable and flexible to novel and heterogeneous XSS attacks (Zhou & Wang, 2019). Therefore, there is a need for an approach that not just improves the precision and recall but is also flexible and effectively adapts to detect new XSS payloads. A *payload* is a code snippet or malicious URL that if not properly sanitized, results in a web application that can be conveniently manipulated by an attacker. In this paper, we propose a model which makes use of a GA, a Statistical Inference mechanism, and threat Intelligence for XSS detection. Additionally, it incorporates machine learning which makes use of static payloads to tune the proposed model for providing protection in a dynamic environment.

The primary contribution of this research is developing a GA along with statistical inference for model construction to prevent XSS. Previously, GAs have been used in static analysis, but in this research a GA is used along with an RL to mitigate XSS. Despite the presumable effectiveness of using RL and statistical inference along with a GA, to the best of our knowledge, this approach has not been reported in the literature for detecting XSS attacks. The main contributions of the research are:

- Using a GA along with statistical inference and RL for XSS attacks detection
- Designing a dynamic cross-over rate
- Proposed approach is adaptive for novel/new XSS payloads
- Comparison of the results with the state-of-the-art methods

2. Motivation of using GA and RL for XSS

In this Section, we discuss the motivation for using the GA, RL for XSS prevention and also provide a review of some other metaheuristic algorithms along with giving the reasons why they were not used in the proposed approach.

2.1. Genetic algorithm

The GA is inspired by Darwin's theory of evolution and it simulates the process of natural selection. Moreover, it can address optimization or search related issues (Hydara et al., 2015). In Marashdih and Zaaba (2016), authors suggest using a GA to overcome XSS for getting better results. Therefore, in Hydara et al. (2015) using a GA for static analysis improves the results, but also produces many false positives. Our proposed technique uses the optimization feature of the GA to search large repositories containing millions of different XSS attacks. The GA identifies the best and most fit chromosomes which results in an optimized solution.

There are many other metaheuristic algorithms like *Differential Evolution* (DE) and *Genetic Programming* (GP) along with a GA. In this paper, we make use of a GA because the proposed chromosome in this study constitutes binary values, which means that either a feature exists in a payload or it is absent. Therefore, one major reason for not choosing the DE is that our chromosomes do not contain continuous values. Moreover, in DE, a mutation occurs before the crossover whereas in the proposed approach if the parents are mutated before the crossover, it can further decrease their fitness value, which would be decreased even further by the crossover operation. Therefore, in the proposed scheme, we have made use of a GA which performs the crossover to ensure that the fitness of a child either remains the same or increases after crossover, which can potentially be increased further after mutation. Similarly, Genetic Programming (GP) is another metaheuristic algorithm, however, it cannot be applied in the current scenario because our solution consists of linear chromosomes of bits and the GP on the contrary, works on a tree structure.

2.2. Reinforcement learning

Reinforcement learning (RL) is a machine learning technique which is used to resolve hard scientific problems (Chen et al., 2019). Through RL, a system learns by interacting with the environment and attains a reward in each interaction. The RL techniques are employed for problems in which it is hard to provide an explicit solution, instead, specifying a reward which can lead towards a solution is easier. For example, in Razzaq et al. (2018), the RL is used by a non-player controller to update its attacking policy by means of a reward which it gets while interacting with a player in an FPS game. Similarly, the RL is used against denial-of-service (DoS) attacks in Peer-to-Peer (P2P) Networks (Dejmal et al., 2008). In the first case, there can be countless actions which can be performed by a player and in the second case several malicious queries can be asked by any peer in the P2P network. Therefore, in both these cases the model learns after interacting with the environment (player actions or malicious queries).

The XSS payloads are heterogeneous in nature and these innovative heterogeneous payloads are continuously being added in online repositories. Therefore, in this proposed work we make our model adaptable to these new payloads (on which our model is not trained previously) by using RL. In Section 5.3, we provide details on how RL is employed in the proposed technique for model tuning.

3. Related work

Daily usage of internet is fraught with many security breaches which the attackers unleash to achieve different motives. In this Section, we review some of the most common types of threats that are faced by different websites on the internet. A common type of threat occurs due to phishing websites which are meant to invade the privacy of common users. In an earlier work (Vrbančič et al., 2018), the authors make use of swarm intelligence algorithms (including bat algorithm, hybrid bat algorithm, and firefly algorithm) for parameter setting of Deep Neural Networks to classify and prevent attacks/threats from phishing websites. However, attacks carried out on legitimate websites for exploiting their vulnerabilities are not uncommon. A lot of research work has been done to identify and prevent the effect of different types of vulnerabilities exposed by attackers on a website. For example, Su and Wassermann (2006) propose an approach, to prevent a *Command Injection Attack* (CIA), by making use of compiler parsing techniques and context free grammars. The "Commix" tool has been developed by the authors of Stasinopoulos et al. (2015) to detect the command injection in three steps. In the first step, the attack vectors are created. In the second step, the vulnerability in an application is detected. In the last step, the application is exploited by different types of attacks.

The output of these attacks is used to declare the target web application as vulnerable or non-vulnerable. A technique for prevention of session fixation attack is proposed by Takamatsu et al. (2012) in which they use simulation of real attacks. Their technique is successful in five real world applications, but it has the drawback of producing false positives. In Everett (2006), the authors discuss the exploitation techniques and preventive measures against attacks on websites that use LDAP for user authentication. Likewise, another vulnerability named SQL injection is precluded by a technique proposed in Shahriar and Zulkernine (2008) which makes use of mutation analysis. All the above-mentioned attacks have their own perils but one of the most frequent and severe attack that cannot just harm user credentials but can also be catastrophic for the entire website is the XSS attack. Therefore, the current study deals with resolving the XSS attacks.

In earlier work, static and dynamic analysis techniques have been extensively employed in the detection and prevention of XSS attacks leading to the development of several tools, methods and algorithms for XSS mitigation.

A comparison of static analysis tools is presented in Wagner et al. (2005), where the authors survey reviews and dynamic testing. One of their findings is that static and dynamic analyses reveal quite different kinds of bugs. They also report the similarities found in bugs detected by static analysis and reviews. A static analysis tool is described in Agosta et al. (2012), which is based on taint analysis along with the essence of symbolic code. Authors of Jovanovic et al. (2006), report another tool for XSS detection, which makes use of data flow analysis for PHP code. In Shar and Tan (2012), the authors propose a two phased method to analyze XSS attacks. The XSS vulnerabilities are detected in the first phase and in the second phase they are removed, however, this approach is also prone to false positives.

Sensitive data flow and context sensitive analysis are employed by Jovanovic et al. (2010) to reveal vulnerable sections of the source code. They use alias analysis in conjunction with taint analysis to find the semantics of the scripting language. There prototype named Pixy can prevent XSS, SQLI and command injection vulnerabilities. Low false positives and high detection speed are the achievements of this paper. However, this approach is for PHP applications only.

Authors in Lalia and Sarah (2018), use script feature analysis to detect server side XSS. They evaluate their proposed approach on only three web applications.

The proposed work in Mohammadi et al. (2016), employs an automated approach to detect XSS for improper encoding of untrusted data. This involves making use of encoding functions to sanitize malicious input and then strings of XSS attacks are automatically generated to evaluate the effectiveness of these encoding functions. The statistics presented in this empirical study show that this approach is not just limited to XSS but is also useful in detecting other injection attacks like SQLIA and CIA. However, their proposed approach is not able to capture all types of XSS.

The work presented in Hydara et al. (2015) uses a Genetic Algorithm (GA) to detect and mitigate XSS in Java based web applications. In the first step of the GA, they employ the path coverage criterion for setting up the initial population. The fitness function in this case depicts the percentage of vulnerable paths (more vulnerable path is the fittest one). They employ the "Roulette wheel selection" for chromosome selection during the crossover operation on chromosomes by setting the probability to 0.5%. Then they perform the mutation operation with a probability of 0.2%. This approach successfully discovers XSS vulnerabilities but is not able to get rid of false positives completely.

Security and XSS vulnerability are the focus of Marashdih and Zaaba (2016) in which the authors do a survey of the prevalent state-of-the-art XSS prevention techniques along with providing probable future directions for developing XSS prevention techniques. Despite an abundance of existing studies in this field, limitations still exist in the latest available techniques which prevent dynamic analysis to obtain accurate results. False positives still contaminate the results of static analysis, and hybrid analysis still faces efficiency issues. They suggest that using a GA can potentially reduce false positives for XSS vulnerability detection.

Like static analysis, many tools exist for XSS mitigation based on a dynamic analysis. KameleonFuzz, developed by Duchene et al. (2014), is one such black-box scanner that generates malicious inputs which are evaluated on an application. KameleonFuzz does an analysis of taint flow in an application to generate malicious payloads. The drawback of this approach is that the application is tested against just those payloads which the tool generates, therefore, it is not able to prevent all kinds of XSS attacks against an application.

In Wang et al. (2017), the authors use a Hidden Markov Model (HMM) to generate a dynamic attack vector. This attack vector is evaluated by its resemblance with the normal payload. A novel approach based on deep learning is presented in Fang et al. (2018). This involves extracting features using Word2vec from XSS payloads to capture the word order information by mapping each payload to a feature vector. They utilize long- and short-term memory in the testing data. This approach also suffers with the problem of false negatives in the results.

A web application can be checked for security vulnerabilities via black-box scanners. Authors in Bau et al. (2010), make use of and compare eight leading black-box scanners to test availability and effectiveness of XSS and SQLI vulnerabilities. To test, they make use of vulnerable web applications, which comprise of known and projected vulnerabilities. They conclude that automated tools are effective but some of them fail to correctly identify vulnerabilities. The tools surveyed by them still need improvements in identifying server XSS.

Variations in XSS payloads cannot be handled with these traditional static and dynamic methods alone because of the modification trends found in XSS payloads with each passing day. Therefore, ML algorithms are utilized to differentiate between malicious and normal payloads. Naïve Bayes, Multilayer Perceptron (MLP), and decision trees have been used on features extracted from web pages in Krishnaveni and Sathiyakumari (2013). However, this approach makes use of a black-box model and the results obtained are incomprehensible to end users. A comparison of the proposed approach with the state-of-the-art is given in Table 1.

The research work in Agosta et al. (2012), Hydara et al. (2015), Jovanovic et al. (2006, 2010), Lalia and Sarah (2018), Mohammadi et al. (2016) and Wagner et al. (2005) focuses on static analysis, but through static analysis they experience the issue of false positives. Authors in Marashdih and Zaaba (2016) suggest that the use of a GA can give better results. In Hydara et al. (2015) a GA is used along with static analysis but this approach is still prone to the issue of completeness. In order to overcome the completeness issue for XSS, dynamic analysis has been employed by Bau et al. (2010), Duchene et al. (2014), Fang et al. (2018) and Wang et al. (2017). Lack of soundness and potentially infinite set of input values testing the system are the limitations of dynamic analysis. Recently, machine learning algorithms have been employed in Krishnaveni and Sathiyakumari (2013) and Zhou and Wang (2019) but the drawback of these proposed ML approaches is that they work on black-box models thus making the obtained results incomprehensible to end users. Moreover, none of these previously proposed approach is adaptable to new XSS payloads.

To summarize the above discussion, we have identified the following research gaps from the literature.

- XSS is still an active and most frequent type of attack. Therefore, a more efficient solution is needed to prevent XSS.
- False positives contaminate the detection of XSS in existing approaches which need to be reduced.
- The existing approaches for XSS face the issue of soundness.
- The existing approaches do not deal with novel/new XSS payloads. An adaptive solution is required for novel/new XSS payloads.

Table 1
Comparison of the proposed approach with the state-of-the-art methods.

Paper	Focus on XSS	Static analysis	Dynamic analysis	Hybrid analysis	Threat intelligence	Machine learning	Genetic algorithm	Results comprehension	Soundness	Adaptive for novel payloads
Agosta et al. (2012)	✓	✓								
Vogt et al. (2007)	✓			✓						
Lalia and Sarah (2018)	✓	✓								
Mohammadi et al. (2016)	✓	✓								
Hydara et al. (2015)	✓	✓					✓			
Duchene et al. (2014)	✓		✓							
Wang et al. (2017)	✓		✓			✓				
Fang et al. (2018)	✓					✓				
Krishnaveni and Sathiyakumari (2013)	✓					✓				
Zhou and Wang (2019)	✓				✓	✓		✓		
Proposed approach	✓				✓	✓	✓	✓	✓	✓

In our proposed approach, we have used a GA for optimization along with RL to make our proposed approach adaptable to new XSS payloads. Our proposed solution has not just reduced false positives but is also sound. Previous approaches are not sound because of not being able to handle the infinite input space issue. This is handled by our proposed approach by means of patterns. The patterns overcome this issue because each pattern comprises numerous payloads and testing a single pattern is similar to testing several payloads residing in that pattern. This is analogous to testing using equivalence classes where member of one equivalence class is a representative of the whole class. Therefore, the proposed approach of testing handles not only the soundness problem but is also less time consuming as it must deal with fewer inputs. Additionally, the proposed approach not just incorporates a white-box model but also produces readily comprehensible results for the end users.

4. Preliminaries

In the following subsections we provide some preliminaries which are necessary to understand the proposed approach presented in Section 5. We start with an introduction of Reinforcement Learning (RL).

4.1. Reinforcement learning (RL)

RL is a machine learning technique in which a model keeps on evolving after interacting with its environment. In the context of vulnerabilities, an environment represents an attack dataset that keeps growing with new payloads added to it. In RL, learning of an agent or a model is based on trial and error interaction with the dynamic environment. In a nutshell, RL evolves the model with the needs of its environment. In our proposed technique, RL will not only provide flexibility but will also aid in minimizing overfitting. In our context, RL will be used to add, remove, or modify the pattern of a payload.

4.2. Threat intelligence

Threat intelligence or cyber threat intelligence is the collection of information or intelligence through human intelligence (HUMINT), open source intelligence (OSINT), intelligence from dark/deep web, or social media intelligence (SOCMINT). This collected information is used as an indicator by an organization to detect and prevent a cyber threat produced by a threat actor (Sari, 2018). We have used OSINT as threat intelligence in this study. There are four different types of OSINT that are open-source information databases (forums and blogs), vulnerability database, malicious domain names and malicious IP addresses (Zhou & Wang, 2019). In this paper, the types of OSINT utilized are malicious domain names and malicious IP addresses. User privacy is protected by preventing them from visiting phishing pages, botnets, and malwares having malicious domain names. Similarly, if a request to our web application is made through a malicious IP address then it is better to prevent it from responding rather than making our web page vulnerable.

Table 2
List of features that form a chromosome.

Feature no.	Feature name	Feature no.	Feature name
1	Length of input	16	Number of onmouseover
2	Number of alert	17	Number of cookie
3	Number of script	18	Number of domain
4	Number of onerror	19	Number of onfocus
5	Number of confirm	20	Number of expression
6	Number of img	21	Number of iframe
7	Number of onload	22	Number of onclick
8	Number of eval	23	Number of single quote mark
9	Number of prompt	24	Number of double quote mark
10	Number of src	25	Number of left angle bracket
11	Number of href	26	Number of right-angle bracket
12	Number of javascript	27	Number of backslant
13	Number of window	28	Number of coma
14	Number of fromcharcode	29	Number of plus
15	Number of document	30	Number of http://, https://, file://

4.3. Feature

A *feature* is an attribute extracted from a payload. Length of input, inclusion of sensitive characters, sensitive words, and redirection links are the four ways by which XSS payloads are deemed different from the normal input. The length of an XSS payload is usually longer when compared to a normal one. This is because the malicious code residing within it makes it enlarged. Besides there are many characters and words that appear in an XSS payload. Alert, eval, script, and < are some of the examples of these sensitive words and characters. Therefore, the appearance time of these words and characters is added as a feature in the proposed approach. Similarly, redirection links may be embedded in a payload to hide the malicious code on the redirection pages. For redirection, some protocols like http and https may appear in a payload. Therefore, the appearance frequency of these protocols is included as a feature in the feature set. Every payload consists of a group of features. Table 2 shows the selected features previously proposed in Zhou and Wang (2019).

4.4. Chromosomes

A chromosome, often called a "genotype", is a collection of features or parameters that demonstrate a solution to a problem which is being solved through a GA. A population is a collection of all chromosomes. A chromosome in the proposed approach is a binary tuple constituting the features listed in Table 2.

4.5. Pattern

The XSS payload is processed and converted into a binary representation. This representation is further separated based on features. Binary representations of these features are redundant and repeated in many payloads and thus are called patterns throughout this paper.

Table 3
Probability of macro chromosomes in data.

Features no.	Macro chromosome	Probability	Features no.	Macro chromosome	Probability
1	11000100011101110010000000000	13.7%	16	11000100011101110010000000000	24.0%
2	11000100011101110010000000000	16.7%	17	10000010000000111110000000000	26.7%
3	11100000000100110100000000000	30.4%	18	10000000010000110110100000000	20.7%
4	10010100110001110110000000000	35.5%	19	11000100011101110010000000000	27.2%
5	10001000010000110100000000010	93.22%	20	10100000001110001101100000000	38.7%
6	10010100110001110110000000000	24.6%	21	10100000000001111101100000000	29.3%
7	11000010000000000110000000000	43.27%	22	11000000000000001110010000000	76.9%
8	10000001000001000010000000000	72%	23	10100000000000110110010000000	59.06%
9	11011000100000110110000000000	21.1%	24	11000000000101110100000010000	32.7%
10	10000000011100110100000000010	24.8%	25	11010100010100110000010100000	15.8%
11	11010100011110110010000000000	37.9%	26	11000010000000000110010010000	22.8%
12	11000000000100110000000000000	17.7%	27	11000000000001110010000001000	55.1%
13	11000010000010110100000000000	29.1%	28	11000000000011110010000000100	6.04%
14	11100000000011110110000000000	20.5%	29	10000000010010111110000000010	5.9%
15	11000100011101110010000000000	24.3%	30	11100000000001111110000000001	21.4%

Patterns that are repeated the greatest number of times within the separated feature dataset are known as macro chromosomes because they occupy a major portion of the dataset. Table 3 shows the macro chromosomes for each feature along with a percentage of the payloads in which the macro chromosome is repeated. In Table 3, the column probability gives the percentage of occurrence of a macro chromosome within a feature file. This shows similarity in payloads and how we can minimize the computation by pattern matching. The macro chromosome for the feature "Number of confirm" comprises 93.22% of all payloads that have the confirm feature on. The lowest probability of occurrence of a macro chromosome is 5.9% which belongs to the "Number of plus" feature. Overall, we can say that if only the macro chromosomes are dealt with then most of the payloads can be correctly identified. Moreover, if only these chromosomes are compromised then this might highly impact the accuracy.

5. Proposed methodology

Our proposed methodology is a combination of a Genetic Algorithm (GA) module, a Statistical Inference module, and a Reinforcement Learning (RL) module. The source code of the implementation is available on the GitHub link.[1] In this section, we will see how these techniques collaborate to prevent XSS attacks. Now we are going to briefly discuss the mechanism involved in declaring a payload vulnerable. First, the best chromosomes are compared for identifying a vulnerability. If differentiation cannot be made, then fit chromosomes are compared such that more differentiating chromosomes are searched first. Most probably a payload will be declared as vulnerable or non-vulnerable in this phase but if the payload still cannot be differentiated then the next phase of statistical inference will be executed. An overview of the proposed approach is shown in Fig. 1.

In Fig. 1, it has been made sure that the payload (to be tested) is searched for vulnerabilities in both the model and the threat intelligence modules. The model contains outputs from both the GA (fit chromosomes, best chromosomes and patterns) and the statistical inference module (Range and Median Chromosome (MC)). The model first checks for vulnerability in outputs from the GA and if it fails to distinguish then statistical inference is used to obtain the vulnerability status of the payload. Both the model and threat intelligence can render the vulnerability status of a payload in binary form. If both or any of these declare it to be vulnerable, then the payload is vulnerable. On the other hand, if both declare it as non-vulnerable only then it is considered as a non-vulnerable payload.

For this purpose, we make use of Boolean variables Tv and Mv. The variable Tv gives the vulnerability status received from threat intelligence module and the variable Mv gives the status received from the model. The variable Tv is true when domain and IP addresses within payloads match with the payload being analyzed. If any of these is true, then the payload is vulnerable. Otherwise, it is non-vulnerable when both are false. The variable Ps shows the status of the vulnerability and is computed as $Ps = Tv \vee Mv$. If Ps is *false*, then it is non-vulnerable otherwise it is vulnerable. Algorithm 3 in Appendix gives the working of the proposed model.

Another important characteristic of the proposed approach is that it is a drill down approach. The best chromosomes are the most generalized form for deciding and comparing our payloads with these require the least computation. If a decision cannot be made through best chromosomes, we drill down to the fit chromosomes. The number of fit chromosomes for a pattern is more than the best chromosomes, and the computation required for comparing payloads with the fit chromosomes is more than that done with the best chromosomes. However, if after comparing the fit chromosomes the final decision is not reached because of an overlapping pattern, then we drill down for knowledge discovery with respect to payloads using statistical inference. When considering the best and fit chromosomes, we are just matching features but in statistical inference we are finding the distance of our payload with both MC_{vul} and MC_{nonvul}. The smaller the distance of the payload chromosome from MC_{vul} the more probable the payload is vulnerable. On the other hand, a payload is declared as non-vulnerable if its distance from MC_{nonvul} is less than that of MC_{vul}. Moreover, the RL part makes our proposed model adaptable to newly discovered malicious patterns. This is done by adding, modifying and deleting a chromosome residing within the model. The Distinguisher, the Declarator, and the Model are the components that are being used by the RL module to upgrade the proposed model. A detailed discussion on the RL module is given in Section 5.3.

5.1. Genetic algorithm

A genetic algorithm consists of a population, computation of a fitness function, parent selection, child formation using crossover and mutation, swapping the least fit individual with the more fit individual and repeating this process on the modified population until the termination condition is met. The flow of the GA used in the proposed work is shown in Fig. A.1 and Algorithm 4 in Appendix provides the pseudocode of the GA used in our work.

Population

The starting point of a GA is a *population*. It comprises of chromosomes or individuals. Our processed training set forms the initial population. Throughout the GA, the population gets updated and keeps iteratively converging to a solution consisting of some target chromosome(s).

[1] https://github.com/IramTariq/XSS-attack-detection.

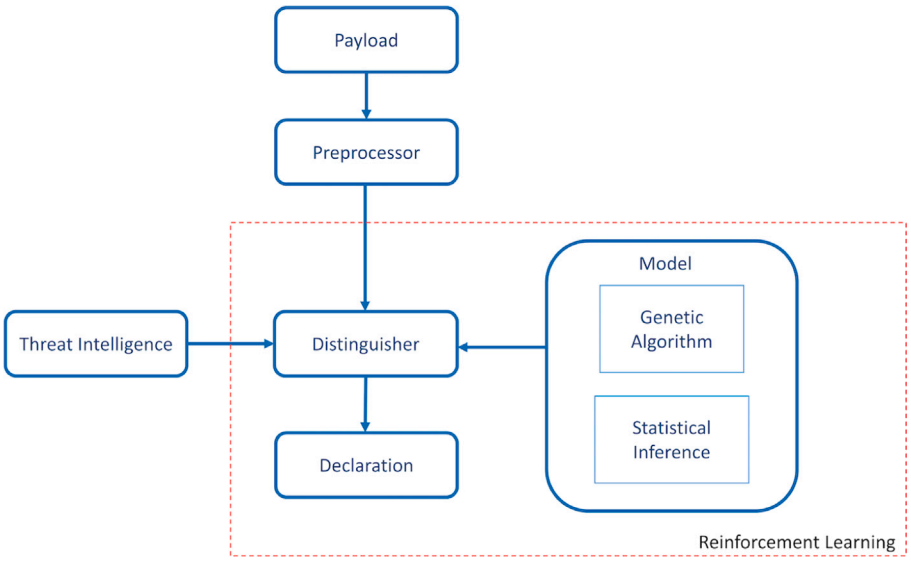

Fig. 1. Overview of the proposed approach.

Fitness function

The fitness is a numeric value that shows how close an individual /chromosome is to a solution. In our proposed technique, the fitness demonstrates the number of constraints satisfied by a chromosome.

Constraints

Constraints of every feature are computed by the following equation:

$$F_i = \frac{\sum_{i=1}^{T} fe_i}{T} \tag{1}$$

where,
T = Total number of payloads
fe_i = Value of a feature within one payload
F_i = Constraint for a feature within the feature set. Its value ranges from 0 to 1.

If F_i is zero, then that feature has no association with that feature set. If the value of F_i is one, then it means that this feature has appeared in all payloads along with the feature around which the data is centered. The value between 0 and 1, has significance equal to those having value 1. The reason is that if out of thousands of malicious records, if only one record has this feature switched on, then we cannot ignore it, since it is malicious and can occur another time. After obtaining F_i values of all the features within a feature file, these values are stored in r_f. Similarly, the rule set R_f is obtained which contains all the values of r_f from one feature file. The process is repeated for all feature's files. The last step involves adding R_f of all feature sets to the constraint set C. This set contains all constraints representing each feature. Since we have 30 features, therefore, the constraint set C constitutes only 30 entities. Each entity is a constraint for a particular feature. The relationship between F_i, R_f, and C is given in (2).

$$F_i \in R_f \in C \tag{2}$$

where, r_f stores the F_i value of a single feature within a feature set. All values of r_f from one file are stored in the rule set R_f and the constraint set C is obtained when R_f values of all features' file become a member of it.

Selection

Selection is the process in which two parents are chosen from a population on which crossover and mutation are applied to create a child. The goal of child creation is to create children which are better in fitness than their parents. For this purpose, selection plays a vital role because if both least fit parents are selected than the child produced will not beat other individuals in fitness and the process to converge to the solution becomes extremely slow (especially when the search space is in millions). In our case, our payloads are sparse and have variations. Therefore, selecting two top fit parents will slow down the convergence process because the payload which has low fitness but is unique and has the potential to take the child closer to convergence, will never get selected and as a consequence, the time to converge to solution will increase. For this reason, usually one most fit and one low fit chromosome are selected to generate a child chromosome. Our approach involves selecting one fittest and one randomly chosen parent whose fitness is lower than the fittest chromosome. Let P, denotes the population of chromosomes obtained after processing training payloads. Parents to perform the crossover operation are selected from this population. The following equations show the characteristics of the selected parents.

$$\begin{aligned} MF &= \prod_{max}(P) \\ LF &= \prod_{min}(P) \\ P_{one} &= MF \\ LF \leq P_{sec} &< P_{one} \end{aligned} \tag{3}$$

Here,
P_{one} = Fitness of the first selected parent
MF = Maximum fitness of the population
P_{sec} = Fitness of the second selected parent. Since P_{sec} is selected randomly so we are not certain about its value. One thing that we can say with certainty about P_{sec} is that its value will always remain less than MF. The reason for randomly selecting the second parent is the sparsity of our payloads. A specific characteristic of some payloads is that they subsume the lowly fit payloads in such a way that fitness of the subsuming payload is more than the lowly fit payload. With random selection if subsuming payloads are selected then it helps in converging the solution which contrasts with choosing a lowly fit individual as a parent.

Crossover

In a GA, the crossover is a phenomenon in which the child is formed after getting genetic information from parent chromosomes. The crossover operation used in this research is of a non-deterministic nature. One of the parents is of maximum fitness $P_{one} = MF$, as it satisfies the maximum number of constraints. Features of P_{one} chromosome that satisfy the constraints are copied to the child chromosomes and the remaining features in the child chromosomes are copied from the randomly selected P_{sec}. After crossover and mutation, if the child's

fitness exceeds the fitness of P_{one} chromosome then in the next iteration, this child replaces P_{one} for crossover. This is because the features that satisfy constraints in a child are different than those in P_{one}. Therefore, crossover points vary in different iterations. Consequently, the crossover rate is non-deterministic, but we can say that if the number of copied features from P_{one} is N and the total number of features within a chromosome is T then, $T-N$ will be the number of features taken from P_{sec}. Therefore, a child chromosome is formed by combining genes from P_{one} and P_{sec}, $N+T-N=T$ and T is the total number of genes within a chromosome. $T-N$ could be less than, equal to, or greater than N.

- Case 1: $T-N>N$
  If the constraints satisfied by P_{one} are less than $\frac{T}{2}$ then obviously the fraction of genes from P_{sec} within child chromosome will be greater than $\frac{T}{2}$.
- Case 2: $T-N<N$
  If the constraints satisfied by P_{one} are greater than $\frac{T}{2}$ then the fraction of genes from P_{sec} within child chromosome will be less than $\frac{T}{2}$.
- Case 3: $T-N=N$
  If the constraints satisfied by P_{one} are equal to $\frac{T}{2}$ then the fraction of genes from P_{sec} within child chromosome will also be equal to $\frac{T}{2}$.

Mutation

The child obtained after crossover is mutated in the next step of the GA. Mutation is the process in which one or few features from genotype/features within chromosome are altered. Mutation is necessary because it takes a step forward or backward to reach a solution. In our work, we mutate only one randomly chosen feature/genotype in a chromosome. The mutation rate M is given by the following equation:

$$M=\frac{1}{TF}*100 \tag{4}$$

Here, TF is the number of total features in a chromosome or the length of a chromosome. As the length of the chromosome in this case is 30, therefore, the mutation rate is set to 3.33%. This means that only one feature from a child chromosome is mutated. Since the chromosome is represented using a binary scheme in the proposed approach, therefore, the mutation operation will convert zeros to ones and vice versa.

5.1.1. Termination condition

The termination condition is met, when the GA outputs a child which satisfies all the constraints. After termination, the GA delivers the target chromosome as an output. We refer to this as a "fit chromosome" in the rest of the paper. Same GA steps are repeated for all features and we get $1*TF$ fittest chromosomes representing each feature and subsuming all features' payloads.

All these fittest chromosomes become the population of the GA and this time the GA returns an output consisting of the "best chromosomes". These best chromosomes subsume the payloads of all the features. The reason for constructing the best and fit chromosomes is to reduce our search space. The best and fit chromosomes are obtained from both vulnerable and non-vulnerable datasets. The difference in feature values of the best chromosomes to vulnerable and non-vulnerable datasets, is used to declare whether a payload from testing data is vulnerable or not. For example

$$\begin{aligned} B_v &= \{f_{1v}, f_{2v}, f_{3v}, \dots, f_{(TF-1)v}, f_{TFv}\} \\ B_n &= \{f_{1n}, f_{2n}, f_{3n}, \dots, f_{(TF-1)n}, f_{TFn}\} \end{aligned} \tag{5}$$

Here, B_v is the best chromosome obtained after processing the vulnerable training data. While, B_n is the best chromosome obtained after applying the GA on non-vulnerable data. These chromosomes (B_v and B_n) which are sequences of Boolean values are represented as vectors. Let MB be the magnitude of difference between B_v and B_n.

Table 4
Difference between vulnerable and non-vulnerable best chromosome.

Bv	11111111111111111111111111111
Bn	11101111111111111111010111111

- If $MB=0$, then it means that the best chromosomes cannot differentiate the testing payloads.
- If $MB>0$, then it means that the best chromosomes have the capability to differentiate the testing payloads.

In the ideal case, MB should be "5.47". The value of MB that we get after training our model is "1.73" because for best chromosomes we found 3 differentiating features that are depicted in Table 4.

To elucidate further, if the feature "fe" in B_v is on but off in B_n then if it is found to be on in the testing payload then we can simply declare it as vulnerable. A Boolean variable B_{Dis} is used for this purpose whose value set to 1 shows that the payload is vulnerable otherwise with its value set to 0 the payload is considered non-vulnerable.

Best chromosomes are the fastest way to declare whether a payload is vulnerable or not but the amount of payloads that it can differentiate depends on the number of differentiating features and the probability of occurrence of that feature in the testing payloads (which is not always the case). Since all the payloads are not able to differentiate through best chromosomes, therefore, the next search space consists of the fit chromosomes. The number of features that are turned on are searched in the next iteration within payloads such that:

$$F=\{f_1, f_2, f_3, \dots, f_n\} \tag{6}$$

Here, the set F represents the number of features that are on in a payload. For all these features f_i, MBf will be calculated. The calculated values of MB are shown in Table 5. A Boolean variable F_{Dis} is also computed which has a value of 1 if the payload is vulnerable otherwise it is 0.

There is a possibility that the target chromosomes of our features could not discriminate the payload. In such a scenario, the binary patterns are consulted for distinction. The Boolean variable Bin is given a value of 1 in the case of a vulnerable payload otherwise it has a value of 0. While, in the case of an overlapping pattern the statistical inference procedure is applied which is elaborated in Section 5.2.

5.2. Statistical inference

In statistical inference, the median of features within each pattern is computed. Here the considered payloads' pattern is changed from binary to numeric. The median chromosome is also calculated in the preprocessing phase. First an overlapping pattern is identified and then payloads where a pattern is repeated are gathered.

$$P_v=\{f_1, f_2, f_3, \dots, f_n\} \tag{7}$$

Here, P_v is the set of vulnerable payloads in which a pattern is repeated, and n is the number of non-zero features in the payload. The numeric value of every feature is arranged in an ascending order within the list L of size SI. The number of lists obtained is equal to n. Then the median of each list is computed as follows:

$$\begin{aligned} M_i &= \frac{SI_i}{2} \\ Med_i &= L\lfloor M_i \rfloor \\ MC_{vul} &= \{Med_1, Med_2, \dots, Med_n\} \end{aligned} \tag{8}$$

The medians of all n features are calculated for a particular pattern. Then, these medians are stored in the form of chromosomes such that at the location of a non-zero feature of a pattern, the respective median value of the feature is placed. The process is repeated for all overlapping patterns. Therefore, for a single pattern, we get two

Table 5
Features and their MB values.

Features	MB	Features	MB
input	1.73	onmouseover	5
Alert	5.09	cookie	4.69
Script	4.89	domain	1.73
Onerror	5.477	onfocus	5
confirm	5	expression	4.35
img	4.89	iframe	4.58
onload	4.69	onclick	5
eval	5.09	singlequotemark	5.09
prompt	5	doublequotemark	5.29
src	5	leftanglebracket	4.58
href	4.35	rightanglebracket	4.35
javascript	3.46	backslant	4.79
window	4.24	coma	4.89
fromcharcode	5.19	plus	4.89
document	5	http	4.89

chromosomes one for the vulnerable pattern and the other for the non-vulnerable pattern. In this phase, the payload is declared as vulnerable if its difference with MC_{vul} is less than MC_{nonvul}. Algorithm 1 lists the steps of the statistical inference procedure used in the proposed approach.

Algorithm 1 Statistical Inferencing

Input: Testing payload of pattern P_t
Output: Flag that declare P_t vulnerable or not
Begin
Extract MC_{vul} and MC_{nonvul} calculated in preprocessing stage.
Calculate Dis_{pn} between P_t and MC_n.
Calculate Dis_{pv} between P_t and MC_v.
if $Dis_{pn} > Dis_{pv}$ **then**
    $Flag \leftarrow 1$
else
    $Flag \leftarrow 0$
end if
End

5.3. Reinforcement learning

We trained our model by training data from online repositories for vulnerable payloads of XSS. These repositories keep on adding new payloads. Once a developer starts defending against a specific vulnerability, an attacker starts attacking in a completely new way. Therefore, the XSS detection model should be flexible and adaptive because the payloads for XSS continuously change forms. Flexibility and adaptability in the model are embedded by RL as shown in Fig. 2, and ensured by implementing Algorithm 2 in the proposed approach.

We tune our proposed model through RL against newly discovered XSS payloads, which are distinguished wrongly by the proposed approach. The addition or modification of patterns in the collection of existing patterns results in a better tuned model. The modified model obtained after this phase, has the potential to distinguish the new payloads which could not be distinguished earlier. The whole process starts when a payload P_t, from the environment, is wrongly distinguished. The value P_d given as an output by our model shows the vulnerability status of P_t. The reward R_t is assigned and the payload is stored in the collection. The process is repeated for the next payload P_{t+1} and R_{t+1} is computed accordingly. The model is modified on the basis of rewards. Fig. 3 shows the updating mechanism of the proposed model.

5.3.1. Preprocessor

The *preprocessing* phase takes the source code, that is to be tested, as an input. The source code is parsed in the next step. This phase

Algorithm 2 Embedding RL in proposed approach

Input: P_t from environment
Output: Tuned model $\mathcal{M}$.
Begin
while P_t.Next != null **do**
    P_s of P_t is calculated from model.
    if P_s of P_t (model) == P_s of P_t (from environment) **then**
        Reward ← positive.
    else
        Reward ← negative.
    end if
    if Reward == Negative **then**
        if P_t != P_o **then**
            P_t is added to model.
            Rerun GA.
        end if
        if P_t == P_o **then**
            $z_j^k = \alpha\ z_j^k + (1\text{-}\alpha)z$
            if z_j^k ¡= β **then**
                f_j.remove(f_j^k)
                Rerun GA.
            end if
        end if
    end if
end while
Return $\mathcal{M}$.
End

is given a separate identity because here we transform our input in a processable format through text mining. The output is a vector containing all the features extracted from the input source code.

5.3.2. Distinguisher

The *distinguisher* distinguishes a payload with the help of the model. Once a preprocessed payload enters the distinguisher, it is matched for similarity with patterns of both vulnerable and non-vulnerable code. The payload is said to be vulnerable if it can harm the system by exploiting it. Insertion of a vulnerable payload allows an attacker to inflict cosmetic to catastrophic damage to a website. For example, this can range from displaying an alert prompt to cookie stealing. On the contrary, a non-vulnerable payload can never result in exploitation of the system.

5.3.3. Reward

The *reward* is that part of the RL algorithm which makes it adaptable to the changing environment. A reward can either be positive or negative. If our results match a standard then a positive reward is assigned. On the contrary, a contradiction with a standard result in the assignment of a negative reward. Repositories that are an origin of our training payload form the environment. By comparing the testing payloads, the model trains itself over new data and in the process a reward is collected for each pattern individually.

5.3.4. Decider

The *decider* uses the reward output. It is given a separate role in the proposed approach because, it has to decide, what sort of refinement is required to be done in the model. Refinement is done by adding, removing, or modifying a pattern after a decision has been made by the decider. A pattern is added when it is not previously a part of our pattern collection. It is modified when an association with a feature is missing. A pattern is deleted when it always wrongly classifies payloads. Additionally, a decider performs refinement in an online manner. In an online mechanism, the model is updated as soon as

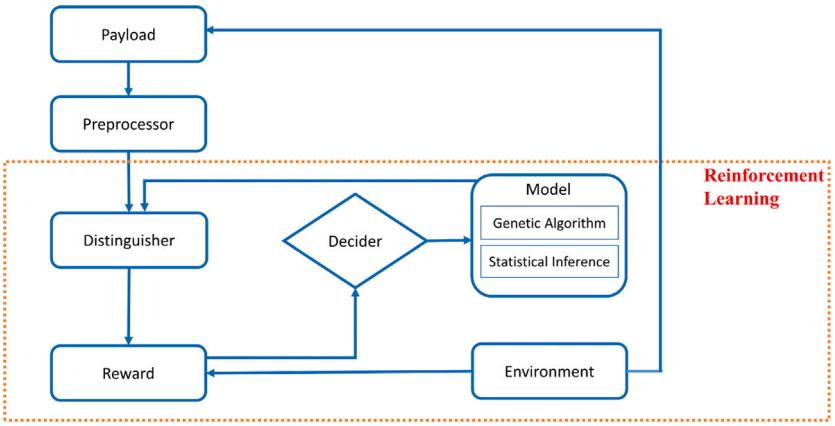

Fig. 2. Embedding RL in proposed approach.

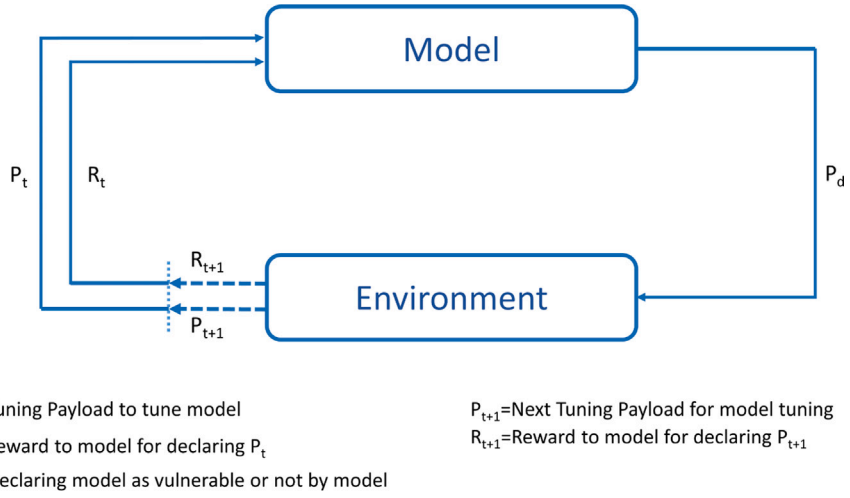

Fig. 3. Learning model of reinforcement learning.

the payload is wrongly declared. The online mechanism for updating patterns avoids wrong discrimination for the same pattern appearing again in some other payload.

5.3.5. Model

The model forms the heart of this work. It employs artificial intelligence for vulnerability analysis. We have used GA, RL, and Statistical inference for vulnerability analysis. These techniques are used to create the best, fit and median chromosomes along with identifying patterns relevant to cross-site scripting vulnerability detection. Whenever a payload is tested it is checked for vulnerability in all of these chromosomes. If the model correctly classifies it, then a positive reward is assigned, otherwise, a negative reward is associated to that pattern. As a consequence of the negative reward, refinement is performed on a pattern residing within the model. In addition to the aforementioned techniques, our approach is augmented by making use of reinforcement learning which has minimized false positives along with providing better accuracy and privacy.

5.3.6. Reward allocation policy

The proposed approach has different features denoted by f_j and each feature has different patterns which we denote by f_j^k. Each pattern has an impact value of z_j^k that gets updated on each reinforcement. Initially z_j^k is given the value of the frequency with which a specific pattern f_j^k appears in a feature set f_j. Additionally, we have the variable α whose values are used to decide when a reinforcement in the model is to be done and is computed as:

$$\alpha = \frac{3 * z_j^k}{4} \tag{9}$$

Similarly, we have a reward value of 1 if the reward is positive and 0 if the reward is negative. When a testing payload P_t from the environment is wrongly classified and it does not overlap with any existing pattern from the trained model, then we add the pattern to its respective f_j. On the other hand, if the P_t overlaps with any of the pattern f_j^k then a reward is assigned to it. This results in the updating of its impact value using the variable β of Eq. (10).

$$\beta = \sigma z_j^k + (1 - \sigma) r \tag{10}$$

where the value of $\beta = 0.8$, means that the previously trained data has a weight of 80% and 20% weight is given to the achieved reward for updating z_j^k. After updating, if the value of $z_j^k \leqslant \beta$, then a pattern is removed. In the case when both an addition and a removal occur the GA is rerun to achieve the modified best and fit chromosomes.

6. Results

We designed the experimental environment in Java 1.8 using the NetBeans IDE 8.0.2. We used 1,35,507 normal records and 50,540 vulnerable payloads collected from GitHub[2] and xssed[3] to train our

[2] https://github.com/duoergun0729/1book/tree/master/data.

Table 6
Comparison of accuracy with previous approaches.

Percentage of malicious records	Proposed approach	Ensemble approach proposed in Zhou and Wang (2019)	SVM	Naive bayes	Decision tree	Random forest	Logistics regression
0% (original)	99.67	96.96	97.76	99.23	97.23	97.76	97.89
5%	99.76	97.59	95.43	96.30	94.24	95.45	95.55
10%	99.75	98.06	93.04	93.37	91.16	93.02	93.13
15%	99.78	97.89	90.64	90.36	88.06	90.58	90.68
20%	99.77	97.64	88.19	87.41	84.90	88.14	88.24
25%	99.85	97.78	85.76	84.44	81.85	85.76	85.82
30%	99.86	97.63	83.39	81.52	78.64	83.30	83.36
35%	99.83	97.88	80.98	78.53	75.54	88.86	80.90
40%	99.87	98.22	78.45	75.54	72.46	78.48	78.54
45%	99.89	98.54	76.16	72.62	69.28	76.02	76.10

model. The testing dataset constituted 3497 vulnerable and 6503 normal records. Testing data was provided by authors of Zhou and Wang (2019). Moreover, the second testing data that was used in Fang et al. (2018) was obtained from the GitHub link.[4] We extracted 30 features from the training dataset. These features were previously proposed by Zhou and Wang (2019). We used JSoup for extracting features from payloads. Features within payloads show association with other features despite variation in payloads observed in output and storage position. The list of extracted features is provided in Table 2. After extracting features, we created separate files representing each feature. For example, the file for the feature "alert" contains all the payloads that have the alert feature switched on (number of alerts > 0). Then a Boolean representation of these payloads was created, and redundancy was removed to improve performance. The GA was applied on these non-redundant binary representations to get the fittest chromosomes of all the respective features. These fittest chromosomes had the characteristic that they subsumed the binary representation of all other chromosomes of the respective feature. These fittest chromosomes were then used as the population of the GA to generate the best chromosomes. The best chromosomes subsumed all representations irrespective of the feature type. In this way the fittest and best chromosomes were generated for both vulnerable and non-vulnerable data. We processed our data to achieve the fittest and the best chromosomes.

The parameters for the GA included, a mutation rate of 3.33%, which meant that only one gene from a chromosome is mutated. The crossover rate as discussed in Section 5.1 was dynamic and non-deterministic in our case. For reinforcement learning, we considered the frequency of a pattern which varied from pattern to pattern. However, an obvious point to be noted is that after reinforcement, the newly updated frequency had 80% of the previous value and 20% of the updated value.

Along with preprocessing, threat intelligence was employed which captured malicious domain names and IP addresses. This enabled to avoid redirections within the payload. The malicious domain was in human readable format and was easily understandable. The IP address was in dotted decimal representation consisting of four parts with each part having digits which can be expressed in 8-bit binary representation. Therefore, an IP address in total took 32-bits in binary notation. Malicious IP addresses and domain names were collected from FireHOL.[5] and PhishTank[6] The vulnerability alarm was triggered whenever threat Intelligence detected a malicious redirection, or our model found a payload to be malicious.

We compare our model and the results obtained from it with the results obtained from well-known classifiers and reported in Zhou and Wang (2019). These classifiers include Naïve Bayes, SVM, Logistic Regression, Random Forest and Decision Tree. We used the Weka tool to test our payloads against the earlier mentioned classifiers. Experiments were performed on the original dataset, selecting and replacing 5%, 10%, 15%, 20%, 25%, 30%, 35%, 40%, and 45% records with vulnerable payloads randomly. Afterwards accuracies obtained from different classifiers were recorded and each experiment was repeated 10 times. Results obtained from the experiments are depicted in Table 6.

Table 7
Precision, Recall, and F1-measure of corresponding percentage of malicious record.

Percentage of malicious record	Precision	Recall	F1-measure
Original	99.50%	99.56%	99.52%
5%	99.60%	99.59%	99.54%
10%	99.62%	99.62%	99.61%
15%	99.71%	99.65%	99.67%
20%	99.65%	99.71%	99.67%
25%	99.79%	99.76%	99.77%
30%	99.76%	99.82%	99.78%
35%	99.79%	99.71%	99.74%
40%	99.82%	99.82%	99.81%
45%	99.88%	99.82%	99.84%

The results show that our proposed approach not only shows better accuracy in its original form but also performs better after random injection of malicious payloads when compared to all comparative classifiers. It is quite noticeable that the accuracy remains pretty consistent and our method has outperformed both in the original setting as well as in its worst injection setting (45%). These results show that with threat intelligence and the proposed genetic model we are able to detect injected malicious records more efficiently. Performance is only affected when incorrectly classifiable payloads replace the correctly identifiable records. However, in our original dataset the percentage of incorrectly identified payloads is 0.2% and so is the case if malicious redirection contaminates our experimental data. Similarly, Table 7, shows the gradual increase in Precision, Recall, and F1-measure values obtained by using our proposed approach when malicious records are added. It can be noted that the values of F1-measure along with precision and recall are also gradually improving with the addition of more malicious payloads. Therefore, we can say that the proposed approach shows better results both in terms of accuracy and F1-measure.

To assess the statistical significance of the results obtained through different methods, we performed detailed statistical significance testing, as suggested in Demšar (2006). First we performed the Shapiro–Wilk test on the accuracies of the algorithms reported in Table 6. The p-values obtained were 0.53, 0.76, 0.88, 0.89, 0.89, 0.90, and 0.88 for the Proposed Method, Ensemble Approach, SVM, Naïve Bayes, Decision Tree, Random Forest, and Logistics Regression, respectively. Hence, showing that the data is not normally distributed for any of the methods, this excluded the possibility of using t-test for statistical significance. Therefore, we used the non-parametric Friedman's test for measuring the statistical significance. It compares the ranks of the methods to measure statistical significance, in the case, when any two methods have the same rank, their average rank is considered. The value of Friedman's test obtained for all the compared methods was

[3] http://xssed.com/.
[4] https://github.com/das-lab/deep-xss.
[5] https://github.com/firehol.
[6] https://www.phishtank.com.

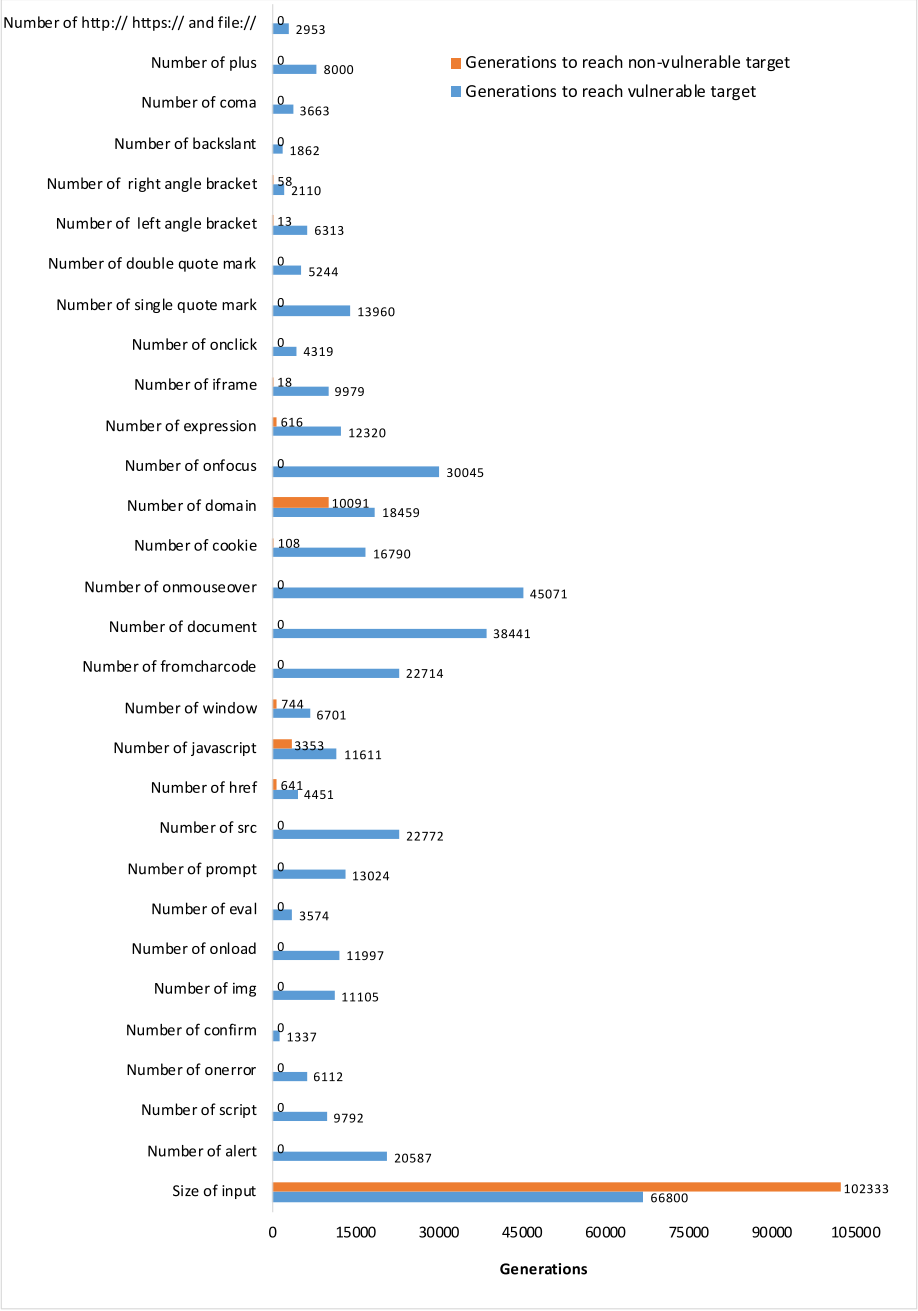

Fig. 4. Comparison of generations to reach target chromosome for vulnerable and non-vulnerable features.

$\chi_F^2 = 45.86$, which is statistically significant. As we intend to compare the proposed method with other methods (see Table 6), we applied Bonferroni correction and compared the proposed method with all the other methods. The values obtained after applying the Bonferroni correction were 5.43, 18.10, 24.84, 48.25, 21.83, and 12.38 for Ensemble Approach, SVM, Naïve Bayes, Decision Tree, Random Forest, and Logistics Regression, respectively. All the values are statistically significant and hence show that the proposed method is statistically significantly better than all the other methods.

Fig. 4 depicts the number of generations taken by the chromosomes to converge to the target chromosomes. The blue line shows the number of generations taken by non-redundant binary representations of different features to reach the vulnerable target chromosomes and the generations employed by each feature payload in achieving the non-vulnerable target chromosomes can be seen on the orange bars in the graph. From this graph, it can be observed that the number of generations produced to reach the vulnerable target chromosomes are more than the number of generations produced to reach the non-vulnerable target chromosomes. The reason is quite understandable if there are variations between payloads, such that with each crossover and mutation the fitness of the child increases, then it takes fewer generations to reach the target chromosome. On the other hand, if the majority of the payloads are similar and only a few are unique then the probability of chromosome selection is greater for similar chromosomes as compared to fewer unique chromosomes. Since, similar ones are selected more often, therefore, it takes more time to converge to the target chromosomes because selection of similar chromosomes will produce similar or slightly different children (due to mutation).

In our second experiment, we have tested our reinforcement learning algorithm. Reinforcement is done by modifying patterns of different features. In our first experiment, we have removed the macro chromosome from each feature and trained our model. The impact on accuracy

Table 8
Impact of reinforcement learning on accuracy.

Features	Accuracy before RL	Accuracy after RL
size of input	85%	98.20%
alert	83%	99.90%
script	69.60%	99.80%
onerror	63.50%	99.00%
confirm	5.78%	99.95%
img	75.40%	100%
onload	56.53%	100.00%
eval	27.30%	99.35%
prompt	78.50%	99.65%
src	75.10%	99.90%
href	62.10%	100.00%
javascript	82.20%	99.90%
window	70.90%	100%
fromcharcode	79.40%	99.90%
document	75.70%	100.00%
onmouseover	76%	100.00%
cookie	73.20%	99.99%
domain	78.90%	99.67%
onfocus	72.80%	100.00%
expression	61.00%	99.70%
iframe	70.50%	99.80%
onclick	23.10%	100.00%
single quote mark	40.94%	100.00%
double quote mark	67.30%	100.00%
left angle bracket	84.00%	99.80%
right-angle bracket	76.90%	99.78%
backslant	44.80%	99.99%
coma	93.96%	100.00%
plus	94.10%	100.00%
http:// https:// and file://	78.50%	99.99%

after reinforcement of model can be seen in Table 8. The ten most impacted features are depicted in Fig. 5.

In our second experiment regarding reinforcement learning, we randomly pick 1, 2, 3, and 4 patterns from each feature which are modified and the model is trained, as discussed in Section 5, such that it can distinguish false positives and false negatives whenever that modified pattern is tested. Our RL updating policy is online. The processing is increased with online updating since our data repeats patterns within it. Therefore, if the pattern that is required to be added in the model, appears more than once then it prevents wrong classification of this pattern in its next appearance. RL adds, deletes or modifies the incorrectly classified chromosome pattern.

It is quite visible from Table 9 that when one of the patterns is altered, slight change in accuracy is observed. It is because the selected pattern (that is randomly chosen) compromises only a few payloads. The accuracy, in this case, does not show a high fluctuation since the number of false positives and negatives are few. Another observation is that by increasing the number of patterns to compromise, a remarkable decrease in accuracy is observed.

It is also observed that when three patterns are compromised then it greatly affects the "Number of Expression" feature and slightly affects the "Number of right-angle brackets" feature. The opposite is true when four patterns are compromised. Therefore, we cannot conclude that the same feature is impacted when the number of randomly chosen patterns increases. In the same way, the effect on the accuracy on the number of features in the graph of three compromised patterns is more than that of the graph of four compromised patterns. The reason is random selection of patterns. With random selection if the selected pattern impacts high number of payloads then fluctuation in the graph will be high. Therefore, percentage of impacted payloads by modifying four chromosomes is lower than those selected by three. This conclusion could be true if we select those patterns that iterate in the maximum number of payloads. In such a case, if four best chromosomes are selected then their impact would be greater than three best selected chromosomes which seems quite understandable.

After observing the impact on accuracy when these patterns are compromised, we denote the original accuracy of a feature as O, the accuracy when one pattern is compromised as P_1, when two patterns are compromised as P_2, when three patterns are compromised as P_3 and when four patterns are compromised as P_4. All these values (P_1, P_2, P_3, and P_4) are subtracted from O to get pattern distance values which are denoted by Pd_1, Pd_2, Pd_3, and Pd_4. The values (Pd_1, Pd_2, Pd_3, and Pd_4) are added and then divided by 4 to get the collective

Table 9
Impact on accuracy when one, two, three, and four patterns are compromised.

Features	Original accuracy	1 Pattern compromised	Two patterns compromised	Three patterns compromised	Four patterns compromised
size of input	98.20%	98.18%	98.56%	98.01%	88.44%
alert	99.90%	99.80%	83.11%	99.87%	99.87%
script	99.80%	99.68%	99.65%	69.30%	69.29%
onerror	99.00%	98.98%	81.96%	98.90%	97.13%
confirm	99.00%	98.99%	98.77%	98.44%	98.10%
img	100.00%	99.90%	99.90%	99.90%	99.90%
onload	100.00%	99.95%	99.74%	99.84%	99.68%
eval	99.30%	99.22%	98.43%	98.87%	98.35%
prompt	99.60%	98.91%	99.45%	91.71%	99.35%
src	99.90%	99.86%	99.75%	99.86%	99.34%
href	100.00%	99.80%	99.91%	99.86%	97.39%
javascript	99.90%	99.87%	99.86%	99.70%	99.83%
window	100.00%	99.96%	99.95%	92.59%	99.90%
fromcharcode	99.90%	99.99%	99.88%	99.35%	98.47%
document	100.00%	99.97%	75.04%	99.93%	99.66%
onmouseover	100.00%	99.99%	99.99%	99.96%	99.65%
cookie	99.90%	99.44%	99.84%	99.15%	99.81%
domain	99.60%	99.59%	99.55%	78.70%	99.35%
onfocus	100.00%	99.90%	99.98%	72.80%	99.94%
expression	99.70%	99.66%	99.60%	60.82%	99.46%
iframe	99.80%	99.76%	99.45%	99.74%	99.72%
onclick	100.00%	99.76%	99.51%	99.27%	95.83%
single quote mark	100.00%	99.70%	99.70%	99.40%	98.95%
double quote mark	100.00%	99.90%	99.82%	99.90%	98.56%
left angle bracket	99.80%	99.14%	99.35%	99.14%	90.21%
right-angle bracket	99.70%	96.85%	91.13%	91.20%	62.60%
backslant	99.90%	98.62%	92.21%	92.21%	94.78%
coma	100.00%	99.82%	99.44%	98.30%	99.53%
plus	100.00%	99.77%	99.30%	99.69%	99.53%
http:// https:// and file://	99.90%	96.33%	74.90%	89.20%	78.50%

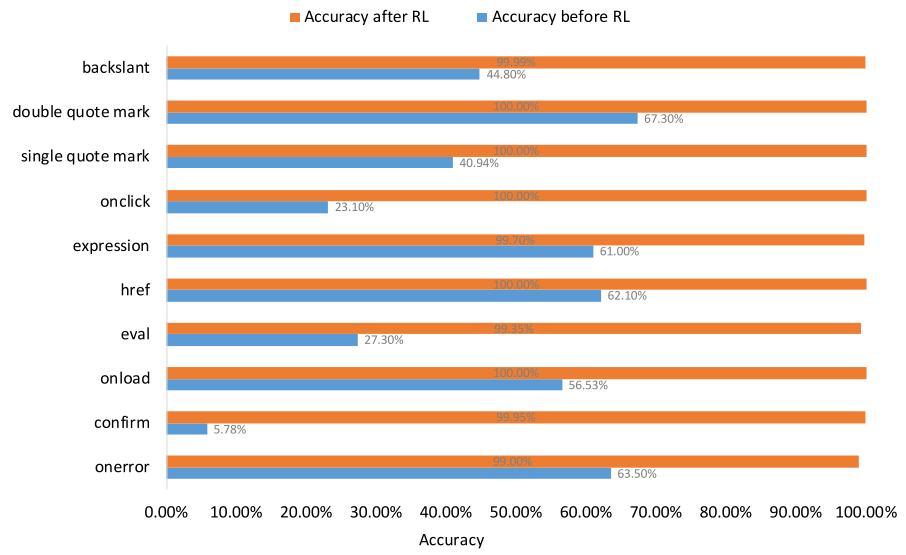

Fig. 5. Impact of reinforcement learning on accuracy.

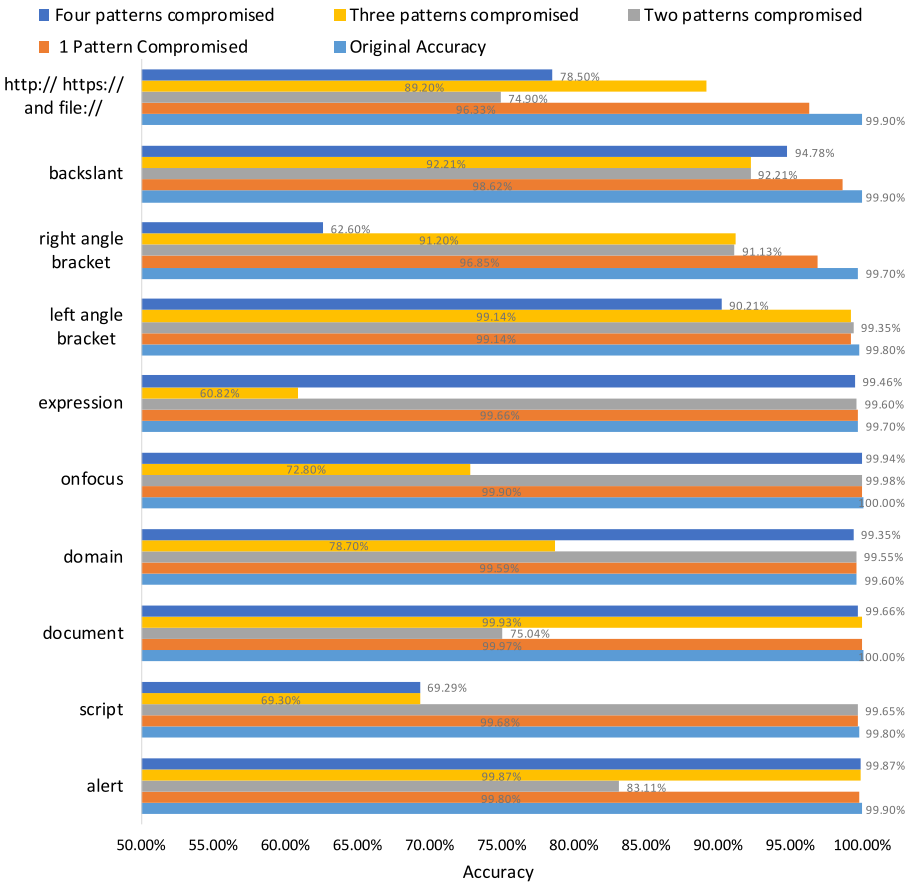

Fig. 6. Impact on accuracy when one, two, three, or four patterns are randomly chosen.

impact value P_c of a feature when P_1, P_2, P_3, and P_4 are compromised. Fig. 6 shows the ten features having the highest P_c values.

We have also compared our approach on data that has been previously used by Fang et al. (2018) and Wang et al. (2014) to test their approaches. Authors in Fang et al. (2018) use deep learning, whereas, Wang et al. (2014) employs ML algorithms which are ADTree and AdaBoost for identification of the XSS vulnerability. The comparison of our techniques with these is shown in Table 10.

It can be seen from Table 10 that our approach has outperformed all previously proposed approaches. Our results not only show better

Table 10
Results comparison with previous techniques.

Classifiers	Precision	Recall	F1 Score
ADTree	0.938	0.936	0.936
AdaBoost	0.941	0.939	0.939
DeepXSS	0.995	0.979	0.987
Our approach	0.998	0.999	0.998

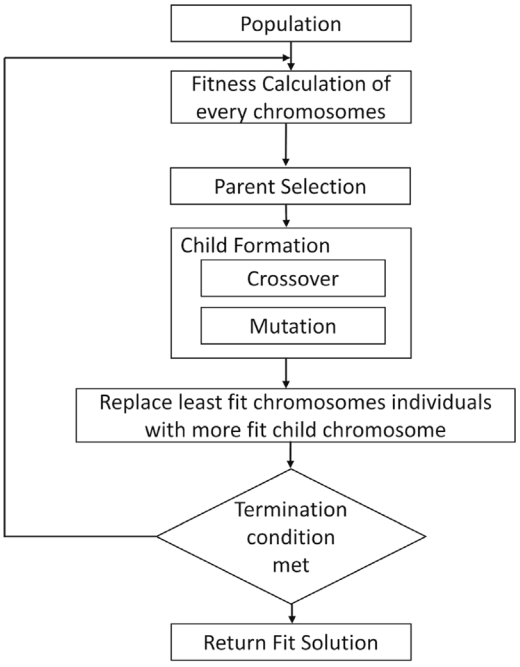

Fig. A.1. Overview of genetic algorithm used in proposed approach.

F1 score but also surpass them in terms of precision and recall values. Therefore, these evaluation results show the effectiveness of our approach in detecting the XSS vulnerability.

7. Complexity analysis of the proposed approach and the compared methods

The training time and run time complexities of the different compared methods along with the proposed method are shown in Table 11.[7] Assuming that n represents the number of training examples, d represents the number of dimensions of the data, dt represents the depth of the tree (where applicable), k represents the number of neighbors (where applicable), g represents the number of generations in the proposed approach and c represents the classes (where applicable).

8. Conclusion

XSS attacks have persistently occupied the topmost position in the list of attacks which show that these types of attacks need more investigation and research as previously proposed approaches (static, dynamic, and hybrid) are unable to detect and prevent them effectively. Recent literature on the mitigation of XSS attacks is based on machine learning techniques which have shown significant promise to address them. However, these techniques are not capable of modifying or adapting themselves for new XSS attacks which are different from those on which the model was trained. To address these issues, our proposed approach uses a GA (as it positively impacts when applied in static analysis), statistical inference (for inspecting numerical payloads) and RL (for adaptability) to effectively detect an XSS attack. Evaluation of the proposed approach on datasets previously used in Fang et al. (2018) and Zhou and Wang (2019) shows that the proposed approach is better than existing approaches in terms of detecting XSS. The proposed approach achieves an accuracy of 99.75% on the Deurgan dataset in the original setting and reaches up to an accuracy of 99.89% in its worst setting (after injecting malicious payloads). The RL module is evaluated by randomly removing patterns from the dataset. These are successfully added and incorporated in the trained model by the RL module of our proposed approach. Another advantage provided by the proposed approach is the white-box nature of the model which results in comprehensible results for the end user.

[7] It is assumed that the preprocessing time is not included in the above complexities.

CRediT authorship contribution statement

Iram Tariq: Conceptualization, Methodology, Investigation, Writing - original draft. **Muddassar Azam Sindhu:** Conceptualization, Supervision, Investigation, Resources, Writing - review & editing, Project administration. **Rabeeh Ayaz Abbasi:** Visualization, Writing - review & editing. **Akmal Saeed Khattak:** Writing - review & editing . **Onaiza Maqbool:** Supervision, Writing - review & editing. **Ghazanfar Farooq Siddiqui:** Writing - review & editing.

Declaration of competing interest

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

Acknowledgments

We are thankful to the reviewers for providing their valuable feedback, which helped us in improving the quality of the manuscript. We are especially thankful to Reviewer 1, for guiding us in properly conducting the statistical significance testing.

Appendix. Genetic algorithm

This appendix gives the pseudocode of the GA used in our work in Algorithms 3 and 4 along with a diagrammatic overview of the genetic algorithm used in the proposed approach in Fig. A.1

Algorithm 3 Proposed Model

Input: Testing payload P_t
Output: P_s of P_t
Begin
Preprocess the payload P_t entered as input.
$T_v \leftarrow$ Threat Intelligence.
if $P_t == P_o$ **then**
    $M_v \leftarrow$ Output of Statistical Inferencing Algorithm.
end if
if P_t != P_o **then**
    $M_v \leftarrow B_{Dis} \vee F_{Dis} \vee Bin$
end if
$P_s \leftarrow T_v \vee M_v$
End

Table 11
Comparison of complexities of proposed approach with other methods.

	SVM	Naive Bayes	Decision tree	Random forest	Logistic regression	Proposed approach
Training complexity	$\mathcal{O}(n^2)$	$\mathcal{O}(nd)$	$\mathcal{O}(nlog(n)d)$	$\mathcal{O}(nlog(n)dk)$	$\mathcal{O}(n)$	$\mathcal{O}(gnd)$
Run time complexity	$\mathcal{O}(kd)$	$\mathcal{O}(cd)$	$\mathcal{O}(max(dt))$	$\mathcal{O}(dt*k)$	$\mathcal{O}(d)$	$\mathcal{O}(n)$

Algorithm 4 Genetic Algorithm

1: Input: Population $\mathcal{P}$, and Constraint set C.
2: Output: Fit Solution S.
3: Begin
4: **while** (! $S \vDash C$) **do**
5: Calculate fitness of each chromosome in population.
6: Select parent P_{one} and P_{sec}.
7: $P_{one} \leftarrow \prod_{max}(Population)$
8: $P_{sec} \leftarrow$ Population.Random() /* Random function always ensures that $P_{sec} \leqslant P_{one}$ */
9: Child.length $\leftarrow P_{one}$.length()
10: **for** `int i=0;i< Child.length;i++` **do**
11: **if** $P_{one}(i) \vDash C$ **then**
12: Child(i) $\leftarrow P_{one}(i)$
13: **else**
14: Child(i) $\leftarrow P_{sec}(i)$
15: **end if**
16: **end for**
17: Mutate(Child)
18: **if** Fitness(Child) > Fitness(P_{sec}) **then**
19: $\mathcal{P}$.remove(P_{one})
20: $\mathcal{P}$.add(Child)
21: **end if**
22: Return S
23: **end while**.
24: End

References

Agosta, G., Barenghi, A., Parata, A., & Pelosi, G. (2012). Automated security analysis of dynamic web applications through symbolic code execution. In *2012 ninth international conference on information technology-new generations* (pp. 189–194). IEEE.

Bau, J., Bursztein, E., Gupta, D., & Mitchell, J. (2010). State of the art: Automated black-box web application vulnerability testing. In *2010 IEEE symposium on security and privacy* (pp. 332–345). IEEE.

Chan, G.-Y., Lee, C.-S., & Heng, S.-H. (2014). Defending against XML-related attacks in e-commerce applications with predictive fuzzy associative rules. *Applied Soft Computing*, *24*, 142–157.

Chen, T., Liu, J., Xiang, Y., Niu, W., Tong, E., & Han, Z. (2019). Adversarial attack and defense in reinforcement learning-from ai security view. *Cybersecurity*, *2*(1), 11.

Dejmal, S., Fern, A., & Nguyen, T. P. (2008). Reinforcement learning for vulnerability assessment in peer-to-peer networks. In *AAAI* (pp. 1655–1662).

Demšar, J. (2006). Statistical comparisons of classifiers over multiple data sets. *Journal of Machine Learning Research*, *7*(1), 1–30, URL: http://jmlr.org/papers/v7/demsar06a.html.

Dowd, M., McDonald, J., & Schuh, J. (2006). *The art of software security assessment: Identifying and preventing software vulnerabilities*. Pearson Education.

Duchene, F., Rawat, S., Richier, J.-L., & Groz, R. (2014). Kameleonfuzz: evolutionary fuzzing for black-box XSS detection. In *Proceedings of the 4th ACM conference on data and application security and privacy* (pp. 37–48). ACM.

Durai, M. K. N., & Priyadharsini, K. (2014). A survey on security properties and web application scanner. *International Journal of Computer Science and Mobile Computing*, *3*(10), 10–4018.

Everett, A. (2006). *Unauthenticated authentication: Null bytes and the affect on web-based applications which use LDAP*. Stillwater: IT Information Security Office, Oklahoma State University.

Fang, Y., Li, Y., Liu, L., & Huang, C. (2018). DeepXSS: Cross site scripting detection based on deep learning. In *Proceedings of the 2018 international conference on computing and artificial intelligence* (pp. 47–51). ACM.

Gupta, B., Agrawal, D. P., & Yamaguchi, S. (2016). *Handbook of research on modern cryptographic solutions for computer and cyber security*. IGI global.

Harman, M., & O'Hearn, P. (2018). From start-ups to scale-ups: Opportunities and open problems for static and dynamic program analysis. In *2018 IEEE 18th international working conference on source code analysis and manipulation (SCAM)* (pp. 1–23). IEEE.

Hydara, I., Sultan, A., Zulzalil, H., & Admodisastro, N. (2015). Cross-site scripting detection based on an enhanced genetic algorithm. *Indian Journal of Science and Technology*, *8*(30), 1–7.

Jovanovic, N., Kruegel, C., & Kirda, E. (2006). Pixy: A static analysis tool for detecting web application vulnerabilities. In *2006 IEEE symposium on security and privacy (S&P'06)* (pp. 6–pp). IEEE.

Jovanovic, N., Kruegel, C., & Kirda, E. (2010). Static analysis for detecting taint-style vulnerabilities in web applications. *Journal of Computer Security*, *18*(5), 861–907.

Krishnaveni, S., & Sathiyakumari, K. (2013). Multiclass classification of XSS web page attack using machine learning techniques. *International Journal of Computer Applications*, *74*(12), 36–40.

Lalia, S., & Sarah, A. (2018). XSS attack detection approach based on scripts features analysis. In *World conference on information systems and technologies* (pp. 197–207). Springer.

Marashdih, A. W., & Zaaba, Z. F. (2016). Cross site scripting: Detection approaches in web application. *IJACSA International Journal of Advanced Computer Science and Applications*, *7*(10).

Mohammadi, M., Chu, B., Lipford, H. R., & Murphy-Hill, E. (2016). Automatic web security unit testing: XSS vulnerability detection. In *2016 IEEE/ACM 11th international workshop in automation of software test (AST)* (pp. 78–84). IEEE.

Ndichu, S., Kim, S., Ozawa, S., Misu, T., & Makishima, K. (2019). A machine learning approach to detection of javascript-based attacks using AST features and paragraph vectors. *Applied Soft Computing*, *84*, Article 105721.

Razzaq, S., Maqbool, F., Khalid, M., Tariq, I., Zahoor, A., & Ilyas, M. (2018). Zombies Arena: fusion of reinforcement learning with augmented reality on NPC. *Cluster Computing*, *21*(1), 655–666.

Sari, A. (2018). Context-aware intelligent systems for fog computing environments for cyber-threat intelligence. In *Fog computing* (pp. 205–225). Springer.

Shahriar, H., & Zulkernine, M. (2008). MUSIC: Mutation-based SQL injection vulnerability checking. In *2008 the eighth international conference on quality software* (pp. 77–86). IEEE.

Shar, L. K., & Tan, H. B. K. (2012). Automated removal of cross site scripting vulnerabilities in web applications. *Information and Software Technology*, *54*(5), 467–478.

Stasinopoulos, A., Ntantogian, C., & Xenakis, C. (2015). *Commix: Detecting and exploiting command injection flaws*. Piraeus, Greece: Dept. Digit. Syst., Univ. Piraeus, White Paper.

Su, Z., & Wassermann, G. (2006). The essence of command injection attacks in web applications. In *Acm sigplan notices, Vol. 41* (pp. 372–382). ACM.

Takamatsu, Y., Kosuga, Y., & Kono, K. (2012). Automated detection of session management vulnerabilities in web applications. In *2012 tenth annual international conference on privacy, security and trust* (pp. 112–119). IEEE.

Vogt, P., Nentwich, F., Jovanovic, N., Kirda, E., Kruegel, C., & Vigna, G. (2007). Cross site scripting prevention with dynamic data tainting and static analysis.. *2007*, In *NDSS* (p. 12).

Vrbančič, G., Fister Jr, I., & Podgorelec, V. (2018). Swarm intelligence approaches for parameter setting of deep learning neural network: Case study on phishing websites classification. In *Proceedings of the 8th international conference on web intelligence, mining and semantics* (pp. 1–8).

Wagner, S., Jürjens, J., Koller, C., & Trischberger, P. (2005). Comparing bug finding tools with reviews and tests. In *IFIP international conference on testing of communicating systems* (pp. 40–55). Springer.

Wang, D., Gu, M., & Zhao, W. (2017). Cross-site script vulnerability penetration testing technology. *Journal of Harbin Engineering University*, *38*(11), 1769–1774.

Wang, R., Jia, X., Li, Q., & Zhang, S. (2014). Machine learning based cross-site scripting detection in online social network. In *2014 IEEE intl conf on high performance computing and communications, 2014 IEEE 6th intl symp on cyberspace safety and security, 2014 IEEE 11th intl conf on embedded software and syst (HPCC, CSS, ICESS)* (pp. 823–826). IEEE.

zhong Zhang, J., & Chai, A. (2016). Mining cross site scripting vulnerabilities based on HTML5 in email systems. In *International conference on computer networks and communication technology (CNCT 2016)* (pp. 765–773). Atlantis Press, http://dx.doi.org/10.2991/cnct-16.2017.106.

Zhou, Y., & Wang, P. (2019). An ensemble learning approach for XSS attack detection with domain knowledge and threat intelligence. *Computers & Security*, *82*, 261–269.