A Signature-Based Intrusion Detection System for Web Applications based on Genetic Algorithm

Robert Bronte
Kennesaw State University
rbronte@kennesaw.edu

Hossain Shahriar
Kennesaw State University
hshahria@kennesaw.edu

Hisham M. Haddad
Kennesaw State University
hhaddad@kennesaw.edu

ABSTRACT

Web application attacks are an extreme threat to the world's information technology infrastructure. A web application is generally defined as a client-server software application where the client uses a user interface within a web browser. Most users are familiar with web application attacks. For instance, a user may have received a link in an email that led the user to a malicious website. The most widely accepted solution to this threat is to deploy an Intrusion Detection System (IDS). Such a system currently relies on signatures of the *predefined* set of events matching with attacks. Issues still arise as all possible attack signatures may not be defined before deploying an IDS. Attack events may not fit with the pre-defined signatures. Thus, there is a need to detect new types of attacks with a mutated signature based detection approach. Most traditional literature works describe signature based IDSs for application layer attacks, but several works mention that not all attacks can be detected. It is well known that many security threats can be related to software or application development and design or implementation flaws. Given that fact, this work expands a new method for signature based web application layer attack detection. We apply a genetic algorithm to analyze web server and database logs and the log entries. The work contributes to the development of a mutated signature detection framework. The initial results show that the suggested approach can detect specific application layer attacks such as Cross-Site Scripting, SQL Injection and Remote File Inclusion attacks.

CCS Concepts

• CCS → Security and privacy → Software and application security → Web application security

Keywords

Intrusion detection system; application layer attacks signatures; genetic algorithm; mutation; log analysis; cross over; selection.

1. INTRODUCTION

An Intrusion Detection System (IDS) is a system that protects computer networks from attacks. These systems work with the firewalls and anti-virus systems to safeguard the system from those with malicious intent [1]. Traditional IDSs use signatures where attacks are defined as a sequence of events to match with network traffic [2]. This approach is accurate as long as the list of attacks is known in advance and signatures are defined before deploying an IDS such as Snort [3] and Bro [4]. There has been little effort to develop signature-based IDS for web applications. Moreover, they rely on regular expressions to detect attacks. For example, a script created to use a PHPIDS [35] allows attack signatures to be expressed using a set of regular expressions. The burden is on the user to keep up with new expressions. To address this limitation of a signature based IDS, in this paper, we propose to develop a Genetic Algorithm (GA) based IDS. GA-based approaches have gained the attention of the research community in recent years. In a signature-based attack detection approach, the network traffic is monitored and the IDS searches for malicious behaviors that match the known signatures [1]. Any signatures with even minor deviations from the attack descriptions would not set off any security alarms, which may leave a system vulnerable [5]. However, a GA-based approach can address this limitation by generating new signatures from existing signatures.

A representative sample of literature works have developed a signature based IDSs [5-14]. The vast amount of research on this topic has focused on the network layer and multiple Denial of Service attacks. Each of the perviously mentioned works has its own strengths, but all of the approaches have universal underlying limitations. First, in the literature, there is little wide-spread coverage of the known web application layer attacks (see Table 1 for the levels we follow in this work). Each study identified at least one of these types of attacks. Second, each of the related works have yet to attain zero false positive and false negative rates with only one exception. Finally, regardless of the type of attack the authors were searching for, each study only considered one type of log data for analysis. This paper addresses these limitations by discussing a signature based IDS framework to detect certain application layer attacks by analyzing data from multiple logs generated by web applications. The included signature based IDS is meant to protect web applications.

SIN'16, July 20–22, 2016, Newark, NJ, USA.
© 2016 ACM 978-1-4503-4764-8/16/07…$15.00.
DOI: http://dx.doi.org/10.1145/2947626.2951964

Table 1: Signature-based attack detection [15]

App Layer	Attack Type	Work(s)
Application	XSS	[9, 14, 16], this work
	RFI	[19], this work
	SQLI	[13, 18], this work
	Brute-Force	[7, 11, 14]
	Buffer Overflow	[12]
	CSRF	[20]
	Zero-days	[5]
Transport	SYN/ACK	[7]
	XMAS Scans	[8]
	DoS	[6, 12]
	Apache2	[10]
Network	Smurfs	[8]
	Ping-of-Death	[8]
	SYN flooding	[7]

Table 1 shows various attack types at different levels and related work that proposed IDS. This table was created based on industry-level data [15]. Our approach analyzes web server log data, trains an IDS using a GA, and detects three common web application attacks (Cross-Site Scripting, SQL Injection and Remote File Inclusion attacks). In addition, the evaluation of previously collected normal data and newly created malicious data via this approach would provide a detailed view of the results.

The remainder of this paper is organized as follows: Section 2 provides a general overview of common web application layer attacks and details related works on signature based IDS and genetic algorithms. Section 3 explains the applicable approach and how we implemented various steps. Section 4 presents the evaluation results. Finally, Section 5 concludes the paper and discusses future work.

2. OVERVIEW AND RELATED WORK

2.1 Overview of common Web Attacks

This paper focuses the detection of three application layer attacks: Cross-Site Scripting (XSS), SQL Injection (SQLI) and Remote File Inclusion (RFI). For completeness, we briefly discuss all six types of attacks listed in Table 1. In Table 1, it also shows example attacks from layers (Transport and Network) that are not of interest to this study. XSS occurs when unsanitized use of data is interpreted or becomes part of generated response pages. XSS is data entering the web application from an untrusted or undetected source. Such attacks as XSS can compromise web server security before these logs are generated, allowing the attackers to sneak through the cracks completely undetected [16]. Chou [13] explains SQL injection as the exploitation of web applications in order to inject some type of code to gain unauthorized access to the backend of databases. In other words, an SQL injection is an attack that reveals vulnerabilities of a database that uses SQL queries to access the database. The user input field is taken advantage of within a web application to craft the SQL statement. The statement allows an attacker to modify or delete anything within the database [18]. A remote file inclusion (RFI) attack allows the attacker to call pieces of code located in another place, later compromising the host [19]. The attacks that occur on the application layer, but are not addressed directly in this paper include brute-force attacks, buffer overflows and Cross-Site Request Forgery attacks (CSRF). A CSRF attack, according to [20], is an attack that allows attackers to carry out unwanted actions on the end user's application after the user has been authenticated on the web application. Backdoor attacks allow remote access to web applications in order to attack a system, resulting in instantaneous abnormal communication between two systems [21]. According to [22], a brute-force attack happens when there is a large number of attempts to log into a user's account. During a buffer overflow attack, malicious code is injected into the web application and control of the application is lost [23]. The next section discusses relevant literature work closely related to our work.

2.2 Related work on Signature IDS

Table 2 shows a summary of related work and a comparison to this work. We classify the works in multiple ways. We look at benchmark data sources, the attributes of examined data, metrics employed by the IDS, the environment where implementation and evaluation was conducted, the types of attacks being covered, and reported performance rates (false positive and false negative rates). Among all the works, one study achieved detection of Zero-day attacks by analyzing web and database server logs, and examining attack code, command payload and traffic generated by the payload [5]. Neelakantan and Rao [6] used protocol information, headers, and packet payloads of packet captures to reduce the total number of false alarms from DDoS attacks. The rules defining the source address, destination address and destination port were used to increase the speed of signature detection in [7]. In [8], the signature of a priori algorithm from the MySQL database logs was proposed to detect known network level attacks. Based on PHP source code, Gupta et al. [9] used a controlled VM to detect XSS attacks achieving 0% FP and 0% FN rate. The authors of [10] used known SNORT and ClamAV signatures to detect signature based attacks.

Additionally, the researchers utilized a honeypot to collect data for a character frequency exclusive signature matching scheme and the Boyer-Moore algorithm was applied to the data set. Through the use of Mail Exchange (MX) records on Windows servers, the authors of were able to brute-force into numerous Hotmail addresses [11]. In [12], Vigna, Robertson and Balzarotti examined Apache logs to collect data on string length and sequence and exploited mutations to detect buffer overflows, directory traversals and other attacks. Chou [13] used web servers that were hosted in a Cloud environment to detect SQL injections, XSS, and brute-force attacks. Finally, in [14] the researchers looked at innerHTML properties such as GET, HTTP header and cookies to determine the presence of mutation-based XSS attacks, denoted as mXSS. This work, in contrast, uses logs collected from both the web server and the MySQL database for analysis. The environment is configured with a single host running multiple virtual machines (VMs) within a virtual cluster. Some of the VMs are running Windows while others are running Linux. We decided to use a signature matching scheme and added genetic operators to introduce changes into the signatures. Such mutation allowed for the GA to detect all of the known attack input. This includes XSS, SQLI and RFI attacks and results in a zero percent false positive rate and a zero percent false negative rate.

2.3 Related work on GA

Table 3 shows some literature works that apply a GA to develop an IDS. Data sources and types of detected attacks vary greatly across the body of signature based IDS literature. For example, in [23], Avancini and Ceccato examined parameters and values of PHP code to find XSS attack vectors. These authors use static analysis of the GA to automate the log analysis procedure. They also minimized the false positive and false negative alarms through path sensitization.

Authors of [24] created their own set of suspicious data and used the DARPA dataset as normal data. When comparing this GA-based IDS to a known signature database called Snort, the authors found that the GA-based IDS outperformed Snort by detecting a higher number of attacks and having a lower false alarm rate than Snort. Danana and Parvat obtained all of their data from the KDD99 dataset [25], and were able to find attacks including Denial of Service, Probing, User-to-root and Remote-to-local attacks. A fuzzy genetic algorithm utilized in [26] pulled data from a six-by-six matrix of response-resource entries to measure the parameters of the fitness function. A multivariate statistical clustering algorithm was suggested to detect web application attacks in [27].

Table 2: Related Work on Signature Based Attack Detection Approaches

Author(s)	Source of data/ benchmark	Attributes of data examined	Metrics used in their model	Environment, configuration, virtual machines	Attacks detected	Performance (FP and FN)
Holm [5]	Web servers and database servers	Attack code, command payload, traffic generated by the payload	Percentage of known attacks detected	Windows 2000, XP, 2003, Vista and Ubuntu 10.04	Zero-day Attacks	False positives are mentioned, but no percentages are provided due to the variety of employed OS
Neelakantan and Rao [6]	Apache, OpenSSH, SMTP	Protocol information, headers, and packet payloads of packet captures	Percent of critical alarms generated	Linux (Red-Hat), Windows 2003 and Windows 2000	DDoS	The article mentions that the total number of false alarms were reduced
Krugel and Toth [7]	Web servers	Rules defining the source address, destination address, and destination port	Average increase in speed of signature-detection	Red-Hat Linux	Brute-Force, Synscan, Portscan	Not reported
Modi et al. [8]	MySQL database log	Signature A priori Algorithm	Proposed solution, not implemented thus far	Eucalyptus on Ubuntu	Known network attacks, DoS derivatives	Propose a low FP rate
Gupta et al. [9]	PHP Source Code	HTML Context including styles, body tag names, etc.	Number of safe vs. unsafe files	Controlled VM	XSS attacks	0% FP and 0% FN rate
Meng et al. [10]	Honeypot	Character-frequency exclusive signature matching scheme and the Boyer-Moore algorithm	Maximum execution time variance using incoming payload or signature set partitions	VMs in Cloud Environment	Known SNORT and ClamAV signature-based attacks	Not mentioned; focused on reducing TIME to process signatures, not FP/FN rates
Parwani et al. [11]	MX records on Windows servers	Expired Hotmail addresses	Number of accounts that were hacked	Windows	Brute Force Attacks	Not discussed
Vigna, Robertson and Balzarotti [12]	Web servers, such as Apache	String length and sequence, exploited mutations	Number of signature-based attacks detected by either SNORT or IIS RealSecure before and after mutating data	Linux, Windows, OpenBSD	Buffer Overflows, DoS, Stack Overflow, Directory Traversal, Double decoding (code execution), Non-exhaustive Signatures	Did not consider FP/FN results because the goal was to provide useful indication about the average quality of signatures under testing
Chou [13]	Web servers in the Cloud	Data models categorized as SaaS, PaaS or IaaS	Increased frequency of identified SQL injection attacks in SaaS, PaaS, and IaaS Cloud settings measured in percent	Linux, Solaris, Windows VMs in Cloud environment	Malware injection, specifically SQL injection; other attacks can be detected with Cloud systems: DDoS, brute-force, session hijacking, XSS, etc.	Not discussed; focus was more on cloud security
Heiderich et al. [14]	Web servers	innerHTML properties: GET, HTTP header, cookies, etc.; mXSS vectors	Page load time in milliseconds versus the page size with and without the performance penalty introduced to users by TrueHTML	VMs running Ubuntu	Mutation-based XSS attacks (mXSS)	Briefly suggested as potential future research, but not directly addressed
This work	Web servers MySQL database	Signature matching scheme with added genetic mutations	Mutation, selection, chromosome cross-overs	VMs running Linux or Windows	XSS, SQLI, RFI	0% FP and 0% FN rate

The discrete variables in the study were measured by frequency and the number of similar characters between two separate activities (attacks) was suggested as a way to lower the number of false alarms. Liu and Fang genetically modified two sets of real numbers to shorten the lengths of the chromosomes to optimize the GA [28]. In another study, network attacks like smurf, teardrop, neptune, portsweep and others were identified in offline, normal audit data as well as in real time, processed data [29]. Normal data and attack data were compared by the authors looking for Denial of Service, Probes, User-to-root and Remote-to-local network-level attacks [30]. Authors of [30] lowered the false alarm rates by implementing an optimal genetic feature selection process and a support vector machine. Despite the potential of the existing signatures in the literature to detect patterns across any operating system, this work will use genetic operators to mutate such patterns for detection.

Table 3: Comparison of Related Work on Genetic Algorithms

Author(s)	Source of data	Metrics	Environment Configuration	Types of attacks	Performance (FP and FN)
Avancini and Ceccato [24]	Parameters and values of PHP code	Integrating a static analysis of the genetic algorithm and taint analysis	Signatures are independent of operating system	Cross-Site Scripting attack vectors	FP and FN were minimized through path sensitization
Barati, Faez and Hakimi [25]	Normal dataset (DARPA) and suspicious dataset	Number of scan attacks detected/missed by this GA-based IDS versus Snort	Operating system is irrelevant	Horizontal and vertical scan attacks	Overall false alarm rate was 10 percent
Danane and Parvat [26]	KDD99 dataset	Accuracy, execution time, memory allocation	Web attacks can occur on any operating system	DoS, Probe, U2R, R2L	Rules added in the testing phase to reduce FP and FN
Fessi, Abdallah, Hamdi and Boudriga [27]	6 by 6 binary matrix of response-resource entries	Rules that only match anomalous connections to show signatures, parameters of fitness function	Operating system is irrelevant; Fuzzy Genetic Algorithm (GA)	Not specified; focused on the fitness value: attack impact ratio	FP rate was "low" but not specified numerically
Zhou and Lui [28]	Multivariate statistical clustering algorithm is suggested to detect attacks; No real data is used in the study	Frequency for discrete variables; the number of similar characters between 2 activities	Patterns are independent of operating system; trans-platform algorithm for pattern recognition	Web-layer attacks under study, but none are specified	Not disclosed but states the goal is "a very low rate"
Liu and Fang [29]	Two sets of real numbers are genetically modified to shorten chromosome length	Delphi method to determine figure 1 in this article; fitness value of each chromosome, total fitness values, selection probability	Operating system is irrelevant and may change on a given host based on software security	No attacks; discusses modified genetic algorithm to optimize	Not FP or FN; rather studied detection reliability (R), time of detection (T), and threshold time (S)
Narsingyani and Kale [30]	Offline data for normal traffic dataset [audit data]; real time data for attack detection [Processed data]	Src_bytes, land, wrong_fragment, [all numerical] and service [nominal]	Environment is not included, but OS does not matter; Compared their work to KDD 99 data set	DoS attacks: smurf, pod, teardrop, Neptune, back, portsweep	Specifically focused on FP rate to improve performance by increasing the number of rules
Senthilnayaki, Verkatalakshmi and Kannan [31]	Attack data and normal data	Protocol type, service, src_bytes, flag, num_failed_logins, Logged_in, srv_diff_host_rate, dst_host_srv_count, is_guest_login, and num_shells	Network-based IDS; operating system unknown/not disclosed	DoS, Probe, U2R, R2L	Stated to be reduced by using optimal genetic feature selection and a support vector machine

3. GENETIC ALGORITHMS

3.1 Steps for making a GA

Generally speaking, a genetic algorithm advances a set of solutions by combining good solutions to craft new ones until the best solution is found. This process is composed of multiple steps [32-35]. In order to generate a genetic algorithm, a general process and set of steps can be followed. The first step is to create an initial population. This population is typically generated in a random manner and may include as many individuals as preferred, from a few to several thousand. An individual is also called a chromosome in the population. Each chromosome in the initial population is then evaluated for fitness. Next, a new population has to be created. This process consists of repeating the steps that use genetic operators, including selection, crossover and mutation, until the new population is established. During the selection phase, the main goal is to keep the best individuals in the population and improve the overall population fitness. Two parent chromosomes are selected for from the population based solely on their fitness. The better the fitness score is, the more likely that the chromosome will be selected for the population. Crossing over, or the sharing of information, takes place between two parents to create new offspring or children. This occurs in hopes of crossing two chromosomes with a high fitness value that will then create an offspring that has the best traits from each parent chromosome. When mutations occur, there are random changes that happen in individual genes. This increases the diversity among the initial population over multiple generations. All of the new offspring are placed into a new population and serve as the base population for the next iteration of the genetic algorithm. Genetic algorithms are used to create repetitive populations until the optimum solution for the population is found or the population's end condition is reached [33-35]. The genetic algorithm generation procedure is outlined in Figure 1.

1. Begin with a random set of solutions (represented by chromosomes) to form population.
2. Evaluate the fitness of each chromosome in the population
3. Create new solution by using genetic operators (selection, cross over) by selecting chromosomes having higher fitness level.
4. Apply mutations randomly on newly generated chromosomes. There is potential for new offspring to inherit traits from any locus on the chromosome.
5. Repeat steps 2-4 until we reach maximum number of iterations, or exceed population size.

Figure 1. Steps of Genetic Algorithm

3.2 Dataset generation for GA-based IDS development

As our goal is to apply GA to improve a signature-based IDS, we start with an attack dataset that we generated by deploying a large scale PHP web applications named Joomla [38]. The applications were interacted automatically using scripts, and provided with malicious inputs. We collected inputs from the sources such as OWASP ([37, 39]). These attack inputs are applied randomly within web requests from a browser. Table 4 shows number of samples of web request having attack inputs.

Table 4: Distribution of attack inputs

Attack type	# of samples
SQLI	1000
RFI	5
XSS	60
Total	1073

Figure 2 shows an example of log data for SQL injection attack where an input field (id) is having a tautology attack (encoded in hexa-decimal format).

```
"GET
/sqlinj/?id=1%27+or+%271%27+%3D+%271%27%29%29
%2F*&Submit=Submit&user_token=c14e5f424d9f279c19ba
507492745d50
```

Figure 2. Example log data for SQL Injection attack

Similarly, Figure 3 shows an example of log data for XSS attack where an image source is supplied with script code.

```
"GET
/xss_r/?name=%3CIMG+SRC%3DJaVaScRiPt%3Aalert%28%26quot%
3BXSS%26quot%3B%29%3E&user_token=f37e5a82a994725092fd315
5bb8cffba
```

Figure 3. Example log data for XSS attack

Figure 4 shows an example of log data for RFI attack, where *FORMAT* field is included with an include statement pointing a file source from attacker controlled website, followed by *exit()* command.

```
"GET
/?FORMAT={${include("http://www.verybadwebsite.com/hack
er.txt")}}{${exit()}}
```

Figure 4. Example log data for RFI attack

3.3 Adopting GA for Signature-based IDS

The GA accepts a set of chromosomes as input, and provides another set of chromosomes as outputs after certain number of iterations while following fitness evaluation, cross over and mutations. For our contribution, we first convert each of the GET request to a chromosome, which is a bit string representation. Figure 5 shows an example representation of chromosome for SQL injection attack (based on Figure 2). Here, we have three blocks of information that include total number of SQL keywords (two of them include OR, =), presence of encoded character (1=yes, 0=no), number of input fields having SQL keyword (one field having SQL keyword). The last block is the decision block, which represents attack type (expressed in four bits). In the literature, there are six common types of SQL injection attacks. Hence, we reserve three bits to express various types of attacks. Here, 0001 means tautology attack. Other attack types include, piggy backing (0010), union (0011), stored procedure (0100), blind SQL injection (0101), and timing attack (0110).

# of SQL keywords	Presence of encoded character	# of fields with SQL keyword	Attack type
010	1	001	0001

Figure 5. Example of a chromosome (C1) for SQL Injection

Since each chromosome length should be same across different types of attacks, we define chromosome for XSS and RFI using three blocks of bit representation, followed by attack type information. Figure 6 shows an example of chromosome for XSS based on Figure 3. Here, three script/html words are present (<script>, <img>, </script>), the input is encoded, and one field has XSS keywords. The attack type can be three: reflected (0111), reflected (1000), and DOM-based (1001).

# of script/html keywords	Presence of encoded character	# of fields with XSS keyword	Attack type
011	1	001	0111

Figure 6. Example of a chromosome (C2) for XSS attack

# of URL	Encoded	# of command	Attack type
001	0	001	1100

Figure 7. Example of a chromosome (C3) for RFI attack

Figure 7 shows an example of RFI chromosome based on Figure 4, where the attack payload includes one URL, and it was not encoded. There is one command included. We use three attack types: URL only (1010), command only keyword (1011), and URL and command (1100).

We define two fitness functions (FF2, FF3) to evaluate chromosome x as follows (Equation (i) to (iii)).

FF1(x): # of attacks detected by x in training dataset/total # of attacks in training data … … (i)

FF2(x): # of attacks detected by x in testing dataset/total # of attacks in testing data … … (ii)

FF3(x): FF1(x) + FF2(x) … … … (iii)

As for example, we can apply FF3 to evaluate the fitness value of SQLI chromosome (C1). If we assume that C1 matches with 1 attack input out of 100 samples, and results in no false positive warning, then its fitness value is 0.01. When we are evaluating fitness function for a chromosome, we are considering the entire dataset including training and testing. We compare bit level representation of chromosomes from training or testing data to detect how many attacks are detected.

The chromosomes are crossed over based on fitness level. We apply one point cross over. For example, if we decide to cross over between C1 and C2, the before and after results would mimic those added below. If we assume in C1 (after cross over), the fourth bit gets mutated from 1 to 0, then we have a new signature (01000010111) for XSS, where attack payload is not encoded.

Before cross over (C1, C2):

010	1	00 1	0001
011	1	00 1	0111

After cross over (C1, C2):

010	1	00 1	0111
011	1	00 1	0001

Figure 8 shows the framework of GA-based IDS. Here, the weblog data is converted to chromosomes and then the GA is applied to generate more chromosomes which act as attack signatures.

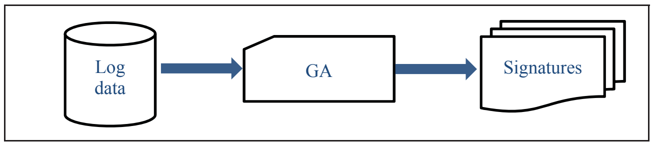
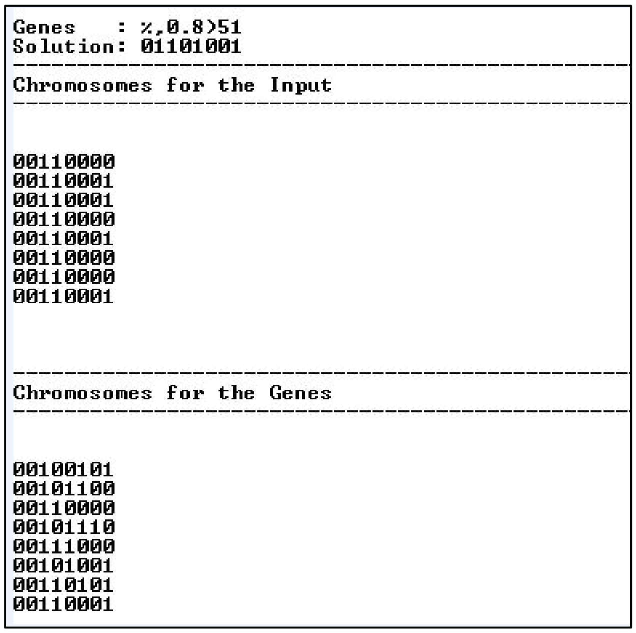


Figure 8: GA-based IDS framework

4. EVALUATION

4.1 Evaluation for GA parameters

We divide our attack dataset (web log files) into two parts: training dataset (30%) and testing dataset (70%). This division is based on some earlier literature work that also developed GA-based classifier (see Table 3). For each of the training dataset, we convert GET or POST requests into chromosome representations by editing and implementing a number of open source PHP class files [40]. Figure 9 shows a screenshot of the application output used while evaluating this approach on a Windows Computer.



```
Genes    : %,0.8>51
Solution: 01101001
------------------------------------------------------------
Chromosomes for the Input
------------------------------------------------------------


00110000
00110001
00110001
00110000
00110001
00110000
00110000
00110001


------------------------------------------------------------
Chromosomes for the Genes
------------------------------------------------------------


00100101
00101100
00110000
00101110
00111000
00101001
00110101
00110001
```

Figure 9. Screenshot of results from GA-based IDS

Figure 10 shows attack detection accuracy for various population sizes while using FF2 as the fitness function and keeping the mutation rate at 0.5. We can observe that the higher the selection rate for a chromosome to cross over, the better accuracy for attack detection capability we achieve.

Figure 11 shows the attack detection accuracy for various population sizes while using FF3 as the fitness function and keeping the mutation rate at 0.7. Each chromosome had varying selection rates, which makes it is easy to see that attacks are detected with more accuracy as the population size increases and the selection rate increases.

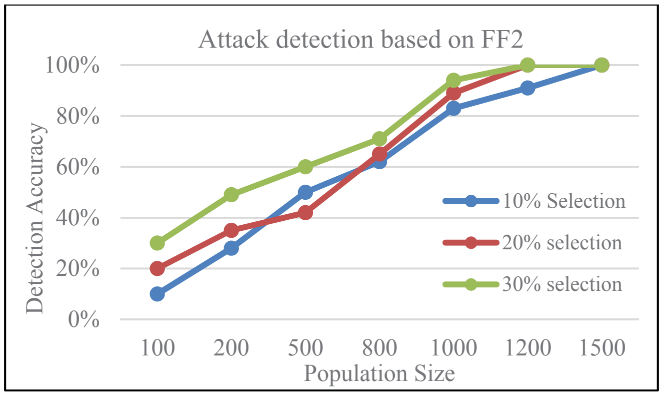


Figure 10. Attack detection accuracy vs. population size for different selection rates (FF2, mutation rate=0.5)

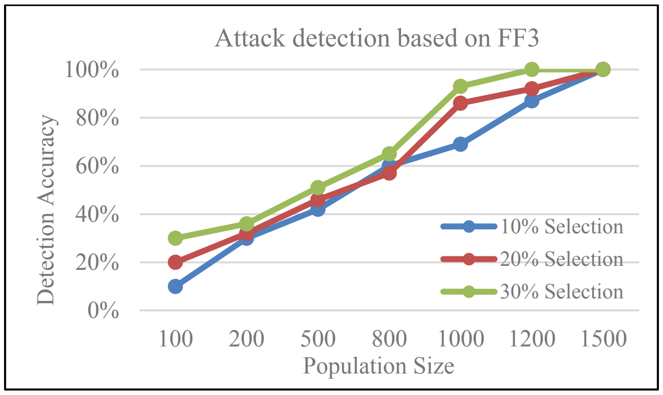


Figure 11. Attack detection accuracy vs. population size for different selection rates (FF3, mutation rate=0.7)

Figure 12 demonstrates that as mutation rate changes, so does the accuracy of attack detection. Here, the selection rate was set at 10 percent and FF2 was used as the fitness function. A higher mutation rate implies that the attacks can be detected with more accuracy and within a smaller population.

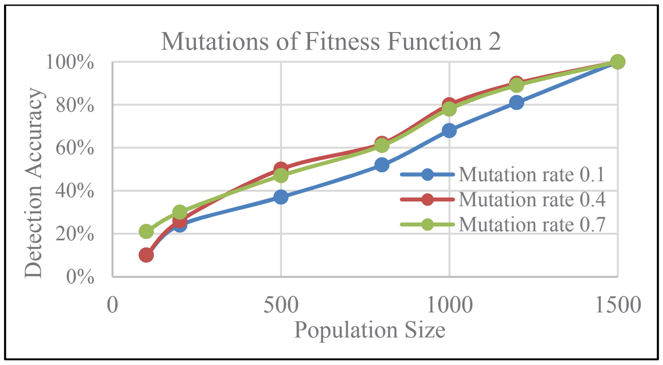


Figure 12. Attack detection accuracy vs. mutation rate (FF2, selection rate=10%)

Figure 13 illustrates that as mutation rate increases, so does the attack detection accuracy. For this situation, FF3 was used as the fitness function and the selection rate was set at 20 percent. A higher mutation rate shows that the attacks can be detected with more accuracy and in a smaller population.

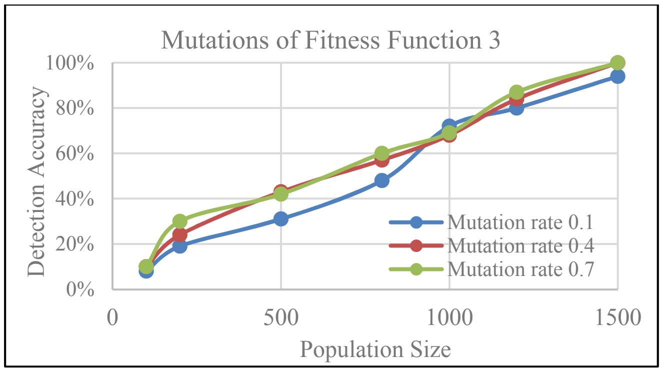


Figure 13. Attack detection accuracy vs. mutation rate (FF3, selection rate=20%)

4.2 Comparing GA with PHP IDS

We compare the GA-based IDS with PHPIDS [36], which is a popular open source web application level attack detector. PHPIDS relies on a set of regular expressions in a configuration file to detect known signatures. Such regular expressions are the signatures of each attack under study. Therefore, by testing a known attack dataset, we compare GA with PHPIDS. Figures 14-16 below illustrate samples of the regular expressions provided in PHPIDS for XSS, SQLI, and RFI. In Figure 14, any script code can be detected that is pre or post pended with an arbitrary string ("(?:\<scri)|(<\w+:\w+)]"). It can also detect data having possible scripts (other than <strong> tag). Similarly, Figure 15 shows an example regular expression that is supposed to detect SQL injection attack inputs having specific keywords (e.g., exists, type). Figure 16 shows an example of regular expression for remote file inclusion that looks for php file.

```
<![CDATA[(?:\<\w*:?\s(?:[^\>]*)t(?!rong)) | (?:\<scri)|(<\w+:\w+)]]>
```

Figure 14. Regular expression for Cross-Site Scripting

```
<![CDATA[(?:\[\$(?:ne|eq|lte?|gte?|n?in|mod|all|size|exists|type|slice|or)\
])]]>
```

Figure 15. Regular expression for SQL Injection

```
<![CDATA[(?:@[\w]+\s*\()|(?:]\s*\(\s*["!]\s*\w)|(?:<[?%](?:php)?.*(?:[
?%]>)?)|(?:;[\s\w|]*\$\w+\s*=)|(?:\$\w+\s*=(?:(?:\s*\$?\w+\s*[(;])|\s*".*
"))|(?:;\s*\{\W*\w+\s*\()]]>
```

Figure 16. Remote File Inclusion Example

To compare the GA-based IDS with PHPIDS, we consider the population set generated by GA and then convert back to string representation of attack inputs. We then pass these inputs to PHPIDS and see if all of them can be detected by PHPIDS. Figures 17 and 18 show example performances of the PHPIDS. In both cases, as the GA generated a greater population (of signatures), the PHPIDS failed to detect all of them. Thus, GA can be a complementary to PHPIDS to detect new attacks using a signature based approach.

5. CONCLUSION

This work contributed mutated signatures to a GA-based IDS where a set of web logs are converted to chromosomes and new attack signatures are generated. The approach addresses one current limitation of signature-based IDS, which is the number of signatures are limited and lack of variation may cause miss an attack in traffic log. We evaluated our approach with a generated attack dataset from a large scale PHP application.

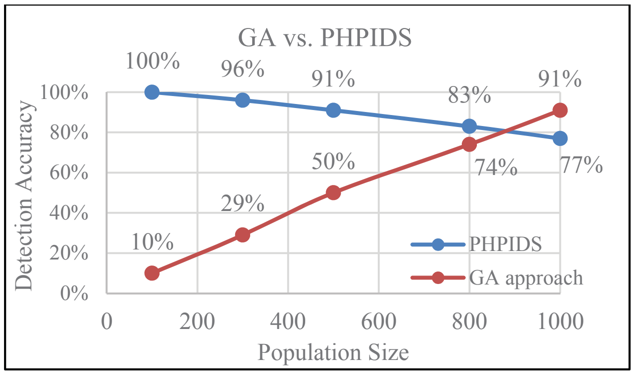


Figure 17: Performance of PHPIDS for GA generated signatures (cross over 10%, mutation 0.5)

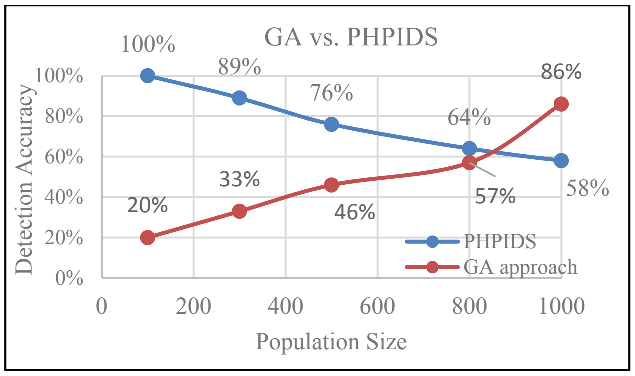


Figure 18: Performance of PHPIDS for GA generated signatures (cross over 20%, mutation 0.7)

The initial results find that the use of a GA is promising and can act as complementary to other existing signature-based IDS approaches. The increased the population size of chromosomes (representing rules), the better it achieves capability to detect new attacks. Further, having increased selection rate and mutation rate, we can generate new attack detection rules that can address the limitation of traditional signature-based IDS such as PHPIDS. We plan to include other web application attack types in our future work. We also plan to explore more varieties of evolutionary algorithms for comparison of performance.

6. REFERENCES

[1] Massicotte, F.; Labiche, Y., "On the Verification and Validation of Signature-Based, Network Intrusion Detection Systems," in Software Reliability Engineering (ISSRE), 2012 IEEE 23rd International Symposium, pp.61-70, 27-30 Nov. 2012 doi: 10.1109/ISSRE.2012.16

[2] Y. Meng; W. Li; L. Kwok, "Design of Cloud-Based Parallel Exclusive Signature Matching Model in Intrusion Detection," IEEE International Conference on Embedded and Ubiquitous Computing (HPCC_EUC), pp.175-182, Nov. 2013. doi: 10.1109/HPCC.and.EUC.2013.34

[3] Snort 2.8.9.0, Accessed from https://www.snort.org

[4] The Bro Network Security Monitor, Accessed from https://www.bro.org/

[5] Holm, H., "Signature Based Intrusion Detection for Zero-Day Attacks: (Not) A Closed Chapter?," in System Sciences (HICSS), 2014 47th Hawaii International Conference on , vol., no., pp.4895-4904, 6-9 Jan. 2014 doi: 10.1109/HICSS.2014.600

[6] Neelakantan, S.; Rao, S., "A Threat-Aware Signature Based Intrusion-Detection Approach for Obtaining Network-Specific Useful Alarms," in Internet Monitoring and Protection, 2008. ICIMP '08. The Third International Conference, pp.80-85, June 29 2008-July 5 2008 doi: 10.1109/ICIMP.2008.24

[7] C. Kruegel, and T. Toth. 2003. Using Decision Trees to Improve Signature-Based Intrusion Detection. In Recent Advances in Intrusion Detection. Pittsburgh, Pennsylvania: Springer Link, 173–191.

[8] Modi, C. N., Patel, D. R., Patel, A. and Rajarajan, M. Integrating Signature Apriori based Network Intrusion Detection System (NIDS) in Cloud Computing. *Procedia Technology*, 62(12), 905-912.

[9] Gupta, M.K., Govil, M.C., Singh, G., Sharma, P. XSSDM: Towards detection and mitigation of cross-site scripting vulnerabilities in web applications. *In Advances in Computing, Communications and Informatics (ICACCI), 2015 International Conference,* 2010-2015, 2015. doi: 10.1109/ICACCI.2015.7275912

[10] Y. Meng; W. Li; L. Kwok, "Design of Cloud-Based Parallel Exclusive Signature Matching Model in Intrusion Detection," in High Performance Computing and Communications & 2013 IEEE International Conference on Embedded and Ubiquitous Computing (HPCC_EUC), 2013 IEEE 10th International Conference, pp.175-182, 13-15 Nov. 2013. doi: 10.1109/HPCC.and.EUC.2013.34

[11] T. Parwani, R. Kholoussi, and P. Karras. 2013. How to hack into Facebook without being a hacker. In *Proceedings of the 22nd International Conference on World Wide Web* (WWW '13 Companion). International World Wide Web Conferences Steering Committee, Republic and Canton of Geneva, Switzerland, 751-754.

[12] G. Vigna, W. Robertson, and D. Balzarotti. 2004. Testing network-based intrusion detection signatures using mutant exploits. In Proceedings of the 11th ACM conference on Computer and communications security (CCS '04). ACM, New York, NY, USA, 21-30. doi: 10.1145/1030083.1030088

[13] T. Chou, "Security Threats on Cloud Computing Vulnerabilities," International Journal of Computer Science & Information Technology, Vol. 5, Issue 3, 2013, pp. 79–88. doi:10.5121/ijcsit.2013.5306

[14] M. Heiderich, J. Schwenk, T. Frosch, J. Magazinius, and E. Z. Yang. 2013. mXSS attacks: attacking well-secured web-applications by using innerHTML mutations. *Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security* (CCS '13). ACM, New York, NY, USA, 777-788. doi: 10.1145/2508859.2516723

[15] MacVittie, L. 2013. The Application Delivery Firewall Paradigm. F5 Networks, Inc. White Paper. https://f5.com/fr/resources/white-papers/the-application-delivery-firewall-paradigm

[16] C. Kruegel and G.Vigna, "Anomaly detection of web-based attacks," *Proceedings of the 10th ACM conference on Computer and communications security* (CCS '03). ACM, New York, NY, USA, 251-261. doi: 10.1145/948109.948144

[17] Buja, G.; Bin Abd Jalil, K.; Bt Hj Mohd Ali, F.; Rahman, T.F.A., "Detection model for SQL injection attack: An approach for preventing a web application from the SQL injection attack," Proc. of IEEE Symposium of Computer Applications and Industrial Electronics (ISCAIE), April 2014, pp.60-64, doi: 10.1109/ISCAIE.2014.701021

[19] H. F. G. Robledo, "Types of Hosts on a Remote File Inclusion (RFI) Botnet," Electronics, Robotics and Automotive Mechanics Conference, 2008. CERMA '08, Morelos, 2008, pp. 105-109. doi: 10.1109/CERMA.2008.60

[20] The Open Web Application Security Project. Top Ten 2013.

[21] F. Schuster and T. Holz. "Towards reducing the attack surface of software backdoors," Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security (CCS '13). ACM, New York, NY, USA, 851-862. doi: 10.1145/2508859.2516716
https://www.owasp.org/index.php/Top_10_2013

[22] Najafabadi, M.M.; Khoshgoftaar, T.M.; Kemp, C.; Seliya, N.; Zuech, R., "Machine Learning for Detecting Brute Force Attacks at the Network Level," IEEE International Conference on Bioinformatics and Bioengineering (BIBE), 2014, pp.379-385, doi: 10.1109/BIBE.2014.73

[23] F. Hsu, F. Guo, and T. Chiueh, "Scalable network-based buffer overflow attack detection," Proceedings of the 2006 ACM/IEEE symposium on Architecture for networking and communications systems (ANCS '06). ACM, New York, NY, USA, 163-172. doi: 10.1145/1185347.1185370

[24] A. Avancini and M. Ceccato. 2010. Towards security testing with taint analysis and genetic algorithms. In *Proceedings of the 2010 ICSE Workshop on Software Engineering for Secure Systems* (SESS '10). ACM, New York, NY, USA, 65-71. doi: 10.1145/1809100.1809110

[25] M. Barati, K. Faez, and Z. Hakimi. 2013. A novel threshold-based scan detection method using genetic algorithm. In *Proceedings of the 6th International Conference on Security of Information and Networks* (SIN '13). ACM, New York, NY, USA, 436-439. doi: 10.1145/2523514.2523580

[26] Y. Danane and T. Parvat, "Intrusion detection system using fuzzy genetic algorithm," *Pervasive Computing (ICPC), 2015 International Conference on*, Pune, 2015, pp. 1-5. doi: 10.1109/PERVASIVE.2015.7086963

[27] B. A. Fessi, S. BenAbdallah, M. Hamdi and N. Boudriga, "A new genetic algorithm approach for intrusion response system in computer networks," Computers and Communications, 2009. ISCC 2009. IEEE *Symposium on*, Sousse, 2009, pp. 342-347. doi: 10.1109/ISCC.2009.5202379

[28] L. Zhou and F. Liu, "Research on computer network security based on pattern recognition," *Systems, Man and Cybernetics, 2003. IEEE International Conference on*, 2003, pp. 1278-1283 vol.2. doi: 10.1109/ICSMC.2003.1244587

[29] S. Liu and Y. Fang, "Application research in computer network security evaluation based on genetic algorithm," *Instrumentation & Measurement, Sensor Network and Automation (IMSNA), 2012 International Symposium on*, Sanya, 2012, pp. 468-470. doi: 10.1109/MSNA.2012.6324623

[30] D. Narsingyani and O. Kale, "Optimizing false positive in anomaly based intrusion detection using Genetic algorithm," *MOOCs, Innovation and Technology in Education (MITE), 2015 IEEE 3rd International Conference on*, Amritsar, 2015, pp. 72-77. doi: 10.1109/MITE.2015.7375291

[31] B. Senthilnayaki, K. Venkatalakshmi and A. Kannan, "Intrusion detection using optimal genetic feature selection and SVM based classifier," *Signal Processing, Communication and Networking (ICSCN), 2015 3rd International Conference on*, Chennai, 2015, pp. 1-4. doi: 10.1109/ICSCN.2015.7219890

[32] M. Obitko, Genetic algorithm. Courses.cs.washington.edu, 1998.https://courses.cs.washington.edu/courses/cse473/06sp/GeneticAlgDemo/gaintro.html

[33] R. Malhotra, N. Singh and Y. Singh. Genetic Algorithms: Concepts, Design for Optimization of Process Controllers. CIS 4, 2 (2011).

[34] Jacobson, L. Creating a genetic algorithm for beginners. The Project Spot, 2012. http://www.theprojectspot.com/tutorial-post/creating-a-genetic-algorithm-for-beginners/3.

[35] S. Zaman, M. El-Abed, and F. Karray. 2013. Features selection approaches for intrusion detection systems based on evolution algorithms. In *Proceedings of the 7th International Conference on Ubiquitous Information Management and Communication* (ICUIMC '13). ACM, New York, NY, USA, Article 10, 5 pages. doi: 10.1145/2448556.2448566

[36] R. Gaucher, PHPIDS: Scalp!, (2008), GitHub repository, https://github.com/nanopony/apache-scalp

[37] R. Bronte, H. Shahriar and H. Haddad. "Information Theoretic Anomaly Detection Framework for Web Application," Proc. of 40th IEEE International Computer and Software Application Workshops (COMPSAC), Atlanta, GA June 10-14, 2016, 6 pp. (to appear).

[38] Joomla CMS, https://www.joomla.org

[39] OWASP, https://www.owasp.org/index.php/Main_Page

[40] T. Brandao, Genetic algorithms in PHP code, an example of evolutionary programming, (2015), Personal programming blog, http://www.abrandao.com/2015/01/simple-php-genetic-algorithm/