DRLGENCERT: Deep Learning-based Automated Testing of Certificate Verification in SSL/TLS Implementations

Chao Chen*, Wenrui Diao†✉, Yingpei Zeng‡, Shanqing Guo*✉, and Chengyu Hu*
*Shandong University, Jinan, China
Email: 1163307648@mail.sdu.edu.cn, {guoshanqing, hcy}@sdu.edu.cn
†Jinan University, Guangzhou, China
Email: diaowenrui@link.cuhk.edu.hk
‡China Mobile (Hangzhou) Information Technology Co., Ltd., Hangzhou, China
Email: zengyingpei@cmhi.chinamobile.com

***Abstract*—The Secure Sockets Layer (SSL) and Transport Layer Security (TLS) protocols are the foundation of network security. The certificate verification in SSL/TLS implementations is vital and may become the "weak link" in the whole network ecosystem. In previous works, some research focused on the automated testing of certificate verification, and the main approaches rely on generating massive certificates through randomly combining parts of seed certificates for fuzzing. Although the generated certificates could meet the semantic constraints, the cost is quite heavy, and the performance is limited due to the randomness.**

To fill this gap, in this paper, we propose DRLGENCERT, the first framework of applying deep reinforcement learning to the automated testing of certificate verification in SSL/TLS implementations. DRLGENCERT accepts ordinary certificates as input and outputs newly generated certificates which could trigger discrepancies with high efficiency. Benefited by the deep reinforcement learning, when generating certificates, our framework could choose the best next action according to the result of a previous modification, instead of simple random combinations. At the same time, we developed a set of new techniques to support the overall design, like new feature extraction method for X.509 certificates, fine-grained differential testing, and so forth. Also, we implemented a prototype of DRLGENCERT and carried out a series of real-world experiments. The results show DRLGENCERT is quite efficient, and we obtained 84,661 discrepancy-triggering certificates from 181,900 certificate seeds, say around 46.5% effectiveness. Also, we evaluated six popular SSL/TLS implementations, including GnuTLS, MatrixSSL, MbedTLS, NSS, OpenSSL, and wolfSSL. DRLGENCERT successfully discovered 23 serious certificate verification flaws, and most of them were previously unknown.

I. INTRODUCTION

The Transport Layer Security (TLS) [15] and its predecessor Secure Sockets Layer (SSL) [18] protocols are the foundation of network security. They are designed to provide security and data integrity assurance for Internet communications. During the SSL/TLS communication, the X.509 certificate is used to authenticate the identity of communicating party. The semantics and syntax of X.509 certificates have many limitations, described semi-formally in dozens of IETF's RFCs (including RFC 2246 [13], 2527 [11], 2818 [26], 4346 [14], 5246 [15], 5280 [12], 6101 [18], and 6125 [27]). Verifying the validity of a certificate is a complex process that includes verifying each certificate in the certificate chain, checking period of validity, public key, extensions, and so forth.

There are many open source SSL/TLS implementations available online, and developers can use these implementations for certificate validation without implementing the code by themselves. However, since the certificate validation is complicated, described in many tedious documents, the authors of SSL/TLS implementations may have their own understandings of how to code the logic. In other words, there is no guarantee for the correctness of these SSL/TLS implementations, which may become the "weak link" in the whole network ecosystem.

Some researchers have noticed this issue and tried to achieve the automated testing of certificate verification in SSL/TLS implementations [9], [10]. One of the mainstream approaches is the differential testing. If multiple certificate verification codes give different verification results for the same certificate, it means some of the certificate verification codes may be flawed. In the differential testing, the effectiveness of test results is decided by the quality of input test cases (i.e., modified certificates) directly. Better test cases could discover more design flaws with less time consumption.

In previous attempts, it is common to generate massive certificates (as test cases) by randomly combining parts of seed certificates, like Frankencert [9] and Mucert [10]. Although the generated certificates could meet the semantic constraints, the cost is quite heavy, and the performance is limited due to the randomness. It may generate a large number of unhelpful certificates which cannot trigger any flaw. On the other side, the deep learning technology shows the powerful capability of information mining and has been widely applied in biology, medicine, graphics, and cybersecurity, and so forth. We find the deep learning is suitable for the task of automated testing of certificate verification.

Our Approach. In this paper, we propose DRLGENCERT, the first framework of applying **D**eep **R**einforcement **L**earning to the automated testing of certificate verification in SSL/TLS implementations. DRLGENCERT accepts ordinary certificates

2576-3148/18/$31.00 ©2018 IEEE
DOI 10.1109/ICSME.2018.00014

as input and outputs the newly generated certificates which could trigger discrepancies with high efficiency. Benefited by the deep reinforcement learning, when generating certificates, our framework could choose the best next action according to results of previous modifications, instead of simple random combinations. At the same time, we developed a set of new techniques to support the overall design, like new feature extraction method for X.509 certificates, fine-grained differential testing module, and so forth.

In our research, we implemented DRLGENCERT and carried out a series of real-world experiments based on six popular SSL/TLS implementations, including GnuTLS [1], MatrixSSL [3], mbedTLS [4], NSS [5], OpenSSL [6], and wolfSSL [7]. The overall performance analysis shows DRLGENCERT is quite efficient, and we could obtain 84,661 discrepancy-triggering certificates from 181,900 certificate seeds after the first training episode of DRL network. It means more than 46.5% generated certificates are effective, which is better than all existing works. Also, DRLGENCERT successfully discovered 23 serious certificate verification flaws on six implementations, and most of them were previously unknown. For example, we found GnuTLS, NSS, and OpenSSL accept version 1, 2, or 4 certificates, even these v1, v2, v4 certificates have v3 extensions that should only exist in version 3 certificate, which violates the RFC documents [12]. The reason is that the version testing and extension testing functions are implemented as two independent components in code. We have reported our findings to the corresponding vendors, and they acknowledged that these issues are real and valuable.

Contributions. The main contributions of this paper are:

- *New framework.* We proposed DRLGENCERT, the first framework on applying deep reinforcement learning to the automated testing of certificate verification in SSL/TLS implementations. DRLGENCERT accepts normal certificates as input and outputs newly generated certificates (as the test cases of differential testing) which could trigger discrepancies with high efficiency.
- *New techniques.* We developed a set of new techniques to enable the overall design of DRLGENCERT, including new feature extraction method for X.509 certificates, fine-grained differential testing, and improved certificate modification actions.
- *Implementation and findings.* We implemented a prototype of DRLGENCERT and evaluated it through a series of real-world experiments. The overall performance analysis shows DRLGENCERT is quite efficient, and we obtained 84,661 discrepancy-triggering certificates from 181,900 certificate seeds. Also, DRLGENCERT successfully discovered 23 serious certificate verification flaws on six popular SSL/TLS implementations, and most of them were previously unknown.

Roadmap. The rest of this paper is organized as follows. Section II gives the background of certificate validation and deep reinforcement learning. Section III provides the overview design of DRLGENCERT. Section IV and Section V illustrate the differential testing module and deep reinforcement learning network module respectively. The evaluation results are summarized in Section VI. Section VII reviews the related work, and finally Section VIII concludes this paper.

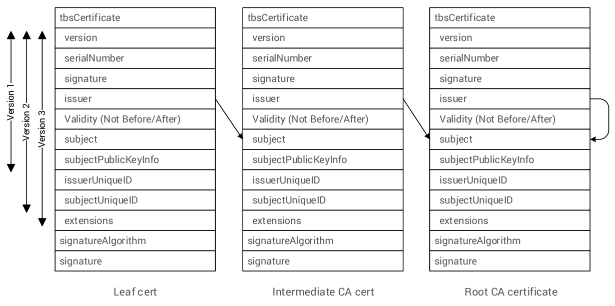

Fig. 1. General certificate chain structure.

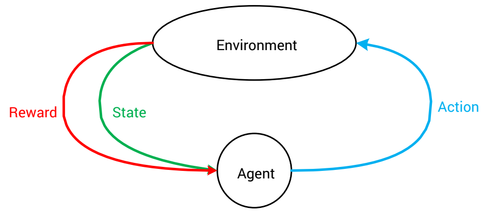

Fig. 2. The process of reinforcement learning.

II. Background

A. Certificate Validation

Certificate validation usually requires two inputs: trusted CA certificates and a chain of certificates to be validated. The general structure of a certificate chain is shown in Figure 1. Starting from the leaf certificate (end-entity certificate), each certificate is issued by a superior certificate until a self-signed CA certificate.

In the verification process, the leaf certificate in the certificate chain is usually verified first to confirm that the content of the certificate is valid, including the validity period and extension content. Then the SSL/TLS client checks whether the certificate is issued by a higher-level certificate in the certificate chain, including verifying the issuer, signature, and extension content. This process continues recursively until the certificate to be verified appears in the trusted certificate set.

B. Deep Reinforcement Learning

Deep reinforcement learning is a combination of deep learning and reinforcement learning, which enables robots to learn independently. The initial achievement is the Deep Q Learning algorithm proposed by DeepMind in 2013 [24].

In the field of artificial intelligence, an agent is used to represent a capable object, such as a robot, an unmanned vehicle, a person, and the like. The problem solved by reinforcement learning is to guide the interaction between the agent and environment. For example, playing a racing game on a computer, the game is the environment, and then we enter actions (keyboard operations) to control. What is observed on the screen is the state of the car and the score. The score is known as the reward, a factor in the reinforcement learning. No matter what kind of task, it contains a series of actions, observations, and feedback value-rewards. The observed information is the state of the agent. When the agent executes the action and interacts with the environment, the environment changes. The reward expresses the degree of good and bad caused by the change. Figure 2 illustrates the process of reinforcement learning. Reinforcement learning learns from previous states, actions, rewards to guide the choice of next action.

Combining deep learning with reinforcement learning can handle more complex tasks. Deep reinforcement learning, through continuous training, can get a policy that makes an agent get as much reward as possible in the task.

III. Overview of DRLgencert

In this paper, we propose DRLgencert, the first deep learning-based testing framework for automatic certificate validation in SSL/TLS implementations. DRLgencert accepts normal certificates as input and outputs the newly generated certificates which could trigger discrepancies with high efficiency. As shown in Figure 3, the DRLgencert framework consists of three main components: *a certificate set*, *the deep reinforcement learning network*, and *the differential testing module* (containing multiple certificate verification programs).

A. Settings

The reinforcement learning is a cyclic process in which an agent takes actions to change its state and interact with the environment to obtain a reward. In this process, the agent judges the merits and demerits of previous actions according to the reward given by the environment and obtains experience, so that it could choose a better behavior in the future same or similar state. In DRLgencert, we have the following settings:

- A certificate instance acts as an agent.
- Multiple SSL/TLS implementations are used as the environment.
- The state is defined as a certificate content feature.
- The action is defined as a certificate modification operation.
- The reward results from the validation results of multiple SSL/TLS implementations.

B. Framework Overview

DRLgencert starts to run from choosing a certificate instance from the prepared certificate set. Before passing the certificate to the deep learning network, the differential testing (see Section IV) is performed on the original certificate (Step 1 in Figure 3). If the test result shows that the certificate has reached the goal we want, namely, the certificate triggers a discrepancy (Step 2, 3, 4), there is no need to modify this certificate. Otherwise, we need to extract features from the certificate based on the pre-defined feature extraction scheme (Step 5, 6). The definition of certificate feature will be discussed in detail in Section V-A later, respectively. The neural network uses the certificate feature as input (Step 7), and the output represents the modification actions. Each number of the output represents the value of the corresponding modification action. As mentioned in Section II-B, the deep reinforcement learning network learns the feedback coming from the agent interacting with the environment by taking actions. In DRLgencert, the certificate modification actions are the actions taken by the deep reinforcement learning on certificates, and more details will be given in Section V-B.

During the training phase, we select the modification action to be adopted based on the ε-Greedy strategy. Each time we choose the most valuable modification action with 90% probability and randomly select a modification operation with 10% probability (Step 8). As an advantage, we could make sure that most modification operations can achieve the desired effect and also allow some certificate to explore new combinations of modification actions. In the testing phase of the neural network, we no longer need to explore new combinations of actions. Instead, we should guarantee the maximum validity of the modification actions. Therefore, the certificate modification action used during the usage phase is always the most valuable one in all modification actions.

After the certificate is modified, we obtain a new certificate (Step 9). This newly generated certificate will be inputted to the differential testing for multiple certificate verification programs (Step 10). After the differential testing, we obtained the verification result set of each verification program. According to the predefined reward definition scheme, the result is transformed to the corresponding reward (Step 2, 3, 11). The definition of reward will be given in Section V-C. In the DRLgencert framework, we set the range of reward for -1 and 100. If reward = 100, it means that the certificate achieves the target we want. It will be collected into our target certificate database (Step 4). Its content and verification result set may help us to analyze and discover possible flaws of certificate verification implementations. Then choose a new certificate from the database, and a new loop begins (Step 12). However, if reward = -1, it indicates that, under our settings, this certificate has no value in analyzing certificate verification implementations, and further change actions to the certificate content are required (Step 13). This process is described in Algorithm 1.

In our framework, we set $max\_modification$ (in Algorithm 1) to 9 and allow up to 10 changes per certificate, that is a maximum of 10 variants except for the initial certificate. If this certificate still cannot trigger a discrepancy, we will discard it and select a new one (Step 14). In our experiments, we found 95.5% of the certificates which get reward = 100 were modified less than 6 times. Therefore, if a certificate undoubtedly has the potential to trigger a discrepancy, 10 times is enough to trigger a discrepancy.

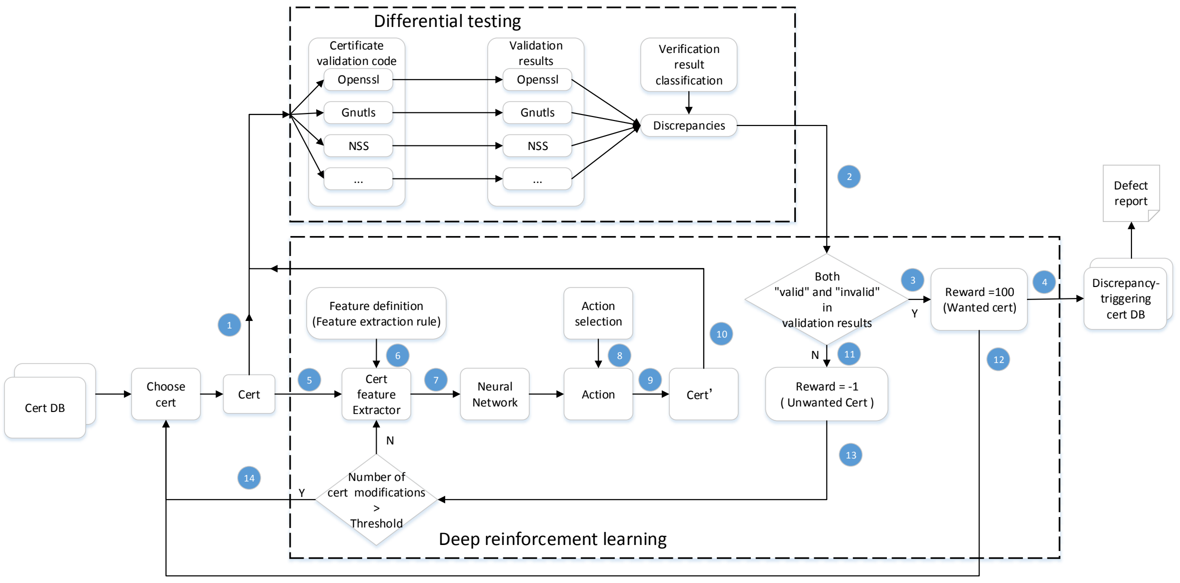
Differential testing
Certificate validation code
Openssl
Gnutls
NSS
...
Validation results
Openssl
Gnutls
NSS
...
Verification result classification
Discrepancies
Defect report
Both "valid" and "invalid" in validation results
Y
Reward =100 (Wanted cert)
Discrepancy-triggering cert DB
N
Reward = -1 (Unwanted Cert)
Feature definition (Feature extraction rule)
Action selection
Cert DB
Choose cert
Cert
Cert feature Extractor
Neural Network
Action
Cert'
N
Y
Number of cert modifications > Threshold
Deep reinforcement learning
1
2
3
4
5
6
7
8
9
10
11
12
13
14

Fig. 3. Overview of DRLGENCERT framework.

Algorithm 1 Modifying certificates

```
Input: $certDB, max\_episode, max\_modification$
Output: $target\_certDB$
 1: $episode \leftarrow 0$
 2: while $episode < max\_episode$ do
 3:     for $cert$ in $certDB$ do
 4:         $modification \leftarrow 0$
 5:         while $modification <= max\_modification$ do
 6:             $action \leftarrow$ NETWORK($cert$)
 7:             $cert \leftarrow$ TAKE_ACTION($cert, action$)
 8:             $reward \leftarrow$ DIFFERENTIAL_TESTING($cert$)
 9:             if $reward == 100$ then
10:                 add $cert$ to $target\_certDB$
11:                 $break$
12:             end if
13:             $modification \leftarrow modification + 1$
14:         end while
15:     end for
16:     $episode \leftarrow episode + 1$
17: end while
18: return $target\_certDB$
```

Through the above process, DRLGENCERT helps us quickly obtain a large number of certificates suitable for differential testing, based on the collected certificate data set.

IV. Differential Testing

Differential testing [23] (also known as differential fuzz testing) is a popular software testing technique that attempts to detect errors of a series of similar applications (or different implementations of the same application) by providing the same input and observing their execution differences.

TABLE I
VERIFICATION PROGRAMS AND SUPPORTED MODES

Program	Version	C-S Mode	File Mode (with verification utility)
GnuTLS	3.6.0	Y	**Y** (certtool)
MatrixSSL	3.9.3	Y	**Y** (certValidate)
MbedTLS	2.6.0	Y	**Y** (cert_app)
NSS	3.28.4	Y	**Y** (certutil)
OpenSSL	1.0.2g	Y	**Y** (openssl)
wolfSSL	3.12.2	**Y**	/

Remarks: The mode we used is labeled with bold **Y**.

Verification Program Setup. In DRLGENCERT, the initial certificates and modified certificates are both classified by six verification programs, including GnuTLS [1], MatrixSSL [3], mbedTLS [4], NSS [5], OpenSSL [6], and wolfSSL [7]. We used the latest versions of these programs (as listed in Table I) and deployed them on Ubuntu 16.04.

Most of these verification programs provide two verification methods: C-S mode and file mode. The only exception is wolfSSL which only supports the C-S mode. In the C-S mode, the client establishes a connection with the server, receives the certificate provided by the server, and validates it. Compared with the C-S mode, the file mode is more convenient. The program directly loads trust certificates and the certificate to be verified, then verifies it. In both modes, when they need to verify a certificate, they call the same functions to verify. The verification process is the same for both. However, because in the c-s mode, the program needs to establish a connection

	GnuTLS	MatrixSSL	MbedTLS	NSS	OpenSSL	wolfSSL
Cert	-2	-2	-4	-7	1	-10

Fig. 4. Example of result encoding.

between the client and server first, the operation is more tedious, and the connection may fail due to network reasons and the certificate verification section cannot be performed. In order to speed up the verification process and avoid the verification failure caused by the connection issue, we selected the file mode as the certificate verification method with a priority in DRLGENCERT. If the file mode is not available, we used the C-S mode. Table I shows the mode we selected in each program.

Verification Result Processing. Since each verification program has its own understanding of certificate verification, it leads to different expressions for the same error. Also, the meanings of the same error code from different programs are usually diverse. The granularity of verification results of different programs is not the same, and the verification result sets are also different.

Inspired by the work of Acer et al. [8], we designed a normalized solution to process the various verification results. In details, we grouped the verification results of each program into 16 categories, with a new error code, making it easy to redefine the reward and analyze verification codes. Table II shows the classification of verification results in our system, and the number in the second column ("*Error Code*") indicates the new encoding value of verification results. Due to the diverse granularity, sometimes multiple verification results from the same verification program are classified into same result type. If there are any validation results that have not been encountered in previous experiments, they are uniformly grouped into the type of "*other error*" and re-classified into an appropriate result category in next experiment through the updated classification policies.

Each verification program has its own understanding of the logic of certificate verification. Therefore, different programs may give different verification results for the same certificate, even giving the opposite results. If a certificate is given different verification results (i.e., rejected for different reasons), it means that the validation logic of the involved verification programs is different, and the logic of some programs may be flawed. As shown in the example of Figure 4, OpenSSL accepts the certificate, while other programs report different rejection reasons respectively.

To our framework, this certificate sample has the value of analysis. In DRLGENCERT, if there exist both acceptance and rejection in the verification results of a certificate, we define it as a discrepancy and record this certificate and its verification results for later analysis.

V. Deep Reinforcement Learning Network

The core part of DRLGENCERT is the deep reinforcement learning network, which is used to generate a large number of certificates suitable for differential testing.

Compared with other popular deep learning algorithms (e.g., CNN and LSTM), deep reinforcement learning is unsupervised, can learn from histories, and provide the optimal choice. The most famous deployment case is AlphaGo [28], every time it could select the optimal strategy based on the current situation. To be specific, we consider: sometimes the program is very sensitive to the changes in input content, just like the existence of adversarial examples [20] in image recognition field. That is, the image recognition program can correctly recognize a dog's photo, but if you make minor changes to the photos, the human cannot distinguish between the difference before and after the two photos. However, the image recognition program may identify the modified image as other objects, such as cars, just because of multiple subtle superimposed changes on several certain pixels. To certificates, we think that the certificate also has such nature that the programs are very sensitive to content changes in certificates even the change is very small. It is unlikely that a one-time modification makes a certificate get a different verify result, so the combination of multiple modifications is a better choice. Therefore, we chose deep reinforcement learning, as it can help us choose the best next action according to the result of a previous modification.

Based on the practical testing and our empirical knowledge, we deployed a three-layer neural network for DRLGENCERT. The concrete configuration parameters are listed in Table III.

Based on the classic definition of Mnih et al. [24], the loss function in DRLGENCERT is defined as Equation 1, in which:

- $reward$: After $cert$ being changed to $cert'$, $reward$ is gotten from differential testing.
- $\max(network(cert))$: Using a $cert$ as input, it is the max value in output.
- $\max(network(cert'))$: Using $cert'$ as input, it is the max value in output.
- γ : It is a constant, between 0 and 1. We set it to 0.9 in our experiment.

As shown in Figure 3, the neural network uses the certificate feature as input, and the output represents the modification actions. The reward (from the differential testing result) determines whether further actions are required. In the following sub-sections, we will discuss the designs of feature, modification action, and reward respectively.

A. Feature Extraction

Before making a certificate change, we need to characterize the certificate entity and use a vector to represent the content of the certificate. We refer to the feature extraction scheme of Dong et al. [16] and design a more accurate extraction scheme.

The certificate consists of a number of different content components. The value type of some parts is the number, others are string or bool. In the DRLGENCERT framework, we

TABLE II
VERIFICATION RESULTS CLASSIFICATION

Verification Result	Error Code	GnuTLS	MatrixSSL	MbedTLS	NSS	OpenSSL	wolfSSL
Valid	1	$\surd$	$\surd$	$\surd$	$\surd$	$\surd$	$\surd$
Unknown issuer	-1	$\surd$			$\surd*$	$\surd$	$\surd$
Validity period error	-2	$\surd*$	$\surd$	$\surd$	$\surd*$	$\surd$	$\surd$
Parsing error	-3	$\surd*$		$\surd$	$\surd*$	$\surd$	
Version error	-4		$\surd*$		$\surd$		$\surd$
Algorithm error	-5		$\surd*$				
Signature error	-6			$\surd*$	$\surd$	$\surd$	$\surd$
Subject/Issuer error	-7		$\surd*$	$\surd$			
Key usage error	-8		$\surd*$		$\surd$		$\surd$
Casic constraints error	-9					$\surd$	
Unknown critical extension	-10		$\surd$		$\surd$	$\surd$	
Chain error	-11				$\surd$	$\surd$	
Self sign	-12					$\surd$	
Connection error	-13						$\surd*$
Other extension error	-14		$\surd*$				
Other error	-15						

$\surd$ indicates that the verification result has been triggered by the program.
$*$ indicates that multiple different verification results are grouped into the category.

$$loss(cert) = \begin{cases} reward - \max(network(cert)) & \text{modifications are finished} \\ reward + \gamma * (\max(network(cert'))) - \max(network(cert)) & \text{modifications are unfinished} \end{cases} \quad (1)$$

TABLE III
NETWORK CONFIGURATION PARAMETERS

Layer	Input Dimension	Output Dimension	Activation Function
L_0: fully connected layer	101	100	ReLU
L_1: fully connected layer	100	100	ReLU
L_2: fully connected layer	100	86	/

extract the relevant features from the *version*, *issuer*, *subject*, *notafter*, *notbefore*, *public key length*, *signature algorithm*, and *extensions* of the certificate. A total of 101 numbers are used as the feature of a certificate, as the format of $\{2, 0, 1, 2, 1, -1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, \dots\}$.

Version. The version of the current mainstream certificate is 3. Version 1 and version 2 are also possible, but we can not limit version to [1, 2, 3] as we are not sure whether other versions of certificates exist and someone may deliberately make other versions of certificates. Therefore, DRLGENCERT directly uses the certificate version value as a feature.

Issuer and subject. In the certificate verification process, generally, only the issuer of the certificate and the subject of the superior certificate are checked to judge whether they are the same, without regard for the meaning indicated by the values of subject and issuer. However, it does not exclude that some processes of certificates verification codes are rather special, so we extract the value of the country of subject and issuer as certificate features, with number label to represent.

Notafter and notbefore. The validity period of certificate is compared with the current time, so we use the comparison result of the validity period and the current time as a feature. -1 means notbefore or notafter is before the current time. 0 means equal and 1 means they are after the current time.

Public key length. Public key length extraction scheme is similar to the processing of version. Its value is a numeric type, and cannot be guaranteed to ensure the value of public key length is in a certain range, so public key length uses its value as a feature.

Signature algorithm. Collect signature algorithm types that appear in the certificate set. Different numbers are used to label different signature algorithms.

Extensions. Extensions are optional content for certificates. The existence and type of extensions in a certificate are not fixed, and users can customize the types of their own extensions. Each extension must contain three contents: oid, iscritical, value. In DRLGENCERT system, we use three ways to extract extension features. According to the number of different types of extensions in collected certificate set and the difficulty of getting the content feature of a certain extension type, we choose different extraction methods for different kinds of extensions.

1) Get feature based on the extension's iscritical and value.
2) Get feature based on the extension's iscritical and isexist.
3) Ignore the extension.

Finally, in DRLGENCERT, we use an array of length 93 to represent extensions.

B. Certificate Modification Action

In our settings, if a certificate cannot trigger a discrepancy in the differential testing, namely, there is no difference between verification results of tested programs, we think the certificate and its validation results have no value for further analysis for validation codes. Then we will modify the content of the certificate, expecting the modified certificate will have the value we want. In DRLGENCERT, we designed a total of 86 modification actions, such as changing "Version", changing "Not before", deleting a certain extension. Each action modifies a part of the certificate and does not destroy its semantic constraints. At the same time, the modification operation will try to change the feature value of the certificate. In brief, in order to change content and feature value of a certificate, we designed these modification operations. We set a maximum of nine times of changes per certificate.

C. Reward

In the reinforcement learning, after an agent takes action to change its state, the environment will feedback a corresponding reward to show the pros and cons of the result generated by the action. In DRLGENCERT, the reward is obtained from the discrepancy's encoding. As described in Section IV, we reclassify the verification results of each validation program and encode them with new error codes, so we use a sequence of error codes to represent a discrepancy as its encoding.

In the preliminary experiment, we defined the reward as: assume that before the modification, the number of categories of cert_1's verification results in 6 tested programs is c_1. After the modification, cert_1 becomes cert_2. The number of categories of cert_2's verification results is c_2. Then we define reward as (c_2-c_1).

If the reward is greater than 0, or c_2 is equal to 6 (the number of verification programs), DRLGENCERT stops the process of modification and starts to select another certificate from the certificate data set. Because that if the reward is greater than 0, it means there are more types of verification results we achieve. The nature of discrepancy is clarified. DRLGENCERT helps to generate a discrepancy or make the discrepancy more distinctive. If reward = 6, then there is no need to make more changes. It has reached the maximum value.

However, when analyzing the generated discrepancies, we found that because of the limited manual efforts, this design is not suitable as it generates too many low-value discrepancies. Therefore, we redefined the reward. The new definition is: if acceptance and rejection both exist in the validation result, that is, there are 1 and non-1 in the validation result encoding, the reward is defined as 100. Otherwise, the reward is -1. During the training process, the neural network will be trained to get a reward as large as possible. Therefore, essentially, we only need to ensure that the reward value used to mark discrepancy is bigger than that used to mark non-discrepancy. In order to make neural network distinguish between discrepancy and non-discrepancy more distinctly, we expanded the gap between their rewards, empirically set to 100 and -1. When the reward is equal to 100, the certificate's modification process is over, as shown in Figure 3.

Comparing the effects of the two reward definitions, although the first strategy eventually obtains more discrepancies as defined in previous work, not every discrepancy has high analytical value as many discrepancies are encoded with different rejection reports. The program that accepts a certificate is highly likely to have flaws when all other programs reject the same certificate. Therefore, from the perspective of differential testing, the discrepancies containing both acceptance and rejection have more value for analysis than those only containing different rejection reports. The second definition helps us filter out these low-value discrepancies.

Since we used manual methods to analyze code based on discrepancies statically, the workload of code analysis is a key factor to consider in system design. If there are enough manual efforts, the first reward definition will be better for finding more flaws as the discrepancies consisting of different rejection reports may also contain hidden flaws. In our case, DRLGENCERT adopts the second reward definition, spending less time on more valuable discrepancies.

VI. RESULTS

We implemented DRLGENCERT and carried out a series of experiments to demonstrate its effectiveness.

A. Certificate Collection

In our experiment, we used ZMap [17] and OpenSSL [6] to collect certificates as seeds. With ZMap, we obtained 300,000 IP addresses that open the 443 port. Then, using OpenSSL s_client to connect to these IPs, we collected 181,900 certificate samples. One of the failure reasons for getting a certificate is that some IPs opened the 443 port but did not provide their certificates.

B. Certificate Generation

Table IV shows the types of discrepancies generated by DRLGENCERT in the experiment. Discrepancies are encoded in the format shown in Figure 4, and each type of discrepancy contains both acceptance and rejection (negative code are all considered as rejection).

Table V shows the discrepancy amount obtained by DRLGENCERT, and we could find the performance is quite efficient. Using the 181,900 certificates as the training set, DRLGENCERT got 84,661 discrepancy-triggering certificates (46.5% effectiveness) in first training episode of DRL network, taking around 206 hours.

Compared with Frankencert [9], our DRLGENCERT could get more discrepancy-triggering certificates. Based on the same 181,900 certificates, Frankencert only achieved 54,791 discrepancy-triggering certificates (30.1% effectiveness) in a total of 181,900 outputted certificates, with around 46 hours. We also tried to deploy Mucert [10] and compared the results. However, Mucert was not maintained anymore, and the deployment procedure is not very friendly, which requires the customized execution environment and program instrumentation. After multiple attempts, we still did not run Mucert successfully.

C. Flaws in Certificate Verification Code

Based on the generated discrepancies, we found 23 flaws in these tested SSL/TLS implementations, as listed in Table VI. Among them, there are 14 flaws not discussed before. These

TABLE IV
DISCREPANCIES GENERATED IN THE FIRST TRAINING EPISODE: TYPE

Sequence number	GnuTLS	MatrixSSL	MbedTLS	NSS	OpenSSL	wolfSSL
1	1	-14	-6	1	1	1
2	-1	-14	-6	-4	-1	1
3	1	-15	-6	1	1	1
4	-1	-15	-3	1	-1	-4
5	-3	-15	-3	1	1	-4
6	-3	-15	-3	-4	1	-4
7	-1	-15	-3	-10	-1	1
8	-3	-14	-3	-4	-1	1
9	-3	-15	-3	-4	-1	1
10	1	-14	-6	-4	-1	1
11	-14	-14	-6	-1	-1	1
12	1	-15	-3	-8	1	-4
13	-3	-14	-6	1	1	1
14	-3	-15	-3	-1	-1	1
15	-1	-8	-6	-4	-1	1
16	-1	-15	-3	1	-1	-13
17	-1	-15	-3	-1	-1	1
18	1	-14	-3	1	1	-4
19	1	-15	-6	-4	1	-13
20	1	-14	-6	1	-1	1
21	-3	-15	-6	-4	1	1
22	1	-14	-3	-4	1	-4
23	-3	-15	-6	-1	-1	1
24	-3	-15	-6	1	1	1
25	1	-15	-3	1	1	-4
26	1	-14	-3	1	1	1
27	1	-15	-3	-4	1	-13
28	-1	-15	-6	-4	-1	1
29	1	-10	-6	-4	1	-13
30	1	-15	-6	1	1	-13
31	-1	-14	-3	-4	-1	1
32	-3	-14	-6	-1	-1	1
32	-3	-14	-6	-1	-1	1
34	1	-15	-3	-4	1	-4
35	1	-10	-3	-4	-10	-13
36	1	-14	-3	1	-1	-4
37	1	-15	-3	1	-10	1
38	1	-14	-3	1	1	-13
39	-3	-15	-6	-4	-1	1
40	-3	-14	-6	1	1	-13
41	1	-15	-6	-8	1	-13
42	1	-8	-3	-4	-1	-4
43	-3	-15	-6	1	1	-13
44	1	-14	-3	-4	-1	-4
45	-1	-15	-6	-1	-1	1
46	1	-15	-3	-8	-10	-13
47	1	-15	-6	-4	1	1
48	1	-10	-6	1	1	-13
49	1	-10	-3	-10	-10	-13
50	1	-15	-3	-4	-10	-13
51	1	-15	-3	1	1	-13
52	1	-15	-3	-10	-10	1
53	1	-10	-3	1	-10	-13
54	-1	-15	-3	-4	-1	1
55	1	-14	-6	1	1	-13
56	-1	-8	-6	-1	-1	1
57	-14	-15	-3	1	1	-4
58	1	-15	-6	1	-1	1
59	-3	-15	-3	-4	-10	1
60	1	-15	-3	-8	1	-13
61	-1	-14	-6	-1	-1	1

The numbers represent different verification results as shown in Table II.

TABLE V
DISCREPANCIES GENERATED IN FIRST TRAINING EPISODE: AMOUNT

Certificate Size	Discrepancy Amount	Proportion (discrepancy/cert library)
6057	3605	59.5%
15179	9404	61.9%
84063	49881	59.3%
148939	78006	52.3%
181900	84661	46.5%

flaws may lead to severe security threats, such as man-in-the-middle attacks. We have reported all of them to the corresponding vendors, and the assessments are in process. Here we analyze some typical flaws as showcases.

1) Incorrect checking of time: The content of the certificate contains two timestamps: notbefore and notafter. When the certificate is validated, the current time must be after notbefore, before notafter. Otherwise, the certificate is invalid. The current time must be GMT (Greenwich Mean Time). After analysis, we found that in these 6 SSL/TLS implementations, both MatrixSSL and mbedTLS use the local system time instead of GMT.

Another finding is: according to RFC 5280 [12, 4.1.2.5], the time checking should check whether notbefore is earlier than the current time and whether notafter is later than the current time. However, MatrixSSL is not this case. The relevant code of time checking in MatrixSSL is as follows (Listing 1):

```
if (psBrokenDownTimeCmp(&beforeTime, &
    timeNowLinger) > 0)
{
/* beforeTime is in future. */
cert->authFailFlags |=
    PS_CERT_AUTH_FAIL_DATE_FLAG;
}
else if (psBrokenDownTimeCmp(&timeNow, &
    afterTimeLinger) > 0)
{
/* afterTime is in past. */
cert->authFailFlags |=
    PS_CERT_AUTH_FAIL_DATE_FLAG;
}
```

Listing 1: Time checking code in MatrixSSL (Part 1)

In Listing 1, `timeNowLinger` and `afterTimeLinger` are one day later than `timeNow` and `afterTime` respectively. The relevant code is shown in Listing 2.

```
/* The default value of allowed mismatch in times
    in X.509 messages and the local clock. The
   default value of 24 hours is mostly
   equivalent to old MatrixSSL behavior of
   ignoring hours, minutes and seconds in X.509
   date comparison. */
#  define PS_X509_TIME_LINGER (24 * 60 * 60)
...
memcpy(&timeNowLinger, &timeNow, sizeof
   timeNowLinger);
err = psBrokenDownTimeAdd(&timeNowLinger,
   PS_X509_TIME_LINGER);
```

Listing 2: Time checking code in MatrixSSL (Part 2)

TABLE VI
DISCOVERED FLAWS

Flaw Type	**GnuTLS**	**MatrixSSL**	**MbedTLS**	**NSS**	**OpenSSL**	**wolfSSL**
Incorrect checking of time		2	1			
Incorrect checking of v1/v2 with X.509 v3 extension	1	1		1	1	
Incorrect checking of v1/v2 intermediate certificate	1	1		1	1	
Incorrect checking of version	1			1	1	1
Incorrect checking of serial number	1			1	1	
Inappropriate error report	1	1		1	1	1
Total	**5**	**5**	**1**	**5**	**5**	**2**

As we can see from the above code, in MatrixSSL, the time check is unique. It compares notbefore with (current time + 1 day) and compares (notafter + 1 day) with the current time. Therefore, with MatrixSSL, the certificate will take effect one day in advance and postpone one day expired. In some scenarios, the lifetime of a certificate may be very short. For example, in Google's ALTS protocol [2], a handshake certificate usually is valid for only 20 hours. The error of extra one day before or after is undoubtedly a serious problem.

2) Incorrect checking of v1/v2 with X.509 v3 extension: According to discrepancy analysis, it is found that NSS, OpenSSL, and GnuTLS accept version 1 certificates with v3 extensions, wolfSSL rejects them. For version 2 certificates with v3 extensions, NSS, OpenSSL, GnuTLS, MatrixSSL accept them, and wolfSSL rejects them. According to the RFC documentation, *this field (v3 extensions) MUST only appear if the version is 3* [12, 4.1.2.9]. So, every v1 or v2 certificate with v3 extensions is unqualified and should not be trusted. Our analysis found that one of the main reasons for this is that in the program code, version testing and extension testing are two independent content. The program does not judge whether the extension can exist based on the version.

3) Incorrect checking of v1/v2 intermediate certificate: Applications may locally trust a v1 root CA. So, v1 root certificate can be trusted. However, all version 1 and version 2 intermediate certificates must be rejected unless they can be verified to be a CA certificate through out-of-band.

If there is a trusted version 1 or version 2 intermediate certificate, since there is no v3 extension to limit its scope, they can act as a rough CA. They can sign any domain name, any number of next level certificates. These subordinate certificates can be used for man-in-the-middle attacks and other bad intentions. The related statements exist in the official RFC documentation: *If certificate i is a version 1 or version 2 certificate, then the application MUST either verify that certificate i is a CA certificate through out-of-band means or reject the certificate. Conforming implementations may choose to reject all version 1 and version 2 intermediate certificates* [12, 6.1.4(k)].

In our test, we found that: NSS, OpenSSL, GnuTLS, MatrixSSL all accept v1/v2 intermediate certificates and consider these certificates valid. Only wolfSSL rejected it. MbedTLS rejected these because of other unrelated reasons, mainly failing to load the certificate.

4) Incorrect checking of version: V3 certificates are now widely used and accepted, and in some cases, v1 and v2 certificates are considered valid. Certificates of any other versions are all invalid. However, the test found GnuTLS, NSS, OpenSSL, wolfSSL will accept v4 certificates, although v3 is the latest version of the X.509 certificate.

5) Incorrect checking of serial number: One of the requirements for the serial number in the RFC documentation is: *the serial number MUST be a positive integer assigned by the CA to each certificate* [12, 4.1.2.2]. However, multiple certificates with negative sequence numbers were found in the generated discrepancy-triggering certificates, and they are trusted by GnuTLS, NSS, and OpenSSL.

Another requirement for serial numbers is: *Conforming CAs MUST NOT use serialNumber values longer than 20 octets* [12, 4.1.2.2]. This statement is somewhat ambiguous. The first meaning is: CA certificate must not contain serialNumber values longer than 20 octets. The second meaning is: CAs must not assign serialNumber values longer than 20 octets to next level certificate. In discrepancy-triggering certificates, there are certificates containing a serial number with a length of 37 octets, and GnuTLS, NSS, and OpenSSL accepted the certificate. If the RFC documentation requirements refer to the second meaning or both, GnuTLS, NSS, and OpenSSL obviously violate the RFC documentation requirements.

6) Inappropriate error report: In some cases, the same certificate may contain several different types of insecure contents that can result in varying degrees of security threats. The test found that in six programs, only mbedTLS gave multiple security warnings at the same time, and other programs reported only one security warning. The reason is that in the verification code of these programs when the first security threat is found, the verification code will directly return. This will introduce a security risk. If a certificate has many different security threats at the same time, for example, expiration for one week and self-signed, users will probably only receive a warning that the certificate expires one week. The bigger security threat self-signed is obscured. Some careless users, aware of the certificate expires only one week, likely choose to manually trust the certificate since it was just invalid, and this may result in a man-in-the-middle attack.

Inspired by this, we made a certificate, containing self-signed, expired, unstructured subject name. The unstructured subject name means the name of the subject in the certificate does not fit the document format, but this does not introduce a security problem. When verifying the certificate, we found that mbedTLS reported both self-signed and expired security warnings. OpenSSL and wolfSSL showed a self-signing warning. GnuTLS and MatrixSSL showed parsing errors (for the unstructured subject name), and we think this description is reasonable and sufficient since this implies the certificate has not been validated. However, NSS only displayed expiration warning. So from this perspective, the programs using NSS's service may suffer from the danger we previously assumed.

VII. Related Work

There are already efforts of detecting vulnerabilities in SSL/TLS implementations. In this section, we will review some typical works. Also, some application cases of deep learning in security areas will be retrospected.

A. Security of SSL/TLS Implementations

The SSL and TLS protocols are the foundation of modern network security. Whether SSL/TLS tools and libraries are correctly implemented are vital, and several previous works focused on validating their implementations.

Stone et al. [30] focused on the certificate hostname verification in TLS implementations. They showed that in security-sensitive applications, if the certificate pinning is used, it can hide the lack of proper hostname verification, enabling MITM attacks. They also presented a tool to detect such kind of flaw.

Sivakorn et al. [29] directly tested the hostname verification in TLS implementations. They presented HVLearn, a novel black-box testing framework for analysis, which is based on automata learning algorithms. Moreover, this framework helps to find several unique violations of the RFC specifications in hostname verification implementations.

Acer et al. [8] studied the causes of Chrome HTTPS certificate errors. Since hundreds of millions of spurious browser warnings are triggered per month, they frustrated users and undermined the trust in browser warnings, which may cause users to ignore the real warning. This finding shows that client-side or network issues caused more than half of errors.

Hawanna et al. [22] proposed a framework which assesses the risk associated with X.509 certificates. By using the random forest machine learning algorithm, this framework provides users with three risk levels, and mentions due to which parameter it bears the risk. Another example of applying machine learning methods to HTTPS security is the work of Dong et al. [16]. Unlike the above work, they did not focus on the flaws in TLS implementations. They aimed to design a method for distinguishing rogue certificates which are valid certificates issued by a legitimate certificate authority (CA) but untrustworthy. In order to implement this method, they built machine-learning models with Deep Neural Networks (DNN) to automate classification. Similar to their idea, inspired by the powerful performance of machine learning, we also apply machine learning to test SSL/TLS implementations. The difference is that we focus on the implementation errors and logic errors in the certificate validation phase. Also, we combine differential testing and deep reinforcement learning, which is different from the models used in the work of Hawanna et al. and Dong et al. and do not need to set the label for data set.

The most closely works to ours are Frankencert [9] and Mucert [10]. We use the same test method – differential testing to test the SSL/TLS implementations. However, since both Frankencert and Mucert generate new certificates by random combinations of parts of certificates, introducing too much randomness. As a result, most of the generated certificates are invalid or valueless for differential testing. In contrast, we are using new ways to get a large number of certificates with the information learned in a neural network. The experimental results confirmed that we can get more valuable certificates.

B. Applications of Deep Learning in Security

There are many applications of deep learning in security. One of the primary applications is to develop classifiers for security tasks. Prade et al. [25] presented a general framework, called DeepBugs which extracts positive training examples from a code corpus, leverages simple program transformations to create negative training examples, and trains a model to distinguish these two. Tsai et al. [31] reviewed 55 related studies focusing on developing single, hybrid, and ensemble classifiers. The work of Dong et al. mentioned above is also a specific example of such kind of applications.

Godefroid et al. [19] demonstrated using sample inputs and neural-network-based statistical machine-learning to automate the generation of an input grammar suitable for input fuzzing. The Portable Document Format (PDF) was used as the complex input format in a case study, and they used the recurrent neural network (RNN) [21] to capture the structure of PDF and then generate new PDF files to fuzz PDF parser.

VIII. Conclusion

In this paper, we presented a new idea of applying deep learning to differential testing of certificate validation. We introduce DRLgenCert, a framework using deep reinforcement learning to generate discrepancy-triggering test cases for testing SSL/TLS implementations. The results show that DRLgenCert is a valid and promising automatic system for generating test cases. After the first round of training, statistics show that it can modify certificates to discrepancy-triggering cases with a probability of more than 46%. By conducting experiments in the real environment, DRLgenCert successfully identified 23 certificate verification flaws on six popular SSL/TLS implementations.

IX. Acknowledgements

This work is partially supported by National Natural Science Foundation of China (91546203), the Key Science Technology Project of Shandong Province (2015GGX101046), the Shandong Provincial Natural Science Foundation (ZR2014FM020), Major Scientific and Technological Innovation Projects of Shandong Province, China (No. 2017CXGC0704), Fundamental Research Fund of Shandong Academy of Sciences (No.2018:12-16), and the Fundamental Research Funds for the Central Universities (No.21618330).

References

[1] GnuTLS. [Online]. Available: http://www.gnutls.org/

[2] Google Application Layer Transport Security (ALTS). [Online]. Available: https://cloud.google.com/security/encryption-in-transit/application-layer-transport-security/resources/alts-whitepaper.pdf

[3] MatrixSSL. [Online]. Available: http://www.matrixssl.org/

[4] mbedTLS. [Online]. Available: https://polarssl.org/

[5] NSS. [Online]. Available: https://developer.mozilla.org/en-US/docs/Mozilla/Projects/NSS

[6] OpenSSL. [Online]. Available: https://www.openssl.org/

[7] WolfSSL. [Online]. Available: http://www.yassl.com/yaSSL/Products-cyassl.html

[8] M. E. Acer, E. Stark, A. P. Felt, S. Fahl, R. Bhargava, B. Dev, M. Braithwaite, R. Sleevi, and P. Tabriz, "Where the Wild Warnings Are: Root Causes of Chrome HTTPS Certificate Errors," in *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security (CCS), Dallas, TX, USA, October 30 - November 03, 2017*, 2017.

[9] C. Brubaker, S. Jana, B. Ray, S. Khurshid, and V. Shmatikov, "Using Frankencerts for Automated Adversarial Testing of Certificate Validation in SSL/TLS Implementations," in *Proceedings of the 2014 IEEE Symposium on Security and Privacy (Oakland), Berkeley, CA, USA, May 18-21, 2014*, 2014.

[10] Y. Chen and Z. Su, "Guided differential testing of certificate validation in SSL/TLS implementations," in *Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering, ESEC/FSE 2015, Bergamo, Italy, August 30 - September 4, 2015*, 2015.

[11] S. Chokhani and W. Ford, "RFC 2527: Internet X.509 Public Key Infrastructure Certificate Policy and Certification Practices Framework," Internet Engineering Task Force (IETF), RFC, 1999.

[12] Cooper, Dave, "RFC 5280: Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile," 2008.

[13] T. Dierks and C. Allen, "RFC 2246: The TLS Protocol Version 1.0," Internet Engineering Task Force (IETF), RFC, 1999.

[14] T. Dierks and E. Rescorla, "RFC 4346: The Transport Layer Security (TLS) Protocol Version 1.1," Internet Engineering Task Force (IETF), RFC, 2006.

[15] ——, "RFC 5246: The Transport Layer Security (TLS) Protocol Version 1.2," Internet Engineering Task Force (IETF), RFC, 2008.

[16] Z. Dong, K. Kane, and L. J. Camp, "Detection of rogue certificates from trusted certificate authorities using deep neural networks," *ACM Transactions on Privacy and Security (TOPS)*, vol. 19, no. 2, p. 5, 2016.

[17] Z. Durumeric, E. Wustrow, and J. A. Halderman, "ZMap: Fast Internet-wide Scanning and Its Security Applications," in *Proceedings of the 22th USENIX Security Symposium, Washington, DC, USA, August 14-16, 2013*, 2013.

[18] A. Freier, P. Karlton, and P. Kocher, "RFC 6101: The Secure Sockets Layer (SSL) Protocol Version 3.0 ," Internet Engineering Task Force (IETF), RFC, 2011.

[19] P. Godefroid, H. Peleg, and R. Singh, "Learn&Fuzz: Machine Learning for Input Fuzzing," in *Proceedings of the 32nd IEEE/ACM International Conference on Automated Software Engineering, ASE 2017, Urbana, IL, USA, October 30 - November 03, 2017*, 2017.

[20] I. J. Goodfellow, J. Shlens, and C. Szegedy, "Explaining and Harnessing Adversarial Examples," *CoRR*, vol. abs/1412.6572, 2014. [Online]. Available: http://arxiv.org/abs/1412.6572

[21] A. Graves, M. Liwicki, S. Fernandez, R. Bertolami, H. Bunke, and J. Schmidhuber, "A Novel Connectionist System for Unconstrained Handwriting Recognition," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 31, no. 5, pp. 855–868, 2009.

[22] V. Hawanna, V. Kulkarni, R. Rane, P. Mestri, and S. Panchal, "Risk Rating System of X. 509 Certificates," *Procedia Computer Science*, vol. 89, pp. 152–161, 2016.

[23] W. M. McKeeman, "Differential testing for software," *Digital Technical Journal*, vol. 10, no. 1, pp. 100–107, 1998.

[24] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. A. Riedmiller, "Playing Atari with Deep Reinforcement Learning," *CoRR*, vol. abs/1312.5602, 2013. [Online]. Available: http://arxiv.org/abs/1312.5602

[25] M. Pradel and K. Sen, "Deep Learning to Find Bugs," TU Darmstadt, Tech. Rep., 2017.

[26] E. Rescorla, "RFC 2818: HTTP Over TLS," Internet Engineering Task Force (IETF), RFC, 2005.

[27] P. Saint-Andre and J. Hodges, "RFC 6125: Representation and Verification of Domain-Based Application Service Identity within Internet Public Key Infrastructure Using X.509 (PKIX) Certificates in the Context of Transport Layer Security (TLS)," Internet Engineering Task Force (IETF), RFC, 2011.

[28] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. Van Den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot *et al.*, "Mastering the game of Go with deep neural networks and tree search," *Nature*, vol. 529, no. 7587, pp. 484–489, 2016.

[29] S. Sivakorn, G. Argyros, K. Pei, A. D. Keromytis, and S. Jana, "HVLearn: Automated Black-Box Analysis of Hostname Verification in SSL/TLS Implementations," in *Proceedings of the 2017 IEEE Symposium on Security and Privacy (Oakland), San Jose, CA, USA, May 22-26, 2017*, 2017.

[30] C. M. Stone, T. Chothia, and F. D. Garcia, "Spinner: Semi-Automatic Detection of Pinning without Hostname Verification," in *Proceedings of the 33rd Annual Computer Security Applications Conference (ACSAC), Orlando, FL, USA, December 4-8, 2017*, 2017.

[31] C.-F. Tsai, Y.-F. Hsu, C.-Y. Lin, and W.-Y. Lin, "Intrusion detection by machine learning: A review," *Expert Systems with Applications*, vol. 36, no. 10, pp. 11 994–12 000, 2009.