Chapter 3

Datasets Used

3.1 Introduction

The Internet is a powerful platform which interconnects numerous individual networks worldwide. It provides public facing as well as internal backbone in various sectors such as government, private organizations, industry and academia. Since its inception, the Internet has come a long way and is now regarded as the storehouse of knowledge in every domain known to humankind. Literally, the Internet allows us to conceptualize the whole world as a 'global village'. Anyone from any part of the world can connect, communicate and share information with any individual. The current growth of the Internet is fueled by the explosion of activities at an unprecedented rate, primarily catalyzed by mobile technology and the social Web. Today, the wide range of devices connected to the Internet is overwhelming. Devices like cell phones, tablets, phablets, laptops, and netbooks serve as vehicles to surf the Internet and consequently generate huge amounts of Internet traffic regularly.

Right from its beginning, the Internet has been a communication hub. Lately, there has been an exponential increase in the number of devices that are connected to the Internet on a daily basis. This implies an increase in the number of users. In fact, as of 2022, 66% of the total worldwide population uses the Internet for various activities. The term *Global Internet Usage* refers to the worldwide count of Internet users[1]. Figure 3.1 gives the number of Internet users globally as of 31st December, 2022 which is a whopping 5.30 billion [2]. Figure 3.2 shows the top 10 countries with

[1] https://en.wikipedia.org/wiki/Global_Internet_usage

[2] https://www.statista.com/statistics/273018/number-of-internet-users-worldwide/

the largest Digital Population (in millions) as of January, 2023 [3]. In a world where

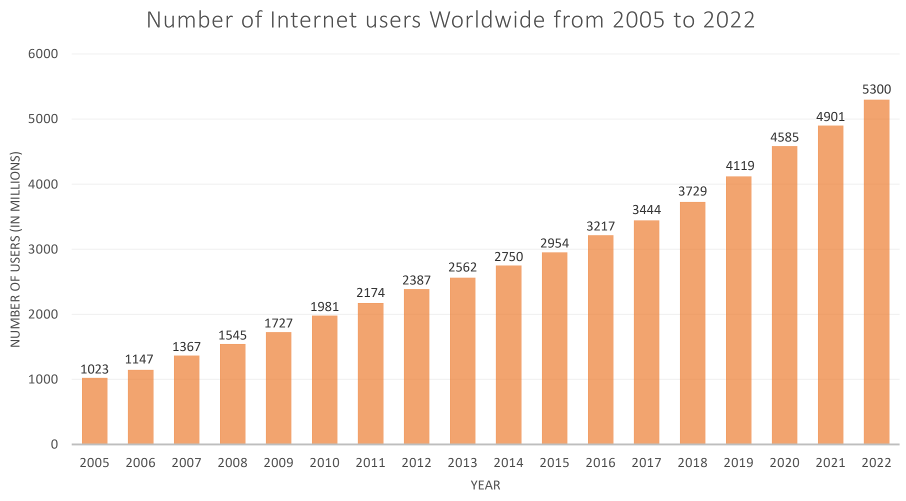


Figure 3.1: Worldwide Internet Users from 2005 to 2022

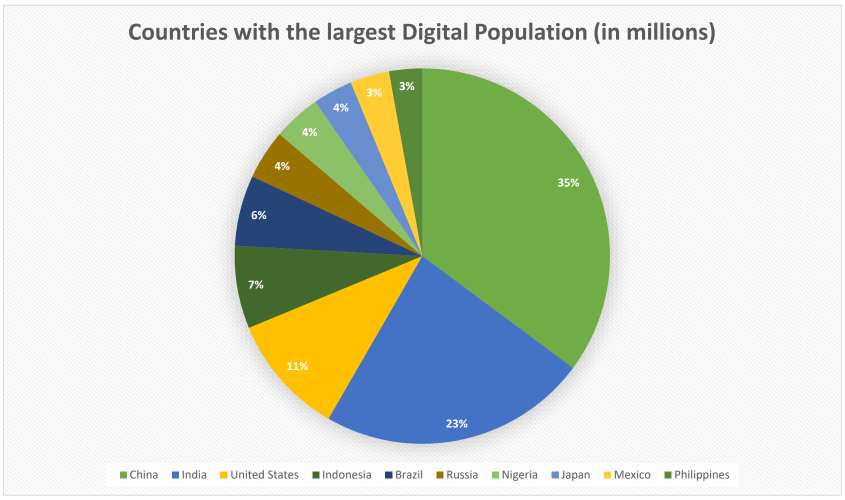


Figure 3.2: Top 10 Countries with Largest Digital Population

most communication occurs through digital means, security is one of the most crucial requirements which needs to be provided at every step. As more and more Web applications are designed and developed to ease our day-to-day activities, the risk of falling prey to cyber-attacks increases. In the worst case, a victim doesn't even know that he/she is under an attack. These attacks carried out by notorious users may focus on stealing sensitive information belonging to the victims, or may also hijack his or her Web sessions. Using the sensitive information so gathered, the attackers may easily impersonate the victim to carry out far more dangerous acts. To defend against such attacks, many Intrusion Detection Systems employ a

[3] https://www.statista.com/statistics/262966/number-of-internet-users-in-selected-countries/

variety of tactics. The developers of the IDSs focus on the detection of on-going attacks as well as prevent them in the future. To this end, a crucial aspect of how well an IDS performs is how well these systems have been evaluated using raw or feature datasets. In other words, training and testing the IDSs with appropriate datasets help establish its effectiveness. However, sometimes such datasets may not be readily available, in which case testbeds need to be constructed for generating the data.

3.1.1 Desired Characteristics of a Dataset

Web-based attacks have garnered significant attention from the research community, and numerous detection mechanisms have been proposed, each with their own pros and cons. Primarily, datasets serve as input to these defense mechanisms to test how effective they are. The nature and type of the input dataset may vary from one defense mechanism to another. The caliber of such an input dataset plays a remarkable role in evaluating the potency of a defense system. It is also important to note that, with the continuous increase in attack instances each day, dataset should be maintained so as to incorporate the most recent attack experience. Below some significant characteristics that a standard dataset should have are discussed.

1. *Labels in the Data*: The labels (class information) of the data instances in the dataset may be any of the predefined classes. For example, if it is a 2-class problem, the data instances may belong to either class A or class B, assuming the two predefined classes are A and B. Importantly, obtaining extensive amount and varieties of data that collectively capture the profile of benign or malicious behavior is hard, in addition to being expensive. Human experts have to expend substantial effort in classifying what data is regarded as malicious and what is regarded as benign. It is worth noting here that data instances that are termed malicious are harder to collect than benign instances, as raw malicious instances are removed quickly for ethical security reasons. In addition, with new and evolving techniques, malicious instances (or the activity that leads to malicious instances) may be easily masqueraded as normal behavior.
   Depending on whether a dataset consists of prior labels or not, a defense system may use two modes: supervised or unsupervised, to learn characteristics of the data.

Supervised techniques [55] make use of labeled datasets in the training phase. A dataset comprising of both labeled malicious and benign instances is used to construct a classification model. In the testing phase, an unknown instance is classified into either the malicious or benign class.

Unsupervised techniques [58] do not make use of labeled datasets. Such techniques group similar data instances into a cluster. Instances belonging to the same cluster are more similar to each other than instances in the other clusters. However, such techniques may fall prey to high false alarm rate.

2. *Adequate number of instances*: A dataset generally should consist of an adequate number of instances so that the defense mechanism receives proper and sufficient training. This assists the defense mechanism build an effective model to conclude under what circumstances an unknown instance should be classified into a particular class.

3. *Adequate number of features*: Features are the backbone of a standard dataset. Meaningful and non-redundant features collectively describe the characteristics or profile of a particular class. In other words, the features should be picked in such a way that they sufficiently describe the characteristics of the problem. The features should have a minimum correlation score among themselves (feature-feature correlation) and a maximum correlation score with the class (feature-class correlation). Too many features to define the profile of a class may lead to over fitting, and too few features may fail to define a class profile properly, hence not sufficient. Thus, there should be an adequate number of features such that the class profile is properly defined as well as over fitting is also avoided.

4. *Balance between the classes*: Balance between the number of instances in each class is important for proper and complete training involving each of the classes. Balance means the occurrence of roughly equal number of samples in each of the classes. If there is an imbalance in the data between the classes, a predictive model will tend to learn more about a particular class than the other. This may result in misclassification of the instances. The problem of class imbalance can be dealt by sampling techniques like oversampling or undersampling.

5. *Recency*: The data constituting the dataset should not be outdated. They should be from recent times so that they cover the newer, more evolved attack

strategies. For instance, if the data is outdated and the defense mechanism is trained using such data, it will fail to identify newly evolved unknown attack instances. So, the dataset should be updated from time to time to incorporate recent relevant data.

6. *Lack of inconsistency*: Data in a standard dataset should be consistent. Defense mechanisms validated using inconsistent data are never reliable and will result in wrong decision making.

7. *Relevance*: The features present in the dataset should be relevant to the problem at hand. Irrelevant features may bring down the performance of the defense mechanism. The feature-to-class relevance should be high and the feature-to-feature correlation should be low. Only then good performance can be expected. This is because, some features may be more important to identify a particular class whereas some features may be not.

8. *Completeness*: Incompletely labeled instances for a given class often leads to failure of most classifiers due to inadequate training. The performance of a supervised learning system mostly depends on quality of the training data instances or samples. So, lack of completeness in a dataset because of the number of instances for any class is not desirable.

3.2 Benchmark Datasets

A dataset plays a key role in evaluating the potency of a defense system. The efficiency of a defense system in terms of accuracy can be evaluated using a relevant and quality dataset. In other words, to assess a defense system appropriately a quality dataset is of paramount importance. A discussion on the datasets used for evaluation of the proposed methods in this thesis are presented below.

3.2.1 Cross-site Scripting Attack Repositories

For Cross-site scripting attacks, there are some well-known repositories such as the XSSed repository and publicly available raw Web archive files as mentioned below. It is convenient if the datasets are divided into two categories: Script-based and URL-based datasets.

1. Script-based datasets: Web crawlers can be used to crawl Web pages for the collection of raw scripts. From these collected scripts, one can extract features with the help of C routines. However, there is no publicly available script-based XSS feature dataset. It is worth mentioning [132] here, which is a repository of malicious scripts. The repository consists of various obfuscated and corresponding deobfuscated malicious JavaScripts. Due to the unavailability of a quality dataset, an urgent need arises for the creation of such a dataset and making it public.

2. URL-based datasets: A URL-based raw dataset consists of various benign and malicious URLs that may be collected from numerous sources. A URL feature dataset consists of URL-based features which helps to distinguish malicious URLs from benign URLs. A few examples of URL datasets are discussed below.

   (a) URL Reputation Data Set [133]: It is a feature dataset which consists of 2.4 million instances and 3.2 million odd features. The instances were collected in real time over a period of 120 days. The anonymized lexical and host-based features are all of integer and real types. A label of +1 denotes a malicious instance and a label of -1 denotes a benign instance. A subset of the original dataset was made publicly available by the authors in [134]. The dataset has been used in research such as [135] and [136].

   (b) XSSed repository [29]: This repository consists of URLs of domains which have already fallen prey to XSS attacks. Along with the URLs, the authors also provide a mirror and additional information if a site has already been fixed. The pages are ranked in accordance with Alexa ranking. Security researchers from various parts of the world contribute to populate the archive. However, the archive has not been maintained since March 2015. The repository has been used in research efforts such as in [60] and many more.

3.2.2 HTTP Flooding Attack Datasets

For HTTP Flooding attacks, there are a few well-known benchmark datasets which are detailed below. In addition to discussing the datasets, a list of open source tools for generating normal and attack traffic is also illustrated in Table 3.1.

Table 3.1: HTTP-Flood Traffic Generator Tools

Tool Name	Programming Platform	Reference	Additional Information
GoldenEye	Python3	`https://github.com/jseidl/GoldenEye`	An HTTP DoS test tool
HULK (HTTP Unbearable Load King)	Python	`https://github.com/R3DHULK/HULK`	
Slowloris	rewritten in Python	`https://github.com/gkbrk/slowloris`	Periodically sends requests to the server
HOIC (High Orbit Ion Cannon)	Visual Basic, C#	`https://sourceforge.net/projects/highorbitioncannon/`	Can send both HTTP GET and POST requests. Can flood 256 websites simultaneously. Available attack intensity: Low, Medium and High
LOIC (Low Orbit Ion Cannon)	C#	`https://sourceforge.net/projects/loic/`	
PyLoris	Python	`https://sourceforge.net/projects/pyloris/`	Scriptable HTTP Dos tool
Slowhttptest		`https://github.com/shekyan/slowhttptest`	Highly configurable
TorsHammer	Python	`https://github.com/Karlheinzniebuhr/torshammer`	Generates slow-rate HTTP-POST traffic
R.U.D.Y (R-U-Dead-Yet)		`https://github.com/sahilchaddha/rudyjs`	Generates HTTP POST traffic by submitting form data in web applications. Able to generate slow rate HTTP traffic
DDoSim	C++	`https://sourceforge.net/projects/ddosim/`	

1. UNSW-NB15 dataset: The authors in [137] use the IXIA[4] traffic generator tool to generate both attack and normal traffic. The collection of traffic took place in two simulations (totaling 31 hours), each simulation capturing 50 GB of data. It is important here to note that the collected traffic mimics both a real modern normal network activity and a synthesized attack network activity. From the captured *.pcap* files a total of 49 features are extracted using different tools such as Bro-IDS[5] and Argus[6]. The different categories of features in the dataset are: flow features, basic features, content features, time-based features, and some additionally generated features. Overall the dataset contains a total of 25,40,044 records of which 22,18,761 are from normal network activity. For simplicity, in the later chapter (Chapter 6) this dataset is named as UNSW dataset only. A brief description of the dataset is given in Table 3.2.

2. CICIDS2017 dataset: This dataset is proposed by the Canadian Institute of Cyber Security to effectively evaluate an IDS [138]. It contains five days of both attack and normal traffic spanning across the victim and attack network. To generate the attack traffic different attack generation tools are used such as Hulk, GoldenEye, and Slowloris to name a few. For feature extraction (from captured .pcap files) the authors used CICFlowMeter[7]for extracting a total of 80 traffic features. Overall the dataset contains 28,30,540 of which 23,59,087

[4] `http://www.ixiacom.com/products/perfectstorm`
[5] `https://www.bro.org/index.html`
[6] `http://qosient.com/argus/index.shtml`
[7] `https://www.unb.ca/cic/research/applications.html`

Table 3.2: Brief Description of UNSW-NB15 Dataset

Parameter name	Value
No. of networks	3
Simulation	Yes
Simulation tool	IXIA tool
No. of distinct IP adresses	45
Format of the collected data	pcap files
Duration of data collection	31
Feature extraction tools	Argus, Bro-IDS
No. of extracted features	49
Feature categories	Flow-based Basic Content-based Time-based Additional
No. of records	25,40,044
Attack families	9

are normal samples and rest 4,71,453 are attack samples. It is to be noted that the dataset contains missing values. More precisely, for 2,88,602 samples the class label information is absent and additional 203 instances contained missing information. For simplicity, such instances are dropped. For easy understanding, in the later chapter (Chapter 6) this dataset is named as CICIDS dataset only.

3. HTTP Flood dataset: This dataset[8] contains 21,60,668 labeled network instances and four attack families and one normal family. Normal instances in the dataset amount to a total of 19,35,959. There are a total of 28 features in the dataset.

3.2.3 Cyber Physical Systems Datasets

For Cyber Physical Systems, three benchmark datasets are considered, each corresponding to a different critical infrastructure facility.

1. SWaT dataset [139]: This dataset belongs to a Secure Water Treatment facility. The values in this dataset are nothing but sensor readings obtained from the real life sized testbed installed at *i*Trust Center for Research in Cyber Se-

[8]`https://www.kaggle.com/datasets/jacobvs/ddos-attack-network-logs?resource=download`

curity, Singapore [9]. SWaT consists of six stages incorporating both physical and control components for treating water via ultrafiltration, de-chlorination and reverse osmosis processes. The facility ran for 11 days straight with 7 days normal operation, and attack scenarios were introduced in the next 4 days. It is a labeled dataset with a total of 4,44,496 instances and 51 features. The sensors and actuators on each of the six stages are targeted for conducting an attack based on which the attacks are categorized into four types: i) *Single Stage Single Point (SSSP) attack*, ii) *Single Stage Multi Point (SSMP) attack*, iii) *Multi Stage Single Point (MSSP) attack*, and iv) *Multi Stage Multi Point (MSMP) attack*. Although a total of 36 attacks (SSSP-36, SSMP-4, MSSP-2, and MSMP-4) are launched, the dataset contains collectively only two labels: Attack and Normal. Table 3.3 gives a brief overview of the dataset.

Table 3.3: A Brief Overview of SWaT Dataset

Symbol in dataset	Sensor/Actuator	Description	Example in dataset	Functions in
LIT	Sensor	Level Indication Transmitter	LIT-101	Stage 1
FIT	Sensor	Flow Indication Transmitter	FIT-301	Stage 3
AIT	Sensor	Analyzer Indication Transmitter	AIT-201	Stage 2
PIT	Sensor	Pressure Indication Transmitter	PIT-501	Stage 5
DPIT	Sensor	Differential Pressure Indication Transmitter	DPIT-301	Stage 3
MV	Actuator	Motorised Valve	MV-101	Stage 1
P	Actuator	Pump	P-101	Stage 1

2. Gas Pipeline and Water Storage dataset [140] [141]: These datasets were captured in a in-house SCADA laboratory facility at Mississippi State University. The three components in the testbeds are sensors and actuators, communication network and supervisory control. An instance in the datasets incorporates both network traffic information along with payload content information. The gas pipeline dataset contains a total of 2,74,627 instances with 20 features comprising both normal and attack operations. On the other hand, water storage dataset consists of 2,36,179 instances and 24 features. Both these datasets have a common set of features in addition to unique features of their own. The common set of payload features are shown in Table 3.4.

[9]`https://itrust.sutd.edu.sg/`

Table 3.4: Common Payload Features for Gas Pipeline and Water Storage Dataset

Feature Name	Description
measurement	pressure in the pipeline or water level
control_mode	Manual, automatic or shutdown
comm_fun	Value of command function code
reponse_fun	Value of response function code
pump_state	Compressor state or pump state
manual_pump_setting	manual mode compressor/pump setting

3.2.4 Other Security Datasets

In addition to the datasets discussed above in sections 3.2.1, 3.2.2 and 3.2.3, a few other security datasets are also used in this work to evaluate the proposed methods. These datasets correspond to Malware attacks, Phishing attacks and the Mirai Botnet.

3.2.4.1 Malware Datasets

Malware attacks throughout the world has seen tremendous increase in the recent years. Using different means Malicious software can be delivered to the innocent victims. Figure 3.3 gives the statistics of malware attacks over the years. As these attacks are on the rise, there is more and more need of defending against such attacks accurately and in near real time. Below are some of the benchmark malware datasets used for evaluation of the proposed defense methods.

1. TUANDROMD: It is a malware dataset proposed by Borah et al. [142] for the Android platform. TUANDROMD is a collective name for two datasets namely Android Dataset 1 and Android Dataset 2. It consists of a total of 3,250 instances (2140 instances in Android Dataset 1 and 1110 instances in Android Dataset 2) and 590 features (242 features in Android Dataset 1 and 348 features in Android Dataset 2). TUANDROMD comprises of Permission-based and API-based features. The permission-based features are the ones which require a user's confirmation to carry on with a task and can subsequently be exploited by an attacker. API-based features are the features which keeps track of each API call made by an application. For simplicity, in the later chapter (chapter 5) TUANDROMD is referred to separately by

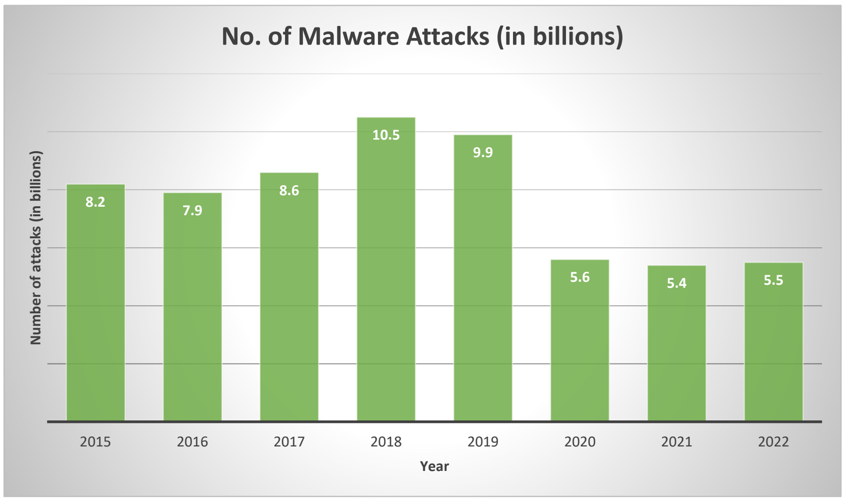


Figure 3.3: Statistics of Malware Attacks from 2015 to 2022

the names Android Dataset 1 and Android Dataset 2. Table 3.5 shows some examples of permission-based features present in the dataset.

Table 3.5: Some Examples of Permission-based Features in TUANDROMD

Feature names
SEND_SMS
RECEIVE_BOOT_COMPLETED
VIBRATE
GET_TASKS
WAKE_LOCK
KILL_BACKGROUND_PROCESSES
SYSTEM_ALERT_WINDOW
ACCESS_WIFI_STATE
DISABLE_KEY GUARD
RECEIVE_SMS
CHANGE_WIFI_STATE
WRITE_EXTERNAL_STORAGE

2. Ransomware Dataset: Sgandurra et al. [143] proposed a multiclass ransomware dataset comprising a total of 1524 samples (582 ransomware samples and 942 benign samples) from 12 classes (11 ransomware families and 1 normal). Each instance in the dataset contains 30,967 features and the categories of feature sets considered are: *Registry keys operations, API stats, Strings, File extensions, Files operations, Directory operations and Dropped file extensions*. According to the authors, several of these features from different feature categories collectively represent ransomware behavior. For example, to maintain their presence across system reboots ransomware often makes use of registry keys. Table 3.6 gives an overview of the 11 ransomware families

in the dataset.

Table 3.6: Different Ransomware Families

Ransomware Family	No. of Instances
Citroni	50
CryptLocker	107
CryptoWall	46
Kollah	25
Kovter	64
Locker	97
Matsnu	59
Pgpcoder	4
Reveton	90
TeslaCrypt	6
Trojan-Ransom	34

3.2.4.2 Phishing Dataset

The authors in [144] prepared a dataset which try to characterize if a particular website is genuine or if it is phishing website. In the literature, it is found that there no definitive features to clearly identify and detect a phishing attack. However, the authors in their proposed dataset have categorized the features into i) Address bar-based, ii) Abnormal-based, iii) HTML and JavaScript-based, and iv) Domain-based features, for effective predictions. The dataset comprises of 11,055 instances and 31 features, and a class label of either Attack or Normal for each instance.

3.2.4.3 Kitsune Network Attack Dataset

The authors in [145], present a dataset consisting of several attack types including Man in the Middle attacks, Denial of Service attacks and Reconnaissance attacks. However, of prime interest is the botnet attack carried out by the Mirai malware. For capturing the traffic an IP-based commerical surveillance system is used. The captured traffic consists of millions of network packets (raw dataset in .pcap format), and from each packet the features are extracted using AfterImage feature extractor[10]. The preprocessed dataset comprises of a total of 764,137 instances

[10]`https://github.com/ymirsky/Kitsune-py/tree/master/KitNET`

with 116 features. All the features collectively describe a snapshot of the network hosts and their behavior in the case of a network packet.

3.3 Generated Datasets

For effective evaluation of Cross-site scripting attacks, an XSS attack dataset generation framework is proposed below as discussed.

3.3.1 XSS Attack Dataset

A kind of Web application script injection attack called the Cross-site Scripting attack has been garnering attention from the early 2000s. In the very beginning, these attacks were used to steal user credentials from vulnerable Web applications. But by degrees, in conjunction with social engineering techniques, the complexity of XSS attacks has evolved tremendously. Now, such attacks can be used not only for credential theft, but also to launch more deleterious attacks like Distributed Denial of Service (DDoS), spreading of malware, defamation attacks, and session hijacking, to name a few. A detailed account of the evolution of XSS attacks over the years is discussed in [146].

3.3.1.1 Motivation

An XSS attack detection system's effectiveness and efficiency can be evaluated with the help of an unbiased XSS dataset. Unfortunately, a standard XSS feature dataset is not publicly available on the Internet. Hence, the creation of one such dataset is important and necessary for the timely, efficient and accurate detection of XSS attacks. Such a dataset will prove its effectiveness only when it is accurate, consistent, relevant and adequate in terms of its characteristics. The unavailability of such a significant dataset, even after XSS being one of the top vulnerabilities in Web applications, motivated us to design and develop an XSS attack dataset. Thus, the primary objective is to cover the gap pertaining to the unavailability of an XSS feature dataset.

3.3.2 Proposed Dataset Generation Framework

The dataset generation framework describes the process used to collect the proposed dataset called XSSD. The framework is divided into three different stages and each stage has different modules which perform a definite task. The overall steps followed in the dataset generation framework for XSSD are shown in Figure 3.4. It is deployed on a testbed setup developed under a restricted environment in the laboratory.

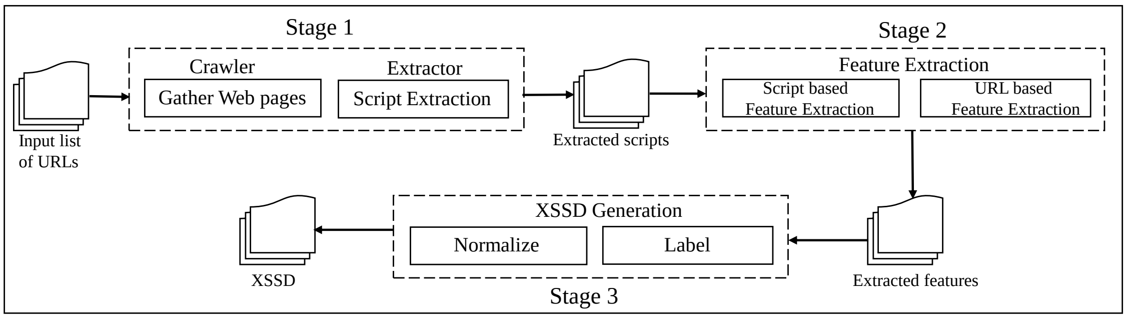


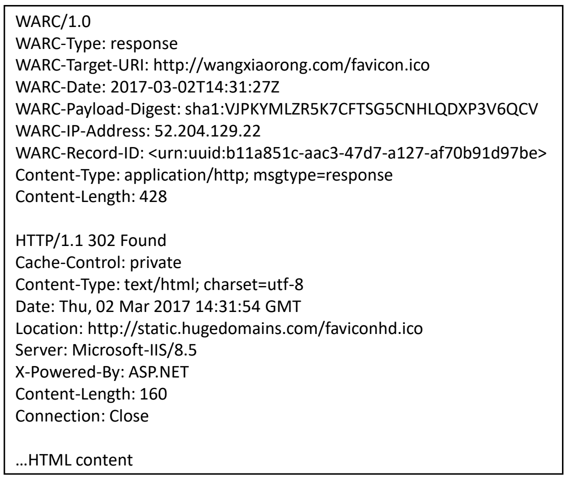
Figure 3.4: Dataset Generation Framework

3.3.3 Stages and Modules

The dataset generation framework has three stages incorporating four modules.The script extractor module and the feature extraction module are purely programming based. The functionality, type of input and other details related to each of the modules are described below. The naming of each module clearly describes its function followed by an $"\_m"$, m meaning module.

1. *Web_c_m*: Operating in Stage 1, the input to the Web crawler module is a *seeds.txt* file containing a list of URLs to gather Web pages. It is powered by the open source Web crawler, Heritrix [147], which is configured to crawl Web pages according to the URL links received as input. On visiting a Website, the crawler indexes the words and content of the Web page. Then it tries to identify any outgoing links from the site. These links are next used for the crawling process. In subsequent iterations, if any new links are added to the *seeds.txt* file, only those newly added links are crawled. After the crawling session is complete, the output obtained is a .warc (Web ARChive) file containing the raw crawled data. The content of these files can be stored in text files for further processing. The .warc files, containing raw crawled material

store three important pieces of information, the metadata of the crawling process (*WARC-Type: metadata*), how the information was requested from the website (*WARC-Type: request*) and the HTTP response from the contacted Website (*WARC-Type: response*) [148]. Figure 3.5 is an example of the WARC content obtained during the crawling session. As seen, the crawler targeted the URI `http://wangxiaorong.com/favicon.ico`. Subsequently, an HTML page is received in response, served from a *Microsoft-IIS/8.5* server.



```
WARC/1.0
WARC-Type: response
WARC-Target-URI: http://wangxiaorong.com/favicon.ico
WARC-Date: 2017-03-02T14:31:27Z
WARC-Payload-Digest: sha1:VJPKYMLZR5K7CFTSG5CNHLQDXP3V6QCV
WARC-IP-Address: 52.204.129.22
WARC-Record-ID: <urn:uuid:b11a851c-aac3-47d7-a127-af70b91d97be>
Content-Type: application/http; msgtype=response
Content-Length: 428

HTTP/1.1 302 Found
Cache-Control: private
Content-Type: text/html; charset=utf-8
Date: Thu, 02 Mar 2017 14:31:54 GMT
Location: http://static.hugedomains.com/faviconhd.ico
Server: Microsoft-IIS/8.5
X-Powered-By: ASP.NET
Content-Length: 160
Connection: Close

…HTML content
```

Figure 3.5: Example of WARC Content

2. *Script_ext_m*: The script extractor module in stage 1, is responsible for extracting only the script content from the Web archive files obtained previously. The input received is in the form of a text file, and the module comprises of C routines. With the help of such routines, one can specifically collect the content enclosed within <script>and </script>tags. The primary reason behind collecting content from only within these two tags is because, this is the only portion where any kind of JavaScript can be embedded in a Web page. The collected scripts are stored in a database, and the feature extraction algorithm is called on each of these scripts.

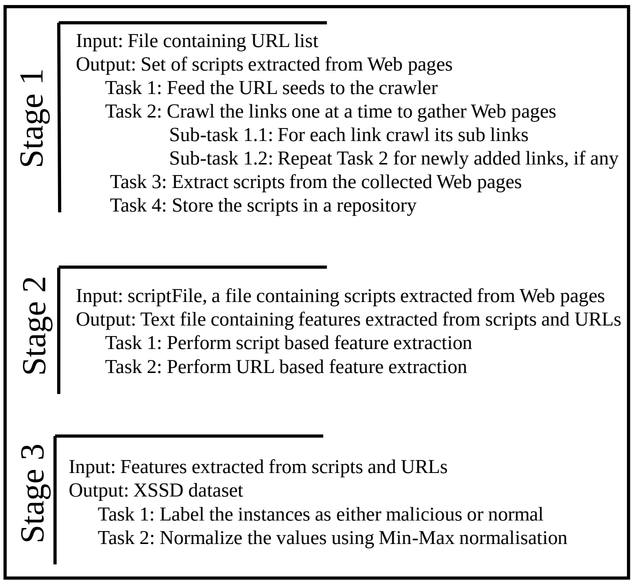
3. *Feature_extractor_m*: This module operating in Stage 2, is the heart of the data generation framework. The main goal of this module is to extract features from the scripts earlier extracted in Stage 1 and URLs collected from different sources. The scripts and URLs collected can both be malicious and legitimate. The extraction of the script-based and URL-based features are done with the help of C routines. To extract the features from each of the

scripts, the module makes use of a C routine named *extractFeature* and several other sub-routines. It receives input in the form of a text file named *scriptFile*, which consists of all the extracted scripts. The work flow of this module is specified by Algorithm 1. As shown in the algorithm, for all the scripts in the *scriptFile* file, features such as the number of characters, the number of lines, the number of methods called, are extracted. All the extracted features are then passed on to the function *dispAndWrite()*, which displays and also writes the extracted features into a .csv file. Thus, the output of this module is a .csv file containing the extracted features for each script in *scriptFile* file. A similar process is followed for extracting the URL-based features.

4. *XSSD_generator_m*: This module in Stage 3 receives as input the .csv file obtained in the previous stage. The .csv file containing the extracted features from the scripts and the URLs are normalized using *Min-Max normalization.* The script and the URL-based instances are properly and accurately labeled as either malicious or normal instances. The end output of this stage is the XSSD dataset.

3.3.4 Tasks and Sub-tasks

Each of the four modules described in the previous section, has designated tasks and sub-tasks to carry out as shown in Figure 3.6. The tasks are described below.



Stage 1

Input: File containing URL list
Output: Set of scripts extracted from Web pages
- Task 1: Feed the URL seeds to the crawler
- Task 2: Crawl the links one at a time to gather Web pages
  - Sub-task 1.1: For each link crawl its sub links
  - Sub-task 1.2: Repeat Task 2 for newly added links, if any
- Task 3: Extract scripts from the collected Web pages
- Task 4: Store the scripts in a repository

Stage 2

Input: scriptFile, a file containing scripts extracted from Web pages
Output: Text file containing features extracted from scripts and URLs
- Task 1: Perform script based feature extraction
- Task 2: Perform URL based feature extraction

Stage 3

Input: Features extracted from scripts and URLs
Output: XSSD dataset
- Task 1: Label the instances as either malicious or normal
- Task 2: Normalize the values using Min-Max normalisation

Figure 3.6: Task and Sub-tasks of the Modules

Algorithm 1: Feature Extraction Algorithm

```
Input  : scriptFile, a file containing all scripts
Output: A file with features extracted from each script
Function extractFeature(scriptFile):
    if (scriptFile) then
        foreach script S_i in scriptFile do
            countChar ← countNumberOfCharacters(S_i);
            countLine ← countNumberOfLines(S_i);
            avgCharPerLine ← countAvgCharPerLine(countChar,
              countLine);
            countPrcntgeBlankSpace ←
              countPercentageOfBlankSpaces(countChar, S_i);
            countCommLines ← countNumberOfComments(S_i);
            avgCommPerLine ← countAvgCommPerLine(countCommLines,
              countLine);
            countWords ← countNumberOfWords(S_i);
            countMethodsCalled ← countNumberOfMethodsCalled(S_i);
            countAvgArgLength ← avgArgLength(S_i);
            countUnicodeSymbol ← countNumberOfUniSymbol(S_i);
            countOctChars ← countNumberOfOctCharacters(S_i);
            countHEXChars ← countNumberOfHEXCharacters(S_i);
            countKeyWords ← countNumberOfKeyWords(S_i);
            humanReadability ← humanReadability(S_i);
            dispAndWrite(countChar, countLine, avgCharPerLine,
              countPrcntgeBlankSpace, countCommLines, avgCommPerLine,
              countWords, countMethodsCalled,
              countAvgArgLength,countUnicodeSymbol, countOctChars,
              countHEXChars, countKeyWords, humanReadability);
        end
    end
```

1. **Tasks in Stage 1**: For collecting various Web pages from the Internet, the HERITRIX Web crawler is fed with an input file *seeds.txt* consisting of a set of URLs. The crawling session yields crawled content in the form of

.warc files, from where Web pages need to be extracted. The primary task is to extract scripts from these Web pages. To accomplish this, shell script commands are written to extract the script content between <script>and </script>tags. In a broader sense, a task of pattern matching is carried out. Once it encounters a <script>tag, it writes the content, starting from the tag to another output file (named *scriptFile*) until it encounters a corresponding </script>tag. The process is continued until the end of the input file. The output of this stage is a text file called *scriptFile* containing the extracted scripts.

2. **Tasks in Stage 2**: The primary task in this stage is the task of feature extraction. The text file, *scriptFile*, containing all the extracted scripts, is fed as input to the feature extraction algorithm described earlier. A total of 14 features are extracted for each script. Similar to script-based feature extraction, URL-based feature extraction is also carried out on the collected URLs.

3. **Tasks in Stage 3**: The last stage in the framework is concerned with the tasks of normalizing and labeling the instances of the dataset as either normal or malicious. Finally, the output of the stage is the XSS attack dataset, XSSD.

3.3.5 XSSD: The Dataset and Its Characteristics

The processes and modules involved in the dataset generation framework have been discussed in detail in the previous sections. The output of the framework is the dataset called XSSD comprising of two feature sets: a script-based feature set for the extracted scripts and a URL-based feature set for the URLs collected from various Web pages. It is worth noting that legitimate URLs are collected from Alexa Top 500 websites [28] and malicious URLs from the XSSed repository [29]. Below a discussion on the two feature sets is presented in detail.

3.3.5.1 Script-based Features

Script-based features describe the characteristics of a script. These features help gather the essence of a script and play a significant role in distinguishing between a malicious and a benign script. These features are extracted from raw scripts using several C routines. The features can be specifically divided into five categories which are discussed below. A total of 14 script-based features are listed in Table 3.7. The features are taken from [59] and grouped into different categories based on their functionality.

a) *Basic features*: As the name suggests, these features describe the basic characteristics of a script. They help differentiate between a malicious and a normal script in a significant way. This is because, a malicious script usually has an abnormally large or abnormally low (example: if the script is shortened using some encoding scheme) number of characters or lines or words, compared to a normal script. The reason behind may be to make it less suspicious. Another feature worth mentioning in this category is the *%ofWhitespace* feature. This feature plays an important role because malicious scripts when compared to normal ones tend to have lower amount of white space, making it difficult for the user to read.

b) *Comment-based features*: This set of features help determine the comment-related aspects of a script. Generally, comments are used for annotation purposes by a programmer, so that a user can easily understand the operations of the script. But, a sophisticated attacker might not use any comments in his/her especially crafted scripts. His/her ultimate goal is to fool the user into doing something that is ill-advised.

c) *Keyword-based features*: These features determine the presence of any keyword in the script. It is seen that, most malicious scripts have keywords such as *xss*, *hacker*, *redirect*, *noscript*, *click*, *trigger*, *javascript*, *cross-site*, etc., in them. These keywords are searched for in a script irrespective of the case they are written in. Most keywords mentioned here do not occur in a normal script, thus, signaling a major difference between the two kinds.

d) *Function-based features*: This set contains only two features, namely the number of methods called and the average argument length. While there can be both built-in JavaScript methods and user-defined functions in a script, in this set only the user-defined ones are considered, because the presence of the built-in ones is already captured in the *Keyword-based* feature set. Attackers may craft their own functions and use them as tools to carry out malicious activities.

e) *Obfuscation-based features*: Obfuscation techniques are used solely to ensure low human readability of the text. While they can be used for legitimate purposes as well, most attackers adopt these techniques for evasion from intrusion detection systems (example: signature-based or static-based IDS). The obfuscation-based features tend to make the script difficult to read for the user. As they can be used for legitimate purpose as well, these features work best when used in conjunction with other features such as those mentioned earlier.

Table 3.7: Script-based Features and their Description

No.	Category	Feature	Feature description
1.	Basic	NoOfChars	Number of characters in the script
		NoOfLines	Number of lines in the script
		AvgCharPerLine	Average number of characters per line
		TotNoOfWords	Total number of words
		%ofWhitespace	Percentage of whitespace in the script
2.	Comment-based	NoOfComm	Number of comments in the script
		AvgCommPerLine	Average comments per line
3.	Keyword-based	NoOfkeywords	Number of keywords in the script
4.	Function-based	NoofMethods	Number of methods called
		AvgArgLength	Average argument length
5.	Obfuscation-based	NoOfOctNum	Number of octal numbers
		NoofUnicodeSym	Number of Unicode symbols
		NoofHEXSym	Number of HEX numbers
		Human Readability	Human readability in terms of 0 or 1. 0:No and 1:Yes. The checking criteria are:
			a) Percentage of words which are >70% alphabetical
			b) Percentage of words where 20%<vowels<60%
			c) Percentage of words which are less than 15 characters long

3.3.5.2 URL-based Features

URL-based features help describe the attributes of a URL, which help distinguish between benign and malicious URLs. For collecting malicious URLs, the XSSed repository and a list of blacklisted domains are used. Some malicious URLs are also gathered from the previously collected malicious scripts. Similarly, the benign URLs are collected from sites listed in the ALEXA Top 500 list. The features are grouped into various categories: basic, keyword-based and obfuscation-based. The seven URL-based features taken from [60] are listed in Table 3.8. The different categories of URL-based features are discussed below.

a) Basic features: The features under this category are the URL length, number of domain occurrences and the number of characters in the URL, which help describe the basic properties of a given URL.

b) Keyword-based: The keyword-based feature indicates the presence of any keyword in the URL which may aid in the discrimination between a normal and malicious URL.

c) Obfuscation-based: Obfuscation-based features render the URL hard to recognize. As a result, human readability is reduced.

3.3.5.3 Characteristics of XSSD

With reference to the points discussed in Section 3.1.1, the following are the characteristics of the proposed XSSD dataset. Figure 3.7 and Figure 3.8 show snapshots of the proposed XSSD dataset.

a) *Labels in the data*: The data rows in XSSD have two labels, Benign/Normal or Malicious. The benign instances constitute the profile of the Normal/Benign class, while the malicious instances constitute the profile of the Malicious/Attack class.

b) *Number of instances*: There are a total of 6695 (6220 scripts and 475 URLs) instances, of which 289 instances are malicious and the rest 6406 instances are benign. As and when needed, the number of instances in any of the classes can be increased by slight changes in the modules of the dataset generation framework.

Table 3.8: URL-based Features and their Description

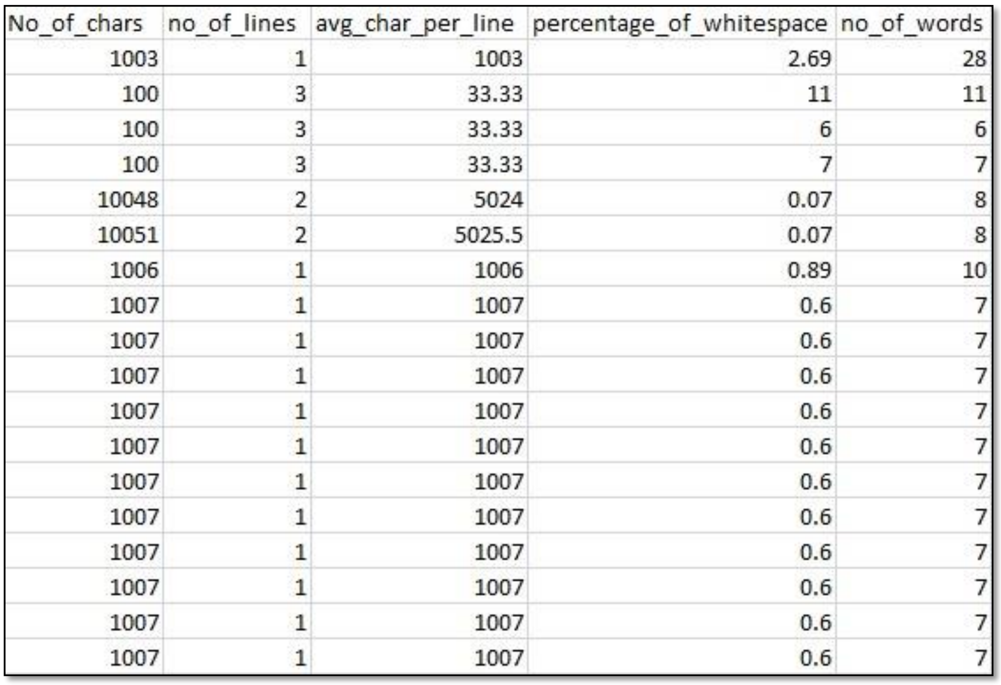
No.	Category	Feature	Feature description
1	Basic	urlLen	The length of the URL
		domOcc	Total number of Domain Occurences
		No.OfChars	Total number of characters in the URL
2	Keyword-based	keyOcc	The Keywords occurred in the URL
3	Obfuscation-based	unicodeOcc	The unicode symbols in the URL
		hexOcc	Number of HEX numbers in the URL
		Human Readability	Human readability in terms of 0 or 1. 0:No and 1:Yes. The checking criteria are:
			a) Percentage of words which are >70% alphabetical
			b) Percentage of words where 20%<vowels<60%
			c) Percentage of words which are less than 15 characters long

c) *Number of features*: A total of 21 features are extracted for XSSD, of which 14 are script-based and 7 are URL-based features. These features help differentiate between a malicious script/URL entity and a normal/benign one.

d) *Balance between the classes*: Although there is a slight imbalance in XSSD, considering the number of instances in each class, it can be done right either by sampling or collecting more instances. The modules in the proposed dataset generation framework are flexible for such adaptations.

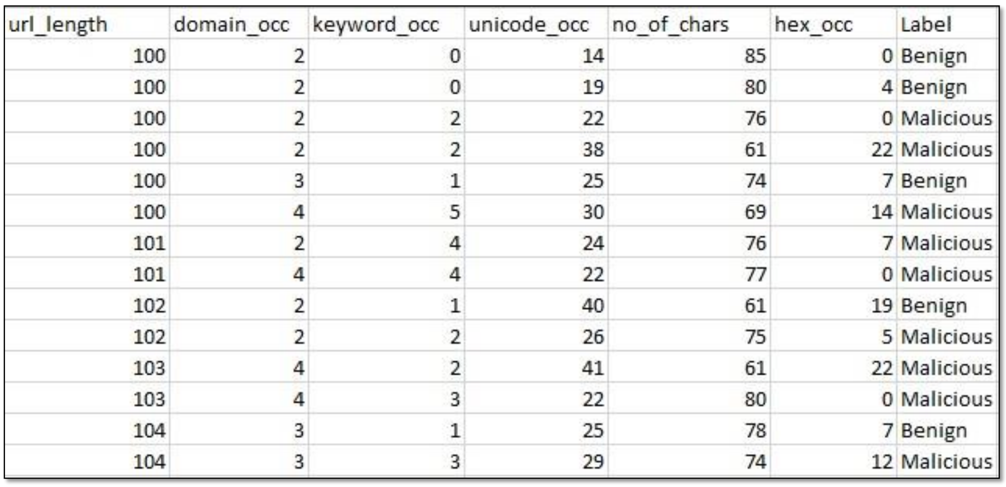
e) *Recency*: The scripts and URLs collected for populating XSSD dataset are recent (dated early 2016). The attack vectors mentioned in [26], which are updated every once in a while (the latest being 4th July 2018) are also included. As and when new attack vectors come up, the dataset can be populated accordingly.

f) *Relevance*: The extracted script-based and URL-based features collectively work to provide a clear distinction between a malicious entity and a normal entity. To the best of our knowledge, these features are relevant in differentiating between the two, and are also concise.



No_of_chars	no_of_lines	avg_char_per_line	percentage_of_whitespace	no_of_words
1003	1	1003	2.69	28
100	3	33.33	11	11
100	3	33.33	6	6
100	3	33.33	7	7
10048	2	5024	0.07	8
10051	2	5025.5	0.07	8
1006	1	1006	0.89	10
1007	1	1007	0.6	7
1007	1	1007	0.6	7
1007	1	1007	0.6	7
1007	1	1007	0.6	7
1007	1	1007	0.6	7
1007	1	1007	0.6	7
1007	1	1007	0.6	7
1007	1	1007	0.6	7
1007	1	1007	0.6	7
1007	1	1007	0.6	7
1007	1	1007	0.6	7

Figure 3.7: A Snapshot of XSSD (Script-based Features)



url_length	domain_occ	keyword_occ	unicode_occ	no_of_chars	hex_occ	Label
100	2	0	14	85	0	Benign
100	2	0	19	80	4	Benign
100	2	2	22	76	0	Malicious
100	2	2	38	61	22	Malicious
100	3	1	25	74	7	Benign
100	4	5	30	69	14	Malicious
101	2	4	24	76	7	Malicious
101	4	4	22	77	0	Malicious
102	2	1	40	61	19	Benign
102	2	2	26	75	5	Malicious
103	4	2	41	61	22	Malicious
103	4	3	22	80	0	Malicious
104	3	1	25	78	7	Benign
104	3	3	29	74	12	Malicious

Figure 3.8: A Snapshot of XSSD (URL-based Features)

3.3.6 Performance Evaluation and Validation

The performance of XSSD can be evaluated with the help of learning algorithms. The reason behind using learning algorithms for evaluation is to assess the effectiveness of XSSD so that it can be used to develop XSS attack detection techniques based on machine learning models. After going through a preliminary preprocessing step, the dataset is fed to a total of five classifiers as shown in Figure 3.9 and 3.10. From the obtained results, it is seen that the Extra Trees and XGBoost classifiers perform better than the others in terms of accuracy.

To evaluate the goodness of classification, the ROC curve is plotted. Figure 3.11 and 3.12 illustrates the ROC curve for the script and URL datasets, respectively. Both the figures depict the TPR and FPRs of different classifiers at different decision threshold values. The classifiers perform well on URL based dataset as compared to the script based dataset. This may be due to the problem of class imbalance which can be handled with sampling techniques or addition of new data instances to the minority class. The corresponding AUC values are reported in Table 3.9.

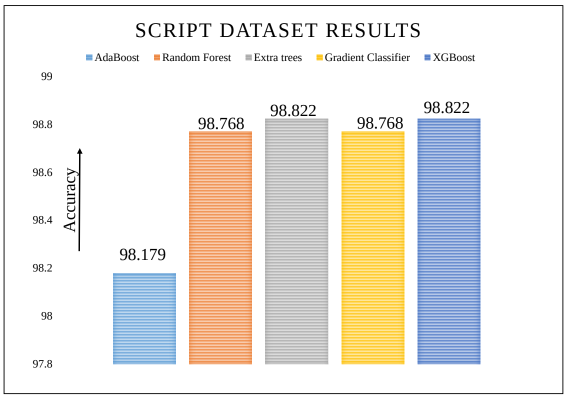


Figure 3.9: Performance Evaluation for the Script Dataset

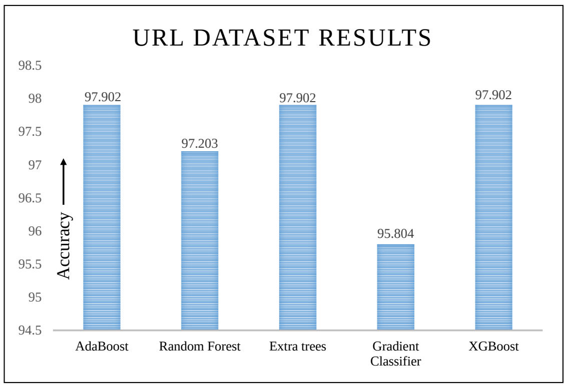


Figure 3.10: Performance Evaluation for the URL Dataset

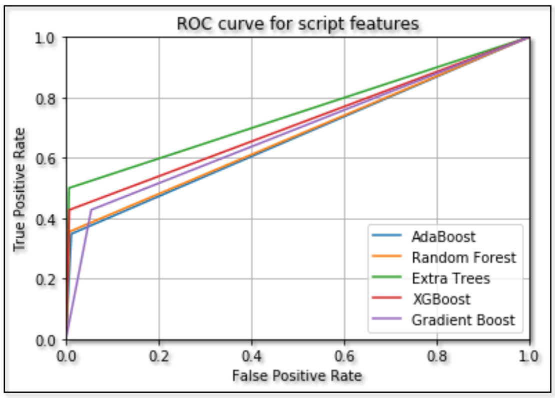


Figure 3.11: ROC Curve for the Script Dataset

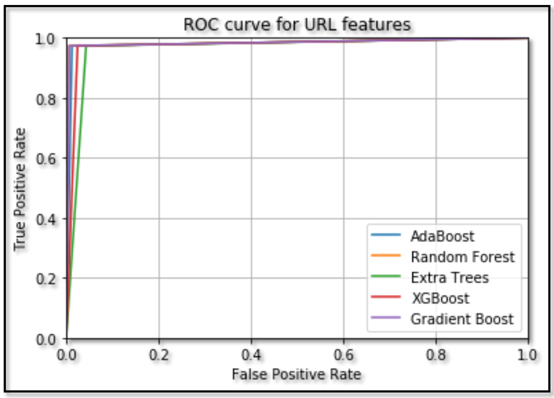


Figure 3.12: ROC Curve for the URL Dataset

3.4 Discussion

Effectiveness of detection methods can be established only with the help of appropriate datasets. In this chapter, several security datasets corresponding to different

Table 3.9: AUC Values for all Five Classifiers

Classifier	Script dataset	URL dataset
AdaBoost	0.668	0.98
Random Forest	0.695	0.968
Extra trees	0.771	0.977
XGBoost	0.71	0.974
Gradient Boost	0.683	0.983

attacks are described. Table 3.10 gives a summary of the datasets used in the proposed methods discussed later on.

Table 3.10: Summary of All the Datasets Used

	Dataset for	Dataset name	No. of samples	No. of features	No. of classes	Year of release
	Cross-site Scripting Attacks	XSSD	6695	14	2	2018
	HTTP Flooding Attacks	UNSW-NB15	25,40,044	49	2	2015
		CICIDS2017	28,30,540	80	2	2017
Security Dataset		HTTP-Flood	21,60,668	28	2	
	Cyber Physical System	SWaT	4,44,496	51	2	2016
		Gas Pipeline	2,74,627	20	2	2015
		Water Storage	2,36,179	24	2	2015
	Malware Attacks	TUANDROMD	3,250	590	2	2020
		Ransomware dataset	1524	30,967	12	2016
	Phishing	Phishing Websites	11,055	31	2	2015
	IoT-Botnet	Kitsune Network Attack	7,64,137	116	2	2018

The datasets discussed in this chapter are used for evaluation purposes in the subsequent chapters. XSSD discussed in Section 3.3.1 is used in Chapter 5 for evaluating MICC-UD, the proposed detection method for evaluating XSS attacks in the application layer. MICC-UD is also evaluated with other datasets like the Malware datasets (Section 3.2.4.1), Phishing dataset (Section 3.2.4.2) and the Kitsune Network Attack Dataset (Section 3.2.4.3). HTTP Flooding Datasets discussed in Section 3.2.2 are used in Chapter 6 for evaluating an incremental detection mechanism named INFS-MICC for detecting HTTP Flooding Attacks. Lastly, Cyber Physical System Datasets discussed in Section 3.2.3 is used in Chapter 7 for evaluating FSRA, an ensemble detection method for detecting attacks in the Critical Infrastructure.