IJARCCE

ISSN (Online) 2278-1021
ISSN (Print) 2319 5940

International Journal of Advanced Research in Computer and Communication Engineering
ISO 3297:2007 Certified
Vol. 5, Issue 10, October 2016

Detecting and Identifying Web Application Vulnerabilities with Static and Dynamic Code Analysis using Knowledge Mining

Vineetha K R[1], **N. Santhana Krishna**[2]

MPhil Scholar, AJK College of Arts and Science, Bharathiar University, Coimbatore

Asst. Professor& HOD of CS, AJK College of Arts and Science, Bharathiar University, Coimbatore

Abstract: Software developers area unit typically tasked to figure with foreign code that they are doing not perceive. A developer may not recognize what the code is meant to try and do, however it works, however it's purported to work, a way to fix it to create it work, however documentation could be poor, outdated, or non-existent. Moreover, knowledgeable co-workers aren't invariably like a shot offered to assist. we have a tendency to believe that if the developer is given with “similar” code – code that's legendary to execute within the same method or reckon constant perform, the developer may 1) realize the similar code additional readable; 2) determine acquainted code to transfer her existing information to the code at hand; 3) gain insights from a distinct structure that implements constant behaviour. Various researchers have investigated code that exhibits static – matter, syntactical or structural – similarity. That is, code that appears alike. Visual variations may vary from whitespace, layout and comments to identifiers, literals and kinds to modified, additional and removed statements (sometimesreferred to as kind one to kind three “code clones”, resp). Kind four could be a catch-all for all semantically similar clones, but it lacks scientific formulations to classify them. Static detection techniques plan to match the ASCII text file text or some static mental representation corresponding to tokens, abstract syntax trees, program dependence graphs, or a mix of multiple program representations found static similarity at the assembly code level. However static techniques cannot invariably notice code that behaves alike, i.e., that exhibits similar dynamic (runtime) behavior, however doesn't look alike. Once the developer doesn't perceive the code she must work with, showing her additional code that appears regarding constant might not be useful. Butexplaining that bound alternative code that appears quite completely different really behaves terribly equally might offer clues to what her own code will and the way it works. so tools that notice and manage statically similar code might miss opportunities to assist developers perceive execution behavior.

Keywords: Code Vulnerability, Code Similarity, Static and Dynamic Webpage, Code Clone, Developer Behaviour, Runtime, Identifiers.

I. INTRODUCTION

The W WW distributions are generating a considerable boost in the order of web sites and web applications. A codeVulnerability is a code portion in source files that is matching or similar to another. It is general view that code clones make the source files very hard to modify constantly.
Vulnerability are launched for various reasons such as lack of a good design, fuzzy requirements, disorderly protection and evolution, lack of suitable reuse mechanisms, and reusing code by copy-and-paste.

Thus, code clone detection can effectively support the improvement of the quality of a software system during software preservation and growth. The very short time-to-scope of a web application, and the need of method for developing it, support an increase al expansion fashion where new pages are usually obtained reusing (i.e. “Vulnerability”) pieces of existing pages without sufficient documentation about these code duplications and redundancies.

The presence of clones increase system difficulty and the effort to test, maintain and change web systems, thus the identification of clones may reduce the effort devoted to these activities as well as to facilitate the migration to different architectures.

Knowledge Mining is that the creation of information from structured (relational databases, XML) and unstructured (text, documents, images) sources. The ensuing data must be in a very machine-readable and machine-interpretable format and should represent data in a very manner that facilitates inferencing.

Though it's methodically the same as data extraction (NLP) and ETL (data warehouse), the most criteria is that the extraction result goes on the far side the creation of structured data or the transformation into a relative schema. It needs either the employ of existing formal data (reusing identifiers or ontologies) or the generation of a schema supported the supply knowledge.

Knowledge management (KM) is that the method of capturing, developing, sharing, and effectively victimisation structure data.It refers to a multi-disciplinary approach to achieving structure objectives by creating the most effective use of information. An established discipline since 1991, kilometer includes courses instructed within the fields of business administration, data systems, management, library, and knowledge sciences. alternative fields could contribute to kilometre analysis, together with data and media, engineering science, public health, and public policy.many Universities supply dedicated Master of Science degrees in data Management.
Many giant firms, public establishments, and non-profit organisations have resources dedicated to internal kilometre efforts, usually as an area of their business strategy, data technology, or human resource management departments. many consulting firms give recommendation concerning kilometre to those organisations. Knowledge management efforts generally target organisational objectives adore improved performance, competitive advantage, innovation, the sharing of lessons learned, integration, and continuous improvement of the organisation. These efforts overlap with organisational learning and should be distinguished from that by a larger target the management of information as a strategic quality and attention on encouraging the sharing of information. Kilometre is AN enabler of organisational learning.

Review of literature

A code clone is a code portion in source files that is identical or similar to another. It is common opinion that code clones make the source files very hard to modify consistently. Clones are introduced for various reasons such as lack of a good design, fuzzy requirements, undisciplined maintenance and evolution, lack of suitable reuse mechanisms, and reusing code by copy-and-paste.[1]. Maintaining software systems is getting more complex and difficult task, as the scale becomes larger. It is generally said that code clone is one of the factors that make software maintenance difficult. This project also develops a maintenance support environment, which visualizes the code clone information and also overcomes the limitation of existing tools [2].One limitation of the current research on code clones is that it is mostly focused on the fragments of duplicated code (we call them simple clones), and not looking at the big picture where these fragments of duplicated code are possibly part of a bigger replicated program structure. We call these larger granularity similarities structural clones [3].Reuse only what is similar, knowing clones helps in reengineering of legacy systems for reuse. Detection of large-granularity structural clones becomes particularly useful in the reuse context . While the knowledge of structural clones is usually evident at the time of their creation, we lack formal means to make the presence of structural clones visible in software, other than using external documentation or naming conventions[4]. Generally speaking, templates, as a common model for all pages, occur quite fixed as opposed to data values which vary across pages. Finding such a common template requires multiple pages or a single page containing multiple records as input. When multiple pages are given, the extraction target aims at page-wide information [5] .One limitation of the current research on code clones is that it is mostly focused on the fragments of duplicated code (we call them simple clones), and not looking at the big picture where these fragments of duplicated code are possibly part of a bigger replicated program structure. We call these larger granularity similarities structural clones [6].

Existing System

Refactoring is a transformation that preserves the external behavior of a program and improves its internal quality. Usually, compilation errors and behavioral changes are avoided by preconditions determined for each refactoring transformation. However, to formally define these preconditions and transfer them to program checks is a rather complex task. In practice, refactoring engine developers commonly implement Vulnerability in an ad hoc manner since no guidelines are available for evaluating the correctness of refactoring implementations. As a result, even mainstream refactoring engines contain critical bugs. We present a technique to test Java refactoring engines. It automates test input generation by using a Java program generator that exhaustively generates programs for a given scope of Java declarations. The refactoring under test is applied to each generated program. The technique uses SafeRefactor, a tool for detecting behavioral changes, as an oracle to evaluate the correctness of these transformations. Finally, the technique classifies the failing transformations by the kind of behavioral change or compilation error introduced by them. We have evaluated this technique by testing 29 Vulnerability in Eclipse JDT, NetBeans, and the JastAdd Refactoring Tools. We analyzed 153,444 transformations, and identified 57 bugs related to compilation errors, and 63 bugs related to behavioral changes.

Problem Identified

Although an oversized endeavor on internet application security has been occurring for quite adecade, the safety of internet applications continues to be a difficult downside. a vital part of thatdownside derives from vulnerable ASCII text file, often written in unsafe languages like PHP.ASCII text file static analysis tools square measure an answer to search out vulnerabilities,however they have an inclination to come up with false positives, and need right smart effort forprogrammers to manually fix the code. we tend to explore the utilization of a mixture of ways tofind vulnerabilities in ASCII text file with fewer false positives. we tend to mix taint analysis,that finds candidate vulnerabilities, with data processing, to predict the existence of false positives. This approach brings along 2 approaches that square measure apparently orthogonal:humans secret writing the data regarding vulnerabilities (for taint analysis), joined with the onthe face of it orthogonal approach of mechanically getting that data (with machine learning, forinformation mining).

An approach for mechanically protectiveweb applications whereas keeping the computer programmer within the loop. The approach consists in analyzing the net application ASCII text filesearching for input validation vulnerabilities, and inserting fixes in the same code to correct these flaws. The computer programmer is unbroken in the loop by being allowed to grasp wherever the vulnerabilitieswere found, and the way they were corrected

Proposed Methodology
A code clone is a code portion in source files that is identical or similar to another. It is common opinion that code clones make the source files very hard to modify consistently. Clones are introduced for various reasons such as lack of a good design, fuzzy requirements, undisciplined maintenance and evolution, lack of suitable reuse mechanisms, and reusing code by copy-and-paste. Thus, code clone detection can effectively support the improvement of the quality of a software system during software maintenance and evolution.
The Internet and World Wide Web diffusion are producing a substantial increase in the demand of web sites and web applications. The very short time-to-market of a web application, and the lack of method for developing it, promote an incremental development fashion where new pages are usually obtained reusing (i.e. “cloning”) pieces of existing pages without adequate documentation about these code duplications and redundancies. The presence of clones increase system complexity and the effort to test, maintain and evolve web systems, thus the identification of clones may reduce the effort devoted to these activities as well as to facilitate the migration to different architectures.This project proposes an approach for detecting clones in web sites and web applications, obtained tailoring the existing methods to detect clones in traditional software systems. The approach has been assessed performing analysis on several web sites and web applications.
Maintaining software systems is getting more complex and difficult task, as the scale becomes larger. It is generally said that code clone is one of the factors that make software maintenance difficult. This project also develops a maintenance support environment, which visualizes the code clone information and also overcomes the limitation of existing tools.

Algorithm
```
intLevenshteinDistance(char s[1..m], char t[1..n])
{
// for all i and j, d[i,j] will hold the Levenshtein distance between
// the first i characters of s and the first j characters of t;
// note that d has (m+1)x(n+1) values
declareint d[0..m, 0..n]

clear all elements in d // set each element to zero
// source prefixes can be transformed into empty string by
// dropping all characters
for i from 1 to m
{
d[i, 0] := i
}
// target prefixes can be reached from empty source prefix
// by inserting every characters
for j from 1 to n
{
d[0, j] := j
}

for j from 1 to n
{
for i from 1 to m
{
if s[i] = t[j] then
d[i, j] := d[i-1, j-1]       // no operation required
else
d[i, j] := minimum
                (
d[i-1, j] + 1,  // a deletion
d[i, j-1] + 1,  // an insertion
d[i-1, j-1] + 1 // a substitution
                )
}
}

return d[m,n]
}
```

Computing the Levenshtein distance is based on the observation that if we reserve a matrix to hold the Levenshtein distances between all prefixes of the first string and all prefixes of the second, then we can compute the values in the matrix in a dynamic programming fashion, and thus find the distance between the two full strings as the last value computed.

Levenshtein distance Algorithm
In information theory and computer science, the Levenshtein distance is a string metric for measuring the amount of difference between two sequences. The term edit distance is often used to refer specifically to Levenshtein distance.

In approximate string matching, the objective is to find matches for short strings, for instance, strings from a dictionary, in many longer texts, in situations where a small number of differences is to be expected. Here, one of the strings is typically short, while the other is arbitrarily long. This has a wide range of applications, for instance, spell checkers, correction systems for optical character recognition, and software to assist natural language translation based on translation memory.
The Levenshtein distance can also be computed between two longer strings, but the cost to compute it, which is roughly proportional to the product of the two string lengths, makes this impractical. Thus, when used to aid in fuzzy string searching in applications such as record

IJARCCE
ISSN (Online) 2278-1021
ISSN (Print) 2319 5940
International Journal of Advanced Research in Computer and Communication Engineering
ISO 3297:2007 Certified
Vol. 5, Issue 10, October 2016

linkage, the compared strings are usually short to help improve speed of comparisons.
Levenshtein distance is not the only popular notion of edit distance. Variations can be obtained by changing the set of allowable edit operations: for instance,

- length of the longest common subsequence is the metric obtained by allowing only addition and deletion, not substitution;
- the Damerau–Levenshtein distance allows addition, deletion, substitution, and the transposition of two adjacent characters;
- theHamming distance only allows substitution (and hence, only applies to strings of the same length).

Edit distance in general is usually defined as a parametrizable metric in which a repertoire of edit operations is available, and each operation is assigned a cost (possibly infinite).

1.**Web URL Identification**

In computing, a Uniform Resource Locator (URL) is a type of Uniform Resource Identifier (URI) that specifies where an identified resource is available and the mechanism for retrieving it. In popular usage and in many technical documents and verbal discussions it is often, imprecisely and confusingly, used as a synonym for uniform resource identifier. The confusion in usage stems from historically different interpretations of the semantics of the terms involved. In popular language a URL is also referred to as a Web address.

2. Information Extraction and Parsing

The HTML Parsing module is a class for accessing HTML as tokens. An HTML Parsing object gives you one token at a time, much as a file handle gives you one line at a time from a file. The HTML can be tokenized from a file or string. The tokenizer decodes entities in attributes, but not entities in text. A program that extracts information by working with a stream of tokens doesn't have to worry about the peculiarity of entity encoding, whitespace, quotes, and trying to work out where a tag ends.

Regular expressions are powerful, but they're a painfully low-level way of dealing with HTML. The system processes the spaces and new lines, single and doubles quotes, HTML comments, and a lot more. The next step up from a regular expression is an HTML tokenize. In this chapter, we'll use HTML Parser to extract information from HTML files. Using these techniques, you can extract information from any HTML file, and never again have to worry about character-level trivia of HTML markup. And automatic passage extraction methods from the body may be worthwhile. Implications of the findings for aids to summarization, and specifically the Text

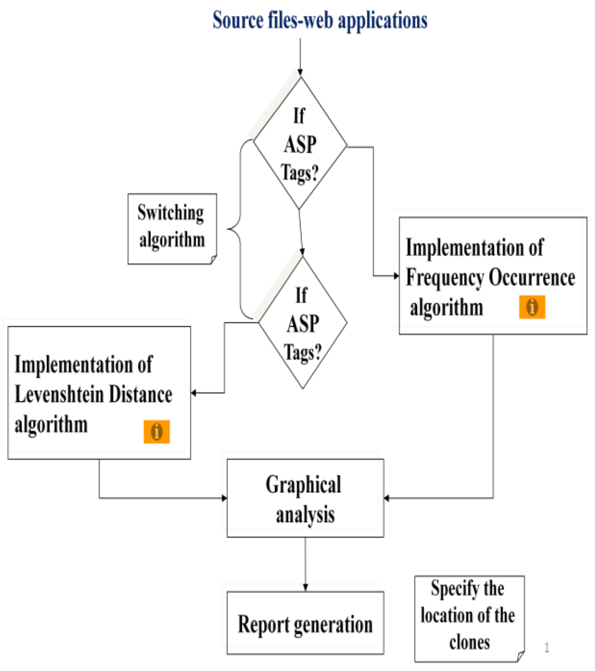
3. **Template Finding & Training**

Levinstein Distance also finds some large trustworthy information such as A1 Books, which provides 86 of 100 books with an accuracy of 0.878 in detecting web vulnerability. Please notice thatLevinstein Distance uses no training data, and the testing data is manually created by reading the authors' names from book covers. Therefore, we believe the performs iterative computation to find out the set of authors for each web page. In order to test its accuracy, we randomly select 100 web page and manually find out their vulnerability.

4.**VulnerabilityDetection and Mining**

Finally, we perform an interesting experiment on finding trustworthy websites. It is well known that Google (or other search engines) is good at finding authoritative websites. However, do these websites provide accurate information? To answer this question, we compare the online bookstores that are given highest ranks by Google with the bookstores with highest trustworthiness found by Levenshtein distance uses iterative methods to compute the website trustworthiness and fact confidence, which is widely used in many link analysis approaches, . The common feature of these approaches is that they start from some initial state that is either random or uninformative. Then, at each iteration, the approach will improve the current state by propagating information (weights, probability, trustworthiness, etc.) through the links.

ArchitectureDiagram


Source files-web applications
If ASP Tags?
Switching algorithm
Implementation of Frequency Occurrence algorithm
If ASP Tags?
Implementation of Levenshtein Distance algorithm
Graphical analysis
Report generation
Specify the location of the clones

Experimental Results

CODE clones are similar program structures of considerable size and significant similarity. Several studies suggest that as much as 20-50 percent of large software systems consist of cloned code . Knowing the location of clones helps in program understanding and maintenance. Some clones can be removed with refactoring , by replacing them with function calls or macros, or we can use unconventional metalevel techniques such as Aspect-

IJARCCE
ISSN (Online) 2278-1021
ISSN (Print) 2319 5940
International Journal of Advanced Research in Computer and Communication Engineering
ISO 3297:2007 Certified
Vol. 5, Issue 10, October 2016

Oriented Programming or XVCL to avoid the harmful effects of clones. Cloning is an active area of research, with a multitude of clone detection techniques been proposed in the literature. One limitation of the current research on code clones is that it is mostly focused on the fragments of duplicated code (we call them simple clones), and not looking at the big picture where these fragments of duplicated code are possibly part of a bigger replicated program structure.We call these larger granularity similarities structural clones. Locating structural clones can help us see the forest from the trees, and have significant value for program understanding, evolution, reuse, and reengineering.

Clone Vulnerability detection

The limitation of considering only simple clones is known in the field. The main problem is the huge number of simple clones typically reported by clone detection tools. There have been a number of attempts to move beyond the raw data of simple clones. It has been proposed to apply classification, filtering, visualization, and navigation to help the user make sense of the cloning information. Another way is to detect clones of larger granularity than code fragments. For example, some clone detectors can detect cloned files , while others target detecting purely conceptual similarities using information retrieval methods rather than detecting simple clones

Document Testing andclustering

Document clustering (also referred to as Text clustering) is closely related to the concept of data clustering. Document clustering is a more specific technique for unsupervised document organization, automatic topic extraction and fast information retrieval or filtering.

A web search engine often returns thousands of pages in response to a broad query, making it difficult for users to browse or to identify relevant information. Clustering methods can be used to automatically group the retrieved documents into a list of meaningful categories, as is achieved by Enterprise Search engines such as Northern Light and Vivisimo or open source software such as Carrot2.

Evaluation tree merging

According to our page generation model, data instances of the same type have the same path from the root in the DOM trees of the input pages. Thus, our algorithm does not need to merge similar subtrees from different levels and the task to merge multiple trees can be broken down from a tree level to a string level. Starting from root nodes <html> of all input DOM trees, which belong to some type constructor we want to discover, our algorithm applies a new multiple string alignment algorithm to their first-level child nodes. There are at least two advantages in this design. First, as the number of child nodes under a parent node is much smaller than the number of nodes in the whole DOM tree or the number of HTML tags in a Webpage, thus, the effort for multiple string alignment here is less than that of two complete page alignments in RoadRunner . Second, nodes with the same tag name (but with different functions) can be better differentiated by the subtrees they represent, which is an important feature not used in EXALG. Instead, our algorithm will recognize such nodes as peer nodes and denote the same symbol for those child nodes to facilitate the following string alignment. After the string alignment step, we conduct pattern mining on the aligned string S to discover all possible repeats (set type data) from length 1 to length jSj=2. After removing extra occurrences of the discovered pattern (as that in DeLa), we can then decide whether data are an option or not based on their occurrence vector, an idea similar to that in EXALG . The four steps, peer node recognition, string alignment, pattern mining, and optional node detection, involve typical ideas that are used in current research on Web data extraction . However, they are redesigned or applied in a different sequence and scenario to solve key issues in page-level data extraction.

Testing and Evaluation of page level

Page classification has been addressed with different objectives and methods. Most work concerned form classification methods that are aimed at selecting an appropriate reading method for each form to be processed. Other approaches address the problem of grouping together similar documents in business environments, for instance separating business letters from technical papers In the last few years the classification of pages in journals and books received more attention. An important aspect of page classification are the features that are extracted from the page and used as input to the classifier. Sub-symbolic features, like the density of black pixels in a region, are computed directly from the image. Symbolic features, for instance the number of horizontal lines, are extracted from a segmentation of the image. Structural features (e.g. relationships between objects in the page) can be computed from a hierarchical description of the document. Textual features, for instance presence of some keywords, are obtained from the text in the image recognized by an Optical Character Recognition program

Testing and Evaluation of record level

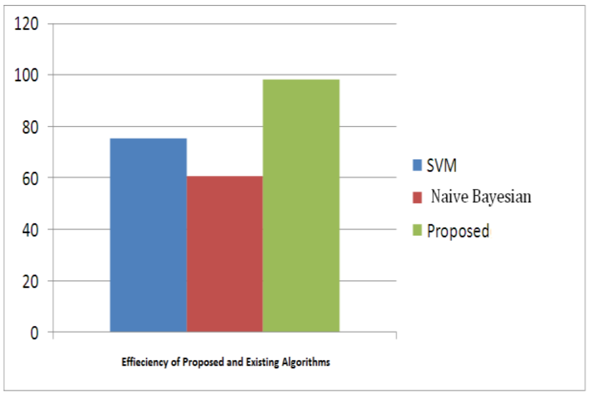
The automatically generated wrapper described in Zhao et al. for instance extracts search result records (SRRs) based on the HTML tag structures. They use learned tag paths pointing to the first (tag) line of candidate records. Their main assumption is that SRRs are usually located in the same sub-tree and its tag path follows certain patterns. Next, they identify the data records in a sub-tree based on learned separators tags and finally generate a regular expression consisting of a path and multiple separators. The regular expression is evaluated on the tag level which requires that the page must be parsed. To compensate path and separator variations, a wrapper is learned from multiple result pages. Compared to our approach the wrapper extracts the data records as a whole (record level) and does not focus on the attributes contained in the data records (attribute level). Additionally, their approach requires a parser, which also corrects the source to apply

the path expressions learned from the parse tree. Path variations can only be handled by learning a wrapper from multiple result pages, i.e. five result pages and one non-result page are needed in the wrapper generation phase. In contrast, ViPER needs only one result
page to generate a wrapper. In Crescenzi et al ,union-free regular expressions are deduced, which cannot capture the full diversity of structures presented in HTML. Wang and Lochovsky [2003] propose a system which applies a deduced regular expression for extracting data records from documents. Here labeling can only be carried out after all records have been expensively aligned, i.e. streaming based processing is impossible. All systems mentioned do not support streaming based web content extraction
The last experiment compares FiVaTech with the two visual-based data extraction systems, ViPER and MSE. The first one (ViPER) is concerning with extracting SRRs from a single (major) data section, while the second one is a multiple section extraction system. We use the 51 Websites of the Testbed referred above to compare FiVaTech with ViPER, and the 38 multiple sections Websites used in MSE to compare our system with MSE. Actually, extracting of SRRs from Webpages that have one or more data sections is a similar task. The results in Table 3 show that all of the current data extraction systems perform well in detecting data record boundaries inside one or more data sections of a Webpage. The closeness of the results between FiVaTech and the two visual-based Web data extraction systems ViPER and MSE gives an indication that until this moment visual informations do not provide the required improvement that researchers expect. This also appeared in the experimental results of ViNTs ; the visual-based Web data extraction with and without utilizing visual features. FiVaTech fails to extract SRRs when the peer node recognition algorithm incorrectly measures the similarities among SRRs due to the very different structure among them. Practically, this occurred very infrequently in the entire test page (e.g., site numbered 27 in the Testbed). Therefore, now, we can claim that SRRs extraction is not a key challenge for the problem of Web data extraction.

Comparative study with graph

Our experiment assumes the following situation: A developer starts building an application and uses classes from a library l that are unknown to her. To help the developer avoid bugs due to incorrect usage of those classes, her IDE supports lightweight type state verification. Whenever the developer changes a method that uses classes of l for which a specification is available, the IDE launches the type state verifier. The verifier then analyzes all changed methods and looks for incorrect usage of classes; if it finds a violation, it is presented to the user. Obviously, we would like to catch as many defects and report as few as possible.
Among the first approaches that specifically mine models for classes is the work by Whaley et al. Theirtechnique mines models with anonymous states and slices models by grouping methods that access the same fields. mine so-called extended finite state machines with anonymous states. To compress models, the algorithm merges states that have the same k-future. In terms of static techniques, there is also a huge number of different approaches.

	QuickAuction	Proposed Method
No. of existing function clones	40	47
No. of candidate function clones	40	58
No. of discovered function clones	36	45
No. of false negatives	4	2
No. of false positives	4	13
Recall	90%	96%
Precision	90%	96%



Accuracy Values

SVM	Naive Bayesian	Proposed
75.2	60.73	96.26
72.3	58.2	95.94

mine object usage models that describe the usage of an object in a program. They apply concept analysisto find code locations where rules derived from usage models are violated. use an inter-procedural path-sensitive analysis to infer preconditions for method invocations. Discover that static mining of automata based specifications requires precise aliasing information to produce reliable results. In the area of web services. mine behavior protocols that describe the usage of a web service. The approach uses a sequence of synthesis and testing stages that uses heuristics to refine an initially mined automaton. In contrast, mines type state automata for web application programs

Identifying the Source of Clones

To find which of the following file categories contributed most clones in web Applications:

i. **Static files –** files that needs to be delivered 'as is' to the browser. Includes markup files, style sheets and client side scripts (e.g., HTML, CSS, XSL, and JavaScript).

ii. **Server pages –** files containing embedded server side scripting. These generate dynamic content at runtime (e.g., JSP, PHP, ASP, and ASP.NET).

iii. **Templates –** files related to additional templating mechanisms used.

The HTML files are first parsed, the HTML tags are extracted and the composite tags are substituted with their equivalent ones;

The resulting HTML-strings are composed by symbols from the A^2 alphabet;

These strings are processed in order to eliminate the symbols not belonging to the A* alphabet;

The final HTML-strings are submitted to the computation of the Extended Co-Citation distance:

The Distance matrix obtained includes the distance between each couple of analyzed ASP-strings.

Example

With the following HTML alphabet table:

/div	/td	align	div	height	img	src	td	width
a	b	c	d	e	f	g	h	i

Consider the following two HTML code lines:

**<td width="18%">
<imgsrc="../images/Nuovo.jpg" width="92" height="27"></td>**

Where in the first row a reduced HTML tags alphabet is reported, while the second row reports the symbols corresponding to each tag. By analysizing the two HTML code lines ,using the table, we can identify the following sequence of HTML tags:

(td, width, img, src, width, height, /td)

And the corresponding strings of symbols:

HTML-string u = hifgieb

Consider the following two HTML code lines:

**<td width="35%"><div align="right">
<imgsrc ="../pic1.jpg" width="92" height="27"></div></td>**

With reference to same alphabet-table used in the previous example we can identify the following sequence of HTML tags:

(td, width,div,align, img, src, width, height,/div, /td) And the corresponding strings of symbols:

HTML-string v = hidcfgieab

The optimal alignment of u and v is:

u	h	i	d	c	f	g	i	e	a	b
v	h	i			f	g	i	e		b

The Extended Co-Citation distance D(u, v) = 3.They are considered as duplicated pages (similar pages) if their distance is small.

It is a Levinstein Edit Distance algorithm that extends the traditional Levinstein Edit Distance concepts. The Levinstein Edit Distance analysis has been used to measure the similarity of papers, journals, or authors for clustering. For a pair of documents p and q, if they are both cited by a common document, documents p and q is said to be cocited.

Then number of documents that cite both p and q is referred to as the Levinstein Edit Distance degree of documents p and q. The similarity between two documents is measured by their Levinstein Edit Distance degree. This type of analysis has been shown to be effective in a broad range of disciplines, ranging from author Levinstein Edit Distance analysis of scientific sub fields to journal Levinstein Edit Distance analysis. In the context of the web, the hyperlinks are regarded as citations between the pages. If a web page p has a hyperlink to another page q, page q is said to be cited by page p. In this sense, citation and Levinstein Edit Distance analyses are smoothly extended to the web page hyperlink analysis.

The extended Levinstein Edit Distance algorithm is presented with a new page source. It is constructed as a directed graph with edges indicating hyperlinks and nodes representing the following pages.

- page u
- Up to B parent pages of u and up to BF child pages of each parent page that are different from u
- Up to F child pages of u and up to FB parent pages of each child page that are different from u

The parameters B, BF, and FB are used to keep the page source to a reasonable size. Before giving the Extended Levinstein Edit Distance algorithm for finding relevant pages, the following concepts are defined

Definition 1:

Two pages P1 and P2 are back cocited if they have a common parent page. The number of their common parents is their back Levinstein Edit Distance degree, denoted as b(P1,P2). Two pages P1 and P2 are forward cocited if they have a common child page. The number of their common children is their forward Levinstein Edit Distance degree, denoted as

f (P1,P2).

Definition 2:

The pages are intrinsic pages if they have same page domain name.

Definition 3:
Two pages are near-duplicate pages if
i. they each have more than 10 links
ii. they have atleast 95 percent of their links in common

Based on the above concepts, the complete Extended Levinstein Edit Distance algorithm to find relevant pages of the given web page u is as follows:

Step 1: Choose up to B arbitrary parents of u.
Step 2: For each of these parents p, choose up to BF children (different from u) of p that surround the link from p to u. Merge the intrinsic or near-duplicate parent pages, if they exist, as one whose links are the union of the links from the merged intrinsic or near-duplicate parent pages, i.e., let Pu be a set of parent pages of u,

$$P_u = \{p_i \mid p_i \text{ is a parent page of } u \text{ without intrinsic and near-duplicate pages, } i \in [1, B]\},$$

let

$$S_i = \{s_{i,k} \mid s_{i,k} \text{ is a child page of page } p_i, s_{i,k} \neq u, p_i \in P_u, k \in [1, BF]\}, i \in [1, B].$$

Then, Steps 1 and 2 produce the following set:

$$BS = \bigcup_{i=1}^{B} S_i.$$

Step 3: Choose first F children of u.
Step 4: For each of these children c, choose up to FB parents (different from u) of c with highest in-degree. Merge the intrinsic or near-duplicate child pages, if they exist, as one whose links are the union of the links to the merged intrinsic or near-duplicate child pages, i.e., let Cu be a set of child pages of u,

$$C_u = \{c_i \mid c_i \text{ is a child page of } u \text{ without intrinsic and near-duplicate pages, } i \in [1, F]\},$$

let

$$A_i = \{a_{i,k} \mid a_{i,k} \text{ is a parent page of page } c_i, a_{i,k} \text{ and } u \text{ are neither intrinsic nor near-duplicate pages, } c_i \in C_u, k \in [1, FB]\}, i \in [1, F].$$

Then, Steps 3 and 4 produce the following set:

$$FS = \bigcup_{i=1}^{F} A_i.$$

Step 5:
For a given selection threshold δ, select pages from BS and FS such that their back Edit Distance degrees or forward Edit Distance degrees with u are greater than or equal to δ. These selected pages are relevant pages of u, i.e., the relevant page set RP of u is constructed as:

$$RP = \{p_i \mid p_i \in BS \text{ with } b(p_i, u) \geq \delta \text{ OR } p_i \in FS \text{ with } f(p_i, u) \geq \delta\}.$$

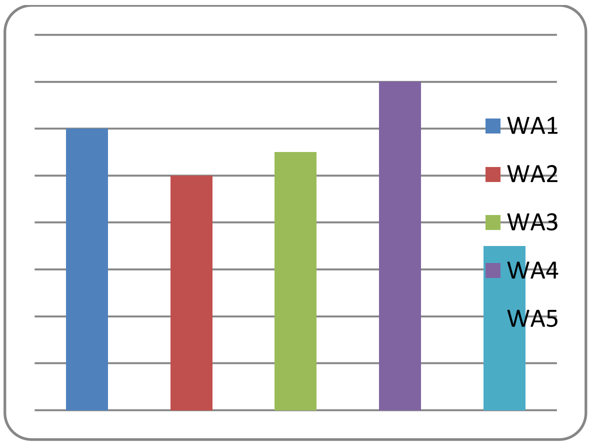
Although the LED algorithm is simple and easy to implement, it is unable to reveal deeper relationships among the pages. For example, if two pages have the same back (or forward) Edit Distance degree with the given page u, the algorithm cannot tell which page is more relevant to u.

Detecting Vulnerability server pages:
Active server pages (ASP) is one of the technologies used to create server pages; we referred to it to define an approach to detect duplicated static server pages. The approach is based on the computation of the LE distance metric.

The built-in ASP objects, together with their methods, properties and collections, may characterize the control component of an ASP page. Thus, an ASP page may be thought of as a sequence of the references to these elements. The reason of that is based on the hypothesis that, if two ASP pages present the same sequence of references to ASP features, they could have the same behavior.The symbol alphabet to be used for the LE distancecomputation will include all the built-in ASP object elements that can be referred in an ASP page.ASP Pages will be analyzed and an ASP-string extracted for each ASP page using an approach similar to the one used to compute the LE distance for HTML pages: for each couple of server pages, both ASP distance and HTML distance are computed; pages Having null distance will, again be considered as clones.only the LE distance has been considered for detecting duplicated static server pages.

Experimental Result
In comparison result, the result of this project will be compare with the existing tools like CCFinder-a clone detection tool, GEMINI- a clone analysis tool, etc.

6. Outcomes



Detection of vulnerability among the specific web applications

CONCLUSION

In this paper we have proposed a method for testing ASP web applications Vulnerability automatically. Our starting point for supportingASP webVulnerability is a crawler for ASP web applications that we proposed in our earlier work which can dynamicallymake a full pass over an ASP web application. Our current work consists of extending the crawler substantially for supporting automated testing of modern web applications Vulnerability. We developed a series of plugins, collectively, for invariant-based testing and test suite generation. o summarize, this paper makes the following contributions:
1) A series of fault models that can be automatically checked on any user interface state, capturing differentcategories of errors that are likely to occur in ASP web applications (e.g., DOM violations, error message occurrences),through (DOM-based) generic and applicationspecific invariants that serve as oracles.
2) A series of generic invariant types (e.g., XPath, template based Regular Expression, JAVASCRIPT expression) forexpressing web application invariants for testing.
3) An algorithm for deriving a test suite achieving all transitions coverage of the state-flow graph obtainedduring crawling. The resulting test suite can be refined manually to add test cases for specific paths or states,and can be used to conduct regression testing of ASP web applications.
4) An extension of our open source ASP web crawler, and the implementation of the testing approach, offering generic invariant checking components as well as a plugin-mechanism to add applicationspecificstate validators and test suite generation.
5) An empirical evaluation, by means of three case studies, of the fault revealing capabilities and the scalability ofthe approach, as well as the level of automation that can be achieved and manual effort required to use theapproach.
Given the growing popularity of ASP web applications Vulnerability, we see many opportunities for using in practice. Furthermore,the open source and plugin-based nature makes our tool a suitable vehicle for other researchers interested in experimentingwith other new techniques for testing ASP web applications. According to our page generation model, data instances of the same type have the same path in the DOM trees of the input pages. Thus, the alignment of input DOM trees can be implemented by string alignment at each internal node. We design a new algorithm for multiple string alignment, which takes optional- and set-type data into consideration. The advantage is that nodes with the same tag name can be better differentiated by the subtree they contain. Meanwhile, theresult of alignment makes pattern mining more accurate. With the constructed fixed/variant pattern tree, we can easily deduce the schema and template for the input Webpages. Although many unsupervised approaches have been proposed for Web data extraction, very few works (Road Runner and EXALG) solve this problem at a page level. The proposed page generation model with tree-based template matches the nature of the Webpages. Meanwhile, the merged pattern tree gives very good result for schema and template deduction. For the sake of efficiency, we only use two or three pages as input. Whether more input pages can improve the performance requires further study. Also, extending the analysis to string contents inside text nodes and matching schema that is produced due to variant templates are two interesting tasks that we will consider next.

REFERENCES

[1] P. Borba, A. Sampaio, A. Cavalcanti, and M. Corn´elio.Algebraic reasoning for object-oriented programming. SCP, 52: 53–100, 2004.
[2] B. Daniel, D. Dig, K. Garcia, and D. Marinov. Automated testing of refactoring engines.In ESEC/FSE '07.ACM, 2007.
[3] M. Gligoric, T. Gvero, V. Jagannath, S. Khurshid, V. Kuncak, and D. Marinov. Test generation through programming in udita. In ICSE '10, pages 225–234, 2010.
[4] V. Jagannath, Y. Y. Lee, B. Daniel, and D. Marinov.Reducing the costs of bounded-exhaustive testing. In FASE '09, pages 171–185, Berlin, Heidelberg, 2009. Springer-Verlag.
[5] G. C. Murphy, M. Kersten, and L. Findlater. How are Java software developers using the Eclipse IDE? IEEE Software, 23:76–83, July 2006.
[6] E. Murphy-Hill and A. P. Black. Refactoring tools: Fitness for purpose. IEEE Software, 25(5):38–44, 2008.
[7] W. Opdyke. Refactoring Object-Oriented Frameworks.PhD thesis, University of Illinois at Urbana-Champaign, 1992.
[8] C. Pacheco, S. K. Lahiri, M. D. Ernst, and T. Ball.Feedbackdirected random test generation. In ICSE '07, pages 75–84, Washington, DC, USA, 2007. IEEE Computer Society.
[9] M. Sch¨afer and O. de Moor. Specifying and implementing refactorings. In OOPSLA '10, pages 286–301. ACM, 2010.
[10] M. Sch¨afer, T. Ekman, and O. de Moor. Challenge proposal: verification of refactorings. In PLPV '09, pages 67–72. ACM.
[11] Yoshiki Higo a,*, Toshihiro Kamiyab,ShinjiKusumoto a, Katsuro Inoue a "Method and implementation for investigating code clonesin a software system", 2006 Elsevier.