The ART of SQL Injection Vulnerabilities Discovery [*]

Chenghong Wang
Harbin Engineering University
Harbin, China
wangchenghong@hr-
beu.edu.cn

Donghong Zhang
Harbin Engineering University
Harbin, China
zhangdonghong@hr-
beu.edu.cn

Jingyu Li
Harbin Engineering University
Harbin, China
lijingyu@hrbeu.edu.cn

Jing Zhao [†]
Harbin Engineering University
Harbin, China
zhaoj@hrbeu.edu.cn

Zhenyu Zhang
Institute of Software
Chinese Academy of Sciences
Beijing 100190
zhangzy@ios.ac.cn

Long Zhang
Institute of Software
Chinese Academy of Sciences
Beijing 100190
zlong@ios.ac.cn

ABSTRACT

SQL injection(SQLi) is one of the top risky vulnerabilities
for web applications which may lead to significant security
consequences such as authentication bypassing, privacy leak-
age, etc.. Therefore, discovering SQLi vulnerabilities is an
important step for ensuring its quality. Effective testing
techniques can ensure that applications are free from SQLi
vulnerabilities before its been released and reduce related
cost of performing manual analysis, monitoring or post de-
ployment of other defensive mechanisms. However, due to
the variety of SQLi attack types the generated test cases for
discovering SQLi are scable and complex. Moreover, suc-
cessful test cases which could discovery SQLi vulnerabilities
consist only a small part of the whole test case space. There-
fore, testing the SQLi vulnerabilities is a time consuming
and inefficient task. Towards this problem we propose an
effective SQLi testing approch based on Adaptive Random
Testing(ART) method. We define the concept of test case
distance and then present the computing method for test
case distance by using TF-IDF algorithm. Then we apply
the Fixed Size Candidate Set(FSCS) algorithm in SQLi test
case selection with the help of test case distance metrics.
We also conduct 6 experimental studies on our testing algo-
rithm, where we tested the SQLi vulnerabilities in 4 open
source benchmarks and 2 CMS applications. The experi-
mental results indicate that our testing algorithm can reduce
the redundant test cases by more than 50% and significantly
increases the testing effectiveness of SQLi vulnerabilities dis-
covering techniques.

Categories and Subject Descriptors

D.2.5 [**Software Engineering**]: Testing and Debugging

General Terms

Security, Algorithms

Keywords

SQL injection; adaptive random test; test case distance met-
rics; selection algorithm

1. INTRODUCTION

Nowadays, database-driven web applications have been
rapidly adopted in a wide range of areas including online
stores, e-commerce, social network services, etc. However
such popularity makes them more attractive to attackers.
The number of reported web attacks is growing sharply [5].
A recent web application attack report show us that, over
a period of nine months, from August 1, 2013 to April 30,
2014 reveals an average increase of around 17% in different
types of web attacks. In addition, the report also found
that web attacks have became more sophisticated and gotten
dramatically longer in length.(44% longer than they were in
previous reports). Web applications are suffering more than
26 attacks in one minute [4]. Other security report indicate
that at least 8% of the web services which belong to some
technology companies like Microsoft and Google contains
different types of security vulnerabilities [18].

Within the Web based vulnerabilities, SQLi vulnerabili-
ties has been labeled as top risky vulnerabilities by Open
Web Application Security[1]. SQLi refers to a class of code-
injection attacks in which data provided by the user is in-
cluded in a SQL query in such a way that part of the user's
input is treated as SQL code [8]. This type of vulnerabil-
ity is ultimately caused by insufficient validation of user-
input [7–9]. By exploiting such vulnerabilities, attackers can
submit malicious requests which contains sophisticated SQL
queries to remote database-driven web applications. As a
result, attackers will capture the privilege of accessing the
underlying database arbitrarily. Attackers can even take

[1]http://www.programmableweb.com,

control of and corrupt the system that hosts the Web applications by elaborately designing the injected SQL codes. During the past few years, SQLi vulnerabilities have already became the top 1 threat for database-driven web applications, and SQL injection attacks have also become the most popular web attack.

Although the cause for SQLi is quite simple and the mechanisms of SQLi are well-understood, the SQLi vulnerabilities still increase. They persist because of a lack of effective techniques for discovering and preventing them. Previous approaches to detecting SQLi vulnerabilities and preventing exploits include static analysis [6, 13, 15], user input filtering [11], defensive programming [3], runtime monitoring [2, 10, 14, 16], combining static analysis and runtime monitoring [8], source code fixing [17], application level intrusion detecting [12], SQL key words randomization [1], testing techniques [?, ?, ?, ?], etc.. Among these previous researches, it is well accepted that security testing method is the most effective way towards discovering SQLi vulnerabilities. Security testing against SQLi vulnerabilities could significantly reduce the related costs of static analysis, monitoring or deploy application level defensive mechanisms, which are acknowledged time consuming and inefficient tasks. On the other hand, security test could guarantee the web applications free from SQLi vulnerabilities before they go into services. It is much easier to repair the applications before the applications' service lifecycle than during such period. However, very few works address the issues of testing techniques towards SQLi vulnerabilities, as SQLi testing suffers from three well-known problems. The problems for SQLi testing could be conclude as follows:

a) *Scalability of Test Case Space*

The difficulty for researchers to address the full scope of the SQLi vulnerabilities is that there are many types of SQLI attacks and countless variations on these basic types. As a result, the test case space should be designed as large as possible. Otherwise some effective attack vectors which could reveal the potential SQLi vulnerabilities of testing applications may be missed.(i.e. Functions and keywords filtering prevents web applications from being attacked by using a functions and keywords black list. If an attackers submits an injection code containing a keyword or SQL function in the black list, the injection will be unsuccessful. Although the technique is very strict and perform in real world, applications with such security mechanism may also vulnerable to SQLi attacks. Attacks can evade inspection by using case changing, character encoding or inline comment methods.) Therefore, it is important to design enough test cases for testing. Equally the scale of test case space increase to a great level.

b) *Sparse Distribution of Effective Test Cases*

Though the test case space is very large, the effective test cases which could flag SQLi vulnerabilities distributed sparsely. In our previous works, we found that the effective test cases for SQLi discovery consist only 2% of the total test case pool. Such small proportion makes the searching task for effective test cases more difficult than normal testing tasks. And the performing of rest 98% can be regard as redundant works which may limit the testing effectiveness.

c) *Large Expense for Executing SQLi Test Cases*

The performing costs for SQLi vulnerabilities discovery are also very high. The judgement of test cases execution results could be a complex work. Some researchers have proposed some automatic mechanism for testing result checking and judging [?, ?, ?]. However, in some cases, it also requires some professional testers to complete the checking and judging tasks manually. i.e. In some web applications the injectable parameters appear in some particular sections like HTTP Head Referer (CVE-2011-3340[2], CVE-2008-0850[3], CVE-2007-1061[4]) section. In this cases, the most accurate and effective way to check or judge the testing is to check by testing engineers. Testing engineer have to check every test cases results during the testing period.

According to all the aforementioned problems of SQLi testing techniques, we know that SQLi testing is indeed a timing consuming and complex task compared to normal software testing. Moreover the high level of testing redundancy for SQLi vulnerabilities discovering even amplify the testing ineffectiveness and complexity. How to reduce the verbosity of test cases have became an important problem for improving SQLi testing effectiveness. To address this problem, we propose a test case selection algorithm for SQLi vulnerabilities discovering in this paper based on Adaptive Random Testing(ART) method. Through various empirical observations showing that many successful test cases are very similar between each others (i.e. Some are encoded by the same encoding method or attached with similar prefix.). More specifically, the successful test cases which could discovery SQLi have a high degree of similarity. As a result successful test cases distributed together to form clusters in total test cases domain.

Due to this distribution of successful test cases, ART is supposed to be the best choice to improve the effectiveness of test case selection tasks. In our work, the approach started from test cases distance metrics quantification, in this part we apply the TF-IDF algorithm to compute the test cases distance due to the previous observation consequences of the successful SQLi test cases distribution. Then we apply a well performed Adaptive Random Testing method called Fixed Size Candidate Set(FSCS) method to select test cases with the help of test cases distance metrics. Through FSCS algorithm, the test case selected to execute should be the most distinct (the farthest) one compared to the previous one. It is because that regions of the unsuccessful test cases domain will also be contiguous. Thus, given a set of previously executed test cases that have not revealed any failures, new test cases located away from these old ones are more likely to reveal failures.

We have also evaluated our test cases selection algorithm by comparing the discovering effectiveness of some reported vulnerabilities(reported on National Vulnerability Database) between testing approach with and without FSCS-ART algorithm. The statistic used to compare the methods was the average number of tests required to detect the first successful

---

[2]http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2011-3340

[3]http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2008-0850

[4]http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2007-1061

test case, which is commonly known as the F-measure [**?**]. In most cases, the F-measure of FSCS-ART was 30-50% lower than that of testing approach without ART selection algorithm. Experimental result indicate that our selection can significantly reduce the SQLi testing redundancy and increase the SQLi vulnerabilities discovering effectiveness.

The main contributions of this work are:

- The presentation of a new technique to test SQLi vulnerabilities that based on adaptive random method.

- The observation and analysis of effective test case distribution for dectecting SQLi vulnerabilities which shows that successful test case tend to be clustered together.

- The presentation of test case distance metrics based on string similarity metrics.

- An experimental evaluation of the technique that shows the effectiveness and the efficiency of this technique.

The remainder of this paper is organised as follows: Section 2 and Section 3 provides a background on SQLi vulnerabilities and reviews related work. Section 4 is the introduction of Adaptive Random Testing and discussion of successful SQLi test case distribution. Section 5 presents our proposed security testing approach. Section 6 presents the evaluation together with a discussion of results and threats to validity. Finally, Section 7 concludes the work.

2. BACKGROUND OF SQLI

2.1 Injection Mechanisms

2.2 Example of SQL injections

3. RELATED WORKS

3.1 SQLi Prevention

3.2 Adaptive Random Testing

Adaptive Random Testing (ART) is an effective improvement of Random Testing (RT). It is based on the observation effective test cases for testing target program tend to be clustered together. ART, therefore, proposes to have randomly selected test cases being more evenly spread throughout the test case domain by employing the location information of the successful test cases. [**?**]

4. TEST CASE DISTRIBUTIONS

Essentially, the testing process can be viewed as taking samples from the set of all SQLi attack payloads(known as the test case space) to the web application under test, executing the samples one by one, and determining whether the malicious inouts been filtered by input validation mechanisms of testing application. If the inputs get through, a payload bypassing is revealed. The presence of a bypassing implies the existence of a SQLi vulnerability. A tester seeks to select test data with a view to minimising the cost for detecting SQLi vulnerabilities. To help the tester in this task, it is natural to firstly consider how effective test cases distribute in total test cases space.

To present our work clearly, let us first define a few terms. Suppose $T = \{t_1, t_2, t_3, ... , t_n \}$ is a SQLi test case space

Table 1: Successful Test Cases for CVE2010-0415

No.	Sucessful Test Case
1	; or 1 = 1 waitfor delay '0: 0: TIME −
2	); or 1 = 1 waitfor delay '0: 0: TIME −
3	'; or 1 = 1 waitfor delay '0: 0: TIME −
4	"; or 1 = 1 waitfor delay '0: 0: TIME −
5	'); or 1 = 1 waitfor delay '0: 0: TIME −
6	"); or 1 = 1 waitfor delay '0: 0: TIME −
7	)); or 1 = 1 waitfor delay '0: 0: TIME −
8	')); or 1 = 1 waitfor delay '0: 0: TIME −
...	...

for some specific web application program (denoted by P) with n test cases. For each test case t_i in T is a type of specific string which could be injeted into web applications through input functions and flag SQLi attacks, we also call these strings SQLi attack payloads. Within test case space T we define $S = \{s_1, s_2, s_3, ... , s_k \}$ to be the effective test cases set with total k test cases, where every test case s_i in S can successfully flag SQLi vulnerabilities in application P. On the other hand $F = \{f_1, f_2, f_3, ... , f_d \}$ are defined as ineffective test cases set or redundant test cases set such that $F, S \subseteq T$ and $F \cap S = \varnothing$. The size number of total test case space, effective test case space and redundant test case space are denoted by n, k, d respectively, where $n = k + d$.

In our recent works, we have tested 29 reported SQLi vulnerabilities (Reported on NVD) by using the test cases (SQLi attack payloads) from real world. After that we have two observations from our testing. First, the test cases space collected for dectecting SQLi vulnerabilities is very huge, in other word the n is very large. However, the effective test cases consist only a small part of the test cases space, the successful ratio k/n is always below 2 percent. Second, effective SQLi test cases which could detect some specific SQLi vulnerability are always have the similar string format or constructure.

Let's take the CVE-2010-0425 (a reported SQLi vulnerability) as an example. Table I is the effective test cases we collected in our testing, it shows that the effective test cases for detecting CVE-2010-0425 are all similar to each others. All successful test cases are tautology attack payloads and the injection type are all the time based blind SQL injection mode (they contain the same terms like waitfor, delay, which are the key words for time based blind SQLi). To improve that the effective test cases share high similarity between each others, we select one effective test case s_j and then compute the string similarity between s_j and other test cases used for dectecting CVE-2010-0425.

From the details we observed, we can conclude that, effective SQLi test cases for testing target tend to be clustered together within whole test case domain, in other word if s_j is an effective test cases twords some specific SQLi vulnerability, the test cases close to s_j (similar with s_j in string format) will more likely to reveal such vulnerability.

5. THE ART OF SQLI DISCOVERY

Clustering distribution of effective test cases for detecting SQLi are indeed common. Therefore, given a set of previously executed test cases that have not revealed SQLi vulnerabilities, new test cases located away from(string format distinct from) these old ones are more likely to reveal

vulnerabilities-in other words, test cases should be more evenly spread throughout the total test case space. According to this intuition, Adaptive Random Testing (ART) method was applied to improve the effectiveness of SQLi discovering technique.

5.1 Test Case Similarity Metrics

From our previous observation of effective SQLi test cases, we found that they, effective SQLi test cases, tend to be clustered together from the string structure point of view. Thus, we define the test case distance metrics based on the string similarity and propose TF-ITSF(Term Frequency-Inverse Testcase Space Frequency) method to quantify test case similarity.

Definition 1 [Test Case Term Space] Each SQLi test case is a type of string, the elements consist of such string are called test cases terms. Suppose T is the test case space for application P. Then, $TS = \{tm_1, tm_2, ..., tm_t\}$ is the test case term space of T, denote the set of all distinct terms appear in whole test case space. The element terms in SQLi test cases area not the same terms in documents, they are more complicated. Table 2 provides a summary of the terms in SQLi test cases.

Definition 2 [Test Case Feature Vector] Suppose $T = \{t_1, t_2, t_3, ..., t_n\}$ is the test case space of testing application P, and $TS = \{tm_1, tm_2, tm_3, ..., tm_k\}$ is the term space, denote k distinct terms in total test case space. And for each test case t_j, the feature vector of t_j is denote as $\mathbf{d_j} = [w_{1,j}, w_{2,j}, w_{3,j}..., w_{k,j}]^{\mathrm{T}}$. Each dimension is called the (term) weight corresponds to a separate term in TS, where

$$w_{i,j} = \frac{\log{(TF_{i,j} + 1.0)} \times \log{\frac{|T|}{|\{t':tm_i \in t'\}|}}}{\sqrt{\sum_{i=1}^{k}[\log{(TF_{i,j} + 1.0)} \times \log{\frac{|T|}{|\{t':tm_i \in t'\}|}}]^2}} \quad (1)$$

- $TF_{i,j}$ is the term frequency of term tm_i in test case t_j.

- $|T|$ is total number of test cases , and $|\{t' : tm_i \in t'\}|$, is number of test cases containing term tm_i.

Usually, $\mathbf{d_j}$ is a column vector with high level dimensions, thus such feature vector will bring lot of costs in computing procedures. Here we apply Principal Component Analysis (PCA) method to reduce $\mathbf{d_j}$.

Definition 3 [Test Case Distance Metrics] Suppose two test case t_i and t_q, their feature vectors are $\mathbf{d_j} = [w_{1,j}, w_{2,j}, w_{3,j}, ..., w_{k,j}]$, $\mathbf{d_q} = [w_{1,q}, w_{2,q}, w_{3,q}, ..., w_{k,q}]$. We define the distance between test case t_i and t_q by using Cosine Distance(or Cosine Similarity) measurement. Such distance is represented as follows:

$$dis(t_j, t_q) = \frac{\mathbf{d_j} \cdot \mathbf{d_q}}{||\mathbf{d_j}|| \, ||\mathbf{d_q}||} = \frac{\sum_{i=1}^{k} w_{i,j} \cdot w_{i,q}}{\sqrt{\sum_{i=1}^{k} w_{i,j}^2} \cdot \sqrt{\sum_{i=1}^{k} w_{i,q}^2}} \quad (2)$$

In definition 2 , it is obviously that $\forall \, t_i \; in \; T$, the (term) weight in feature vector $\mathbf{d_i}$ are all belongs to $\mathbb{R}$. Therefore, the outcome of Equation (2), test cases distance (similarity), is neatly bounded in [0, 1]. The resulting distance

(similarity) ranges from 0 indicating orthogonality (decorrelation), to 1 meaning exactly the same, and in-between values indicating intermediate similarity.

5.2 Candidate Test Case Selection Algorithm

In this paper, we use the first ART method proposed, Fixed Size Candidate Set (FSCS)[**] algorithm to select candidate test cases. During the test case selection procedure, we apply adaptive random testing method in candidate test case selection algorithm to realize the aforementioned purpose. In our process, the selection algorithm makes use of two sets of test cases, named executed set and candidate set which. Suppose $E = \{e_1, e_2, e_3, ..., e_f\}$ is the executed set, and $C = \{c_1, c_2, c_3, ..., c_g\}$ be the candidate set such that $E \cap C = \emptyset$.

E is the set of distinct test cases that have already been selected and executed but without revealing any SQLi vulnerability while the C is a set of test cases that are randomly selected from T. The executed set is initially empty and the first test case is randomly chosen from the test case space. The executed set is then incrementally updated with the selected element c_h, ($c_h \in C$) until a SQLi vulnerability is revealed. The selected element c_h should satisfy the following condition: For all $j \in \{1, 2, 3, ..., g\}$,

$$\min_{i=1}^{f} dis(c_h, e_i) \geq \min_{i=1}^{f} dis(c_j, e_i)$$

where dis is defined in Definition 3. Specific algorithm of FSCS-ART is described in Algorithm 1.

Algorithm 1 Fixed Size Candidate Set - ART

Input:
  The set of test case space for application P, T;
  The size of candidate test case set, κ;

1: $E = \{\}$
2: $C = \{\}$
3: Training ensemble of classifiers E on $T_n \cup P_n$, with help of data in former batches;
4: $E_n = E_{n-1} \cup E$;
5: Classifying samples in $U_n - T_n$ by E_n;
6: Deleting some weak classifiers in E_n so as to keep the capacity of E_n;
7: **return** E_n;

---

6. EXPERIMENTS

6.1 Experiment Setup

6.2 Scenarios

6.3 Result & Analysis

7. CONLUSION

8. REFERENCES

[1] S. W. Boyd and A. D. Keromytis. Sqlrand: Preventing sql injection attacks. In *Applied Cryptography and Network Security*, pages 292–302. Springer, 2004.

[2] G. Buehrer, B. W. Weide, and P. A. Sivilotti. Using parse tree validation to prevent sql injection attacks. In *Proceedings of the 5th international workshop on Software engineering and middleware*, pages 106–113. ACM, 2005.

[3] W. R. Cook and S. Rai. Safe query objects: statically typed objects as remotely executable queries. In *Software Engineering, 2005. ICSE 2005. Proceedings. 27th International Conference on*, pages 97–106. IEEE, 2005.

[4] T. B. et. al. Web application attack report. 2015.

[5] M. Fossi and E. Johnson. Symantec global internet security threat report. *White Paper, Symantec Enterprise Security*, 1, 2009.

[6] X. Fu, X. Lu, B. Peltsverger, S. Chen, K. Qian, and L. Tao. A static analysis framework for detecting sql injection vulnerabilities. In *Computer Software and Applications Conference, 2007. COMPSAC 2007. 31st Annual International*, volume 1, pages 87–96. IEEE, 2007.

[7] W. Halfond, J. Viegas, and A. Orso. A classification of sql-injection attacks and countermeasures. In *Proceedings of the IEEE International Symposium on Secure Software Engineering*, pages 65–81. IEEE, 2006.

[8] W. G. Halfond and A. Orso. Amnesia: analysis and monitoring for neutralizing sql-injection attacks. In *Proceedings of the 20th IEEE/ACM international Conference on Automated software engineering*, pages 174–183. ACM, 2005.

[9] W. G. Halfond and A. Orso. Detection and prevention of sql injection attacks. In *Malware Detection*, pages 85–109. Springer, 2007.

[10] W. G. Halfond, A. Orso, and P. Manolios. Using positive tainting and syntax-aware evaluation to counter sql injection attacks. In *Proceedings of the 14th ACM SIGSOFT international symposium on Foundations of software engineering*, pages 175–185. ACM, 2006.

[11] D. LeBlanc and M. Howard. *Writing secure code.* Pearson Education, 2002.

[12] J.-C. Lin and J.-M. Chen. The automatic defense mechanism for malicious injection attack. In *Computer and Information Technology, 2007. CIT 2007. 7th IEEE International Conference on*, pages 709–714. IEEE, 2007.

[13] V. B. Livshits and M. S. Lam. Finding security vulnerabilities in java applications with static analysis. In *Usenix Security*, pages 18–18, 2005.

[14] T. Pietraszek and C. V. Berghe. Defending against injection attacks through context-sensitive string evaluation. In *Recent Advances in Intrusion Detection*, pages 124–145. Springer, 2006.

[15] Z. Su and G. Wassermann. The essence of command injection attacks in web applications. In *ACM SIGPLAN Notices*, volume 41, pages 372–382. ACM, 2006.

[16] Z. Su and G. Wassermann. The essence of command injection attacks in web applications. In *ACM SIGPLAN Notices*, volume 41, pages 372–382. ACM, 2006.

[17] S. Thomas and L. Williams. Using automated fix generation to secure sql statements. In *Proceedings of the Third International Workshop on Software Engineering for Secure Systems*, page 9. IEEE Computer Society, 2007.

[18] M. Vieira, N. Antunes, and H. Madeira. Using web security scanners to detect vulnerabilities in web services. In *Dependable Systems and Networks, 2009. DSN'09. IEEE/IFIP International Conference on*, pages 566–571. IEEE, 2009.

9. THE *BODY* OF THE PAPER

Typically, the body of a paper is organized into a hierarchical structure, with numbered or unnumbered headings for sections, subsections, sub-subsections, and even smaller sections. The command `\section` that precedes this paragraph is part of such a hierarchy.[5] LaTeX handles the numbering and placement of these headings for you, when you use the appropriate heading commands around the titles of the headings. If you want a sub-subsection or smaller part to be unnumbered in your output, simply append an asterisk to the command name. Examples of both numbered and unnumbered headings will appear throughout the balance of this sample document.

Because the entire article is contained in the **document** environment, you can indicate the start of a new paragraph with a blank line in your input file; that is why this sentence forms a separate paragraph.

9.1 Type Changes and *Special* Characters

We have already seen several typeface changes in this sample. You can indicate italicized words or phrases in your text with the command `\textit`; emboldening with the command `\textbf` and typewriter-style (for instance, for computer code) with `\texttt`. But remember, you do not have to indicate typestyle changes when such changes are part of the *structural* elements of your article; for instance, the heading of this subsection will be in a sans serif[6] typeface, but that is handled by the document class file. Take care with the use of[7] the curly braces in typeface changes; they mark the beginning and end of the text that is to be in the different typeface.

You can use whatever symbols, accented characters, or non-English characters you need anywhere in your document; you can find a complete list of what is available in the *LaTeX User's Guide* [?].

9.2 Math Equations

You may want to display math equations in three distinct styles: inline, numbered or non-numbered display. Each of the three are discussed in the next sections.

9.2.1 Inline (In-text) Equations

A formula that appears in the running text is called an inline or in-text formula. It is produced by the **math** environment, which can be invoked with the usual `\begin. . .\end` construction or with the short form `$. . .$`.

---

[5]This is the second footnote. It starts a series of three footnotes that add nothing informational, but just give an idea of how footnotes work and look. It is a wordy one, just so you see how a longish one plays out.

[6]A third footnote, here. Let's make this a rather short one to see how it looks.

[7]A fourth, and last, footnote.

You can use any of the symbols and structures, from α to ω, available in LaTeX [**?**]; this section will simply show a few examples of in-text equations in context. Notice how this equation: $\lim_{n\to\infty} x = 0$, set here in in-line math style, looks slightly different when set in display style. (See next section).

9.2.2 Display Equations

A numbered display equation – one set off by vertical space from the text and centered horizontally – is produced by the **equation** environment. An unnumbered display equation is produced by the **displaymath** environment.

Again, in either environment, you can use any of the symbols and structures available in LaTeX; this section will just give a couple of examples of display equations in context. First, consider the equation, shown as an inline equation above:

$$\lim_{n\to\infty} x = 0 \qquad (3)$$

Notice how it is formatted somewhat differently in the **displaymath** environment. Now, we'll enter an unnumbered equation:

$$\sum_{i=0}^{\infty} x + 1$$

and follow it with another numbered equation:

$$\sum_{i=0}^{\infty} x_i = \int_0^{\pi+2} f \qquad (4)$$

just to demonstrate LaTeX's able handling of numbering.

9.3 Citations

Citations to articles [**?, ?, ?, ?**], conference proceedings [**?**] or books [**?, ?**] listed in the Bibliography section of your article will occur throughout the text of your article. You should use BibTeX to automatically produce this bibliography; you simply need to insert one of several citation commands with a key of the item cited in the proper location in the `.tex` file [**?**]. The key is a short reference you invent to uniquely identify each work; in this sample document, the key is the first author's surname and a word from the title. This identifying key is included with each item in the `.bib` file for your article.

The details of the construction of the `.bib` file are beyond the scope of this sample document, but more information can be found in the *Author's Guide*, and exhaustive details in the *LaTeX User's Guide* [**?**].

This article shows only the plainest form of the citation command, using `\cite`. This is what is stipulated in the SIGS style specifications. No other citation format is endorsed or supported.

9.4 Tables

Because tables cannot be split across pages, the best placement for them is typically the top of the page nearest their initial cite. To ensure this proper "floating" placement of tables, use the environment **table** to enclose the table's contents and the table caption. The contents of the table itself must go in the **tabular** environment, to be aligned properly in rows and columns, with the desired horizontal and vertical rules. Again, detailed instructions on **tabular** material is found in the *LaTeX User's Guide*.

Table 2: Frequency of Special Characters

Non-English or Math	Frequency	Comments
$\emptyset$	1 in 1,000	For Swedish names
π	1 in 5	Common in math
$	4 in 5	Used in business
Ψ_1^2	1 in 40,000	Unexplained usage

Figure 1: A sample black and white graphic (.eps format).

Immediately following this sentence is the point at which Table 1 is included in the input file; compare the placement of the table here with the table in the printed dvi output of this document.

To set a wider table, which takes up the whole width of the page's live area, use the environment **table*** to enclose the table's contents and the table caption. As with a single-column table, this wide table will "float" to a location deemed more desirable. Immediately following this sentence is the point at which Table 2 is included in the input file; again, it is instructive to compare the placement of the table here with the table in the printed dvi output of this document.

9.5 Figures

Like tables, figures cannot be split across pages; the best placement for them is typically the top or the bottom of the page nearest their initial cite. To ensure this proper "floating" placement of figures, use the environment **figure** to enclose the figure and its caption.

This sample document contains examples of **.eps** and **.ps** files to be displayable with LaTeX. More details on each of these is found in the *Author's Guide*.

As was the case with tables, you may want a figure that spans two columns. To do this, and still to ensure proper "floating" placement of tables, use the environment **figure*** to enclose the figure and its caption. and don't forget to end the environment with figure*, not figure!

Note that either **.ps** or **.eps** formats are used; use the `\epsfig` or `\psfig` commands as appropriate for the different file types.

9.6 Theorem-like Constructs

Other common constructs that may occur in your article

Figure 2: A sample black and white graphic (.eps format) that has been resized with the `epsfig` command.

Table 3: Some Typical Commands

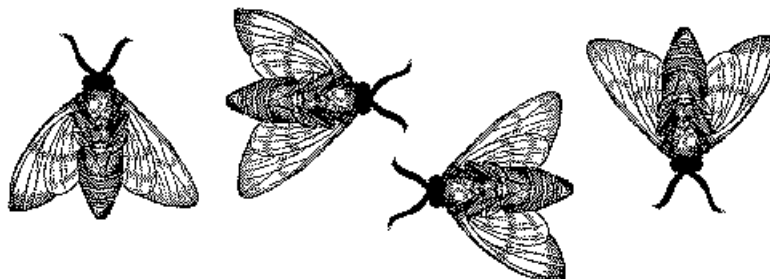| Command	A Number	Comments
\alignauthor	100	Author alignment
\numberofauthors	200	Author enumeration
\table	300	For tables
\table*	400	For wider tables

Figure 3: A sample black and white graphic (.eps format) that needs to span two columns of text.
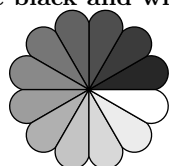

Figure 4: A sample black and white graphic (.ps format) that has been resized with the `psfig` command.

are the forms for logical constructs like theorems, axioms, corollaries and proofs. There are two forms, one produced by the command `\newtheorem` and the other by the command `\newdef`; perhaps the clearest and easiest way to distinguish them is to compare the two in the output of this sample document:

This uses the **theorem** environment, created by the `\newtheorem` command:

THEOREM 1. *Let f be continuous on $[a, b]$. If G is an antiderivative for f on $[a, b]$, then*

$$\int_a^b f(t)dt = G(b) - G(a).$$

The other uses the **definition** environment, created by the `\newdef` command:

Definition 1. If z is irrational, then by e^z we mean the unique number which has logarithm z:

$$\log e^z = z$$

Two lists of constructs that use one of these forms is given in the *Author's Guidelines*.

There is one other similar construct environment, which is already set up for you; i.e. you must *not* use a `\newdef` command to create it: the **proof** environment. Here is a example of its use:

PROOF. Suppose on the contrary there exists a real number L such that

$$\lim_{x \to \infty} \frac{f(x)}{g(x)} = L.$$

Then

$$l = \lim_{x \to c} f(x) = \lim_{x \to c} \left[gx \cdot \frac{f(x)}{g(x)} \right] = \lim_{x \to c} g(x) \cdot \lim_{x \to c} \frac{f(x)}{g(x)} = 0 \cdot L = 0,$$

which contradicts our assumption that $l \neq 0$. □

Complete rules about using these environments and using the two different creation commands are in the *Author's Guide*; please consult it for more detailed instructions. If you need to use another construct, not listed therein, which you want to have the same formatting as the Theorem or the Definition [?] shown above, use the `\newtheorem` or the `\newdef` command, respectively, to create it.

A *Caveat* for the TeX Expert

Because you have just been given permission to use the `\newdef` command to create a new form, you might think you can use TeX's `\def` to create a new command: *Please refrain from doing this!* Remember that your LaTeX source code is primarily intended to create camera-ready copy, but may be converted to other forms – e.g. HTML. If you inadvertently omit some or all of the `\def`s recompilation will be, to say the least, problematic.

10. CONCLUSIONS

This paragraph will end the body of this sample document. Remember that you might still have Acknowledgments or Appendices; brief samples of these follow. There is still the Bibliography to deal with; and we will make a disclaimer about that here: with the exception of the reference to the LaTeX book, the citations in this paper are to articles which have nothing to do with the present subject and are used as examples only.

11. ACKNOWLEDGMENTS

This section is optional; it is a location for you to acknowledge grants, funding, editing assistance and what have you. In the present case, for example, the authors would like to thank Gerald Murray of ACM for his help in codifying this *Author's Guide* and the **.cls** and **.tex** files that it describes.

APPENDIX

A.   HEADINGS IN APPENDICES

The rules about hierarchical headings discussed above for the body of the article are different in the appendices. In the **appendix** environment, the command **section** is used to indicate the start of each Appendix, with alphabetic order designation (i.e. the first is A, the second B, etc.) and a title (if you include one). So, if you need hierarchical structure *within* an Appendix, start with **subsection** as the highest level. Here is an outline of the body of this document in Appendix-appropriate form:

A.1   Introduction

A.2   The Body of the Paper

A.2.1   Type Changes and Special Characters

A.2.2   Math Equations

Inline (In-text) Equations.

Display Equations.

A.2.3   Citations

A.2.4   Tables

A.2.5   Figures

A.2.6   Theorem-like Constructs

A Caveat for the TEX Expert

A.3   Conclusions

A.4   Acknowledgments

A.5   Additional Authors

This section is inserted by LATEX; you do not insert it. You just add the names and information in the `\addition-alauthors` command at the start of the document.

A.6   References

Generated by bibtex from your .bib file. Run latex, then bibtex, then latex twice (to resolve references) to create the .bbl file. Insert that .bbl file into the .tex source file and comment out the command `\thebibliography`.

B.   MORE HELP FOR THE HARDY

The sig-alternate.cls file itself is chock-full of succinct and helpful comments. If you consider yourself a moderately experienced to expert user of LATEX, you may find reading it useful but please remember not to change it.