

DOI: 10.53555/ks.v12i5.3326

S-RealSpec: A Security Extension to Detect SQLI attack and Sensitive Data Exposure

Muniba Murtaza^{1*}

^{1*}Department of Computer Science, Faculty of Computing and Information Technology, International Islamic University, Islamabad, 192122, Pakistan Email: muniba.phdcs135@iiu.edu.pk

***Corresponding author :** Muniba Murtaza

Email: muniba.phdcs135@iiu.edu.pk

Abstract: As security flaws can result in considerable financial losses in rework and a bad reputation due to subpar web apps, there is a growing area of the security of web applications. Online application security is becoming more and more of a concern since security holes can cost a lot of money in rework and damage the reputation of a business because of poor online applications. Poor modelling and design processes that neglect to model and create essential logging requirements and data validation security features and apply them haphazardly during development are the root cause of SQL Injection attacks and sensitive data exposure, among other types of attacks. Throughout the software development life cycle, specification languages are used to describe the security requirements for secure logging and data validation. To counteract attacks involving the sensitive data exposure, the specification languages do not, however, include detailed particular security requirements for secure logging and data validation. Additionally, this research project offers RealSpec security extension to detect SQLI attacks and sensitive data exposure. Early in the requirement analysis and design process, the goal of this effort is to define, record, and validate security requirements and integrate security throughout software development. To transform specification from design to implementation level a custom compiler is then used to convert the requirements into C++ code. The suggested method then compares the C++ code to attack patterns; if an attack is found, the system throws an exception.

Keywords: security feature; security requirements; model-driven security; MDS; evaluation framework; secure auditing; secure logging; specification languages; SQL Injection; data validation.

1 Introduction

The use of software systems in our daily lives has grown more and more necessary. A software system or logs include users' private and sensitive information, making it possible for even a minor security flaw to reveal sensitive data (Sharma, 2020; Van den Berghe et al., 2017). Careful security engineering in overall system design is frequently disregarded, according to an examination of current software development methods (Jürjens, 2002; Hayati et al., 2008). Post-hoc security need additions lead to poor integration with other system requirements (Hayati et al., 2008), which lowers the quality of the resultant software system (Bagale et al., 2021). If the security needs are clearly established in the system analysis and design phases, together with other system requirements, high software maintenance costs can be avoided (Khwaja and Urban, 2002). According to Khwaja and Urban (2002), the post hoc approach also results in an incompatibility of design and implementation models and unfulfilled security needs. Growing business demands are a concern that is associated with software issues, namely security issues (Zeyun and Dawood, 2016; Vashishtha and Dhawan, 2023). The developers are hesitant to select a suitable formal security model in this rigidly market-driven context since it takes a lot of learning (Ghozali et al., 2022a, 2022b). Since developers aren't security experts, they need to know how to model both fundamental and complex security requirements by choosing the right model and tool chain (Vashishtha and Kapoor, 2023). The concept of modeldriven security (MDS) has been the focus of research for the past ten years. It incorporates security into the overall design of a system (Jürjens, 2002; Hayati et al., 2008; Khwaja and Urban, 2002; Lucio et al., 2014; Memon et al., 2008; Fernández et al., 2006; Hochreiner et al., 2015; Ghozali, 2022; Ocoró et al., 2023). By defining security requirements as a high-level abstraction rather than a particular platform-related implementation, which is subsequently translated into platform-specific models, MDS additionally offers platform independence (Hochreiner et al., 2015).

A web application's framework display shows how it functions, and its unique features; a danger indicates the strength and resources of the attackers; and a security feature describes the web application's behaviour as the engineers designed it (Saleh et al., 2021; Priscila et al., 2023). If a security feature such as data validation is applied during early design phase it can counteract sensitive data exposure and Structured Query Injection (SQLI) attack. These are a primary objectives of security feature framework (SEFF) (Khwaja et al., 2020; Sharma and Sharma, 2021a, 2021b). Sensitive data exposure demonstrates that users' confidential and private details, like location information, photos, login passwords, or other documents stored on a website or a mobile device of the user, may be released in an unwanted manner. The top ten list of Open Web Application Security Project (OWASP) vulnerabilities includes sensitive data disclosure. An attacker may purposefully or unintentionally leak sensitive data through a poorly programmed application (Zhu et al. 2011). Tom-Skype (Zhu et al. 2011) is a text writer that

creates temporary duplicates of data, which can reveal sensitive information. TaintEraser (Zhu et al. 2011) is a prevention method for preventing secret information from being exposed. TaintEraser tracks tainted sensitive data and replaces it with random bytes for export to the network and local file system using application-level dynamic taint monitoring. It preserves a hidden list of kernel-level tainted elements in user space to keep track of which file is open. The user must, however, manually identify confidential data in the beginning, which is a limitation of such a mitigation method. The SQL injection attack is launched through the Web application, with the intruder inserting specially crafted user input into SQL queries to get data from the database (Gu 2020; Bisht et al. 2010; Pham and Subburaj 2020; Nikiforakis et al., 2011).

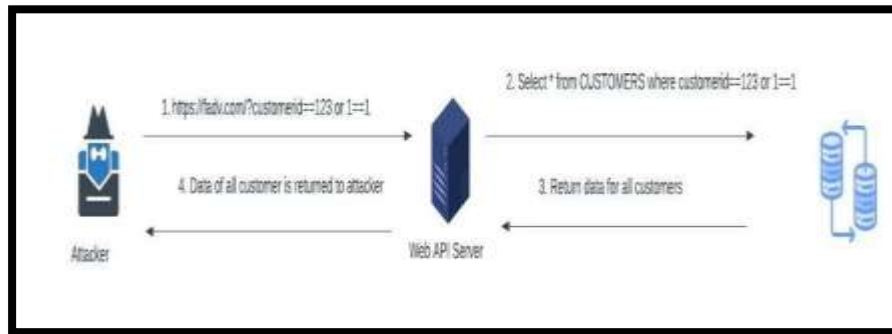


Figure 1 Structured Query Injection attack

This thesis only addresses tautology and error-based SQLIA; modifications can address other versions. SQLIA tautology is depicted in Figure 1. SQLIA based on errors. One potential SQLI approach is to utilize a template for the query in SQL that the user will accept (Wang et al. 2019). SQLCheck (Gu 2020), CANDID (Bisht et al. 2010), Pham and Subburaj provided classification methodology (Pham and Subburaj 2020) and DIAVA (Gu 2020) proposed multi-level regular expression method to identify SQLIA are additional techniques to prevent SQLI attacks. This thesis also specifies security requirements in RealSpec (Khwaja 2009; Khwaja 2015; Khwaja 2002). RealSpec is executable specification language for embedded systems.

The organisation of this paper is as follows: Section 2 proposes a security requirements to prevent SQLI attack and sensitive data exposure. Section 3 proposes a solution to model security requirements at analysis and design phases. Section 4 provide a custom tool to transforms the RealSpec (Khwaja, 2009) security specification to C++ code at implementation phase and testing of the C++ code is also performed and in case attack is detected then an exception is thrown. Section 6 concludes the paper.

2 Security Requirements to detect SQLI and Sensitive Data Exposure

We established a SEFF in our earlier research (Khwaja et al., 2020) to assess security aspects in programming languages. For programming languages, SEFF offers an extensive feature set of security measures. We aim to measure the effect of a programming language of choice on the security of software written in that language. If such a factor exists, software engineers or their managers may take it into account when selecting the programming system to use for a certain task. This information could help with risk reduction and more efficient use of resources (Khan et al., 2019). There are various reasons to believe that the characteristics of a programming language may have an impact on the security of applications written in that language (Uddin et al., 2022). The study has shown that type systems, for example, are able to statically identify (and thereby prevent, by preventing the compilation of certain types of faults). Static typing, in general, can highlight flaws that could be potentially hidden until they were used in a dynamically typed language. Furthermore, one language's standard frameworks might be easier to use than another, which would reduce their mistake rate (Alarood et al., 2022).

With the use of an updated exception resolution system, developers might be able to identify dangerous circumstances and avoid them (Rani et al., 2021). However, the distinctions among programming languages extend well beyond the scripts themselves (Ullah et al., 2020). Every language has its own society, and these societies may have distinct values and beliefs. Therefore, our goal is to determine whether language choice affects the application's overall security in a quantifiable way. If this is the case, it could be useful to determine whether a specific weak point is better addressed by one language over another (Sharma et al., 2021). Should this be the case, authors may focus their attention on the classes for which their programming style does not offer sufficient support, and they may become less concerned about the classes for which data show a strong dialect (Mast et al., 2021). UMLsec (Jürjens, 2002; Hochreiner, et al., 2015) were the first to introduce security notations into software specifications and designs in 2001 by extending the standard UML profile, and it also provides a baseline for comparison with other notations (Hochreiner et al., 2015). UMLsec uses UML diagrams, stereotypes, tags, and constraints in modelling security requirements such as user authentication support, input validation, access control, database query security, type system, and partial support of log message control. UMLsec models secure auditing using a state chart diagram, providing log entry accountability (Hochreiner et al., 2015). Even though UMLsec provides comprehensive security feature coverage (Khaled, 2021). SysML-sec (Roudier and Apvrille, 2015), Secure Descartes (Inukollu and Urban, 2020), UMLsec (Jürjens, 2002;

Hochreiner, et al., 2015; (Jürjens and Shabalin, 2005); Hayati et al. (2008); S-Promela (Abbassi and El Fatmi, 2009); SecureSOA (Rafe and Hosseinpouri), 2015); Ponder (Naqvi et al., 2006) are some of the executable specification languages that provide

security requirements in analysis and design phases. However, the mapping from SEFF and detailed security features and sub-features are not modelled by any of the mentioned specification languages.

In order to assess the security capabilities of specification languages, this section suggests using secure auditing, a security requirement extracted from SEFF (Khwaja et al., 2020) (Tripathi and Al-Shahri, 2023). Early in the development process, a specification language that covers all security requirements thoroughly can aid in the formulation of accurate, thorough, and consistent security requirements, which can improve software design and implementation (Khwaja, 2009). A complete set of security features that are abstracted to the specification language framework are provided by using the SEFF framework for programming languages as a baseline. This also helps close gaps in the transformation of abstracted specification features into some platform-specific programming language (Basha and Sivakumar, 2020). As a result, a clear relationship between security requirements and security features may be found.

Error handling and log file protection and data validation security features and its subfeatures in Table 1 are extracted from (Khwaja et al., 2020) which show the possible attacks that are mitigated if a certain feature is applied. Table 1 comprises four columns. The security elements and sub-features from (Khwaja et al., 2020) are listed in columns 1 and 2. Column 3 show security requirements mapped in RealSpec and column 4 indicates the attacks mitigated.

In Table 2, security features mapping to abstract security requirements is shown. Column 1 and Column 2 shows features and sub-features from SEFF. Column 3 is justified by Column 4 to determine whether it can or should be included at the modelling level or not. Column 5 is the name of abstracted security requirement.

Table 1 Security Features that mitigate SQLI attack and Sensitive Data Exposure

Feature	Sub-Feature	Security Requirements Mapped in the case study	Attacks Mitigated
Error Handling and Protection	HandlingError and Logging Protection	This feature is specified using logfile resource	Sensitive Data Exposure
	Log Information Level	This feature is specified using log file severity level	Sensitive Data Exposure
	Error Message Control	This feature is specified by verbosity control of error message using roles and privileges for error messages	SQLI,
Input Validation	Database Query Security	This feature is specified using database resource and validating user input against SQLI attack pattern.	SQLI
	User Input Security	By data validation operator net and pattern matching	SQLI
	Input Buffer Size Check	This feature is specified by verifying buffer limit and an exception is thrown in case buffer limit has been crossed.	SQLI

Table 2: Security feature mapping

Security feature	Security feature	sub-Abstracted for modelling?	Reason	Security requirement
Error handling and logging protection	Log protection	fileYes	Logging is used for accountability and record-keeping (Fernández et al., 2006). It can be specified as: Graphical notation using class diagrams (Hochreiner et al., 2015) Audit constraints as (Fernández et al., 2006) or custom-made grammar (Sommestad et al., 2012)	Secure auditing
	Log message control	ErrorYes	Log message control stores information in logs based on the severity level, such as information, debug, trace, warning, and error (Li et al., 2018). Message control is essential to abstract logs, as studies have shown that developers do not log with appropriate severity levels (Li et al., 2018). It can be specified as: Log message control using a graphical notation, such as using the log method of logger class (Hochreiner et al., 2015)	Log message control ror message control
	message control			

			Constraints limit the information to be stored in the logs (Hochreiner et al., 2015).
			Error message control means controlling the verbosity of an error message. Some error message details can guide the attacker to exploit the possible values for the wrong entry. Error message control modelling can help identify and specify these controls as constraints, like the log message control feature. It can be specified as: A low verbose error message constraint for a specific role and a verbose error message for a developer or authorized user.
Input validation	Database query security	Yes	Database query security is essential because tainted user input can lead to SQLI. Database query security modelling can prevent SQLI from properly constructing database queries using specification language construct for the query, along with rules for correct query formation (Hochreiner et al., 2015). It can be specified as: Prevention of XSS and SQLI tags (Hochreiner et al., 2015). Constraints defining denylists features of SQLI, XSS attacks, or safelists of acceptable inputs.
	User input security	Yes	Modelling input validation ensures the system operates on correct and meaningful input (Inukollu and Urban, 2020). It can be specified as • Safelist and denylist constraints for user input (Hayati et al., 2008; Sommestad et al., 2012).
	Input buffer size check	Yes, access limit prohibition	Checking buffer boundary limits before taking input from the user is a must. It can be specified as Attack prevention mechanism as (Sommestad et al., 2012)
Constraint to check bounds and then throw an exception			

Security features	Sub-features	SysML-sec (Rowder and Azeville, 2013)	Secure Describes (Sudholc and Urban, 2020)	UMLsec (Aljreis, 2007; Hochreiner, et al., 2015; Aljreis and Shobaili, 2003)	Hayati et al. (2008)	S-Process (Abbasz and El Fares, 2009)	SecureSOA (Rafiq and Hussainpour, 2015)	Powder (Nasri et al., 2006)
Error handling and log file protection	Secure auditing	NS	FS #14 Maintenance, monitoring, and analysis of logs	FS By protection tags for logs (Hochreiner, et al., 2015) <<encrypt>> tags	NS	NS (Inukollu and Urban, 2020)	FS Cryptography (Rafiq and Hussainpour, 2015)	P.S. Auditing from (Nasri et al., 2006)
	Secure error message	NS	NS	NS	NS	NS	NS	NS
	Secure Error Message control	NS	NS	NS	NS	NS	NS	NS
Data validation	Database Query Security	NS	NS	FS Database class and tags to protect queries (Hochreiner et al., 2015) and separate input FS	FS Using activity diagram and OCL constraints. Does not FS Using activity diagram.	NS	NS	NS
	Input Validation	NS	NS	NS	NS	NS	NS	NS
	Buffer bound limit access prohibition	NS	FS #13 boundary defence	NS	diagram and OCL constraints. Does not Using OCL constraints	FS (Inukollu and Urban, 2020)	NS	FS (Inukollu and Urban, 2020)

Table 3 shows the security feature is fully supported, partially supported or not supported by above mentioned specification languages.

4 Specification of security features that detect SQLI attack and Sensitive Data Exposure

RealSpec defines Log File as a resource as a shown in Table 4.1. The first of the three inputs for the logfile are encryptionStatus, which is followed by pol and severityLevel. For instance, to ensure thread safety in a concurrent environment, pol is a policy specified for mutex resources, severityLevel of the log is used to select the level of information to be stored in log files, and encryptionStatus is required to figure out whether the log file is encrypted. Currently, two policies are supported: the default policy, which uses first come, first served (FCFS), and the priority system, that uses the thread's priority. ERROR, WARNING, TRACE, DEBUG, INFO, ALL, and OFF are only some of the severity levels of logs that the log4Net and Log4j programming language libraries provide. Internally, a logfile is described as a list resource. Moreover, there is an index which is used to hold a cursor point where a file can be read from and in case of write mode the index is set to 0 that means index is set at starting position of the file. Algorithm1 and Algorithm 2 show secure auditing write and read function respectively.

Table 4.1 Log File Signature

Signature	Logfile (bool encryptedStatus, int pol, int severityLevel)
System variables	
Private variables	list ldisk=[]; bool status= encryptedStatus; int index; mutex file(pol); list qlist; generic input; generic buffer;
User variables	
User functions	int open (int mode); generic operator << (generic input, generic p); generic operator>> (generic buffer); bool isEncrypted(); bool fileSize(); generic loglevel (int severity);

Algorithm 1 Secure Auditing Write function

Require: ldisk is log file, ldisk is a mutex file to check mutual exclusion while write operation to ldisk, encryption status checks encrypted text or plain text, index variable to check current location to read a file, eod to show end of file

Ensure: mutual exclusion while write operation, encrypt the log statement if the log is stored in encrypted form.

Start

Initialize ldisk to logfile

While index !=eod **do**

For each ldisk open in write mode **do Lock** ldisk

If ldisk is encrypted

Write ldisk in encrypted form

Else

Write ldisk

End if

Unlock ldisk

End While

Algorithm 2 Secure Auditing Read function

Require: ldisk is log file, encryption status checks encrypted text or plain text, index variable to check current location to read a file while read operation to ldisk, eod to show end of file.

Ensure: decrypt the log statement if the log is stored in encrypted form, Start

Initialize ldisk to

logfile

While index !=eod **do**

For each ldisk open in read mode **do If** ldisk is

encrypted **Decrypt** ldisk **read** ldisk *Else*

Read ldisk **End if**

End While

Table 4.2 Specification of Error Message Control

Signature	SecureErrorMessage ()
System variables	
Private variables	List Policy1
User variables	String role int username int actions generic object
User functions	Bool isAuthorizedUser() generic CheckMessageVerbosity()

Secure Auditing and Log Message Control specification in RealSpec

```

Resource logfile (bool encryptionStatus, int pol, int severityLevel) {
List ldisk = [];
int index;
int encrypted = encryptionStatus;
int keysize = 128; mutex file(pol); list qlist=[];
int initLevel;
int open (int mode, int severityLevel) = reinitialize ()
where {
reinitialize() = case mode of {
WRITEONLY: CheckSeverity(SeverityLevel);
Default: READ_ONLY()
where {
qlist = if ldisk !=nil then ldisk; READ_ONLY()= qlist;
}
}
}
generic checkSeverity(int SeverityLevel) = Severity
where { ldisk=[]; index =0;
Severity= case severityLevel of {
ALL: initLevel=int.MAXLEVEL;
FATAL: initLevel=100;
ERROR: initLevel=200;
WARNING: initLevel=300
INFO: initLevel=400;
DEBUG: initLevel=500;
TRACE: initLevel=600;
Default: initLevel=0;
}
}
int fileSize(generic p) = (size asa file.lock(p)) asa file.unlock()
where {
size asa dis==nil
where {
size= 0 fby size + length(hd(dis)); dis= ldisk fby tl(ldisk);
}
}

Bool isEncrypted()= encryptedStatus;
generic operator << (generic input, generic p) = ((write () fby msg) asa file.lock(p)) asa
file.unlock() where {
write() = TRUE
where {
ldisk = if isencrypted() && input.size<=initLevel then ldisk <> [% encrypt (input, keysize) %] else if
input.size<=initLevel && !isencrypted()ldisk <> [% input %] else printMsg;
index = index + 1;
where {
printMsg= "input is verbose, severity level ^ mkString (SeverityLevel) ^ number of character allowed be
written are ^ mkstring(initLevel)
}
}
}

```

```

generic operator >> (generic p) =(read() asa file.lock(p)) asa file.unlock()
where {
read()= buffer asa ldisk==nil
where {
buffer = if isencrypted then decrypt(hd(dis), keysize)
else hd(dis); dis= ldisk fby t1(ldisk)
}
}

```

Explanation

Read(>>) and write (<<) operators are overloaded for the logfile. The open function accepts the mode, such as read or write mode, and performs the function based on the defined mode. When a logfile is opened, the open function mode parameter is passed to the open function to check if the file is opened for read or write. Mode variable takes constant values such as *WRITE_ONLY* and *READ*. Function **open(int mode, int severityLevel)** checks if the mode is *WRITE_ONLY*, then the logfile contents are wiped, and the index pointer is set to 0. The severityLevel shows which logging level should be applied as logging everything can exhaust system resources, and logging too less can complicate debugging. Thus, the recommended level of logging is information. The multithread-safe logfile write() and read(>>) methods are provided by employing the mutex resource. The logfile is overloaded for both << and >>. When a thread locks the logfile while writing to the disc, the write () operator writes a string to the logfile. The logfile ldisk is also unlocked soon after the thread finishes writing to it. The operator first finds whether the logfile, ldisk, is encrypted. Subsequently it encrypts the input string using keysize. Lastly, it upgrades the logfile, ldisk, by converting the input variable into a list using the list operator [%%%] and appending to the end of the ldisk using the append operator. <>. Lastly, the index pointer is updated by 1. The highlighted area shows the immutability feature preventing race conditions, thereby preventing TOCTOU. Function *CheckSeverity(int initLevel)* checks the severity of a message to be stored in a log file. The variable *initLevel* is assigned with an allowed number of characters to be stored in a log file. This specifies the log severity level.

In RealSpec, log message control is shown by severity level case statement where fatal where fatal mode can store 100 characters, error stores 200 characters, warning shows 300 characters, info stores 400, debug stores 500 characters, trace store 600 characters.

Algorithm 3 Secure Error Message

Require: List policy to store organization policy, r is role, level is number of words to control message wordiness, m is the error message, E is the total number of epochs.

Ensure: each role is assigned with a specific message wordiness level to control message verbosity

```

Start
Initialize policy to OrganizationPolicy
For each epoch e= 1 to E
If r is authorized user for level on m
Show level number of words from m to r
Else
    Show error message
End if
End for

```

Secure error message specification in RealSpec

```

System SpecifyErrorMessageControl{ Resource {
List Policy1;
}
Process {
ControlErrorMessageVerbosity();
}
}
Resource Policy (string userName, int roleName, int permission, generic Obj) {
string role=userName; int Subject=roleName; int Action= permission; generic object=obj;
bool isAuthorizedUser(int actor, int perm, generic obj) = CheckPolicy()
where{
CheckPolicy()=if checkPolicyExist(actor, action, object) then true else

```

```

False;
}
Process ControlErrorMessageVerbosity () { Generic CheckAction()= if
Policy1.isAuthorizedUser(Policy.RName,Policy.Action,Policy.Obj) then PrintMsg else throw InvalidActionException();
Where {
PrintMsg= mkstring(RName)^ "Authorised User can see verbose error messages"
}
Exception InValidException()= Message Where{
Message= "Invalid Action";
}
}

```

Explanation

In RealSpec, a secure error message can be specified as an object, and an authorised actor can get permission to see a lengthy error message and an unauthorised user can only see a less informative error message. Here, a SP is created to specify a secure error message (Table 5).

Table 4.3 Execution

<i>t0</i>				<i>t1</i>				<i>t2</i>			
Rname	Actor	Action	Object	Ali Rname	Actor	Action	Object	Rname	Actor	Action	Object
Sam	Developer	Stacktrace access	Error message		Tester	Stacktrace access	Error Message	sonic	Visitor	Verbose Error Message	Error Message
Sam authorised user can see verbose error messages				Ali authorised user can see verbose error messages				Invalid action			

Specification of Data Validation in RealSpec

RealSpec defines data validation as a resource as a shown in Table 4.4. There are two user variables and two processes one input validation and one is output validation. Process *InputValidation()* has one operator net *dataSanitization()* that detects SQLI pattern such as 1==1 in the input and if found it throws error message. RealSpec specification of data validation is given below.

Table 4.4 Specification of Data Validation Resource

Signature	DataValidation ()
System variables	
Private variables	
User variables	String input;
User functions	generic dataSanitization(generic query)

Algorithm 4 Input Validation

Require: I is user input, p is the pattern to find in the input, E is the total number of epochs, q is a query

Ensure: regex find P in I

Start

Initialize I with user input q

For each Epoch e=1 to E **do** **For** each **where** clause **do**

Find P in user input

If found **then** attack print attack message

Else print output

End if

End for

End for

In Figure 4.1, user gives an input the web application checks for the input and finds out if there is some pattern that matches with SQLI attack pattern then if the pattern is found then the web browser throws an exception otherwise the request is processed. *Specification of InputValidation in RealSpec*


```

System DataValidation {
Processes
  { InputValidation;
  }
}
    Process InputValidation() {int num1, num2, x; String input;
// sqli detection function
    Bool dataSanitization (string input)= sanitize
where { pattern =
(“.*\\b(OR|WHERE|1==1|tr
ue==true)\\b.*”);
sanitize= if match(input, pattern) then sqliDetectionMsg() else printMsg;
where {
sqliDectectionMsg()= "Invalid input: SQL injection tautology detected." printMsg= "no SQLI detected";
}
}
// exception
Exception
InvalidDataException=..

```

Testing

The process InputValidation(), if input is given by illegitimate user as “1==1-- ana OR” then the output will be displayed as Invalid input: SQL injection tautology detected if select ename from employees where id==001 is given then no SQLI detected is shown..

4.5. Bound Access Prohibition

The input buffer or output buffer can be represented by a list in RealSpec and If accessing passes the bounds it will result in nil. This is shown in table 4.4 BoundCheckList is a list resource. The system BufferBoundCheck has one private variable and one user variable input. It has two operator nets BoundCheck() and addToList(). The addToList() adds items to the list as soon as asa the buffer reaches to nil. The operator asa will evaluate the right-hand side first. This means the buffer bound is check first for its limit and then items are added to the list. This operator <> appends the input to the BufferCheckList by first converting input to the list item using [% %] and if the list reaches its limit then exception BufferOutOfBoundCheckException() is thrown. This whole function iterates for n number of times using i=1 fby i+1 in which i is initialized with 1 and then fby stands for followed by gives the subsequent values of i.

Table 4.5 specification of bound access prohibition

Signature	BufferBoundCheck()
System variables	
Private variables	list BoundCheckList
User variables	String input
User functions	generic BoundCheck() generic addToList()

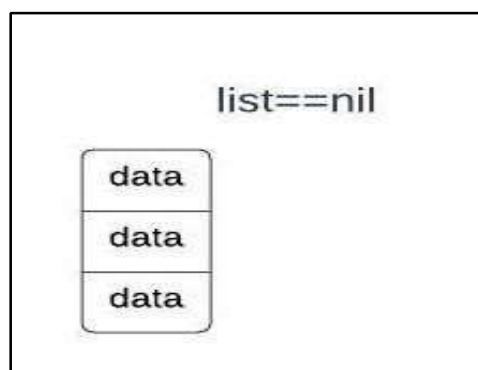


Figure 4.3 Specification Buffer Bound Access

Bound Access Prohibition specification in RealSpec

```

System BufferBoundCheck {
Resources {
list BoundCheckList=[];
}
}

```

```

Process {
BoundCheck();
}
};
Process BoundCheck() { generic addToList()= addDataTo�List() asa
BoundCheckList!=nil where { addDataTo�List()= if
BoundCheckList[j]!= nil then BoundCheckList <>
[% input%] else
BufferOutOfBoundCheckEx ception(); i=1 fby i+1;
}
Exception
BufferOutOfBoundCheckEx ception()= PrintMsg
where {
PrintMsg= "accessing past the buffer bound is illegal";
}
}

```

Testing

Similar to Java programming language where writing past the length of the buffer throws an exception preventing buffer overflow attack. BoundCheckList has three data elements and addToList() will always check if the buffer size has reached maximum by checking for nil otherwise continue to append data in the list.

	t0	t1	t2
d0	2	4	Nil
d2	3	6	Nil
d3	4	8	Nil

5 Verification of the specification

The evaluation of the specifications in chapter 4 and 5 is executed by a prototype compiler. There are three components of the RealSpec compiler: lexical analyzer, parser, and the code generator (specification of the compiler is given in (Khwaja 2009). RealSpec compiler is written in C# language.

Using the following packages

```

using System;
using System.Collections.Generic using System.IO; using
System.Text;

```

Figure 5.1 shows the overall working of the compiler. At first, the specification written in RealSpec is given to lexical analyzer to tokenize the statements. Next, the parser takes that input tokens and the language grammar is also given to match. Further, the tokens are mapped to C++ code. Moreover, the C++ code is fed into its compiler and the code is given test cases for attacks. If the attack pattern is detected then exception is thrown otherwise normal output is shown. Before verification can start, model checking usually entails a significant amount of preparatory effort. This includes establishing system attributes using logical formalisms and translating the specification into a formalism compatible with a modelling tool. Unlike model checking approaches, which involve an upfront formalism conversion, RealSpec's execution model is clearer and more similar to programming languages, making it easier to simulate and monitor system behaviors directly.

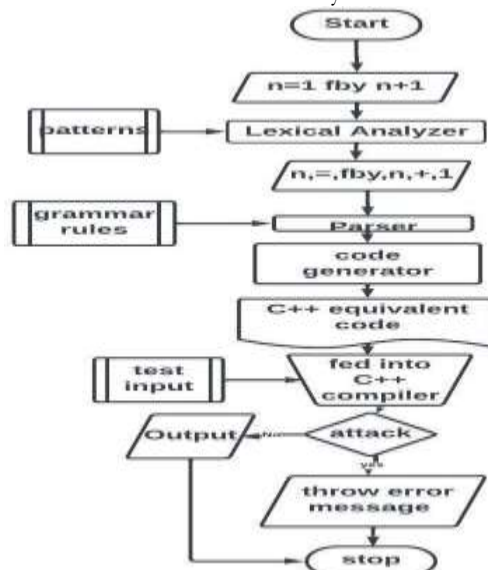
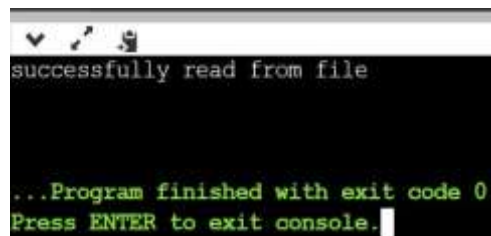


Figure 5.1 Compiler of RealSpec and transformation of RealSpec Specification to C++ code



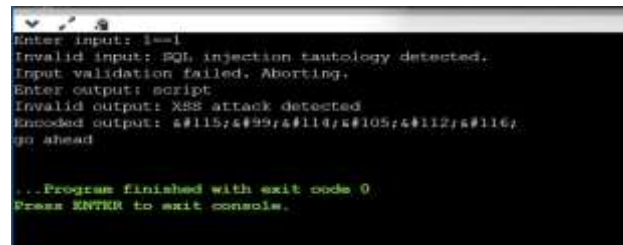
```

successfully read from file

...Program finished with exit code 0
Press ENTER to exit console.

```

Figure 5.2 output of the C++ code when there is no attack



```

Enter input: i=1
Invalid input: SQL injection tautology detected.
Input validation failed. Aborting.
Enter output: script
Invalid output: XSS attack detected
Encoded output: %115%99%119%105%112%116%
go ahead

...Program finished with exit code 0
Press ENTER to exit console.

```

Figure 5.3 output of the C++ code where attack is detected

6. Conclusions

Web application security remains the main issue in today's web-based environment. Substantial research efforts show that researchers have been specifying security requirements in specification languages through the software development phases. The most common platform for developing novel software applications is now the web. As a result, fresh web apps are constantly being created, which raises the significance of these programs' cybersecurity. Web apps control the personal, private, and financial data of users which if leaked can cause SQLI attack and sensitive data exposure. Organisations may incur costs due to web application holes, including possible direct cash losses, increased technical support needs, and damage to reputation and brand. This study proposed security requirements from security features framework (SEFF) that mitigates SQLI attacks and sensitive information exposure. By eliminating safeguard gaps from modelling to implementation, this direct mapping from SEFF security features to security requirements aids in addressing security holes in implementation and, presumably, lowers the number of SQLI attack and sensitive data exposure. Additionally, utilising RealSpec constructs to show specification of security requirements and then a mapping is shown to transform these specifications to C++ code. Moreover, attack pattern is used to check if SQLI attack and sensitive data exposure is detected and in case the attack is not detected then normally output is displayed and if the attack is detected then exception is thrown.

a. Future work

Future work is to specify security features from SEFF that mitigate other attacks such as buffer over flow, return oriented programming attacks, other variation of SQLI attack, cross-site scripting attacks and broken authentication.

References

1. Abbassi, R. and El-Fatmi, S.G. (2009) 'S-Promela: an executable specification security policies language', *2009 First International Conference on Communications and Networking*, IEEE.
2. Alarood, A.A., Alsolami, E., Al-Khasawneh, M.A., Ababneh, N. and Elmedany, W. (2022) 'IES: hyper-chaotic plain image encryption scheme using improved shuffled confusion-diffusion', *Ain Shams Engineering Journal*, Vol. 13, No. 3, p.101583.
3. Basha, A.R. and Sivakumar, G. (2020) 'An energy and delay aware optimal data aggregation using three fold algorithm for WSN', *International Journal of Business Information Systems*, Vol. 33, No. 3, p.299, DOI: 10.1504/ijbis.2020.105827.
4. Bagale, G.S., Vandadi, V.R., Singh, D., Sharma, D.K., Garlapati, D.V.K., Bommisetti, R.K., Gupta, R.K., Setsiawan, R., Subramaniaswamy, V. and Sengan, S. (2021) 'Small and medium-sized enterprises' contribution in digital technology', *Annals of Operations Research*, pp.1–24, Press.
5. Bisht, P. Madhusudan, P. and Venkatakrishnan, V.N. 'CANDID.' *ACM Transactions on Information and System Security*, vol. 13, no. 2, pp. 1-39, 2010, doi: 10.1145/1698750.1698754.
6. Busch, M. (2011) *Integration of Security Aspects in Web Engineering*, Master's thesis, Inst. für Informatik Ludwig-Maximilians-Uni.
7. Fernández, M.E., Trujillo, J., Villarroel, R. and Piattini, M. (2006) 'Access control and audit model for the multidimensional modeling of data warehouses', *Decision Support Systems*, Vol. 42, No. 12, pp.1270–1289.
8. Ghazali, M.T. (2022) 'Mobile app for COVID-19 patient education – development process using the analysis, design, development, implementation, and evaluation models', *Nonlinear Engineering*, Vol. 11, No. 1, pp.549–557, DOI: 10.1515/nleng-2022-0241.

10. Ghozali, M.T. and Abdissalam, E. (2020) 'The evaluation of clinical pharmacy services performance at community health centers of Sebatik Island regency of Nunukan province of north Kalimantan (Indonesia-Malaysia Border)', *Research Journal of Pharmacy and Technology*, Vol. 13, No. 7, p.3187, DOI: 10.5958/0974-360x.2020.00564.8.
11. Ghozali, M.T., Amalia Islamy, I.D. and Hidayaturrohim, B. (2022a) 'Effectiveness of an educational mobile-app intervention in improving the knowledge of COVID-19 preventive measures', *Informatics in Medicine Unlocked*, Vol. 34, No. 101112, p.101112, DOI: 10.1016/j.imu.2022.101112.
12. Ghozali, M.T., Satibi, S., Ikawati, Z. and Lazuardi, L. (2022b) 'The efficient use of smartphone apps to improve the level of asthma knowledge', *Journal of Medicine and Life*, Vol. 15, No. 5, pp.625–630, DOI: 10.25122/jml-2021-0367.
13. Gu, H. 'DIAVA: A Traffic-Based Framework for Detection of SQL Injection Attacks and Vulnerability Analysis of Leaked Data.' *IEEE Transactions on Reliability*, vol. 69, no. 1, pp. 188-202, 2020, doi: 10.1109/tr.2019.2925415.
14. Hayati, P., Jafari, N., Rezaei, S.M., Sarenche, S. and Potdar, V. (2008) 'Modeling input validation in UML', *19th IEEE Australian Conference on Software Engineering*, IEEE, Perlin, Australia, 25–28 March, pp.663–672.
15. Hochreiner, C., Frühwirth, P., Ma, Z., Kieseberg, P., Schrittwieser, S. and Weippl, E.R. (2015) 'Genie in a model? Why model-driven security will not secure your web application', *J. Wirel. Mob. Networks Ubiquitous Comput. Dependable Appl.*, Vol. 5, No. 3, pp.44–62.
16. Inukollu, V.N. and Urban, J.E. (2020) 'Secure descartes: a security extension to descartes specification language', *International Journal of Software Engineering and Applications*, Vol. 11, No. 5, pp.1–11, DOI: 10.5121/ijsea.2020.11501.
17. Jürjens, J. (2002) 'UMLsec: extending UML for secure systems development', *International Conference on the Unified Modeling Language*, Springer, Berlin, Heidelberg, 30 September– 4 October, pp.412–422.
18. Jürjens, J. and Shabalin, P. (2005) 'Tools for secure systems development with UML: security analysis with ATPs', in *Fundamental Approaches to Software Engineering*, pp.305–309, Springer, Berlin, Heidelberg.
19. Kasal, K., Heurix, J. and Neubauer, T. (2011) 'Model-driven development meets security: an evaluation of current approaches', *44th Hawaii International Conference on System Sciences*, IEEE, Koloa, Kauai, Hawaii, USA, 4–7 January, pp.1–9.
20. Khaled, A. (2021) 'IoT for dynamic information systems: concepts and principles', *International Journal of Business Information Systems*, Vol. 36, No. 4, pp.527–552.
21. Khan, F.A., Abubakar, A., Mahmoud, M., Al-Khasawneh, M.A. and Alarood, A.A. (2019) 'Cotton crop cultivation oriented semantic framework based on IoT smart farming application', *International Journal of Engineering and Advanced Technology*, Vol. 8, No. 3, pp.480–484.
22. Khan, M.U.A. and Zulkernain, M. (2009) *A Survey on Requirements and Design Methods for Secure Software Development*, Technical Report 2009–562, School of Computing, Queen's University, Kingston, Ontario, Canada.
23. Khwaja, A.A. (2009) *RealSpec: An Executable Real-Time Specification Language*, Arizona State University, USA.
25. Khwaja, A.A. (2015) 'Modeling big data analytics with a real-time executable specification language', *Handbook of Research on Trends and Future Directions in Big Data and Web Intelligence*, pp.289–312, IGI, USA.
26. Khwaja, A.A. and Urban, J.E. (2002) 'A synthesis of evaluation criteria for software specifications and specification techniques', *International Journal of Software Engineering and Knowledge Engineering*, Vol. 12, No. 5, pp.581–599.
27. Khwaja, A.A., Murtaza, M. and Ahmad, H.F. (2020) 'A security feature framework for programming languages to minimise application layer vulnerabilities', *Security and Privacy*, Vol. 3, No. 1, pp.1–30.
28. Li, H., Shang, W. and Hassan A E, (2018) 'Which log level should developers choose for a new statement?', *25th IEEE International Conference on Software Analysis, Evolution, and Reengineering (SANER)*, Campobasso, Italy, IEEE, 16 October, pp.1684–1716.
29. Li, Z., Dawood, S.R.S. and Zhang, X. (2017) 'An appraisal of Asia-Pacific cities as control and command centres embedded in world city network', *Engineering, Technology & Applied Science Research*, Vol. 7, No. 4, pp.1879–1882.
30. Lucio, L., Zhang, Q., Nguyen, P.H., Amrani, M., Klein, J., Vangheluwe, H. and Le, T.Y. (2014) 'Advances in model-driven security', *Advances in Computers*, Vol. 93, No. 2, pp.103–152.
31. Mast, N., Khan, M.A., Uddin, M.I., Ali Shah, S.A., Khan, A., Al-Khasawneh, M.A. and Mahmoud,
32. M. (2021) 'Channel contention-based routing protocol for wireless ad hoc networks', *Complexity*, Vol. 2021, No. 1, pp.1–10.
33. Memon, M., Hafner, M. and Breu, R. (2008) 'SECTISSIMO: a platform-independent framework for security services', *Proceeding of the Workshop on Modeling Security (MODSEC@ MoDELS)*, Toulouse, France, ceur-ws.org, 28 May, pp.1–10.
34. Naqvi, S., Massonet, P. and Arenas, A. (2006) *A Study of Languages for the Specification of Grid Security Policies*, Technical Report TR-0037, Institute on Knowledge and Data Management.
35. Nikiforakis, N. Meert, W. Younan, Y., Johns, M. and Joosen, W. 'SessionShield: Lightweight Protection against Session Hijacking.' *Engineering Secure Software and Systems Third International Symposium*, pp. 87-100, 2011, doi: 10.1007/978-3-642-19125-1_7.
36. Nguyen, P.H., Kramer, M., Klein, J. and Le, T.Y. (2015) 'An extensive systematic review on the model-driven development of secure systems', *Information and Software Technology*, Vol. 68, No. 12, pp.62–81.
37. Ocoró, M.P., Polo, O.C.C. and Khandare, S. (2023) 'Importance of business financial risk analysis in SMEs according to COVID-19', *FMDB Transactions on Sustainable Management Letters*, Vol. 1, No. 1, pp.12–21.
38. Pham B.A. and Subburaj, V.H. 'An experimental setup for detecting sql attacks using machine learning algorithms', *Journal of The Colloquium for Information Systems Security Education*, vol. 8, 2020.
39. Priscila, S.S., Rajest, S.S., Tadiboina, S.N., Regin, R. and Andrés, S. (2023) 'Analysis of machine learning and deep learning methods for superstore sales prediction', *FMDB Transactions on Sustainable Computer Letters*, Vol. 1, No. 1, pp.1–11.

40. Rafe, V. and Hosseinpouri, R. (2015) 'A security framework for developing service-oriented software architectures: secure SOA', *Security and Communication Networks*, Vol. 8, No. 17, pp.2957–2972, DOI: 10.1002/sec.1222.
41. Rajendrakumar, R. (2022) 'The prediction of online shopping consumers' attitude based on certain statements of consumer and marketing factors with reference to post graduate students in Coimbatore City', *International Journal of Business Information Systems*, Vol. 39, No. 4, pp.472–496.
42. Rani, R., Kumar, S., Kaiwartya, O., Khasawneh, A.M., Lloret, J., Al-Khasawneh, M.A., Mahmoud, M. and Alarood, A.A. (2021) 'Towards green computing oriented security: a lightweight postquantum signature for IoT', *Sensors (Basel, Switzerland)*, Vol. 21, No. 5, p.1883.
43. Rossini, A., Rutle, A., Mughal, K.A., Lamo, Y. and Wolter, U. (2011) 'A formal approach to data validation constraints in MDE', *International Workshop on Harnessing Theories for Tool Support in Software*, Vol. 26, pp.65–76.
44. Roudier, Y. and Aprville, L. (2015) 'SysML-Sec: a model-driven approach for designing safe and secure systems', *2015 3rd International Conference on Model-Driven Engineering and Software Development (MODELSWARD)*, IEEE, pp.655–664.
45. Saleh, M.A., Hajar Othman, S., Al-Dhaqm, A. and Al-Khasawneh, M.A. (2021) 'Common investigation process model for internet of things forensics', *2021 2nd International Conference on Smart Computing and Electronic Enterprise (ICSCEE)*, IEEE, Malaysia.
46. Sharma, D.K., Lohana, S., Arora, S., Dixit, A., Tiwari, M. and Tiwari, T. (2021) 'E-commerce product comparison portal for classification of customer data based on data mining', *Materials Today: Proceedings*, South Africa, <https://doi.org/10.1016/j.matpr.2021.05.068>.
47. Sharma, P.K. (2020) 'Some common fixed point theorems for a sequence of self-mappings in fuzzy metric space with property (CLRg)', *Journal of Mathematical and Computational Science*, Vol. 10, No. 5, pp.1499–1509.
48. Sharma, P.K. and Sharma, S. (2021a) 'Prevalent fixed point theorems on MIFM-Spaces using the (CLR_SR) property and implicit function', *Journal of Mathematics and Computer Science*, Vol. 25, No. 4, pp.341–350.
49. Sharma, P.K. and Sharma, S. (2021b) 'On common alpha fixed point theorems', *Journal of Mathematical and Computational Science*, Vol. 11, No. 1, pp.87–108.
50. Sharma, S. and Sharma, P.K. (2023) 'Stability analysis of an SIR model with alert class modified saturated incidence rate and Holling functional type-II treatment', *Computational and Mathematical Biophysics*, Vol. 11, No. 1, pp.1–10.
51. Sommestad, T., Ekstedt, M. and Holm, H. (2012) 'The cyber security modeling language: a tool for assessing the vulnerability of enterprise system architectures', *IEEE Systems Journal*, Vol. 7, No. 3, pp.363–373.
52. Srinivas, K., Velmurugan, P.R. and Andiyappillai, N. (2023) 'Digital human resources and management support improve human resources effectiveness', *FMDDB Transactions on Sustainable Management Letters*, Vol. 1, No. 1, pp.32–45.
53. Tripathi, S. and Al-Shahri, M. (2023) 'Problems and prospects on the evolution of advertising and public relations industries in Oman', *FMDDB Transactions on Sustainable Management Letters*, Vol. 1, No. 1, pp.1–11.
54. Uddin, M.I., Ali Shah, S.A., Al-Khasawneh, M.A., Alarood, A.A. and Alsolami, E. (2022) 'Optimal policy learning for COVID-19 prevention using reinforcement learning', *Journal of Information Science*, Vol. 48, No. 3, pp.336–348.
55. Ullah, Z., Zeb, A., Ullah, I., Awan, K.M., Saeed, Y., Uddin, M.I., Al-Khasawneh, M.A., Mahmoud, M. and Zareci, M. (2020) 'Certificateless proxy reencryption scheme (CPRES) based on hyperelliptic curve for access control in content-centric network (CCN)', *Mobile Information Systems*, Vol. 2020, No. 7, pp.1–13.
57. Van den Berghe, A., Scandariato, R., Yskout, K. and Joosen, W. (2017) 'Design notations for secure software: a systematic literature review', *Software & Systems Modeling*, Vol. 16, No. 3, pp.809–831.
58. Vashishtha, E. and Dhawan, G. (2023) 'Comparison of Baldrige criteria of strategy planning and Harrison text', *FMDDB Transactions on Sustainable Management Letters*, Vol. 1, No. 1, pp.22–31.
59. Vashishtha, E. and Kapoor, H. (2023) 'Implementation of blockchain technology across international healthcare markets', *FMDDB Transactions on Sustainable Technology Letters*, Vol. 1, No. 1, pp.1–12.
60. Wang, P., Lu, K., Li, G. and Zhou, X. 'DFTracker: detecting double-fetch bugs by multi-taint parallel tracking.' *Frontiers of Computer Science*, vol. 13, no. 2, pp. 247-263, 2019, doi: 10.1007/s11704-016-6383-8.
61. Zanjani, S.M., Barzoki, A.S., Kazemi, A. and Shahin, A. (2020) 'Proposing a technical human resource competence model in cyberspace media: a qualitative approach', *International Journal of Business Information Systems*, Vol. 33, No. 4, p.429, DOI: 10.1504/ijbis.2020.10027508.
62. Zeyun, L. and Dawood, S.R.S. (2016) 'Development of the regional financial centers in China: a quantitative study based on the province-level data', *Mediterranean Journal of Social Sciences*, Vol. 7, No. 3, S1, p.374.
63. Zuo, Y. (2023) 'Big data and big risk: a four-factor framework for big data security and privacy', *International Journal of Business Information Systems*, Vol. 42, No. 2, pp.224–242.
64. Zhu, D., Jung, J., Song, D., Kohno, T. and Wetherall, D. 'TaintEraser.' *ACM SIGOPS Operating Systems Review*, vol. 45, no. 1, pp. 142-154, 2011, doi: 10.1145/1945023.1945039.
65. OWASP Secure Coding Practices, 'Quick Reference Guide.' https://owasp.org/www-pdfarchive/OWASP_SCP_Quick_Reference_Guide_v2.pdf (accessed: Aug. 05, 2024).