

Proposed Secure Activity Diagram for Software Development

Madhuri N. Gedam¹, Bandu B. Meshram²

Research Scholar, Dept. of Computer Engineering, Veermata Jijabai Technological Institute (VJTI), Mumbai, India¹
Professor, Dept. of Computer Engineering, Veermata Jijabai Technological Institute (VJTI), Mumbai, India²

Abstract—Unified Modeling Language (UML) activity diagrams are derived from use case diagrams. It becomes essential to incorporate security features and maintain consistency in the diagrams during analysis phase of Software Development Life Cycle (SDLC). As part of current software development practices, software security must be a constant effort. The activity diagrams are used to model business process. The detailed analysis of activity diagram is done. The challenge lies in viewing the main activity diagram from attacker's perspective and providing defense mechanism to mitigate the attacks. This paper presents an extension of the activity diagram named SecUML3Activity to provide security with Object Constraint Language (OCL) constraints using Five Primary Security Input Validation Attributes (FPSIVA) parameters for input validation. It also proposed three security color code notations and stereotypes in activity diagrams. White color is used to represent activity diagram in normal state. Red color in dotted line is used to represent attack activity components. Blue color with double line is used to represent the defensive activity components. The defense mechanism algorithm against SQL Injection (SQLI) attack, Cross Site Scripting (XSS) attack, DoS/DDoS attack, access validation attack is provided. The mapping of Secure 3-Use Case diagram with SecUML3Activity diagram is done through mathematical modeling.

Keywords—Unified modeling language; activity diagram; object constraint language; SQL injection; use case diagram

I. INTRODUCTION

Unified modeling language (UML) is a visual language rather than a programming language to help software developers. It is used to build real-time systems and shows visual representation of the behavior and structure of the system in software development [3]. UML modeling can be done with the help of tools like StarUML, Microsoft Visio, ArgoUML, MagicDraw, BOUML, Visual Paradigm and the like [24]. UML or Object Constraint Language (OCL) is used in designing financial systems where incorporation of security is a primary concern.

Activity diagram is used to show the diagrammatic flow of events taking place in a use case diagram. It shows the dynamic behavior of a system like control flow and object flow from one action to another which is one of the main UML modeling techniques [1][4]. It is used to model security requirements in the business processes, modeling parallel and concurrent flows in an actual system and illustrate the scenario of detailing complex use cases [2][7][25].

In earlier work, Colored Petri Net (CPN) has been proposed to ensure consistency between use cases and activity diagrams [26][27][28]. Jurjens proposed UMLsec to specify security information during the development of security critical systems and provided tool-support for formal security verification using security scenarios into a system design [33]. UMLsec employs use case diagrams to capture security requirements. UMLsec defines 21 stereotypes to represent fair exchange, non-repudiation, role-based access control, secure communication link, confidentiality, integrity, authenticity, freshness of a message, secure information flow among components, and guarded access. Some stereotypes also have associated tags and constraints.

The foundation of secure SRS is consideration of security requirements to mitigate severe vulnerabilities mentioned in the vulnerability databases [15]. Secure SRS considers security requirements like input validation, multi-factor authentication to enhance UML use case, class and state transition diagrams [4][5]. In this paper, we proposed security stereotypes, colored notations to distinguish main activity diagram from attackers' activity diagram and defensive activity diagram. FPSIVA parameters based on OCL constraints are used to provide defense mechanisms in activity diagram and mitigate vulnerability in the analysis phase of SDLC. These stereotypes help developers to build functionalities carefully and flawlessly during the implementation process. Also, defense mechanism algorithms are proposed in this research work to build secure activity diagrams. The consistency between UML diagrams is maintained through relationship between proposed SecUML3Activity diagram and Secure 3-Use Case diagram proposed by authors in earlier work [2].

The paper is organized as follows. Section II describes a detailed literature survey of activity diagram, notations to draw activity diagram, relationship of activity diagram with use case diagram. Section III covers the proposed secure activity diagram: SecUML3Activity with security color notations and stereotypes using FPSIVA parameters, and defense mechanism algorithm. Section IV is used for result and discussion related to this work. Section V concludes the paper and gives direction to the future work.

II. LITERATURE SURVEY

UML diagrams are used to visualize various perspectives of the software system. Since they are dependent on each other, the consistency between the diagrams is desired in earlier phases of SDLC. In comparison to static modeling, consistency is a more delicate issue in dynamic modeling. Non-compliance

of consistency among these diagrams lead to errors being introduced during software development and make it vulnerable to attacks like SQLI, XSS, DoS/ DDoS attack and access validation attack [12][13]. Relational Language for Advanced Security (ReAlSec) is a security engineering tool to find security threats [31]. A specification cannot be fully represented by a UML diagram on its own. Consequently, the dynamic diagrams would require a common notation among them. The external behavior of the systems to be built is meant to be expressed using use case diagrams and activity diagrams. Activity diagrams are used to show the dynamic behavior aspect of a given system by modeling data flow [1][2][18].

The Object Constraint Language (OCL) is a declarative language and forms part of the UML standard and plays a crucial role in the analysis phase of SDLC. It is an expression language used to describe constraints and other modeling artifacts that cannot be stated using conventional grammatical notations [4][28]. OCL constraint is acting as a restriction on a model to ensure consistency. Although it is designed at the class level, its semantics are applied at the object level [1][3][9][10][29]. The security of activity diagrams can be enhanced using OCL [2][4].

Activity diagrams are basically used to represent flow of events used in use case diagrams, modeling complex requirements and implementation details [11][26]. These diagrams look like data flow diagrams (DFDs) in structured analysis (SA), However, DFDs in SA are used for capturing, analyzing, and documenting requirements. They are best suited for modeling parallel and concurrent flows in an actual system. The activity can be explained as an operation of the system [6][7]. Due to the richer constructs, it offers, such as concurrency, split, and synchronization, the UML activity diagram has been utilized in process modeling and workflow modeling [20]. They have a significance in software testing [10][17][30]. They are divided into two kinds such as atomic activity diagram and compound activity diagram based on sub activity state. Managing the compound activity diagram is a significant problem when creating test cases [1].

A. Analysis of UML Activity Diagram

Some definitions of the activity diagram can be presented in a formal manner.

1) Activity diagram (AD) is a tuple consisting of –

$$AD = (N, E, C, R)$$

where N, E, C, R are a finite set of activity nodes, directed edges, containment and flow relationship between the nodes or containments respectively.

Activity nodes consist of action nodes N_a , object nodes N_o and control nodes N_c .

$$N = N_a \cup N_o \cup N_c$$

Directed edges are a finite set of edges.

$$E = \{e_1, e_2, e_3, e_n\}$$

C contains graphical elements for containment and it is formally defined as a tuple consisting of activities, interruptible regions, exception handlers, expansion regions.

$$C = (Activities, IR, EH, ER)$$

The flow relationship R is explained as follows.

$$R \subseteq (N \vee C) \times E \times (N \vee C)$$

The control node consists of given disjoint sets as below.

$$N_c = I \cup D \cup M \cup P \cup J \cup F$$

where I, D, M, P, J and F are finite sets of initial nodes, decision/branch, merge, forks, joins and final nodes that cover activity final and flow final nodes. So, F can be denoted as $F = F_a \cup F_f$, where F_a is a finite set of activity final nodes and F_f is a finite set of flow final nodes. And F are finite sets of initial nodes, decision/branch, merge, forks, joins and final nodes that cover activity final and flow final nodes [19].

2) Activity diagram (AD) is a tuple consisting of –

$$D = (A, T, F, C, aI, aF)$$

where A, T, F, C, aI, aF are a finite set of activity states, completion transitions, guard conditions, flow relationship, initial activity state and final activity state respectively and described as below.

$$A = \{a_1, a_2, \dots, a_m\}$$

$$T = \{t_1, t_2, \dots, t_n\}$$

$$C = \{c_1, c_2, \dots, c_n\}$$

$$F \subseteq (A \times T \times C) \cup (T \times C \times A)$$

$$aI \in A$$

$$aF \in A$$

There is only one transition t such that $(aI, t, a) \in F$, and $(a, t', aI) \notin F$ or $(aF, t', a) \notin F$ for any t', a . The activity diagram is used to represent composite activities. Each activity node is handled individually and treats concurrent activities as an interleaving sequence of activities [17][18].

3) Since every use case *useCase* gets converted to activity diagram, the complete set of all activity diagrams AD contains many $ad_{useCase}$.

$$ad_{useCase} \in AD$$

As each activity diagram consists of initial node, activity nodes and activity partitions,

$$AD = \{IN, AN, AP\}$$

where, IN denotes the initial node. Every activity diagram must have an initial node.

$$IN_{useCase} \in IN$$

AN denotes the activity node. There may be zero or more activity nodes in an activity diagram.

$$AN_{useCase} \in AN$$












AP denotes the activity partitions. There may be zero or more activity partitions in an activity diagram.

$$AP_{useCase} \in AP$$

B. Activity Diagram Notations

The graphical notations are used for modeling activity diagrams including nodes and edges. The diagrams must ensure their semantics to conform to the UML activities metamodel [6][8]. The activity diagram notations are shown in Table I.

TABLE I. ACTIVITY DIAGRAM NOTATIONS

Element	Symbol	Description
Start		Start symbol in activity diagrams is used to indicate the start of a process or workflow.
Activity		It outlines the tasks that participate in a modeled process. It serves as the foundation of an activity diagram.
Connector		It indicates the directional flow or control flow of the activity. After a step in an activity is complete, the flow is continued by an outgoing arrow. A step in an activity is initiated by an incoming arrow.
Joint/Synchronization bar		Two ongoing tasks get combined and reintroduce them to a flow in which only one task is carried out at once.
Fork		Two concurrent operations are split from one main flow of activity.
Decision		Minimum two paths diverge at a decision and users get to view options. This symbol indicates the branching or merging of various flows.
Send signal		It conveys to a receiving activity that a signal is being sent.
Receive signal		It shows that an event has been accepted. Flow that comes from this action is completed once the event is received.
Option loop		It gives the designer the ability to depict a repeating sequence inside the loop symbol.
Flow final		It denotes the end of a particular process flow. The end of a process should be done with a flow final symbol.
Condition text	[Condition]	The developer comes to know under what condition an activity flow should split off in that direction.
End		It denotes the finish of an activity and the end of all process flows.

C. Relationship of Activity Diagram with use Case Diagram

A systematic mapping of activity diagram with use case Diagram is described below [22][23][26][27][28].

Rule 1: Every use case must be represented by at least one activity diagram, else there will be inconsistency leading to fault in software development.

$$\exists useCase \in UseCase_{sysModel}: \exists AD_{useCase} AD_{sysModel}$$

Rule 2: An actor in a use case must be an activity partition in the corresponding activity diagram.

$$\exists actor, (assoc(actor, UseCase), \exists ap \in AP_{useCase})$$

Rule 3: Let use case diagrams UC1 includes UC2 where UC1 is the including use case and UC2 is included use case. Then event flows of both UC1 and UC2 must be specified in

the activity diagram. The action node in UC1 should refer to the activity diagram specifying use case UC2.

$$include = (including, included) \in Include$$

$$where\ including, included \in UseCase : act_{included} \in ACT_{including}$$

Rule 4: Every flow of event mentioned in the use case description or implied therein needs to be described in detail in the related event of the activity diagram. This rule is only applicable if the use case is further described in the activity diagram.

Rule 5: The event in the use case diagram has a one-to-one mapping with an action/activity state in the corresponding activity diagram.

D. Object Constraint Language (OCL)

OCL is a formal specification language that can be used to define expressions and constraints on object-oriented models and other object modeling artifacts. IBM created the Object Constraint Language in 1995. It was initially used as a business engineering language, but it was later incorporated into the Unified Modelling Language (UML) as a formal specification language. Starting with version 1.1, OCL was included in the official OMG (Object Management Group) standard for UML. It enables programmers to communicate restrictions and guidelines that control the organization and operation of software systems. OCL 2.0 is the latest version as of September 2021. OCL is a powerful language with built-in capabilities for iterating over collections of objects, finding the value of an item, and navigating across a group of related objects. Primitive types such as Integer, Real, Boolean, and String, as well as Collections types such as Set, Bag, ordered set, and Sequence, are included in OCL's predefined standard library [14]. OCL can be used in many ways. For any expression over a UML model, it can be used as a query language to specify invariants on classes and types in the class model, type invariants for stereotypes, pre- and post-conditions on operations and methods, guards, target (sets) for messages and actions, constraints on operations, and derivation rules for attributes [3][4]. As each OCL expression has a type, it is considered as a typed language [4].

An activity diagram becomes more comprehensible when it is modeled using UML notations. To non-technical individuals, such as a client, the pictorial depiction makes knowledge transfer simple. However, there can be certain discrepancies in the diagrams if a programmer uses them as a reference when building implementation code. For instance, it's possible that the diagram doesn't show the beginning values for some characteristics or doesn't clearly indicate the limitations. In these circumstances, it is impossible for the programmer to develop the entire program without consulting the required specification or other documentation. OCL aids in the improvement of the UML diagrams and, as a result, writes the complete code for the same [4].

1) **INVARIANT**: It is a constraint that specifies a condition that must always hold true for a particular class or a set of objects. Invariants are used to define the integrity rules of a system and ensure the consistency of the data.

Example-
context Person

inv: self.age > 0 and self.age < 120

A "Person" class that has an invariant specified on it. According to the invariant, a person's age must be more than 0 and less than 120. By doing this, it is ensured that a person's age is within a suitable range and that inaccurate or unrealistic figures are avoided.

Invariants are typically expressed in the context of a class and use the keyword "context" followed by the class name. The "self" keyword refers to the instance of the class on which the invariant is being evaluated. In this case, "self.age" points to the age of the Person object.

2) **PRE-CONDITIONS:** In OCL, preconditions and postconditions are used to define the conditions that must hold true before and after an operation or method is executed, respectively. They help define the expected behavior and constraints associated with an operation.

Syntax

context <classifier>: <operation> (<parameters>)

Pre [<constraints name>]:

<Boolean OCL expression>

The examples of a precondition in OCL is as below.

Let's consider a class called "BankAccount" with a method "withdraw" that deducts a specified amount from the account balance. The precondition for this method could be that the withdrawal amount should be positive and not exceed the current balance.

context BankAccount :: withdraw(amount: Integer)

pre: amount > 0 and amount <= self.balance

In this example, the precondition specifies that the "amount" parameter passed to the "withdraw" method should be greater than 0 and less than or equal to the current balance of the bank account. This ensures that a valid withdrawal amount is provided and prevents overdrawing from the account.

3) **POST-CONDITIONS:** Preconditions and postconditions are used to document and enforce the expected behavior of operations. They help in validating inputs and ensuring the desired outcomes or effects of operations on objects or systems.

Syntax

Context <classifier> :: <operation> (<parameters>)

Post [<constraints name >]:

<Boolean OCL expression>

The examples of a postcondition in OCL is-

Let's consider the same "BankAccount" class with a method "deposit" that adds a specified amount to the account balance. The postcondition for this method could be that the account balance should increase by the deposited amount.

context BankAccount::deposit(amount: Integer)

post: self.balance = self.balance@pre + amount

In this example, the postcondition specifies that the "balance" property of the bank account after executing the "deposit" method should be equal to the balance before the method was called plus the deposited amount. This ensures that the deposit operation updates the account balance correctly [14].

III. PROPOSED SECURE ACTIVITY DIAGRAM: SECUML3ACTIVITY

In this proposed SecUML3Activity diagram, dynamic aspects of the system are shown with security stereotypes in color code notations, OCL constraints and defense mechanism algorithms. An illustration of a dynamic security specification is the operation of an authentication mechanism. There is not a comprehensive design-level behavioral definition of security stereotypes in any of the dynamic security standards that developers and programmers might employ during the implementation stage. In this paper, we are proposing security features for SecUML3Activity diagram which is an extension of detailed analysis of Login Use Case of Secure 3-Use Case diagram proposed by authors [4].

A. Proposed Security Notations and Stereotypes in Activity Diagrams

It is easier to understand an activity diagram when it is modeled using UML notations. Information can be easily communicated to non-technical staff members, such as a client by way of a picture. However, there might be certain gaps in the UML diagrams when a programmer uses them to write implementation code. For instance, it is possible that the diagram doesn't show the initial values for certain attributes or doesn't clearly define the constraints. Writing the entire code without consulting the requirement specification or other documentation becomes challenging for the programmer. OCL plays an important role to clarify the UML diagrams, and accordingly write the complete code for the same. Since activity diagram is a behavior model, the relationship between model elements is usually more complex. Software engineering research encourages systematic literature review for identifying, evaluating, and interpreting research question [32]. The use of colors has been recognized in software engineering research to make software modeling more comprehensible. The proposed activity diagram notations are represented in various colors codes along with color description as mentioned in Table II.

The proposed colored notations are helpful in visual representation and reduce the cognitive load of software developers. The white color is used to represent normal activity diagram notations; red color in dotted line represents attacks performed by external entities. The double lined blue color is used to represent the attack mitigation and providing defense mechanism. These colored notations for SecUML3Activity

diagrams are mentioned in Table III for attack and defensive activity.

TABLE II. COLOR NOTATIONS DESCRIPTION

Color	Description
White	White represent component is in normal state.
Red	Red color is used to represent/highlight insecure or threatened components. These components are more likely to get attacked successfully by outside entities.
Blue	Blue is to represent the defensive or precautionary components. These components act as defensive measures to avoid or mitigate attack.

TABLE III. PROPOSED NOTATIONS FOR SECUML3ACTIVITY DIAGRAM

Symbol	Activity Diagram Notations	Attack Notations	Defense Notations
Start			
Activity			
Connector			
Joint/Synchronization bar			
Fork			
Decision			
Send signal			
Receive signal			
End			

The stereotypes proposed by authors are used to develop secure implementation. The developer can prevent attacks like Buffer Overflow (BOF), SQL Injection (SQLI), Encryption, Session Expiration, Connection flooding for login into the system.

Stereotype: << BufferOverflow >>

Tag: {BOF}

Stereotype: <<Encryption>>

Tag: {Encryptfield}

Stereotype: <<SQLi>>

Tag: { SQLfield }

If the logged in user remains idle for more than specified time, the session must be forcibly killed to prevent session expiration attacks using

Stereotype: <<SessionExpiry>>

Tag: {Exp_Time}

If there is a vulnerability in an application to allow more connections than the service provider supports, the stereotype must be inserted in the diagram part that represents the maximum number of allowed connections.

Stereotype: <<maxconn>>

Tag: {Maxconn}

B. Proposed Secure Constraints in SecUML3Activity Diagram

The OCL constraint is proposed to check the length, any special characters in entered username and password in the login page with the help of constraints. The foundation for applying Five Primary Security Input Validation Attributes (FPSIVA) in the web design phase is OCL [16][21]. It defines FPSIVA which can be used to design activity diagrams in software development. The below mentioned stereotypes used in activity diagrams are designed using FPSIVA parameters.

<<Precondition >>

Context Login :: checkCredentials()

Pre: user name <=12

Pre: Pwd >=8

<<Invariant >>

Context Login :: checkCredentials()

user name =Boolean

Pwd=Boolean

<<Invariant >>

Context Login :: checkCredentials()

Is active=Boolean

It must be ensured that post condition invariants to be applied after login entry to homepage as mentioned below –

<<Postcondition>>

Context Home :: checkAllowUser()

Post: valid User=Boolean

Post: Pwd=Ecrypt(password)

The password needs to be encrypted for transferring over the communication network. Number of attempts by malicious user can be detected with the help of following constraint -

Context Login Invariant :

No. of attempt : self. User > 5

FPSIVA parameters can be used for input validation in activity diagram such as -

(i) var.type : < type > - It is used to validate the type of input data and verify if it can be accepted < type >.

user name: string

(ii) var.format : < pattern > - It is used to validate the format of input data and verify if it can be accepted < pattern >.

(iii) var.length : < number > - It is used to validate the length of input data.

user name.length : 12, Pwd.length :8

(iv) var.Charset: < pattern > - It is used to check characters with its < pattern >

user name.charset : [A-Z, a-z, 0-9].

(v) var.value : < reasonableness > - It is used to check reasonable values of input data.

No of attempts.value:5.

C. SecUML3ActivityDesign

The proposed SecUML3Activity diagram for Login use case of College Management System (CMS) is divided into three swim lanes like Login Activity, Attack Activity, and Defense Activity. The complete flow of the system is shown by the Activity diagram. The swimlane of the activity diagram will be mapped with 2 swim lanes. The first swimlane will be simulated for attack. Each activity with a dotted line in red color notation and second swimlane will be simulated for providing defense mechanisms in blue color double line notations for the attacks. Due to space constraint, we have shown the Login activity of the case study. The proposed security color code notations, stereotypes and constraints are simulated with the login activity diagram in College Management case study as shown in Fig. 1. The login activity end element A is connected to start element of attack activity diagram. The end element of attack activity diagram B is connected to start element of defense activity diagram.

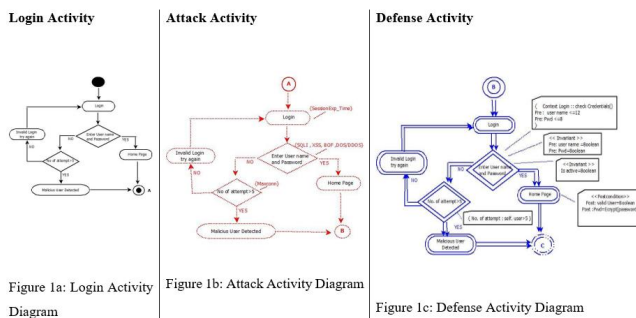


Fig. 1. Proposed SecUML3Activity diagram of Login CMS Use Case.

For clear visualization of the activity diagram, each activity as well as complete proposed SecUML3Activity diagram is shown in Appendix A (Fig. 2 to Fig. 5).

D. Proposed relationship between SecUML3 Activity and Secure 3 Use Case Diagram

Activity diagrams are basically behavioral representations of use case diagrams with the flow of events. The login use case proposed by the author at [2] is simulated in SecUML3 Activity Diagram. The Secure 3-UseCase is already proposed with the Secure SRS model with CIA-AAA verification during authentication of user's login to the system. The security of use cases is enhanced by considering functional requirements, non-functional requirements, and quality attributes in Secure SRS model [2]. The notations, stereotypes and defense algorithms used in Secure 3-UseCase are inherited in SecUML3 Activity Diagram of College Management System (CMS) to mitigate the attacks in the real world.

Based on SecUML3Activity diagram for Login use case shown in Fig. 1, i.e.

$$Login_{CMS} \in Secure3UseCase_{CMS}$$

$$adLogin \in SecUML3AD_{CMS}$$

where Initial node is $IN_{Login} \in IN_{CMS}$

and Activity partition is $Faculty \in AP_{Login}$

There is consistency between Secure 3-UseCase diagram and SecUML3Activity Diagram through the relationship mentioned below.

$$\exists Login \in Secure3UseCase_{CMS}: \exists adLogin \in SecUML3AD_{CMS}$$

E. Proposed Defense Mechanism Algorithm

The following Defense Mechanism Algorithm against Web based attacks were defined.

1) *SQL injection*: SQLI attacks take place on software applications through different methods like Tautologies, Illegal/Logically Incorrect Queries, UNION Query, Piggy-Backed Queries, Timing Inference attack.

Incident \in {Web page Field Access, URL Header Access}

Algorithm 1: Defense Mechanism Algorithm against SQL Injection

INPUT: SQL Injection through text fields in the web page.

OUTPUT: A secure web page that is free from SQLi.

Start

Read text entered by user in text fields

Create insert parameterized queries instead of string concatenation

Create roles.

For each role,

```
{
  Assign a User
}
```

For each user,

```
{
  Grant appropriate permissions to accomplish Role Based Access Control
}
```

```
| }  
  
| If User has permission to perform action on Database  
| | {  
| |   Fire Query  
| | }  
| Else  
| | {  
| |   Drop user inserted malicious query.  
| |   Use escape Queries for user inputs to get rid of  
| |   special characters.  
| | }  
End
```

2) *Cross Site Scripting (XSS)*: XSS attack occurs when dynamic content that hasn't been checked for malicious content, proper validation makes entry into a web page field.

Incident \in {Web page Field Access, URL Header Access}

Algorithm 2: Defense Mechanism Algorithm against Cross Site Scripting

INPUT: Input field on web page, URL header access used for taking input

OUTPUT: External script is executed

Start

Insert <body onload=alert (Testing XSS')> into input field.

Submit input

```
| If alert is shown in web browser then  
| | {  
| |   Simple XSS is performed.  
| |   Web service is vulnerable to XSS attack.  
| | }  
| Else  
| | {  
| |   Web service is not vulnerable to XSS attack.  
| | }
```

End

3) *Check for DoS/DDoS attacks*: DoS/DDoS attacks are classified into different types like Ping of Death, TCP SYN Attack, ICMP Smurf, UDP Flood attack. TCP SYN Attacks arising due to bugs in operating system can be prevented using security patches. Intrusion Detection Systems (IDS) are helpful to identify and stop illegal intrusion into the systems. Firewalls can be placed into the network to block traffic coming from unknown IP. Routers can be used to limit network access and dropping suspected traffic using Access Control List (ACL).

Incident \in {URL Header Access}

Algorithm 3: Defense Mechanism Algorithm against DoS/DDoS attack

INPUT: DoS/ DDoS attack through URL header access of web page.

OUTPUT: A secure web page that is free from DoS/ DDoS attack.

Start

Read (User Inputs like Source IP address, Destination IP address, Payload)

Extract IP header

```
| If Source IP  $\in$  BlackIP List, then  
| | {  
| |   Drop Packet  
| | }  
| Else if Payloadsize > Payloadthreshold then  
| | {  
| |   Drop packet and add to BlackIP List  
| | }  
| End if
```

End

4) *Check for access validation*: Due to absence of centralized middleware in Web page, it becomes necessary to specify the address (URI) of the page and the transport protocol (HTTP). Hence, Access validation can be done with the help of secure key management like security tokens for secure authentication.

Incident \in {Web page Field Access, URL Header Access}

Algorithm 4: Defense Mechanism Algorithm to check for access validation

INPUT: request through text fields, URL header access in the web page.

OUTPUT: A secure web page that is free from mis-user access attack.

Start

Issue security tokens to WSC through Security Token Service
Bind the same security token with WSP

Validate security token for transaction

```
| If WSC Security Token  $\in$  WSP Security Token, then  
| | {  
| |   Allow payload for transaction  
| | }  
| Else  
| | {  
| |   Drop payload and cancel the transaction  
| | }
```

End

IV. RESULT AND DISCUSSION

Based on extensive literature survey, security with OCL constraints using Five Primary Security Input Validation Attributes (FPSIVA) parameters for input validation is provided. The web modeling of software applications consisting of various security-colored notations and stereotypes in secure activity diagrams is proposed to distinguish main activity diagram from attackers' activity diagram and defensive activity diagram. Also, defense mechanism algorithms are proposed to build secure activity diagrams. The consistency between UML diagram is maintained through relationship between proposed SecUML3Activity diagram and Secure 3-

Use Case diagram proposed by authors in earlier work. The various B-Tech and M-Tech software Projects are implemented using secure analysis.

Proposed SecUML3Activity diagram is derived from the Secure 3 Use Case diagram proposed by the author in their earlier work [2]. The proposed strategy is to maintain consistencies between these UML diagrams to avoid errors, defects, vulnerabilities that may arise in software development. This relationship between these two UML diagrams is well explained through mathematical modeling. The input validation of parameters is done through OCL constraints using Five Primary Security Input Validation Attributes (FPSIVA) parameters. The use of colors has been recognized by Software Engineering research to make graphical software models easier to follow, hence as per requirement of secure activity diagrams, three security color code notations and stereotypes in activity diagrams are proposed to distinguish the activities. White color is used to represent activity diagram in normal state. Red color in dotted line is used to represent attack activity components. Blue color with double line is used to represent the defensive activity components. The defense mechanism algorithms against SQL Injection (SQLI), Cross Site Scripting (XSS), DoS/ DDoS attack, access validation is also provided for making system more secure and robust.

V. CONCLUSION AND FUTURE WORK

The main purpose of this research is to provide security in activity diagrams to prevent external and internal attacks on the web application. The defects, errors, and problems in the software systems occur due to inconsistencies between UML diagrams in analysis phase.

The security features of SecUML3Activity diagram in analysis phase of SDLC can be mapped with component diagram of software architecture, secure data structure design and secure algorithms design against top 10 attacks on software. This standardized proposed secure UML stack with defense mechanism can be used as the reference document for the coding phase and help developers to build more secure applications. The work is in progress.

REFERENCES

- [1] M. Abbasa, R. Rioboob, C. Yellese, C. Snookd, "Formal Modeling and Verification of UML Activity Diagrams (UAD) with FoCaLiZe", Elsevier, pp. 1-27, September 2020.
- [2] M. Gedam, B. Meshram, "Proposed Secure 3-Use Case Diagram," International Journal of Systems and Software Security and Protection, IGI Global, pp. 1-18 2022.
- [3] S. Hayat, F. Toufik, M., "UML/OCL based design and the transition towards temporal object relational database with bitemporal data", Elsevier, pp. 1-10, August 2019.
- [4] E. Sunitha, P. Samuel, "Enhancing UML Activity Diagrams using OCL", 2013 IEEE International Conference on Computational Intelligence and Computing Research, pp. 1-6, IEEE, 2013.
- [5] M. Mohsin, M. Umair Khan, "UML-SR: A Novel Security Requirements Specification Language", 2019 IEEE 19th International Conference on Software Quality, Reliability and Security (QRS), pp. 342-349, IEEE, 2019.
- [6] Y. Abushark, T. Miller, J. Thangarajah, M. Winikoff, J. Harland, "Requirements specification via activity diagrams for agent-based systems", 31, 423-468 (2017). Springer, pp.1-46, February 2016.
- [7] A. Rodríguez, E. Fernández-Medina, J. Trujillo, M. Piattini, "Secure business process model specification through a UML 2.0 activity diagram profile", Decision Support Systems, Elsevier, pp. 446-465, 2011.
- [8] An Oracle White Paper, "Getting Started With Activity Modeling", Oracle Corporation, USA, pp.1-9, May 2007.
- [9] L. Tan, Z. Yang, J. Xie, "OCL Constraints Automatic Generation for UML Class Diagram", IEEE, pp. 392-395, 2010.
- [10] T. Ahmad, J. Iqbal, A. Ashraf, D. Truscan, I. Porres, "Model-based testing using UML activity diagrams: A systematic mapping Study" Computer Science Review Elsevier, pp. 1-15, July 2019.
- [11] Analysis and Design: The Making of Information Systems. Springer, Berlin, Heidelberg, pp. 235-351, 2008.
- [12] E. Germán, Rodríguez, J. Torres, P. Flores, D. E Benavides, "Cross-site scripting (XSS) attacks and mitigation: A survey", Computer Networks, Elsevier, pp.1-27, 2019.
- [13] I. Martínez, A. Campazas-Vega, A. Higuera, V. DelCastillo, C. Aparicio, C. Fernández-Llamas, "SQL injection attack detection in network flow data", Computers & Security, Elsevier, pp.1-11, 2023.
- [14] Object Constraint Language-OMG Document Number: formal/2014-02-03, pp.1-262.
- [15] M. Gedam, B. Meshram, "Vulnerabilities & Attacks in SRS for Object-Oriented Software Development," Lecture Notes in Engineering and Computer Science: Proceedings of The World Congress on Engineering and Computer Science 2019, 22-24, San Francisco, USA, pp.94-99, October, 2019.
- [16] M. Gedam, J. Varshapriya, B. Meshram, "Proposed Secure Content Modeling of Web Software Model," Proceedings of The National Conference on Recent Innovations In Engineering Science & Technology, pp. pp.13001-13005, April 2019.
- [17] W. Thanakorncharuwit, S. Kamonsantiroj, L. Pipanmaekaporn, "Generating Test Cases from UML Activity Diagram Based on Business Flow Constraints", ACM, pp. 155-160, December 2016.
- [18] L. Yu1, X. Tang, L. Wang1, L. Xuand, "Simulating Software Behavior based on UML Activity Diagram", Changsha, China, ACM, pp. 1-4, October 2013.
- [19] X. Dong, N. Philbert, Zongtian, Wei Liu, "Towards Formalizing UML Activity Diagrams in CSP", International Symposium on Computer Science and Computational Technology, IEEE, pp.1-4, 2008.
- [20] D. Yang, L. Tong, "Modeling E-government Administrative Processes Using Unified Modeling Language", 2006 IEEE International Conference on Service Operations and Logistics, and Informatics. IEEE, pp. 983-987, 2006.
- [21] P. Hayati, N. Jafari, S. Rezaei, S. Sarenche, "Modeling Input Validation in UML", 19th Australian Conference on Software Engineering, IEEE, pp. 663-672, 2008.
- [22] M. Alanazi, "Basic Rules to Build Correct UML Diagrams", International Conference on New Trends in Information and Service Science, IEEE, pp. 72-76, 2009.
- [23] D. Torre, Y. Labiche, M. Genero, M. Elaasar, "A systematic identification of consistency rules for UML diagrams", The Journal of Systems & Software, pp.1-29, 2018.
- [24] M. Ozkaya, F. Erata, "A Survey on the Practical Use of UML for Different Software Architecture Viewpoints". Information and Software Technology, Elsevier, pp.1-27, 2020.
- [25] M. Guilherme, Tatibana, F. Barreto V. Benitti, "Use case or activity diagram, that is the question!", ACM, pp. 1-7, 2019.
- [26] J. Chanda, A. Kanjilal, S. Sengupta, S. Bhattacharya. "Traceability of Requirements and Consistency Verification of UML UseCase, Activity and Class diagram: A Formal Approach", International Conference on Methods and Models in Computer Science (ICM2CS), pp. 1-4, 2009.
- [27] Y. Shinkawa, "Inter-Model Consistency in UML Based on CPN Formalism", 13th Asia Pacific Software Engineering Conference (APSEC '06), pp.414-418, 2006.
- [28] P. G. Sapna, H. Mohanty. "Ensuring Consistency in Relational Repository of UML Models", 10th International Conference on Information Technology (ICIT 2007), pp.217-222, 2007.

- [29] E. Fernandez-Medina, M. Piattini and M.A. Serrano, "Specification of security constraint in UML," Proceedings IEEE 35th Annual 2001 International Carnahan Conference on Security Technology , IEEE, pp 19 Oct. 2001.
- [30] M. Shirole, M. Kommuri, R. Kumar, "Transition sequence exploration of UML activity diagram using evolutionary algorithm. Proceedings of the 5th India Software Engineering, ACM, pp.97-100, 2012.
- [31] M. Hamdi , N Essaddi and N Boudriga," ReAISec: A Relational Language for Advanced Security Engineering," 2009 International Conference on Advanced Information Networking and Applications, Bradford, UK, 29 May 2009.
- [32] B. Kitchenham, "Guideline for Performing Systematic Literature Reviews in Software Engineering", EBSE Technical Report, pp.1-65, July 2007.
- [33] J. Jurjens, "UMLsec: Extending UML for Secure Systems Development", Lecture Notes in Computer Science, Springer, pp. 412–425, 2002.

APPENDIX-A

1) Login Activity Diagram

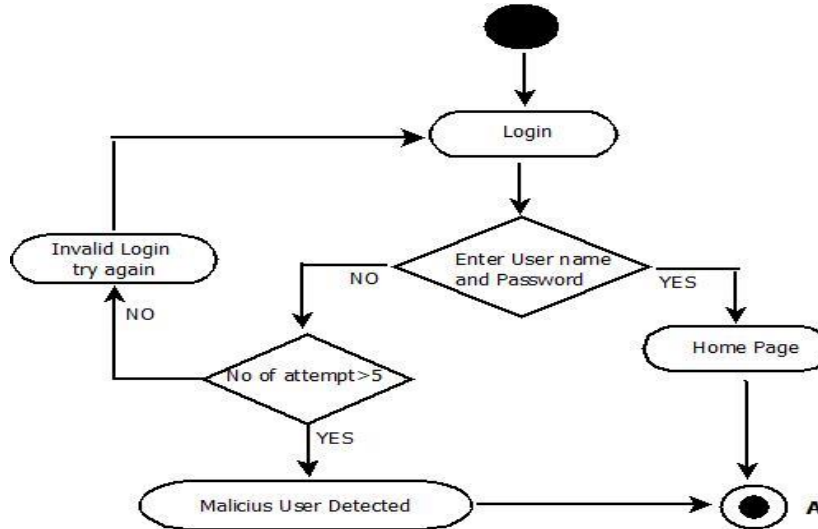


Fig. 2. Login activity.

2) Attack Activity Diagram

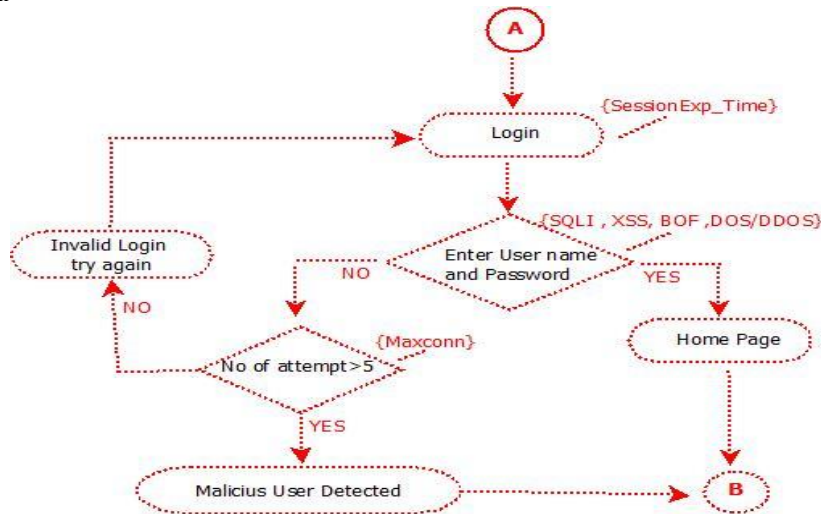


Fig. 3. Attack activity.

3) Defensive Activity Diagram

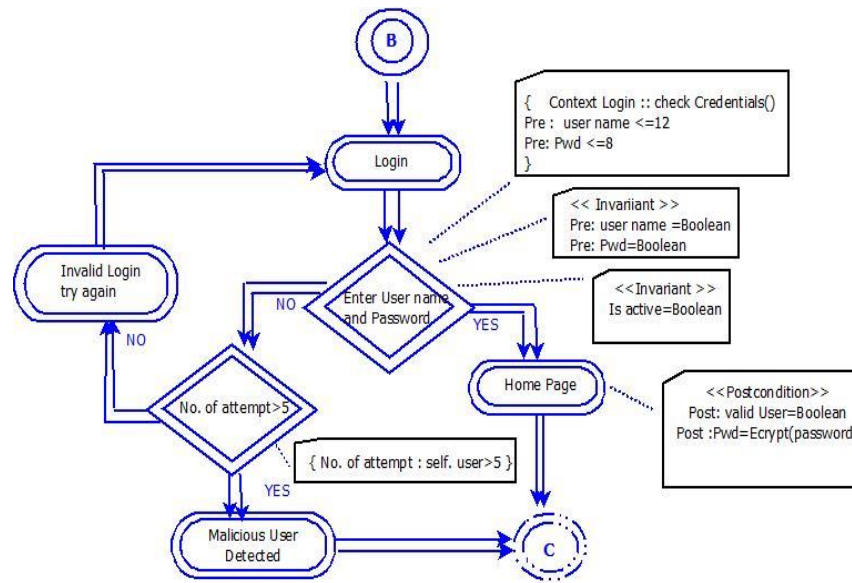


Fig. 4. Defensive activity.

4) Proposed SecUML3Activity Diagram

Login Activity

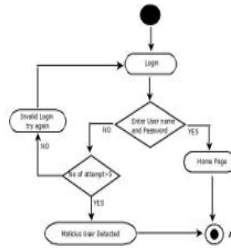


Figure 1a: Login Activity Diagram

Attack Activity

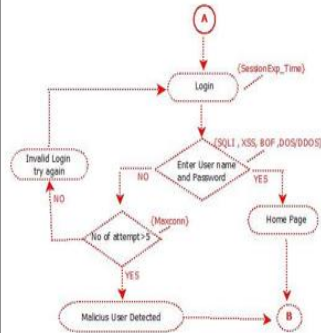


Figure 1b: Attack Activity Diagram

Defense Activity

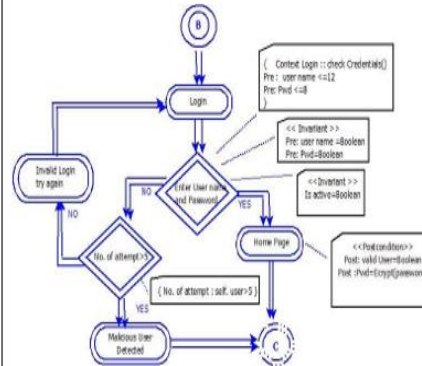


Figure 1c: Defense Activity Diagram

Fig. 5. Proposed SecUML3Activity.