Role of Coupling in Vulnerability Propagation

Object Oriented Design Perspective

A. Agrawal and R. A. Khan

Department of Information Technology,
BabasahebBhimraoAmbedkar University,
Vidyavihar, Raebarelli Road, Lucknow
alka_csjmu@yahoo.co.in, khanraees@yahoo.com

Abstract—Design phase of the software development is the most appropriate one for incorporating security decisions. Unfortunately, no efficient methodology or tool exists to address security at design phase. Almost negligible work has been reported to assess the impact of object oriented design characteristics on security, though the effort has already been made for quality. Since, the approach presented in the paper investigates how coupling induces vulnerability propagation in an object oriented design, it acquires significance. An algorithm has been introduced to compute Coupling Induced Vulnerability Propagation Factor (CIVPF) for an object oriented design. A case study of Automated Teller Machine (ATM) has been carried out to validate the approach.

1. INTRODUCTION

Software security is concerned with maintaining confidentiality, integrity and availability of data managed by the software [14]. A software failed to offer the three attributes of security is termed as vulnerable software [14]. Vulnerabilities are the defects introduced in the software during its development [15]. These vulnerabilities are the only reason behind confidentiality, integrity and availability breach of the software. That is why less vulnerable software which may protect an organization’s assets from attacks, is in high demand among security seekers. This has given birth to the development of strict security mechanisms. But merely applying security mechanisms does not solve the problem. Strict security mechanisms badly affect usability and also no security mechanism guarantees complete security [7]. The problem becomes worse when the vulnerabilities are propagated from one component of the software to the other. The reason behind is simple; increasing availability of a vulnerable entity would also increase the vulnerability [8].

Since design phase prepares skeleton of the software, making changes and corrections in the phase is much easier than to make them in the subsequent phases [5]. A single design flaw may manifests itself causing serious security related consequences. An incidence reported at Choice Point Corporation, where theft of personal data from a database of background files of most American Citizens occurred, is one of the burning examples of security related problems because of design flaws [10]. Unfortunately, vulnerability detection and correction process in design phase is still almost manual. This consumes more time, resources and efforts [11].

In order to minimize vulnerabilities, the propagation of vulnerabilities must be controlled [8]. Further, the means by which these propagation are made (i.e. the design characteristics), need to be analyzed to make appropriate decision regarding the same. Coupling is one of the means responsible for the vulnerability propagation [18]. A humble attempt has been made to investigate the impact of coupling on propagation of vulnerabilities and its minimization for object oriented design.

The rest of the paper is organized as: next section discusses about existing approaches in the area and their limitations. An approach for measuring Coupling Induced Vulnerability Propagation Factor (CIVPF) has been discussed in section 3. In section 4, a case study of Automated Teller Machine (ATM) has been carried out to validate the approach. Section 5 discusses about interesting results of the approach and their interpretation. Advantages and limitations of the approach are listed in section 6. Paper concludes at section 7.

2. EXISTING APPROACHES AND THEIR SUITABILITY

Most of the existing approaches [3-6, 17] for minimizing threat exploiting vulnerable design are subjective in nature, time consuming and inefficient. On implementing these approaches for minimizing vulnerabilities, a detailed study of the given class hierarchy, sequence diagram, collaboration diagram etc. are required in order to identify
doors to attackers [5]. After identifying the attack prone points, security mechanisms are applied thinking that it would protect the vulnerable design from exploitation. Unfortunately, all the approaches do not discuss the way the vulnerability is propagated. In absence of any foolproof security mechanism to be implemented to minimize vulnerabilities [7, 8], it becomes necessary to think on reducing vulnerabilities by other means. Just like quality [1, 9], design characteristics may be mapped with the security vulnerability. A little work has been done on examining how vulnerability is dependent on the design characteristics including inheritance, coupling, cohesion, encapsulation etc [20]. It is worthwhile looking at minimizing vulnerability propagation by adjusting the software design characteristics in design phase itself for making the design more resistant to attacks.

3. THE APPROACH

An object oriented design (OOD) is centered on key entities as object, attribute, method and classes. These entities are arranged to form the design of software through design concepts including Inheritance, Coupling, Cohesion, Encapsulation, polymorphism etc. In order to measure and minimize Vulnerabilities and their propagation, design concepts must be investigated to find out whether they propagate vulnerability or restrict its propagation. Vulnerability minimization may be carried out by avoiding the use of concepts causing vulnerability propagation and by encouraging the use of concepts restricting the vulnerability propagation.

Coupling is the dependency of an entity to the other. Considerable research has been made to identify and assess the effect of Coupling on quality. It has been proven that higher coupling has negative impact on quality [1]. As security is an attribute of quality, it becomes necessary to examine the effect of coupling on security too. At the same time, it has also been realized that development of large and complex systems can not be possible without Coupling [16]. It is an unavoidable design decision. Whether it is quality or security, Coupling can not be ignored at all. So, when it comes to security, instead of minimizing overall Coupling, the main focus should be on minimizing coupling of security related entities.

1.1 Terminology used

Following is the terminology used in the approach:

3.1.1 Vulnerable Attribute and Vulnerable Method

Software vulnerability is defined as ‘susceptibility to attack’, i.e. every entity in the software which is attack prone is considered to be vulnerable. Hence, an attribute is considered to be vulnerable if it possesses one or more of the following characteristics [2]:

- Provides entry points for external application
- Processes confidential information
- Involves in internal network communication
- Allows user authentication and control

A method is considered to be vulnerable if it uses a vulnerable attribute.

3.1.2 Root Vulnerable Class

A class is termed as root vulnerable class if it declares at least one vulnerable attribute.

3.1.3 Induced Vulnerable Class

A class is termed as Induced Vulnerable Class if it is coupled with root vulnerable class through a vulnerable attribute or is coupled with one or more induced vulnerable classes.

3.1.4 Coupling Induced Vulnerability Propagation

If a class A is a Root Vulnerable Class and class B is defined in such a way that

- Class B is inheriting a vulnerable attribute of class A or
- A vulnerable method of class A is called by some method of class B or
- A vulnerable attribute of A is passed as parameter to some method of B or
- Vulnerable class A is the type of an attribute of class B.

Then A and B would said to have Coupling Induced Vulnerability Propagation and class B would be called an Induced Vulnerable Class. The phenomenon is termed as Strong Vulnerability Propagation or First Level Propagation.

However, if a class A is an Induced Vulnerable Class and B is a class such that one of the attribute of class B is of type A, then this type of Coupling Induced Vulnerability Propagation would be Weak Vulnerability Propagation or Second/ Higher Level propagation. The term ‘weak’ has been taken for the propagation because in this type of propagation, classes declaring the attribute of type induced vulnerable class, may or may not use vulnerable attribute.
1.2 Measuring CIVP of an Object Oriented Design

The aim of the work is to investigate how vulnerabilities are propagated from root vulnerable classes to others. All the classes communicating with root vulnerable class through its vulnerable attribute would be strong vulnerable. The other classes in design communicating with strong vulnerable classes are weak vulnerable and the vulnerability propagation becomes weaker and weaker for the next levels of communication. If there are p vulnerable attributes and M classes are root vulnerable because of these p attributes (p ≥ M, as a class may have more than one vulnerable attributes), computation of Coupling Induced Vulnerability Propagation Factor may be performed as:

\[
\text{CIVPOOD} = L_1 + L_2 + \ldots + L_p
\]

Hence,

\[
\text{CIVPOOD} = \sum_{i=1}^{p} L_i \quad (1)
\]

So, Coupling Induced Vulnerability Propagation Factor comes out as:

\[
\text{CIVPF} = \frac{\text{CIVPOOD}}{N} = \frac{\sum_{i=1}^{p} L_i}{N} \quad (2)
\]

Where, N is the total number of classes in OOD. An algorithm computing CIVPF has been described in the next subsection.

1.2.1 The Algorithm

If M classes out of N classes in an OOD are root vulnerable classes, an algorithm has been proposed to compute overall CIVP using which it computes Coupling Induced Vulnerability Propagation Factor (CIVPF). The algorithm takes as input a list class_list[1..N], and a queue VUL[1..M]. class_list[1..N] contains the classes present in the design and VUL[1..M] consists of the root vulnerable classes. The algorithm produces Induced Vulnerable trees corresponding to all vulnerable attributes present in the design. Roots of the Induced Vulnerable Trees are root vulnerable classes. Since a root vulnerable class may have more than one vulnerable attributes, hence it may appear in the roots of more than one induced vulnerable trees. The algorithm also produces Coupling Induced Vulnerability Propagation Factor (CIVPF) of an OOD. The algorithm maintains several additional data structures including three lists namely, parent_list, atr_list, atr_type and a queue IVC. The parent_list contains all the parent classes of an induced vulnerable class. atr_list is the list of all attributes (own, inherited and imported) corresponding to a class. The list atr_type of a class C consists of all the types, the class C declares to its attributes. The queue IVC contains all induced vulnerable classes.

\[
\text{CIVPF (Class_list[1..N], VUL[1..M])}
\]

1. civp = 0
2. for [i = 1; i ≤ N; i++]
3. parent_list [i] ← Ø
4. for [i = 1; i ≤ M; i++]
5. vul ← head (VUL)
6. for each vul_atr ∈ vul
7. vul_link ← 0
8. for each class C ∈ class_list[1..N] – vul
9. if vul_atr ∈ atr_list[C] or vul ∈ atr_type [C]
10. then parent_list[C] ← vul
11. vul_link ← vul_link + 1
12. insert [IVC, C]
13. while IVC ≠ Ø
14. ivc ← head (IVC)
15. for each class C ∈ class_list[1..N] – ivc
16. if ivc ∈ parent_list [C] and ivc ∈ atr_type [C]
17. then parent_list [C] ← ivc
18. insert [IVC, C]
19. vul_link ← vul_link + 1
20. dequeue (IVC)
21. dequeue (VUL)
22. civp ← civp + vul_link
23. civpf ← civp/N

Working

The working of CIVPF algorithm is as follows: Initially, variable civp is set as 0 and since there is no vulnerability propagation so parent_list of each class is set as Ø. From line 4 to 12 of the algorithm, every time a root vulnerable class is taken from the head of queue VUL. For each vulnerable attribute of class vul all the classes which have vulnerable attribute vul_atr in their atr_list or class vul in their atr_type are inserted into the queue IVC. Class vul is inserted into their parent_list. The variable vul_link is incremented for each induced vulnerable class showing vulnerability propagation. From steps 13 to 19 of the algorithm, the class in the head of the queue IVC is assigned to variable ivc till the queue is empty. For each ivc, if it is not present in the parent_list of any class C and is present in the atr_type of C, then C is inserted into the queue IVC. The class ivc is put into the parent_list of C. vul_link is incremented to reflect that a more vulnerable link is added because of coupling. In steps 19 and 20, IVC and VUL are de queued, as propagation of vulnerability for the classes present in their head is now calculated. In step 22, the overall vulnerability propagation in the form of vul_link is added with civp to give overall Coupling Induced Vulnerability Propagation Factor of a given object oriented design.
4. A CASE STUDY

A case study of Automated Teller Machine [13] has been carried out to validate the claim that the algorithm CIVP will report all the induced vulnerable classes and links. A comparison has been made with the outcome on implementing the algorithm to the results obtained using existing manual approach (threat modeling). All of the ways, a system can be attacked, identified using threat modeling, have been compared with the results produced by implementing the proposed algorithm. If the results obtained using manual method is same as the results on implementing algorithm, the proposed algorithm may be treated as validated.

The working of ATM is as follows: Every time when a customer inserts a card into the ATM, a session is allotted to him. The PIN is asked from the customer and is send to bank for verification. If PIN is verified, the customer is allowed to perform one or more transactions (withdrawal, transfer, enquiry, deposit). If it is not, PIN is again asked from the customer and the process is repeated thrice. If the PIN entered from the customer is still incorrect, the card is blocked. The session allotted to the customer is aborted and the ATM gets ready to serve other customers.

The ATM design has 22 classes, listed as:
(ATM, CustomerConsole, Receipt, ReceiptPrinter, session, transaction, withdrawal, transfer, enquiry, deposit, status, message, money, balances, log, cashDispensor, NetworkToBank, OperatorPanel, CardReader, card, AccountInformation, EnvelopeAcceptor)

1.1 Computing Vulnerability Propagation(VP) Manually

In order to find out vulnerability propagation manually, all of the design documents of ATM including Collaboration Diagram, Sequence Diagram and Class Hierarchy are to be made available. On going through the design documents of ATM, it is found that the attribute pin, declared in class ‘session’ is the only attribute through which an attacker can get access of system. This makes the class session vulnerable which is considered as a root vulnerable class. Communication of the vulnerable class with other classes through the attribute pin will make the other classes vulnerable. If all the vulnerable classes are placed in a set, initially, there will be only one element in the vulnerable set as:
Vulnerable set = {session}

Sequence diagram of class ‘session’, as shown in figure-1, depicts how a vulnerable attribute pin in root vulnerable class ‘session’ propagates vulnerability in design. Class ‘session’ interacts with class ‘customerConsole’ and asks for pin to perform a transaction interacting with class ‘transaction’. Classes ‘withdrawal’, ‘deposit’, ‘transfer’ and ‘enquiry’ inherit the class ‘transaction’. On summarizing the entire communication, the set of vulnerable classes may be shown as:
Vulnerable set = {session, transaction, CustomerConsole, withdrawal, transfer, deposit, enquiry}

Further, to verify the vulnerability propagation because of the vulnerable class ‘transaction’, its sequence diagram must be made available. Figure-2 shows the sequence diagram of class ‘transaction’ where class ‘transaction’ communicates with a class ‘networkToBank’ through class ‘message’ carrying attribute pin. This makes the class ‘networkToBank’ along with ‘message’ vulnerable. Further, class ‘status’ uses attribute pin to verify its correctness. Vulnerable class ‘networkToBank’ sends a message to the class ‘log’ and makes it vulnerable. On completion of transaction, a class ‘receiptPrinter’ is asked to print a receipt using class ‘receipt’. This communication makes classes receiptPrinter and ‘receipt’, vulnerable. Therefore, these classes should be added to the set of vulnerable classes. Hence,

Vulnerable set = {session, transaction, CustomerConsole, withdrawal, transfer, deposit, enquiry, NetworkToBank, message, log, receipt, receiptPrinter, status }

A transaction may be any of four types: withdrawal, transfer, deposit, enquiry. Collaboration of these classes with others must be analyzed. For simplicity, only withdrawal transaction collaboration diagram (shown in figure-3) has been taken for the discussion. The induced vulnerable ‘withdrawal’ class communicates with the classes ‘CustomerConsole’, ‘CashDispensor’, ‘Receipt’, ‘Message’. Hence

Vulnerable set = {session, transaction, Customer Console, withdrawal, transfer, deposit, enquiry, NetworkToBank, message,log, receipt, receiptPrinter, status, CashDispensor }.

Similarly, study of Collaboration diagrams of other transaction types of class ‘session’ along with other Sequence Diagrams; adds four more classes in the vulnerable set. As a result, the final set of all vulnerable classes in the given ATM design may be shown as:
Vulnerable set = {session, transaction, CustomerConsole, withdrawal, transfer, deposit, enquiry, NetworkToBank, message,log, receipt, receiptPrinter, status, CashDispensor, EnvelopeAcceptor, ATM, OperatorPanel, card reader}
When finding out the complete set of vulnerable classes, it has been observed that all of the design documents including Collaboration Diagram, Sequence Diagram and class hierarchy must be made available. It is further to pinpoint that a strong brainstorming is required to implement the same.

4.2 Computing Vulnerability Propagation (VP) Through Proposed Approach

For implementation of the proposed approach any type of documents including Collaboration Diagrams, Sequence Diagrams, State Diagrams and Class Hierarchy are not required. The only thing needed is the detailed design of classes showing all the methods and attributes. All that needed to prepare separate attribute list atr_list of each class and attribute type atr_type used in each class. Since there is only one vulnerable attribute i.e. pin, which possesses the causes of being vulnerable mentioned as:

- It provides an entry point for the user.
- It checks whether a user is authentic or not.
- It works as a communication interface/channel between user and ATM machine.
- An attacker will have to somehow compromise pin in order to know the details of others account and hence perform the illegal actions as transfer, withdraw, inquiry etc.

The list of 22 classes present in the ATM design and vulnerable class ‘session’ (because of its own vulnerable attribute pin) will form input of the algorithm CIVPF. The algorithm will produce a single Induced Vulnerable tree as shown in figure-4. The arrows are vulnerable links showing vulnerability propagation. Dashed arrows reflect weak vulnerability propagation. The nodes of the trees represent the sets of all vulnerable classes; a malicious user may communicate for his benefit or damaging the information of others. Coupling of all the classes through vulnerable attribute pin makes it possible. The nodes of the tree can be represented by the set vulnerable node as:

{session, transaction, CustomerConsole, withdrawal, transfer, deposit, enquiry, NetworkToBank, message, log, receipt, receiptPrinter, status, CashDispensor, EnvelopeAcceptor, ATM, OperatorPanel, card reader}

If we closely observe the two results, (first obtained from the manual method, second on implementing the proposed approach), these both are the similar. This verifies the claim and hence the proposed method. Further, the algorithm calculates CIVPF for the ATM design as follows:

1. $civp = \text{all the vulnerable links} = 23$ (calculated using algorithm)
2. Total number of classes in design = 22 (given)
3. $civpf = \frac{23}{22} = 1.05$

![FIGURE 1. SESSION SEQUENCE DIAGRAM](image_url)
The proposed method has been implemented using a simple case study of ATM with only one vulnerable attribute. The manual method used in identifying the vulnerability propagation through the attribute is quite complex and time consuming. Possibility of error is also high for large and complex design. A design with large number of vulnerable attributes will make the implementation more complex and confusing.

On the other hand, the proposed approach seems to be simple, efficient and more practical than the manual one. Even in presence of number of vulnerable attributes, induced vulnerable trees corresponding to each attribute may be easily obtained using the proposed algorithm. This, of course, will save a lot of time and effort in identifying the vulnerable classes in design because of the coupling through vulnerable attributes.
5. INTERPRETATION AND DISCUSSION

The two extreme cases may be there. In first case, vulnerable classes do not communicate with other classes through their vulnerable attributes. In this case

\[ L_{i} = 0 \]

for each root vulnerable class

No induced vulnerability propagation is made in such case and CIVPOOD will be min. therefore, numerator of equation (2) will be

\[ \text{CIVPOOD (min)} = \sum_{i=1}^{p} L_{i} = 0 \]  \hspace{1cm} (3)

In second extreme case, a root vulnerable class may induce vulnerability in remaining of the N-1 classes making them strongly induced vulnerable classes. These induced vulnerable classes, except their parent vulnerable classes, may propagate weak vulnerability among them. The case may be more clearly understood by referring figure-5. Class C1 is a root vulnerable class which induces strong vulnerability to classes C2, C3, and C4. Now, C2 may propagate vulnerability to C3, and C4. C3 again may propagate vulnerability to C4. Therefore, \( L_{i} \) may be calculated as:

\[ L_{i} = N^{*}(N-1)/2 \]

for each root vulnerable class

In this case, IVPOOD will be max and numerator of equation (2) will be

\[ \text{CIVPOOD (max)} = \sum_{i=1}^{p} L_{i} = p^{*} N^{*}(N-1)/2 \]  \hspace{1cm} (4)

Equation (3) and (4) are summarized to yield the inequality

\[ 0 \leq \text{CIVPOOD} \leq p^{*} N^{*}(N-1)/2 \]

Using the above inequality, the range of CIVPF of equation (2) comes out to be

\[ 0 \leq \text{CIVPF} \leq p^{*} (N-1)/2 \]  \hspace{1cm} (5)

Since, it is assumed that \( p > 0 \), and \( N \geq 1 \) for any object oriented design. Hence, the interpretation about CIVPF may be drawn as:

"The higher the CIVPF of a design, the higher vulnerable the design is."

By minimizing the CIVPF of a design, its vulnerability can be minimized. Two given designs may also be compared on the basis of their CIVPF. Again, for the purpose of discussion, inequality in equation (5) may be written in another way as:

\[ 0 \leq \text{CIVPF} \leq \left( \sum_{i=1}^{p} L_{i} \right) \text{ max} / N \]  \hspace{1cm} (6)

From the inequality (5) and (6), it is clear that CIVPF depends upon vulnerable attributes \( p \), number of classes \( N \) and overall number of vulnerable links \( L_{i} \). Neither the number of vulnerable attributes can be minimized nor the number of classes in a design. Hence, when it comes to minimize Vulnerability of a given design must be reduced to get a better design.
The Induced Vulnerable Trees corresponding to each vulnerable attribute produced by the algorithm show the direction of vulnerability propagation from one class to the others. The propagation becomes weaker and weaker when traversing from root class to leaf classes of the Induced Vulnerable Tree. By carefully inspecting the trees, unnecessary coupling may be avoided. Hence, the trees prove to be an aid to minimize vulnerability and come up with a better design.

6. ADVANTAGES AND LIMITATIONS

Advantages of CIVPF are listed as

- Designs of different versions of object oriented software as well as designs of different software may be compared at design phase itself.
- It may be used as a readymade solution to protect assets and security critical entities by minimizing their coupling.
- An existing design may be made less vulnerable by avoiding unnecessary coupling of vulnerable attributes.
- The approach is quick, effective and practical.
- With slight modification, the approach may be implemented for procedural design too.

Contribution of other design characteristics has been ignored when computing vulnerability propagation. Impact of inheritance on vulnerability propagation has already been covered in the work [8]. The effect of remaining characteristics on vulnerability propagation has been left as future work.

![Figure 5. Extreme Highest Case of CIVP](image)

6. CONCLUSION

Software security has become a new buzzword in the industry. Designing and developing secure software has become the main objective of developers [12]. The existing approaches just rely on security mechanisms. As no security mechanism guarantees hundred percent security, hence they do not provide much help. The approach presented focuses on minimizing vulnerabilities of an object oriented design by controlling their propagation via coupling. CIVPF is assessed so that unnecessary coupling can be avoided to come up with a better design. A detailed case study of Automated Teller Machine (ATM) has been carried out to assess the effectiveness of proposed approach over the other existing one.

REFERENCES


ABOUT THE AUTHORS

Alka Agrawal. Alka Agrawal is working as a research scholar in the department of Information Technology, B.B.A. University (A Central University) Lucknow, India. Her research area is Software Security.

R. A. Khan. R. A. Khan is working as a Reader in the Department of Information Technology, B.B.A.University (A Central University) Lucknow, India. He has been working in the area of software quality, software security and software testing.