Integrating Security Concerns into Software Development

Sabah Al-Fedaghi and Fajer Al-Kanderi

Computer Engineering Department, Kuwait University
sabah@alfedaghi.com, Eng.fjk@gmail.com

Abstract

It has become clear in software development that functionality and security must go hand in hand in cases where security concerns are to be incorporated early in stages of design. An essential aspect of such a process is threat modeling that integrates security with functional specification. Such an approach includes construction of two models: a functional model and a security (*threat*) *model. The problem of integration involves assimilating security concerns while developing system specifications. One solution is to represent the dynamic behavior of security attacks as statechart diagrams and integrate the attacks into the functional behavior of the system; however, such an approach results in a complex set of fragmented descriptions lacking an underlying conceptual representation that can be tailored to include security concerns. This paper introduces a flow-based diagrammatic representation that includes such features. The advantages of the methodology are demonstrated through contrasting it with a statechart-based study case.*

Keywords: *Software development, security concerns, threat modeling, conceptual modeling*

1. Introduction

Information system security has become a central concern in the face of growing uncertainty caused by greater dependence on information technology. Developing an effective defense requires a comprehensive understanding of vulnerabilities to attacks and exploitation. Any strategy adopted in this context should address all the potential points of weakness in the system.

System designers can plan defenses in a systematic way and can provide users with strategies for minimizing possible security weaknesses in the environment in which the system is deployed. Most importantly, a systematic approach to security can help designers, developers, and users to forsake naïve views that technology alone will provide a solution and to adopt a more sophisticated and holistic view of system security by proper identification of the areas of potential risk [1].

It has become clear in software development that functionality and security must go hand in hand in cases where security concerns are to be incorporated early in stages of design. According to [2], exposure to security risks can originate or increase as a result of practices that include the following:

- Security considered late in the development lifecycle
- Lack of definition of security requirements

- Inadequate definition or implementation of security policies

Early stages in a software development cycle involve requirement analysis, and design. A more comprehensive view of vulnerability creation would take into account that the source of vulnerability might be found at any point in these stages. Specifically, specification of requirements may lack security constraints and thus cause vulnerability.

Additionally, "a great deal of software is designed in ad hoc fashion, and security is often a victim of the chaos" [3]. Problems arise from difficulties in the specification of imprecise security concerns. "As software becomes more complex, the chance of design errors and ambiguities increases and as a result system vulnerability to security attacks increases" [4].

An important aspect in this context is the need to understand the vulnerabilities of the software system to prevent the system from being compromised. One approach to providing protection against vulnerability is to react to each vulnerability by applying patches to close "holes" in the software, thus countering any exploitation. Large databases of vulnerabilities disclosed in previous years are publicly available, *e.g.*, the National Vulnerability Database [5], the Open Source Vulnerability Database [6], and the CVE database [7].

Alternatively, protection can be based on "finding" vulnerabilities and offsetting possible attacks. This includes threat modeling that integrates security into the design process [8, 9]. Abstract representations of attacks can provide a framework for identifying points of potential security concern and to develop protection strategies. "A threat model … helps people determine the highest level security risks posed to the product and how attacks can manifest themselves. The goal is to determine which threats require mitigation and how to mitigate the threats" [3]. It "allows experts from different discipline to communicate and understand security concerns through a model that is familiar to most of them. Yet, these models represent security attacks at a high level of abstraction" [4].

Typically, modeling diagrams of a system (*e.g.*, DFD, UML) are used to derive and determine threats "relying on the solid understanding of the system architecture. In this process, software is first described using DFD diagrams. Security threats are then represented as attack trees that depict different potential attacks against a system" [4].

This paper focuses mainly on the general problem of integrating security into the development process, typically involving the construction of two models. First, a functional model is necessary to assure the accuracy of requirements and design. Second, a security (threat) model serves to describe different possible attacks.

The main hindrance in software security is that software developers lack the understanding of security concepts. Software is designed with the mindset of its functionalities and cost, where the focus is on the operational behavior, while security concerns are neglected or marginally considered. [4]

Ariss and Xu [4] proposed integrating functionality and security concerns through an approach that models threats "using statecharts and attack trees [that] are integrated into system models."

The main purpose is to increase the security awareness of software engineers by modeling the dynamic behavior of security attacks and integrating it with the functional specifications. From this model, and through the focus on the behavior of an attack against the system behavior, it is possible to verify system functionalities in order to identify and mitigate vulnerabilities and remove ambiguities, errors, conflicts and inconsistencies in the system design [4].

This paper introduces an alternative diagramming methodology based on the notion of flow. It is shown that this flow-based methodology is suitable for refining the functional model based on analysis of attacks.

2. Problem: Integrating Security Concerns and Requirement Specification

The problem of integrating security into the development process involves assimilating security concerns while developing system specifications. This problem is manifested in terms of identifying and modeling security threats during the development process using security databases that collect and maintain cases [10, 11].

Security problems are typically described using diagrams such as fault trees [12], attack/defense trees [13], misuse cases [14-16], anti-goals [17] and Petri nets [18, 19]. As the dominant diagraming methodology, UML has been extended to incorporate security considerations utilizing such notions and tools as "attack patterns" [20, 21], sequence diagrams [22], and misuse cases [19]. UMLsec, an extension of UML, utilizes statecharts [23].

This paper introduces a new diagrammatic methodology to integrate functional requirements and security concerns in the early stages of software development. It is not possible to contrast such a new methodology against all these approaches, which have similar aims and utilize the different types of diagrams mentioned above; consequently, without loss of generality, we focus on one very recent work in this direction in detail, as follows.

3. Sample Approach

Ariss and Xu [4] integrate security concepts into the development process of software systems for the purpose of validation and verification of functional specifications [24, 25]. Additionally,

The proposed approach represents the dynamic behavior of security attacks as statechart notations and integrates the attacks into the functional behavior of the system. This integrated model allows a better understanding of security issues through a semantically well-defined representation and through the analysis of both security and functional requirements [4].

Basically, they integrate attack tree representations into statechart-based functional models.

Attack trees are complex to integrate. This is because an attack tree is not intended to describe the behavior of the system while the threat is occurring. Instead, the attack tree represents the point of view of an attacker by describing how an attacker can perform an attack following a step by step format. On the other hand, fault trees represent hazards which are the product of the behavior of the system; that means all the nodes are part of the functional behavior. As for attack trees, many of the nodes might stand for external behavior.

Ariss and Xu [4] describe the application of their proposed approach by using a shopping cart system as a study case. In this system, a customer selects and purchases products online. The system's functional model is represented in Figure 1. According to [4], the shopping cart system contains the following components: A shopping cart that keeps track of the selected items a customer is purchasing. It allows the customer to add/remove products from the current session. The system also contains a browsing component that allows a user to browse the available items and view their descriptions. The shopping cart system also comprises a sign-in component to allow users to either create an account, login to a new account, or login using an existing account. A checkout component enables a customer to purchase the items that have been selected and added to the shopping cart. A back-end database is needed to store customer information, inventory, and item descriptions. Assets of value to the e-commerce company include customer information, the company's reputation, and its inventory.

The process starts with integrating one attack tree into the statechart of the shopping cart system where vulnerabilities in the system are identified and then mitigated. The process continues with another attack and repeats again until the rest of the attacks are integrated, analyzed and the threats are mitigated… the list of identified threats was by no means exhaustive [4].

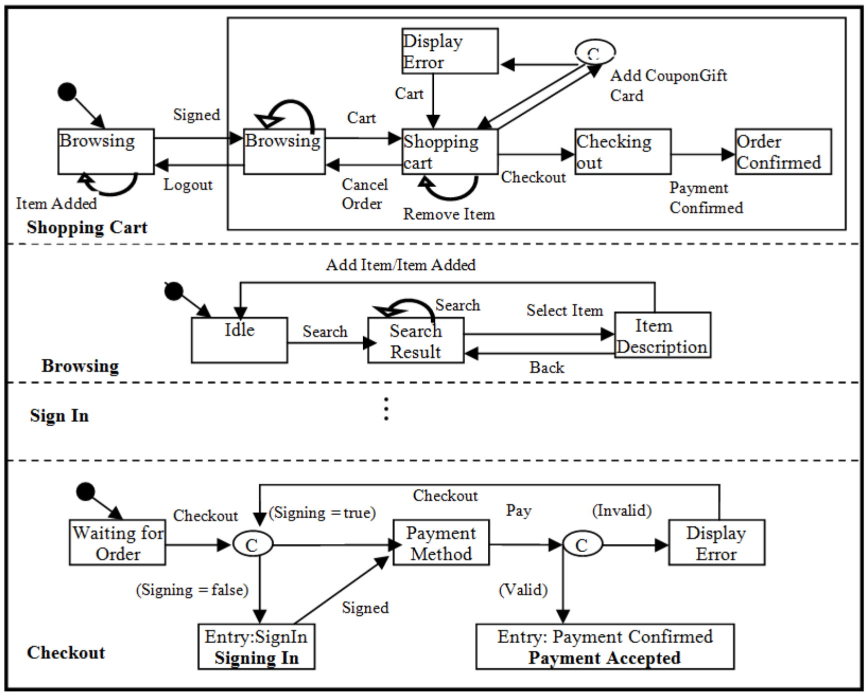


Figure 1. Partial view of the statechart of a shopping cart system (from [4])

Example: (Purchase an item at a reduced price): In this attack, the malicious user changes an item price to purchase it more cheaply than at the actual price. Figure 2 shows the case attack tree. “A malicious user first logs in into his/her user account and then modifies the price of a selected product, adds it to the shopping cart and proceeds with the checkout process” [4].

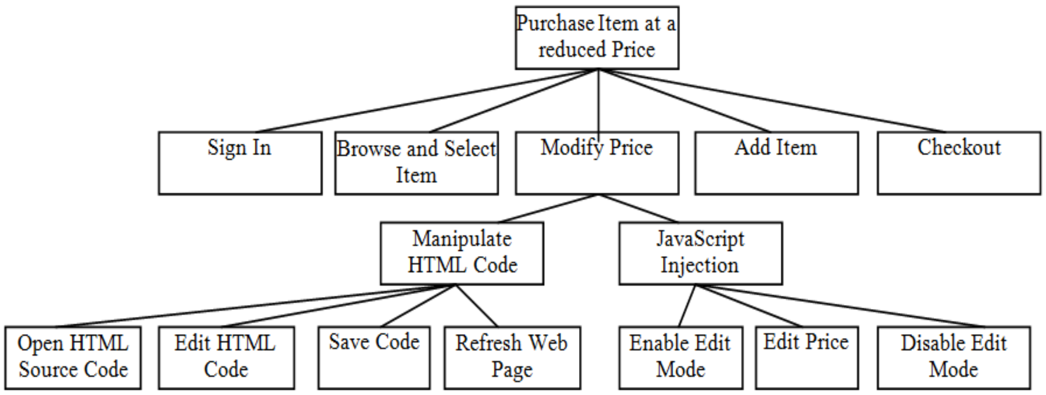


Figure 2. Attack tree modeling purchase of an item at a reduced price (from [4])

Ariss and Xu [4] then perform the following steps:

1. Deduce a threat formula from the attack tree, *e.g.*,

   Purchase Item at a Reduced Price = (∧, Sign In, Browse and Select Item, (∨, (∧, Open HTML Source Code, Edit HTML Code, Save Code, Refresh Web Page), (∧, Enable Edit Mode, Edit Price, Disable Edit Mode)), Add Item, Checkout)

2. Check for any availability of priority AND gates
3. Decompose leaf nodes into simple definitions
4. Deduce the semantics table
5. Expand primitive functional and threat conditions
6. Transform gates in the threat formula
7. Verify identification of vulnerabilities

At the end the authors produce the diagram in Figure 3, integrating the shopping cart model with threat concerns.

First, the newly formed statechart components are added to the system statechart as orthogonal parts. Next, the functional behavior of the system should be modified to indicate the occurrence of the attack. This is done by adding a new state to the shopping cart component. The Browsing, Sign In, and Checkout functional components have not been modified during the integration process and thus are not shown by the figure in order to limit the size of the model [4].

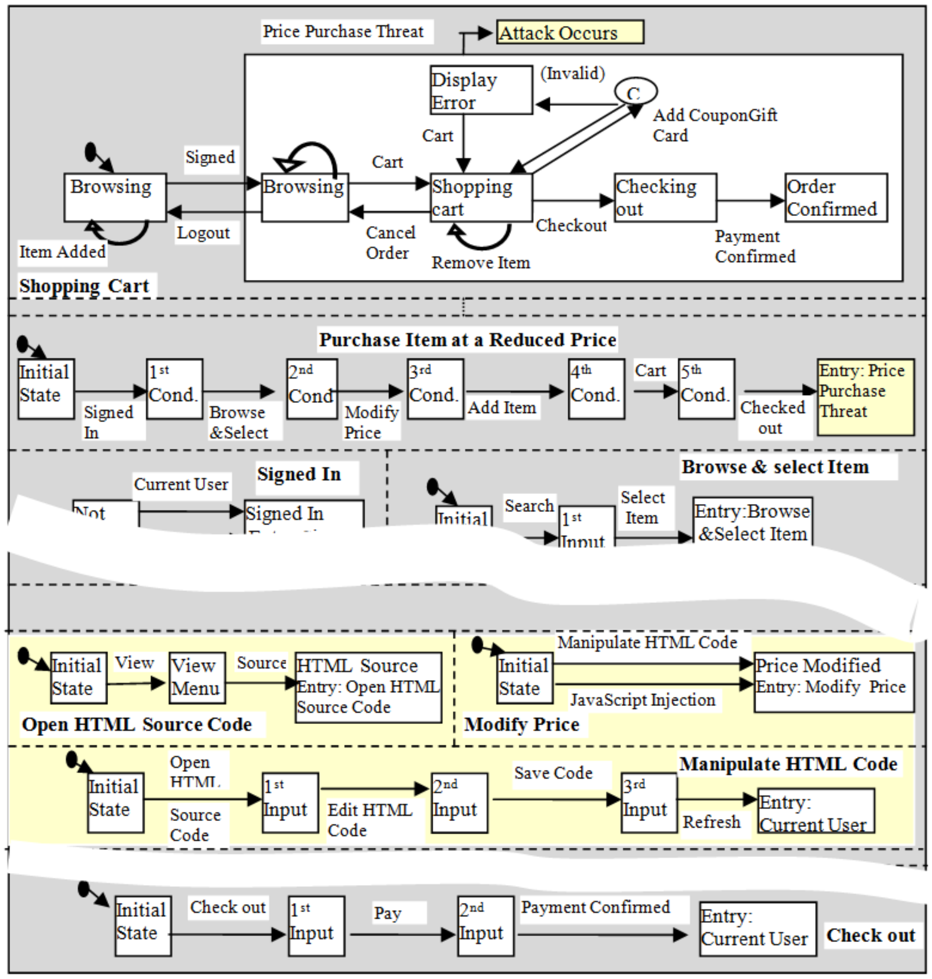


Figure 3. Partial view of the integrated shopping cart model with threat concerns included (from [4])

While this description of such admirable work in incorporating attacks into the original functional specification is not easy to understand, it is sufficient for our purpose: contrasting it with the diagrammatic representation of requirements and attacks created with our Flowthing Model (FM), because the difference is not in the details; rather, it is in the method of depiction of ideas. FM is simpler, more complete, and more suitable for incorporating new specifications than the method developed by [4]. This is analogous to preferring one language over another based on simplicity, expressive notions, and understandability in representing the same concepts.

The next section reviews the notion of flow systems from previous publications in various application areas [26-29].

4. Flowthing model

The Flowthing Model (FM) is a uniform method for representing "things that flow," called *flowthings*. Flow in FM refers to the exclusive (i.e., being in one and only one) transformation among six *states* (also called stages): transfer, processing, creation, release, arrival, and acceptance.

The fundamental elements of FM are as follows:

Flowthing: A thing (*e.g.*, information, money, electrons) that has the capability of being created, released, transferred, arrived, accepted, and processed while flowing within and between systems.

A flow system (referred to as *flowsystem*) is a system with six stages and transformations (boundaries) among them.

Spheres and subspheres: These are the environments of the flowthing, such as a company and the departments within it, an instrument, a computer, an embedded system, a component, and so forth. A sphere can have multiple flowsystems in its construction if needed.

Triggering: Triggering is a transformation (denoted by a dashed arrow) from one flow to another, *e.g.*, flow of electricity triggers the flow of water.

We use *Receive* as a combined stage of *Arrive* and *Accept* whenever arriving flowthings are always accepted.

In addition to the fundamental characteristics of flow in FM, the following types of possible operations exist in different stages:

1. *Copying*: Copy is an operation such that *flowthing* f $\Rightarrow$ f. That is, it is possible to copy f to produce another flowthing f in a system S. In this case, S is said to be S *with copying* feature, or, for short, *Copy S*. For example, any informational flowsystem can be copy S, while physical flowsystems are non-copying S. Notice that in copy S, stored f may have its copy in a non-stored state. It is possible that copying is allowed in certain stages and not in others.

2. *Erasure*: Erasure is an operation such that *flowthing* f $\Rightarrow \varnothing$, where $\varnothing$ denotes the *empty flowthing*. That is, it is possible to erase a flowthing in S. In this case, S is said to be S *with erasure* feature, or, for short, *erasure S*. Erasure can be used for a single instance, all instances in a stage, or all instances in S.

3. *Canceling*: *Anti-flowthing* f^- (f with superscript –) is a flowthing such that $(f^- + f) \Rightarrow \varnothing$, where $\varnothing$ denotes the empty flowthing, and + denotes the presence of f^- and f.
It is possible that the anti-flowthing f^- is declared in a stage or a flowsystem. If flowthing f triggers the flow of flowthing g, then anti-flowthing f^- triggers anti-flowthing g^-.

An example of the use of these FM features is erasure of a flow, as in the case of a customer who orders a product, then cancels the order, an action that might require the cancellation of several flows in different spheres that were triggered by the original order.

Formally, FM can be specified as FM = $\{S_i (\{F_j\}, T_l), \{(F_{ij}, F_{ij})\}, 1 \le i \le n, 1 \le j \le m, 1 \le l \le t\}$

That is, FM is a set of spheres $S_1, ...S_n$, each with its own flowsystems F_{ij},... F_{im}. T is a type of flowthing T_1,..., T_t. Also, F is a graph with vertices V that is a (possibly proper) subset {Arrive, Accept*, Process*, Create*, Release*, Transfer*}, where the asterisks indicate secondary stages. For example, {Copy, Store, and Destroy} can represent these secondary stages.*

A flowsystem may not need to include all the stages; for example, an archiving system might use only the stages Arrive, Accept, and Release. Multiple systems captured by FM can interact with each other by triggering events related to one another in their spheres and stages.

5. FM-based Representation

Figure 4 shows the FM representation of the online shopping system that corresponds to the statechart diagram shown in Figure 3. The figure includes two spheres: User and System. Both spheres include many subspheres, for example, Login, Shop, and Checkout. Where a subsphere comprises only a single flowsystem, for simplicity's sake, we draw one rectangle to represent the subsphere and a particular flowsystem. For example, the Login subsphere includes only the Login-page flowsystem; in a more elaborated description, it would be possible to include, say, an error-message flowsystem in this subsphere. Note that the Cart subsphere appears in three contexts: the shop, system checkout, and user checkout.

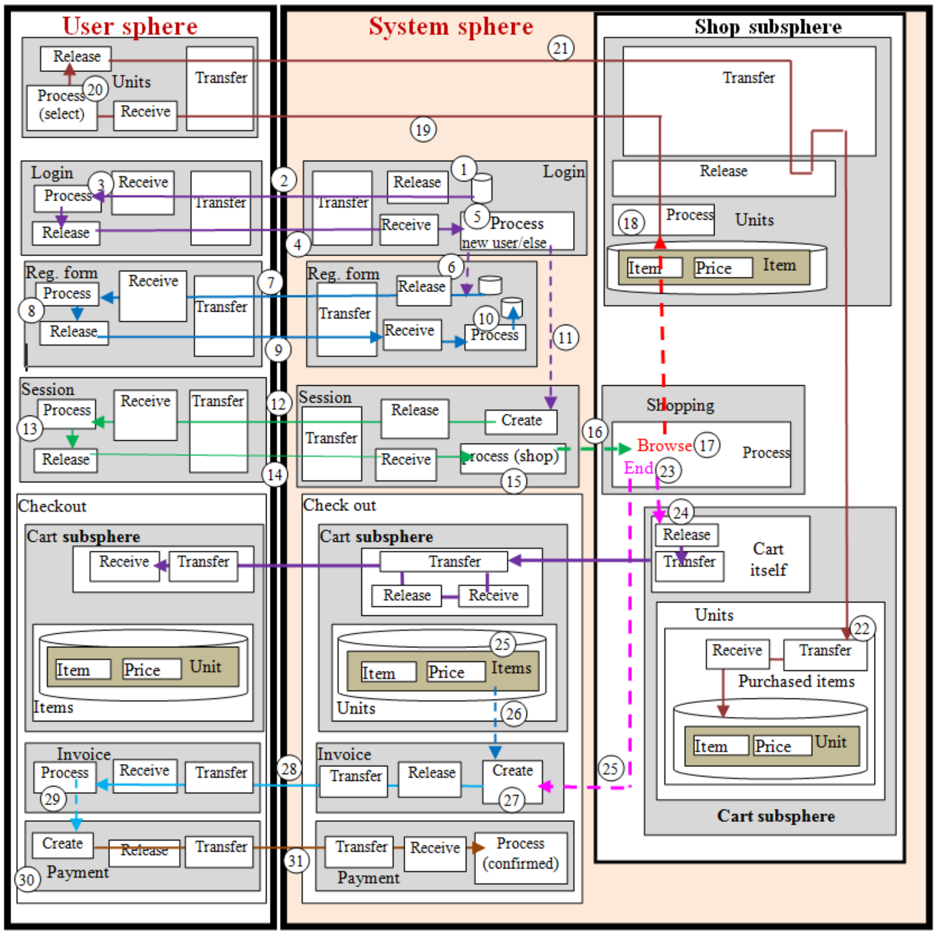


Figure 4. FM representation of the shopping cart system

In Figure 4 at circle 1, login information is initially retrieved, processed (*e.g.*, put into Web-page format), and sent to the user (2), who processes it (3) (*e.g.*, adds his/her login information), and sends it back (4) to the system. The system processes the incoming login information (5), and if the user is new, it triggers (5) retrieving and sending a registration form to the user (7), where it is processed (filled out) (8) and sent back to the system (9); the new user is then added to the list of users (10).

On the other hand, if the user decides to be a registered user, this triggers (11) opening a session for the user that includes shopping. This is accomplished by sending a page to the user (12), where it is processed (13) and sent back (14) to be processed (15) by the system. We assume here that other services are offered besides shopping that are not shown, such as refund, maintenance, and so on.

Suppose the user wants to enter the (virtual) shop. This selection triggers *Process* in shopping (16) and Browse is selected (17), giving the user the option to view all available items. User's choices then trigger the retrieval of *units* (item descriptions plus prices); these descriptions are processed by the system (put into list format) and released (19) to the user. User selects (20) the desired items and sends an order for them back to the shop (21) to be added to the cart (22). Since the purpose here is to illustrate the modeling technique, in this case we ignore the added process of removing items from the cart (*e.g.*, this function can be added alongside *Browse*, at 17).

Upon finishing, the user selects *End* in Shopping (23), which triggers moving the cart to Checkout (24) and activating invoice/payment processes (25). Note that the items flowsystem in the context of Cart is simply a list of items (in storage) in no stages. The presence of the cart in the checkout area (26) triggers the invoice process to provide a list of the selected items, which is used to create an invoice (27) that flows to the user (28). The user processes the invoice (29) to trigger (30) creation of payment that flows (31) to the system.

Contrasting the FM representation with the statechart diagram of Figure 1, we see the former presents a conceptual description united in a sequence of steps built on streams of flow that trigger each other. It can be used as a foundation for building a functional description of a system that evolves from an initial specification to refined specifications according to security concerns (see Figure 5). To demonstrate this, the next section applies the threat of users changing prices as discussed previously.

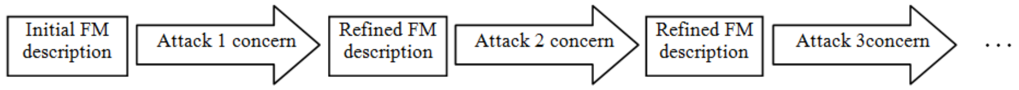


Figure 5. The FM representation lends itself to refinement according to security concerns specified by threats modeling

6. Security-based refinement of FM description

In this section, we refine Figure 4 in response to the security concern involving purchase of an item at a reduced price. In this attack, the malicious user changes an item price to make it lower than the actual price. As a starting point, Figure 6, showing a partial view of Figure 4, emphasizes regions in the FM representation of the system that can be compromised by an attack, rendering them untrusted regions. Region A includes the component that displays the list of items and their prices from which the user selects. This component receives items and returns selected items.

Note that implementation is based on the FM description. For example, the User's page that displays items and prices is built upon region A in Figure 6. The flowsystem in Region A can be viewed as the conceptual basis of a software component that receives the units of (item information, price) and gives the user the option of choosing any one of these units.

Suppose that an attack causes a compromise of the coding of component A (*e.g.*, by causing a dump of the binary code). Note that at this level of modeling, we are not concerned about implementation issues such as the type of software used in the system, or the type of technique used by the attacker. One solution to counter such an attack is illustrated in Figure 7. The functional model is modified in region B; once an index (item number j in the list) of the purchased item is received, the item information and price are retrieved from its database.

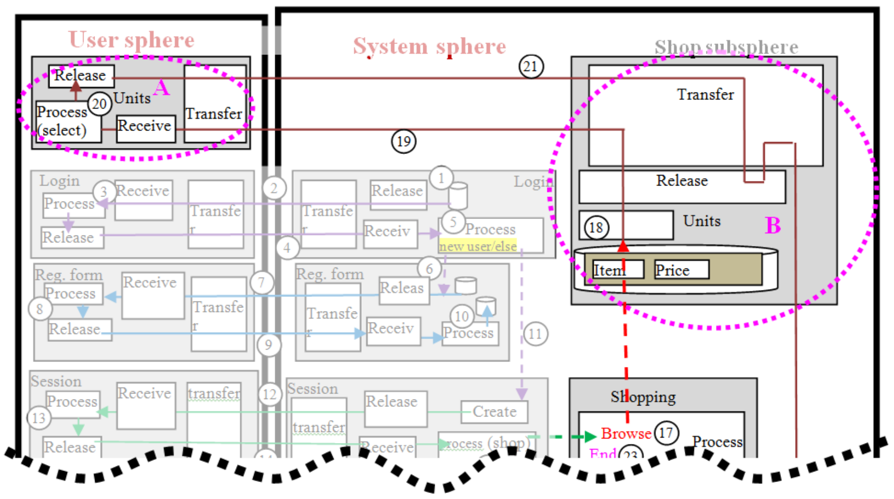


Figure 6. Regions to be analyzed when examining "purchase of an item at reduced price" attack

Now suppose that the security concern centers on compromises to region B, where the attacker takes over this region's software component in order to change the prices. One possible solution is to modify the next stage in the flow, the cart (see Figure 8), such that it receives only the index of the item, then retrieves information and prices from its own database.

Finally, suppose the Cart subsphere also cannot be trusted; it is possible to modify the cart component such that it takes information and prices from the system database, as illustrated in Figure 9.

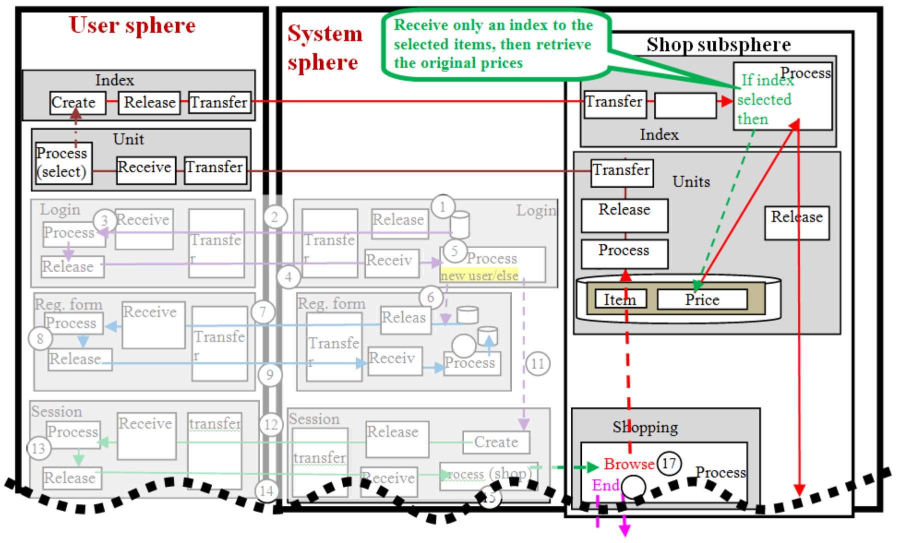


Figure 7. Change in the original flow such that prices are taken from the original database before the item is added to the cart

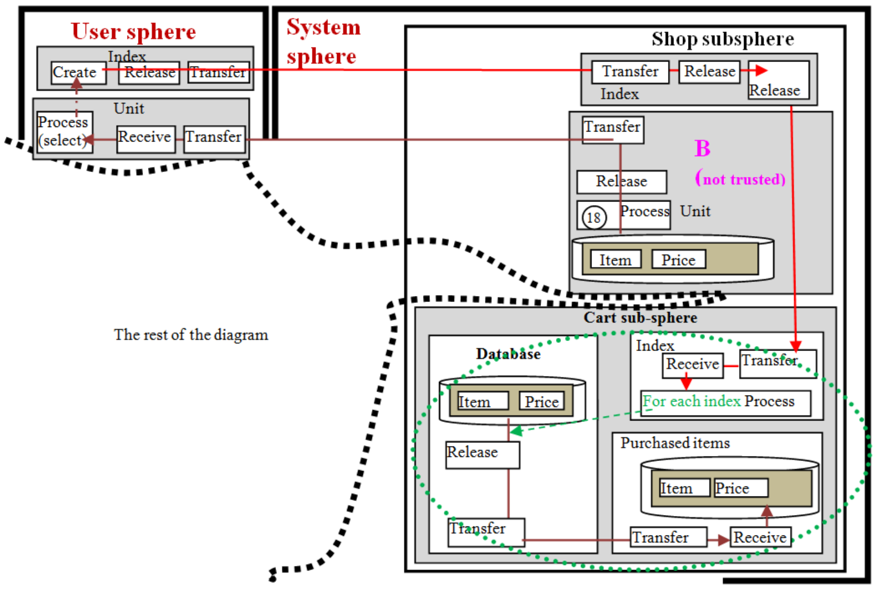


Figure 8. If region B is not trusted as a source of prices, then prices are taken from the cart's database

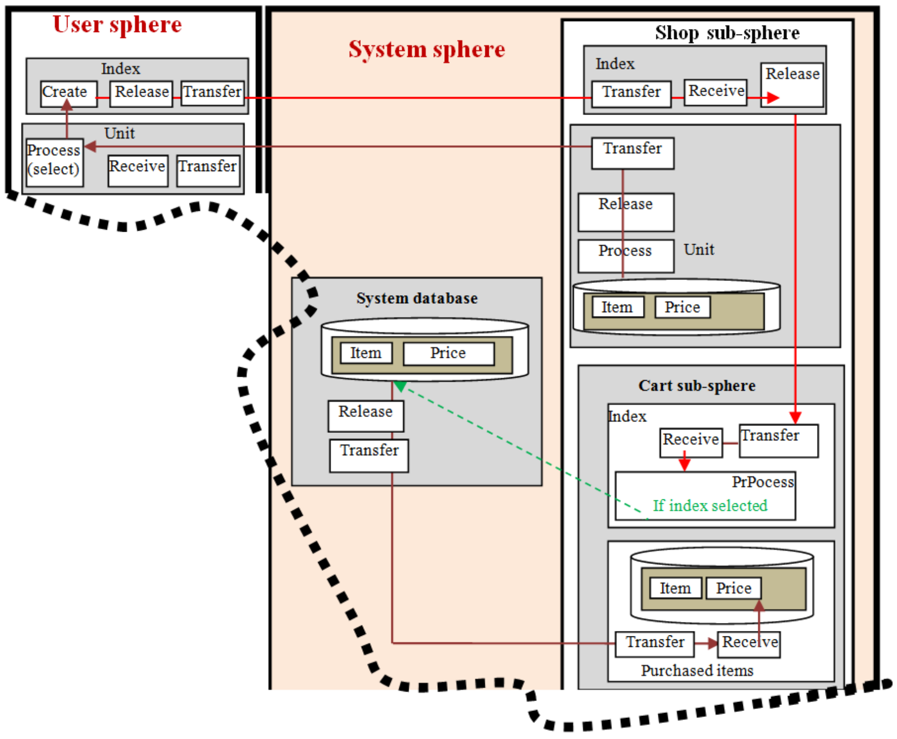


Figure 9. The cart component is not trusted; hence, prices are taken from the system database

7. Conclusion

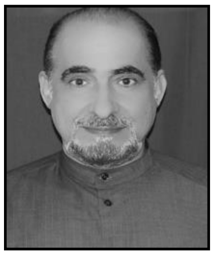
This paper is concerned with the problem of integrating security concerns while developing system specifications. It is proposed that such an integration can be accomplished through adopting a flow-based diagrammatic representation that is more amendable to incorporating modifications that counter possible attacks. The advantages of the methodology are demonstrated through contrasting it with statechart-based methodology.

References

[1] T. Roebuck, "A Diagrammatical Framework for Information Systems Attacks", in Second Annual Conference on Privacy, Security and Trust, Fredericton, New Brunswick, Canada. http://dev.hil.unb.ca/Texts/PST/pdf/roebuck.pdf, **(2004)**.
[2] P. N. Romano, "Project Risk Management", http://www.successatit.com/17.html.
[3] M. Howard and D. LeBlanc, "Writing Secure CODE", Microsoft Press, ISBN : 978-0-73561-722-3, **(2004)**. http://librairie.immateriel.fr/fr/read_book/9780735617223/ch04.
[4] O. Ariss and D. Xu, "Secure System Modeling: Integrating Security Attacks with Statecharts", Int. J. Software Inf, http://www.ijsi.org/1673-7288/6/i121.htm, vol. 6, no. 2, **(2012).**
[5] National Vulnerability Database (NVD), (2012 – access). http://nvd.nist.gov.
[6] Open Source Vulnerability Database (OSVDB), (2012 – access). http://osvdb.org .
[7] CVE database, (2012 – access). http://cve.mitre.org.
[8] F. Swiderski and W. Snyder, "Threat Modeling", Microsoft Press, **(2004)**.
[9] P. Torr, "Demystifying the threat-modeling process", IEEE Security and Privacy, vol. 3, no. 5, **(2005)**, pp. 66-70.
[10] NIST Computer Security Resource Center (CSRC), National Vulnerability Database Version 2.2 (2012 – access) http://nvd.nist.gov/.
[11] Common Criteria for Information Technology Security Evaluation, Version 3.1, Revision 3, **(2009)** July. http://www.commoncriteriaportal.org/thecc.html.
[12] G. Helmer, J. Wong, M. Slagell, V. Honavar, L. Miller and R. Lutz, "A software fault tree approach to requirements analysis of an intrusion detection system", Requirements Eng., vol. 7, no. 4, **(2002)**, pp. 177–220.
[13] B. Kordy, S. Mauw, S. Radomirović and P. Schweitzer, "Foundations of attack-defense trees, in FAST'10", Proceedings of the 7th International Conference on Formal Aspects of Security and Trust, 80-95, Springer-Verlag Berlin, Heidelberg, ISBN: 978-3-642-19750-5, **(2011)**.
[14] M. N. Johnstone, "Modelling misuse cases as a means of capturing, security requirements", in Proceedings of the 9th Australian Information Security Management Conference, Perth, Western Australia, **(2011)** December 5-7.
[15] I. Alexander, "Misuse cases: use cases with hostile intent", IEEE Software, **(2003)**, pp. 58-66.
[16] D. Xu and J. Pauli, "Threat-driven design and analysis of secure software architectures", J. Inf. Assurance Security, vol. 1, no. 3, **(2006)**, pp. 171-180.
[17] A. Van Lamsweerde, "Elaborating security requirements by construction of intentional anti-models", in Proc. Intrnational Conf. Software Eng., **(2004)**, pp. 148-157.
[18] R. Gorrieri and M. Vernali, "Classification of security properties, in Foundations of Security Analysis and Design", vol. 2171 of LNCS, Springer, **(2011)**, pp. 331-396.
[19] D. Xu and K. E. Nygard, "Threat-Driven Modeling and Verification of Secure Software Using Aspect-Oriented Petri Nets", IEEE Trans. Software Eng., vol. 32, no. 4, **(2006)**, pp. 265-278.
[20] S. Faily, J. Lyle, C. Namiluko, A, Atzeni and C. Cameroni, "Model-driven architectural risk analysis using architectural and contextualised attack patterns", in MDSec'12, Innsbruck, Austria, (**2012)** September 30.
[21] D. Mouheb, C. Talhi, V. Lima, M, Debbabi, L. Wang and M. Pourzandi, "Weaving security aspects into UML 2.0 design models", in Proceedings of the 13th Workshop on Aspect-Oriented Modeling, **(2009)**.
[22] J. Kong and D. Xu, "A UML-based framework for design and analysis of dependable software", in Proceedings of COMPSAC'08, **(2008)**.
[23] R. Matulevičius and M. Dumas, "A Comparison of SecureUML and UMLsec for Role-based Access Control", in Proceedings of the 9th Conference on Databases and Information Systems, **(2010)**, pp. 171-185.
[24] O. Ariss and D. Xu, "Modeling security attacks with statecharts", in Proceedings of the 2nd International ACM Sigsoft Symposium on Architecting Critical Systems, **(2011)**.

[25] O. Ariss, D. Xu, and J. Wu, "Building a secure system model: integrating attack trees with state- charts", in Proceedings of the 5th International Conference on Secure Software Integration and Reliability Improvement, **(2011)**.
[26] S. Al-Fedaghi and B. Al-Babtain, "Modeling the Forensics Process", Int. J. Security Appl., vol. 6, no. 4, **(2012)**.
[27] S. Al-Fedaghi, "Toward a New Methodology of Software Evolution", J. Next Generation Inf. Tech., vol. 3, no. 4, **(2012)**, pp. 78-87.
[28] S. Al-Fedaghi, "Developing Web Applications", Int. J. Software Eng. Appl., vol. 5, no. 2, **(2011)**, pp. 57-68.
[29] S. Al-Fedaghi, "Toward Flow-Based Semantics of Activities", Int. J. Software Eng. Appl., Accepted **(2012)**.

Authors



Sabah Al-Fedaghi holds an MS and a PhD in computer science from the Department of Electrical Engineering and Computer Science, Northwestern University, Evanston, Illinois, and a BS in Engineering Science from Arizona State University, Tempe. He has published two books and more than 150 papers in journals and conferences on software engineering, database systems, information systems, computer/ information privacy, security and assurance, information warfare, and conceptual modeling. He is an associate professor in the Computer Engineering Department, Kuwait University. He previously worked as a programmer at the Kuwait Oil Company, where he also headed the Electrical and Computer Engineering Department (1991–1994) and the Computer Engineering Department (2000–2007).

Fajer Al-Kanderi is a graduate student in the Computer Engineering Department at Kuwait University. Her research interest includes parallel computing, cloud computing, threat analysis, and flow modeling. She is currently working as a computer engineer in the Council of Ministers General Secretariat.