Secure Software Architectural Patterns Designed with Secure Connectors

Michael Shin[1], Taeghyun Kang[2] and Hassan Gomaa[3]
[1]Department of Computer Science, Texas Tech University, Lubbock, TX, U.S.A.
[2]Department of Computer Science and Math., University of Central Missouri, Warrensburg, MO, U.S.A.
[3]Department of Computer Science, George Mason University, Fairfax, VA, U.S.A.

Keywords: Software Architectural Patterns, Secure Connector, Secure Software Architecture, Component-based Software Architecture, Secure Software Design, Message Communication Patterns, Security Patterns, Model-based Design, UML.

Abstract: This paper addresses secure software architectural patterns designed with secure connectors, where security concerns are encapsulated in secure connectors, separately from application concerns. Because secure software architectural patterns address security and application concerns, the design of the patterns needs to blend those concerns; thus, they can be complicated. Secure connectors can reduce the complexity of the design of secure software architectural patterns by separating security and application concerns. In this paper, secure connectors are designed for secure software architectural patterns by considering the security patterns required by application components and the communication patterns between the components. Secure connectors make the design of secure software architectural patterns more maintainable and evolvable. We have implemented a secure distributed baseball game application using the secure MVC software architectural pattern to validate our research.

1 INTRODUCTION

With security becoming a challenge in concurrent and distributed software applications, secure software architectural patterns address security concerns and application concerns together to facilitate the development of secure software architectures for applications. Secure software architectural patterns have been proposed in multiple studies (Schumacher et al., 2006; Fernandez-Buglioni, 2013), where security concerns are realized with security components such as reference monitors or digital signatures, and application concerns are modeled with application components containing application logic. However, the application logic still needs to coordinate security components for secure communication between application components. Mixing security concerns with application concerns makes secure software architectural patterns more complex, leading to a challenge in the maintainability or evolution of architectures. It is therefore necessary to separate security concerns from application concerns when designing secure software architectural patterns.

In this paper, we propose that secure software architectural patterns be re-designed with secure connectors to reduce the complexity of the architectural patterns. Secure connectors (Shin et al., 2012, 2016a, 2016b, 2017, 2018, 2019, 2021) encapsulate security concerns separated from application components, handling message communication and security between application components. Secure software architectural patterns in this paper are presented with application components dealing with only application-specific logic and secure connectors providing security for secure interaction between application components. Secure connectors can make application components have no direct interaction with security components.

This paper's secure software architectural patterns involve secure blackboard, distributed publish/ subscribe, broker, and model-view-controller (MVC) software architectural patterns for concurrent and distributed applications. A concurrent and distributed baseball game application (Fernandez-Buglioni, 2013) has been implemented to validate our approach using the secure MVC software architectural pattern designed with secure connectors.

In this paper, section 2 describes related work. Section 3 describes secure connectors. Section 4 describes the design of secure software architectural

Shin, M., Kang, T. and Gomaa, H.
Secure Software Architectural Patterns Designed with Secure Connectors.
DOI: 10.5220/0011310500003266
In *Proceedings of the 17th International Conference on Software Technologies (ICSOFT 2022)*, pages 484-491
ISBN: 978-989-758-588-3; ISSN: 2184-2833

patterns using secure connectors. Section 5 describes the validation of our approach, and section 6 concludes the paper.

2 RELATED WORK

2.1 Software Architectures and Secure Connectors

This section describes related work for software architectures and secure connectors. The authors in (Van den Berghe et al., 2017) present existing modeling notations to represent security properties in secure software design models and provides comparative analysis among the notations. In model-driven security (Basin et al., 2011), systems are modeled with security requirements, and the security infrastructures are generated using the models. Model-driven security provides a way to bridge a gap between security and design models of systems.

A distributed component-based software architecture (CBSA) in (Gomaa et al., 2001; Gomaa, 2011) is composed of components and connectors. In (Ren et al., 2005), a connector-centric approach is used to model, capture, and enforce security with software connectors. The methods in (Al-Azzani et al., 2012) propose SecArch to evaluate software architectures with security concerns, which is an incremental evaluation tool for secure architectures.

The work in (Shin et al., 2012) describes secure asynchronous and synchronous connectors for modeling the software architectures for distributed applications and the design of reusable secure connectors. The research in (Shin et al., 2016a, 2016b, 2017) addresses the design of secure connectors in terms of maintainability and evolution for secure software architectures. The study in (Shin et al., 2018, 2019) describes the secure connectors for software product lines. The very recent work of authors in (Shin et al., 2021) extended the secure connectors for complex message communications in software architecture, where secure connectors provide more than one communication pattern. The research in (Gomaa et al., 2010; Albassam et al., 2016) has investigated designing dynamically adaptable and recoverable connectors.

2.2 Secure Software Architectural Patterns

Software architectural patterns (Buschmann et al., 1996; Shaw and Garlan, 1996; Taylor et al., 2010) are recurring architectural styles used in various software applications. Software architectural structure patterns (Gomaa, 2011) address the static structure of the architecture, whereas the architectural communication patterns (Gomaa, 2011) describe the dynamic communication between components.

Security patterns in (Schumacher et al., 2006; Fernandez-Buglioni, 2013) address the broad range of security issues that should be considered in the stages of the software development lifecycle. The authors describe the problem, context, solution, and implementation of security patterns in a structured way with a template.

The secure software architectural patterns for middleware (Schumacher et al., 2006; Fernandez-Buglioni, 2013) providing services to applications have been designed with application and security classes and the relationships among classes in the static model. The objects of the classes and their interaction are modeled in the dynamic model. Security classes model authentication, authorization, and digital signature security services. Application objects contain the sequence logic to interact with security objects directly to achieve security services.

3 SECURE CONNECTORS

A secure connector is a distributed connector consisting of a secure sender connector and a secure receiver connector that communicate with each other. A secure sender or receiver connector consists of a security coordinator, zero or more security pattern components (SPCs), and one or more communication pattern components (CPCs).

3.1 Security Coordinator Components

A security coordinator, which is either a security sender coordinator or a security receiver coordinator, is designed to integrate the communication patterns and security patterns selected for a secure connector. The security sender and receiver coordinators need to be designed for each secure connector whenever one or more CPCs and zero or more SPCs are selected for the connector. A template (Shin et al., 2018, 2019) for the high-level security coordinator can be designed for each communication pattern. The template is customized for each secure connector based on the security pattern(s) selected.

3.2 Security Pattern Components

A security pattern addresses a specific security technique that realizes a security service, which is software functionality for realizing a security goal, such as confidentiality, integrity, or availability. A security pattern is designed using one or two security pattern components (SPCs), as depicted in Fig. 1. For instance, the confidentiality security service can be realized using the symmetric encryption security pattern (Fig. 1a) composed of the symmetric encryption encryptor and decryptor SPCs with their interfaces (Fig.1d). Another example is shown for the authenticator and authorization security patterns, which are realized respectively with the authenticator SPC (Fig. 1b) and the authorization SPC (Fig. 1c) with their interfaces (Fig. 1d). Each port of a component is defined in terms of provided and/or required interfaces (Gomaa, 2011; Rumbaugh et al., 2004). Each security pattern component (Fig. 1) has a provided port through which the component provides security services to other components.

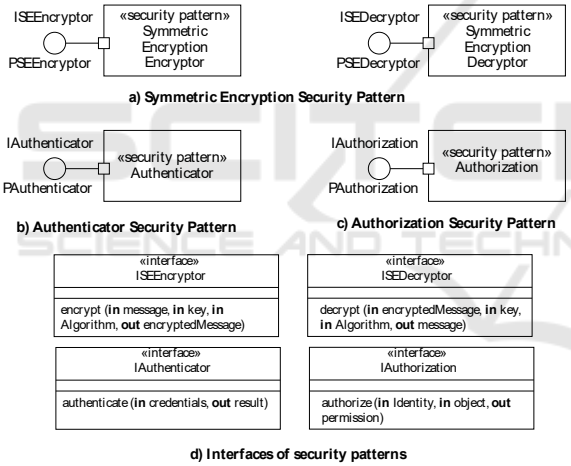

Figure 1: Security Pattern Components and their interfaces.

3.3 Communication Pattern Components

Each communication pattern is designed with a sender communication pattern component (CPC) and a receiver communication pattern component (CPC), encapsulated in a secure sender connector and a secure receiver connector, respectively. In asynchronous message communication, an asynchronous message is sent from a sender component to a receiver component and is stored in a queue if the receiver is busy. The sender component can continue to send the next message to the receiver component until the queue is full. Fig. 2a depicts the Asynchronous Message Communication (AMC) Sender CPC and Asynchronous Message Communication (AMC) Receiver CPC for the secure asynchronous message communication connector. The AMC Sender CPC (Fig. 2a) has the provided PAsyncMCSenderService port through which the Security Sender Coordinator component sends to the AMC Sender CPC a message being sent to the receiver application component. In contrast, it requests a service from the AMC Receiver CPC via the required RAMCNetwork port. Similarly, the AMC Receiver CPC (Fig. 2a) has a required RAsyncMCReceiverService port to communicate with a receiver component, and the provided PAMCNetwork port to receive a message from the AMC Sender CPC. Fig. 2b depicts the interfaces provided by each port of the AMC Sender and Receiver CPCs.

In a similar way, the Synchronous Message Communication with Reply (SMCWR) Sender CPC and Synchronous Message Communication with Reply (SMCWR) Receiver CPC for the secure Synchronous Message Communication with Reply connector are modeled in (Shin et al., 2019).

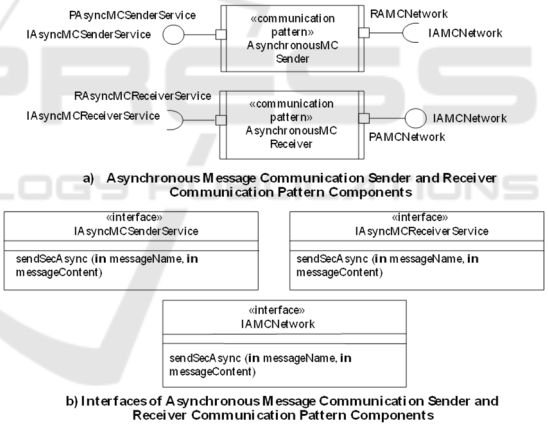

Figure 2: Asynchronous Message Communication Sender and Receiver Communication Pattern Components and their Interfaces.

4 DESIGN OF SECURE SOFTWARE ARCHITECTURAL PATTERNS

4.1 Secure Blackboard Software Architectural Pattern

The secure blackboard software architectural pattern (Fernandez-Buglioni, 2013) provides security measures for the data stored in the blackboard

component to prevent it from being compromised. Any knowledge source can update the data in the blackboard if the control component does not check the rights of knowledge sources to change the data. The control component is a gateway to access the blackboard component, authenticating knowledge sources to access the blackboard to update the data. The components in the secure blackboard software architectural pattern communicate with each other via secure channels to avoid data breaches in communication. The blackboard component is necessary to record all knowledge sources' actions to the data for auditing.

The secure blackboard software architectural pattern (Fernandez-Buglioni, 2013) can be designed with secure connectors that contain authentication, authorization, and encryption security patterns required by the pattern, separately from application components to reduce the complexity of the pattern. Fig. 3 depicts the structural view of the interaction between Knowledge Source and Control components in the secure blackboard software architectural pattern, designed with a secure AMC connector to apply an operation to the blackboard asynchronously (Fernandez-Buglioni, 2013). A Knowledge Source component sends the Control component a message containing its credentials, role, data, and an operation to be taken on the blackboard. With the message received from the Knowledge Source component via a provided PSecAsyncSenderService port, the Security Sender Coordinator in the secure AMC sender connector invokes the Symmetric Encryption Encryptor component (Figs. 1 and 4) via a required RSEEncryptor port to encrypt the message. And it then sends the encrypted message to the AMC Sender Communication pattern component via a required RAsyncMCSenderService port. The AMC Sender communication pattern component sends the encrypted message via a required RNetwork port to the AMC Receiver communication pattern component, which forwards the message to the Security Receiver Coordinator via a required RSecurityService. The message is decrypted by the Symmetric Encryption Decryptor component (Figs. 1 and 4) using a secret key retrieved by the Security Receiver Coordinator from the Control component via a required RSecAsyncReceiverService port. The Authenticator component (Figs. 1 and 4) checks the decrypted credentials to identify the Knowledge Source component. The Authorization component (Figs. 1 and 4) verifies that the Knowledge Source component has a right to access and operate data in the blackboard. Fig. 4 depicts the security sender and receiver coordinators in the secure sender and receiver connectors for Knowledge Source and Control components and their interfaces.

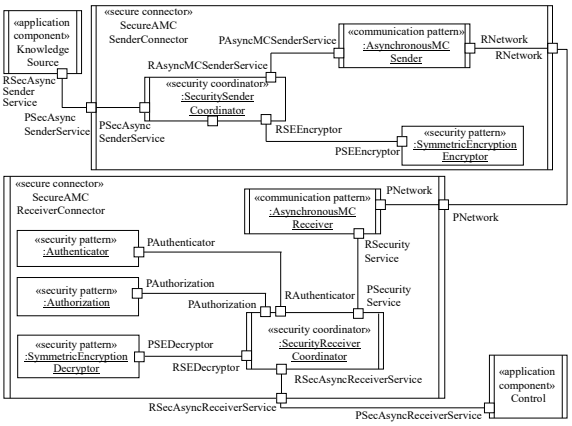

Figure 3: Knowledge Source and Control Components designed with Secure Connectors in the Secure Blackboard Architecture.

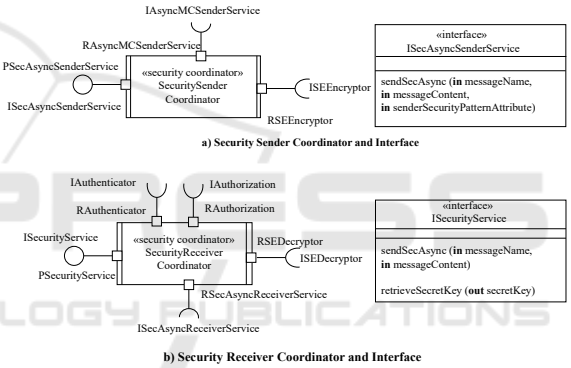

Figure 4: Security Coordinator Components of Secure Connectors for Knowledge Source and Control components and Interfaces.

4.2 Secure Publish/Subscribe Software Architectural Pattern

The secure publish/subscribe software architectural pattern (Fernandez-Buglioni, 2013) addresses the security measures for secure subscription and notification in the pattern. A publisher must check each subscriber's identity who joins a group to prevent imposters from receiving event notification messages. A subscriber should verify that each notification is a genuine message received from the publisher it has registered. An attacker can read messages transmitted between a publisher and subscribers; thus, the messages must be sent or received through a secure channel.

The secure publish/subscribe software architectural pattern can be designed with secure connectors that encapsulate security patterns

separately from the publisher and subscriber application components. Fig. 5 depicts the secure publish/subscribe software architectural pattern designed with a secure AMC connector through which a publisher component securely notifies a subscriber component of event messages. The secure AMC sender and receiver connectors are designed with Symmetric Encryption (Fig. 1) and Digital Signature security patterns. The Symmetric Encryption security pattern encrypts the event notification messages sent by the publisher component to the subscriber component for the confidentiality of the messages in communication. The Digital Signature security pattern is utilized for the subscriber component to verify the publisher component's authenticity.

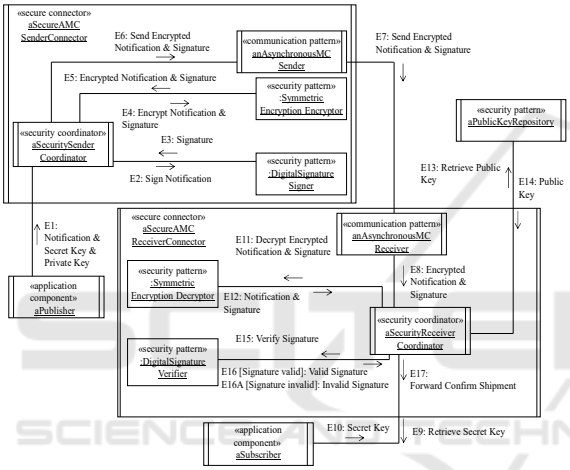

Figure 5: Publisher and Subscriber Components designed with Secure Connectors in Secure Publish/Subscribe Architectural Pattern.

Fig. 5 depicts the UML communication diagram that describes the dynamic behavioral view of event notification of a publisher component to a subscriber component. When the publisher component sends a notification to the subscriber component, the notification is signed by the Digital Signature Signer component in the secure AMC sender connector. The notification and signature are then encrypted by the Symmetric Encryption Encryptor component (Fig. 1) in the secure AMC sender connector. The encrypted notification and signature are sent to the subscriber component via the secure AMC receiver connector and then decrypted by the Symmetric Encryption Decryptor component (Fig. 1) using a secret key retrieved from the subscriber component. The secure AMC receiver connector requests the publisher component's public key from the Public Key Repository component, designed for a certificate authority in the public key infrastructure. The signature is verified by the Digital Signature Verifier component in the secure AMC receiver connector using the publisher's public key. Fig. 6 depicts the security sender and receiver coordinator components and interfaces specification, respectively, in the secure sender and receiver connectors, designed for event notification messages between the publisher and subscriber components. We can specify the pseudocode for the Security Sender and Receiver Coordinator components (Shin et al., 2019) in the secure connector.

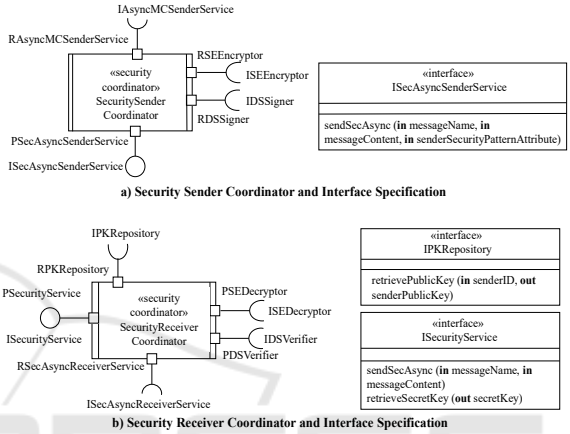

Figure 6: Security Coordinator Components of Secure Connectors for Publisher and Subscriber Components and Interfaces.

4.3 Secure Broker Software Architectural Pattern

The secure broker software architectural pattern requires authentication, access control, and encryption security (Fernandez-Buglioni, 2013) for securely brokering the interactions between clients and servers. To avoid identity spoofing, a secure broker needs to provide mutual authentication between clients and servers. A client's access to a server should be controlled so that only a client with a right is allowed to access the server. All messages communicated among clients, broker, and servers necessitate cryptographic encryption to prevent attacks.

The secure broker software architectural pattern can be designed with secure connectors containing authentication, authorization, and encryption security patterns, separately from client, broker, and server components. The secure (receiver) connector (Fig. 7) identifies a client's credentials using the Authenticator security pattern (Fig. 1), controlling the client's access to a server's service using the

Authorization security pattern (Fig. 1). The Symmetric Encryption security pattern (Fig. 1) encrypts a client's service request being sent by a client component to the broker and a server's reply forwarded by the broker to the client component.

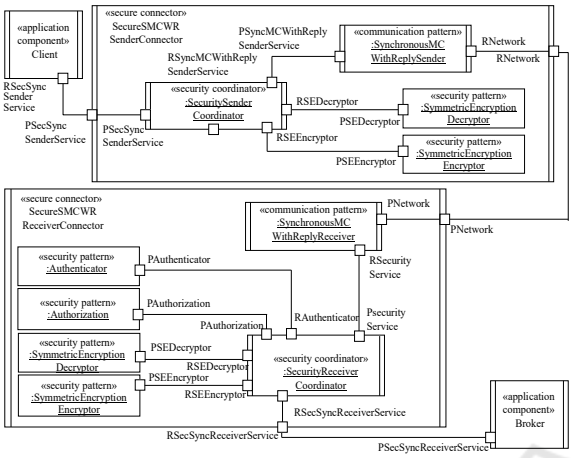

Figure 7: Client and Broker Components designed with Secure Connectors in Secure Broker Architectural Pattern.

Fig. 7 depicts the structural view of secure broker forwarding in the secure broker architectural pattern, designed with secure SMCWR sender and receiver connectors for the client and broker components, respectively. When the client component sends a service request to the broker, the service request with the client's credentials and role are encrypted by the Symmetric Encryption Encryptor SPC in the secure SMCWR sender connector. The encrypted service request is decrypted by the Symmetric Encryption Decryptor SPC in the secure SMCWR receiver connector, and then the credentials are verified by the Authenticator SPC. The Authorization SPC determines whether a client component has a right to access a server. If all security checks are valid, the service request message is sent to the broker. A reply received from a server is encrypted by the Symmetric Encryption Encryptor SPC in the secure SMCWR receiver connector. The encrypted reply is decrypted by the Symmetric Encryption Decryptor SPC in the secure SMCWR sender connector and then sent to the Client component.

4.4 Secure MVC Software Architectural Pattern

The secure Model-View-Controller (MVC) software architectural pattern (Fernandez-Buglioni, 2013) is required to maintain an acceptable level of security among its components against threats, so it needs authentication, encryption, authorization, log records, and input validation. Authentication is necessary to verify that the remote users to access the controller component are authentic. The secure MVC software architectural pattern needs encryption to protect the data transit between components against eavesdropping. The secure MVC software architectural pattern should allow only authorized users to change sensitive data in the model. The model component necessitates recording all accesses to sensitive information for auditing. The user input to the controller component is necessary to be neutralized to clean corrupted input.

Fig. 8 depicts the secure interaction between the Model and View components designed with a secure connector in the secure MVC software architectural pattern. The Model and View components communicate via the secure MV sender and receiver connectors that encapsulate the security patterns separately from the application components. The Model component has a provided and required PRSecSenderService port through which it sends a data change notification message and receives a view update confirmation message to/from the secure MV sender connector asynchronously. The connector communicates those messages with the secure MV receiver connector via a required RAMCNetwork port and a provided PACMNetwork port. The secure MV receiver connector for the View component retrieves the data changed in the Model component through a required RSMCNetwork port that connects to a provided PSMCNetwork port of the secure MV sender connector. The View component has a provided and required PRSecReceiverService through which it communicates the data change notification, changed data, and view update messages with the secure MV receiver connector.

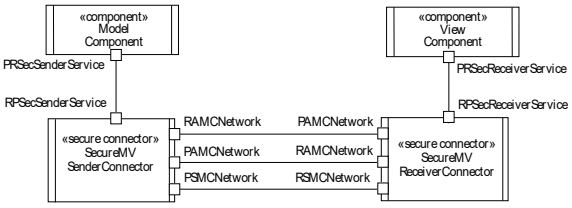

Figure 8: Model and View Components designed with Secure Connectors in Secure MVC Architectural Pattern.

The internal structural view of the secure MV sender connector (Fig. 8) is modeled in Fig. 9, where the model component sends messages to and receives them from the view component to realize the "change propagation" use case in the secure MVC software architecture pattern. The secure MV sender connector (Fig. 9) is designed with three CPCs: an AMC Sender,

an AMC Receiver, and an SMCWR Receiver CPCs; and three security pattern components (SPCs) required by the Model component, which are an Authorization, a Symmetric Encryption Decryptor, and a Hashing Signer SPCs. The Security Sender Coordinator component and its interfaces to the Model component is specified in (Shin et al., 2021).

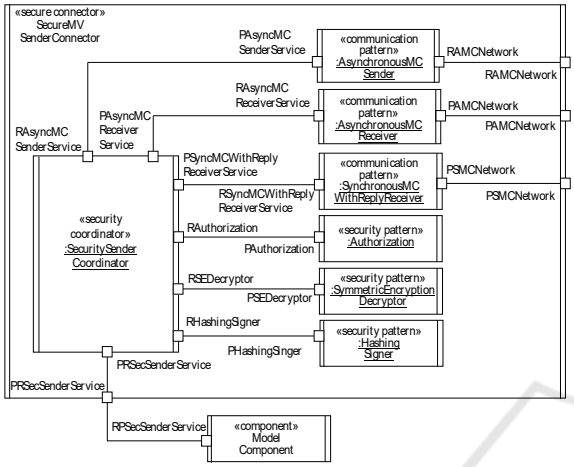

Figure 9: Secure MV Sender Connector for Model Component.

5 VALIDATION

To validate this research, we implemented the secure MVC software architectural pattern designed with secure connectors (in section 4.4) that consist of one or more security pattern components and more than one communication pattern component. The secure connectors for the secure MVC software architectural pattern are a reusable form of secure connectors. The secure connectors for the secure blackboard, publish/subscribe, and broker software architectural patterns (in section 4) are designed with only one communication pattern (i.e., either synchronous or asynchronous communication pattern). However, the secure MVC software architectural pattern necessitates the secure connectors that provide more than one communication pattern component for application components

For validation, a distributed baseball game application (Fernandez-Buglioni, 2013) was developed using the secure MVC software architectural pattern, structured into the user interface, controller, model, and view components. A scorer entered or updated the game scores through the Android App-based user interface component, which ran on smartphones, separated from the remote controller component that received the scorer input and delivered it to the model component. The game scores were stored and maintained in the model component on a server implemented using the Spring Boot framework. When the game scores in the model component were changed, the model component notified the controller and view components of the changed game scores. The view component implemented using Android App running on fans' smartphones read the game scores from the model component in the game server and redisplayed them to fans. The controller component also enabled or disabled user interface menu functions according to the changed game scores.

The validation was conducted by designing and implementing three secure connectors among the scorer interface component, controller component, model component, and fan's view component in the secure MVC-based baseball game application. For the secure connector between the model component and view component (Figs. 8 and 9), We implemented the secure sender connector with three communication pattern component (CPC) threads in Java: AMC sender CPC, AMC receiver CPC, and SMCWR receiver CPC threads, and the secure receiver connector with three communication pattern component (CPC) threads in Java: AMC receiver CPC, AMC sender CPC, and SMCWR sender CPC threads. We also implemented the Symmetric Encryption Encryptor and Decryptor security pattern components (SPCs) using the Data Encryption Standard (DES) algorithm and the Hashing Signer and Hashing Verifier security pattern components (SPCs) using the Secure Hash Algorithm (SHA). Authorization SPC was implemented to verify the validity of the view component's role ID. The security sender coordinator and the security receiver coordinator were implemented using each thread to integrate the CPC threads and SPCs.

Similarly, we implemented the secure connector between the controller component and model component, and the secure connector between the scorer interface component and controller component.

6 CONCLUSIONS

This paper has described the secure blackboard, publish/subscribe, broker, and MVC software architectural patterns, designed with secure connectors that encapsulate security components separately from the application components in order to reduce the complexity of the architectural patterns. The secure connectors for the architectural patterns

were designed with zero or more security patterns required by application components, one or more communication patterns to realize message communication between the components, and security coordinators to integrate security and communication patterns. The secure software architectural patterns were designed using a component-based model depicting the component's ports, interfaces, and connectors. Secure connectors make the secure software architectural patterns more maintainable and evolvable. We also implemented a distributed baseball game application using the secure MVC software architectural pattern designed with secure connectors to validate our approach.

We envision future work to extend this research. We can extend our approach to other secure software architectural patterns. The secure connectors designed for secure software architectural patterns could also be further validated using model checkers to check correctness, deadlock, and security properties. Our validation would thus become more concrete with model checking. In addition, we could investigate the secure connectors adaptable to changing communication patterns and security patterns at runtime. Moreover, we could extend this research to designing secure connectors that recover the failures to communication or security.

REFERENCES

Al-Azzani, S. and Bahsoon, R., 2012, August. SecArch: Architecture-level evaluation and testing for security. In Software Architecture (WICSA) and European Conference on Software Architecture (ECSA), 2012 Joint Working IEEE/IFIP Conference on (pp. 51-60).

Albassam, E., Gomaa, H. and Menascé, D.A., 2016, July. Model-based Recovery Connectors for Self-adaptation and Self-healing. In ICSOFT-EA (pp. 79-90).

Basin, D., Clavel, M. and Egea, M., 2011, June. A decade of model-driven security. In Proceedings of the 16th ACM symposium on Access control models and technologies (pp. 1-10). ACM.

Buschmann, F., Meunier, R., Rohnert, H., Sommerlad, P., 1996. Pattern Oriented Software Architecture: A System of Patterns, John Wiley & Sons.

Fernandez-Buglioni, E., 2013. Security patterns in practice: designing secure architectures using software patterns. John Wiley & Sons.

Gomaa, H., Menascé, D.A. & Shin, M.E., 2001. Reusable component interconnection patterns for distributed software architectures. Proceedings of the 2001 symposium on Software reusability putting software reuse in context - SSR 01.

Gomaa, H., Hashimoto, K., Kim, M., Malek, S., and Menascé, D.A., 2010, March. Software adaptation patterns for service-oriented architectures. In Proceedings of the 2010 ACM Symposium on Applied Computing (pp. 462-469). ACM.

Gomaa, H., 2011. Software modeling and design: UML, use cases, patterns, and software architectures. Cambridge University Press.

Ren, J., Taylor, R., Dourish, P. and Redmiles, D., 2005, May. Towards an architectural treatment of software security: a connector-centric approach. In ACM SIGSOFT Software Engineering Notes (Vol. 30, No. 4, pp. 1-7). ACM.

Rumbaugh, J., Booch, G., and Jacobson, I., 2004. The Unified Modeling Language Reference Manual, Addison-Wesley.

Schumacher, M., Fernandez, E.B., Hybertson, D., Buschmann, F. and Sommerlad, P., 2006. Security Patterns, Wiley.

Shaw, M., and Garlan, D., 1996. Software Architecture: Perspectives on an Emerging Discipline, Pearson.

Shin, M.E., Malhotra, B., Gomaa, H. and Kang, T., 2012, July. Connectors for Secure Software Architectures. In SEKE (pp. 394-399).

Shin, M.E., Gomaa, H., Pathirage, D., Baker, C. and Malhotra, B., 2016. Design of Secure Software Architectures with Secure Connectors. International Journal of Software Engineering and Knowledge Engineering, 26(05), pp.769-805.

Shin, M., Gomaa, H. and Pathirage, D., 2016, June. Reusable Secure Connectors for Secure Software Architecture. In International Conference on Software Reuse (pp. 181-196). Springer, Cham.

Shin, M., Gomaa, H. and Pathirage, D., 2017. Model-based Design of Reusable Secure Connectors. In 4th International Workshop on Interplay of Model-Driven and Component-Based Software Engineering (ModComp) 2017 Workshop Pre-proceedings (p. 6).

Shin, M., H. Gomaa, and D. Pathirage, 2018, July. A Software Production Line Approach for Feature Modeling and Design of Secure Connectors. In 14th International Conference on Software Technologies (ICSOFT2018), Porto, Portugal (Best paper award).

Shin, E. M., H. Gomaa, D. Pathirage, 2019, August. A Software Product Line Approach to Design Secure Connectors in Component-Based Software Architectures. In Communications in Computer and Information Science, Springer, Communications in Computer and Information Science book series (CCIS, Volume 1077).

Shin, M., Kang, T., and Gomaa, H., 2021. Design of Secure Connectors for Complex Message Communications in Software Architecture, 2nd European Symposium on Software Engineering, Nov. 19-21, Larissa, Greece.

Taylor, R. N., Medvidovic, N., and Dashofy, E., 2010, Software architecture: foundations, theory, and practice. Wiley.

Van den Berghe, A., Scandariato, R., Yskout, K. et al., 2017. Design notations for secure software: a systematic literature review, Software & Systems Modeling 16, 809–831.