WHITE PAPER

Security Basics for Financial Applications

Abstract

Security is the principal requirement for online financial applications. Data privacy, customer trust, and long-term growth all depend on how secure a financial application is. As these applications are accessed from various devices and through numerous channels, financial organizations strive hard to implement a foolproof security system. In this white paper, we will discuss the core security measures that can be considered while building financial applications. We will start with core design concepts for financial applications, move on to the different security techniques and best practices, and finally, provide a basic security design for financial applications. The financial applications referred in this white paper include web applications, financial portals, and other finance domain-related online applications.

Infosys
be more

Financial applications

Finance applications include applications performing financial transactions such as online banking portals, online insurance applications and such for which security is a prime concern. Most of the e-commerce and retail applications invariably deal with payment transactions and hence security would be an important feature of these applications as well. Online finance applications face a host of threats such as identity theft, session hijacking, password hacking etc. which has long term impact on revenue and user trust.

Online financial applications provide a variety of features, such as dashboard views, reports, personalized customer pages, and for particular financial domains, the required information aggregation as well. They are integrated with core banking systems or finance enterprise applications, to provide domain-specific features. Additionally, they also offer a wide range of personalization and customization features that enable finance organizations to launch personalized campaigns and features for targeted segments.

Core security concerns of financial applications

The key security aspects in financial applications include secure authentication, authorization, data encryption, transport-level security, role-based access and robust permission models, data privacy and integrity, and security extensions.

Key security vulnerabilities and ways to tackle them

Based on our experience in financial applications, we have created a list of the common security challenges, the details of security vulnerabilities, and the effective measures to address them:

Multiple sessions

Financial applications usually do not allow multiple sessions due to security and data-integrity concerns. Using a combination of the following approaches should restrict these multiple sessions:

- Create session filters to intercept every user request, and use a database-driven table to check the multiple session information, in order to restrict the user sessions based on the session data
- Restricting the user sessions at the server side: Server modules (such as core banking modules) keep track of user

Combining these two approaches allows a truly single session-based implementation.

Man-in-the-middle attacks and session hijacking

In this kind of attack, the hacker may intercept traffic between the requestor and the finance portal. The first step in preventing such an attack is to use a secure transport layer, such as HTTPS, for all secure interactions. The rule applies for finance services as well. In addition to this, we must encrypt secured data (such as user information, finance information) during transit, and decrypt before rendering to the client.

Request spoofing and cross-site request forgery (CSRF)

In such a vulnerability, the attacker may send a forged request to the server. The attacker can gain access to request parameters using techniques such as snooping, and can then construct an attack-script to make the portal believe that the request is coming from a genuine source. For instance, if the attacker obtains the session ID or is able to intercept the request, he / she can use the session details to initiate a financial transaction. An effective way to prevent this attack is to use a security token with each request that is validated on the server-side. We will learn more about this technique and its implementation in the following pages.

Injection attacks

The attacker can use SQL injection techniques to gain information access. The vulnerability can be exploited by appending SQL keywords and comments (such as appending '1=1' to the query string).

In order to effectively mitigate this, we need to:

- Validate all user input on the client as well as the server-side and maintain a blacklist of characters for this validation
- Encode or remove HTML and SQL-reserved characters
- Use prepared statements instead of direct SQL commands in the application. Use object-relational mapping (ORM) tools, such as Hibernate,for database interactions

Key security features of online finance applications

Besides core security features such as authentication, authorization, single-sign-on, session management, account management, financial application should also deal with other security such as follows:

Federated identity and access management

The identity and access management can be federated across the domain, and share the same identity store and access manager for all user groups. This strategy works best in subsidiaries and acquired entities.

Strong nonrepudiation using DS

Nonrepudiation means that a party cannot deny the authenticity of their signature upon the sending of a message.

In the digital world, nonrepudiation can be achieved through digital signatures.

Nonrepudiation occurs based on the following two criteria– The user is the legitimate identity that sent the content/ performed the transaction;the content/ transaction details are not modified in the middle, as data integrity is ensured through a hashing algorithm.

Crypto libraries

Crypto libraries are the cryptographic libraries used in Internet standards to provide encryption algorithms.

The functionality includes key generation algorithms, key exchange agreements, and public-key cryptographic standards. For finance applications, it is recommended to use a salted password-hashing algorithm as discussed below.

Endpoint security – Anti-malware / virtual keyboards.

Endpoint security refers to protecting the endpoint device to comply with anti-malware and virus protection to prevent, detect, and remediate any malicious programming on the system. This helps in restricting any malicious program that might be tracking keystrokes to obtain sensitive information.

Virtual keyboards help in reducing the risk of key loggers logging keystrokes, as the clicks happen only on the virtual keyboard to input the data. This also makes it more difficult for malware programs to track the clicks and obtain the input data. However, there is a possibility that the malware can take screenshots upon each click and hence, the endpoint should be secure and updated with respect to anti-malware programs as well.

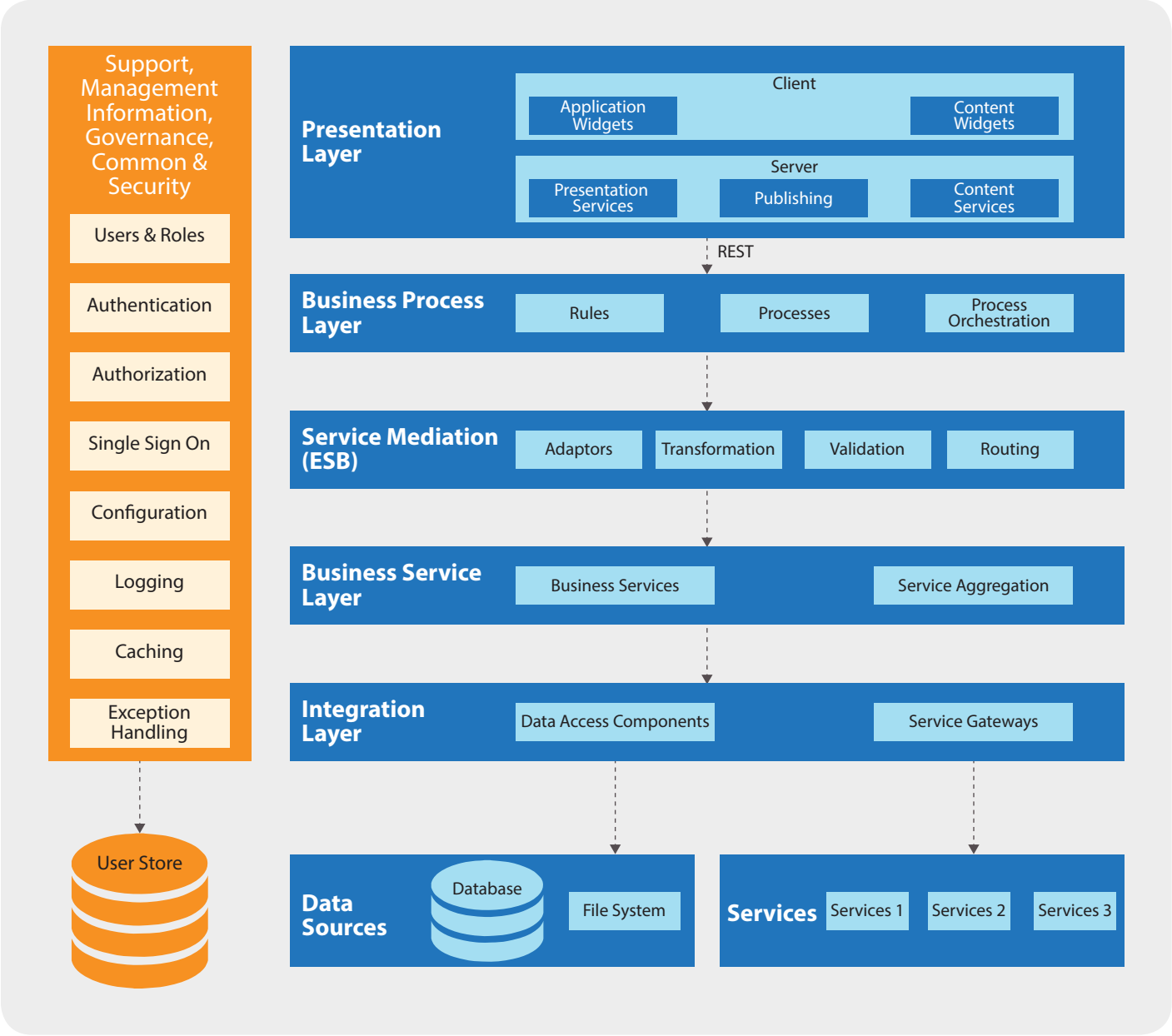
Financial application key design considerations

Listed below are some of the key design aspects that need to be kept in mind while developing finance applications. These are additional design factors along with security.

- Open standards-based technology and integrations – This includes using standards related to HTML, CSS, and accessibility to name a few
- Layered architecture using MVC pattern – This provides a clear layer-wise separation of components with each layer handling a distinct responsibility. MVC enables loose coupling, separation of concerns, and flexibility to change the components in each layer independently
- Modular and extensible component design –Each of the solution components will be designed such that it can be reused for future needs
- Adoption of services-oriented architecture for integration – An ESB middleware can be used to handle complex and multiple services and enable service-oriented integration between different banking systems Leveraging open-source technologies wherever applicable
- Continuous build and integration approach for execution –Tools such as Jenkins Continuous Integration can be used to maintain build quality
- Performance, availability, and scalability –Performance should be thought through, right from the component design to the performance testing stage. In addition to performance-based design, other performance optimization techniques can be adopted, including:
    - The solution will also be tested iteratively to ensure that the desired performance service-level agreement (SLA) is met
    - Scalability can be achieved by using the appropriate infrastructure and hardware
    - Incorporation of a governance model to proactively check the heartbeat of the systems to ensure system availability and uptime
- Reusability and automation: Reusing the existing components and frameworks will profoundly impact developer productivity, faster time-to-market, and significantly increase the overall quality. Based on the given business requirements, the following components are marked for reuse, partially/completely:

High Level architecture of finance applications

A high level overview of a sample finance web application is shown below:



A typical, n-tier MVC architecture for finance applications has various components, with MVC architecture providing separation of concerns for the various layers. Service-oriented integration is the de-facto standard for integration here. Presentation layer consists of financial widgets, portlet, reports, and handle other presentation concerns. Business layer processes business logic, rules and business processes. Typically a message oriented middleware such as ESB would be used for service mediation and to handle concerns such as routing, protocol transformation, validation etc. Business layer exposes various business services and integration layer integrates with necessary enterprise interfaces.

The different security aspects of financial applications

After having discussed core security concerns, let us deep-dive into the other security aspects of financial applications. Here, we provide comprehensive coverage of security techniques and proven methodologies to effectively address security issues. Let us start by looking at core security features required in a typical financial application:

Authentication

Authentication is all about allowing a user to login based on a username and credentials. However, would that be enough for a typical banking or financial application?
In banking applications, the primary authentication mechanism should not stop at validating the user credentials. So what else would be required?

Password policy

Let us go with an encryption algorithm to encrypt and store the password. SHA, MD5 are some of the encryption algorithms that generate an encrypted password of the user and store them in the database. We all agree that it will never be possible to decrypt the encrypted password from these secure encryption algorithms, but hackers will not try the obvious. Would they? They would try to identify any loopholes in these secure encryption algorithms and exploit them instead. The authentication mechanism should thus be foolproof against brute force attacks. If you wonder what brute force attack is, it is a trial-and-error method used by hacker application programs to decode encrypted data such as passwords.

For the same password input, these encryption algorithms generate the same encrypted value. Thus, it is easy to maintain a lookup table of all possible combination of passwords. In some cases, if not most, the user's password will be one of a dictionary word. Hackers can thus create a lookup table for each dictionary word with a mapping encrypted password. If they gain access to the encrypted password, they can easily crack the password using this lookup table.

Although this vulnerability can be limited to a certain extent, by introducing stronger password policies that restrict the use of direct dictionary words. Hackers have gone a step further already– they have begun to suffix, prefix, or insert the possible special characters, numeric values, as well as the appropriate casing around it. They are thus able to create the lookup table regardless.

What is the solution now?

Salted Password Hashing

Salted password hashing algorithms– these algorithms, such as the BCrypt Password Encoder, generate different encryption values for the same password input. How is it possible? For every password, the algorithm generates a random 'salt' that is inserted to the original password before it is encrypted. Thus, different encryption values are generated for the same password input, and lookup tables can never be created for such algorithms. The salt is embedded along with the encrypted value, and only the algorithms themselves can gain access to the salt, and validate the supplied password. And these can never be decrypted.

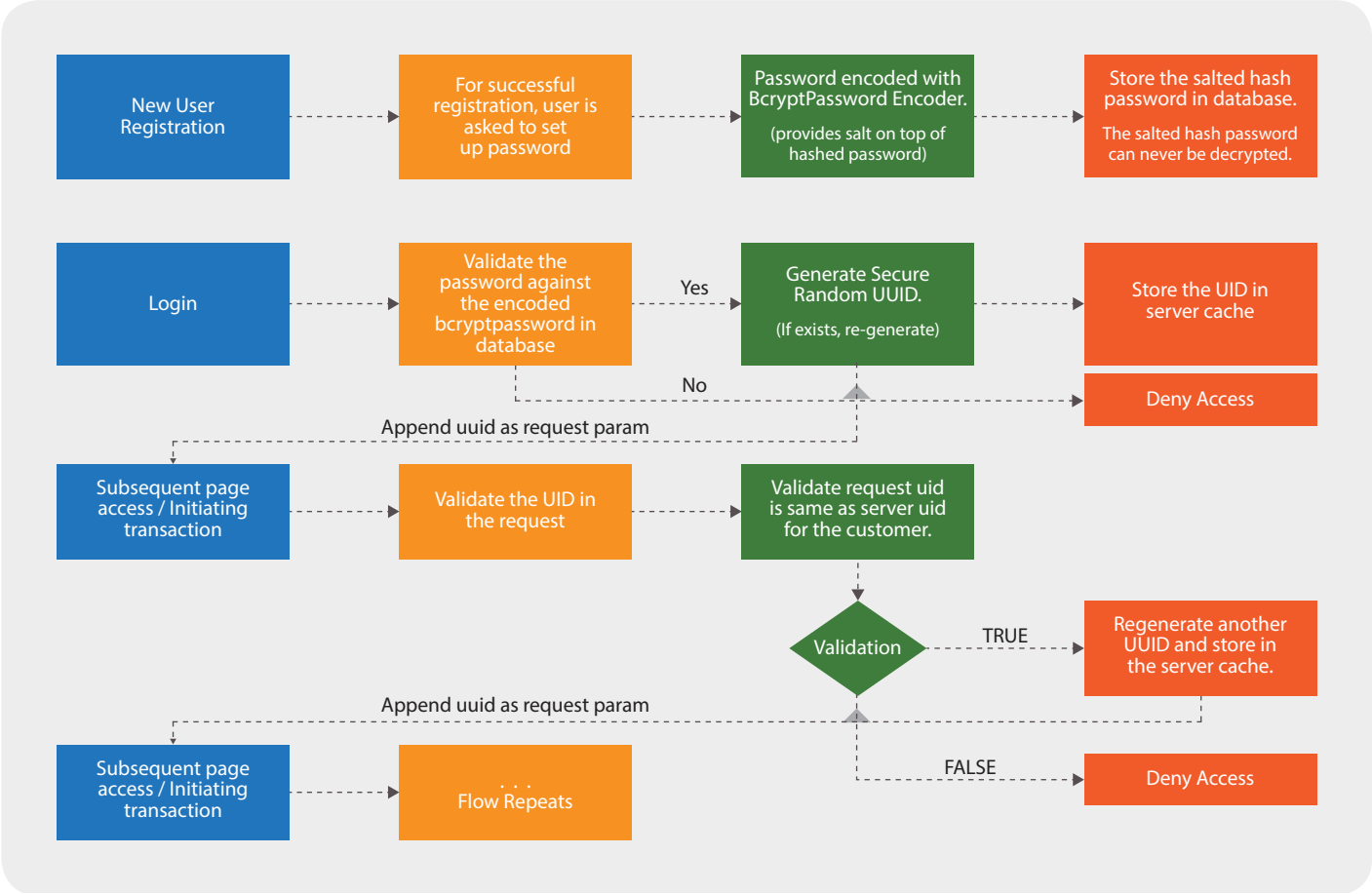
Request identification

Banking applications need to go that one-step extra – they should also authenticate each request and authenticate the request even after post-login. This will prevent any session hijack. Now how is this achieved?

Immediately after validating user credentials, a unique identifier (let us call it a Request ID) can be generated and mapped to the customer ID in the server cache or persistence storage. The same value would be returned to the portal client, which would be appended to this Request ID for the subsequent requests. The request filter would then validate the Request ID sent by client, against the unique identifier stored in the server cache or persistence storage. If they match the request is allowed, and another Request ID is generated and sent to the portal client, which also stores the new unique identifier in server cache or persistence storage. This process repeats for every request, enhancing the security measures at each step.

- Session hijacking is not possible as every request needs to have the Request ID,which the server expects from the portal client
- The refresh, back, and forward navigation within the history is not possible on browsers as the session is destroyed when a valid unique identifier cannot be supplied to the server for validating the request
- Sophisticated attacks like CSRF (Cross Site Request Forgery) can also be prevented using Request IDs
- Once again, browser refresh is not possible, and a user cannot resubmit the same request, as the valid unique identifier cannot be supplied to the server for validating the request
- Simultaneous sessions can be prevented in the same way, as the new session cannot provide the Request ID the server expects – thus, no new session can be established. If security is highly critical, both simultaneous session can be destroyed, prompting users to authenticate themselves again

A sample secure user-registration and user-authentication flow is shown in the following diagram:



Authorization

Authorization implements fine-grained access control by providing role-based access to resources (such as functions, pages, and widgets).

- Access manager acts as a centralized control center to enforce all authorization policies and rules
- Once the authorization policies are defined in the access manager, the request to access the application is allowed only if the user has privileges
- Identity Governance Administration can be used to govern the degree of access each identity (user) has to each aspect / module of the application
- The application can be integrated with the access manager to check for user privileges
- Access privileges to internal resources within the application can be maintained within the application database itself

Identity and Access Management (IDAM)

- The finance application can be integrated with the identity and access management application to enable single sign-on, as well as role-based access
- Identity Management: For a newly joined employee, the identity can be created through IDAM, and the thus-created user would flow to the user store. A random password can also be pushed through IDAM to the user store. If security is the primary concern, then half of the password can be sent to his reporting manager, and the remaining half to the HR manager. The new employee can collect the password fragments from both, login to the corporate system, and then be required to change the password
- Application Access Management: The access provided to each user can be configured in the IDAM using two approaches:
  - Integrating the application with the user store authentication provider
  - Where productized solutions adopted for the finance application depend on a user table in the database for all their operations, we have two additional options to achieve the goal:
    - API access: The user can be created in the application using the API call from the IDAM. For this to be feasible, the application should expose an API, through which the user table can be updated
    - Database access: IDAM would have to access a database view, with write access to the users table, and be able to create or update the users in the table.
- All security aspects to access the user management API or user database should be established to ensure identity management is secure
- Fine-grained access management: The application has greater control on the internal resources each user can access.

  How can an IDAM provide this kind of fine-grained access management?

  It can be achieved by updating the user and role-mapping table of the application either via API, or directly in the database table from the IDAM. As mentioned earlier, all security aspects for accessing the API or the database should be established to ensure identity management security

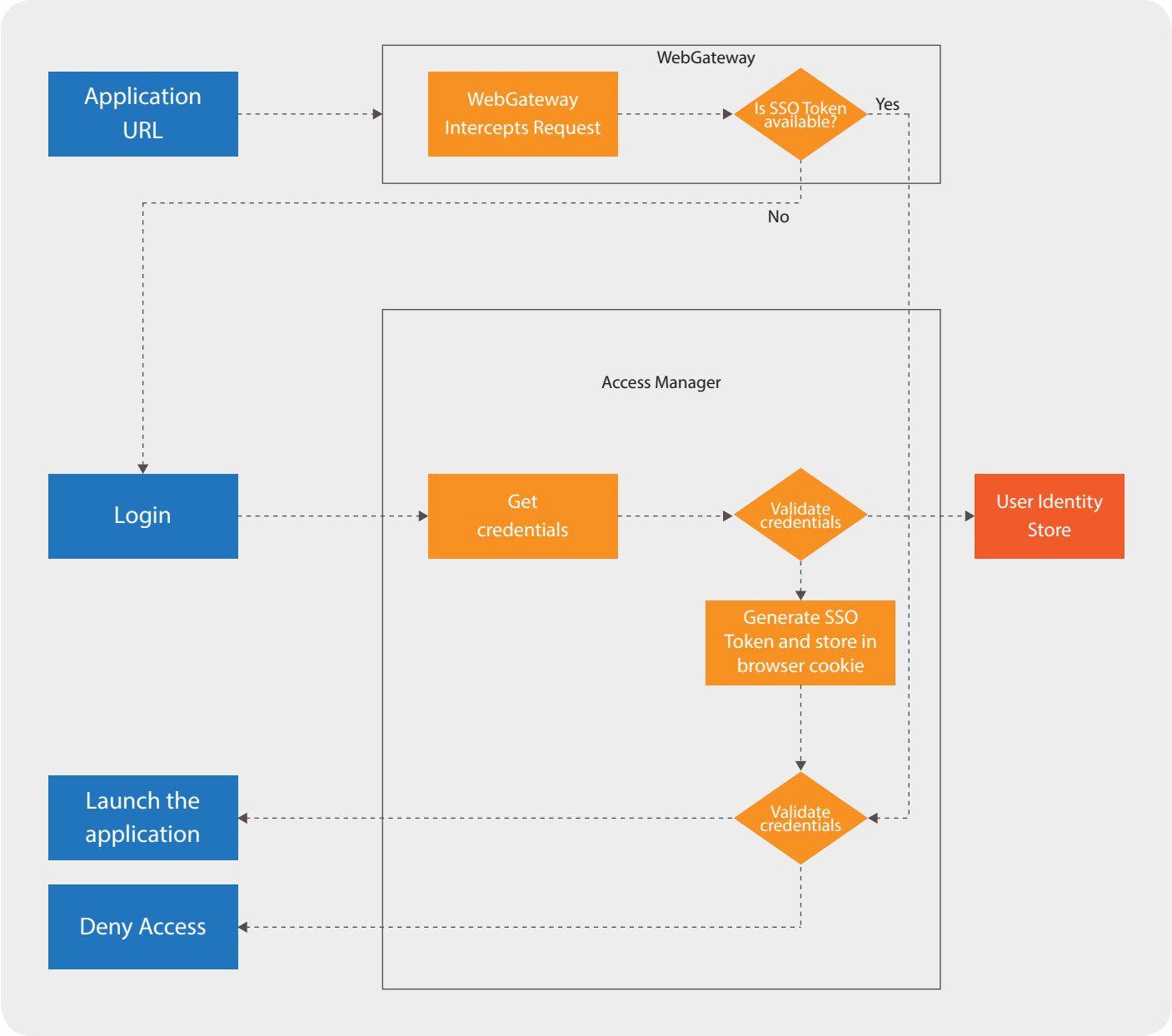
Single sign-on (SSO)

Banking application can have several internal applications that are managed by internal users. If the user has the privilege to access these applications, single sign-on becomes a handy option as he/she does not have to enter separate credentials for accessing each application.

When the user logs into one of the applications, the user credentials can be sent to the access manager, which will verify the credentials from the user identity store. After validating the credentials, the access manager generates the SSO token and sends it to the client browser cookie. The SSO token then remains in the browser cookie and enables the user to login to other applications seamlessly.

Additionally, a webserver security plug-in can be installed on the webserver, which will intercept the request and redirect the user to the login page if the SSO token parameter is absent in the request. If the SSO token exists, it will be passed to the access manager for validation. Once the SSO token is authenticated and the user has the right privileges, the access manager provides access to the application seamlessly.

The authorization policies would also be defined in the access manager and the request to access other application would be allowed only if the user has the privilege. A sample SSO flow is shown in following diagram:



Two-factor authentication is needed to step up the primary authentication along with a supplementary authentication for financially critical operations. In some cases, the regulatory body makes two-factor authentication mandatory for transactions such as fund transfers.

Two-factor authentication can be implemented with any fool-proof mechanism, such as -

- One-Time Password (OTP) – When the transaction is initiated, the application calls the OTP server through an ESB service call .The OTP server then connects with the SMS gateway to send the generated OTP to the customer's device. Once the customer enters the OTP, it is validated again by the application through a service call to the OTP server via the ESB. Once the OTP is authenticated, the transaction is allowed to proceed.
- Security questions – When the transaction is initiated, the application prompts few security questions that have already been setup by the user. Once the security answers are validated, the transaction is allowed to proceed.
- Hard tokens – When the transaction is initiated, the application prompts for a token number generated on the hard token held by the user. The token is authenticated and then the user is allowed to proceed with the transaction.

Alerts and notification

Alerts and notifications are also a key part of the security measures that banks take. In case of a fund transfer, beneficiary addition, modification, or deletion, the customer should be alerted through SMS and Email. Generally, banks provide a cooling period to perform a fund transfer, when a beneficiary is added. The user can perform a fund transfer only after this cooling period.

The alert mechanism ensures that the customer remains informed of all the critical events that can have a financial impact, such as beneficiary addition, fund transfer, change password/security questions and answers, and eDemand draft. This can help the customer react quickly if the account is compromised in any way – for example, a lost ATM card or an unauthorized swipe.

The notification mechanism keeps the customer informed of future events that may require some action to be taken. For example, a password expiry notification or renewal notification, through SMS or Email, can keep the customer informed and prepared to act.

About the Authors

Shailesh Kumar Shivakumar

Infosys

Shailesh Kumar Shivakumar has over 14 years of industry experience. He currently works as a Senior Technology Architect at Digital Practice in Infosys Technologies. His areas of expertise includeEnterprise Java technologies, portal technologies, web technologies, and performance engineering. He has published two books related to enterprise web architecture,enterprise portals, and UXP and has four patent applications. He has published several papers and presented talks in IEEE conferences in the areas of web technologies and performance engineering. He has successfully lead several large-scale enterprise engagements for Fortune 500 clients.

Babu Krishnamurthy

Infosys

Babu Krishnamurthy has over 11 years of industry experience. He currently works as a Technology Architect at Digital Practice in Infosys Technologies. His domain expertise includesbanking, ecommerce, and networking. His areas of expertise in technology include Enterprise Java technologies, portal technologies, eCommerce technologies, and web technologies.