

Mark Bercasio • Barryn Chun • Egan Nakano • Shaziney Natividad

# nómisma νόμισμα

## Requirements & Design

ICS 491 – Assignment 2: Standardized Design Requirements



**Team Omega**

July 17, 2016

# Table of Contents

1	Introduction
2	Requirements
2.1	Security Requirements
	Secure Coding in Java
	Authentication
	Logging
	Privileges
2.2	Privacy Requirements
2.3	Tracking Security Flaws During Development
	Source Code Analysis
	Runtime Analysis
2.4	Quality Gates / Bug Bars
	End User Scenarios
	Administration Scenarios
	Definition of Terms
	Server
	Client
	Definition of Terms
2.5	Security and Privacy Risk Assessments
	Security Risk Assessment
	System Purpose and Description
	Information Security Levels and Overall System Security Level
	Risk Determination
	Safeguard Determination
	Privacy Risk Assessment
3	Design
3.1	Design Requirements
	Secure Coding in Java
	Authentication
	Privacy
	Privileges
3.2	Attack Surface Analysis & Reduction
3.3	Threat Modeling
4	Conclusion
5	References

## **1 Introduction**

Planning for requirements and design in a development project is of utmost importance when it comes to secure coding practices. Before implementation can begin, it is key to map out the system components starting from the user all the way up to data manipulation. Along this network, we, as secure developers, are able to assess the data flows and possible threats that may leave the implementation vulnerable.

The project we are working on is a money tracking application (named Nomisma) that is used for keeping track of expenses incurred during trips. Users will have their own personal account from which they are able to store their transactions and analyze what they are spending their money on. The application also tracks how much money is being spent on certain outlets, with categories including food, travel, entertainment, and emergency expenses. Each transaction the user enters reduces the total balance and is added on to the specific category it was used for. While the app doesn't connect to a banking system, it still holds sensitive information (user accounts and user purchasing preferences) that needs to be secured.

In the following sections, we discuss our money tracking application with regards to security. We begin by looking at the requirements: what facets of our application needs to be secured the most, and most importantly, how do we plan on dealing with attacks that threaten the confidentiality, integrity, and availability of sensitive information.

## **2 Requirements**

### **2.1 Security Requirements**

Various security requirements should be met in order to prevent or reduce potential security vulnerabilities in a program.

#### **Secure Coding in Java**

Security flaws common to Java can be avoided by preventing potential tampering with the code. As such, secure coding practices detailed in the Design Requirements section of this document must be followed throughout the lifespan of Nomisma's creation.

#### **Authentication**

User access should always be verified through authentication. Users should be required to use passwords, which follow a reasonable password policy. Passwords should never be recoverable, and authentication must never be bypassable.

#### **Logging**

Confidential data, such as unencrypted passwords, should not be logged, but attempts to log into the software should be audited [1]. Data accepted from users must be constrained to what is appropriate and safe and validated as such.

## **Privileges**

The application should be functional without the use of elevated privileges, as having them can pose as security vulnerability [1]. Privileges or authorized access should be minimized in all manners possible.

## **2.2 Privacy Requirements**

Protecting the user's privacy is crucial, especially since it is considered a basic human right in many areas of the world [2]. To properly protect the user's privacy, it is important to understand and protect the entire scope of what is the user's data. This is not only limited to the information which may be manually provided by them, but also data that can be gathered indirectly, data regarding their usage of a program, such as logs or history, and data related to the user's system, for example, an IP address [2]. To best protect all categories of the user's data, data collected should be minimized, and those who have access to it should be strictly limited.

## **2.3 Tracking Security Flaws During Development**

During the development process, tracking security flaws is absolutely necessary for preventing problem. This is primarily done through analysis of not only the program's source code, but a dynamic program analysis as well.

### **Source Code Analysis**

Software development tools should be used to analyze the program's source code frequently throughout the entire writing process, from beginning to end. Such analyzers typically find common coding errors such as buffer overflow conditions, improper function calls that may lead to format string issues and the use of functions that may not be called safely [3].

### **Runtime Analysis**

In addition to analyzing the source code during the development process, the program should also be analyzed during runtime, including runtime testing. Analyzing the source code limits one to finding flaws that may not otherwise be known until certain interactions and behaviors in the program are observed [3]. By analyzing the program during its runtime, one can observe how it interacts with the database.

A program can be dynamically analyzed through two methods: by debugging and through the use of a code scanner [3]. Debugging requires stop-and-start testing, which can be very time consuming. Its effectiveness may also be limited to the knowledge of the programmer(s). Using a code scanner can save time because it can run with preconfigured checks for certain security weaknesses. Ideally, both methods should be used since there are cases when a program is better understood by the programmer; however, in the event that the programmer does not have a strong coding background, a code scanner should be used for a program's runtime analysis.

## 2.4 Quality Gates / Bug Bars

<b>End User Scenarios</b>	
<b>Usage notes:</b> These scenarios apply to consumers, clients, and administrators on end user accounts.	
Critical	<ul style="list-style-type: none"><li>● Lack of notice and consent<ul style="list-style-type: none"><li>○ Example:<ul style="list-style-type: none"><li>■ Lack of pronounced and explicit notice and opt-in consent before the transfer of sensitive personally identifiable information (PII) from the user's system.</li></ul></li></ul></li><li>● Lack of data protection<ul style="list-style-type: none"><li>○ Example:<ul style="list-style-type: none"><li>■ User PII is stored in persistent database(s) without a means of user authentication for access and modification of stored information.</li></ul></li></ul></li><li>● Lack of user controls<ul style="list-style-type: none"><li>○ Example:<ul style="list-style-type: none"><li>■ Collection and transmission of non-critical PII, while lacking a means (in application) for the user to prevent or stop collection and transfer of PII.</li></ul></li></ul></li><li>● Lack of internal data management and control<ul style="list-style-type: none"><li>○ Example:<ul style="list-style-type: none"><li>■ Stored PI access is not limited to only those who have a valid need to access it and/or there are no procedures in place to revoke access after it is no longer needed.</li></ul></li></ul></li><li>● Insufficient legal controls<ul style="list-style-type: none"><li>○ Example:<ul style="list-style-type: none"><li>■ Product transmits data to an agent or independent third party that has not signed a legally approved contract</li></ul></li></ul></li></ul>
Important	<ul style="list-style-type: none"><li>● Lack of notice and consent<ul style="list-style-type: none"><li>○ Example:<ul style="list-style-type: none"><li>■ Lack of pronounced and explicit notice and opt-in consent before the transfer of non sensitive PII from the user's system.</li></ul></li></ul></li><li>● Lack of user controls:<ul style="list-style-type: none"><li>○ Example:<ul style="list-style-type: none"><li>■ Continuous or ongoing collection and transmission of non sensitive user PII or anonymous data without an option in the UI for the user to stop and prevent collection and transmission.</li></ul></li></ul></li><li>● Lack of data protection:</li></ul>

	<ul style="list-style-type: none"> <li>○ Example: <ul style="list-style-type: none"> <li>■ Permanently stored non sensitive PII lacks a means to prevent access during storage or transfer of said data.</li> </ul> </li> <li>● Data minimization: <ul style="list-style-type: none"> <li>○ Example: <ul style="list-style-type: none"> <li>■ Sensitive PII is sent to a third party, when it is unnecessary in relation to the application's purpose.</li> </ul> </li> </ul> </li> </ul>
Moderate	<ul style="list-style-type: none"> <li>● Lack of user controls: <ul style="list-style-type: none"> <li>○ Example: <ul style="list-style-type: none"> <li>■ PII is collected and stored locally as hidden metadata, where the user has no ability to remove the metadata.</li> </ul> </li> </ul> </li> <li>● Lack of data protections: <ul style="list-style-type: none"> <li>○ Example: <ul style="list-style-type: none"> <li>■ Temporarily stored non sensitive PII lacks a means to prevent access during storage or transfer of said data.</li> </ul> </li> </ul> </li> <li>● Data minimization: <ul style="list-style-type: none"> <li>○ Example: <ul style="list-style-type: none"> <li>■ Anonymous data or non sensitive PII is sent to a third party, when it is unnecessary in relation to the application's purpose.</li> </ul> </li> </ul> </li> <li>● Lack of internal data management and controls: <ul style="list-style-type: none"> <li>○ Example: <ul style="list-style-type: none"> <li>■ Data stored lacks a retention policy</li> </ul> </li> </ul> </li> </ul>
Low	<ul style="list-style-type: none"> <li>● Lack of consent/notice <ul style="list-style-type: none"> <li>○ Example: <ul style="list-style-type: none"> <li>■ PII is collected and stored locally as hidden metadata without discernable notice, but is not accessible by others and will not be transferred in the event files or folders are shared.</li> </ul> </li> </ul> </li> </ul>

[4]

<b>Administration Scenarios</b>  <b>Usage notes:</b> These scenarios/situations apply to administrators acting in their administrative roles.	
Critical	<ul style="list-style-type: none"> <li>● Lack of controls: <ul style="list-style-type: none"> <li>○ Example: <ul style="list-style-type: none"> <li>■ Automated transmission of sensitive user PII without sufficient notice and explicit consent in the UI from the administrator before transfer.</li> </ul> </li> </ul> </li> <li>● Lack of/insufficient privacy disclosure:</li> </ul>

	<ul style="list-style-type: none"> <li>○ Example: <ul style="list-style-type: none"> <li>■ Lack of a development guide for administrators, providing legal advice.</li> </ul> </li> </ul>
Important	<ul style="list-style-type: none"> <li>● Lack of controls: <ul style="list-style-type: none"> <li>○ Example: <ul style="list-style-type: none"> <li>■ Automated transmission of non sensitive user PII or anonymous data without sufficient notice and explicit consent in the UI from the administrator before transfer.</li> </ul> </li> </ul> </li> <li>● Lack of/insufficient privacy disclosure: <ul style="list-style-type: none"> <li>○ Example: <ul style="list-style-type: none"> <li>■ Lack of disclosure to administrators about the storage or transfer of PII in a development guide.</li> </ul> </li> </ul> </li> </ul>
Moderate	<ul style="list-style-type: none"> <li>● Lack of controls <ul style="list-style-type: none"> <li>○ Example: <ul style="list-style-type: none"> <li>■ Lack of shown or provided mechanism(s) to help administrators prevent unintentional disclosure of user data</li> </ul> </li> </ul> </li> </ul>

[4]

## Definition of Terms

**Anonymous data:** Non-personal data, without any connection to an individual, having no inherent link to a user.

**Explicit consent:** An individual is presented, in clear terms, with an expressed option to agree or disagree with an action taken, like the collection, use, or disclosure of personal information.

**Implicit consent:** An explicit action from the user to express consent is not required, as consent is implied in the operation that the user begins.

**Metadata:** Data that provides information about other data.

**Persistent storage:** The data stored continues to be available even after the user exits the application.

**Personally identifiable information:** Any data or information that directly identifies or can be used to identify, contact, or locate a specific individual [5]. Non-sensitive Personally Identifiable Information can be transmitted unencrypted without resulting in harm to the individual, and can easily be gleaned from public records, phone books, corporate directories, and websites [5].

**Sensitive personally identifiable information:** Any data or information that can be used to discriminate, facilitate identity theft, or permit unauthorised access to a specific individual's

account. Sensitive Personally Identifiable Information is information that could result in harm to the individual whose privacy has been breached, when disclosed. This includes things like biometric information, heritage, unique identifiers(i.e. Social Security Numbers, passports, etc.), passwords or pins, and financial information.

- Critical:** Release could create legal or regulatory liability issues for the organization.
- Important:** Release could create a high risk of negative reaction by privacy advocates and/or damage the organization's image.
- Moderate:** Some users and privacy advocates may raise questions and concerns, but repercussions will be limited.
- Low:** May cause user queries, unlikely to raise scrutiny by privacy advocates.

**Temporary storage:** The data stored is only available when the application is running.

Server	
Critical	<ul style="list-style-type: none"> <li>● Elevation of privilege : <ul style="list-style-type: none"> <li>○ Remote anonymous user <ul style="list-style-type: none"> <li>■ The ability to either execute arbitrary code or obtain more privilege than intended</li> </ul> </li> <li>○ Stack overflows in remotely callable code, all write access violations, exploitable read access violations, etc.</li> </ul> </li> </ul>
Important	<ul style="list-style-type: none"> <li>● Denial of service: <ul style="list-style-type: none"> <li>○ Anonymous <ul style="list-style-type: none"> <li>■ Persistent DoS <ul style="list-style-type: none"> <li>● Sending a malicious packet that results in a Blue Screen of Death.</li> </ul> </li> <li>■ Temporary DoS <ul style="list-style-type: none"> <li>● A single remote client consumes all available memory/resources on a server by establishing sessions and keeping them open.</li> </ul> </li> </ul> </li> <li>○ Authenticated <ul style="list-style-type: none"> <li>■ Persistent DoS on a high value target. <ul style="list-style-type: none"> <li>● Sending a small amount of packets causes a service failure for a high value asset, such as a domain controller or certificate server.</li> </ul> </li> </ul> </li> </ul> </li> <li>● Elevation of privilege:</li> </ul>



	<ul style="list-style-type: none"> <li>○ Remote authenticated user <ul style="list-style-type: none"> <li>■ The execution of arbitrary code with extensive user action</li> </ul> </li> <li>○ Local authenticated user <ul style="list-style-type: none"> <li>■ Examples: <ul style="list-style-type: none"> <li>● Execution of arbitrary code</li> <li>● Unauthorized access to file systems; i.e. arbitrary writing to the file system</li> </ul> </li> </ul> </li> <li>○ Stack overflows in remotely callable code, all write access violations, exploitable read access violations, etc.</li> <li>● Information disclosure(targeted): Cases where an attacker can locate and read specific information from anywhere in the system. <ul style="list-style-type: none"> <li>○ Example: <ul style="list-style-type: none"> <li>■ Attacker can covertly collect PII without user consent.</li> </ul> </li> </ul> </li> <li>● Spoofing: An entity (computer, process, server, user, etc.) is able to impersonate or pose as a specific entity. <ul style="list-style-type: none"> <li>○ Examples: <ul style="list-style-type: none"> <li>■ Computer connecting to server is able to pass itself off as a valid user using a protocol that is supposed to provide strong authentication.</li> </ul> </li> </ul> </li> <li>● Tampering: <ul style="list-style-type: none"> <li>○ Modification of any high value asset data in a common or default scenario where the modification is retained after restarting the OS/application.</li> <li>○ Permanent or persistent modification of any user/system data in a common or default scenario. <ul style="list-style-type: none"> <li>■ Examples: <ul style="list-style-type: none"> <li>● Modification of application databases or files in a common or default scenario, such as with an SQL injection.</li> <li>● Modification of OS or application settings without user consent in a common or default scenario.</li> </ul> </li> </ul> </li> </ul> </li> <li>● Security features: Breaking or bypassing any provided security features. <ul style="list-style-type: none"> <li>○ Examples: <ul style="list-style-type: none"> <li>■ Disabling or bypassing a firewall without informing the user or obtaining user consent.</li> <li>■ Reconfiguring a firewall and allowing connection to other processes.</li> </ul> </li> </ul> </li> </ul>
--	---

Moderate	<ul style="list-style-type: none"> <li>● Denial of service: <ul style="list-style-type: none"> <li>○ Anonymous <ul style="list-style-type: none"> <li>■ Temporary DoS <ul style="list-style-type: none"> <li>● Multiple remote clients consume all available memory/resources on a server by establishing sessions and keeping them open</li> </ul> </li> </ul> </li> <li>○ Authenticated <ul style="list-style-type: none"> <li>■ Persistent DoS <ul style="list-style-type: none"> <li>● A logged in user causes stack overflows in remotely callable code, all write access violations, exploitable read access violations, etc.</li> </ul> </li> <li>■ Temporary DoS with amplification <ul style="list-style-type: none"> <li>● Server executes a stored procedure installed by some product and consumes all of the CPU for a matter of minutes.</li> </ul> </li> </ul> </li> </ul> </li> <li>● Spoofing: An entity (computer, process, server, user, etc.) is able to impersonate or pose as a random entity that cannot be specifically selected <ul style="list-style-type: none"> <li>○ Examples: <ul style="list-style-type: none"> <li>■ Client user/computer is able to pass itself off as a different user/computer using a protocol that is supposed to provide strong authentication.</li> <li>■ Client properly authenticates with server, but receives the session of another user that is connected at the same time.</li> </ul> </li> </ul> </li> <li>● Tampering: <ul style="list-style-type: none"> <li>○ Permanent or persistent modification of any user/system data in a specific scenario. <ul style="list-style-type: none"> <li>■ Examples: <ul style="list-style-type: none"> <li>● Modification of application databases or files in a specific scenario.</li> <li>● Modification of OS or application settings without user consent in a specific scenario.</li> </ul> </li> </ul> </li> <li>○ Temporary modification of data in a common or default scenario that is not retained after restarting the OS/application/session.</li> </ul> </li> <li>● Information disclosure(targeted): Cases where an attacker can locate and read specific information, including system information, from specific locations.</li> </ul>
----------	---

	<ul style="list-style-type: none"> <li>○ Example: <ul style="list-style-type: none"> <li>■ Targeted disclosure of the existence of a file.</li> <li>■ .Targeted disclosure of a file version.</li> </ul> </li> <li>● Security assurances: <ul style="list-style-type: none"> <li>○ A feature or another product that customers expect to offer security protection. <ul style="list-style-type: none"> <li>■ Processes that are running under normal user level privileges unless admin credentials have been provided.</li> </ul> </li> </ul> </li> </ul>
Low	<ul style="list-style-type: none"> <li>● Information disclosure(untargeted): <ul style="list-style-type: none"> <li>○ Runtime information <ul style="list-style-type: none"> <li>■ Example: <ul style="list-style-type: none"> <li>● Leak of random heap memory.</li> </ul> </li> </ul> </li> </ul> </li> <li>● Tampering: <ul style="list-style-type: none"> <li>○ Temporary modification of data in a specific scenario that is not retained after restarting the OS/application.</li> </ul> </li> </ul>

[6]

Client	
Critical	<ul style="list-style-type: none"> <li>● Elevation of privilege(remote): The ability to execute arbitrary code or to gain more privilege than intended. <ul style="list-style-type: none"> <li>○ Examples: <ul style="list-style-type: none"> <li>■ Unauthorized file system access writing to the file system.</li> <li>■ Execution of arbitrary code without extensive user actions.</li> <li>■ Stack overflows in remotely callable code, all write access violations, exploitable read access violations, etc.</li> </ul> </li> </ul> </li> <li>● Network worm or unavoidable cases where the client is compromised without warning or prompts.</li> </ul>
Important	<ul style="list-style-type: none"> <li>● Information disclosure(targeted):</li> <li>● Elevation of privilege (remote): The execution of arbitrary code with extensive user action. <ul style="list-style-type: none"> <li>○ All write access violations, exploitable read access violations, etc. in remote callable code with extensive user actions.</li> </ul> </li> </ul>

	<ul style="list-style-type: none"> <li>• Elevation of privilege (local): Low-privilege user(s) can elevate themselves to another user, administrator, or local system. <ul style="list-style-type: none"> <li>◦ All write access violations, exploitable read access violations, etc. in local callable code.</li> </ul> </li> <li>• Denial of service: <ul style="list-style-type: none"> <li>◦ System corruption denial of service, requiring reinstallation of components and/or systems.</li> <li>◦ Drive by denial of service <ul style="list-style-type: none"> <li>■ Non authenticated system DoS</li> <li>■ No default security features or boundary mitigations</li> <li>■ No user interaction</li> <li>■ No auditable trail</li> </ul> </li> </ul> </li> <li>• Spoofing: The ability for an attacker to present a visually identical UI, that however is different, that users must rely on to make valid trust decisions in a default or common scenario. <ul style="list-style-type: none"> <li>◦ Examples: <ul style="list-style-type: none"> <li>■ Displaying a falsified login prompt to gather user/account credentials</li> <li>■ Displaying a different file name in a “Do you want to run this program?” dialog box than the one that the program will actually be loaded.</li> </ul> </li> </ul> </li> <li>• Tampering: Permanent modification of any user data or data used in trust decisions in a common/default scenario that will be retained after restarting the OS/application. <ul style="list-style-type: none"> <li>◦ Examples: <ul style="list-style-type: none"> <li>■ Modification of significant OS/application settings without user consent.</li> <li>■ Modification of user data.</li> </ul> </li> </ul> </li> <li>• Security features: Breaking or bypassing any security feature. <ul style="list-style-type: none"> <li>◦ Examples: <ul style="list-style-type: none"> <li>■ Disabling or bypassing a firewall without informing the user or obtaining user consent.</li> <li>■ Reconfiguring a firewall and allowing connection to other processes.</li> <li>■ Using weak encryption or storing keys in plain text.</li> </ul> </li> </ul> </li> </ul>
Moderate	<ul style="list-style-type: none"> <li>• Denial of service: Permanent denial of service, requiring a cold reboot or causing a bug check/ critical failure. <ul style="list-style-type: none"> <li>◦ Example: <ul style="list-style-type: none"> <li>■ Opening the application causes a blue screen of death, etc.</li> </ul> </li> </ul> </li> </ul>

	<ul style="list-style-type: none"> <li>● Spoofing: The ability for an attacker to present a visually identical UI, that however is different, that users are used to trusting in a specific scenario. <ul style="list-style-type: none"> <li>○ Examples: <ul style="list-style-type: none"> <li>■ Modification of significant OS/application settings without user consent.</li> <li>■ Unauthorised modification of user data.</li> </ul> </li> </ul> </li> <li>● Information disclosure(targeted): Cases wherein the attacker can gain access to read information on the system, including system information that was not intended to be revealed. <ul style="list-style-type: none"> <li>○ Examples: <ul style="list-style-type: none"> <li>■ Targeted existence of file</li> <li>■ Targeted file version number</li> </ul> </li> </ul> </li> </ul>
Low	<ul style="list-style-type: none"> <li>● Denial of service: Temporary denial of service, requiring the application to be restarted. <ul style="list-style-type: none"> <li>○ Example: <ul style="list-style-type: none"> <li>■ Opening an unsupported file type causes the application to crash.</li> </ul> </li> </ul> </li> <li>● Spoofing: The ability for an attacker to present a visually identical UI, that however is different, that is a small or single component of a larger attack scenario. <ul style="list-style-type: none"> <li>○ Examples: <ul style="list-style-type: none"> <li>■ User is led to a malicious web site, and clicks on a button in a spoofed dialog box, thusly being vulnerable to attack through a different browser bug.</li> </ul> </li> </ul> </li> <li>● Tampering: Temporary modification of any data that is not retained after restarting the OS/application. <ul style="list-style-type: none"> <li>○ Examples: <ul style="list-style-type: none"> <li>■ Leak of random heap of memory.</li> </ul> </li> </ul> </li> </ul>

[6]

## Definition of Terms

**Authenticated:** Any attack wherein the network must provide authentication, implied in this is that logging must be able to occur.

**Anonymous:** Any attack wherein the network does not need to provide authentication to be completed.

**Client:** Either software that is run locally on a single computer or software that accesses shared resources provided by a server over a network.

**Default or common:** Any features that are available and active “out of the box” or that apply to more than 10 percent of users.

**Server:** Computer that is configured to run software that awaits and fulfills requests from client processes that run on other computers.

**Critical:** A security vulnerability that would be rated as having the greatest potential for damage.

**Important:** A security vulnerability that would be rated as having a significant potential for damage, but less than Critical.

**Moderate:** A security vulnerability that would be rated as having the moderate potential for damage, but less than Important.

**Low:** A security vulnerability that would be rated as having the lowest potential for damage.

**Scenario:** Any features that need special customization or cases to enable, or that reach less than 10 percent of users

**Targeted information disclosure:** Ability to explicitly and purposefully select(target) specific information.

**Trust decision:** A trust decision is any time a user takes an action under the belief that some information is being presented by a specific entity, as in the system or a specific local/remote source.

**Temporary DoS:** A type of denial of service where the target cannot perform normal operations due to an attack, the response to an attack is in the same magnitude as the attack, or the target is able to gain normal functionality shortly after the end of the attack.

**Temporary DoS with amplification:** A type of denial of service where the target cannot perform normal operations due to an attack, the response to an attack is of a larger magnitude to the attack, or the target is able to gain normal functionality after the end of the attack, but it takes some time.

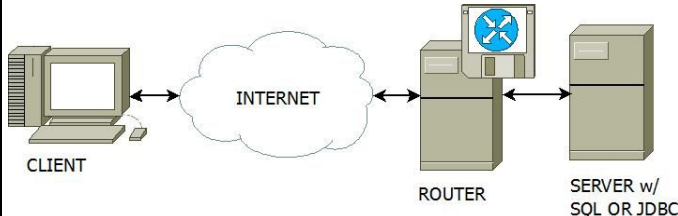
**Permanent DoS:** A denial of service attack that requires an administrator to start, restart, or replace all parts of a system. Also, any vulnerability that automatically restarts the system.

## 2.5 Security and Privacy Risk Assessments

### Security Risk Assessment

In general, the authentication and client-server connection portions of Nomisma will need threat modeling and security design reviews. The authentication system of the application, as well as server-side facing SQL service will also require penetration testing to ensure maximum reliability and security. As Nomisma relies on connections to a remote database, Remote Procedure Call fuzz testing [10] may be appropriate. All of this must occur before the application is released for public use.

System Purpose and Description

Function and purpose of the system	Store and keep track of financial information
General functional requirements	Store financial transaction information and possibly statistics of linked accounts
Business processes, applications, and services supported	No additional services
System components	Java, JDBC (SQL)
Environmental factors	Component compatibility with local and server system
Network diagram with system boundaries (attach)	 <pre> graph LR     CLIENT[CLIENT] &lt;--&gt; INTERNET((INTERNET))     INTERNET &lt;--&gt; ROUTER[ROUTER]     ROUTER &lt;--&gt; SERVER[SERVER w/ SQL OR JDBC]             </pre>
General information flow	Initial authentication data sent to server, further transactions containing data and management queries
Technical and business users (list)	varies
System ownership (shared or dedicated)	Client: shared Server: dedicated

Information Security Levels and Overall System Security Level

Information Category	User Authentication (username, passwords)
Information Security Level	High
Information Category	User Personal Information
Information Security Level	High
Overall System Security Level	High

Risk Determination

Item No.	Threat Name	Risk Description	Likelihood of Occurrence	Impact Severity	Risk Level
1	Password	A user's password is	Moderate	Confidentiality,	Low

	Compromise	compromised		Single User	
2	Brute Force Password Attack	Random passwords for a user are repeatedly submitted	<b>High</b>	Any Number of Users	Low
3	DDoS	Abnormally high traffic is directed at the webserver	Moderate	Availability - All remote users	Moderate
4	SQL Injection	Malicious code is sent to the server as a query	Moderate	<b>Integrity - Server, Confidentiality</b>	<b>High</b>

### Safeguard Determination

Item No.	Recommended Safeguard Description	Residual Likelihood of Occurrence	Residual Impact Severity	Residual Risk Level
1	Encourage users to protect their password	Varies	Moderate	Moderate
1	Allow users to change their password	Varies	Moderate	Moderate
1	Authentication from secondary device	Low	Moderate	Low
2	Account Locking after consecutive incorrect attempts	Low	Low	Low
3	Server is made to close connections inactive for a while and client reconnects them when reactivated	Moderate	Low	Low
4	Input Sanitization	Low	Low	Low

(Table based on "Information Security Risk Guidelines" from the Executive Office for Administration and Finance [11])

## Privacy Risk Assessment

As the data to be handled by Nomisma will be financial transactions and account data, its Privacy Impact Rating falls under categories P1-High Privacy Risk and P2-Moderate Privacy Risk. By itself, the client side program should not interact with existing user files or filetypes, but does send and receive data to and from the server. Unauthorized access to P2 data will be prevented with a combination of client-server authentication, two-step verification, and input sanitization. None of this data is stored locally. A disclosure describing the full terms and conditions should be made available upon first startup and anytime during the program's runtime from the Help menu or elsewhere.



## **3 Design**

### **3.1 Design Requirements**

To maintain security in compliance with the aforementioned security and privacy requirements, the following design requirements must be adhered to throughout the entire development process of Nomisma.

#### **Secure Coding in Java**

To prevent potential security vulnerabilities, especially tampering with code, the software must be created with secure coding practices. Such prevention can be made possible with the following practices. The software should limit access to classes, methods and variables by making everything private [14]. Software should be written in a way that prevents an attacker from making new instances of defined classes. Wherever possible, all classes and methods should be finalized [14]. All classes should also be made to be non-cloneable [14].

#### **Authentication**

For enhanced security, Nomisma should be designed with a two-step authentication method. Users will be required to provide his or her password along with a unique key, which is provided via his or her associated email account. This mechanism must not be bypassable. Authorization must only take place after authentication. In the event that a user is unable to provide the appropriate keys after five failed attempts, a procedure to change forgotten passwords should be implemented.

#### **Privacy**

To better protect the user's privacy, or protect his or her data, it's best to not collect data in the first place. That is to say, the program should only have to collect and protect data that is absolutely necessary. In the case of Nomisma, it should only collect and validate a user's authentication key(s) and any information they wish to manage using the application. Limiting the data that is kept by the program can also be done by reducing the sensitivity of the data that retained by the program. For example, a subset of a user's phone number can be stored in place of the entire number. User data can also be converted to a less sensitive form using one-way hash functions, which convert text into a string of digits that is nearly impossible to convert back into the original text. Access to all forms of a user's data should be strictly limited to persons that have legitimate business purposes for accessing the data. Thus, stringent permissions and access levels should be implemented.

#### **Privileges**

The application should be functional without the use of elevated privileges. Privileges or authorized access should be minimized in all manners possible. Any libraries that a user can write to must be segregated from those that involve running processes.

### **3.2 Attack Surface Analysis & Reduction**

Money tracking software is a high value target for hackers because sensitive information regarding bank accounts, credit cards, and bills can easily be stolen with a successful attack.

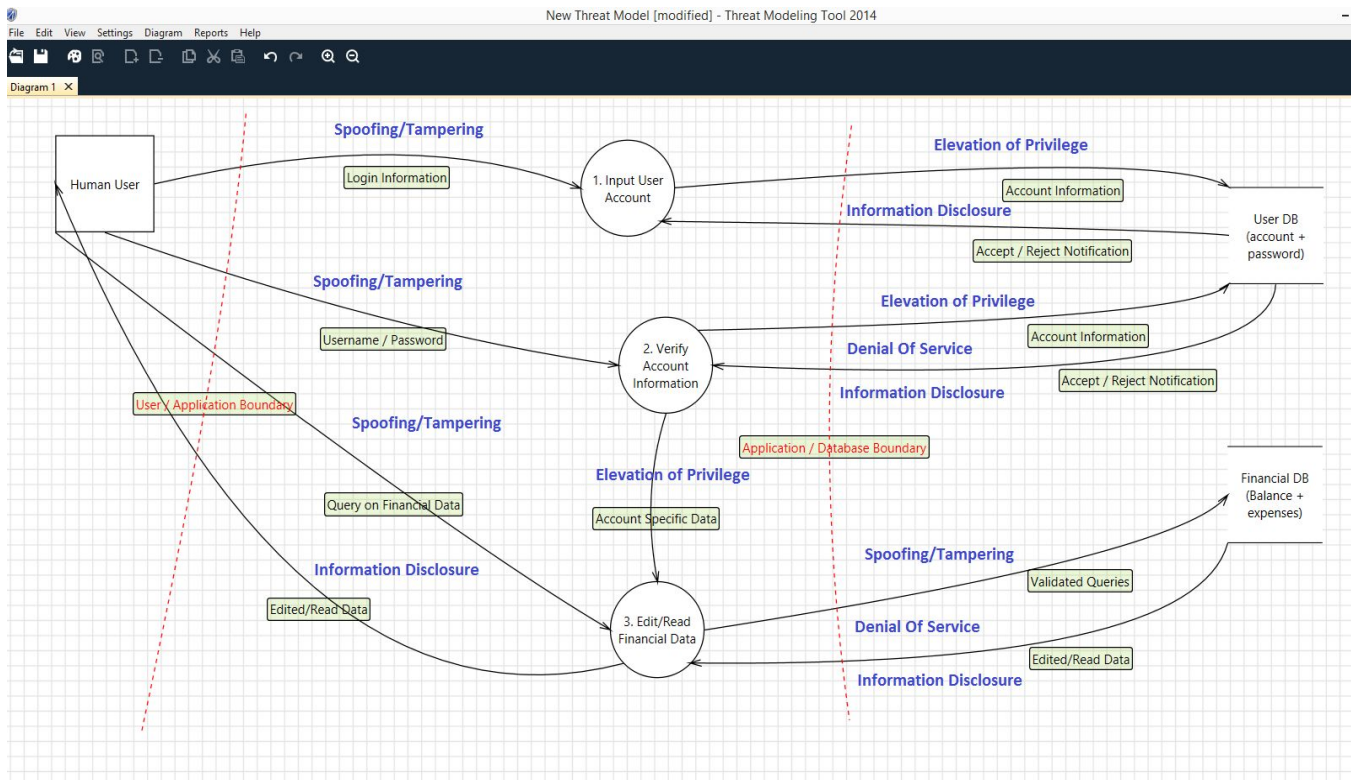
Aside from that, identity theft and expense fraud are all likely possibilities of a breached network. As for privilege levels, users are only able to input text in order to enter username/password. For database alteration, users need to have first logged in. Users can only read or alter their specific row in the database table.

A similar program to our money management applet is called Mint. Mint is a finance management app that aggregates every income and expense of the user in order to show the user's current financial situation [15]. Given how mint is an aggregate of so many financial transactions and balances, it is difficult to spot which factors are vulnerable. US financial institutions (banks, etc) send data for the app to use. Without some form of a verification system, unauthorized institutions could send faulty information to the app [16].

Since a login is required for many personal accounts, issues could arise from input exploitation. If passwords are stored in a SQL database, users are able to use SQL injection to bypass verification [17]. In addition, brute force or tampering with account access can be disastrous if they aren't accounted for. Mint has a mobile version and there is always the threat of theft of the smartphone or tablet [16]. In this case, there needs to be a method of deleting information on the user's account (similar to canceling a stolen credit card).

The largest, most costly threat lies in database access. With all the information regarding users' accounts and transactions being stored on a database, it is the prime target for hackers. A denial of service attack targeting the database could halt all operations. In addition, an attacker may be able to abuse a bug or error in code in order to elevate their privilege, allowing access to the database. Updating the database with patches and regular checks of CVEs (common vulnerabilities and exposures) is considered good practice [18]. Many databases have their own vulnerabilities and checking the vendor for new updates is key in preventing information theft.

### 3.3 Threat Modeling



**Figure 1:** Data Flow Diagram of proposed program. Squares represent external entities, circles represent processes, arrows represent data flows, and open-ended rectangles represent data stores. Red text indicates privilege boundaries. Blue text indicates possible threats in regards to data flow.

## 4 Conclusion

In essence, cybersecurity is best planned out before any implementation can begin. In order for Nomisma to succeed, the team needs to identify the key aspects of the program that are going to be left vulnerable. Requirements planning (in terms of security) analyzes the portions of the program that is most susceptible to vulnerability. We have established that we will be using a structured and secure method of coding in Java. In addition, the program will require some form of authentication as to control privileges. In order to check for these security requirements, continual analyses/audits will be conducted during the development and release period.

Creating risk assessments of the intricacies of the project allow us to look at the security issues which we need to handle before the product can ship. By answering questions regarding various aspects of the project (general background, authentication methods, authorization, cryptography, etc), we are able to create a base from which features can be included or edited

to fit the assessment. Also, conducting various scenarios of program exploits allow the team to establish which sections of the program need refining.

Looking into the user or client scenarios, we have learned that spoofing and tampering are the most likely causes of security failure for Nomisma. As a result, user privileges in terms of client to application scenarios need to account for such things such as brute force attacks and SQL injection. On the side of application/database issues, the main threat comes from information disclosure from the database itself. As such, secure coding practices need to be targeted towards the various database read/write commands. Most importantly, the transferal of data from the database to the application and from the application to the client needs to be carefully observed for the possibility of leaked information.

## 5 References

- [1] "Security Development Checklists", Mac Developer Library, 2016. [Online]. Available: [https://developer.apple.com/library/mac/documentation/Security/Conceptual/SecureCodingGuide/SecurityDevelopmentChecklists/SecurityDevelopmentChecklists.html#/apple\\_ref/doc/uid/TP40002415-CH1-SW1](https://developer.apple.com/library/mac/documentation/Security/Conceptual/SecureCodingGuide/SecurityDevelopmentChecklists/SecurityDevelopmentChecklists.html#/apple_ref/doc/uid/TP40002415-CH1-SW1). [Accessed: 12- Jul- 2016].
- [2] Privacy Guidelines for Developing Software Products and Services, 3rd ed. Microsoft, 2008, pp. 5-10.
- [3] Integrating Security into Development, No Pain Required, 1st ed. SANS Institute, 2016, pp. 6-7.
- [4] "Appendix M: SDL Privacy Bug Bar (Sample)", Microsoft, 2016. [Online]. Available: <https://msdn.microsoft.com/en-us/library/cc307403.aspx>. [Accessed: 12- Jul- 2016].
- [5] "personally identifiable information (PII)", SearchFinancialSecurity, 2016. [Online]. Available: <http://searchfinancialsecurity.techtarget.com/definition/personally-identifiable-information>. [Accessed: 12- Jul- 2016].
- [6] "Appendix N: SDL Security Bug Bar (Sample)", Microsoft, 2016. [Online]. Available: <https://msdn.microsoft.com/en-us/library/cc307404.aspx>. [Accessed: 12- Jul- 2016].
- [7] "Security Briefs - Add a Security Bug Bar to Microsoft Team Foundation Server 2010", Microsoft, 2016. [Online]. Available: <https://msdn.microsoft.com/en-us/magazine/ee336031.aspx>. [Accessed: 12- Jul- 2016].
- [8] "What Is The Process of Creating A Bug Bar?", Stack Exchange, 2016. [Online]. Available: <http://programmers.stackexchange.com/questions/232425/what-is-the-process-of-creating-a-bug-bar>. [Accessed: 12- Jul- 2016].
- [9] "Different Types of Consent", PrivacySense.net, 2015. [Online]. Available: <http://www.privacysense.net/different-types-consent/>. [Accessed: 12- Jul- 2016].
- [10] "Phase 4: Verification", *Microsoft Developer Network*, 2016. [Online]. Available: <https://msdn.microsoft.com/en-us/library/cc307418.aspx>. [Accessed: 17- Jul- 2016].
- [11] "Information Security Risk Assessment Guidelines", *Administration and Finance*, 2016. [Online]. Available: <http://www.mass.gov/anf/research-and-tech/cyber-security/security-for-state-employees/risk-assessment/risk-assessment-guideline.html>. [Accessed: 17- Jul- 2016].

- [12] "A step-by-step SMB IT security risk assessment process", *TechTarget*, 2016. [Online]. Available: <http://searchmidmarketsecurity.techtarget.com/tip/A-step-by-step-SMB-IT-security-risk-assessment-process>. [Accessed: 17- Jul- 2016].
- [13] "Appendix C: SDL Privacy Questionnaire", *Microsoft Developer Network*, 2016. [Online]. Available: <https://msdn.microsoft.com/en-us/library/cc307393.aspx>. [Accessed: 17- Jul- 2016].
- [14] G. McGraw and E. Felten, "Twelve rules for developing more secure Java code", *JavaWorld*, 1998. [Online]. Available: <http://www.javaworld.com/article/2076837/mobile-java/twelve-rules-for-developing-more-secure-java-code.html>. [Accessed: 12- Jul- 2016].
- [15] "Mint: Money, Bill Pay, Credit Score & Investing", Mint, 2016. [Online]. Available: <https://www.mint.com>. [Accessed: 12- Jul- 2016].
- [16] "How it Works: Security", Mint, 2016. [Online]. Available: <https://www.mint.com/how-mint-works/security#toc>. [Accessed: 12- Jul- 2016].
- [17] "mysql Search Results", Common Vulnerabilities and Exposures, 2016. [Online]. Available: <https://cve.mitre.org/cgi-bin/cvekey.cgi?keyword=mysql>. [Accessed: 12- Jul- 2016].
- [18] D. Mohindra and W. Snavey, "SEC55-J. Ensure that security-sensitive methods are called with validated arguments", SEI CERT Oracle Coding Standard for Java, 2015. [Online]. Available: [https://www.owasp.org/index.php/SQL\\_Injection](https://www.owasp.org/index.php/SQL_Injection). [Accessed: 12- Jul- 2016].

# ICS 491 Assignment 3: SDL: Implementation

(Pages 22-25)

Mark Bercasio, Barryn Chun, Egan Nakano, Shaziney Natividad

July 24, 2016

## Introduction

In order to uphold an acceptable level of code security and quality in Nomisma, it was important that we agreed upon a standard set of coding tools and practices. Below is a list of applications we have deemed adequate for code management, analysis, and testing-- as well as a list of deprecated methods and libraries for our team to avoid. Tracking which versions work for our project allows us to analyze why certain aspects of the project work the way they do or don't work at all.

## List of Approved Tools

### Unmanaged/Managed Code:

Compiler/Tool	Minimum Required Version and Switches/Options	Optimum / Recommended Version and	Comments
---------------	---	-----------------------------------	----------

		Switches/Options	
Java Compiler	JDK 1.7	JDK 1.8	Include in the project library (should be default)
(IDE) Netbeans	8.0.1	8.1	Primary development environment for code management and testing
Notepad++	6.9	6.9.2	Optional text editor for light code editing and inspection

#### **Testing Tools:**

Compiler/Tool	Minimum Required Version and Switches/Options	Optimum / Recommended Version and Switches/Options	Comments
Static Code Analysis	Netbeans IDE 8.0.1 JDK Version 7 and higher	Best to use <a href="#">jsr305-2.0.0.jar</a> in order to optimize static analysis tools	Include <a href="#">jsr305-2.0.0.jar</a> in the project library

#### **Database Tools:**

Compiler/Tool	Minimum Required Version and Switches/Options	Optimum / Recommended Version and Switches/Options	Comments
MySQL Database	MySQL Community 5.7 <a href="#">mysql-installer-web-community-5.7.13.0.msi</a>	N/A Use minimum requirements	Use installation settings provided to the user
MySQL Connector (Java)	MySQL Connector Java 5.1.39 jar file <a href="#">mysql-connector-java-5.1.39.zip</a>	Best practice to use the latest MySQL Connector (5.1.39)	Include the jar file in the project library

#### **Deprecated Functions to be Avoided in Nomisma:**

Code/Library	Reason for Deprecation/Alternative Option
--------------	---



<b>java.io.DataInputStream.readLine()</b> <a href="https://docs.oracle.com/javase/8/docs/api/java/io/DataInputStream.html#readLine--">https://docs.oracle.com/javase/8/docs/api/java/io/DataInputStream.html#readLine--</a>	Convert to use the <b>BufferedReader</b> class. This method can be used to read user inputs of authentication.
<b>java.lang.String.getBytes(int, int, byte[], int)</b> <a href="https://docs.oracle.com/javase/7/docs/api/java/lang/String.html#getBytes(int,%20int,%20byte[],%20int)">https://docs.oracle.com/javase/7/docs/api/java/lang/String.html#getBytes(int,%20int,%20byte[],%20int)</a>	Can be replaced with <b>java.lang.String.getBytes()</b> . This method could have uses in encoding data transfers.
<b>java.sql.ResultSet.getBigDecimal(int, int)</b> <a href="https://docs.oracle.com/javase/7/docs/api/java/sql/ResultSet.html#getBigDecimal(int,%20int)">https://docs.oracle.com/javase/7/docs/api/java/sql/ResultSet.html#getBigDecimal(int,%20int)</a>	Can be replaced with <b>java.sql.ResultSet.getBigDecimal(String)</b> or <b>java.sql.ResultSet.getBigDecimal(int)</b> . These methods can be used to gather decimal (money) values from table columns.
<b>java.sql.Date</b> <a href="https://docs.oracle.com/javase/7/docs/api/java/sql/Date.html">https://docs.oracle.com/javase/7/docs/api/java/sql/Date.html</a>	When developing under Java 8 and above, it is preferred to use the java.time package <a href="https://docs.oracle.com/javase/8/docs/api/java/time/package-summary.html">https://docs.oracle.com/javase/8/docs/api/java/time/package-summary.html</a> . These methods can be used to log user transactions to confirm integrity.
<b>java.sql.DriverManager.getLogStream()</b> <a href="https://docs.oracle.com/javase/8/docs/api/java/sql/DriverManager.html#getLogStream--">https://docs.oracle.com/javase/8/docs/api/java/sql/DriverManager.html#getLogStream--</a>	Can be replaced with <b>java.sql.DriverManager.getLogWriter()</b> . The DriverManager manages the establishment of connections. In our case, it manages which JDBC drivers to make connections with.

## Static Code Analysis

<https://netbeans.org/kb/docs/java/code-inspect.html>

For static code analysis, we have decided to work with Netbeans' built-in static code analysis feature. After testing it out a few times, it is clear that this tool is extremely beneficial in capturing missed coding errors. From the packages and java classes, the inspector feature creates a tree of any warnings or bugs that it detects. For example, it tells the user of unused imports (packages) and methods/classes with missing javadocs (which we will include as we progress on with the project). In case of our project, it seems that we've been placing a lot of try and catch statements without actually confirming that an exception is thrown.

The static code analysis for Netbeans also provides a Java hints configuration. In essence, if there are any errors or unsafe coding patterns, the feature provides possible alternatives. In our case, it helped clean up our code by identifying our exception catching could be more specific given how we used the broad (Exception e) catch statement. Surprisingly, almost all the hints provided were helpful in improving the function and clarity of our code.

One last vital feature of the static code analysis tool is the findbugs option. In our case, the bugs that were found did not play a drastic role in the functioning of our project (with our current knowledge of the program); however, it was still important that we fix the code to eliminate said bugs. As mentioned

previously, one of the more common bugs in our project was catching an exception when none was thrown. This could be fixed by narrowing down what type of exceptions are caught. Also, we found that an if else tree in our project could possibly ignore the exception (still figuring out the reason).

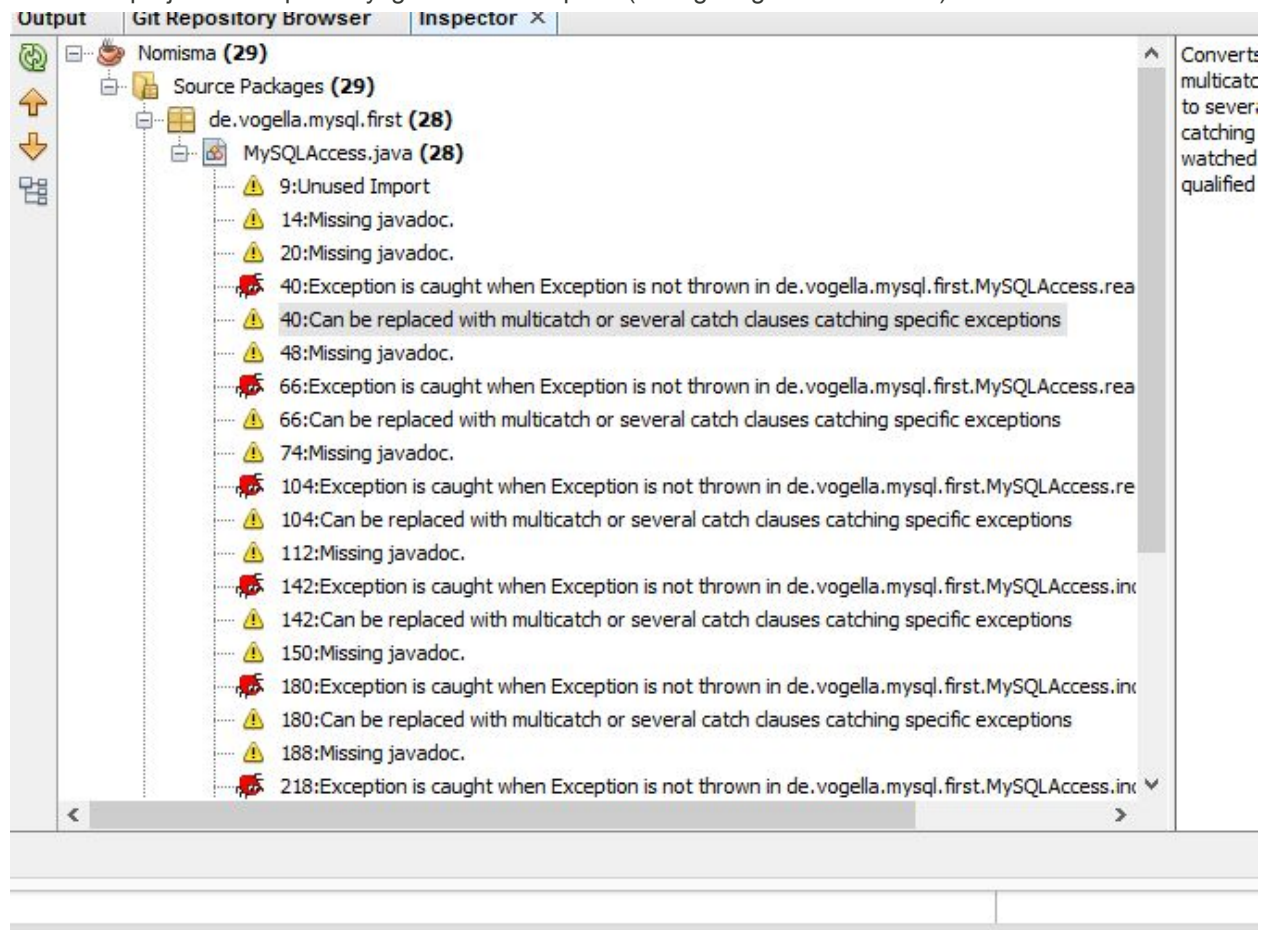


Figure 1: Inspector for Nomisma project. It is comprised of a combination of java hints (yellow) and findbugs (red).

## Conclusion

Creating a list of the different coding tools at our disposal is liberating in the sense that we, as programmers, are able to capture the key aspects of the design (especially from a security standpoint) from which we are able to focus on. By listing the compilers and tools we are using, we are able to keep version control of not only our code, but also the foundations of our code. We can continually upgrade our tools in order to improve how the code functions. For example, updates to the MySQL database could improve query times, or a new version of the Netbeans static analysis tool might be able to track a wider range of bugs and warnings. Speaking of the analysis tool, we were amazed by how well it was able to detect many erroneous lines of code in our program that four sets of eyes weren't able to detect. While it may not capture every warning or bug, it already has helped us tremendously by notifying us of the catch exception issue in some of our methods. Finally, researching a list of deprecated functions essentially prevented us from making costly mistakes. By including alternative methods, future vulnerabilities may have been fixed.

Mark Bercasio • Barryn Chun • Egan Nakano • Shaziney Natividad

# nómisma νόμισμα

## Verification

ICS 491 – Assignment 4: Security Development Lifecycle



**Team Omega**  
July 31, 2016

## **Introduction**

For our group to maintain an acceptable level of code security and quality in Nomisma, it was imperative that we implement and various forms of data verification. The three main ways we tested our program for vulnerabilities were through dynamic analysis, fuzz testing, and through an attack service review. Dynamic analysis lets us perform checks on functionality, to monitor our application for privilege issues, connection issues, and other critical security problems. Fuzz testing is the deliberate induction of program failure by way of introduced malformed or random data, which helps to reveal potential security issues with minimal resource investment. Finally an attack surface review helps to ensure that design or implementation changes to our application have been taken into account and to identify any new attack vectors that may have emerged as a result of such changes are identified and mitigated.

## **Dynamic Analysis**

Once again, the team used the static analysis tool provided by Netbeans to check for possible security vulnerabilities and bugs. We began analysis by running the tool on our JDBC driver class, from which we were able to locate various connection issues regarding the database. For one thing, we were using two different schemas which not only caused performance issues, but also query issues. In addition, the tool notified us of certain variables being public to the user which were security issues (so we changed properties of the variables).

While we were not able to detect any other critical bugs, the analysis tool did help in cleaning up our code for readability. It notified us of unused packages and possible exceptions that could be narrowed or specified for better clarity. Given how we are still using placeholders for some classes, the tool was kind enough to mention how certain methods are never called.

As for input/output, we checked for various SQL injection commands in the username input. From what we've learned in the previous weeks, we sanitized the user input as to prevent unwanted commands. Given how we aren't using HTML or javascript, we weren't too worried about XSS attacks. However, we still needed to validate user input as to prevent brute force attacks (which we will be working on in our next implementation). We've also tested out our other transaction classes in order to see if they are updating the database correctly. We did notice how certain edge cases would lead the user to have a negative balance and we're still in the air whether or not to keep that as a bug or a feature (it's a good thing to know your debt). Otherwise, it seems our project is coming along as planned, though we can still continue researching up possible security threats and fixing our code to handle other vulnerabilities.

## **Fuzz Testing**

One possible hack that can break our system is a simple brute force attack. As of now, our username and password policies aren't enacted, so users can enter simple alpha-numerical passwords of any length. Because of this, it is much easier to simply guess a user's account information through repeated tries. We did not have a bot to continually input account and password combinations, so we were unsuccessful hacking the program in the regard. However, if we did have the capabilities of brute forcing the system, I'm sure we'd have access. Though not yet implemented, we plan to limit how many login attempts a user can have in entering the system.

Another hack that we forgot to account for was SQL injection. At first, we simply took in user input and turned them into variables which would be appended on to a SQL query as a string. The first SQL injection we attempted ultimately released all the information of the account table. So, we fixed this issue by implementing PreparedStatements that would directly input any data into the tables, rather than running the possible SQL injection.

A third (but yet untested) vulnerability may lie in the connection portion of our Nomisma program, considering the username and password to access the database are readily available in the source code. If reverse-engineering Java code is a possibility, then it could be an open door into the SQL server were it hosted remotely, based on that account's permissions. We will figure out a proper way to do SQL authentication by the release of our project.

### **Attack Surface Review**

From the tools we listed in the previous weeks, we haven't had any new updates or changes in how the work. However, we did add onto the list of testing tools by using JUnit to test for things such as input/output of our various methods. Since the timespan of the project is just a few weeks, there probably is not going to be any major updates to the tools we use. In addition, the only tool that may would need an update in terms of a security standpoint would be our MySQL database, which at the moment of writing this report, is still up to date.

In case of vulnerabilities, we looked back into deprecated functions and made sure that none of our additional methods had them in use. In addition, we took a look back to our threat modeling diagram and found that we needed to make sure all vulnerabilities regarding the database needed to be fixed (prime target for exploit). As such, we made changes to how we ran our MySQL connector so that our queries would be safer (PreparedStatements).

### **Conclusion**

After having completed the three types of verification, we were able to identify and fix a number of flaws in our program. From our dynamic analysis we found and fixed a number of non-critical minor bugs, as well as an issue with incompatible schema, and identified that we still need to improve validation of user input. From our fuzz testing we identified and fixed three main vulnerabilities in Nomisma, the first being a simple brute force attack, which we plan to counter with a yet to be implemented log in limit. The second was SQL injection, which we countered by implementing prepared statements, rather than direct input. And the third which may lie in the connection portion of our program. Also in doing an attack surface review we found that for the scope of this project, very little has changed in terms of tools, with the exception of perhaps our MySQL database. And as far as vulnerabilities, we made sure to thoroughly examine our code for depreciated functions and to fix prime targets for exploits. Implementing the verification step of the security development lifecycle has been a very beneficial and useful exercise, as evidenced by the improvements brought about by our findings.

### **References**

---

<https://www.microsoft.com/en-us/SDL/process/verification.aspx>

Mark Bercasio • Barryn Chun • Egan Nakano • Shaziney Natividad

# nómisma νόμισμα

Incident Response and Release  
ICS 491 - Assignment 5: SDL - Release



**Team Omega**  
August 7, 2016

## **Introduction**

As Nomisma nears its completion, it is important to set aside resources in order to maintain security procedures. An incident response team is needed for specific matters regarding client to company relations, legal correctness in the program's release and use, and escalation management should there be a security breach. If ever there is a case in which user information is compromised, a privacy escalation process is enacted to mitigate any further losses and to inform the public of possible accounts that have been compromised. Finally, a security review is done to analyze what security rating our project is currently at. From this, we can pinpoint ways of fixing certain issues in a possible release patch or project overhaul (if required).

## **Incident Response Plan**

### **Privacy Escalation Team**

Escalation Manager (Mark Bercasio): The escalation manager must be ready to handle any cyber-security breaches and needs to work quickly in order to mitigate possible losses while investigating the vulnerabilities that resulted in the exploit.

Legal Representative (Barryn Chun): The legal representative looks into the lawful actions in which the company collects user information and data in order to adhere to the legality of set laws. Changes in company policy must first be passed by the legal representative before they can be set.

Public Relations Representative(Egan Nakano): The public relations representative is tasked with maintaining company reputability and communicating with the client-base possible issues and changes regarding company policy or any breaches in user security.

### **Group Emergency Contact**

Group Email: teamomega491@gmail.com

### **Privacy Escalation Process**

A *privacy escalation* is defined as an internal process in which the details of a privacy related incident are communicated. A privacy escalation would be applicable in the event of a data breach or theft, a failure to meet privacy commitments, privacy related lawsuits or regulatory inquiries, contact from media outlets or privacy advocate groups regarding an incident, or other like incidents that may arise.

Using the provided group email, any employee can contact the team regarding a potential privacy escalation and thereby submit a privacy escalation request. At the first notice of an incident, the escalation manager will begin by reviewing the content of the escalation to determine if additional information is required. If that is the case, then the escalation manager is responsible for working with the person or persons reporting and any other necessary contacts to determine the source, impact and scope of the incident, validity of the incident/situation, and compile a summary of the known facts and timeline resolution expectations, after which the information should be distributed to the appropriate team member(s) as needed, ensuring that all aspects of the escalation can be resolved and resolution should then be sought.

Appropriate resolutions to the incident should be determined by the team in conjunction with the reporting party and/or any other relevant contacts. Appropriate resolutions may include: communications and training, human resource actions, as applicable in the deliberate misuse of data, internal incident management, or external communications. External communications would include things like: breach notification, public relations outreach, documentation updates, online published help articles, or even product or service changes if necessary. After all of the appropriate resolutions are in place, the team should

conduct a review of the effectiveness of all taken actions. An effective resolution is considered one that resolves the concerns of the reporting party and associated user concerns, in addition to ensuring similar events do not occur in the future.

### **Final Security Review**

After going through the completed program a few times, we have decided to grade it with a passed FSR, but with exceptions. This is because we are unable to provide database access without actually including a hardcoded username and password required to access the SQL database. After researching possible fixes, it seems the best way to fix the issue would be externalizing the connection string in a different file. The file would have OS restrictions on it as to require security elevation in order to access it. In addition, the file would need to be encrypted so that its contents can't be read. Another fix would require the creation of new limited SQL accounts for each user, which would be too tedious to implement with the scope of our project. The project was intended to work with a client-server connection and the database access string would be limited to the server as to prevent people from reverse-engineering the login information.

### **Conclusion**

In order for our project to succeed post release, the team is going to need to handle maintenance checks that hinder a lot on security personnel. A privacy escalation team is almost mandatory in project requiring user input and database management. Rather than hoping hackers don't target the system, it is best to expect it and prepare. Escalation managers are there in order to plan for such attacks and most importantly, to handle the situation after attacks do occur. Legal representatives work to ensure that company actions are legally sound and can be enforced by law. Public relations are there so that company reputability is not at stake after an attack. A privacy escalation process is set up so that breaches can be handled in an organized manner as to keep company integrity as well as user security. A review of what has occurred is then done to make sure any future attacks are handled to the best of the team's abilities. The final security review of a passed FSR with exceptions makes it clear that the program is by no means perfect in a security sense. An issue regarding SQL access cannot be fixed without major changes to the design and functionality of the program, which, with given the proper time and resources, can be implemented in order to appease to a larger audience.

### **Release and Archive**

<https://github.com/markent94/Nomisma>