Date: 16 December 2022

**User Management - Threat Model**

**Delivery Order: Some Project**

**For**

**Some Organization**

Version 1.0

|  |
| --- |
|  |

Table of Contents

[1 Overview 3](#_Toc456074981)

[1.1 Scope 3](#_Toc456074982)

[1.2 Threat Risk Modeling 3](#_Toc456074983)

[2 Threat Assessment 4](#_Toc456074984)

[2.1 DREAD 4](#_Toc456074985)

[2.1.1 Damage Potential 4](#_Toc456074986)

[2.1.2 Reproducibility 5](#_Toc456074987)

[2.1.3 Exploitability 5](#_Toc456074988)

[2.1.4 Affected Users 5](#_Toc456074989)

[2.1.5 Discoverability 5](#_Toc456074990)

[2.2 OWASP (Top 10) 6](#_Toc456074991)

[2.2.1 Injection (APP3540/APP3570/APP3810) 6](#_Toc456074992)

[2.2.2 Broken Authentication and Session Management (APP3350/APP3370/APP3405/APP3410/APP3415/APP6250) 6](#_Toc456074993)

[2.2.3 Cross-Site Scripting (XSS) (APP3580) 7](#_Toc456074994)

[2.2.4 Insecure Direct Object References (APP3450/APP3600) 7](#_Toc456074995)

[2.2.5 Security Misconfiguration (APP3110/APP3370/APP3450/APP6250) 8](#_Toc456074996)

[2.2.6 Sensitive Data Exposure (APP3210/APP3220/APP3250/APP3310/APP3330/APP3340) 8](#_Toc456074997)

[2.2.7 Missing Function Level Access Control (APP3450/APP3460/APP3480) 9](#_Toc456074998)

[2.2.8 Cross-Site Request Forgery (APP3585) 9](#_Toc456074999)

[2.2.9 Using Components with Known Vulnerabilities (APP2080) 10](#_Toc456075000)

[2.2.10 Unvalidated Redirects and Forwards (APP3600) 10](#_Toc456075001)

[2.3 STRIDE 11](#_Toc456075002)

[2.3.1 Spoofing Identity 11](#_Toc456075003)

[2.3.2 Tampering with Data (APP3510) 11](#_Toc456075004)

[2.3.3 Repudiation (APP3640/APP3680/APP6140) 11](#_Toc456075005)

[2.3.4 Information Disclosure 12](#_Toc456075006)

[2.3.5 Denial of Service 12](#_Toc456075007)

[2.3.6 Elevation of Privilege 13](#_Toc456075008)

[3 References 14](#_Toc456075009)

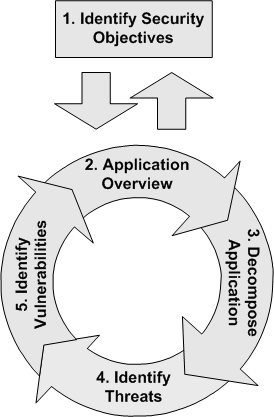
# Overview

## Scope

The intent of this document is to identify and provide mitigation for the critical web-based vulnerabilities and how they may affect components within the SOME PROJECT Architecture. Additionally, some best practices are enumerated with certain vulnerabilities to establish potential governance for future SOME PROJECT GOTS applications.

## Threat Risk Modeling

The Open Web Application Security Project (OWASP) recommends Microsoft’s threat modeling process because it works well for addressing the unique challenges facing web application security and is simple to learn and adopt by designers, developers, code reviewers, and the quality assurance team. There are five major steps encompassed by the Microsoft Threat Modeling Process:

1. Identify Security Objectives
2. [](http://www.google.com/url?sa=i&rct=j&q=&esrc=s&source=images&cd=&cad=rja&uact=8&docid=OAkNr2qVzOqS2M&tbnid=UwqiZikMjndXFM:&ved=0CAUQjRw&url=http://msdn.microsoft.com/en-us/library/ff648006.aspx&ei=cULJU8byGen48QGR7oDgDA&bvm=bv.71198958,d.aWw&psig=AFQjCNH_ACjLYpn2v0RG_mecyk__-ixOsg&ust=1405785067407519)Survey the Application
3. Decompose the Application
4. Identify Threats
5. Identify Vulnerabilities

# Threat Assessment

The threat risk modeling process has five steps, enumerated below. They are:

1. Identify Security Objectives – performed by US Navy with Application Security and Development Technical Implementation Guide (STIG)
2. Survey the Application – Conducted by SOME GROUP SOME PROJECT team 2017/02
3. Decompose it – Developers are aware of the authentication and authorization techniques used within the SOME PROJECT Architecture as well as the security measures identified in this document.
4. Identify Threats – Performed in conjunction with the DREAD and STRIDE models utilized below.
5. Identify Vulnerabilities – Enumerated in this document.

Note that all perspectives on this document take into account the defenses in place in the SOME PROJECT Architecture as well as the User Registration application for Increment 1.

## DREAD

DREAD is a classification scheme for quantifying, comparing and prioritizing the amount of risk presented by each evaluated threat. The DREAD acronym is formed from the first letter of each category below. DREAD modeling influences the thinking behind setting the risk rating, and is also used directly to sort the risks. The DREAD algorithm, shown below, is used to compute a risk value, which is an average of all five categories.

Risk\_DREAD = (DAMAGE + REPRODUCIBILITY + EXPLOITABILITY + AFFECTED USERS + DISCOVERABILITY) / 5

The calculation always produces a number between 0 and 10; the higher the number, the more serious the risk. Here are some examples of how to quantify the DREAD categories.

### Damage Potential

If a threat exploit occurs, how much damage will be caused?

* 0 = Nothing
* 5 = Individual user data is compromised or affected.
* 10 = Complete system or data destruction

### Reproducibility

How easy is it to reproduce the threat exploit?

* 0 = Very hard or impossible, even for administrators of the application.
* 5 = One or two steps required, may need to be an authorized user.
* 10 = Just a web browser and the address bar is sufficient, without authentication.

### Exploitability

What is needed to exploit this threat?

* 0 = Advanced programming and networking knowledge, with custom or advanced attack tools.
* 5 = Malware exists on the Internet, or an exploit is easily performed, using available attack tools.
* 10 = Just a web browser

### Affected Users

How many users will be affected?

* 0 = None
* 5 = Some users, but not all
* 10 = All users

### Discoverability

How easy is it to discover this threat?

* 0 = Very hard to impossible; requires source code or administrative access.
* 5 = Can figure it out by guessing or by monitoring network traces.
* 9 = Details of faults like this are already in the public domain and can be easily discovered using a search engine.
* 10 = The information is visible in the web browser address bar or in a form.

**Note:** When performing a security review of an existing application, “Discoverability” will often be set to 10 by convention, as it is assumed the threat issues will be discovered.

## OWASP (Top 10)

The OWASP Top 10 List contains the most common vulnerabilities to take into account when performing threat assessments on web applications. This list is identified directly in the *Application Security and Development STIG v4* underSTIG ID *APSC-DV-001460.*

### Injection (APP3540/APP3570/APP3810)

In an injection attack, the attacker sends simple text-based attacks that exploit the syntax of the targeted interpreter. Almost any source of data can be an injection vector, including internal sources. Injection attacks can be used to run malicious queries or code, and can result in leaking or modifying sensitive information. The best way to find out if an application is vulnerable to injection is to verify that all use of interpreters clearly separates untrusted data from the command or query. For SQL calls, this means using bind variables in all prepared statements and stored procedures, and avoiding dynamic queries.

**Mitigation:** The User Management application is not yet integrated with a database, so SQL concerns are nonexistent. All user input transferred with external API calls to OpenAM and Portal for ArcGIS is properly validated using Java specification JSR 303 (Bean Validation).

4.6 = (DAMAGE (8) + REPRODUCIBILITY (3) + EXPLOITABILITY (3) + AFFECTED USERS (10) + DISCOVERABILITY (0)) / 5

### Broken Authentication and Session Management (APP3350/APP3370/APP3405/APP3410/APP3415/APP6250)

Attacker uses leaks or flaws in the authentication or session management functions (e.g., exposed accounts, passwords, session IDs) to impersonate users.

Session management encompasses the techniques employed by web applications to transparently authorize a user for every HTTP request without having the user repeatedly login. The onus of session management must be borne by the application itself due to the stateless nature of the underlying HTTP protocol. Session management entails the application sending the client (in most cases, a web browser) a session token after successful authentication. In most cases, this token is passed via the Set-Cookie directive of HTTP and is stored on the client. The session token must then be sent by the client along with every HTTP request to the server to identify itself to the web-based application. The application can then determine whether the client is authorized to access the page being requested.

Session management mechanisms can be broadly classified into client-side and server-side mechanisms. This classification is based on the contents of the session token passed between the client and the application.

**Mitigation:** SOME PROJECT applications use OpenAM as the Authentication Brokerage Service to enforce authentication and authorization for access control to applications within the SOME PROJECT Architecture. All session management, including the creation of system-generated session identifiers, encryption of session information, and enforcement of session validity are managed by OpenAM. On each page request to applications within the SOME PROJECT, OpenAM performs a check on the user’s session token to ensure the session is associated to an authenticated user.

6 = (DAMAGE (7) + REPRODUCIBILITY (2) + EXPLOITABILITY (10) + AFFECTED USERS (10) + DISCOVERABILITY (6)) / 5

### Cross-Site Scripting (XSS) (APSC-DV-002490)

Attacker sends text-based attack scripts that exploit the interpreter in the browser. Almost any source of data can be an attack vector, including internal sources such as data from the database.

Applications are vulnerable if they do not ensure that all user supplied input is properly escaped, or they do not verify it to be safe via input validation, before including that input in the output page. Without proper output escaping or validation, such input will be treated as active content in the browser.

**Mitigation:** The User Management application uses Bean Validation (Java specification JSR 303) on all user input to verify its safety. Additionally, the HTTP Header “X-XSS-Protection” is set for all outgoing responses to enable the browser’s built-in XSS filter. The HTTP Header “Content-Security-Policy” is also added for refined XSS protection for newer browsers.

6.4 = (DAMAGE (9) + REPRODUCIBILITY (5) + EXPLOITABILITY (5) + AFFECTED USERS (8) + DISCOVERABILITY (5)) / 5

### Insecure Direct Object References (APP3450/APP3600)

Attacker, who is an authorized system user, simply changes a parameter value that directly refers to a system object to another object the user isn’t authorized for. Is access granted?

Applications frequently use the actual name or key of an object when generating web pages. Applications don’t always verify the user is authorized for the target object. This results in an insecure direct object reference flaw. Testers can easily manipulate parameter values to detect such flaws. Code analysis quickly shows whether authorization is properly verified.

**Mitigation:** Access to the User Management application requires authentication via two-way SSL with DoD PKI (CAC) credentials. User account management is controlled by OpenAM leveraging the hosting agency’s Microsoft Active Directory as the directory server to store user information. SOME PROJECT users are directly mapped (via user attribution) to their valid DoD PKI credentials.

The User Management application’s URLs are protected via OpenAM authorization policies, where access to each URL endpoint is associated to the intended authorized group(s). Additionally, exposure of elevated content is not made available through HTTP request parameters, as the domain of these accepted values is restricted to avoid such an exploit.

8.6 = (DAMAGE (5) + REPRODUCIBILITY (10) + EXPLOITABILITY (10) + AFFECTED USERS (9) + DISCOVERABILITY (9)) / 5

### Security Misconfiguration (APP3110/APP3370/APP3450/APP6250)

Attacker accesses default accounts, unused pages, unpatched flaws, unprotected files and directories, etc. to gain unauthorized access to or knowledge of the system.

Security misconfiguration can happen at any level of an application stack, including the platform, web server, application server, database, framework, and custom code. Developers and system administrators need to work together to ensure that the entire stack is configured properly. Automated scanners are useful for detecting missing patches, misconfigurations, use of default accounts, unnecessary services, etc.

**Mitigation:** All applicable Security Technical Implementation Guides (STIGs) have been applied to the operating system, application servers, web servers, and database servers existent in the SOME PROJECT Architecture. Additional security hardening guidance from OWASP for Apache Tomcat (the User Management application’s Java container) has been followed. Default accounts and applications have been removed from these architecture components. Additionally, the Principle of Least Privilege has been applied to OS user accounts to diminish risk of unauthorized access. Ansible has been integrated into the deployment process for automated provisioning of these security configuration items.

5.6 = (DAMAGE (10) + REPRODUCIBILITY (5) + EXPLOITABILITY (4) + AFFECTED USERS (5) + DISCOVERABILITY (4)) / 5

### Sensitive Data Exposure (APP3210/APP3220/APP3250/APP3310/APP3330/APP3340)

Attackers typically don’t break crypto directly. They break something else, such as steal keys, do man-in-the-middle attacks, or steal clear text data off the server, while in transit, or from the user’s browser.

The most common flaw is simply not encrypting sensitive dat2. When crypto is employed, weak key generation and management, and weak algorithm usage is common, particularly weak password hashing techniques. Browser weaknesses are very common and easy to detect, but hard to exploit on a large scale. External attackers have difficulty detecting server side flaws due to limited access and they are also usually hard to exploit.

**Mitigation:** The User Management application utilizes two-way SSL via DoD PKI to secure communications both to and from the user. Keys and associated cryptographic algorithms are set to be FIPS 140-2 compliant to meet expected DoD security standards. The “autocomplete=off” attribute exists on the User Registration web page’s form to disable caching of its input fields.

5.6 = (DAMAGE (5) + REPRODUCIBILITY (5) + EXPLOITABILITY (5) + AFFECTED USERS (4) + DISCOVERABILITY (9)) / 5

### Missing Function Level Access Control (APP3450/APP3460/APP3480)

Attacker, who is an authorized system user, simply changes the URL to a parameter to a privileged function. Is access granted? Anonymous users could access private pages that aren’t protected.

Applications do not always protect application functions properly. Sometimes, function level protection is managed via configuration, and the system is misconfigured. Sometimes, developers must include the proper code checks, and they forget.

Detecting such flaws is easy. The hardest part is identifying which pages (URLs) or functions exist to attack.

**Mitigation:** The User Management application does not distinguish elevated access to its resources, as only the User Registration service currently exists. Access to the RESTful URL for registration is enforced by an OpenAM authorization policy. A user attempting to force browse to a specific URL they are not authorized for will result in a redirect to the application’s default error page.

5.8 = (DAMAGE (5) + REPRODUCIBILITY (4) + EXPLOITABILITY (10) + AFFECTED USERS (4) + DISCOVERABILITY (6)) / 5

### Cross-Site Request Forgery (APSC-DV-002500)

Attacker creates forged HTTP requests and tricks a victim into submitting them via image tags, XSS, or numerous other techniques. If the user is authenticated, the attack succeeds.

CSRF takes advantage the fact that most web apps allow attackers to predict all the details of a particular action. Because browsers send credentials like session cookies automatically, attackers can create malicious web pages which generate forged requests that are indistinguishable from legitimate ones. Detection of CSRF flaws is fairly easy via penetration testing or code analysis.

**Mitigation:** All requests to the User Management application’s RESTful web services are filtered to ensure the existence of the “X-Requested-With” custom header, ensuring the request is restricted to JavaScript AJAX requests within the site’s same origin.  
  
*Reference*: https://www.owasp.org/index.php/Cross-Site\_Request\_Forgery\_(CSRF)\_Prevention\_Cheat\_Sheet#Protecting\_REST\_Services:\_Use\_of\_Custom\_Request\_Headers

5 = (DAMAGE (10) + REPRODUCIBILITY (1) + EXPLOITABILITY (1) + AFFECTED USERS (10) + DISCOVERABILITY (5)) / 5

### Using Components with Known Vulnerabilities (APP2080)

Attacker identifies a weak component through scanning or manual analysis. He customizes the exploit as needed and executes the attack. It gets more difficult if the used component is deep in the application.

Virtually every application has these issues because most development teams don’t focus on ensuring their components/libraries are up to date. In many cases, the developers don’t even know all the components they are using, never mind their versions. Component dependencies make things even worse.

**Mitigation:** Software products in use within the SOME PROJECT Architecture must be DADMS-approved and align with the hosting agency’s software update policies. Additionally, IAVM alerts are closely monitored so that affected software is scheduled for patching as soon as updates come available.

5 = (DAMAGE (5) + REPRODUCIBILITY (5) + EXPLOITABILITY (5) + AFFECTED USERS (5) + DISCOVERABILITY (5)) / 5

### Unvalidated Redirects and Forwards (APP3600)

Attacker links to invalidated redirect and tricks victims into clicking it. Victims are more likely to click on it, since the link is to a valid site. Attacker targets unsafe forward to bypass security checks.

Applications frequently redirect users to other pages, or use internal forwards in a similar manner. Sometimes the target page is specified in an unvalidated parameter, allowing attackers to choose the destination page. Detecting unchecked redirects is easy. Look for redirects where you can set the full URL. Unchecked forwards are harder, because they target internal pages.

**Mitigation:** OpenAM uses the “goto” URL query parameter for URL redirection. A “Valid goto URL domains” parameter exists within OpenAM configuration to remove the possibility of redirection or forwarding to URLs outside the defined domain. The User Management application does not include user parameters when forming a redirect destination.

6.8 = (DAMAGE (5) + REPRODUCIBILITY (5) + EXPLOITABILITY (5) + AFFECTED USERS (10) + DISCOVERABILITY (9)) / 5

## STRIDE

STRIDE is a classification scheme for characterizing known threats according to the kinds of exploit that are used (or motivation of the attacker). The STRIDE acronym is formed from the first letter of each of the following categories.

### Spoofing Identity

“Identity spoofing” is a key risk for applications that have many users but provide a single execution context at the application and database level. In particular, users should not be able to become any other user or assume the attributes of another user.

**Mitigation:** The User Management application leverages SOME PROJECT’s OpenAM instance for authentication to prevent the possibility of identity spoofing. OpenAM leverages the hosting agency’s Microsoft Active Directory instance to store SOME PROJECT user account information, including mappings of DoD PKI user information (the user’s DoD ID) to a specific user attribute.

### Tampering with Data (APP3510)

Users can potentially change data delivered to them, return it, and thereby potentially manipulate client-side validation, GET and POST results, cookies, HTTP headers, and so forth. The application should not send data to the user, such as interest rates or periods, which are obtainable only from within the application itself. The application should also carefully check data received from the user and validate that it is sane and applicable before storing or using it.

**Mitigation:** The User Management application utilizes client-side validation with regular expressions where appropriate on the User Registration web page. Server-side validation is performed using the Java specification JSR 303, Bean Validation, which is verified prior to application processing on the user input.

### Repudiation (APP3640/APP3680/APP6140)

Users may dispute transactions if there is insufficient auditing or recordkeeping of their activity. For example, if a user says, “But I didn’t transfer any money to this external account!”, and you cannot track his/her activities through the application, then it is extremely likely that the transaction will have to be written off as a loss. Therefore, consider if the application requires non-repudiation controls, such as web access logs, audit trails at each tier, or the same user context from top to bottom. Preferably, the application should run with the user’s privileges, not more, but this may not be possible with many off-the-shelf application frameworks.

**Mitigation:** All page requests to the User Management application are enforced by an OpenAM Web Policy Agent; additionally, these page requests are recorded within the OpenAM Web Policy Agent’s audit logs. The User Registration service captures users who are either successful or unsuccessful in registering a new user account.

### Information Disclosure

Users are rightfully wary of submitting private details to a system. If it is possible for an attacker to publicly reveal user data at large, whether anonymously or as an authorized user, there will be an immediate loss of confidence and a substantial period of reputation loss. Therefore, applications must include strong controls to prevent user ID tampering and abuse, particularly if they use a single context to run the entire application. Also, consider if the user’s web browser may leak information. Some web browsers may ignore the no caching directives in HTTP headers or handle them incorrectly. In a corresponding fashion, every secure application has a responsibility to minimize the amount of information stored by the web browser, just in case it leaks or leaves information behind, which can be used by an attacker to learn details about the application, the user, or to potentially become that user. Finally, in implementing persistent values, keep in mind that the use of hidden fields is insecure by nature.

Such storage should not be relied on to secure sensitive information or to provide adequate personal privacy safeguards.

**Mitigation:** The User Management application does not use local storage mechanisms or caching on the client-side. Server-side persistence is not explicitly in use by the application; user registration leverages external user stores for persistence. OpenAM ensures valid authentication before access control decisions are performed.

### Denial of Service

Application designers should be aware that their applications may be subject to a denial of service attack. Therefore, the use of expensive resources such as large files, complex calculations, heavy-duty searches, or long queries should be reserved for authenticated and authorized users, and not available to anonymous users.

For applications that do not have this luxury, every facet of the application should be engineered to perform as little work as possible, to use fast and few database queries, to avoid exposing large files or unique links per user, in order to prevent simple denial of service attacks.

**Mitigation:** Access to SOME PROJECT applications, including the User Management application, does not support anonymous user access. All access requests require authentication and authorization from OpenAM.

### Elevation of Privilege

If an application provides distinct user and administrative roles, then it is vital to ensure that the user cannot elevate his/her role to a higher privilege one. In particular, simply not displaying privileged role links is insufficient. Instead, all actions should be gated through an authorization matrix, to ensure that only the permitted roles can access privileged functionality.

**Mitigation:** User account management is delegated to OpenAM, the Authentication Brokerage Service. The Principle of Least Privilege is applied to defined authorization policies for the User Management application. Each page request is enforced by OpenAM Web Policy Agents that utilize the current user’s credentials against the application-level policies defined in OpenAM.

# References

Application Threat Modeling

(<https://www.owasp.org/index.php/Application_Threat_Modeling>)

Threat Risk Modeling

(<https://www.owasp.org/index.php/Threat_Risk_Modeling>)

Threat Modeling Web Applications

(<http://msdn.microsoft.com/library/ms978516.aspx>)

At a Glance: Web Application Threat Modeling

(<http://msdn.microsoft.com/en-US/library/ms978523.aspx>)

How To: Create a Threat Model for a Web Application at Design Time

(<http://msdn.microsoft.com/en-US/library/ms978527.aspx>)

Walkthrough: Creating a Threat Model for a Web Application

(<http://msdn.microsoft.com/en-US/library/ms978538.aspx>)

Template: Web Application Threat Model

(<http://msdn.microsoft.com/en-US/library/ms978531.aspx>)

Template Sample: Web Application Threat Model

(<http://msdn.microsoft.com/en-US/library/ms978534.aspx>)

Cheat Sheet: Web Application Security Frame

(<http://msdn.microsoft.com/en-US/library/ms978518.aspx>)