GPMS Vulnerability Documentation

Vulnerability and type	Code Responsible	Attack Description	Code Fixes
XSS on User signup page	File: UserService.java Method: signUpUser()	Malicious user can use XSS to store scripts in fields such as the username or name fields. This script will then run whenever a user accesses a page where this information will appear.	Uses a utility class called UserInputValidator.java that uses whitelisting techniques to filter user input. This is implemented within the signup process in UserService.java lines: 841-866. (Code 1 and 5)
Behind-the-scenes proposal updates from users with improper permissions	File: ProposalService.java Method: saveProposal()	Old implementation saved all sections of a proposal for every user regardless of their permissions. If a malicious user were to modify the front-end and change the contents in the fields, then it would save into the proposal.	Implements proper XACML decision making by building a JSON object with the proper resources inside. This is implemented in ProposalService.java in saveProposal() lines: 1583-1690. (Code 3)
File-path Injection	File: FileService.java Method: downloadFile() & uploadFile()	A user can always download/upload a file regardless if their session is active or not. As long as they have the GET URL they can download any file as long as they have the name.	Checks for valid user session before allowing uploading/downloading. This is implemented in FileService.java in uploadFile() and downloadFile() lines: 69-75 and 137-143. (Code 4)
Automatic user registration denial of service	File: UserService.java Method:signUpUser()	A malicious user could create a program to automatically register new users continuously. This would eventually lead to the system crashing and the web application becoming unavailable.	The signup page now uses reCAPTCHA to validate that the user is not a bot. Proper back-end and front-end validation is included to make sure the the reCAPTCHA response is not being faked. Implemented in UserService.java in

			signUpUser() lines: 861-866. Also uses a utility class called VerifyRecaptcha.java to validate the response from the reCAPTCHA widget. A simple front-end validation was created in SignUp.js lines: 357-368, but this code simply exists for user convenience and must not be relied upon by itself. (Code 2 and 6)
Inefficient login process leading to denial of service	File: UserService.java Method: login()	Every time a user wants to login to the application, the application loads every user and their information into a data structure. With a lot of users in the database, this could lead to the system crashing.	Instead, the username/email is sent back to the UserProfileDAO to search for the specific user with the email/username. Implemented in UserService.java in login() lines: 923-969. (Code 7)
XSS in proposal fields	File: ProposalService.java and ProposalDAO.java Methods: Entirety of saving proposal process	Fields throughout the proposal process are susceptible to XSS. Primary fields including signature field, title field, and sponsor fields.	Whitelisting techniques from UserInputValidator.java are implemented during the saving process. Implemented primarily in ProposalDAO.java throughout the entirety of the file. (Code 5)
Improper front-end validation on sign-up process	File: SignUp.js Function: signUpUser()	JQuery validation rules did not conform to the rules that were in the signup process on the backend.	Added and changed rules in the JQuery validation to match the rules that exist on the backend. Implemented in SignUp.js on lines: 12-146. (Code 8)

Code

1. Code that validates user input during the signup process in UserService.java lines: 839-859

```
//Author: Patrick Chapman
//User input validation
UserInputValidator inputValidator = new UserInputValidator();

//This is kind of copy-paste, but might have to do
//for now.

//validate user name
inputValidator.validateUsernameInput(userInfo.get("UserName").textValue(), MAX_USERNAME_LENGTH);
//validate first name
inputValidator.validateInput(userInfo.get("FirstName").textValue(), MAX_NAME_LENGTH);
//validate last name
inputValidator.validateInput(userInfo.get("LastName").textValue(), MAX_NAME_LENGTH);
//validate  address
inputValidator.validateInput(userInfo.get("Street").textValue(), MAX_ADDR_LENGTH);
//validate user city
inputValidator.validateInput(userInfo.get("City").textValue(), MAX_CITY_LENGTH);
//validate user state
inputValidator.validateInput(userInfo.get("State").textValue(), MAX_STATE_LENGTH);
//validate user country
inputValidator.validateInput(userInfo.get("Country").textValue(), MAX_COUNTRY_LENGTH);
```

2. Code that passes reCAPTCHA widget response key to a verification method that sends key to Google. Located in UserService.java lines: 861-865.

```
//Back-end verification for recaptcha response
String gRecaptchaResponse = userInfo.get("Recaptcha").textValue();
if(!VerifyRecaptcha.verify(gRecaptchaResponse)){
        throw new Exception("Captcha did not verify correctly.");
}
```

3. Code that uses proper XACML decision making to determine what proposal sections to save for a user. In ProposalService.java lines: 1597-1960.

```
//Author: Patrick Chapman
//Update: 8/17/17
//Currently uses XACML policies to determine
//if user has permission to save initial proposal
//details.
//Update: 8/27/17
//Adding in XACML policy decision making for
//OSP section.
if (root.has("policyInfo")) {
        JsonNode policyInfo = root.get("policyInfo");
        if (policyInfo != null && policyInfo.isArray()
                && policyInfo.size() > 0) {
                HashMap<String, Multimap<String, String>> attrMap = proposalDAO
                                .generateAttributes(policyInfo);

                //Adding edit action and proposal information resources
                attrMap.replace("Action", PIActionReplace);
                attrMap.replace("Resource", PISectionReplace);
                String decision = ac.getXACMLdecision(attrMap);

                //From XACML decision
                if (decision.equals("Permit") ||
                        decision.equals("NotApplicable")) {
                                proposalDAO.getProjectInfo(existingProposal, proposalID
                                proposalInfo);
                                proposalDAO.getSponsorAndBudgetInfo(existingProposal, proposalID,
```

```
                                                proposalInfo);
                                        proposalDAO.getCostShareInfo(existingProposal, proposalID,
                                                proposalInfo);
                                        proposalDAO.getUniversityCommitments(existingProposal, proposalID,
                                                proposalInfo);
                                        proposalDAO.getConflictOfInterest(existingProposal, proposalID,
                                                proposalInfo);
                                        proposalDAO.getAdditionalInfo(existingProposal, proposalID,
                                                proposalInfo);
                                        proposalDAO.getCollaborationInfo(existingProposal, proposalID,
                                                proposalInfo);
                                        proposalDAO.getConfidentialInfo(existingProposal, proposalID,
                                                proposalInfo);
                        }



//OSP section resource
Multimap<String, String> OSPSectionReplace = ArrayListMultimap.create();
OSPSectionReplace.put("proposal.section", "OSP Section");

//Getting proposal user title
JsonNode proposalUserTitle = root.get("proposalUserTitle");

//Determining which user is editing OSP section
if(proposalUserTitle.textValue().equals(raTitle)){
        OSPSectionReplace.put(approveRA, "READYFORAPPROVAL");
}else if(proposalUserTitle.textValue().equals(directorTitle)) {
        OSPSectionReplace.put(approveDirector, "READYFORAPPROVAL");
}
//Creating OSP edit action
Multimap<String, String> ApproveActionReplace = ArrayListMultimap.create();
ApproveActionReplace.put("proposal.action", "Edit");

//Adding in attributes/resources for XACML decision
attrMap.replace("Action", ApproveActionReplace);
attrMap.replace("Resource", OSPSectionReplace);
String OSPdecision = ac.getXACMLdecision(attrMap);

if (!isAdminUser) {
        // OSP Section
                if (!proposalID.equals("0")
                        && !action.equals("Disapprove")
                        && OSPdecision.equals("Permit")) {

                                proposalDAO.getOSPSectionInfo(existingProposal,
                                proposalInfo);

                } else {
                        if (!proposalID.equals("0")) {
                                existingProposal.setOspSectionInfo(oldProposal
                                                .getOspSectionInfo());
                        }
                }
                proposalIsChanged = notifyUsersProposalStatusUpdate(
                                existingProposal, oldProposal, authorProfile,
                                proposalID, signatures, irbApprovalRequired,
                                requiredSignatures, authorUserName, root,
                                proposalUserTitle);
        } else {
                proposalIsChanged = notifyUsersProposalStatusUpdate(
                                existingProposal, oldProposal, authorProfile,
                                proposalID, authorUserName);
        }
}
```

```
}
}
//End of Patrick Code
```

4. Code checks to see if a valid user is logged in before allowed a download/upload. Located in FileService.java at lines: 69-75 and 137-143.

Part 1.

```
//Author: Patrick Chapman
//Validate User sesssion when downloading file
HttpSession session = request.getSession();
if (session.getAttribute("userProfileId") == null) {
        throw new ServletException("Invalid session!");
}
//End Patrick Code
```

Part 2.

```
//Author: Patrick Chapman
//Validate User sesssion when uploading file
HttpSession session = request.getSession();
if (session.getAttribute("userProfileId") == null) {
        throw new ServletException("Invalid session!");
}
//End Patrick Code
```

5. Code represents UserInputValidator.java used to validate incoming user input.

```
/**
 * Used for validating strings that the user of the application will pass in.
 *
 * @author Patrick Chapman
 * @date 3/23/17
 *
 * @update 7/13/2017 2:05 PM
 * Added validation  for numeric-only and usernames.
 */
public class UserInputValidator {

        /**@author Patrick Chapman
         *
         * Generic validation for user inputs. Does not allow for common
         * XSS characters and tags.
         *
         * @param userInput, input string to be validated
         * @param length, the length of the string to be validated
         * @throws Exception, if the string does not meet the username requirements
         */
        public void validateInput(String userInput, int length) throws Exception{
                System.out.println(userInput);
                String userInputCopy = userInput.toLowerCase();
                //Allow alpha-numeric, punctuation, and whitespace
                if(!userInputCopy.matches("^[a-zA-Z0-9,;.?!()\\s]*$") ||
                userInputCopy.contains("script")){
                        throw new ValidationException("Invalid characters passed in.");
                }

                if((userInputCopy.length() > length) || (userInputCopy.length()) == 0){
                        throw new ValidationException("Field does not contain the right amount of characters!");
                }
        }
```

```
	/**@author Patrick Chapman
	 *
	 * Validates user input for a field that is only allowed numeric
	 * values.
	 *
	 * @param userInput, input string to be validated
	 * @param length, the length of the string to be validated
	 * @throws Exception, if the string does not meet the username requirements
	 */
	public void validateNumberInput(String userInput, int length) throws Exception{
		System.out.println(userInput);
		String userInputCopy = userInput.toLowerCase();
		//Allow numbers only
		if(!userInputCopy.matches("^[0-9,.\\s]*$")){
			throw new ValidationException("Input may only contain numbers!");
		}
		if((userInputCopy.length() > length) || (userInputCopy.length() == 0)){
			throw new ValidationException("Fields do not contain the right amount of numbers!");
		}
	}

	/**@author Patrick Chapman
	 *
	 * Validates the username input from a user.
	 *
	 * @param userInput, input string to be validated
	 * @param length, the length of the string to be validated
	 * @throws Exception, if the string does not meet the username requirements
	 */
	public void validateUsernameInput(String userInput, int length) throws Exception{
		System.out.println(userInput);
		String userInputCopy = userInput.toLowerCase();
		//Allow alpha-numeric ONLY for the usernames
		if(!userInputCopy.matches("^[a-zA-z0-9]*$") ||
		userInputCopy.contains("script")){
			throw new ValidationException("Invalid characters passed in Username field.");
		}

		if((userInputCopy.length() > length) || (userInputCopy.length()) == 0){
			throw new ValidationException("Field does not contain the right amount of characters!");
		}
	}

	/**@author Patrick Chapman
	 *
	 * Validates the date input from a user.
	 *
	 * @param userInput, input string to be validated
	 * @param length, the length of the string to be validated
	 * @throws Exception, if the string does not meet the username requirements
	 */
	public void validateDateInput(String userInput, int length) throws Exception{
		System.out.println("tes: " + userInput);
		//Allow only date characters
		if(!userInput.matches("^[-a-zA-Z0-9:/\\s]*$") ||
		userInput.contains("script")){
			throw new ValidationException("Date field contains illegal characters.");
		}

		if((userInput.length() > length) || (userInput.length()) == 0){
			throw new ValidationException("Date field contains too many characters!");
		}
	}
}
```

6. Class validates the reCAPTCHA response given by the reCAPTCHA widget.

```
public class VerifyRecaptcha {

	public static final String url = "https://www.google.com/recaptcha/api/siteverify";
	public static final String secret = "6LfUzyMUAAAAAJvwrjA0recUUL-vaC8WPhB8Qjfc";
	private final static String USER_AGENT = "Mozilla/5.0";

	public static boolean verify(String gRecaptchaResponse) throws IOException {
		if (gRecaptchaResponse == null || "".equals(gRecaptchaResponse)) {
			return false;
		}

		try{
		URL obj = new URL(url);
		HttpsURLConnection con = (HttpsURLConnection) obj.openConnection();

		// add reuqest header
		con.setRequestMethod("POST");
		con.setRequestProperty("User-Agent", USER_AGENT);
		con.setRequestProperty("Accept-Language", "en-US,en;q=0.5");

		String postParams = "secret=" + secret + "&response="
					+ gRecaptchaResponse;

		// Send post request
		con.setDoOutput(true);
		DataOutputStream wr = new DataOutputStream(con.getOutputStream());
		wr.writeBytes(postParams);
		wr.flush();
		wr.close();

		int responseCode = con.getResponseCode();
		System.out.println("\nSending 'POST' request to URL : " + url);
		System.out.println("Post parameters : " + postParams);
		System.out.println("Response Code : " + responseCode);

		BufferedReader in = new BufferedReader(new InputStreamReader(
					con.getInputStream()));
		String inputLine;
		StringBuffer response = new StringBuffer();

		while ((inputLine = in.readLine()) != null) {
			response.append(inputLine);
		}
		in.close();

		// print result
		System.out.println(response.toString());

		//parse JSON response and return 'success' value
		JsonReader jsonReader = Json.createReader(new StringReader(response.toString()));
		JsonObject jsonObject = jsonReader.readObject();
		jsonReader.close();

		return jsonObject.getBoolean("success");
		}catch(Exception e){
			e.printStackTrace();
			return false;
		}
	}
}
```

7. Code that eliminates loading every user into a data structure during the login process. Located in UserService.java lines: 923-969.

```
//Author: Patrick Chapman
//Updated: 8/27/2017
//Rewrote login process to only search for user by email/username
//instead of loading a user profile for every single user.
//Update: 8/29/17
//Fixed error whenever user entered in account that did not exist

//Finds user by email or username
UserProfile user = userProfileDAO.findAnyUserWithSameEmail(email);
if(user == null){
		user = userProfileDAO.findAnyUserWithSameUserName(email);
}

boolean isFound = false;
//if user is found
if(user != null) {
		if (user.getUserAccount().getUserName().equals(email)
				|| user.getWorkEmails().contains(email)) {
				if (PasswordHash.validatePassword(password, user
						.getUserAccount().getPassword())
						&& !user.isDeleted()
						&& user.getUserAccount().isActive()
						&& !user.getUserAccount().isDeleted()) {
							isFound = true;

							userProfileDAO.setMySessionID(req, user.getId()
									.toString());
							java.net.URI location = new java.net.URI(
									"../Home.jsp");
							if (user.getUserAccount().isAdmin()) {
									location = new java.net.URI("../Dashboard.jsp");
							}
							return Response.seeOther(location).build();
							} else {
									isFound = false;
							}

		} else {
				isFound = false;
		}
						}
if (!isFound) {
		java.net.URI location = new java.net.URI(
						"../Login.jsp?msg=error");
		return Response.seeOther(location).build();
}
//End of Patrick code
```

8. Code for the rules in the SignUp.js lines: 12 - 146.

```
	var validator = $("#form1")
	.validate(
	   {
	      ignore: ".ignore",
	      rules: {
	         username: {
	            required: true,
```

```
            minlength: 3,
            maxlength: 20,
            noSpace: true,
            noSpecial: true
        },
        password: {
            required: true,
            minlength: 8,
            maxlength: 64,
            noSpace: true
        },
        confirm_password: {
            required: true,
            minlength: 8,
            maxlength: 64,
            equalTo: "#txtPassword"
        },
        workEmail: {
            required: true,
            email: true
        },
        firstName: {
            required: true,
            maxlength: 40,
            noSpecial: true
        },
        lastName: {
            required: true,
            maxlength: 40,
            noSpecial: true
        },
        dob: {
            required: true,
            dpDate: true
        },
        gender: {
            required: true
        },
        street: {
            required: true,
            minlength: 3,
            noSpecial: true
        },
        city: {
            required: true,
            noSpecial: true
        },
        state: {
            required: true,
            noSpecial: true
        },
        zip: {
            required: true
        },
        country: {
            required: true,
            noSpecial: true
        },
        mobileNumber: {
            required: true
        }
    },
    errorElement: "label",
    messages: {
        username: {
```

```
            required: "Please enter a username",
            minlength: "Your username must be between 3 and 20 characters",
            maxlength: "Your username must be between 3 and 20 characters",
            noSpace: "Username cannot contain spaces",
            noSpecial: "Username cannot contain special characters"
        },
        password: {
            required: "Please provide a password",
            minlength: "Your password must be between 8 and 64 characters",
            maxlength: "Your password must be between 8 and 64 characters", //Removed no spaces requirement
        },
        confirm_password: {
            required: "Please confirm your password",
            minlength: "Your password must be between 8 and 64 characters",
            equalTo: "Please enter the same password as above",
            maxlength: "Your password must be between 8 and 64 characters"
        },
        workEmail: {
            required: "Please enter your work email",
            email: "Please enter valid email id"
        },
        firstName: {
            required: "Please enter your firstname",
            maxlength: "Your firstname must be at most 40 characters long",
            noSpecial: "First name cannot contain special characters"
        },
        lastName: {
            required: "Please enter your lastname",
            maxlength: "Your lastname must be at most 40 characters long",
            noSpecial: "Last name cannot contain special characters"
        },
        dob: {
            required: "Please enter your date of birth",
            dpDate: "Please enter valid date"
        },
        gender: {
            required: "Please select your gender"
        },
        street: {
            required: "Please enter your street address",
            minlength: "Please enter valid your street address",
            noSpecial: "Street cannot contain special characters"
        },
        city: {
            required: "Please enter your city",
            noSpecial: "City cannot contain special characters"
        },
        state: {
            required: "Please select your city",
            noSpecial: "Street cannot contain special characters"
        },
        zip: {
            required: "Please enter your zip code"
        },
        country: {
            required: "Please select your country",
            noSpecial: "Country cannot contain special characters"
        },
        mobileNumber: {
            required: "Please enter your mobile phone number"
        }
    }
});
```