**Mass Assignment: Insecure Binder Configuration Development Mitigation SOP**

To ease development and increase productivity, most modern frameworks allow an object to be automatically instantiated and populated with the HTTP request parameters whose names match an attribute of the class to be bound. Automatic instantiation and population of objects speeds up development, but can lead to serious problems if implemented without caution. Any attribute in the bound classes, or nested classes, will be automatically bound to the HTTP request parameters. Therefore, malicious users will be able to assign a value to any attribute in bound or nested classes, even if they are not exposed to the client through web forms or API contracts.

**Classic Example**

public class User{

private String name;

private String lastname;

private int age;

private Details details;

//all other methods here

}

public class Details {

private Boolean is\_admin;

private int id;

private Date login\_date;

//all other methods here

}

@RequestMapping(method = RequestMethod.POST)

public String registerUser(@ModelAttribute(“user”) User user, BindingResult result, SessionStatus status){

if(db.save(user).hasErrors()){

return “CustomerForm”;

} else {

status.setComplete();

return “CustomerSuccess”;

}

}

It is a best practice to control which attributes will be bound to the model object so that even if attackers figure out other non-exposed attributes of the model or nested classes, they will not be able to bind arbitrary values from HTTP request parameters.

In this example above, there is no protection against attributes being bound that are not supposed to be edited by the user. This will bind the HTTP request parameters to any attribute in the User or Detail classes.

**Example**

The below implementation comes from ClaimController.java:

//Request Mappings

private static final String UPDATE\_CLAIM\_DATES\_REQUEST\_MAPPING = “/updateClaimDates”;

//Model Attributes

public static final String CLAIM = “claim”;

@ResponseBody

@Authorize(policy = SystemPolicy.EDIT\_CLAIM\_DETAILS)

@RequestMapping(value = UPDATE\_CLAIM\_DATES\_REQUEST\_MAPPING, method = RequestMethod.POST)

public AjaxResponse<ClaimSummary> claimDatesUpdate(ClaimSummary partialClaim, @RequestParam String claimID){

//pass claim summary data

}

**Defense Against Mass Assignment: Insecure BinderConfiguration**

It is security best practice to tell Spring specifically what fields to automatically bind to. This is done through a custom method annotated with @InitBinder that provides a “whitelist” of fields that are allowed to be bound for a particular object.

The code below demonstrates a very basic binder configuration:

@InitBinder

public void initBinder(WebDataBinder binder, WebRequest webRequest){

binder.setAllowedFields(“claimDateDt”, “suspenseDate”);

}

The two fields which are expected from the client are passed to the setAllowedFields method of the data binder. Now, any other fields that are set in the body of HTTP requests will not be bound to the ClaimSummary object.

If a client sent these attributes to the controller:

“claimID” : “1”

“claimDateDt” : “05/02/2015”

“suspenseDate” : “06/02/2015”

“veteranPersonId” : “2”

In the controller method:

RequestParam claimId=1;

The partialClaim object would have these values if allowedFields is NOT set:

claimDateDt=05/02/2015

suspenseDate=06/02/2015

veteranPersonId=2

The partialClaim object would have these values if allowedFields is set:

claimDateDt=05/02/2015

suspenseDate=06/02/2015

veteranPersonId=null

As you can see, veteranPersonId will remain a null value and not be set if allowedFields is properly configured and set. This ensures this value will not be inadvertently used/set during processing in the controller method.

***Other Remediation Strategies***

**Replace Model Attributes with Explicit Method Parameters**

Rather than pulling in the entire ClaimSummary object, only pull in the necessary attributes with @RequestParam annotations to auto-bind with explicit method parameters.

**Utilize Jackson Annotations**

The Jackson library can control the binding process by using the following annotations: @JsonIgnore, @JsonIgnoreProperties, @JsonIgnoreType, and @JsonInclude. By utilizing these, annotations on the POJO attributes, the binder can ignore those specific attributes. For example, in the User object below, the role attribute would be ignored:

public class User {

@JsonIgnore

private String role;

private String name;

}

**Data Transfer Objects (DTOs)**

DTOs can be used for an architectural approach. Using a DTO, input is not bound directly to the object and instead, only the fields that are meant to be edited by the user would be included in the DTO.

**Defense Example *(fixing Classic example above)***

Adding in the implementation of initbinder, specific fields are disallowed from being bound to the object. In this case, the user role, age, and admin privileges are unable to be set.

final String[] DISALLOWED\_FIELDS = new STRING[]{“details.role”, “details.age”, “is\_admin”};

@InitBinder

public void initBinder(WebDataBinder binder){

binder.setDisallowedFields(DISALLOWED\_FIELDS);

}

**Specific Explanation**

In the specific example, several attributes could be bound, including claimDateDt and suspenseDate. All attributes that should not be edited by the user should be disallowed from being bound as shown below.

**Recommendation**

//Request Mappings

private static final String UPDATE\_CLAIM\_DATES\_REQUEST\_MAPPING = “/updateClaimDates”;

//Model Attributes

public static final String CLAIM = “claim”;

**//DataBinder Fields**

**private static final List<String> UPDATE\_CLAIM\_INFO\_ALLOWED\_FIELDS = Arrays.asList("endProductClaimType.code", "endProductClaimType.name", "modifiedEndProductCd", "claimDateDt",  
"suspenseDate", "suspenseReasonCd", "suspenseReasonComments", "gulfWarRegistryInd");**

**@InitBinder(CLAIM)**

**public void initBinderClaim(WebDataBinder binder, WebRequest webRequest){**

**String requestUri =**

**((ServletWebRequest)webRequest).getRequest().getRequestURI();**

**if((requestUri) != null && (requestUri.contains(UPDATE\_CLAIM\_INFO\_REQUEST\_MAPPING))){**

**SystemWebBinderUtils.addAllowedFieldsToWebDataBinder(**

**UPDATE\_CLAIM\_INFO\_ALLOWED\_FIELDS, binder);**

**}**

**}**

@ResponseBody

@Authorize(policy = SystemPolicy.EDIT\_CLAIM\_DETAILS)

@RequestMapping(value = UPDATE\_CLAIM\_DATES\_REQUEST\_MAPPING, method = RequestMethod.POST)

public AjaxResponse<ClaimSummary> claimDatesUpdate**(@ModelAttribute(CLAIM)** ClaimSummary partialClaim, @RequestParam String claimID){

//pass claim summary data

}

**References**

1. [OWASP, Mass Assignment](about:blank)
2. [Spring, Spring MVC Known Vulnerabilities and Issues](about:blank)
3. [INPUT-1: Validate inputs, Oracle](http://www.oracle.com/technetwork/java/seccodeguide-139067.html%235)