During our analysis of CSRF issue over various application like: FTCS , CECFG etc. We have come up with following analysis:

1. As we are aware that coverity scans are static scans(code level scanning) and are not a dynamic one. It doesn’t care even you pass CSRF token from UI to REST calls, based on the rule it will still show issues.
2. When we tried to pass CSRF token from UI form as hidden parameter, even then coverity was not satisfied .

**Solution to satisfy coverity:**

* CSRF issue was coming during POST request as coverity says we are not securing the entry point.

**BEFORE CODE CHANGE**:

@Override

       @RequestMapping(value = MappingConstants.***CECFG\_SAVE\_CONFIG***, method = RequestMethod.***POST***, headers = {CecfgConstants.***MIME\_TYPE\_JSON***})

       @ResponseBody

**public** CecfgConfigVO saveConfig(@RequestBody CecfgConfigVO cecfgConfig,HttpSession httpSession) {

**return** cecfgService.saveConfig(cecfgConfig , (String)httpSession.getAttribute(CecfgConstants.***COMMON\_VALIDATION\_SERVICE\_URL***),

                           (String)httpSession.getAttribute("AUDIT\_LOG\_URL"),(String)httpSession.getAttribute("FTCS\_SESSION\_ID"),httpSession);

       }

In the above code snippet coverity says we are not securing our entry point because we are not defining the way we want to produce and consume our request.

**AFTER CODE CHANGE**

@Override

@RequestMapping(value = MappingConstants.***CECFG\_SAVE\_CONFIG***, method = RequestMethod.***POST***, consumes = MediaType.***APPLICATION\_JSON\_VALUE***, produces = MediaType.***APPLICATION\_JSON\_VALUE***,headers =  {CecfgConstants.***MIME\_TYPE\_JSON***})

       @ResponseBody

**public** CecfgConfigVO saveConfig(@RequestBody CecfgConfigVO cecfgConfig,HttpSession httpSession) {

**return** cecfgService.saveConfig(cecfgConfig , (String)httpSession.getAttribute(CecfgConstants.***COMMON\_VALIDATION\_SERVICE\_URL***),

                           (String)httpSession.getAttribute("AUDIT\_LOG\_URL"),(String)httpSession.getAttribute("FTCS\_SESSION\_ID"),httpSession);

       }

In the above code when we added the way request should produce and consume as request which is as JSON, coverity is **SATISFIED** and not giving CSRF issue.

**CONCLUSION**

There are two point now:

1. **To satisfy coverity-** We can use the above solution to satisfy coverity and in above solution we are anyway expecting a JSON always (rather than HTML tags of attacker) so one way protecting our calls.

1. **Solving CSRF issue by passing token**

As all our application is running from BEN network and it is not exposed to outside world so we are least vulnerable to such attacks. If we have to implement token based system for every request, then a lot of code changes  and efforts at UI and backend are required which I think is not necessary in our case.