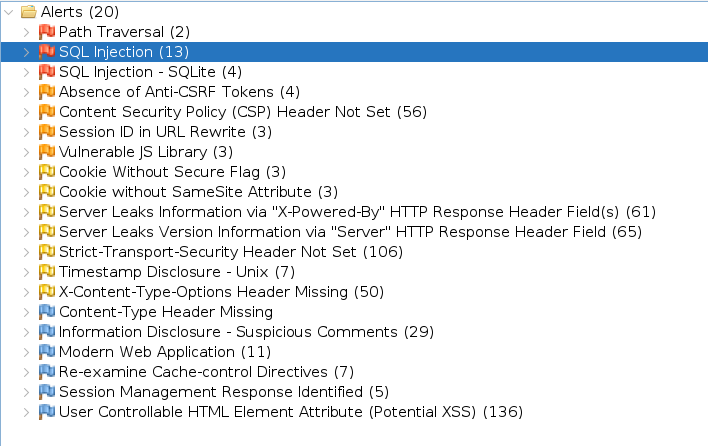
**Agregação de Informações**

Antes de tentar injeções SQL no site desejado, é preciso antes fazer uma etapa de agregar informações relevantes do site que possam ajudar numa SQL Injection bem sucedida. Para isso, foi utilizado a ferramenta ZAP, que é específica para Pentests legítimos, mais especificamente a ferramenta de Automated Scan para procurar vulnerabilidades no site. O scan não foi feito de forma completa, apenas até 38%, porque até ai já tinha demorado quase 2 horas, e não era a intenção do grupo sobrecarregar ainda mais o servidor do SIGAA

Fora achados esses alertas, que indicam algumas vulnerabilidades no site do SIGAA:



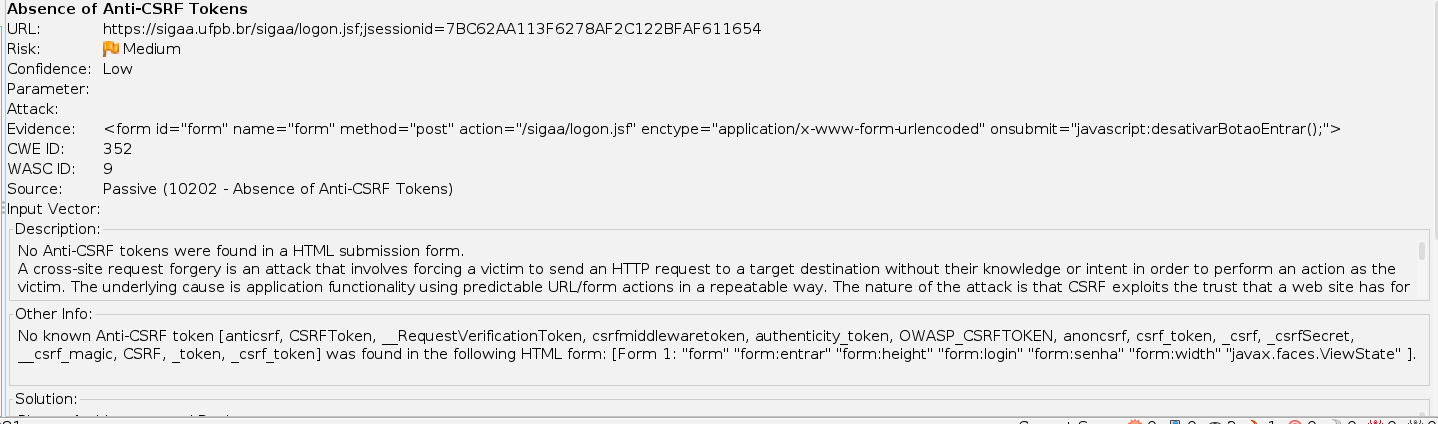
Como o nosso foco de fato é de SQL Injection, não vamos tentar possibilidades além disso. É notório que a ferramenta não encontrou vulnerabilidades relacionadas a Injections no login, porém vamos tentar algumas injeções na mão para checar se de fato está tudo nos conformes. Além disso, como esse projeto tem também o intuito de melhorar o SIGAA, encontrando vulnerabilidades a fim de reportar ao STI para que elas sejam corrigidas, vou entrar um pouco mais a fundo em alguns alertas encontrados pelo Crawler usado no scan automático:

**Bootstrap desatualizado:**



Hoje, o Bootstrap tá na versão 5.3. O sigaa utiliza a 3.3.4. Mesmo se não tivesse nenhuma vulnerabilidade conhecida na versão 3.3.4, ainda sim seria um risco já que não é mais uma versão suportada e já bem deprecada. Porém, como dito em algumas issues do github (<https://github.com/twbs/bootstrap/pull/28236>), há a possibilidade de Cross-Site Scripting em certos pontos do framework

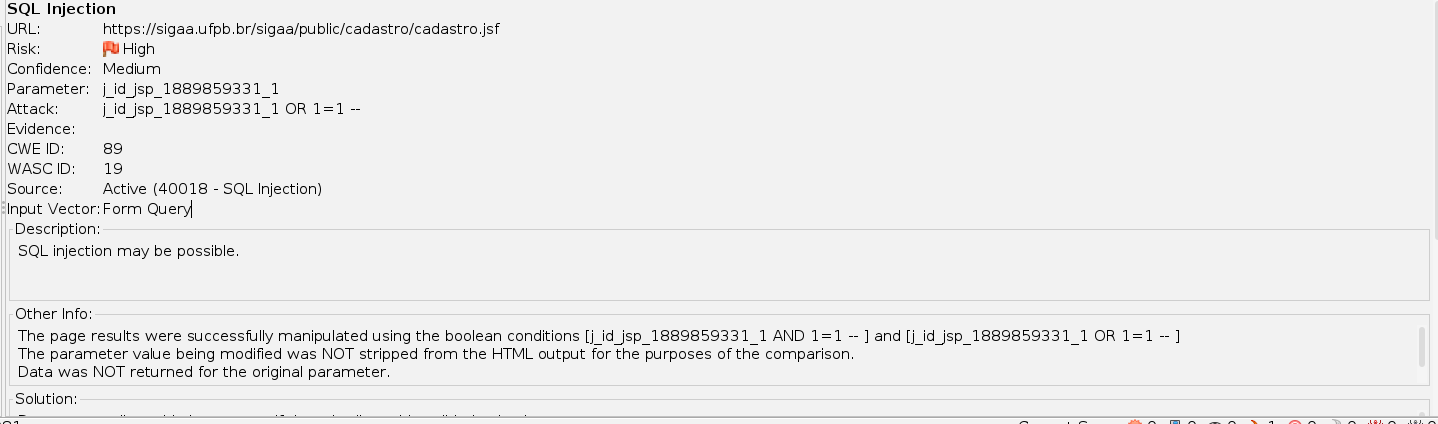
**Ausência de tokens Anti-SCRF:**



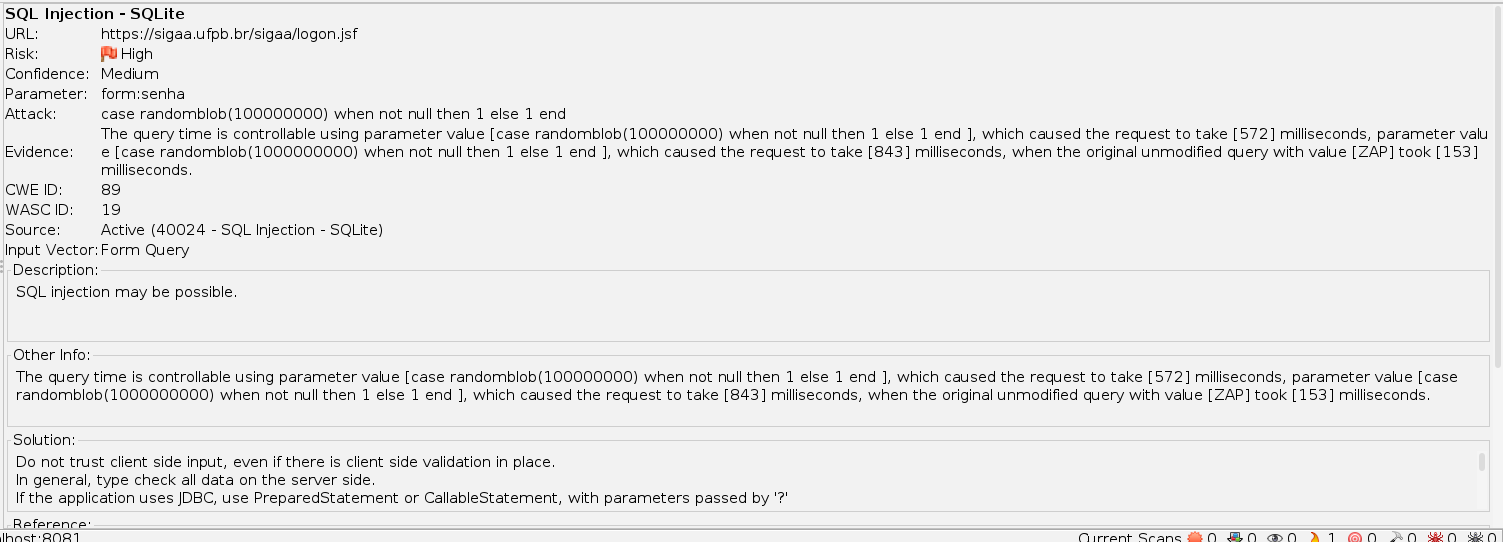
Um ataque CSRF ocorre quando um usuário legítimo se autentica num site, e outra aplicação do seu computador (provavelmente um malware né) consegue acessar esse site através da autenticação feita anteriormente, mesmo que isso não tenha sido dado permissão a esse aplicativo

A confiança desse scan é baixa, então é possível que existam sim esses tokens. Porém, eles não foram encontrados no form de login

**SQL Injections:**

****

Apesar de não ter sido identificada nos logins, algumas possíveis formas de SQL Injections utilizando operadores booleanos foi encontrada

Além disso, um outro check encontrou SQL Injections no padrão do SQL Lite:  
  


Ou seja, basicamente qualquer atacante consegue fazer esse scan e descobrir injeções possíveis além de descobrir o banco de dados utilizado, o que já diminui muito as buscas de ataques de injeção que precisam ser feitas

**SQL Injections**

Apesar do scan não ter indicado que existem vulnerabilidades de SQL Injection no login, decidi testar algumas injections mais conhecidas para realmente tirar a prova de que tá tudo certo

Os testes basicamente foram feitos na mão, usando o firefox e testando se alguma falha seria encontrada ao colocar algumas injeções mais famosas. Por exemplo:



Que é uma injeção básica famosa e basicamente testa 2 tipos de erros:

* Primeiro se a primeiras aspas ‘ gera um erro. Se as strings utilizadas no comando SQL forem definidas com apenas aspas simples, uma aspas no usuário poderia gerar alguma requisição do tipo SELECT \* FROM USERS WHERE name = ‘’nome’, em que a aspas provocaria um erro
* Se a aspas não provocar um erro, é basicamente tentado se o resto da requisição não ficaria SELECT \* FROM USERS WHERE name = ‘nome’ or 1=1. Como 1=1 está sempre correto, isso retornaria todos os usuários possíveis

Mais especificamente, as injeções testadas:

**Injeções básicas:**

| **Usuário** | **Motivo** |
| --- | --- |
| ‘ | Tenta fechar a string para provocar erro |
| “ | Tenta fechar a string para provocar erro |
| ‘ or 1=1 | Uso de 1=1 para retornar qualquer coisa |
| ‘ or 1=0 | Uso de 1=0 na tentativa de não retornar nada |
| ‘ and 1=1 | Uso de 1=1 para retornar qualquer coisa |

**Injeções Cegas com Delay**

| **Usuário** | **Motivo** |
| --- | --- |
| ' or sleep(2) and 1=1# | Se der certo, será percebido um delay na resposta |
| ' or sleep(2)# | Se der certo, será percebido um delay na resposta |
| admin' and sleep(2)# | Se o admin existir, um delay será percebido |
| ' union select sleep(2),null# | Delay será percebido e o null serve para entender a quantidade de colunas retornada |
| ' union select sleep(2),null,null,null,null# | Delay será percebido e o null serve para entender a quantidade de colunas retornada |

**Injeções com comentários**

| Usuário | Motivo |
| --- | --- |
| ' or 1=1# | O # seria o comentário que desativaria o resto da consulta, então apenas 1=1 seria checado |
| ' or 1=1 -- | O – seria o comentário que desativaria o resto da consulta, então apenas 1=1 seria checado |
| ' or 1=1 // | O // seria o comentário que desativaria o resto da consulta, então apenas 1=1 seria checado |
| or 1=1# | O # seria o comentário que desativaria o resto da consulta, então apenas 1=1 seria checado. Só que agora sem parâmetro |
| ' or 1=1-- sd | O – seria o comentário que desativaria o resto da consulta, então apenas 1=1 seria checado |
| ' AND (select 1)=1 | Verifica se a subconsulta retorna 1, que é trivialmente verdadeiro. |

Porém, no login do sigaa nenhuma tentativa foi bem sucedida

A ferramenta de testes usada identificou com que o cadastro de discente pode ter alguma injeção SQL passando no cadastro. Algumas delas no CPF, data de nascimento, nome e email. Porém, muitas parecem falsos positivos. Os testes foram feitos basicamente assim:



Testando a injection em um campo por vez. Porém, nada foi encontrado. Na verdade, muitas das verificações parecem ter sido feitas já no front end da página. O site checa por exemplo se o CPF é válido, se o email corresponde ao email do cadastro, se a data de nascimento é válida. Ou seja, não parece ter muitas vulnerabilidades de segurança nessa parte do SIGAA também.