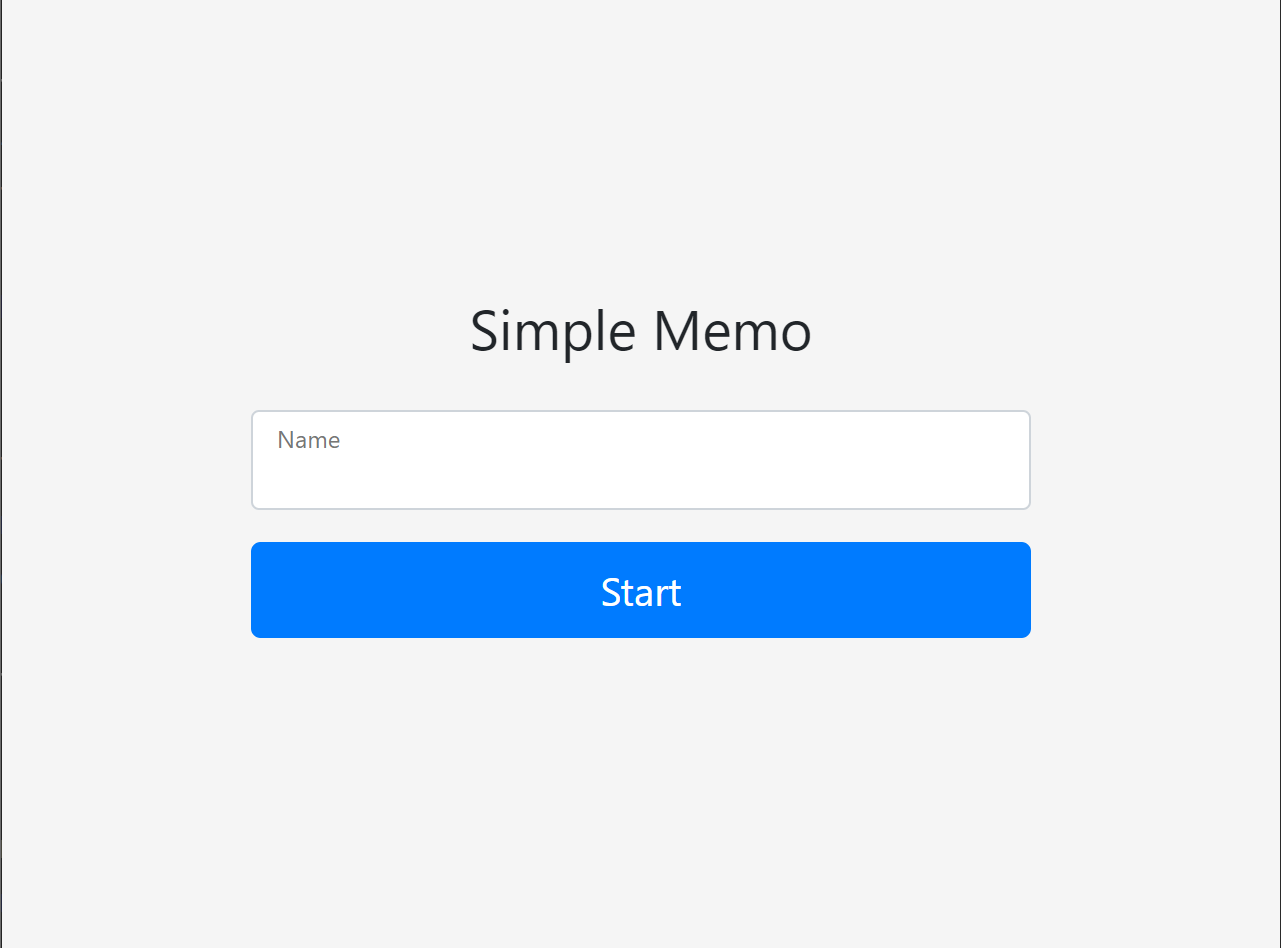
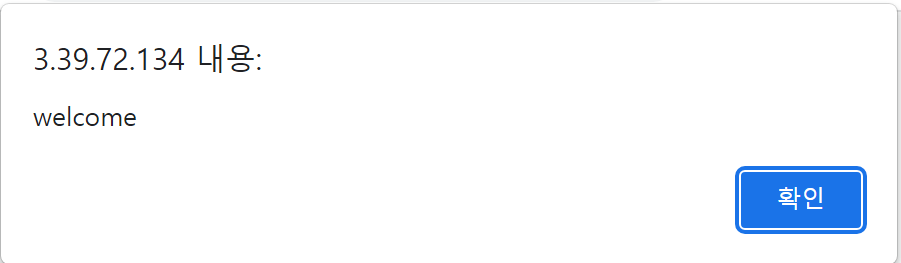
**babyFirst**

해당 문제는 jsp로 이루어진 웹해킹 문제입니다. Servlet은 **MemoServlet** 하나밖에 존재하지 않았고, jsp 파일은 **list.jsp**, **read.jsp**, **write.jsp** 3개의 파일이 존재하고 있었습니다.

텍스트이(가) 표시된 사진

자동 생성된 설명

먼저 사이트에 들어가 보면, 아래와 같이 name을 입력하는 창이 나오고, name을 입력할 경우 welcome이라는 알람과 함께 **list.jsp**로 이동하고 있는 모습을 볼 수 있습니다.

****

**List.jsp**파일에서 받아오는 부분을 확인하기 위해 **MemoServlet**의 **doGet()**을 확인해 보았습니다. Login을 했는 지 확인하고, login을 했다면 요청하는 url에 따라 url이 memo/list이면 **list.jsp**로, memo/read면 **read.jsp**로 forwarding 해주고 있는 모습입니다.

텍스트이(가) 표시된 사진

자동 생성된 설명

그 아래 코드에서 확인한 doPost 함수에서 memo/write라는 url이 숨겨진 것을 확인했습니다. 로컬에서 hashCode를 확인해 봤더니, 문자열 **“write”**를 해쉬한 값이 113399775가 나왔습니다. 하지만 write를 하려면 login이 필요했고 이를 위해 아래와 같이 파이썬으로 코드를 작성하였습니다.



세션을 오픈하여 login을 진행해주고, 해당 세션으로 write를 해주는 코드를 작성하였습니다.

텍스트이(가) 표시된 사진

자동 생성된 설명텍스트이(가) 표시된 사진

자동 생성된 설명

이후에 read에서 어떤 행위를 하는 지 살펴보다가 해당 index를 가지는 row를 불러오는 과정에서 중간에 **lookupImg()**라는 함수를 확인했습니다. 그래서 **lookupImg()**를 더 살펴보았습니다.

텍스트이(가) 표시된 사진

자동 생성된 설명

**lookupImg()**함수를 분석한 결과는 다음과 같습니다.

1. 정규표현식으로 memo가 [] 괄호에 감싸져 있는 지 확인한다.
2. [] 괄호에 감싸져 있다면 양 끝에 [] 괄호를 제거한다.
3. [] 괄호 안에 있던 문자가 **a-z**로 시작하며, 끝에 **:** 특수기호를 가지는 지 확인한다.
4. 해당 문자열이 file로 시작하지 않으면, URL클래스에 넣어 생성하고, 파일을 받아온다.

텍스트이(가) 표시된 사진

자동 생성된 설명

docker 파일에서 flag의 위치를 확인할 수 있었습니다.

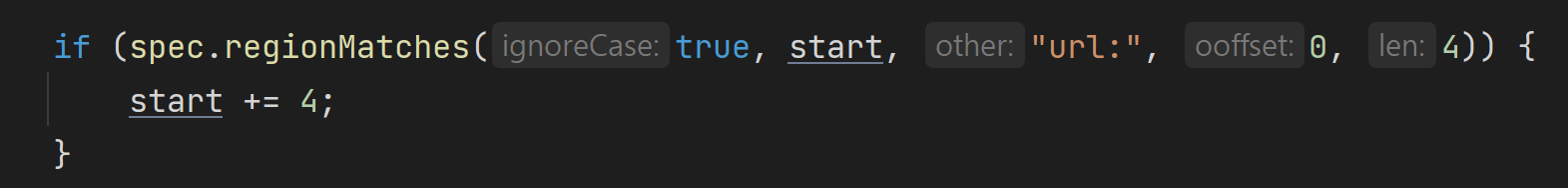
텍스트이(가) 표시된 사진

자동 생성된 설명

Flag에 접근하기 위해 **file: url**을 사용하려 했으나, **정규표현식 + file로 시작하는 문자열은 안된다**는 조건에서 막혔습니다. 분명히 파일을 가져오는 방법에 URL을 사용했다는 점에서 취약점이 있을 것이라고 생각하고 OpenJDK 11버전의 URL클래스를 뒤져 보기 시작했습니다.

뒤져 보던 중 classpath라는 url을 발견했고, classpath로 접근해 보려 했지만, flag의 위치가 classpath의 바깥 ROOT directory에 존재했기 때문에 접근할 수 없었습니다.

URL 클래스를 더 뒤져보던 중 이상한 protocol을 발견했는데, 아래와 같이 protocol이 url: 으로 시작하면 생략하고 start를 4만큼 더한다는 내용이 있었습니다.



해당 부분에서 힌트를 얻어 아래와 같이 **url:file:flag**경로로 접근했고, base64로 인코딩 된 flag를 획득할 수 있었습니다.

텍스트이(가) 표시된 사진

자동 생성된 설명

텍스트이(가) 표시된 사진

자동 생성된 설명

텍스트이(가) 표시된 사진

자동 생성된 설명